

Introducing GDL

(Geometric Description Language)

IN the early days of ArchiCAD, the forerunner of ArchiCAD (RaDar) ran on an old Hewlett Packard as a 3D Piping design solution. Yes, the original Graphisoft team designed 3D piping for nuclear power stations! Then they got more and more interested in the problem of wrapping buildings around the piping. Gradually they moved into architecture.

Not having a mouse or windows, the ancestor of ArchiCAD was something like GDL – a scripting language to produce 3D form. The arrival of the Apple Lisa followed by the Macintosh enabled programmers to put the building plan into one window, the building tools into another (to form a palette of clickable buttons), coordinates in another palette, and so on.

ArchiCAD as we now know it became visual – place elements, click, drag and click again. Designers could drop walls and other building elements into a plan window, stretch them to fit and change their properties; view the 3D results in another window. GDL survived as a speciality, evolving in its own way as a means of building library objects. For some, GDL is the reason for picking ArchiCAD over its competitors.

Working with GDL consists of four main areas of knowledge

GDL COMMANDS

– produce 3D and 2D entities – like CONE, BLOCK, SPHERE, TUBE, POLY, ARC, REVOLVE and so on.

3D ORGANISATION

– look at the object analytically and understand what 3D shapes it contains and what rules they follow.

PROGRAMMING

– the organization of the script in a structured way, observing rules of syntax.

INTERFACING

with ArchiCAD – to enable the object to respond to conditions in the main project.

GDL COMMANDS – in their raw state are available to you in the GDL manual. It is difficult to understand their use and syntax until you have to apply them for real. Therefore, the cookbook approach in this book gives you an easier way of learning them by doing small projects, a bit at a time. The cookbook method is based on small projects, growing in complexity and a critical look at what we have learned from it. This is analogous to the process of architectural education. The GDL explanations in this cookbook runs systematically through all the 2D and 3D commands with small examples of the commands at work.

3D ORGANISATION – You may not be able to build something in every detail, but you look at it, and decide what parts of it need to be included. You also decide how the object will behave if it is to be stretched or hinged, and what 3D primitive shapes – blocks, cylinders, cones, surfaces – would best produce the object.

PROGRAMMING – A program is just a sequence of instructions. You can read instructions to someone over the telephone, you can struggle through the instructions for setting the controls on your new video, you can explain a cookery recipe to a friend, or plan the most efficient way of mowing the lawn. There is even a program required in making a cup of tea! In all these cases you use a program, even if you didn't think of it as a program. Suppose you write down the sequence of tasks, you would be sure to get them into the right order. You would try to avoid doing tasks you

had just done, or doing tasks that would undo ones you had just done. You can write the sequence as a series of pictures, or as written commands. That's all there is to it!

Which language would you use? For cookery, you would speak in English, or French or Spanish, but you would use similar food ingredients. For GDL, you use the language of BASIC, and your ingredients are commands to produce primitives like BLOCK, CYLIND and CONE; and you can go on to make more complex shapes like TUBE. BASIC is the easiest of the major programming languages, devised in the seventies, and was predominant on micros during the eighties. BASIC is easy to learn – so is GDL.

Graphisoft must be thanked for choosing BASIC as the model for their programming language. We can all write a few lines of code, and see 3D objects springing up into existence. It's fun! It's intellectually challenging! One's capabilities are vastly increased – in the 'old days' of programming, BASIC could build menu driven programs for dull things like accounts and calculations, many of which were far easier to do in spreadsheets. Now, with GDL, towers, chairs, bridges, space structures, buildings and trees can all grow from your keyboard.

INTERFACING – GDL objects can behave slightly intelligently in that they can know storey, scale, frame number, object's location, current pen, camera position and so forth, and behave accordingly. If they are windows and doors, they can know current wall thicknesses and materials.

So, you think you can't program...



WELL you can! Squirrels can program, birds can program, spiders do it, small children do it – in that they plan a series of actions, they try to get them in the right order, they correct their own mistakes when they make them, and they achieve a result – perhaps a hoard of nuts, a nest full of eggs, a web, or a Lego model.

The British Standards Institute (Sept '99) received the 'Ig Nobel' Prize for Literature in that they wrote BS.6008: *Method for Preparation of a Liquor of Tea*, and took 5000 words to do it, going into the minutest detail! We will not go quite so far, but let's use tea making as an example of human programming. Let's write down a series of 'Operations' involved in making Tea. Without going into too much detail, list the sequence of operations to someone who knows the obvious things like how to turn on taps. Anybody reading this (above the age of 10) should be able to recite the attached list in the right order.

Lovers of tea should try this website for more about the history and culture of tea:

<http://www.stashtea.com/>

1. Fill Kettle with water
2. Boil water
3. Get teapot
4. Fill teapot with bags
5. Fill with water
6. Get cups
7. Wait until ready
8. Pour out
9. Add Milk
10. Serve Tea

1

Now because we are going to teach a computer how to make tea, you need to rewrite the list giving each Operation a number (with a colon), and use an exclamation mark to make each Operation name into a Label. For a human, you might number each task 1, 2, 3, etc, but for computers it's easier to use larger numbers (because later, you could insert extra things you hadn't thought of, like getting the sugar lumps.)

```
100: !Fill Kettle with water
200: !Boil water
300: !Get teapot
400: !Fill teapot with bags
500: !Fill with water
600: !Get cups
700: !Wait until ready
800: !Pour out
900: !Add Milk
1000: !Serve Tea
```

2

NEXT, you look at each Operation and realise that none of them are single Actions. Each operation consists of a number of Actions which might include Error Checking. For example, if the teapot contains last night's teabags, then Wash Teapot. This might require quite a lot of IF statements, followed by the Actions taken as a result of the IF statements.

Depending on the level of knowledge or stupidity of the machine, you instruct it appropriately. For example we assume here that the machine knows how to turn a tap or a kettle on. In the same way, GDL knows that Cylinders are round and that Cones are tapered *and* round without us having to define 'roundness' mathematically.

The final program for making tea

Goes around until kettle filled: 2 IF statements

A single line way of writing two IF statements

People are 'integer' quantities; but this makes doubly sure that you can make tea for 2.

For the 'parameter' of tea, you could let the user select from a list of choices.

In the event of a major problem you might have to jump back to the start.

```
100: !Fill Kettle with water
IF kettle (empty) THEN (Fill kettle with water)
IF kettle (filled) THEN CONTINUE: !Boil water

200: !Boil water
IF water (boiling) THEN (turn kettle off) ELSE (boil water)

300: !Get teapot
IF (number of people) less or equal to 2 THEN (GET smallteapot)
IF (number of people) greater than 2 THEN (GET largeteapot)
IF teapot (dirty) THEN (wash out teapot) UNTIL (clean)
IF teapot (clean) THEN (swirl hot water) UNTIL (warm)
IF teapot (warm) THEN (Fill teapot with bags) !continue

400: !Fill teapot with bags
LET teatype=USERCHOOSE(EarlGrey,Lapsang,Camomile,Tetleys,Green)
IF teapot (large) THEN add 2 teabag USING teatype
IF teapot (small) THEN add 1 teabag USING teatype

500: !Fill with water
POUR water INTO teapot UNTIL teapotfil (full)
IF (not enough water) THEN GOTO 100:
PLACE teacosy UPON teapot
START timeclock USING seconds
```

3

Continued next page

This was going on while you were getting the cups.

Another repeating 'Loop'

Another example of getting the parameter from the user's choice.

```

700:!Wait until ready
IF TIMECLOCK less than 120 seconds THEN WAIT

800:!Pour out
REPEAT (on the teatray)
  IF fullness=0 THEN POUR (a cup) UNTIL fullness=7/8
  UNTIL (enough cups for everybody)

900:!Add Milk
LET milkcondition=USERCHOOSE(with milk,without milk)
REPEAT (for each cup on the teatray)
  IF milkcondition=(with milk) THEN (add milk to cup)
  UNTIL (each cup tested)

1000:!Serve Tea
1100:!Tea too hot, pour into saucer and drink from that
!...vicar is shocked!

```



4

Now this isn't written in BASIC, and it isn't written in GDL, but it's close to both of those two, and shows that programming can be like writing in English in a slightly mathematical style.

OK, we can't all be perfect. The British Standard says put the milk in the cup first, and of course, I haven't tested to see if the users have chosen sugar, or want spoons, saucers or biscuits. And what happens if there isn't enough tea for everyone? Then you have to loop back and top up the teapot, or even reboil the kettle. This could be built into the program. That level of detail and error correction would make the program more professional and user friendly.

I hope this shows the sort of thing you can do, and has persuaded you that you can indeed write a program.

How do you make an Object?

Look at it as if through half closed eyes.....

See it like this in your mind's eye, reduce the object to what you feel is essential about it; and then you ask...

Can you draw it in 3D on paper?

If you can sketch it in 3D, you are well on the way to understanding how you would build it in GDL; and then you ask...

What are its primitive forms?

Most things can be simplified to the simple shapes of cylinders, blocks, and cones; furthermore, you can perceive tubes, surfaces, prisms and lathed elements; and then you ask...

Does it need to LOOK right, or BE right?

Is this a quick little object that adds authenticity to a 3D rendering but doesn't need to be accurate? – or is it required to look good in detailed sections and plans in a construction drawing? And then you ask...

What Level of Detail (LOD) does it require?

If it's for a rendering, you may be seeing it fleetingly during a flythrough, or requiring it for an Artlantis closeup. For a construction detail, it could be scale sensitive so that it shows detail at 1/20 and 1/50 but simplifies at 1/100 and 1/200. Your object may be making too many polygons if it is curvy. Consider LOD constantly; and then you ask...

Could it be 2D only?

Some objects are only required for the 2D drawings. People, trees and furniture could be 2D entities that appear only in plans sections and elevations; perhaps you needn't bother with the 3D; and then you ask...

Does it need to be stretchy?

We are all familiar with autoscripted objects being stretchy. But does it really need to be stretchy? Or can the stretchiness be controlled to make the object stretch smartly? And then you ask...

Does it need coded GDL?

If you know GDL already you might dive in without hesitation. But can it be built easier with Profiler or Trussmaker or using ArchiCAD's tools? although it wouldn't be smart... then you ask...

Will it be an Object, or a Tool?

It may be a single piece of furniture. But if it's a cladding panel, fencing, handrail or similar, it may be more akin to a Tool, whereby if you stretch it, you get more and more objects. And then you ask...

How smart does it really need to be?

If it is a tool and it is an investment, you can make it smart – know the 3D rules of its existence, how to behave when stretched. But some objects can be so smart they can be annoying to the user, and can take too long to develop. And then you ask...

Does it need to appear in listings?

This is increasingly important with manufacturers jumping to the realization that they can make libraries of their products in GDL, not just in DXF. The object may need to have accurate names and serial numbers in the property scripts and smart value lists to ensure that the name/number corresponds to the object. And then you ask...

Can complex problems be solved with textures?

Recognise when 3D forms or surfaces are so complex that a texture will solve them better than modelling – for example leaves, tiles etc. And then you ask...

Have I got the ability to make it?

If not, then here's the chance to learn new tricks!

Armchair 01



☐ Template ☒ Placeable

Variable	Type	Name	Value
A	[Icon]	X Dimension	520
B	[Icon]	Y Dimension	520
zzyzx	[Icon]	Height	660
gs_detlevel_3D	Abc	3D	Detailed
gs_seat_height	[Icon]	Seat Height	400
gs_arc	[Icon]	Backrest Arc Radius	120
gs_3D_repre...			
gs_resol	[Icon]	Resolution	18
gs_full_edit	[Icon]	Extra Hotspot Editing	On
gs_shadow	[Icon]	Shadow	On
AC_show2DHot...	[Icon]	Show 2D Hotspots in 3D	Off
gs_2D_repre...			
gs_min_space	[Icon]	Minimal Space	Off
gs_cont_pen	[Icon]	Contour Pen	7
gs_fill_type	[Icon]	Fill Type	65
gs_fill_pen	[Icon]	Fill Pen	91
gs_back_pen	[Icon]	Fill Background Pen	91
gs_material			
gs_frame_mat	[Icon]	Frame Material	14
gs_seat_mat	[Icon]	Seating Material	67
gs_back_mat	[Icon]	Backrest Material	66
gs_list			
gs_list_cost	[Icon]	Cost	0.00
gs_list_manufact...	Abc	Manufacturer	
gs_list_note	Abc	Note/Remarks	

1 2 3 4 5 6 7 8

9 10 11 12 13 14 15 16

Parameters

Components

Descriptors

Master Script

2D Script

3D Script

Property Script

Parameter Script

Interface Script

Comment

2D Symbol

2D Full View

3D View

Preview Picture

Subtype Hierarchy

- General GDL Object
 - Drawing Symbol
 - Model Element
 - Building Element
 - Beam
 - Built In Column
 - Covering
 - Foundation
 - Railing
 - Ramp
 - Roof
 - Slab
 - Special Construction
 - Stair
 - Wall
 - Wall End
 - Distribution Element
 - Electrical Element
 - Equipment Element
 - Furnishing
 - Bed
 - Built In Furniture
 - Other Furniture
 - Seating**
 - Storage
 - Table
 - Macro
 - Window_Panel_Macro
 - Opening
 - March Opening
 - Roof Opening
 - Slab Opening
 - Wall Opening
 - Site
 - Animal
 - People
 - Plant
 - Sport Field
 - Street Furniture
 - Traffic
 - Transport Element

Properties

Name

Seating

Version

20

GUID

{19C8C24-4ED7-4C25-9114-6B08642D44F1}-50271A0D-824F-4B43-81D9-14BC34F4C3C7

Location

[C:\Program Files\ArchCAD 8.1\Add-Ons\Standard\Necessary\Scripts\Seating.gsm]

☐ Template ☒ Placeable

GDL is not just for 3D objects!

GDL can also make Windows, Doors and Skylights which know how to cut holes in walls or roofs; you can make Lamps, Beams, Zone stamps and Labels. They can be pure script (e.g. a list of materials); Macro object (e.g. door handle, tap); Property object, or Stairmaker part. 3D and 2D objects can have subtype definitions that distinguish them from each other, such as seating or roofing. Click on the <Select Subtype> button to see the list.

This classification system is part of GS's commitment to Industry Foundation Classes (IFCs). The list of subtypes here is a massive increase from the ones offered in earlier AC versions, and it provides you with a comprehensive list of product types ensuring that your GDL library parts will be compatible with IFC.

Look at GDL

FROM the File>GDL Objects menu, open any 3D object and have a quick look at the way the GDL library part editing dialog is organised. Here is the dialog and parameter table for a typical chair from ArchiCAD 8's library. (Earlier GDL Cookbooks show how 5.0, 6.0, 6.5 or 7.0 GDL dialogs were organised.)

For total beginners, you only need to use the 2D and 3D Scripts. You quickly develop your skills and make use of the Master Script and Parameter Script, and with a little more practice the UI Script and Property Script. Buttons enable you to see the scripts. If you press the grey buttons, the scripts appear in the main window. If you press the white buttons, the scripts appear in independent floating windows.

The preview window (top left) offers you, at a glance, a choice of 2D Symbol, 2D View, elevation, 3D hidden line, 3D shaded views. The right hand button can display a pasted-in photorendered view of the object. By clicking on the 3D image you can view the object from different angles.

PARAMETERS: Press this button and the GDL environment displays the Parameter Table, which is the usual condition in the GDL Dialog. This shows the parameters, their names and types and the values you wish them to have when you test or save the object. You can organise the order of parameters, insert titles, indent parameters under titles, hide parameters. We fill out the table of parameters by hitting the <New> button, and filling in the small details. A, B and ZZZX are 'obligatory parameters' – they already exist – but we can make many more. We click on white or grey buttons to view the scripts.

COMPONENTS and **D**ESCRPTORS buttons are used when you build a 'Property Object'. These allow you to build a parameter table for use with Property objects – attached to building elements such as walls or floors.

MASTER Script is the most powerful one and it is read first. It can be where you put everything that is not specifically 3D or 2D, all the housekeeping tasks - i.e. Calculations, checking the results of Menus, Idiotproofing (in case people enter silly values), Defining Materials, setting Flags etc.

Information in the Master Script can be used by both the 2D and 3D Scripts and other scripts, so it centralises decision making on your part – if you didn't have this, you would have to do repetitive calculations and error checking in each script.

2D Symbol is a 2D window into which you can paste a 2D image, or draw using 2D tools, but it will only be displayed if there is no 2D Script, or if the 2D Script tells it to be displayed. With GDL knowledge, you are better to write a 2D Script. The 2D Symbol can have up to 16 layers. Because the word 'layers' is already used in the 3D, the word for them is 'fragments'. The 2D Script can decide which fragments to show: you can hide or show complex permutations of 2D objects – eg, a motorcar or tree object could have plan and elevational views of itself stored in fragments. For simple objects using only the 2D symbol, you can click on the little fragments buttons to decide which will show.

2D Script: This can be used simply to tell GDL to draw the symbol you see on your floorplan (it could be done with a drawn symbol instead). At its simplest it can just draw a view of the 3D object - at its most complex, it can display the object from several directions, include self labelling and dimensioning, pictures, etc, with user control of linetypes, fills and pens. By designating smart hotspots, the 2D Script can also govern how stretchy the object can be. By leaving this script blank, the GDL object will display what ever is drawn into the 2D Symbol window. If you leave both blank, the object will not show in the project.

2D Full View: Displays the symbol you will get on the floor plan as a result of the 2D Script, including your hotspots. This can be scripted using LINE2, RECT2, ARC2, POLY2 etc, but can also include elements from the 2D symbol if the FRAGMENT2 command is used.

3D Script is the primary means of building parametric 3D objects. If the object is simple, almost all the work can be done in the 3D Script.

3D View is generated by the 3D Script – shows what your model will look like, and using 3D settings, you will be able to see it from every angle in shaded, hidden line or wireline mode. You can set this to Raster or Analytic. The small quickview window at the top is set in Analytic mode, so if your model is complex, it's best to leave it in 2D or Preview mode and use the 3D view to see your model.

PROPERTY Script: Here you can write Components and Descriptor commands if the object is to be in Elements, Components and Zone lists.

PARAMETER Script: This is a convenient place to build all your pop-down menus, and for locking or altering parameter values; however, the author mostly prefers to do this in the Master Script so that the logic sequence is maintained – the menus are built, and results are checked.

COMMENT is a small text field in which you can write a small set of instructions to your user on how to use the object, or could place a copyright statement/ signature. You can enter the URL of websites relevant to the object, such as the manufacturer of the furniture or component.

PREVIEW Picture is a window containing a pasted in bitmap image of the object – it will resize to 128x128 pixels. It tells the user what the object will look like in its setting, and could come from Artlantis or an ArchiCAD rendering. This becomes the Icon of the object in the Settings box browser and in Windows.

USER Interface Script enables you to build a Custom dialog box with text fields, images, buttons and input fields for the user to enter parameters. It is great for complex objects where the user might need more explanation on the use of the object or the visual choices of style and form, or where the number of parameters could become bewildering to the user. The UI can be organised into tidy pages with pictures and instructions to the user, more helpful parameter descriptions, checkboxes or buttons to link the pages.

FRAGMENTS: The 16 numbered cubes under the preview window.... These are used for the 2D Symbol and are less used than the 2D Script. 2D Fragments are analogous to 2D Layers. Clicking on the cubes will hide or show the fragments in the 2D Symbol window.

NEW button gives you a new parameter, always Length, and you can change this to Material or other parameter type by clicking on the popup parameter palette.

DELETE : Will delete parameters (not undoable).

SELECT SUBTYPE: Decides on the object's classification (see the comments on IFCs).

TEMPLATE: Allows you to define your own Subtype.

DETAILS: Set bounding box hotspots etc.

PLACEABLE: The object may be used only as a Macro, such as a door handle or tap.

Working in 3D Space: 3D Cursor

JUST HOW do you write in GDL in 3D? Well, when you are word processing, you move your cursor to a location and start typing. Move the cursor, and type again, and the new words appear at the new position.

With GDL, you have a **3 pointed cursor**; you can move or rotate the cursor; when you issue a 3D GDL command like BLOCK or CONE, you will get a 3D object wherever the 3D cursor happens to be. Move the 3D cursor to a new position and issue the same command and now it appears in the new position.

Thus to make a chair, you can move to each corner, plant a chair leg, then raise the cursor up to plant the seat and finally, move the cursor to plant the back of the chair.

When you have achieved a group of things (like a bunch of chair legs), you have completed a task. So now return the cursor to the global origin. Now you can depart and do another task, for example the seat of the chair, and later the back.

When you have finished, you can do a small 2D Script so that a 2D symbol will appear in the project plan, save the file, and you are done.

FIRST GET acquainted with the idea of the X, Y, Z universe that you have to work in.

All locations are defined in these coordinates. We have to simplify our ideas of space so that poor idiot computers can understand where we want the 3D cursor to go. If you want to move sideways, you move the cursor in the **X** direction with an ADD command. If you move forward, you move it in a **Y** direction. If you move up or down, you move it in a **Z** direction. You can Rotate the cursor with a ROT command and the XYZ world gets rotated too. You can even stretch or shrink the cursor with a MUL command.

- The '**G**' (global) coordinates remain fixed at the **Origin** of the model. When you bring an object into the project plan, this origin is what decides the height of the object in the project. The origin should be planned carefully – preferably at the base of the object, and at the axis of any symmetry or rotational axis that you perceive.

TIPS and Tricks are spread through the GDL Cookbook like nuggets of gold and are mostly about GDL, sometimes about ArchiCAD or Life in general.

The best place to find Tips and Tricks at their place of origin is to be a member of the **ArchiCAD-Talk** web-based conference, on the Internet. Although I have tested out all my own tricks here, many of the ideas for them started from questions and answers on **ArchiCAD-Talk**.

There is also a specialised list server conference on the Internet called **GDL-Talk**, which permits more technical questions and is used by the experts.

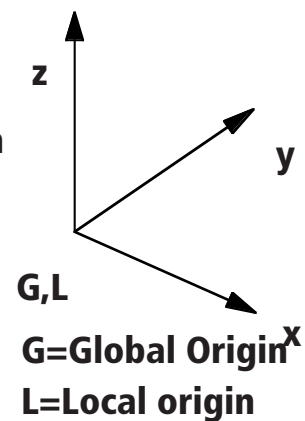
Try ArchiCAD-Talk and GDL-Talk!

Join a User Group and the GDLA

GRAPHISOFT are always willing to listen to constructive criticism or wishlists. Your voice is more effective if you are a member of a user group; also, you might find from the more experienced users that many of the things you want are already possible. If there isn't a group around, then form one with friends, locally. Register the group with Graphisoft in your country.

Another major group you could join would be the **GDL Alliance**, a truly international collection of GDL writers.

The Cartesian XYZ Universe



- The '**L**' (local) coordinates are, in the 3D sense, the **Moving Cursor** in your model – they travel with you, wherever you are drawing an component. Always try to return the GDL cursor to the origin before you start out with another element of the model.

Enjoy a dual view of your script!

In the parameter table, if you press the grey buttons, the scripts appear in the main window. If you press the white buttons, the scripts appear in independent floating windows. You can press both and have both open at once, very good for when you have a complex script and need to work on different parts of the script at the same time:- e.g. when fixing subroutines, while working on the routines that call those subroutines. You also have stretch buttons at the bottom right of the parameter table and can have a larger view of the script.

Use the GDL Manual!

MOST ArchiCAD commands are covered in full in various sections in the GDL Cookbook. But please note that it is not the function of the Cookbook to replace the GDL Manual. You need to use both. You will find the GDL Manual a lot easier to read now you have tried the Cookbook. You will also find sections of the Cookbook easier if you dip into the manual. And while you are in ArchiCAD, you have an extensive HELP menu with most of the manual's contents available at a glance.

First 3D Elements: Primitives

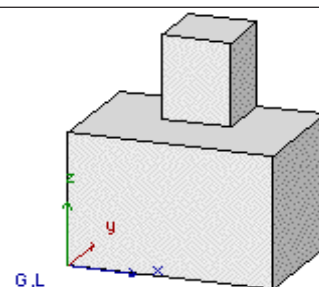
UNTIL you feel ready to progress to PRISM and other more powerful commands, get used to building 3D GDL objects using the items on this page. These are the essential primitives of 3D building. It is amazing how much you can do with these. These examples also demonstrate the simple-to-use cursor movement commands of ADD and ROT.

BLOCK or **BRICK** is the primary building block of GDL, and with X,Y,Z dimensions for the parameters, you can't get simpler.

In GDL projects it is actually rather limited to use; you have to get the cursor to the lower left corner of a block, and you have no control over pen lines, corners, faces etc.

```
!Block Demonstration
!Syntax:- BLOCK x,y,z
BLOCK 0.3,0.2,0.25
  ADD 0.1,0.1,0.25
BLOCK 0.1,0.1,0.2
DEL 1
```

*Some existing Graphisoft library objects use **BRICK** – but it works to the same syntax.*

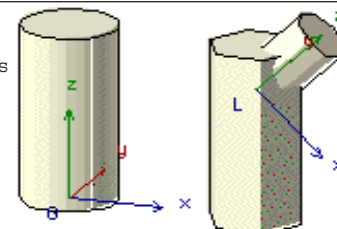


CYLIND drives a cylinder up the Z axis. If you want a cylinder at an angle, you first have to rotate the cursor with a ROT command so that the Z-axis points in the desired direction.

Note that RESOL 6 or 8 can change Cylinders into hexagons or octagons. If you do not write a RESOL command, the cylinder will have 36 sides.

```
!Cylind demonstration
!Syntax:-
! CYLIND height,radius
CYLIND 0.2,0.05
  ADDx 0.2
RESOL 6
  CYLIND 0.2,0.05
RESOL 8
  ADDz 0.15
  ROTy 45
  CYLIND 0.1,0.03
```

By the way, if you want the cursor to return to the origin, you would have to finish this script with a DEL 3.

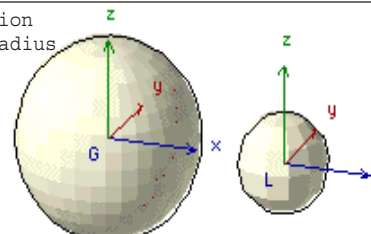


SPHERE is the easiest one of all – but the one most likely to cause you to have so many polygons that ArchiCAD will have trouble rendering it. Use strict control of the surface resolution with RESOL or TOLER commands.

If you do not specify resol, the resol value is assumed to be 36, which means that the sphere will have 648 faces! RESOL 10 is the smallest that still looks spherical.

```
!SPHERE demonstration
!Syntax:- SPHERE radius
!RESOL sets the
!number of faces
SPHERE 0.1
  ADDx 0.2
RESOL 12
  SPHERE 0.05
DEL 1
```

By the way, the single sphere on the left, if saved as a DXF 3D file would require 21,000 lines of code to describe it.

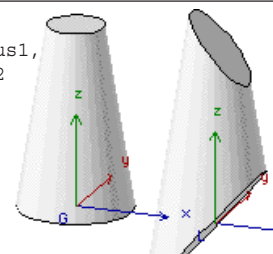


CONE is like a cylinder in which the radius can be varied at the top and bottom.

It also includes a cutting command, to decide if the top should be cut square (90,90 degree cuts) or cut like a ship's funnel, at an angle. The angles are always cut along the X-axis, and must be positive.

```
!CONE demonstration
!Syntax:- CONE height,radius1,
!           radius2,cut1,cut2
CONE 0.2,0.06,0.03,90,90
  ADDx 0.15
CONE 0.2,0.06,0.03,40,140
```

The number of faces on a Cone can also be controlled with the RESOL command

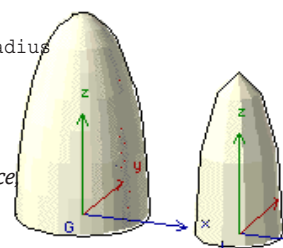


ELLIPS produces a hemisphere, but by changing the height, it can be distorted into a bullet shape. RESOL can be used to make it look polygonal. ELLIPS is always solid. If you want a hollow dome, you need to use REVOLVE.

Note; the exclamation mark (!) is used here to start a line with a Comment on it.

```
!ELLIPS demonstration
!Syntax:- ELLIPS height,radius
ELLIPS 0.2,0.06
  ADDx 0.15
RESOL 12
ELLIPS 0.16,0.04
```

When you use any curved surface always set the lowest resolution that will look acceptable – to reduce rendering time.



3D & 2D Cursor Movement

ADD dx,dy,dz, ADDx dx, ADDy dy and ADDz dz are the way to move the 3D cursor from location to location. Most frequently, you add in one direction, X, Y or Z, but when you want to move in two directions or more it's more economical to use the ADD dx,dy,dz command to do 3 in one go. The best way to get the hang of these movements is to launch into typing in some of the exercises.

ADD2 dx,dy is the 2D equivalent of ADD, but is not as often used, because many of the 2D commands can be issued without moving from the origin.

For example in 3D, you might go:

```
ADD 1,1.5,0: CIRCLE 0.5: DEL 1
```

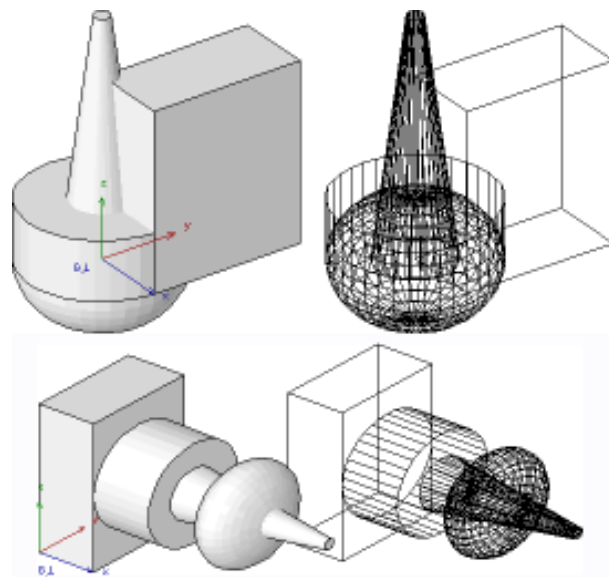
but in 2D it is:

```
ADD2 1,1.5: CIRCLE2 0,0,0.5: DEL 1
```

but this can be shortened to:

```
CIRCLE2 1,1.5,0.5
```

DEL is the way to undo the cursor movements – you specify the number of undoes you want to undo.



Above: The primitives all exist in the same 3D space, a jumble of forms. **Below:** after a few easy cursor movements, they are all distributed correctly

Try making something very simple!

TRY writing something using the simple GDL commands you have just learnt. If you issue the commands (BLOCK, CYLIND, SPHERE and CONE) you find them all jumbled together at the origin and all being a horrible colour. Immediately you can see that they look better if you specify a PEN and a MATERIAL. Use a Material name that is in your Library.

This looks like nothing in the real world, so you have to learn 3D cursor movement from day one. With a few ADD, ROT and MUL commands, you can create something usable. It's a case of lifting the cursor into position, issue the 3D command; either progress to the next 3D object, or use DEL to return to the origin. Experiment with changing the dimensions and modify the cursor movement statements.

```
PEN 1
MATERIAL 'Surface, Whitewash'
BLOCK 1.0, 2.0, 3.0
CYLIND 1.0, 1.0
SPHERE 1.0
CONE 4, 0.5, 0.1, 90, 90
```

Try changing the dimensions of these simple solids

```
!Cursor Movement
PEN 1
MATERIAL 'whitewash'
BLOCK 1.0, 2.0, 3.0
ADD 1.0,1.0,1.5
ROTy 90
CYLIND 1.0,1.0
ADDz 2.0
MULz 0.5
SPHERE 1.0
ADDz -2.0
CONE 5, 0.5, 0.1, 90,90
DEL 5
```

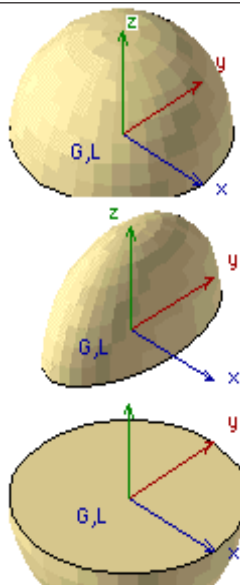
Try changing the values of the ADD, ROT and MUL command to see what happens. Click in the 3D View window in GDL

MUL means **Multiply** – along the axis stated. So **MULz -1** converts all references to Z to negative, **MULx 0.5** halves everything in the X direction – and so on. You can use a **MUL x,y,z** command and do three multiplications at once.

What MUL means in reality is that you can use it to change the scale of entire objects or parts of objects – all ArchiCAD Autoscripted objects use MUL to enable objects to be resized, or to be stretchy.

Objects can easily be mirrored by **MULT**iplying by minus 1 around the mirroring axis. Symmetrical objects are labour saving, as you build just half, then mirror around X, Y or Z.

MUL2 x,y is the equivalent command in 2D Scripting environment. MUL2 is expressed in terms of x and y. Mirroring around X would be MUL2 -1,1.



```
!MUL Demonstration
ELLIPS 1,1
```

```
!MUL Demonstration
MULx 0.5
ELLIPSE 1,1
DEL 1
```

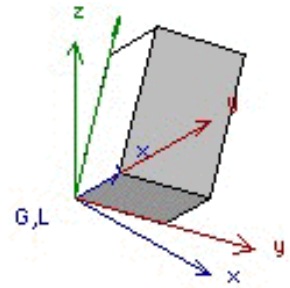
```
!MUL Demonstration
MULz -1
ELLIPSE 1,1
DEL 1
```

Starting with a plain ELLIPS... (half a sphere) you can make it shrink to an elliptical plan shape by MULx or can flip it vertically with a MULz.

ROT means **Rotate the Cursor** around an axis. You use it in the form of **ROTx**, **ROTy** or **ROTz**. You use them one at a time. You use them in the order that logic tells you – the order is vital. Rotation is normally positive if it is in the anti-clockwise direction. If you want to rotate clockwise, you can rotate a negative quantity such as **ROTz -90**.

There is a combined **ROT** command, but it is not used in the way that you use the combined **ADD** and **MUL** commands. The **ROT** command rotates the object around a vector – defined as the line connecting 0,0,0 to a point defined by an X,Y,Z definition. It is unexplained and unillustrated in the GDL manual, so here is a simple example! It is very effective if the circumstance arises where it is needed: when it is, you will recognise its usefulness.

```
PEN 7
MATERIAL 'zinc'
ROT 1,1,1,-60
BLOCK 1.0,0.5,0.8
DEL 1
```



ROT2 is the equivalent in 2D of the **ROTz** command. One of the problems of 2D Scripting is that you don't have a visible cursor, so if you do a lot of **ADD2** and **ROT2** and **MUL2**, you need to imagine the cursor's location.

DEL n means "Delete the most recent Cursor Movement command" – **ADD**, **MUL** or **ROT** or **XFORM**. **DEL 2** means delete the last two.

DEL is the most important command of this page. If **ADD** and **ROT** are components of the engine, then **DEL** is the braking system – just as important in the performance of your motorcar. If you go somewhere, you have to be able to return! If you do not control the cursor, your model will be unmanageable after a page of script. Therefore, use **DEL** to return to the Origin as often as possible.

The bad way to get back to the origin is to repeat all your moves backwards. For every **ROTy 45**, you do a **ROTy -45**, and so on, in reverse order. This is not good! Use the **DEL** command. The **ROT**, **MUL**, **ADD** and **XFORM** commands are stored in a stack, and if you **DEL 1**, it knocks 1 off the stack. If you **DEL 4**, it knocks the previous 4 off the stack – in 'last-in-first-out' order. You can also **DEL** using a variable like *DEL num*.

If you use subroutines, **DEL** must match the number of Cursor movements within a subroutine. Look at the small number of commands on this and the next page to see how **DEL** is used.

The **DEL** command cannot run past the stack. If you **DEL 2** when there is only one item to delete using **DEL**, you will get an error message.

DEL TOP removes all the Cursor commands – returns you to the Origin of the model by brute force.

DEL TOP is useful because GDL does not declare an error when you use it, even if there is nothing to **DEL**. So you can put it at the end of a script, and not worry about an error occurring. However, it is far better to control the model by using **DEL** intelligently – with every subroutine – so that you never need to use the **DEL TOP** command until the **END** statement.

DEL TOP is dangerous – apart from encouraging you to program in a sloppy fashion, it can wipe out global commands (like '**ROTx 90**' or **MUL** commands) that existed at the beginning of the script.

XFORM is a complex command that can combine **ADD**, **ROT** and **MUL** all at the same time. It's the kind of thing that appears when you import a **DXF** into GDL, machines like it, but humans don't. The only time that we need to use **XFORM** is when we want to **SKEW** the object, **XFORM** skews brilliantly.

A Note on Spelling

SOME of my friends in the USA and elsewhere must think sometimes that the author of the Cookbook is quite illiterate – I mean who but a limey could possibly spell 'inquiries' as 'enquiries', 'realize' as 'realise', 'color' as 'colour', 'story' as 'storey', or 'meter' as 'metre'?

Well there is English and there is AmE. Remember the romantic song about 'I say tom-ar-to and you say tom-ay-to'?? Vive la Difference! For me to be consistent, I am mostly sticking to UK spelling conventions.

OK you guys? Have a nice day!

Typographical note: Upper/Lower?

In reading this you have already expressed an interest in GDL – so let's bring in a good discipline at the start.

Write all **GDL commands** in **UPPER** case, and all variables (parameter names) and comments in **lower** case. The *machine* is not case sensitive, but *you* are, and when reading and debugging your scripts later, you will find it easier if you follow this advice.

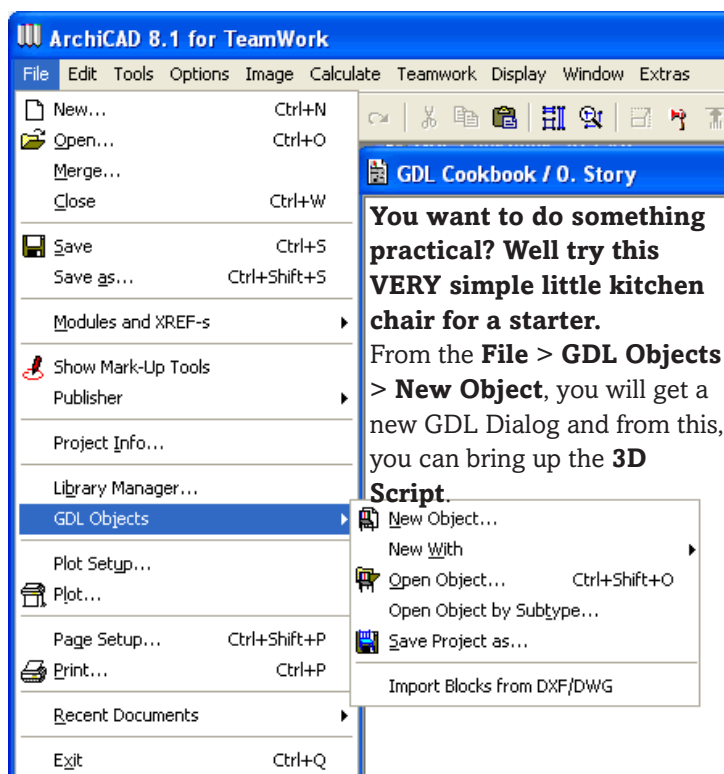
If you find yourself working on GDL as part of a team you need to agree on a format, so this is a good rule to follow. It is followed in the Cookbook.

Comma confusion?

In some countries, it is common to write decimal points as commas e.g. 1,25. In other countries it is common to write thousands with commas for example, a million is 1,000,000. If you have a localised ArchiCAD, this is how numbers may appear in the parameters box. But in GDL, the scripts are hard coded so only one convention can be followed. Dots are used to form the decimal point, and commas are used to separate numbers.

Many of the errors in your early efforts in creative GDL will be with misplaced dots and commas. Be rigorously careful with them.

First 3D object: a simple Chair



IDEALLY, you make everything parametric, but we shall start with a 3D Script for a small chair using non parametric dimensions. Later, we can convert it to parametric scripting. You can do it in Metric or Imperial.

You start by looking at the object 'with half closed eyes' and decide what simple elements it really consists of - leaving out inessential detail. Next, each object has an 'Origin' so in this case, we decide on the front left hand leg.

We build this entirely with BLOCK and CONE, to keep things simple. We move the 3D cursor around the model, drawing a BLOCK or CONE in each place. The syntax of BLOCK and CONE are very easy.

Notice that the stages of the script are marked with labels (or 'comments') which all start with an exclamation mark. When you look at the script later, you will know what's what if you have labelled frequently.

As you build each Block or Cylinder, you use the **ADDx**, **ADDy**, **ADDz** or **ADD** commands to move the cursor to the next location, then build the next 3D shape. After a group of commands is completed, get the cursor back to the start before doing the next group, using the **DEL** command - this 'undoes' the number of ADDs that you have made. Make sure that at the very end, you move the cursor back to the origin of the model using DEL.

If you do not specify a material, your objects will turn out all one colour, which can be disconcerting. So the **PEN** and **MATERIAL** command is one of the first ones you learn. **PEN 1** in my ArchiCAD is black -

This simple chair starts life from its front right leg. Do each part in stages, and move the cursor back to its origin at the end. As we develop it further, we can add more 3D options, but we keep it simple for now.

set whatever colour you want. If you do not have 'Pine, Shiny' in your Materials library, then find another wood material name to use for the chair.

This example is done in Metric and Imperial - to please the U.S. readers of the GDL Cookbook. All native GDL scripting has to be done in **Metres** (not Millimetres). If you wish to write in Imperial, you can, but each number must be clearly signified with Foot and Inch apostrophes. For Imperial users, the first things I suggest (apart from changing to Metric) is to work entirely in Decimal Inches, as this avoids spurious punctuation marks and maths operators. And of course, future scripts will be neither: they will be parametric!

2D Symbol

The ArchiCAD objects that are autoscripted from the Wall/Floor/Roof tools always have their own 2D symbol or script. When you write a script yourself, you need to create the 2D symbol.

So here, you can write a tiny script. Open the 2D Script Window and use this one-line 'killer command': **PROJECT2 3,270,2**. It is very powerful and solves your need for a 2D symbol at a stroke. It draws a Plan view of the 3D object [3], with a camera angle of 270° [270] and it is in hidden line [2]. Your 2D symbol will be given bounding box Hotspots by ArchiCAD.


```
!Simple Chair Example
!Non-Parametric, Metric
```

```
PEN 1
RESOL 12
```

```
!--All the legs-----
MATERIAL "Wood-Pine, shiny"
CONE 0.5, 0.015,0.03, 90,90
  ADDx 0.45
CONE 0.5, 0.015,0.03, 90,90
  ADDy 0.55
CONE 0.5, 0.015,0.03, 90,90
  ADDx -0.45
CONE 0.5,0.015,0.03,90,90
  DEL 3
```

```
!--The Seat and upholstery-----
ADDz 0.45
BLOCK 0.45,0.55,0.045
```

```
MATERIAL "Surface-Fabric"
ADD 0.025,0.025,0.045
BLOCK 0.40,0.50,0.005
DEL 2
```

```
!--Back Legs, panel & upholstery
MATERIAL "Wood-Pine, shiny"
ADD 0,0.55,0.5
CONE 0.6,0.03,0.02, 90,90
  ADDx 0.45
CONE 0.6,0.03,0.02, 90,90
  DEL 2
```

```
ADD 0,0.54,0.70
BLOCK 0.45,0.025,0.35
```

```
MATERIAL "Surface-Fabric"
ADD 0.05,-0.01,0.02
BLOCK 0.35,0.01,0.30
DEL 2
```

Start with the Title, set a Pen and a curve control (RESOL) early on.

For the first operation, state your material - it's better to use a parameter in future! All four cones are the same, get one right, you get them all right. Use ADD to move about and place them. DEL 3 because we needed 3 moves.

Start the seat with a new Title. The seat is a block, with a thin covering of fabric on top. DEL back to the Origin before building the legs.

Start the back with a new Title and restate material. We can use an ADD command which can go in X,Y,Z directions at one leap. The Cones are similar to the legs, but with the narrow radii at the top. Returning to the seat height (not all the way back to the origin) place a wood block for the back infill, then make another small ADD move to instal a thin layer of Fabric for the back upholstery. Then you DEL right back to the origin – and you are finished.

Now all you need is a 2D symbol. This can be generated from a short one-line 2D Script, using PROJECT2.

Now it's time to think about ways to make this parametric.

```
!Simple Chair Example
!Non-Parametric, Imperial
```

```
PEN 1
RESOL 12
```

```
!--All the legs
MATERIAL "Wood-Pine, shiny"
CONE 20", 5/8",1.2", 90,90
  ADDx 1'-6"
CONE 20", 5/8",1.2", 90,90
  ADDy 22"
CONE 20", 5/8",1.2", 90,90
  ADDx -18"
CONE 20", 5/8",1.2", 90,90
  DEL 3
```

```
!--The Seat and upholstery
ADDz 18"
BLOCK 18",22",1.75"
```

```
MATERIAL "Surface-Fabric"
ADD 1",1",1.75"
BLOCK 16",20",1/4"
DEL 2
```

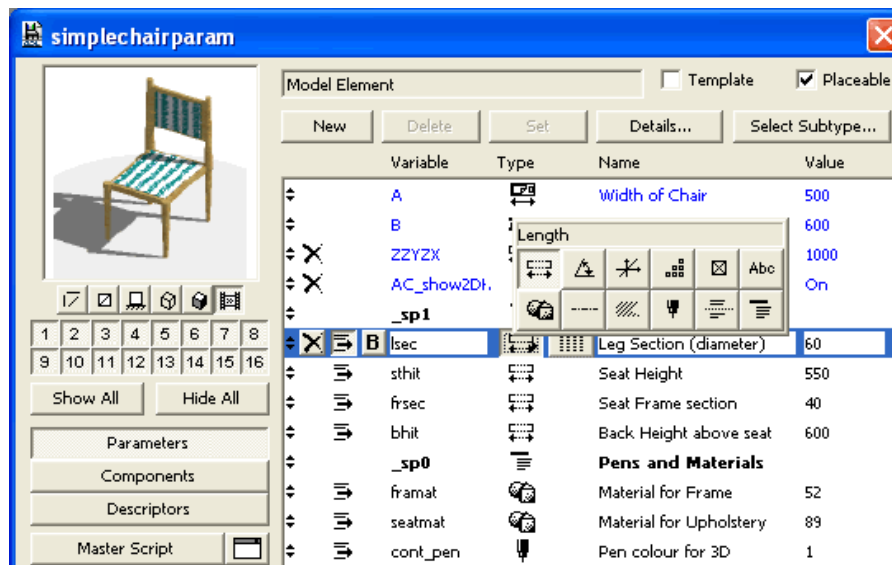
```
!--Back Legs, panel and upholstery
MATERIAL "Wood-Pine, shiny"
ADD 0,22",20"
CONE 24",1.2",0.8", 90,90
  ADDx 18"
CONE 24",1.2",0.8", 90,90
  DEL 2
```

```
ADD 0, 21.8", 26"
BLOCK 18",1",14" !Backpanel
```

```
MATERIAL "Surface-Fabric"
ADD 2", -3/8", 1"
BLOCK 14", 3/8", 12" !Cushion
DEL 2
```

```
!Chair - 2D-Script
PROJECT2 3,270,2
```

Make your Chair Parametric!



THE first chair isn't Smart and it isn't Parametric, so let's do something to make it better. Open the chair object you made earlier, and make some parameters. Click the <NEW> button a few times and make some parameters for lengths and materials. Hold the mouse down on the parameter icon, and up pops an iconic palette so you can set its purpose.

Parameters can be: Dimension (based on your system preferences, metric or imperial), **Angle**, **Real**

number, **Integer**, **Boolean** choice, **Text**, **Material**, **Line** type, **Fill** pattern, **Pen** colour, **Separator** (white space), or **Title** (bold font). Keep the parameter names to between 2 and 10 characters.

You can rearrange the order of parameters using the little <up-down> arrows, so group them into a logical order. You can put in some titles for groups of parameters, e.g. Materials, and then click on the little <indent> button to make the parameters part of that group under the title.

Make your chair parametric!

When you create parameters, change their **Variable** names to sensible ones like 'lsec' for leg section and 'sthit' for seat height. The **Description** here is what the User will see when using the dialog box that you are now creating. Put in nominal **Values** for each parameter. In this example, they are in millimetres.

```
!Simple Chair Example
!Non-Parametric

PEN 1
RESOL 12

!--All the legs-----
MATERIAL "Wood-Pine, shiny"
CONE 0.5, 0.015,0.03, 90,90
  ADDx 0.45
CONE 0.5, 0.015,0.03, 90,90
  ADDy 0.55
CONE 0.5, 0.015,0.03, 90,90
  ADDx -0.45
CONE 0.5,0.015,0.03,90,90
  DEL 3

!--The Seat and upholstery-----
ADDz 0.45
BLOCK 0.45,0.55,0.045

MATERIAL "Surface-Fabric"
ADD 0.025,0.025,0.045
BLOCK 0.40,0.50,0.005
DEL 2

!--Back Legs, panel & upholstery
MATERIAL "Wood-Pine, shiny"
ADD 0,0.55,0.5
CONE 0.6,0.03,0.02, 90,90
  ADDx 0.45
CONE 0.6,0.03,0.02, 90,90
  DEL 2

ADD 0,0.54,0.70
BLOCK 0.45,0.025,0.35

MATERIAL "Surface-Fabric"
ADD 0.05,-0.01,0.02
BLOCK 0.35,0.01,0.30
DEL 2
```

Make the Pen parametric... and add a parameter for the material for the legs.

Now we can use A as a parameter for the width and B for the depth of the chair. This means that we can make the chair stretchy.

Because blocks are built from the bottom left corner, you raise the seat block to the seatheight and then drop it by the frame section, so that its top surface is at the correct height.

The fabric material should also be parametric, this is one area where the user must have a choice!

With the cones we have made some fairly arbitrary assumptions about the ratio of width for top and bottom, and the heights of the back panel. These are 'designer's decisions' – you have to decide what the user is going to be allowed to decide, and what the designer decides. Novice GDL writers tend to make too many parameters (in their new-found enthusiasm) and then the object is confusing to the user with so many parameters.

3D Script

Take the 3D Script of the Simple chair, and change all the 2" or 0.05 to 'lsec' (leg section). Continue to work through all the dimensions changing them to parameters.

We will return to this chair later to learn about Sub-routines.

```
!Simple Chair Example
!Parametric

PEN cont_pen
RESOL 12

!--All the legs-----
MATERIAL framat
CONE sthit, lsec/3,lsec/2, 90,90
  ADDx A
CONE sthit, lsec/3,lsec/2, 90,90
  ADDy B
CONE sthit, lsec/3,lsec/2, 90,90
  ADDx -A
CONE sthit,lsec/3,lsec/2,90,90
  DEL 3

!--The Seat and upholstery-----
ADDz sthit-frsec
BLOCK A,B,frsec-0.002

MATERIAL seatmat
ADD lsec/2,lsec/2,frsec
BLOCK A-lsec,B-lsec,0.005
DEL 2

!--Back Legs, panel and upholstery
MATERIAL framat
ADD 0,B,sthit
CONE bhit,lsec/2,lsec/3, 90,90
  ADDx A
CONE bhit,lsec/2,lsec/3, 90,90
  DEL 2

ADD 0,B-0.01,sthit+bhit*0.3
BLOCK A,lsec/2,bhit*0.6!Backpanel

MATERIAL seatmat
ADD lsec/2,-0.01,lsec/2
BLOCK A-lsec,0.01,bhit*0.6-lsec
DEL 2
```

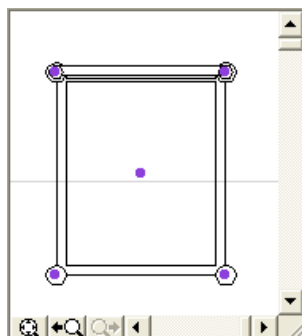
2D Symbol

We can stay with PROJECT2 to make the 2D symbol, but we can be a bit clever with HOTSPOTS. Previously we relied on default bounding-box hotspots, but now we can write our own. HOTSPOT2 x,y is the word!

Since you now converted the chair to use A and B as parameters for width and depth, you can easily define the XY positions of the HOTSPOTS in the 2D GDL.

```
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A,B
HOTSPOT2 0,B
HOTSPOT2 A/2,B/2
PROJECT2 3,270,2
```

Now the Hotspots will show in the plan. The corner ones will enable stretching, and the central A/2,B/2 spot is a good 'Pickup' spot.



Dimensional settings

EVERYBODY works in different dimensions depending on their country and regional building standards. The first simple Chair was hard coded (written into the script) in metres and in feet/inches.

The improved Chair is parametric. Most of the exercises here are in metric, in millimetres – even if the scripts are parametric, the Parameter Tables are usually metric. The settings you use do not affect the hard coding in the scripts, but they make the Parameter Table look different. If you are not sure of your Units, open a NEW Library object and look at A and B. If A is 1.00 you are in metres, and if A is 1000 you are in millimetres. If A is 3'-3.37" it's feet and decimal inches. If you wish to develop GDL in non-metric, I advise you to work in Decimal Inches.

You are advised to have a project floor plan file open in the background while you work in GDL to get the navigator palette to work correctly, and to set the dimensional environment. If you modify dimensional settings remember to set the number of decimals you can display.

About 2D Scripting

UNLESS you have a 2D Symbol or a 2D Script, your object will not appear on the Project Plan! If you been using ArchiCAD's conventional tools to make library objects, you have been used to it making automatically generated scripts, and automatically drawn 2D symbols generated from the view of the object. But now, you have to do it!

HOTSPOT2 x,y places a Hotspot at the point x,y in the 2D symbol of the object. It lights up when you select the object.

The purpose of HOTSPOTS are 4-fold:

1. To enable you to see objects when touched or marquee'd,
2. to pick up the object,
3. to make objects stretchy, and
4. to provide 'gravity' for snapping objects to line or to other hotspots.

With ArchiCAD 8, hotspots have acquired massive importance as it possible to apply hotspots to different parts of a GDL model and articulate parts of the model – e.g. opening doors and pulling/pushing drawers in a furniture cabinet. Of which much more anon!

LINE2 $x1,y1, x2,y2$: draws a line from one XY location to another XY location – relative to your current cursor position.

HOTLINE2 $x1,y1, x2,y2$: following a LINE2, makes the whole line sensitive to your mouse – easy selection of objects!

RECT2 $x1,y1, x2,y2$: draws a rectangle from one XY location to another XY location diagonally opposite – relative to your current cursor position and angle. RECT2 is always wireline (you can see through it).

CIRCLE2 x,y, rad : draws a circle at point x,y with radius rad .

PROJECT2

PROJECT2 is a command that you MUST learn, even with your first scripted Object!

Syntax: PROJECT2 Viewtype, Camera Angle, Visibility

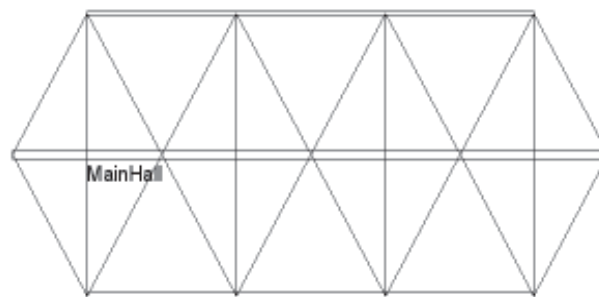
Until you learn parametric 2D Scripting, the easiest thing is to write a short 2D Script containing **PROJECT2 3,270,2**. This reads the 3D Script and draws whatever it can see, faithfully, in wireline or hidden line view. Be warned – it can slow down the time it takes to draw the symbol if the object is complex.

PROJECT2 should be learnt parrot fashion, you will use it many times!

PROJECT2 has interesting possibilities. For example, it can be told to display the object in the Project Plan in elevation or axonometric form or from the underside: very useful if you wish to generate drawings of your object from all directions without having to organise sections in ArchiCAD. Note that -7, -8 and -9 are the same as their +ve equivalent, just upside down!

Visibility value: the last digit. 1 gives you Wireline, 2 gives you Hidden Line, 3 gives a solid material quality to the 2D symbol, -1 gives you a vectorial fill.

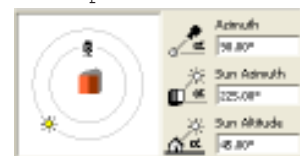
!Typical very short 2D Script for any 3D object
PROJECT2 3,270,2



Complex objects may be very slow to draw in 2D if you use the PROJECT2 command. A 2D Script using LINE2 and accurately reflecting the trigonometry of the object without showing excess detail may be perfectly adequate for the 2D symbol, as above. You could offer the user an option to show greater detail if the plan is larger than 1/50 or 1/4"-1' scale. With windows and doors, the frame sections are so small that they would just be a dense blob of ink at 1/50 or 1/100, so some intelligent 2D Scripting is very desirable.

!Typical short 2D Script for any 3D object
!using A and B for width and depth

```
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A,B
HOTSPOT2 0,B
HOTSPOT2 A/2,B/2
PROJECT2 3,270,2
```



3,270,2

4,270,2

5,270,2

6,270,2

7,270,2

8,270,2

9,270,2

-3,270,2

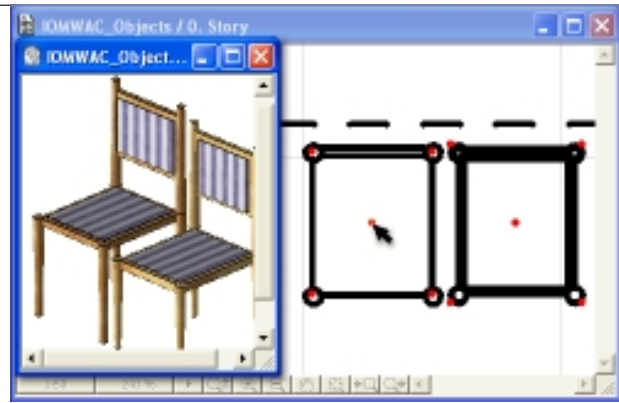
-6,270,2

2D Script for the simple chair

FOR the Simple Chair, we used a PROJECT2 command – the killer command that does most of the 2D work for you! But you can see the comparison here – Project2 (right) is untidy, with too many pen lines. A scripted 2D symbol (left) is faster for ArchiCAD to draw, and it puts you into the driving seat – enabling you to decide how the object looks. Let's write a 2D Script. Find and open your Simple Chair model. Open the 2D Script.

This chair is built with the parameters A and B for stretchiness. Have Project2 in the script at first, write your script after it, make sure that your new lines follow the outline correctly, and when the scripted version looks good enough, comment out the Project2. Put the Hotspots before the 2D statements.

You can build the whole chair symbol from CIRCLE2 and LINE2. We can't use RECT2 because the corners intersect the round legs – this looks fine in 3D, but horrid in 2D. Some skillful work with LINE2 simulates the rectangular seat intersecting the legs.



!Simple Chair Example
!Parametric 2D Script

```
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A,B
HOTSPOT2 0,B
HOTSPOT2 A/2,B/2

LINE2 s2,B-s4, A-s2,B-s4 !back
LINE2 s2,B+s4, A-s2,B+s4 !back
LINE2 0, s2, 0,B-s2 !side
LINE2 A, s2, A,B-s2 !side
LINE2 s2,0, A-s2,0 !front
```

```
!PROJECT2 3,270,2
s4=1sec/4 !Quarter diam
s2=1sec/2 !Half diameter
PEN cont_pen
CIRCLE2 0,0,s2
CIRCLE2 A,0,s2
CIRCLE2 A,B,s2
CIRCLE2 0,B,s2
```

Notice that the use of variables like 's2' and 's4' saves a lot of typing and reduces the risk of spelling errors.

PEN

PEN sets the pen colour. But remember, this also changes the pen thickness. If you use True Line Weight, this matters hugely.

On PEN, there are some points to note: PEN must have a value. You cannot have PEN zero. PEN 1 gives you a Black line – but if you have recently opened a DWG file then PEN 1 might now be Red: this demonstrates immediately the benefit of making your object parametric. People might modify the pen palette in ArchiCAD and the palette values can be re-assigned in colour and line thickness. This can have mischievous effects on your model. It is best therefore not to hard code the pen colours, but to make them with parameters so that the user can select them. So every good object should have a pen parameter.

PEN can be changed during the execution of the script, so that different parts of the model can be drawn separately. My advice is: it is best to keep the pen to a single colour in the 3D, so that the object looks right in wireline and in hidden line. Use Material as a way to vary the appearance of the object. I also recommend a separate pen for the 2D symbol, different from the one you use for the 3D. If you choose one pen for both, the pen that looks right in 3D might be too dark and chunky for the drawing scale in the 2D.

It is not essential to set any pen colour for simple objects as the pen colour can be set by the user in the main Object Settings dialog box. If you are making a GDL macro, you should avoid setting a pen parameter, because these will be inherited from the senior object.

You cannot make your own Pen

This has always been most frustrating to me. You can Define Material, so why shouldn't you Define a Pen? You can use a REQUEST() statement to suggest a PEN, and it will find the one with the closest RGB value, but you will only request it by RGB colour and have no control over pen thickness. There ought to be a DEFINE PEN r,g,b,t statement – but there isn't.

Consistent parameter names

You are advised to work with standard parameter names. GS used to use 'cpn' for contour pen, but with AC8 and IFC in mind, GS now use 'gs_cont_pen' as their primary pen name. I also recommend 'gs_symb_pen' if you make a separate one for 2D – if you have a scripted 2D object with polygons and lines and text. You will also need pen values for Background and Fills. In this case, try to use the correct Subtype, which will prompt you with suitable parameter names for most of your pens. It is worth the short time it takes to get your head round POLY2_A and B in which you can specify all these pens.

Material obtained from Pen

PEN can be used as a convenient way to create bright coloured surfaces. If you set MATERIAL to zero, then the surface colour becomes the PEN colour. This is why most of your first efforts at modelling come out as GREEN, BLACK, PINK or another colour – the material is becoming the same colour as your default pen colour if no material has been specified.

But this method produces a very matte effect. It is possible to generate a full material definition from a pen colour using REQUEST() statement. This will give a far better appearance, with characteristics like shininess, transparency and emission fully controllable by you. See later exercises for this.

More about GDL 2D

First advice on 2D Scripting

WHENEVER you can, and if you have the skill, always try to write a 2D Script with a parametric quality to match your 3D object.

2D Scripting is good because...

1. You get faster speed of drawing in the project plan. With PROJECT2, AC must do a 3D hidden line projection of the object before drawing it – slow! Scripted objects draw instantly.
2. A Symbol drawn into the 2D symbol window might be pretty, but it is not parametric.
3. The 2D Script can itself be highly parametric, error checking, allowing hotspots, informative, self labelling, and scale aware.

If it is too complex to script in 2D, and too slow to generate a PROJECT2 image from the 3D Script, consider using a symbol: use a PROJECT2, and follow the advice on the right (explode> copy> paste> fragment2> hotspot).

More about 2D Hotspots: stretching!

HOTSPOT2 x,y should be used in every 2D Script. **They make your objects stretchy:** if you use A and B as dimensions, the Hotspots at the outer corners may enable you to stretch the object. Stretchy objects are most useful and user friendly.

If the Hotspots are written as illustrated here, and if A and B are the distances apart of the corner hotspots, then the object will be STRETCHY!

ARC2 draws a circular line, which you can control by location, radius and angles (in degrees). Angles start from the Horizontal and move in an **anticlockwise** direction.

!Syntax: **ARC2** x,y,radius,startangle,endangle
 ARC2 0,0,0.5,90,180

You also have **HOTARC2** (AC9) which has the same syntax, but allows easy selection in the plan.

TEXT2 x,y,'string' is for adding text to an object's symbol. Apart from being able to label the object, the object can use it to display a level of intelligence, e.g. flashing up a text warning if the object has bad parameters, or displaying the final size of a stretched object, the pitch of a stretched staircase, or the serial number of a manufacturers component.

It is only worth using if you are first prepared to learn how to **DEFINE STYLE**; if you do not, you will have no control over the size of the text, or how it plots at different drawing scales. The text can be adaptable to different scales, including not displaying itself at all when it would be too small to read (e.g above 1/200). Define Style needs more space for explanation, so it will come later in the book.

Modifying Objects

AUTOSCRIPED objects saved from the 3D view are given a drawn symbol, and some code in the 3D Script makes them stretchy.

Autoscripted objects saved directly from the Floor Plan are fully scripted in 2D as a mass of LINE2s and HOTSPOT2s. I often delete the whole lot and replace them with a single PROJECT2 command. This is essential if you intend to alter or rotate the object in the 3D code. If you are willing to, you could try to rescript it in a much simpler, more usable, parametric form.

Another method is to write a **PROJECT2**, place the object in the Floor Plan, **Explode** the object, then **Copy** the 2D symbol thus created and **Paste** it into the 2D symbol window of the object. Add the command **FRAGMENT2 ALL,1** to the 2D Script. Retain Project2 until you get the symbol and drawing **aligned**. Then **delete** PROJECT2. You can add **Hotspots** manually to the 2D symbol. If the symbol is too full of lines, replace them with fills, much tidier.

```
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A,B
HOTSPOT2 0,B
```

```
HOTSPOT2 -A/2,-B/2
HOTSPOT2 A/2,-B/2
HOTSPOT2 A/2,B/2
HOTSPOT2 -A/2,B/2
```

Stretchy: A typical listing of HOTSPOTs when the Origin of the model is at the bottom left.

Stretchy: A typical listing of HOTSPOTs when the Origin of the model is at the centre of the model.

You can send a Hotspot directly to a location, e.g. HOTSPOT2 1.6,0.9 or you can write it the long way as:

```
ADD2 1.6,0.9           !The same result
HOTSPOT2 0,0           HOTSPOT2 1.6,0.9
DEL 1
```

Are your HotSpots Stretchy AND Squeezy?

IF you want objects to be Squeezy as well as Stretchy, you need to utter your HOTSPOT2 commands <BEFORE> the commands which actually draw the 2D objects. Otherwise, objects may stretch, but not reduce in size correctly.

You must also provide some hotspots that are not on the stretchy corners or you will have trouble picking up your object – it may stretch every time you try to drag it!

If you write any scripted hotspots at all, the default bounding box will disappear. If you WANT the bounding box, hit 'Details' button to force them to appear.

A and B based Hotspots are somewhat redundant, because Graphical Hotspots can make EVERY part of a GDL model manipulable with hotspots for sliding and rotating.

IF... THEN... ELSE... ENDIF

Or... how machines make decisions

WHEN we talk about ‘programming’ we are often thinking of the ability of our script to tell the computer to follow a path, make decisions, follow courses of action based on those decisions or upon pre-ordained rules, and thus display a level of ‘artificial intelligence’ – you can make the object ‘smart’!

The **IF** statement is the essential mechanism by which your object can make decisions.

Level 1: **IF** can be used to follow rules blindly – sound but not smart.

Level 2: **IF** can be used to allow the user considerable flexibility, and save them when an error could occur – getting a bit smart!

Level 3: **IF** can enable an object to have a great variety of 3D options each of which obey good rules, and have a good user interface – getting even smarter!

Syntax of IF... THEN...

Simplest example of an IF statement would be a one-liner like this:

```
IF A >=10.0 THEN LET A=10.0
```

Say it in English: “IF the value of A is greater than or equal to 10 metres, then let A be equal to 10 metres.”

```
IF s=13 THEN GOSUB 250
```

If the IF condition is ‘false’ (i.e. ‘s’ is not equal to 13), then the program continues to the next line. You can offer an alternative course of action in the same line:

```
IF p>13 THEN n=1 ELSE n=0
```

If the first IF condition is not ‘true’ then it will take an alternative course of action. In this example, one is setting a flag called ‘n’ to values of 1 or zero.

```
IF Curr<0 THEN
  IF cload=1 THEN
    PRINT "Holes not allowed"
  END
  ELSE
    cload=1
  ENDIF
ELSE
  IF Curr>=900 AND Curr<1000 THEN
    S900X=CurrX-OrX
    S900Y=CurrY-OrY
  ELSE
    IF int(Curr/2)%2 THEN
      IF int(Curr/64)%2 THEN
        StatusE=2
      ELSE
        StatusE=0
      ENDIF
    ELSE
      StatusE=1
    ENDIF
  ENDIF
  IF int(Curr/4)%2 THEN
    StatusP=-1
  ELSE
    StatusP=1
  ENDIF
  IF Curr>=4000 AND Curr<=4100 THEN
    GOSUB 4000
  ELSE
    SHX[PtCount]=CurrX-OrX
    SHY[PtCount]=CurrY-OrY
    SHM[1][PtCount]=StatusE
    SHM[2][PtCount]=StatusP
    PtCount=PtCount+1
  ENDIF
ENDIF
ENDIF
```

This small chunk of a script by one of the worlds most expert GDL artists, Oleg Shmidt reveals how complex you can get with nested IF... ENDIF statements. You have to keep your head about you if things get this complex!

Long IF Statements

IF statements like this can stay on one line but you may wish to refer to a long sequence of commands. This can be done with a GOSUB, or you can use a ‘Long IF statement’ which contains that routine. A long IF statement takes this form:

```
IF condition THEN !carriage return after the THEN
!routine contained in here....
ENDIF
```

ENDIF: means that it has reached the end of the long version of the IF statement.

ELSE: You could have two complicated tasks to do, one if the IF condition is true, and the other if it is false. You use ELSE to help decide whether to do one task or the other. In this case, ELSE must be on a line by itself.

Leave the first line with a ‘THEN’ hanging off the end of the line. Everything on every line after that will then work if the IF statement is true. If the IF statement is false, GDL ignores the whole group, until it reaches the ENDIF statement – logically. If there is an alternative action required (in case the first IF statement is false, then you can use the ELSE command to make it do something else.

```
IF s=13 THEN
  ADDz 1.2
  GOSUB 250
  nq=1
  DEL 1
ELSE
  ADDx 0.3
  GOSUB 150
  DEL 1
ENDIF
```

Boolean IF statements

```
IF s>13 OR s<26 THEN GOSUB 250
IF nq=1 AND s<13 THEN GOSUB 150
```

Where a Boolean choice is to be made, you can use AND and OR and XOR. You can also use brackets. Be careful with these. Speak them slowly to yourself to make sure you haven’t got it wrong.

```
IF (s<=3 AND t<=4) OR (s>=6 AND t>=7) THEN st=1
```

Nested IF Statements

As the GDL parser encounters each IF... THEN, it remembers it, and as it encounters each ENDIF, it signs it off. So you can ‘Nest’ many IF.. THEN.. ENDIFs inside each other, and the machine keeps count accurately. **You being human**, may lose count though, so I don’t encourage excessively complex nests.

Shortened IF statements

It is possible to write shortened IF statements that do not actually include the word IF. The following line:

```
var1=var2=12
```

means the same as:

```
IF var2=12 THEN var1=1
IF var2<>12 THEN var1=0
```

The time it takes to think out the shortened version takes longer than it takes to type out the long version. Which do you prefer? If you are concerned with maintenance, your scripts will be better if they are closest to spoken english – like the long form. This could be written with one line:

```
IF var2=12 THEN var1=1 ELSE var1=0
```

Boolean Parameters

One of the parameters types is Boolean, which provides you with a checkbox, for setting the parameter to ON/OFF. This has a value of only 1 or 0, and so converts the user choice to numbers. As the number can only be one or zero, IF statements for Boolean parameters can be simple. Suppose you have a parameter 'shad' for 'Shadows ON/OFF?', you can write:

```
IF shad THEN SHADOW ON ELSE SHADOW OFF
```

You don't need to say here 'IF shad=1', because it is implicit that if shad is *anything* other than zero it is assumed to be TRUE else it is FALSE.

Idiotproofing

It is easy enough to get something working in 3D, with a passable 2D symbol. But if you want to make objects for others to use, you can improve your objects by building in safeguards against bad user data or just careless stretching. We have all seen GS windows which become blank openings if stretched wrongly.

This chair we made a few pages back would fail if it was so narrow that the legs touched, or if the seat height was a negative. There are structural reasons why the seat height cannot be more than 900mm, so we need to build in some 'idiotproofing' IF statements to protect the chair from errors.

Master Script

This is where the Master Script comes into play. We can use this for some 'Idiotproofing' routines – define the limits of lengths, widths etc of the object.

The Smaller Than or Equal to <=, and the Great Than or Equal to >= are useful symbols. (Hint, the larger side of the chevron indicates the larger quantity.)

The **MAX()** statement is a way to ensure that the Pen colour cannot be zero.

```
!Master Script
!Simple Chair Parametric
!-----Idiotproofing-----
!Width
IF A<=0.3 THEN A=0.3
IF A>=0.8 THEN A=0.8
!Depth
IF B<=0.3 THEN B=0.3
IF B>=0.9 THEN B=0.9
!Seat height
IF sthit<=0.3 THEN sthit=0.3
IF sthit>=0.9 THEN sthit=0.9
!Leg section
IF lsec<=0.03 THEN lsec=0.03
IF lsec>=0.08 THEN lsec=0.08

cpn=MAX(cpn,1)

PARAMETERS A=A, B=B,
            sthit=sthit,
            lsec=lsec, cpn=cpn
```

Here the new values of A and B are being placed into the field in the Object Settings box, visible to the user.

The PARAMETERS statement could be done in the Parameter Script.

PARAMETERS Script and statement

This command is very useful. Normally the parameters table informs the scripts of what to do. The PARAMETERS command is a way for you push back new values into the table so that they are visible to the user. Pedantically I should tell you to put this in the Parameters Script, but for simple object, I like to keep the Idiotproofing and PARAMETERS statements closely associated.

This Master Script shows how PARAMETERS works. This is just a short example, you need to write more error checks in a higher quality object.

Note the syntax here:

PARAMETERS value displayed in the table = value determined by the error checking routine.

Master Script

MASTER Script is where you can do parameter calculations which are not specifically 3D or 2D, but govern the object as a whole – the results of calculations are sent to the other scripts. You can use this script to correct errors, but just as importantly you can provide feedback to the user with the corrected figures.

Master Script came in with AC6, and was a great blessing. Prior to that, any internal parameters that needed to be calculated (such as the circumference and sweep angle of an arch) had to be done twice, in both the 2D and 3D Scripts – and perhaps in the Property script too. You needed duplicate sets of IF statements to check for bad user parameters ('idiotproofing'). If you subsequently changed or added to these, you had to go to each script, copying and pasting. With ValueLists (Popdown menus), there is even more parsing of parameters to be done.

ValueLists can be built in the Master Script or in the Parameter Script.

A sophisticated GDL writer uses Master Script to define one's own Materials, set out manufacturer's rules and set up dynamic Arrays – and all other such housekeeping tasks.

GDL strobes through all the scripts and graphical hotspots repeatedly (it's a very complex circular process) but mostly it executes the Master Script first. The Master Script passes all the internal parameters on to the 2D and 3D and other scripts as required. The Parameter Script is activated when an object's settings are viewed and parameters edited.

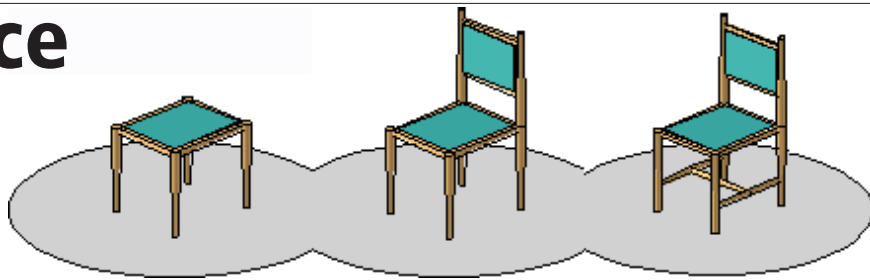
Put as much as possible into the Master Script. This way, you only have one place to check when you want to modify your error checking routines. You must never write the **END** command here – it will obey! So you do not normally use it for subroutines.

The Rule is:

- **Thinking and error checking and decision making should take place in the Master Script.**
- **2D and 3D Scripts should concentrate solely on building 2D and 3D form.**

User Choice

Boolean Parameters



3D Options: the Chair Back

THE 3D object could offer the user many options, and your user needs to know how to select them.

I prefer to offer the user a pop-down menu even if there are only two choices – the choice is made clear. But for a question to which the answer is ON or OFF, a Boolean parameter is ideal – and the answer to an ON/OFF Boolean question is simply 1 or 0.

In the chair, we will have an option to change the chair between a Stool/Table, and a Chair with the back. That is a simple ON/OFF question.

If you had options for 'No back', 'Back' and 'Back with Arms', then you would have 3 possibilities, and you would need a popdown menu.

We have to make the new parameter 'bakon' and click the popup icon palette to make sure it's a Boolean. While you are about it, make a 'Title' parameter which serves to put a title line in bold type in the parameters table.

The 3D Script is modified with a simple IF... THEN... ENDIF statement at the start and end of the script for the chairback.

Now, try clicking the ON/OFF button in your object to see the effect in the 3D Preview window.

More 3D Options: Leg Braces

There are many additional ways to enhance this chair. Let's try something easy, adding some Leg Braces. Other options could be Arms, or a Curving Back to make it more comfortable.

As with the Boolean option for the chairback, we add a new parameter, 'brace', and write a new routine at the end of the 3D Script, as on the right here.

We could have used Cylinders here but it would be unrealistic. The braces have to be mortice & tenoned into the legs, and the legs are too slender unless the tenoned ends are smaller in diameter. Braces in wooden furniture quite commonly swell in the middle, and the CONE command is a good way to do this. CONEs only build up the Z-Axis, so you have to ROTx -90 to lay the Cones on their side. Use 2 cones to build each brace.

With each brace built, it's easiest to return to the Origin using DEL, and use a single ADD command to get to the location for the next one.

Try a Pop down menu?

In a later variation, we could have 'O' braces as well as 'H' braces, and these would need a pop-down menu to decide and subroutines to perform the 3D.

bhit	Back Height above seat	600.0
Title	3D Options	
bakon	chair Back ON/OFF?	On <input checked="" type="checkbox"/>
Title	Pens and Materials	
frammat	Material for Frame	52
seatmat	Material for Upholstery	89
cont_pen	Pen colour for 3D	1

```
!--Back Legs, panel and upholstery
IF bakon THEN
MATERIAL frammat
ADD 0,B,sthlt
  CONE bhit,lsec/2,lsec/3, 90,90
  ADDx A
  CONE bhit,lsec/2,lsec/3, 90,90
  DEL 2

  ADD 0,B-0.01,sthlt+bhit*0.3
  BLOCK A,lsec/2,bhit*0.6 !Back
MATERIAL seatmat
  ADD lsec/2,-0.01,lsec/2
  BLOCK A-lsec,0.01,bhit*0.6-lsec
  DEL 2
ENDIF
```

Note that we could have written: IF bakon=1 THEN... But if the answer to the question is a simple True/ False, you can write simply: IF bakon THEN... If the zero (false) result would activate the routine, we could write: IF NOT(bakon) THEN...

Title	3D Options	
bakon	Chair Back ON/OFF?	On
brace	Leg braces ON/OFF?	On <input checked="" type="checkbox"/>
Title	Pens and Materials	
frammat	Material for Frame	52

```
!Braces
IF brace THEN
MATERIAL frammat
ADDz sthit/3
  ROTx -90
  CONE B/2,lsec/4,lsec/3,90,90
  ADDz B/2
  CONE B/2,lsec/3,lsec/4,90,90
  DEL 3

  ADD A,0,sthlt/3
  ROTx -90
  CONE B/2,lsec/4,lsec/3,90,90
  ADDz B/2
  CONE B/2,lsec/3,lsec/4,90,90
  DEL 3

  ADD 0,B/2,sthlt/3
  ROTz -90
  ROTx -90
  CONE A/2,lsec/4,lsec/3,90,90
  ADDz A/2
  CONE A/2,lsec/3,lsec/4,90,90
  DEL 4
ENDIF
```

The decision to make the diameters lsec/3 and lsec/4 is a manufacturers decision, there's no point in burdening the user with too many parameter questions.

User Choice

Popdown Menus: ValueLists

3D Options: Menus do it better

POPDOWN Menus are a great enhancement to GDL. One popdown menu can offer the user a host of configurations or choices. The popdown menu guarantees that the user cannot make a spelling mistake. Users have all used popdown menus in ArchiCAD Library parts, so it's the most helpful way for your object to be told how to perform by the user. This makes the user's decision more informed. The choices can be manufacturer-specific with proper names or model numbers. If your popdown menu offers three distinct choices for the shape of an object, the user cannot mistakenly enter a fourth and wrong choice.

GDL objects can now be much friendlier to the user, and for the developer, there is less need to write error catching routines.

Let's make a Popdown menu

Technically we call them ValueLists. Making a ValueList is easy. We have to set up the list of values in the script and offer them to the user. Make a new parameter – either text or a number – and then make a list of values for the ValueList menu in the Parameter Script or the Master Script. The VALUES command is the way to store the values. This then forms the popdown menu, in the same way that we get Materials, Pens, Lines, etc. offered to us.

If you learn to write User Interface scripts, you can create an advanced ValueList which I call a Pictorial Value List – you can offer the user a scrolling set of picture tiles of the sort that you may have seen if you have used the Door and Window libraries from GS.

My advice is that wordy (text) menus as most often the most friendly and you can write the routine in the Master Script that turns the user's choice into numbers so that it will work in the other scripts.

For example, for a smart car model, you can script a pop-up menu saying 'Saloon', 'Coupe', 'Estate', 'Convertible' and your Master Script can interpret that and use an IF statement to set a flag of values 1, 2, 3 and 4. These numbers can be used in subroutines to produce the correct shape of car.

Simply, a ValueList can take this form:

```
VALUES 'cartyp' 'Rolls-Royce','Bentley',
      'Jaguar','BMW','Mercedes'
```

Note that the parameter name is in quote marks because here we are building it, not using it. Later on your 3D Script might include IF statements like:

```
IF cartyp='Bentley' THEN
!Build the car
ENDIF
```

Making a 'flag'



The previous IF statement is case sensitive and spelling sensitive, so it often goes wrong. What you should do is to make a 'flag' in the Master Script, and work from that. The benefit of a flag is that as an integer number it cannot be spelt wrong.

```
VALUES 'cartyp' 'Rolls-Royce','Bentley',
      'Jaguar','BMW','Mercedes'

IF cartyp= 'Rolls-Royce' THEN ct=1
IF cartyp= 'Bentley' THEN ct=2
IF cartyp= 'Jaguar' THEN ct=3
IF cartyp= 'BMW' THEN ct=4
IF cartyp= 'Mercedes' THEN ct=5
!ct' is a flag
```

Later in the 3D Script you might write a simpler IF statement like this:

```
IF ct=2 THEN !'Bentley'
!Build the car
ENDIF
```

It's important that you write a small !comment! after the IF statement because you are using a flag not a string – you would have a headache remembering what all the flags meant when you look at the script later.

The best method for building and parsing ValueLists

The little routine here saves you the bother of parsing it – create some variables (e.g. "ct1") for each line of the menu, and then in the 3D you can use those variables to decide what to do.

```
ct1= 'Rolls-Royce'
ct2= 'Bentley'
ct3= 'Jaguar'
ct4= 'BMW'
ct5= 'Mercedes'
VALUES 'cartyp' ct1,ct2,ct3,ct4,ct5
```

*This method
works best in
the Master
Script*

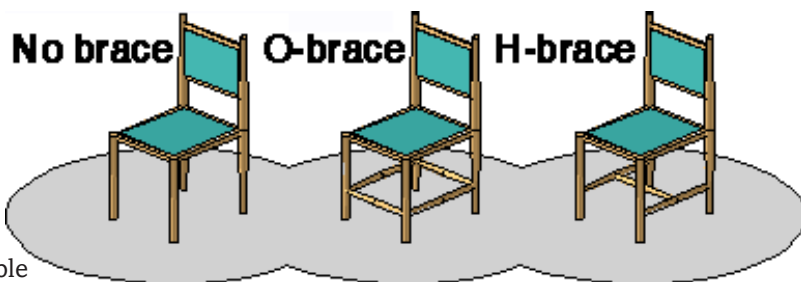
These variables are a tidy way to build the ValueList and you do not have to bother to parse the menu as we did above. Later in the 3D Script you might write an easy IF statement like this:

```
IF cartyp=ct2 THEN !'Bentley'
!Build the car
ENDIF
```

We will return to Value lists many times: there is much more to say about them.



Add a popdown Menu to the chair



WE can create an additional leg brace style to the chair. As a simple example, let's make a choice between an 'H' leg brace and an 'O' leg brace for the chair. If you include the option of 'No Brace' then that makes three, so we need a ValueList.

In fact, I use a ValueList even when there are only two choices as you can name the choices so much more explicitly than the 'On/Off' of a Boolean choice.

Do it for the chair

Make a new text parameter 'bracetyp' and build the menu in the Master Script or the Parameter script. (I prefer the Master Script). We make a simple flag 'brace' from the result of the menu. The menu decides if 'brace' is 1 or 0 now, so you can click the little X icon to hide brace from the user. In fact you can delete it altogether from the parameter table because the user no longer needs it.

We also need to modify the 3D Script that we developed for the 3D Options. The two braces that go from front to rear are unchanged.

We introduce two IF statements for the lateral braces, based on our Popdown menu parameter 'bracetyp' – and nested within the over all IF statement. Build these with CONE as for the single H-brace.

It's time to learn **subroutines** as all these braces are basically the same object and it's a lot of typing. We also need to know more complex 3D commands than BLOCK and CONE.

Now add some Arms with a Boolean

We can make another small refinement using a Boolean choice, and it illustrates a small level of intelligence: if the object doesn't have a Back, it cannot have Arms. It also illustrates the use of MUL. We can make the arm with a cone, but make it elliptical in section.

!Add this to the end of the 3D Script

!Arms

IF arms THEN

MATERIAL framat

ADD 0,0,sthith !Left

CONE 0.2 ,lsec/2,lsec/3,90,90

ADD 0,-lsec, 0.2

ROTx -90

MULy 0.5

CONE B+lsec,
lsec*0.7,lsec*0.45, 90,90

DEL 4

ADD A,0,sthith !Right

CONE 0.2 ,lsec/2,lsec/3,90,90

ADD 0,-lsec, 0.2

ROTx -90

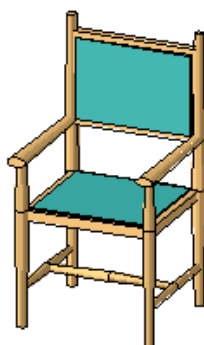
MULy 0.5

CONE B+lsec,lsec*0.7,
lsec*0.45, 90,90

DEL 4

ENDIF

!Add to the Master Script
IF bakon=0 THEN arms=0



!Add to the Master Script

!-----Pop down Menu-----

bv0="No leg braces"

bv1="O brace"

bv2="H brace"

VALUES "bracetyp" bv0, bv1, bv2

IF bracetyp=bv0 THEN

brace=0

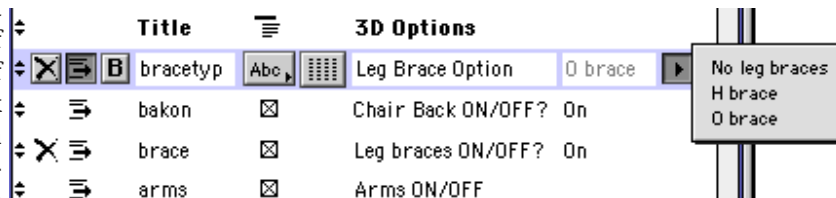
ELSE

brace=1

ENDIF

We build the ValueList with the smart method, using variables – makes parsing easier.

Here we will still make a flag, because we used 'brace' before.



!-----Braces front to rear-----

IF brace THEN

MATERIAL framat

ADDz sthit/3

ROTx -90

CONE B/2,lsec/4,lsec/3,90,90

ADDz B/2

CONE B/2,lsec/3,lsec/4,90,90

DEL 3

ADD A,0,sthith/3

ROTx -90

CONE B/2,lsec/4,lsec/3,90,90

ADDz B/2

CONE B/2,lsec/3,lsec/4,90,90

DEL 3

The two braces that go from front to rear are unchanged

IF bracetyp=bv1 THEN !---H Brace-----

ADD 0,B/2,sthith/3

ROTx -90

ROTx -90

CONE A/2,lsec/4,lsec/3,90,90

ADDz A/2

CONE A/2,lsec/3,lsec/4,90,90

DEL 4

ENDIF

Now we build in routines for the cross braces, either one in the centre for the H-brace, or two, front and back for the O-brace.

IF bracetyp=bv2 THEN !---O Brace-----

ADD 0,0,sthith/3

ROTx -90

ROTx -90

CONE A/2,lsec/4,lsec/3,90,90

ADDz A/2

CONE A/2,lsec/3,lsec/4,90,90

DEL 4

ADD 0,B,sthith/3

ROTx -90

ROTx -90

CONE A/2,lsec/4,lsec/3,90,90

ADDz A/2

CONE A/2,lsec/3,lsec/4,90,90

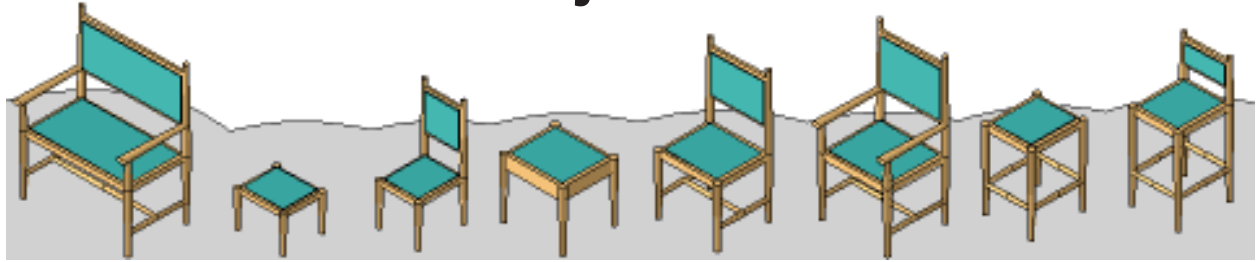
DEL 4

ENDIF

ENDIF

As these braces are almost all the same shape, it seems a pity to have to keep typing them over and over again. It's time we learnt Subroutines.

User Choice: MultiObject



Extending ValueLists

WE can make one GSM library part contain the code for many objects, in effect for it to become a super object, which I nickname the MultiObject. This is more economical than having a library of many different objects, all similar – because improvements and maintenance are easier to a few well written objects.

GDL for Manufacturing

The MultiObject is a very important concept in 'GDL for Manufacturing'. GDL is smarter than a dumb DXF object, but one facility that makes it appear smartest by comparison is its ability to operate as a Configurator: to contain configurations of a designed object, that can be selected by the user, and which when selected will follow the correct rules.

If permitted by the manufacturer then 'custom configuration' in GDL is an additional power that allows an object to be designed or configured by the user according to systematic rules defined by the manufacturer. The dimensions and attributes can be recorded in data output or in the object settings dialog that allow the now-unique object to be manufactured. In this way, one good kitchen cabinet library part can have over a million variations. CNC machines now make custom objects possible, even with small production runs.

Make the Chair a MultiObject

If the user chooses to keep this chair as a custom chair, it might be that they intend to commission the furniture-maker to make a special, and this customization is a useful visualiser of the end result. Otherwise, they are happy to use one of the standard products, and have a choice of fabrics and woods, perhaps.

This chair can be adapted to be a MultiObject. It is amazingly simple – you do not have to change the 3D Script at all – the work is done in the Master Script.

Make a new parameter [text], let's call it 'chairtyp'. Build a Popdown menu definition in the Master Script.

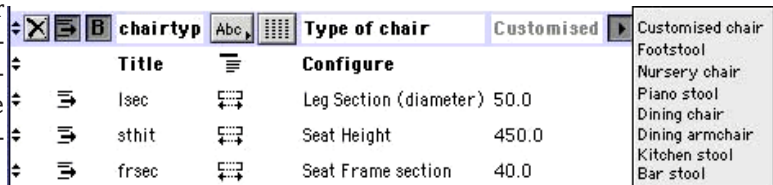
If the chair is to be one of the standard items, we should LOCK all the customizable parameters like seat height. If they were not locked, it would confuse the user who would think they could still customize parameters. Alternatively, with HIDEPARAMETER, you could hide them altogether.

We simply use the Master Script to define a value for the parameters for each of the standard versions

of the chair: for seat height, leg section etc, overriding the choice in the parameter table.

At the end we could choose to use the PARAMETERS command to show all or some of the values in the parameter table.

Should you wish to extend this chair further, a User Interface would allow you to make a pictorial menu. For the moment, ensure that your interface is tidy by using the **indents** to form parameter groups, as below.



```
!Cookbook Chair: Master Script
!-----Menu for Chair-----
cv0='Customised chair'
cv1='Foot stool'
cv2='Nursery chair'
cv3='Piano stool'
cv4='Dining chair'
cv5='Dining armchair'
cv6='Kitchen stool'
cv7='Bar stool'
```

```
VALUES 'chairtyp' cv0,cv1,
        cv2,cv3,
        cv4,cv5,
        cv6,cv7
```

```
IF chairtyp<>cv0 THEN
!Grey out custom parameters
LOCK 'A','B','sthit'
LOCK 'lsec','frsec'
LOCK 'bakon','bracetyP',
      'bhit','arms'
ENDIF
```

```
IF chairtyp =cv1 THEN !Footstool
A=0.35: B=0.30: sthit=0.28:
lsec=0.04: frsec=0.04: arms=0
bakon=0: bracetyP=bv0
ENDIF
```

```
IF chairtyp =cv2 THEN !Nursery chair
A=0.30: B=0.35: sthit=0.32:
lsec=0.04: frsec=0.04: arms=0
bakon=1: bracetyP=bv0: bhit=0.5
ENDIF
```

```
IF chairtyp =cv3 THEN !Piano stool
A=0.45: B=0.40: sthit=0.45:
lsec=0.05: frsec=0.1: arms=0
bakon=0: bracetyP=bv0
ENDIF
```

Build the menu using variables.

The VALUES statement follows.

Lock out the customising parameters.

Specify the dimensions for each predefined chairtype

Continued next page

```

IF chairtyp =cv4 THEN !Dining chair
A=0.45: B=0.40: sthit=0.45:
lsec=0.05: frsec=0.05: arms=0
bakon=1: bracttyp=bv1: bhit=0.55
ENDIF
IF chairtyp =cv5 THEN !Dining armchair
A=0.50: B=0.45: sthit=0.45:
lsec=0.05: frsec=0.05: arms=1
bakon=1: bracttyp=bv1: bhit=0.65
ENDIF
IF chairtyp =cv6 THEN !Kitchen stool
A=0.35: B=0.40: sthit=0.60:
lsec=0.05: frsec=0.05: arms=0
bakon=0: bracttyp=bv2
ENDIF
IF chairtyp =cv7 THEN !Bar stool
A=0.35: B=0.40: sthit=0.65:
lsec=0.05: frsec=0.05: arms=0
bakon=1: bracttyp=bv2: bhit=0.3
ENDIF

```

Normally I hate multistatement lines, but for this purpose it makes sense: the eye can run down the list easily.

!More parameter checking

```

IF bracttyp=bv0 THEN
brace=0
ELSE
brace=1
ENDIF

IF bakon=0 THEN arms=0

```

!Feedback to the user

```

PARAMETERS A=A, B=B,
sthit=sthit, arms=arms,
lsec=lsec, cpn=cpn

```

The PARAMETERS command is akin to the speedometer of a car :- whereas the parameter table and the "IF chairtyp" routines are driving the object (the accelerator), the PARAMETERS command is providing the user with information about the performance of the object, e.g. it's like a speedometer.

Stretchy Object: Beam Tool

THIS is a useful little object that I give to my students, and it makes 3D modelling in ArchiCAD almost as easy as using good old ModelShop of years gone by. It is simply a stretchy rectangular object that can be dropped into the plan, stretched into shape, set material and height. Put several of them together, save them and you have a new Library object without having to use GDL! It's a great way to knock up pergolas, frames, furniture etc. – easier than using Slabs and Walls. I made a glass Elevator car entirely out of BeamTool – with lots of detail – and a group of my students built an entire timber framed house!

↕ A		X Dimension	400
↕ B		Y Dimension	1000
↕ ZZYX		Z Dimension	300
↕ gs_cont_pen		Pen colour	1
↕ bmat		Material	18
↕ tilt		Tilt of Beamtool	0.00
↕ X AC_show2DH...		Show 2D Hotspots in 3D	On

3D Script

For starters, just do it with BLOCK. If you wanted to be able to vary the top, side and bottom colours, you would have to do it with a cPRISM_command.

2D Script

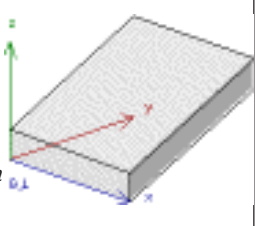
The HOTSPOT commands give the object stretchiness. When the object tilts, you can no longer use RECT2 so you use an IF..ENDIF statement to generate a PROJECT view of the object.

```

SLAB 5,higt,
0,0,0,
A,0,0,
A,B,B*SIN(tilt),
0,B,B*SIN(tilt),
0,0,0

```

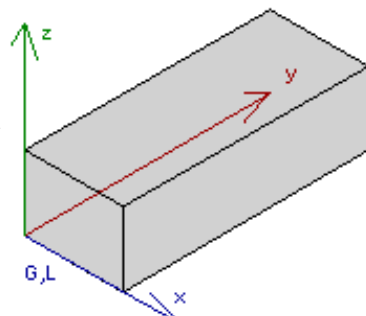
SLAB is similar to PRISM, except that you specify the z-height of each corner. Beware, if the surface is not planar, it may not show correctly.



Main value of this exercise:

- Make things stretchy – do this whenever possible.
- Using a dual 2D Script – If it is difficult (tilted) THEN use PROJECT2, ELSE when it is easy (untilted) use a 2D Script.

Here we use the classic stretchy parameters of A for Width, B for Depth and 'zzyzx' for Height. This will stretch in the 2D and in the 3D windows.

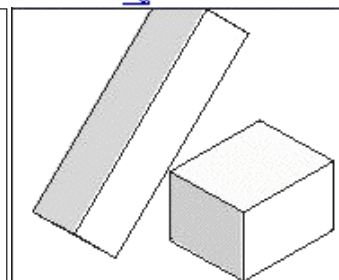


```

!Simple beamtool
!3D Script
PEN gs_cont_pen
MATERIAL bmat

ROTx tilt
BLOCK A,B,zzyzx
DEL 1

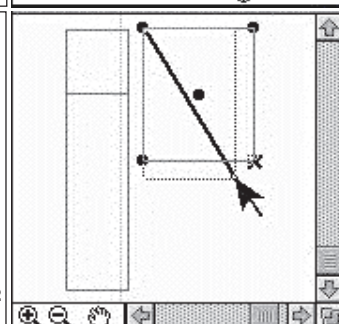
```



```

!2D Script
PEN gs_cont_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
IF tilt THEN
PROJECT2 3,270,2
ELSE
RECT2 0,0, A,B
HOTSPOT2 A,B
HOTSPOT2 0,B
HOTSPOT2 A/2,B/2
ENDIF

```



End note

If you do this with SLAB instead of BLOCK, you can turn it into a Rafter tool, with sheer end-edges.

More 3D Commands: Prisms

SOONER or later, you have to produce shapes which cannot be done with Blocks and Cylinders. **PRISM** can form complex outlines in 3D. **PRISM** (and its variants) is the most common 3D statement in creative GDL. I use **PRISM** in preference to using **BLOCK**, even for rectangular objects. Because you enter the XY locations of the points, you can stay at the origin and the Prisms can happen according to their XY locations. It is **always best to draw the Prism out on paper** before you enter the XY coordinates, numbering all the points. When you get further into Prisms, you can texture, drill, bend, curve and chamfer them. Powerful!

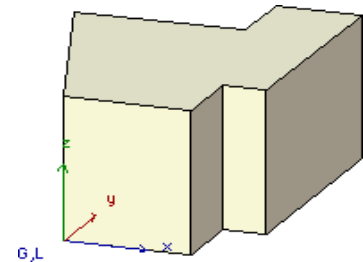
PRISM : the Ordinary Prism

PRISM is a polygonal shaped object which has the same colour all over, no hollow faces or curved surfaces, and no holes.

The local origin can be anywhere, but the prism is always built up the Z-axis, off the X-Y Plane. The first thing you write after **PRISM** is the number of points on the prism. Include the start point a second time, at the end. You should declare the **MATERIAL** property before you write the prism.

PRISMs can have *negative* thickness values and thus grow *down* instead of *up* the Z-axis.

```
!Syntax:- PRISM number,height,
!          x1,y1, x2,y2,...etc
MATERIAL "Sandstone"
PRISM 9, 0.2,
0.00,0.00,
-0.10,0.30,
0.10,0.30,
0.10,0.40,
0.20,0.40,
0.20,0.10,
0.15,0.10,
0.15,0.00,
0.00,0.00
```



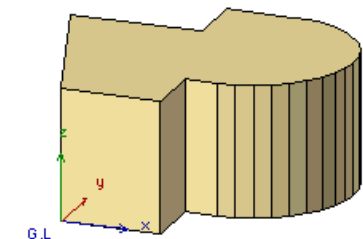
PRISM_ : Prism Underscore

CPRISM_ gives you the power to control the faces and to include curves and holes by using masking values. For now, just put **15** after every XY location; later in the Cookbook, the role of *masks* is explained in more detail.

You can finish off the final coordinate with a 15, or if you plan to add holes, you can put a negative number (-1 will do) which tells GDL that you have finished drawing the outline.

With curved edges (Polylines), you can use **PRISM** to produce Cylinders and ovals.

```
!Syntax:- PRISM_ number,height,
!          x1,y1,15,
!          x2,y2,15,...etc
MATERIAL "Sandstone"
PRISM_ 9,0.2,
0.00,0.00,15,
-0.10,0.30,15,
0.10,0.30,15,
0.10,0.40,15,
0.20,0.40,15,
0.20,0.10,1000+15,
0.15,0.10,15,
0.15,0.00,15,
0.00,0.00,-1
```



*Note, that by adding 1000 to the 15, you can create interesting tangential curves. This is your first view of the power of **Polylines***

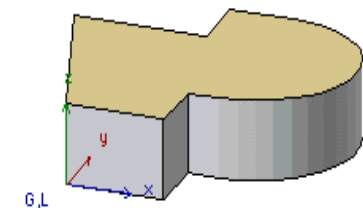
cPRISM_ : Coloured Prism

cPRISM_ is similar to **PRISM_** with the addition of a line to describe the three materials of the top surface, bottom surface and side surface. It also uses masking codes for the edges, and supports polylines (curves) and holes.

When you make a library object by the conventional method from the floorslab tool, ArchiCAD always does each floor slab as a **cPRISM_** in the 3D Script.

Note that the **PRISM_** above had many lines on the curved surface. This is explained in the section on **Masking**. Techniques of masking are learnt by a combination of thinking it out and trying it out.

```
!Syntax:-
! cPRISM_ topmat,botmat,sidmat,
!          number,height,
!          x1,y1,15,
!          x2,y2,15,...etc
cPRISM_ "Sandstone","Pine","Zinc",
9,0.1,
0.00,0.00,15,
-0.10,0.30,15,
0.10,0.30,15,
0.10,0.40,15,
0.20,0.40,15-2,
0.20,0.10,1015-2,
0.15,0.10,15,
0.15,0.00,15,
0.00,0.00,-1
```



Get the numbers right

ALL 3D commands that use XY coordinates require you to enter the '**number**' of coordinates in the statement. For example **PRISM 4,0.2, x,y, etc etc** means that there are 4 points on the prism, with a height of 0.2. This rule also applies to **REVOLVE**, **SWEEP** and all complex 3D forms.

Although most coordinate based 3D objects are self closing, its best to re-write the start point at the end, to indicate that you have closed the polygon. Also, it's a good discipline to go in an anti-clockwise direction, although I regret to say that all the examples on this page are going clockwise.

More about Prisms

bPRISM_ : Bent Prism

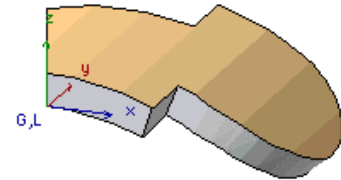
bPRISM_ (bent PRISM underscore) is similar to cPRISM, with the added ability to curve it – by adding in a value for a Radius. It always curves in a downward direction, along the X-axis.

If you want to curve *upwards*, you can issue a MULz -1 (multiply all Z values by -1) command first. You can also enter a negative value for the depth of the prism, and bring it below the origin (and thus make a tighter curve). The MUL command also reverses the materials of Top and Bottom.

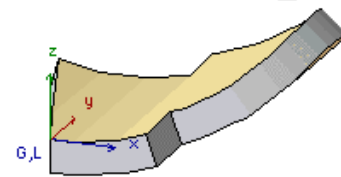
With bPRISM, you can make long helices e.g. curved rising walls. You can have a negative Z value, i.e. the bPRISM_ can go downwards.

bPRISM requires every point to have a masking value. Start by giving it 15, then change some to 13 [15-2] to remove vertical lines.

```
!Syntax:-
! bPRISM_ topmat,botmat,sidmat,
!      number,height,radius,
!      x1,y1,15,
!      x2,y2,15,...etc
bPRISM_ "Sandstone","Pine","Zinc",
      9,0.05,0.4,
      0.00,0.00,15,
      -0.10,0.30,15,
      0.10,0.30,15,
      0.10,0.40,15,
      0.20,0.40,15-2,
      0.20,0.10,1013,
      0.15,0.10,15,
      0.15,0.00,15,
      0.00,0.00,-1
```



```
!you can make it
!curve in
!the other direction
MULz -1
GOSUB 360: !bPRISM
DEL 1
```



Drilling Holes in Prisms

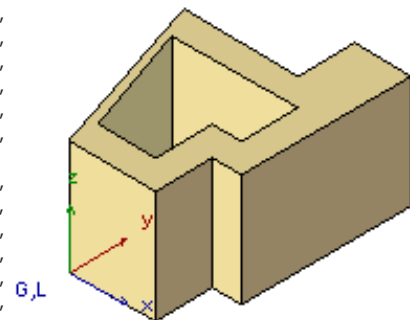
PRISMs may not be able to have holes through them, but the addition of an Underscore makes **PRISM_** which can have holes & curved lines. All the later Underscore prisms support hole-drilling.

With PRISM_, you end each XY location with a 15. The trick with holes is that you describe the outline of the object, followed by XY lists of the outlines of each hole. You end the description of the outline and each hole by repeating the first point as the last point, with a -1. You can have as many holes as you like if you follow that sequence.

My tip for hole making is to express the number of points as I have done here, e.g. 9+7, not as 16. This way, you can remind yourself how many points make the outlines, and how many are in the hole.

```
!Syntax:- PRISM_ number,height,
!      x1,y1,15,
!      x2,y2,15,...etc
MATERIAL "Sandstone"
PRISM_ 9+7,0.2,
      0.00,0.00,15,
      -0.10,0.30,15,
      0.10,0.30,15,
      0.10,0.40,15,
      0.20,0.40,15,
      0.20,0.10,15,
      0.15,0.10,15,
      0.15,0.00,15,
      0.00,0.00,-1,
      0.00,0.05,15,
      -0.07,0.25,15,
      0.15,0.25,15,
      0.15,0.15,15,
      0.10,0.15,15,
      0.10,0.05,15,
      0.00,0.05,-1
RETURN
```

The end point of each group of points must be the same as the start point.



HPRISM_/FPRISM_ : Chamfered Prism

HPRISM_ allows you to chamfer the top edge of a prism, and to have different materials for each surface. Put an angle in and you get a hipped roof effect. Put in zero for angle, and you get round edges.

Before AC8's HPRISM, you only had FPRISM, the same but without HPRISM's hillstatus code which allows you to control visibility of lines on the Hill surface, 0 for visible, 1 for smooth.

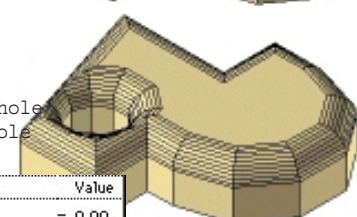
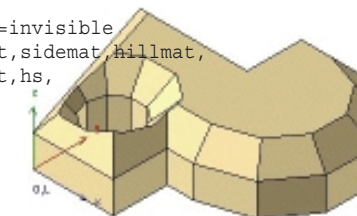
With AC8, RESOL and TOLER work correctly with both, but worked wrongly on earlier versions.

Most realistic furnishing and other objects have rounded edges; this FPRISM/HPRISM command is a real help; but whenever possible, use an angled chamfer, which suggests a rounded look without creating hundreds of polygons.

SPRISM_ is a prism that puts a Cutplane on the top surface. Frankly it's easier to use CUTPLANE.

```
!HPRISM demonstration
!Syntax:- HPRISM_ topmat, botmat, sidmat, hillmat,
!number of pts, thickness, angle,
!hill_height, hillstatus
```

```
RESOL rsl
PEN 1
hs=0 !0=visible, 1=invisible
HPRISM_ topmat,botmat,sidmat,hillmat,
      9+2,0.1,sang,hhit,hs,
      0.00,0.00,15,
      -0.10,0.30,15,
      0.10,0.30,15,
      0.10,0.40,15,
      0.20,0.40,13,
      0.20,0.10,1013,
      0.15,0.10,15,
      0.15,0.00,15,
      0.00,0.00,-1,
      0.05,0.10,900, !hole
      0.05,360,4000 !hole
```



Variable	Type	Name	Value
+	slant angle	sang	= 0.00
+	Hill Height	hhit	= 0.040
+	Resolution of curvature	rsl	= 10

Status and Masking values

STATUS & Masking values – are ways of controlling the visibility of edges and surfaces in 2D and 3D entities. Newcomers to GDL hate them, but they are worth learning about. When they are explained, the GDL user rapidly recognises the values of masking, and learns how to apply it.

Status Values

STATUS Values mostly apply to things which are 2 Dimensional in nature (even if they are 3D objects). Mostly they decide if a Line is to be drawn or missed out. POLY_ and POLY2_ are the prime examples of 2D elements that use Status Values.

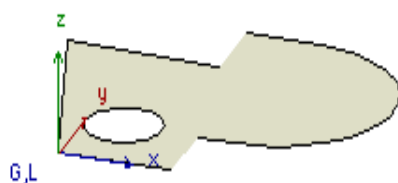
3D Scripts which define a 2D outline for 3D form-making also use Status values – for example, in defining a profile in TUBE, SWEEP and REVOLVE.

Masking Values

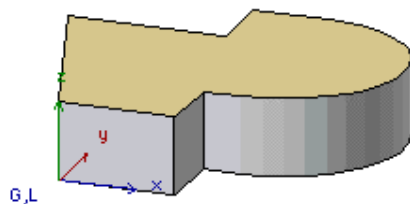
MASKING Values apply mostly to 3D objects – elaborating the details of how the object is to appear in 3D – lines and surfaces. On a 3D Prism, the Top and Bottom surfaces are always drawn. But the definition of the side surfaces and edge lines can be done with masking values.

On Prisms, there are only 0-15 permutations of lines and surface to each facet of the prism. When you are not sure and want the whole prism to be drawn, just write the XY values of each point followed by 15 – that draws everything.

If you use a mask of 8 at a point on the prism, it draws the surfaces without the lines. If you want to make one side draw differently from another, experiment with the script to make sure that you attach the different value to the correct line.

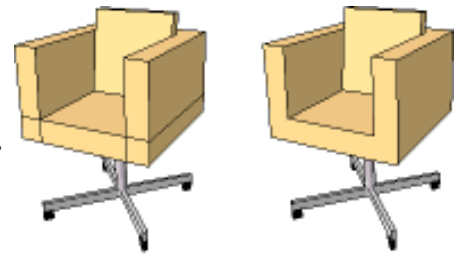


For Planar elements the choice of drawing line edges is either Yes or No



For Prisms, there are 0-15 permutations for the status of edges. This is an example of 13: no vertical lines visible on the curved surface.

See the
Masking
example in the
**Bocaccio
Chair**



Masking Values on solids

WITH more complex objects, you can declare an overall masking value for the whole thing, which is written on the first line. EXTRUDE, REVOLVE, PYRAMID, SWEEP, MASS and RULED are objects with masking values, and there are more. For safety, just start by writing 63 for most objects until you need to change it.

Certain objects also have special considerations so that some masking values apply and others are ignored. For example, REVOLVE requires a masking value for the starting angle surface and the ending surface. The masking values for this will behave differently from a SWEEP. Therefore for Masking values, you need to have handy access to a manual. The Cookbook gives a handy summary of the object masking for each complex 3D command.

Adding 64 to the object masking (e.g. 63+64) will reveal all the rough construction lines which make it. For example in a COONS or a REVOLVE, you will see all the triangulation lines.

Let's explain Prism masking again...

MASKING values are the numbers that come after each XY location in the listing of a PRISM_ (underscore) and other similar 3D objects. In Autoscripted objects, these are either 15 or 79, or -1.

- 1 draws the bottom edge line..
- 2 draws the vertical line..
- 4 draws the top edge line.
- 8 draws the side surface as solid.
- 64 smooths the edge.

You can add these yourself in any combination.

Combinations:

79 (15+64) draws everything, but smooths edges in Open GL or photorendering.

7 (1+2+4) draws all the lines, but omits the face.

13 (1+4+8) draws top and bottom lines, and the face, omits the vertical line.

-1 means that you have reached the end of an outline..

The Cookbook is full of countless examples of Masking codes in action.

Working with PRISMs: apply them to the Chair

You need to know how to define a prism outline by describing the X and Y coordinates. The simple rectangle here starts with its bottom left at the origin just like a BLOCK command. Although it has 4 corners, we count this as 5 points, because you start from the origin, proceed (preferably in a counter clockwise direction) giving the X,Y values of each point, and you repeat the first point last. In GDL the points can be defined by real dimensions (metres or feet+inches) or by parameters.

The Origin

Notice that you no longer need to sit over the origin, as with Block and Cylinder. If you can define the X,Y locations, the prism can be anywhere, with the limitation that the bottom surface is always at the Local Zero, and the height is always vertical – down or upwards. So you often have to do an ADDz to get to the right place.

Improve the chairseat with PRISM

Let's keep PRISM simple for now and see how it applies to the chair. To build the seat of the chair we have several methods.

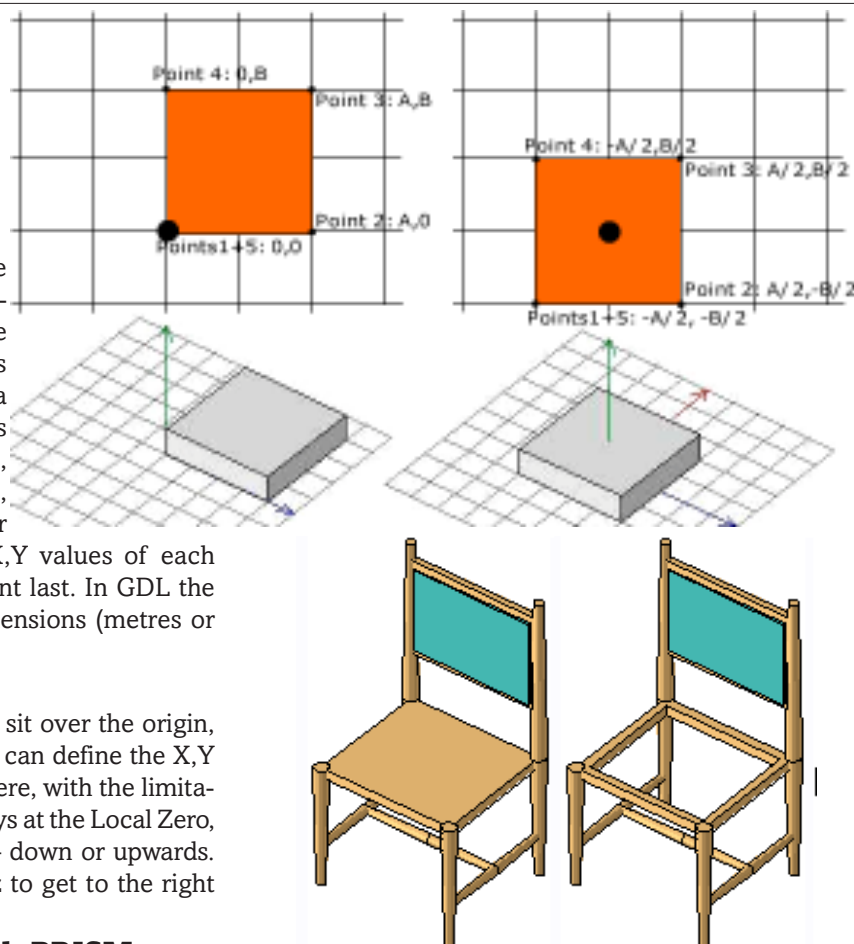
- The PRISM statement can merely replace the BLOCK statement of earlier – the origin at zero and the farthest corner at A,B.
- We could drill the PRISM with a hole, thus getting the effect of a frame. To do this, we need the prism to know when we have stopped drawing the outline and started drawing the hole. For this, we use a PRISM_ (prism underscore) and we then append a drawing code or 'masking code' to the end of each line. We use a masking code of 15 after each line, and one of minus one (-1) to denote the last line of the outline. We could then set the upholstery more realistically into the frame.
- Keeping the theme of a frame, we could build 4 prisms, one for each side of the frame. The benefit of this would be to control the woodgrain in each piece.

Masking Codes and Drilling the Hole

To drill the hole, we put a comma at the end of the first prism outline, and then continue the list of X,Y coordinates and masking codes to describe the 'inline'. The number of points in the Prism is now 5+5 or 10, so remember to change that.

Improve the chair frame - centre the frame members for more realism!

As we have a mission to improve the look of the chair, we need to centre the frame elements to the legs – we didn't do this before, to keep things simple. Centring it



!Alternative Codes for the Seat frame....
!BLOCK A,B,frsec-0.002

!Simple Prism
PRISM 5,frsec-0.002,
0,0,
A,0,
A,B,
0,B,
0,0

!Same prism with drawing codes
PRISM_ 5, frsec-0.002,
0,0, 15,
A,0, 15,
A,B, 15,
0,B, 15,
0,0, -1

!Prism with hole, for frame effect
ft=0.03
PRISM_ 5+5, frsec-0.002,
0,0, 15,
A,0, 15,
A,B, 15,
0,B, 15,
0,0, -1,
ft, ft, 15,
A-ft,ft, 15,
A-ft,B-ft, 15,
ft, B-ft, 15,
ft, ft, -1

Temporarily make a small variable of 'ft' for frame thickness

The masking code of '15' means 'draw everything' and '-1' means 'completion of an outline'. Let's assume the frame thickness is a parameter called 'ft' and make ft=30mm or 1 1/4". This could be a parameter for the user to change, but here it is defined by the manufacturer, so is written in the script.

makes more realistic mortice and tenon joints. So using a half-frame width of 'f2' we can alternate the outside and inside lines of the prism by f2 either side of the 0, 0, A, B rectangle.

Centring the frame members

!Prism with frame effect by drilling hole
!Improved statement with f2

```
ft=0.03
f2=ft/2
PRISM_ 5+5, frsec-0.002,
  -f2, -f2, 15,
  A+f2, -f2, 15,
  A+f2, B+f2, 15,
  -f2, B+f2, 15,
  -f2, -f2, -1,
  f2, f2, 15,
  A-f2, f2, 15,
  A-f2, B-f2, 15,
  f2, B-f2, 15,
  f2, f2, -1
```

To save a lot of 'ft/2' expressions, make a short term variable called 'f2'

If you look at the script adapted for the frame thickness, you can see that the outline for the hole is the same as the main outline with just the plus and minus signs exchanged - very labour saving!

!Prism with frame effect by making 4 Prisms

```
ft=0.03
f2=ft/2
PRISM 5,frsec-0.002,
  -f2,0,
  f2,0,
  f2,B,
  -f2,B,
  -f2,0
```

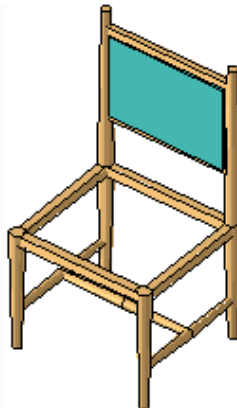
```
ADDx A
PRISM 5,frsec-0.002,
  -f2,0,
  f2,0,
  f2,B,
  -f2,B,
  -f2,0
DEL 1
```

```
PRISM 5,frsec-0.002,
  -f2,-f2,
  A-f2,-f2,
  A-f2, f2,
  -f2, f2,
  -f2,-f2
```

```
ADDy B
PRISM 5,frsec-0.002,
  -f2,-f2,
  A-f2,-f2,
  A-f2, f2,
  -f2, f2,
  -f2,-f2
DEL 1
```

To improve the chair further, it is better to build the frame closer to its manufactured sequence - 4 separate prisms for each frame section allows us to control the wood grain on each.

This change will not alter the 3D or line view, but will work better in Photorender if we write a routine later for texture directions.



Improve the upholstery

Now that we have made a hole in the seat by building a realistic frame, we need to fill the hole. Let's build some seat upholstery. We can improve the chair by softening the look of the upholstery by using a prism for the cushion and chamfering the edges. One way to do this is to use FPRISM_, the simpler version of HPRISM_.

CPRISM_

To learn FPRISM_, we first need to extend our knowledge of the PRISM. If you want to variegate the materials on a PRISM, you can use a cPRISM_ (C-prism-underscore) which allows you to specify Top, Bottom and Side Materials. This is the command that ArchiCAD uses when it writes floorslabs when you are creating autoscripted objects.

Look up the syntax for cPRISM_. You add the 3 material parameters first, followed by the same syntax as

```
!Cushion using BLOCK
! ADD 1sec/2,1sec/2,frsec
! BLOCK A-1sec,B-1sec,0.005
! DEL 1
```

Now disable this old BLOCK routine - we have something much better now!

```
!Way to do it with PRISM_
! PRISM_ 5,frsec+0.010,
! f2, f2, 15,
! A-f2, f2, 15,
! A-f2,B-f2, 15,
! f2,B-f2, 15,
! f2, f2, -1
```

Before writing a CPRISM_ you could write it as a normal PRISM_ and then add the materials.

```
!Way to do it with cPRISM_
! cPRISM_ seatmat,frammat,seatmat
! 5,frsec+0.010,
! f2, f2, 15,
! A-f2, f2, 15,
! A-f2,B-f2, 15,
! f2,B-f2, 15,
! f2, f2, -1
```

CPRISM_ now includes the materials at the start of the statement. This uses the same XY coordinates as the hole in the wood frame.

```
!Cushion Chamfered with FPRISM_
FPRISM_ seatmat,frammat,
  seatmat,seatmat,
  5,frsec+0.010, 30, 0.01,
  f2, f2, 15,
  A-f2, f2, 15,
  A-f2,B-f2, 15,
  f2,B-f2, 15,
  f2, f2, -1
```

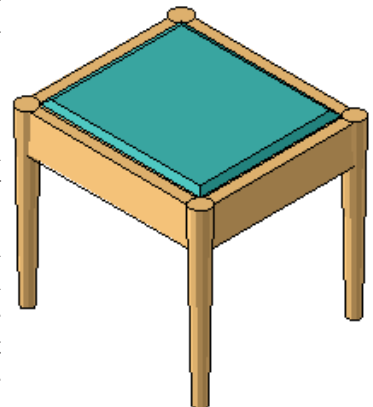
Now convert the CPRISM_ to an FPRISM_ by adding the Hill Material, Hill Angle and Hill Height. Comment out the previous Prism statements.

for PRISM_. As with the PRISM_ you must put 15 after every XY point, and a -1 at the last point. You can decide if the bottom material of the upholstery is to be the same as frame or fabric or something else.

FPRISM_

We can adapt CPRISM_ to form an FPRISM_ in which we can chamfer the edge surfaces. Because we can chamfer the top surface, we need to specify a 4th material - for the 'hill' - and follow that with two parameters after the prism height, hill angle and hill height.

The 'hill' can be curved or angled, but the curved hill creates too many polygons. For this chair, an angled hill of 300 is ideal. A hill angle of 0 would result in a curve, but do not use this option here.



Read up on BASIC!

For further reading on the syntax of BASIC, it is worth considering getting a book on it. Although these are probably more difficult to find in the 21st Century than books on C++, there must be some in libraries and well stocked bookshops. If not, you will have to learn from the GDL Cookbook!

GDL has moved on a lot from the original BASIC so it is possible to learn GDL without knowing all of BASIC.

Bocaccio Chair

Demonstration of Prism Masking

THIS chair is a simplified edition of the Bocaccio chair, to show how prism masking works in practical use. It also includes a material definition. The chair is simply constructed. If this object was for commercial use, one would build with softer edges & corners.

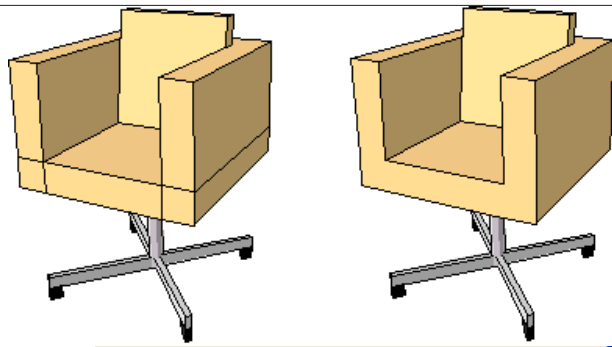
We want a cube that has a cubic chunk carved out of it. We can build it up from a seat prism and a prism for the arms. The left hand chair shows the chair with all masking values set to 15. Although the seat element is a square, it is a good idea to include two vertical lines that coincide with the inside line of the arm. This gives you the chance to remove the top line under the arm, but show it across the seat cushion.

3D Script

Look at the masking on the Prisms for the upholstery elements. 15 means 'show all', so if you write '15-4' that means 'show all except the top line'. We use Loops here which are explained more fully, later.

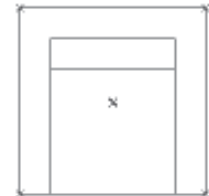
2D Script

For a commercial object one writes a proper script. It's easy to copy the outline for the arms Prism, convert them to a POLY2 and you are almost done. The chair object on the CD is fully 2D Scripted.



↕	A	↕	600
↕	B	↕	600
✕	ZZYZX	↕	Z Dimension 1000
✕	AC_show2DH...	☑	Show 2D Hotspots in 3D On
↕	armhit	↕	Arm height 300
↕	sethit	↕	Seat height 500
↕	upthik	↕	Upholstery thickness 100
↕	lgdiam	↕	Leg Diameter 80
↕	_sp0	≡	Materials and Pens
↕	setmat	☑	Seat material 14
↕	legmat	☑	Leg Material 11
↕	gs_cont_pen	☑	Pen Colour 1

```
!Bocaccio Chair 2D
HOTSPOT2 0, 0
HOTSPOT2 A/2, B/2
HOTSPOT2 -A/2, B/2
HOTSPOT2 -A/2, -B/2
HOTSPOT2 A/2, -B/2
PROJECT2 3,270,2
```



!---Master Script---

```
DEFINE MATERIAL 'blak' 4,0,0,0 !R,G,B
blak=IND(MATERIAL,'blak')
p=0.01 !one 3d pixel
dm=MAX(A,B)*1.2 !extent of leg
IF upthik>B/6 THEN upthik=B/6
```

This is an example of a Material Definition. It works best if you convert the new material into a numeric index with the IND() statement.

!Bocaccio Chair !----3D Script----

```
PEN gs_cont_pen
```

!---Seating---

```
MATERIAL setmat
ADDz sethit-upthik
PRISM_7,upthik, !Seat
-A/2, -B/2, 15-4,
-A/2+upthik, -B/2, 15-2,
A/2-upthik, -B/2, 15-4-2,
A/2, -B/2, 15-4,
A/2, B/2, 15-4,
-A/2, B/2, 15-4,
-A/2, -B/2, -1
```

We are assuming that 15 will draw everything, and we work this by deducting the lines we want to hide. These numbers could be written as 11. If you write it analytically as in this example, you can easily run your eye down the list and notice errors.

```
ADDz upthik
PRISM_9,armhit, !Arms
-A/2, -B/2, 15-1,
-A/2+upthik, -B/2, 15,
-A/2+upthik, B/2-upthik, 15,
A/2-upthik, B/2-upthik, 15,
A/2-upthik, -B/2, 15-1,
A/2, -B/2, 15-1,
A/2, B/2, 15-1,
-A/2, B/2, 15-1,
-A/2, -B/2, -1
```

It's a good idea to tabulate the values neatly in vertical line.

This point closes the prism

The arm omits the lower ink line around the outside surface.

!---Cushion---

```
ROTY -90
ADDz -(A-upthik*2)/2
PRISM_5,A-upthik*2, !Cushion
armhit+upthik,B/2-upthik,15,
armhit+upthik,B/2-upthik*1.5,15,
0,B/2-upthik*2,15,
0,B/2-upthik,15,
armhit+upthik,B/2-upthik,-1
DEL 4
```

The cushion is built on its side then rotated into position. Return to the Origin before starting the legs.

!---Leg---

```
MATERIAL legmat
RESOL 8
ADDz 0.04
CYLIND sethit-upthik-0.04,0.025
DEL 1
```

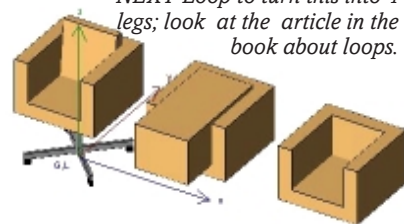
The chair is stretchy because it uses A and B. This cylinder stays at the central origin.

!Black feet, Steel legs

```
FOR k=1 TO 4
ROTY 45+k*90
MATERIAL legmat
ADDz 0.04
PRISM 5,0.04,
dm/2, -p,
0,-p,
0, p,
dm/2, p,
dm/2, -p
DEL 1
MATERIAL blak
PRISM 5,0.04,
dm/2, -p,
dm/2-p*4,-p,
dm/2-p*4, p,
dm/2, p,
dm/2, -p
DEL 1
NEXT k
```

The user is free to set a new material for the main leg – it can be stainless steel or painted. But the feet are black. To make sure that they are always black, define your own material, set it and use it.

We create just one quarter part of the leg and use a FOR NEXT Loop to turn this into 4 legs; look at the article in the book about loops.



The same job can now be done better and more easily with Solid Element Operations

Advice Corner for GDL newbies

- **Can you draw the object on paper?** If you can't you can't build it. But if you are part of a team, can you draw it out, flowchart style, as a sequence of tasks? ...and write it in the same sequence?
- **What investment value does your object really have?** Will it be used once, for your eyes only, or could it be used dozens of times in coming years by many people? Imagine someone else using the library object, and ask yourself which aspects of the model really need to be parametric? What happens if the user enters negative quantities or angles? How many choices should the user be offered?
- **Where should the model origin be?** This is the point that defines the object's height and location in the project model. It's better to start from a centre of symmetry or rotation.
- **Can your object be subdivided into 'tasks' or elements?** Look for axes of symmetry, elements which repeat themselves, elements which are clearly lathed round an axis, elements which curve or twist, elements which hinge or bend. Can you make use of symmetry? – you can build half or quarter of the model only, and then multiply or mirror the rest. If you modify parameters such as length and height, do elements which elongate also have to thicken? Will they then become larger than the element they join on to, or punch through it? Is there a logical sequence to the assembly? Do the elements have 'sub-elements'?
- **Are there hinging or sliding elements?** In how many planes do they rotate or slide? Is there a succession of moving elements, such as in the human arm? (in which, you get rotation as well as hinging!)
- **What elements are repetitious and can be done with loops?** Are some quantities unknown? Perhaps they will be generated after the user has entered a parameter – e.g., the number of rungs of a ladder depends on the height and slope of the ladder, assuming a regular spacing.
- **Which commands (in GDL) would best achieve the result?** If you analyse the model into primitives, are the elements formed from cylinders, cones, prisms, tubes, extrusions, cut objects, blocks or spheres? There are frequently more ways than one to 'cook your fish'.
- **What maths or circle geometry problems are involved?** How many arcs or circles, and can you locate their centres, and calculate their radii? Are curved shapes circular, elliptical or parabolic?
- **Can you estimate the required 'level of detail' accurately?** Do you really need to show glazing fittings, or handles on the doors? Could you use more simple planar elements in the 3D model? Is your level of detail (LOD) likely to generate too many polygons? Do you need detailed 2D symbols?
- **Are you planning to animate the model at some stage?** Objects can reduce their level of detail when the camera is a long way away, or turn to face the camera. Wheels can turn, doors open.

Graphisoft Newsletter

Summer 2000

on 4/7/00, Sarah Elliott (Graphisoft_HU) wrote:

What are the most important things you tell the people who attend your GDL classes about object making?

DNC Replies:

Here are the five most important!

First thing is....

•Dont Panic!!•

Its not as difficult as you expect once you are into it, and when you get an error message, read the message, look at the script; have you spelt the parameters right? Are your commas in the right place? Try excluding part of the script to see if other parts work. GDL has few bugs; for beginner level GDL there are none to speak of; so it's a relief to know that the error will be found through hard logical thinking.

Second thing is....

•Always have pen and paper handy•

If you can't draw it on paper, you can't write it in GDL. Drawing it out on paper will put your mind at ease to concentrate on the logical process of actually building it. For many commands such as prisms and revolves, half the job is done if you have drawn it out, preferably marking the origin and numbering each point along the profile.

Third thing is....

•Think in parameters•

Always try to work (and think) in parameters not in numbers – it's easy to develop your own lingo of 'wid', 'dep', 'len', 'hit' and 'thik', and easier to write in these terms knowing that a simple change in the parameter value will modify your object easily.

Fourth thing is...

•Steal from ArchiCAD•

Some objects are so complex that the parametric method above is harder than beating your head against the wall. You may be better sketching them out in the ArchiCAD floorplan and then dragging them and dropping into the script. You may have to settle for making them parametric only by stretching and mirroring.

Fifth thing is....

•Structured GDL is best•

The moment it gets more complicated than one page full of script, convert the script to subroutines, so that each part of the model is a tiny object in its own right. This also saves on repetitive typing, and makes it easy to isolate errors.

First published in Graphisoft Newsletter, Summer 2000

GDL Miscellany

END and Line Label numbers

END is one of the first things I write, after I have written a Title of the object, the Date and my Name. It is because everything in front of **END** is the 'Executive script' and everything after **END** is going to be a useful Subroutine that needs to be stored until used. Even the most complex of objects need be no more verbose than the example here. It is immensely easier to detect errors or make changes if the script is written as subroutines. Put a line of dashes after the **END** statement.

```
!Timber chair
!Date and Author
PEN gs_cont_pen
RESOL 12

GOSUB 100:!All the legs
GOSUB 200:!The Seat and upholstery
IF bakon THEN GOSUB 300:!Back
GOSUB 400:!Braces
IF arms THEN GOSUB 500:!Arms

END:!  
-----
100:!All the legs
MATERIAL framat
CONE sthit, lsec/3,lsec/2, 90,90
ADDx A
CONE sthit, lsec/3,lsec/2, 90,90
ADDy B
CONE sthit, lsec/3,lsec/2, 90,90
ADDx -A
CONE sthit, lsec/3,lsec/2, 90,90
DEL 3
RETURN
```

LINE label numbers are always written with a colon (e.g. '250:'), and should be followed by a text comment (e.g. '!Chair legs'). They do not have to be in sequence, but it helps you if they are.

Early forms of BASIC required a number on EVERY line, which is why numbers were spaced with increments e.g. 10: 20: 30: 40: etc. – enabling you to insert extra ones later. Now you have freedom to write lines without numbers except when you need to Label. When you use subroutines, there is nothing to stop you using large numbers like 100: 1000: or 10000: – they are merely labels.

PRINT

Error messages, Debugging and Idiotproofing

PRI~~N~~T is a way of getting GDL to tell you or the user something – it is used in pointing out errors, or in debugging. The warning box only shows when you display in 3D and you need to have error messages permitted in Options>Preferences.

Error Messages

My advice is to NOT use it for telling the user something unless you have a very important object, and a very important message to tell the user. If you do it for commonplace objects like windows, the user could be faced with 20 or 50 identical shouting warning boxes, and would not know which object was wrongly set. They may not see it if they have the option turned off.

For errors, it is far better to write 'Idiotproofing' statements in the Master Script where your object quietly corrects the errors made by a user e.g. if the timber window is less high than 400mm, the horizontal transom can be omitted, or the window is locked at 400mm (you decide which). With the PARAMETERS command you can feed information back to the user by showing a correct value in the Parameter Table.

Another way to correct errors is to have the 2D object flag up an error. Some GS objects bleep a message in the 2D "Bad Parameters!". If Idiotproofing is included, then stretchy objects will stretch to a safe size and stop. Another way to make objects safe is to have a ValueList to set the safe sizes for something – e.g. door opening angles. When you open it, the door snaps to safe angles.

```
slenr=colht/colwid !Slenderness Ratio
IF slenr>24 THEN
  PRINT "Warning, Column Slenderness Ratio is:",slenr
ENDIF
```

Warning, Column Slenderness Ratio is: 36



Stop

Continue

Debugging with PRINT

PRINT is most useful during the writing and debugging process, e.g. to see if a calculation is giving the right result. Execution of the 3D Script is halted at the Print command, so it's a good way to test something even when it isn't fully working. You can then hit 'Continue' to see the rest of the script working.

It's also useful if you lack a pocket calculator capable of Trig or complex functions. Type in a quick expression into the 3D Script, click on the 3D view and get the answer.

ValueLists can provide 'snapping'

VALUESLISTs can do more than provide a popdown menu. You can also save a lot of work with IF statements by using them to define a range of values, and you can even set the intermediate stepping values. Objects using Graphical Hotspots will adopt a distinctive 'snapping' mode.

```
VALUES 'len' RANGE [0.05,1.0] STEP 0.05,0.05
VALUES 'wid' RANGE [2",3'-4"] STEP 2",2"
```

This will set the values to a range from 50mm to a metre snapping to 50mm increments.

What is zzyzx ?

ZZYZX is a 'dynamic parameter' denoting **HEIGHT**. **A** and **B** are used to denote width and depth – ArchiCAD recognises them as special 'magic' parameters that can make objects stretchy. They are given special prominence in the Settings dialog box. **Zzyzx** is the magic parameter used for **height**, and if you use it, the height will be displayed in the Info Box Palette. Objects with **zzyzx** can be stretchy in the 3D view – a 3D hotspot set

will appear at the height specified by **zzyzx** and if you grab the spots, the 'pet palette' displays a 'stretch vertically' option.

With new graphical hotspots since AC8, the usefulness of **A**, **B** and **zzyzx** are reduced because they assume that stretchiness works on a theoretical cuboid surrounding the object. GHSs make every part of an object stretchy.



MODEL

MODEL: sets a viewing mode for the model, or parts of the model. With the MODEL command, you can single out parts of the model to draw differently from the rest.

With a Boolean Yes/No command in the dialog box, part of your object could change from shaded to transparent wireline.

In GDL Solid Element Operations, you could use wireline to show the outline of bits that had been subtracted. They would disappear in renderings, but show in Open GL and 3d hidden line.

Syntax:

MODEL WIRE: MODEL SURFACE: MODEL SOLID

SHADOW

SHADOW: sometimes you must turn off shadow rendering even when the rest of the object is shadowed – for example glazing fittings – which if curvy and casting shadows could prevent ArchiCAD ever finishing a render. In a model of a car, you can speed up rendering by issuing a SHADOW OFF command before drawing hubcaps or door handles.

SHADOW ON and SHADOW OFF can be used in IF statements – for example, if the object is further than 50 metres from the camera location then do not cast shadows.

Example:

```
IF dd>30 THEN SHADOW OFF ELSE SHADOW ON
```

The wickedness of GOTO

GOTO is something that I never use, as going to is the worst thing that you can do if you believe in the structured programming approach. You can issue a command such as: 'GOTO 250', and it will jump to that numbered line. It doesn't care if the number is part of a subroutine, it doesn't check as to whether it is already in a subroutine. It just jumps, even if it is illegal. Chaos can be the result. If you want to modify such an object, it may be impossible and you may find that it's quicker to rewrite the object from scratch.

GOSUBs are OK because they return after they have completed; they remember where they came from. GOTOs have no way to return safely; the execution of the programme carries on like a runaway horse.

Early Graphisoft library objects are littered with dozens of GOTO's, perhaps to discourage users editing the scripts. Some commercial GDL writers use GOTOs to confuse a reader by making it difficult to find the real code.

The use of Subroutines, Loops and IF.. THEN.. ELSE commands should give you ways of getting round every possible situation where you might want or need to use GOTO.

Before FOR... NEXT Loops were invented, one could make something that behaved like a loop, using GOTO and k=k+1 to keep count. But now, there should be no need for it in clean scripting.

If the GOTO is going to a location so close that it is within 'eyeshot', i.e. when viewing the script the destination can be seen without scrolling, then I would reluctantly accept its use – just (if the numbering system was rational and sequential).

One acceptable use of GOTO is as a substitute for the END statement in a Master Script. You cannot END here – or the object will not execute the 2D or 3D Scripts. So if you have a command GOTO 99999 instead of END, you can use GOSUB and subroutines in a Master Script, with a 99999: at the final line of the script.

```
p=0
if p GOTO 1
PRINT "hello"
1:END
```

```
p=1
if p THEN 1
PRINT "hello"
1:END
```

You can even omit the THEN, or omit the GOTO – as long as one of them is present. But why am I telling you this? It's just bad to use GOTO except in emergency.

Cascading parameters

FROM ArchiCAD 6.5 onwards, it has been possible to organise the Parameters table into logical groups – you see this in all the objects in the ArchiCAD Library.

While building the GDL object, you see the usual box of parameters with the scroll bar. You should now organise the parameters into tidy groupings with bold titles, popdown menus, and line spaces. Emphasis very important parameters with a Bold font.

With the emphasis on IFC and the use of subtypes, the number of parameters that we use in GDL has increased alarmingly. When you set an object to a subtype, you get many many parameters created for you in blue type, that are standardising some of the

2D representation and ArchiFM parameters. Use these. But you will be annoyed that new parameters created by you always start life at the bottom and have to be moved upwards into position. If you do not want these to be visible, hit the 'X' button so they are hidden from the user.

UI: Moving beyond the Parameter table

It is possible to build a custom User Interface that does away with the traditional parameter box altogether and makes it possible to build objects that have the sort of interface you normally expect from a well written API add-on. You could also build a set of duplicate user interface windows in different national languages. This is the subject of extended study later in the book.

Now, let's cascade!

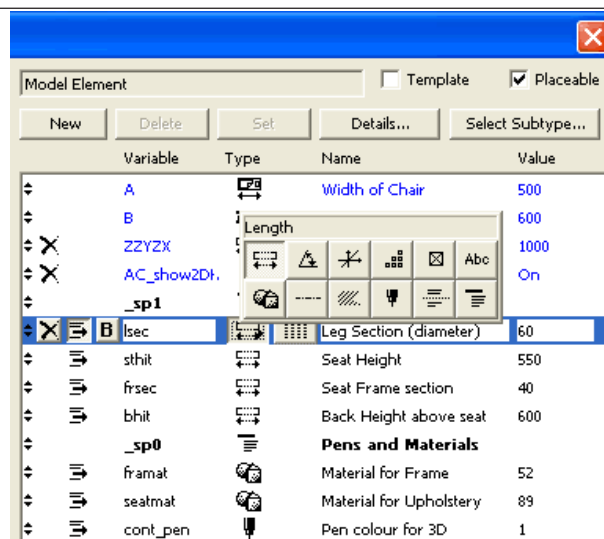
AS you build each parameter, you have a little vertical gripper at the extreme left with which you can change the order of parameters. You also have three buttons to touch on; the first one, with a cross (as in 'ZZYZX' here) means 'hide it'; perhaps you are not using it, or it is ONLY to be used in the advanced User Interface boxes. The second button with an arrow means 'indent the parameter name', making the parameter hierarchically subsidiary to the nearest unindented above it. The third button with a large 'B' sets the parameter description to Bold, even to ones which are indented. You can indent a parameter under another, you are not restricted to doing it only under titles.

Typical groupings would be 'Pens and Materials' or '3D options'. By organising the table into these groupings, you get triangular 'rocking' buttons, giving you 'cascading parameters' – the user can open groupings according to choice and need. You should make a grouping when the number of parameters exceed 6 to 7 – it is considered very bad GDL organisation to make the user have to scroll deep down the table.

Although you need to make all the parameters here, you can hide them from the user with HIDEPARAMETER command, to keep the box tidy. If you hide the top parameter of a group (under which the rest cascade), using either a cross or a command, you will hide the entire group. Titles are parameters with names, they can be hidden too.

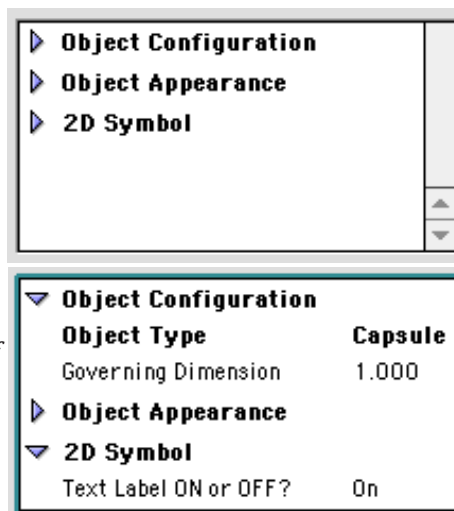
If you use **A**, **B** or zzyzx these should remain untouched at the TOP of the parameter table and not in any hierarchical group. When the object is saved and then viewed through its settings box, A, B and zzyzx become main Infields, not in the Parameter Table.

Value lists automatically activate if they find a valid script in the Parameter or Master Script. In the advanced UI, you can have Pictorial Value lists.



This is a user's typical parameter box all neat and tidy and folded up.

This is the same user's parameter box with 2 of the groups opened up ready for inspection and use.



ArchiGuide for Library Parts.

ArchiGuide is a wonderful facility, so good I wondered whether it was worth writing the GDL Cookbook edition. But you can have both! Try it at:

http://www.graphisoft.com/support/archicad/archiguide/library_gdl.html

Making it friendly for the User

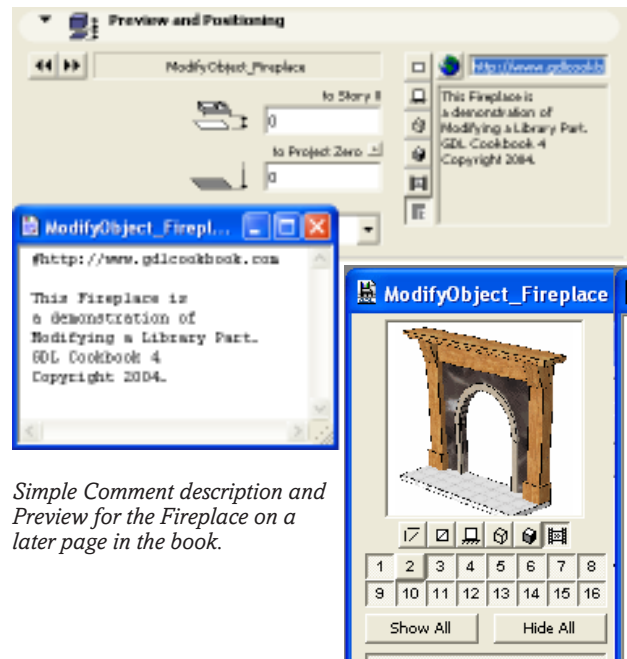
There are two oft-overlooked buttons in the GDL editing window, **Comment** and **Preview** – all good objects should use them!

COMMENT: is a good way to document your object so that a user has a guide as to its author, its purpose or to how it functions. If you are building objects for sale, the comment box could be like a tiny manual – or a copyright notice. You can place a Web address in there preceded by a hash sign, and objects may enable users to refer to a website e.g.

#http://www.gdlcookbook.com

Comment has a no word-wrap, so you have to write it without carriage returns, and check out the wrap by saving and viewing the settings. It cannot scroll, so you cannot write much.

If its important to provide a lot of information, you have two other methods, an extended Popdown menu, or a scripted User Interface.



Simple Comment description and Preview for the Fireplace on a later page in the book.

This Popdown menu is very valuable in this Norwegian window object for two reasons.

All of the parameters have been concealed except this one, because the entire parameter set is displayed in the User Interface. Secondly, the menu can be arranged to show itself in English or Norwegian, it is very easy to switch language.

PREVIEW PICTURE: is used to show what the object should look like. You can place a picture from the clipboard straight into the Preview Picture Window – or better still, this could be one you made earlier with Artlantis.

To make a Preview picture, marquee a good looking Render or Open GL image of the object, making the marquee as square as possible. Open the Preview window (grey button in GDL) and paste it in. It will be resized as near as it can to 128x128. Line images do not paste well, you need enough body in the image to have something to click on when you want to select it in a browser.

Every object you make should be equipped with a Preview image, even autoscripted ones. If you use a PC, your Preview image will become your icon in Windows. In the GDL browser window, the Preview image is the main means of identifying the object.

In a really complex object, you can write a UI showing the many variations of the object, but this necessitates storing picture files in a loaded library.

Sometimes in AC8, Preview images do not survive if GDL objects are sent by email – whether from Mac or PC. The solution is to wrap them in an archive, zip the archive and send that.

Organising the Parameter Table

We already touched on the value of clicking the 'indent' button to drop organise parameters into tidy groups. You can also use the following commands to life easier for the user.

PARAMETERS: This makes the result of idiotproofing or calculations visible to the user.

LOCK: This leaves the parameter visible to the user, but greys out the data and does not permit the cursor to be clicked in the infield – good for display.

HIDEPARAMETER: This enables you to hide a parameter if it is not required. You may have a table that can have either cylindrical or square legs. If the user selects the square leg, the parameter for the diameter should be hidden. It is a pity that there is *not* a SHOWPARAMETER command – for those occasions when you want to be sure it's visible.

```
IF A>=2.0 THEN A=2.0
PARAMETERS A=A, B=0.6,
          zzyzx=0
```

```
LOCK 'B'
HIDEPARAMETER 'zzyzx'
```

Note, PARAMETERS: names are not in quotes.

HIDEPARAMETER and LOCK: names are in quotes.

This limits 'A' to 2 metres and informs the user of that; it defines the other sizes.

'B' is uneditable, but here the user still needs to see it. The 'zzyzx' can be hidden altogether from the settings palettes.

How not to write GDL?

IF you are working at home, you may have looked through pages in the GDL Cookbook. You may have begun to get some ideas about good style and form in GDL. Look at the script on this page, and **CRITICISE!** Come back to it later for another read when you have done more scripting.

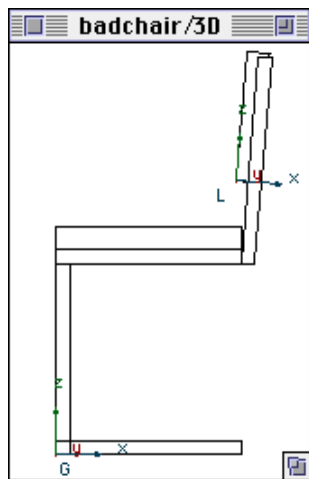
Do not type this script in!!

....it's an example of the untidy kind of scripts you write when you first start writing GDL scripts.

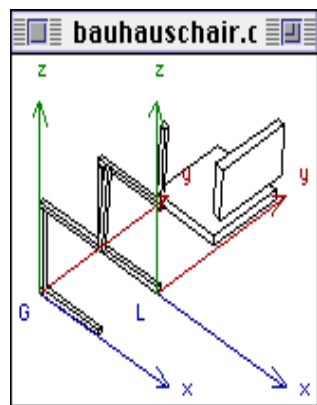
Let's notice the good point(s) first:

- The script results in a chair! Therefore, it is not all bad!!

```
! Tubular Steel chair
! badly written GDL
Pen 1
! Draw Legs
  block .03,.03,.45
  block .45,.03,.03
  addz .45
  block .45,.03,.03
  addx .45
  roty 5
  block .03,.03,.45
  roty -5
  addz -.45: addx -.45
  addy .45
  block .03,.03,.45
  block .45,.03,.03
  addz .45
  block .45,.03,.03
  addx .45
  roty 5
  block .03,.03,.45
  roty -5
  addz -.45: addx -.45
  addy -.45
Material "Chrome"
!Seat
addz .48
Material "LimeStone"
!Seat
block .45,.48,.05
addx .44
addz .15
!Back
  roty 5
  addy .03
  block .05,.42,.28
```



The side elevation shows that it still works, even though it is a bad script!



If you make mistakes with a sequential script like this, you get tangles like the one here – and you will find it very difficult to disentangle the distorted framework.

```
!2D Script
PROJECT2 3,270,2
```

Criticism of Script

- The script is 'sequential' instead of being organised into a modular structure. For example, in this script, the entire leg is written once, and then it is written all over again, nearby. This is wasteful, and difficult to edit later. The legs, being the same, should be repeated in a neater way, either using a **Loop** or a **Subroutine**.
- The PEN colour should also be a parameter in case pen colour 1 is not what the user wants.
- Make even more use of **Comments**, **Indents** and **Carriage Returns** in the script to make it easier to understand. Some indents are used in this example, but there is no logic to them. Indent cursor movements.
- The Model starts with an origin in the bottom left corner, although it would be better for a symmetrical object to start from the Centre line.
- The material for the legs is specified after they have been drawn. They will turn out green!
- The entire script is written in lower case. Although lower case is easier to read, it is more difficult to pick out commands from comments and variable, and it is therefore more difficult to edit the script. **Always** use **UPPER case** for GDL code and **lower case** for comments, string and parameters and variables.
- The script makes occasional use of **multistatement** lines using the colon: – eg addy -.03:roty -5 on one line. This is legal syntax, but is bad because you will have problems debugging the script.
- It is bad to write numbers in the form .03. Always give low numbers a leading zero, e.g. 0.03.
- It is not parametric. The script specifies all materials and dimensions very specifically – whereas you should provide the user with all the parameters they need in the dialog box. All these material names change in new versions of ArchiCAD and in different countries, so they will not be recognised by GDL.
- At the end of the script, no attempt is made to bring the cursor back to a final resting place at the origin.
- Although the author brings the cursor back before beginning the next operation (most commendable!), there are far easier ways to do it; instead of wriggling backwards in reverse, use the DEL command.
- PROJECT2 is quite legal in the 2D Script but it is always better to try a real script with RECT2 and LINE2 commands. Also, you should add in some Hotspots.

This cookbook is all about learning to used 'structured programming' techniques. It is easier to learn from mistakes than from successes – so please look at the criticisms on this page when you have tried some scripts yourself.

So, what is Structured Programming?



IN the context of GDL, it means taking a very disciplined approach to model building, in particular these three rules:

ANALYSE the model to understand all aspects of its symmetry, repetition, rotation, sliding and other essential geometries.

ORGANISE the script into a number of subroutines and/or macros that reflect the 3D organisation of the object.

MAINTAIN strict control over the location of the 3D cursor. Return to the origin after each operation.

FOR small models, you can write BLOCK this, CYLINDER that, CONE the other and finish up with simple models – the scripting may not be elegant, but it works.

However, if you make the model bigger, you will produce a script of spaghetti-like complexity. A week or two later, you won't remember your way round your own script! By contrast, if you structure your 3D Script with more discipline, you can build models of great complexity without losing track of what you are doing – and you are able to repair it or amend it without despair. Professional programmers despise BASIC because they associate it with amateurishly written, unstructured code that only just works, that cannot produce merchantable applications, that can only be debugged by the original writer, and so on. The virtue of systems like PASCAL, LISP and 'C' is that programs are 'object oriented', modular, structured, and so on. Perhaps this is why AutoCAD uses Lisp, and why MiniCAD uses PASCAL for their equivalents of GDL.

But listen! BASIC, as used in GDL, does everything that is needed to produce 3D objects, and is perfectly capable of being written in an 'object oriented', modular, structured way, that can be debugged by other people and which can produce library objects that are of merchantable quality. It's just the way you do it!!

3D Model Analysis

SCRIPING is easier if you understand the 3D nature of the object. All that really matters is that you analyse the object accurately, regardless of the human or computing language you are using or thinking in. For architects and designers who are in the business of 3D, this should be the easiest part.

Think these out on paper: if you cannot draw the object freehand in pencil, then you cannot script it.

The starting language of 3D is the 3D primitive – the Block, Cylinder, Cone, Sphere, Extrusion etc. As you elaborate your vision, you have prism shapes which curve, have drilled holes, rounded or chamfered surfaces; then you employ extrusions taken through curved or chamfered angular pathways, saddle shapes and surfaces, lathed objects, elements repeated around axes, and many such variations.

At the next level of perception, objects contain a set of **assemblies**. For example, the jib of a tower crane is not a slab or a cylinder, it's a complex 3D lattice assembly, but it still needs to swing on its swivel on the tower as a single object. If the tower is raised in height, the jib must go up too.

In complex objects, there is always a **hierarchy of organisation of assemblies**. The traveller mechanism holding the hook (that drives on rails along the jib) is another complex mechanism, and should be treated as such. But when the jib swings, the traveller must swing too. The hook and the cable assembly with it must both swing with the jib and move with the traveller, because they are 'below' both the traveller and the jib in the hierarchy. Assemblies which might change, like lights which come on, elements that want to slide, doors which open or rotate, should be treated as objects. In GDL you organise these using **subroutines** or **macros**.

Origin and 3D Cursor

THE location of the 3D cursor is something you must control tightly: you need to return to the origin before you tackle the next major part of the model. If you have a model with a complex hierarchy of assemblies, you depend heavily on your subroutines to make sure that the cursor location is always safe. Never use DEL TOP in a structured model because it blows the floor out from your hierarchical structure.

Look at the scripts in the GDL Cookbook, and you will see a constant re-iteration of the value of Structure in programming.

About Subroutines...

Re-read these rules frequently when your first attempts do not work correctly!

THE essence of structured programming in GDL is that 3D models should consist of subroutines which reflect the 3D nature of the model.

Another benefit of using subroutines is to avoid ever having to type out complex GDL routines more than once.

The example here is what you should be aiming towards. Every **GOSUB** command is telling GDL to **GO** and run the **SUB**routine of that line number.

The 'Executive Script'

THE first part of the script (as far as the **END** statement) can be thought of as your 'executive script'. This is the main controlling script, that decides on what will be drawn, and in what order. When GDL reaches the **END** statement, it stops. After the **END** statement you carry a lot of useful baggage called Subroutines. Each element of the model could be a subroutine – analogous to an assembly in the 3D. They can only be used if they are called by the executive script, or by another subroutine. The **END** statement prevents them from being read accidentally.

It is a good idea to keep the executive script short and decisive. When you want to repair, modify or extend the model, you will know exactly which subroutine to go to. Make sure that every **GOSUB** command has a comment attached so you know precisely what it is doing.

When assigning number labels, keep them well spaced (e.g. in 10s or 100s) so you can insert extra number labels if you think of more things to add later.

```
!Swivel Chair - 3D executive script
!Date and Author
```

```
GOSUB 100: !Foot of chair
GOSUB 200: !Shaft of chair
GOSUB 300: !Seat of Chair
GOSUB 400: !Back of Chair
END !-----
```

*A typical
Executive Script.
Mark the END statement
with a row of dashes as a
clear break between
executive script and
subroutines.*

The 'Subroutine'

THE subroutine is a short parcel of code that performs an action; having done its job the execution of GDL returns precisely to the part of the executive script that it came from.

Subroutines live in the region following the **END** command. A subroutine starts with a number label so that you use that number when you write the **GOSUB** command – e.g. **GOSUB 200**.

The subroutine should behave like an **elemental object** in its own right. If you want to move the element sideways, or swivel it, you apply those move or swivel commands to the cursor, and then **GOSUB** the subroutine – it moves or swivels as required.

Here is a typical subroutine, numbered 250: The number label must always be followed by a colon: and you should add a !comment line after the number, so that you know what the subroutine is actually required to do.

In each subroutine, there must be exactly the right number of **DELs** to ensure that the cursor returns to the same place as it was at the start of the subroutine.

```
250: !Two Spheres subroutine
SPHERE 0.1
ADDx 0.2
    SPHERE 0.05
    DEL 1
RETURN
```

A typical Subroutine

Golden Rules for Subroutines!

- Subroutines are always written after the **END** statement. **END** is essential – you must **never** accidentally run from the executive script into the subroutines.
- Subroutines **always** start with a **Numerical Label**, and finish with the statement '**RETURN**', otherwise, GDL will accidentally run on into the next subroutine.
- **The Subroutine must be entirely self contained** as regards cursor movement – the **3D cursor** must finish up **exactly** where it was at the start of the subroutine. So you **DEL** all the **ADD**, **ROT** and **MUL** commands within the Subroutine. Never use **DEL TOP** for this purpose.
- **Never** **GOSUB** the subroutine you are already in, or GDL will go on trying to do this **for ever** (Cmd-period to stop).

- **Never** use a **GOTO** command to jump out of a subroutine – GDL will get quite lost!
- Think **hierarchically** – Subroutines can **GOSUB** other subroutines: For example, in the human body, upper Arm is an object written with a subroutine with its own rules for rotation and movement; but the forearm will swivel round the elbow. Arm needs another subroutine to draw the object called Hand, and the Hand will need one for each of the objects called Finger.
- The 2D, 3D and Properties scripts can use Subroutines; but they cannot jump to one in another script from the one you are in!
- You cannot use **END** in a **Master Script**.

Subroutines: Convert the chair

FIND the **Simple chair** we made earlier. We shall try to 'structure' it. The first practical step to making your script modular (or structured) is to learn how to use subroutines. Although we have managed so far to do this chair, even the Multichair, without subroutines, we could have saved ourself some typing by using subroutines whenever something was to be repeated – such as the legbraces and arms. Furthermore, we could add more 3D options by using them e.g. if we wanted a choice between Conical, Sabre and Clawfoot legs. Subroutines are a way to clean up the code into tidy modules. Without them, we would get quite buried in IF statements.

Step by step

Here, we try changing the script from the raw (but parametric) script into subroutines. It is an easy progressive change.

First apply label numbers to the groups of code that represent each logical operation e.g. the Legs, the Seat, etc. Each one will become a subroutine. Make sure that in each group, the number of DELs is right in that the cursor goes back to the origin each time. Use 100's to increment the numbers of your groups.

Next, put in the END statement, clearly marked with a comment marker and a row of dashes. This signifies the end of the main part of the 3D Script (the 'executive script').

What follows the END statement is a series of Subroutines which have to be called with a GOSUB command. Copy and paste some of the routines down to below the END statement. Now they are a subroutine: no longer part of the executive script. Put RETURN at the end of each subroutine so that execution returns to where it came from. Now, in the Executive script, write GOSUB followed by the Line number and a copy of the name of the subroutine – do it as I have done here.

If it doesn't work (you get errors) it's due to forgetting the colon after the line numbers, or forgetting END or RETURN. Perhaps you didn't get the number of DELs right. Go over my example until you get it completely right.

Do it for the chair

The examples on this page show how the legs and seat can be converted. For the rest of the chair, do have a look at the examples on the CD or try doing it to your own example.

Whenever you are in doubt, go back to the page on Subroutines, and thoroughly check through the **Golden Rules for Subroutines**.

```
!Timber chair for GDL Cookbook
!Parametric NO Subroutines

PEN gs_cont_pen
RESOL 12

100:!All the legs
MATERIAL framat
CONE sthit, lsec/3,lsec/2, 90,90
  ADDx A
  CONE sthit, lsec/3,lsec/2, 90,90
  ADDy B
  CONE sthit, lsec/3,lsec/2, 90,90
  ADDx -A
  CONE sthit, lsec/3,lsec/2, 90,90
  DEL 3
```

200:!The Seat and upholstery

```
ADDz sthit-frsec
```

```
!Do four prisms for seatframe
```

```
ft=0.03
```

```
f2=ft/2
```

```
PRISM 5,frsec-0.002,
```

```
-f2,0,
```

```
f2,0,
```

```
f2,B,
```

```
-f2,B,
```

```
-f2,0
```

```
ADDx A
```

```
PRISM 5,frsec-0.002,
```

```
-f2,0,
```

```
f2,0,
```

```
f2,B,
```

```
-f2,B,
```

```
-f2,0
```

```
DEL 1
```

```
PRISM 5,frsec-0.002,
```

```
-f2,-f2,
```

```
A-f2,-f2,
```

```
A-f2, f2,
```

```
-f2, f2,
```

```
-f2,-f2
```

```
ADDy B
```

```
PRISM 5,frsec-0.002,
```

```
-f2,-f2,
```

```
A-f2,-f2,
```

```
A-f2, f2,
```

```
-f2, f2,
```

```
-f2,-f2
```

```
DEL 1
```

```
!Seatcushion
```

```
MATERIAL seatmat
```

```
!Way to do it: chamfered with FPRISM_
```

```
FPRISM_ seatmat,frammat,seatmat,seatmat,
```

```
5,frsec+0.010, 30,0.01,
```

```
f2, f2, 15,
```

```
A-f2, f2, 15,
```

```
A-f2,B-f2, 15,
```

```
f2,B-f2, 15,
```

```
f2, f2, -1
```

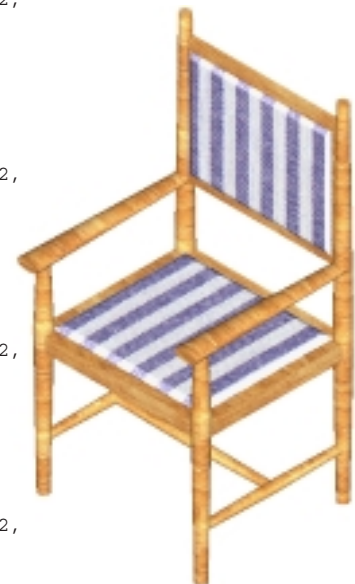
```
DEL 1
```

```
ENDIF
```

```
!The rest of the script, Back and
```

```
!Braces and Arms, not shown here, to save space.
```

Note that conversion of the chair to subroutines will have no effect on the appearance of the chair, or on the parameters available to the user. The purpose of doing this is to make it possible to make the chair object more complex, add more options and reduce the number of IF statements. There is no need to change anything in the Master Script.



!Timber chair for GDL Cookbook
!Parametric WITH Subroutines

PEN gs_cont_pen
RESOL 12

3D Script with Subroutines

GOSUB 100:!All the legs
GOSUB 200:!The Seat and upholstery

!As you progress, add in the remaining routines as Subroutines:

IF bakon THEN GOSUB 300:!Back
GOSUB 400:!Braces
IF arms THEN GOSUB 500:!Arms

END:!

100:!All the legs

MATERIAL framat
CONE sthit, lsec/3,lsec/2, 90,90
ADDx A
CONE sthit, lsec/3,lsec/2, 90,90
ADDy B
CONE sthit, lsec/3,lsec/2, 90,90
ADDx -A
CONE sthit, lsec/3,lsec/2, 90,90
DEL 3

RETURN

200:!The Seat and upholstery

ADDz sthit-frsec
!Do four prisms for seatframe
ft=0.03
f2=ft/2
PRISM 5,frsec-0.002,
-f2,0,
f2,0,
f2,B,
-f2,B,
-f2,0
ADDx A
PRISM 5,frsec-0.002,
-f2,0,
f2,0,
f2,B,
-f2,B,
-f2,0
DEL 1

PRISM 5,frsec-0.002,
-f2,-f2,
A-f2,-f2,
A-f2, f2,
-f2, f2,
-f2,-f2

ADDy B
PRISM 5,frsec-0.002,
-f2,-f2,
A-f2,-f2,
A-f2, f2,
-f2, f2,
-f2,-f2
DEL 1

!Seatcushion

MATERIAL seatmat
!Way to do it chamfered with FPRISM_
FPRISM_ seatmat,frammat,seatmat,seatmat,
5,frsec+0.010, 30,0.01,
f2, f2, 15,
A-f2, f2, 15,
A-f2,B-f2, 15,
f2,B-f2, 15,
f2, f2, -1
DEL 1
RETURN

These PRISM routines are very repetitive and could be converted to subroutines. We could do the same to the CONEs for the leg braces

300:!Back Legs, panel and upholstery

MATERIAL framat
ADD 0,B,sthit
CONE bhit,lsec/2,lsec/3, 90,90
ADDx A
CONE bhit,lsec/2,lsec/3, 90,90
DEL 2
ADD 0,B-0.01,sthit+bhit*0.3
BLOCK A,lsec/2,bhit*0.6 !Back panel
MATERIAL seatmat
!Do it chamfered
ADD lsec/2,0,lsec/2
ROTx 90
FPRISM_ seatmat,frammat,seatmat,seatmat,
5,0.010, 30,0.008,
0, 0, 15,
A-lsec, 0, 15,
A-lsec,bhit*0.6-lsec, 15,
0,bhit*0.6-lsec, 15,
0, 0, -1
DEL 2
DEL 1
RETURN

400:!Braces

IF brace THEN
MATERIAL framat
ADDz sthit/3
ROTx -90
CONE B/2,lsec/4,lsec/3,90,90
ADDz B/2
CONE B/2,lsec/3,lsec/4,90,90
DEL 3
ADD A,0,sthit/3
ROTx -90
CONE B/2,lsec/4,lsec/3,90,90
ADDz B/2
CONE B/2,lsec/3,lsec/4,90,90
DEL 3
IF bracetyp=bv1 THEN !--H Brace
ADD 0,B/2,sthit/3
ROTz -90
ROTx -90
CONE A/2,lsec/4,lsec/3,90,90
ADDz A/2
CONE A/2,lsec/3,lsec/4,90,90
DEL 4
ENDIF
IF bracetyp=bv2 THEN !--O Brace
ADD 0,0,sthit/3
ROTz -90
ROTx -90
CONE A/2,lsec/4,lsec/3,90,90
ADDz A/2
CONE A/2,lsec/3,lsec/4,90,90
DEL 4
ADD 0,B,sthit/3
ROTz -90
ROTx -90
CONE A/2,lsec/4,lsec/3,90,90
ADDz A/2
CONE A/2,lsec/3,lsec/4,90,90
DEL 4
ENDIF
ENDIF
RETURN

500:!Arms

MATERIAL framat
ADD 0,0,sthit
CONE 0.2,lsec/2,lsec/3, 90,90
ADD 0,-lsec, 0.2
ROTx -90
MULy 0.5
CONE B+lsec,lsec*0.7,lsec*0.45, 90,90
DEL 4
ADD A,0,sthit
CONE 0.2,lsec/2,lsec/3, 90,90
ADD 0,-lsec, 0.2
ROTx -90
MULy 0.5
CONE B+lsec,lsec*0.7,lsec*0.45, 90,90
DEL 4
RETURN

This is a chance to catch up with the previous pages: here is a round up of the complete 3D Script, formed into subroutines.

Once you have the script organised into subroutines, there are ways to tighten up the code – eliminate duplication by making smaller subroutines which branch out from the main ones.

Sometimes the code for an object can be reduced to a fraction of its former size.

Subroutines: Tighten up the Code and improve the Texture

Code Compression

WE can look at ways to convert the minor parts of the chair to smaller subroutines such as legbraces and arms – this tightens up the code, and makes future maintenance and modification easier. With more experience you learn to write in a more economical way from the start, recognising immediately when something is coming up that it will be repeated so it should go into a subroutine straight away.

We labelled the original subroutines in the hundreds – 100, 200 etc. Thus any minor subroutines to do with the legs could be numbered 110, 120 etc. For the seating, we could use numbers like 210, 220. For the braces we could use 410, 420 etc. It will provide us with some logical grouping.

We can do an example of fragmenting the subroutines on this page, and the remainder can be seen if you view the object from the CD.

This process does not do anything to improve the 3D, but it shortens the code and, most importantly, improves the maintenance of the object. For example, the seat frame with prisms required repetition – same for the legbrace. If we had curly legs for the chair that might require a lot of code and a single subroutine could suffice for each leg. Then if you edit one of the legs, the rest get updated too.

On an earlier page, you saw the script for the four prisms of the seatframe, as we first created them. Here is an improved version for the seating and the braces.

Texture matters with wood objects!

You can make a beautiful wood-based object in 3D, but if the wood texture is going all in the wrong direction people will not take your GDL skills seriously. At the time of writing, all the Graphisoft Doors and Windows have incorrect woodgrain on them, and this is something I cannot understand, as texture is so easy to get right.

Textures in ArchiCAD itself align themselves dynamically by guessing at the aspect ratio of the wall or slab they are attached to. If you don't like the result, you can use 'Align texture' in the Edit menu to correct it.

Textures in GDL require you to write some code to drive the texture in the right direction, because GDL will not guess. For the beginner in GDL, it's difficult to get textures right without knowing some arcane secrets of GDL. You are about to find out those secrets.

The easier way for the GDL writer (apart from not bothering) is to require the user of your object to make all the textures in ArchiCAD including the correct orientation. You make a different parameter for materi-

```
200:!!The Seat and upholstery
ADDz sthit-frsec
!Do four prisms for seatframe
ft=0.03
f2=ft/2
GOSUB 210:!!Front to Back frame
ADDx A
GOSUB 210:!!Front to Back frame
DEL 1
GOSUB 220:!!Left to Right frame
ADDy B
GOSUB 220:!!Left to Right frame
DEL 1
!Seatcushion
!Chamfered with FPRISM_
FPRISM_ seatmat,frammat,
      seatmat,seatmat,
      5,frsec+0.010, 30,0.01,
      f2, f2, 15,
      A-f2, f2, 15,
      A-f2,B-f2, 15,
      f2,B-f2, 15,
      f2, f2, -1
GOSUB 999:!!Paint Texture
DEL 1
RETURN
```

```
210:!!Front to Back frame
PRISM 5,frsec-0.002,
-f2,0,
f2,0,
f2,B,
-f2,B,
-f2,0
ROTz 90
GOSUB 999:!!Paint Texture
DEL 1
RETURN
```

```
220:!!Left to Right frame
PRISM 5,frsec-0.002,
-f2,-f2,
A-f2,-f2,
A-f2, f2,
-f2, f2,
-f2,-f2
GOSUB 999:!!Paint Texture
RETURN
```

This is a restructuring of subroutine 200.

The 'ft' and 'f2' statements could move to the Master Script.

999 is a Texture Subroutine

In this script the two frame pieces are placed in subroutines 210: and 220: Subroutine 200 now calls those subroutines.

Just to improve the appearance, we can add a texture direction to each subroutine. This is itself a subroutine, 999:, and this contains code to paint texture horizontally. By rotating it to vertical in subroutine 998, we get vertical texture.

als horizontal and vertical. You then have the user being forced to choose textures for legs and arms separately – this is a real nuisance for the user, and very very lazy on the part of the GDL writer.

If you want textures to go upwards or at angles, you can write routines in the GDL that make the texture go in the direction you want. For example, the textures in the legs must go vertically, and in the front to back frame elements must also go from front to back.

Let the user choose **one** single material for the frame. Assume that they choose one that is horizontal in nature, and then let the GDL code organise the woodgrain. This is best!

```

400:!Braces
IF brace THEN
!Front to Back
MATERIAL framat
ADDz sthit/3
ROTz -90
GOSUB 410:!Brace Front to back
ADDx A
GOSUB 410:!Brace Front to back
DEL 3

IF bracetyp=bv1 THEN    !--H Brace
ADD 0,B/2,sthit/3
ROTz -90
ROTz -90
GOSUB 420:!Brace left to right
DEL 3
ENDIF

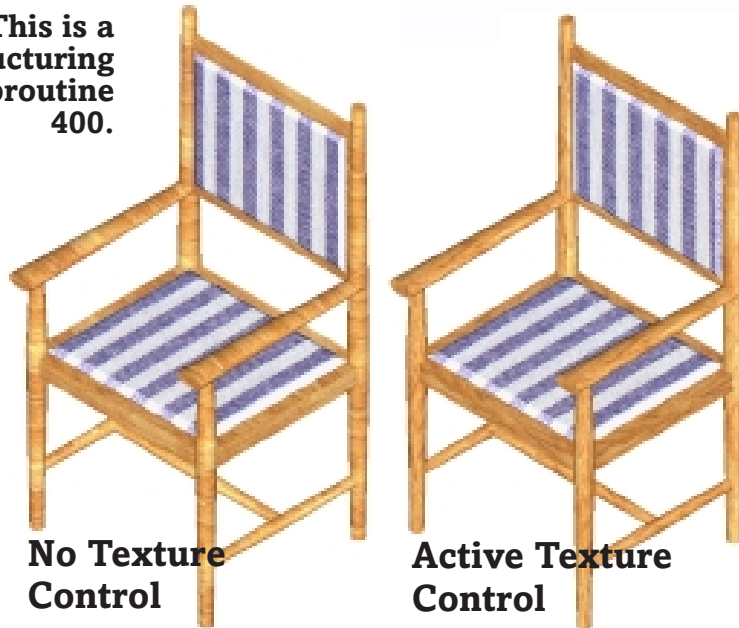
IF bracetyp=bv2 THEN    !--O Brace
ADD 0,0,sthit/3
ROTz -90
ROTz -90
GOSUB 420:!Brace left to right
DEL 4
ADD 0,B,sthit/3
ROTz -90
ROTz -90
GOSUB 420:!Brace left to right
DEL 3
ENDIF
ENDIF
RETURN

410:!Brace Front to back
CONE B/2,lsec/4,lsec/3,90,90
ADDz B/2
CONE B/2,lsec/3,lsec/4,90,90
DEL 1
GOSUB 998:!Paint upright
RETURN

420:!Brace left to right
CONE A/2,lsec/4,lsec/3,90,90
ADDz A/2
CONE A/2,lsec/3,lsec/4,90,90
DEL 1
GOSUB 998:!Paint upright
RETURN

```

**This is a
restructuring
of subroutine
400.**



**No Texture
Control**

**Active Texture
Control**

***What's the secret?** Well if we define 4 vertices which accurately describe the 3 pointed XYZ cursor, it can be told to point the texture along the **X-Axis**.*

```

998:!Paint upright
ROTz 90
GOSUB 999:!Paint Texture
DEL 1
RETURN

999:!Paint Texture
!Horizontally
BASE
VERT 0,0,0
VERT 0.1,0,0
VERT 0,0.1,0
VERT 0,0,0.1
COOR 256+2, -1,-2,-3,-4
BODY -1
RETURN

```

*The **BASE** starts off a new Texture routine. The **VERT**s describe the Centre, the X-Axis, the Y-Axis and the Z-Axis in that order. The **COOR** coordinates the 4 together, with a code of 256+2 which means "wrap cubically according to the local coordinates". The **BODY** command actually completes the job by painting the texture. Although it's based on the X-Axis, the cubic mode will look after the other faces of the cuboid.*

Textures such as brickwork and woodgrain are generally horizontal in ArchiCAD, but recently Graphisoft brought in some more wood texture bitmaps that are vertical, such as Mahogany. This was a mistake, and they promised me to try to be more consistent in future.

This is something you won't find any help from the GDL manual on. This particular trick wasn't even in GDL Cookbook 3. You found it here first!

There are two ways to organise woodgrain in a model like this. Subroutine 999 is a technical piece of GDL code that will be explained in more detail later, but let's just say here that the routine guarantees that a horizontal texture will be painted horizontally. Simple as that!

If you rotate this subroutine 999 around Y or Z, you can easily redirect the texture. It's like rotating a toothpaste tube, and then letting the texture squirt out. The X-Axis is used in GDL for many things, ELBOWs and LIGHTs and REVOLVEs for example. By pointing the X-Axis, you can squirt texture out like toothpaste along the X-Axis of the theoretical tube.

You can rotate the texture to any angle, but in the chair we have two main directions, Horizontal and Vertical. Subroutine 998 is simply a device to point the texture vertically. The texture subroutine must come AFTER the object you make. Everything that was made, back to the previous use of the subroutine, will be painted. So go to every subroutine in the script and insert either a GOSUB 998 or GOSUB 999 and you will get the textures right.

How does it work?

There is an extremely brief explanation of the magic subroutine 999 in the caption above. Why do we need to know the directions of the Y and Z axes? Because if we play games with these, we can skew the textures. And beside Cubic wrapping, on another occasion, we can try Spherical and Cylindrical wrapping modes.

Working with PRISMs: Curve the Chairback

WHILE this section is about Prisms, we can look at the bendy BPRISM_ and apply it to the back of the chair.

We would like to curve the back panel to make it more comfortable and authentic. The width of the chair is 'A'. Make a parameter for the bulge of the chairback 'bbulg' and use the formula based on A to calculate how much curvature to apply to the chairback. See the formula illustrated.

Make a new parameter 'bbulg' (dimension). First the Master Script does the calculations.

```
!-----Add to Master Script-----
!Back bulge Circle calculation formula
IF bbulg<=0.01 THEN bbulg=0.01
IF bbulg>=0.08 THEN bbulg=0.08
bangl=(90-ATN((A/2)/bbulg))*2
brad=(A/2)/SIN(bangl)
bcirc=2*PI*rad*bangl*360
PARAMETERS bbulg=bbulg
```

We have two idiotproofing lines. PARAMETERS reports the corrected dimensions back to the user.

We must calculate the sweep angle – the half angle 'bangl' between the edge and the centre. From that we work out 'brad' the radius. As BPRISM is like a flat surface bent into a curve, we work out the length of the chord. As BPRISM has to be organised around the centre of the chair for it to bend symmetrically, we need only the half length of the chord. So look at the 3D script as it was before, and make the modification.

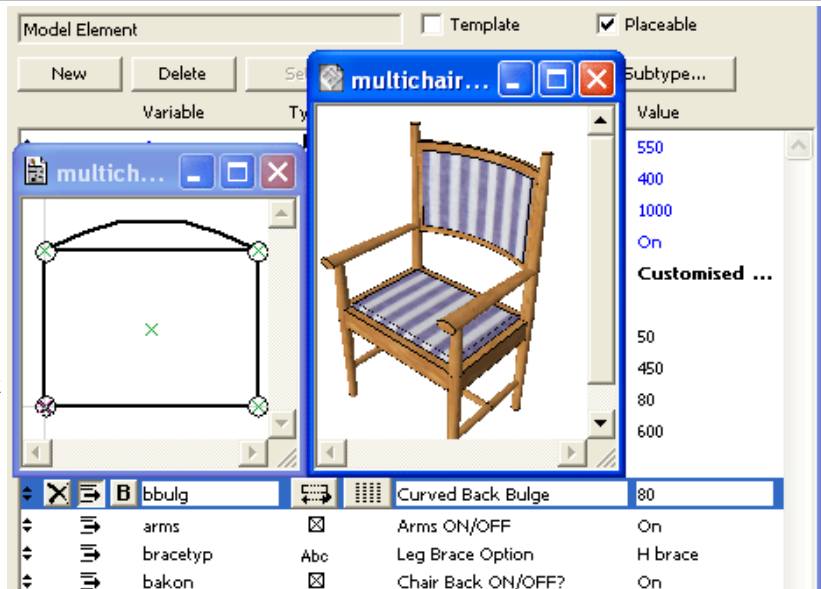
```
!Modify 3D Script
!Back Legs, panel and upholstery
IF bakon THEN
MATERIAL framat
ADD 0,B,sthit
CONE bhit,lsec/2,lsec/3, 90,90
ADDx A
CONE bhit,lsec/2,lsec/3, 90,90
DEL 2

!Back panel
ADD A/2,B-lsec/3+bbulg,sthit+bhit*0.9
ROTx -90
BPRISM_ framat,framat,framat,
5,lsec/3,brad,
-bcirc,0,15,
bcirc,0,15,
bcirc,bhit*0.6,15,
-bcirc,bhit*0.6,15,
-bcirc,0,-1

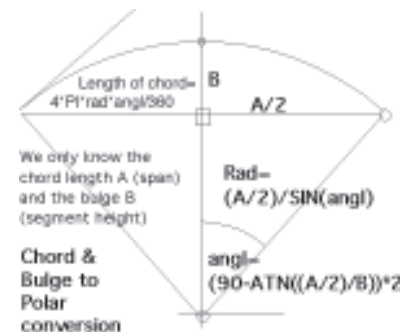
!Cushion, not chamfered
ADDz -lsec/6
BPRISM_ seatmat,framat,seatmat,
5,lsec/6,brad,
-bcirc +ft,ft,15,
bcirc -ft,ft,15,
bcirc -ft,bhit*0.6-ft,15,
-bcirc +ft,bhit*0.6-ft,15,
-bcirc +ft,ft,-1

DEL 3
!BLOCK A,lsec/2,bhit*0.6 !original Back panel
ENDIF
```

We use ADD to get to the centre of the chair. Because BPRISM_ curves downwards, we have to get behind the chair and use ROTx so that the Z axis is facing backwards (away from the chair). Play with the script of the chair changing the ROTx from -90 to 90.



The chairback is a chord. Work out from the bulge what the sweep angle is, and work out the radius. We also need the circumferential distance travelled by the back.



In the 3D Script, we built the back legs, as before, but for the back panel, we have to do some 3D manoeuvres. The height of the prism will be from the topmost part – 'sthit+bhit*0.9'. Try changing the values to see where the back moves to. We made the half-width of the prism the half-circumference, so that whatever the bulge value, it will always fit perfectly.

The back cushion – by adjusting the Z height, we can shift a small distance and issue another BPRISM_ for the upholstery, using soft fabric. DEL 3 takes us right back to the origin.

Final touch of quality – Update the 2D Script

From the illustration you can see that the 2D Script has been updated. Let's do this. We could 'cave in' and do a PROJECT2. Since we took the trouble to calculate the circle geometry of the chairback, let's put that to use. Comment out the LINE2 commands and try an ARC2 for the back and a single line for the frame.

```
!Put these lines in for the curved back
!LINE2 s2,B-s4, A-s2,B-s4 !back
!LINE2 s2,B+s4, A-s2,B+s4 !back
ARC2 A/2,B-brad+bbulg,brad,90-bangl+2,90+bangl-2
!HOTARC2 A/2,B-brad+bbulg,brad, !for AC9 users
! 90-bangl+2,90+bangl-2 !for AC9 users
LINE2 s2,B, A-s2,B !back. HOTLINE2 for AC9 users.
```

We do not have to use ADD2 commands here. We position the arc with an XY coordinate, and the position is at the centre of the back, as with the prism. The sweep angle of the arc can be 'bangl' either way. By deducting 2° either way, it looks better in relation to the legs.

Curvy shapes: Polygon Control



MOST of the workload of the processor during rendering seems to be working out the values of light and shade on each surface (polygon) in the model. If it is trying to work out Shadows, it's also working out the outline of the object and what the shadow will do when falling on each polygon. **Fewer polygons means faster rendering.** An excess of polygons will lead to excessive rendering times, or it may cripple the model altogether – not enough RAM to finish! Isn't Open GL a major blessing! Sometimes, the object you are doing may not be visible (such as a nut or bolt) so why give it 36 or more polygons, especially when there could be thousands of nuts and bolts in the model? So control the curvature!

RESOL: is the easiest way of controlling the resolution of curvature of curved surfaces such as Cones and Cylinders. RESOL tells GDL how many polygonal surfaces there are to be in a single 360° rotation.

The default resolution is 36, so a single Sphere will have 648 surfaces right from the start. Get it down to the lowest practical quantity – or even lower. If you are Photorendering or using Open GL the resolution can be set even lower because you have a smoothing function that makes even a hexagonal cylinder look round. It is a good tip to set resolution to an odd number – 5, 7 or 9 look more cylindrical than the patently hexagonal look of RESOL 6. RESOL cannot be less than 3.

RESOL can be controlled by a variable, for example RESOL 6+10/dd will make the resolution higher as the distance to camera 'dd' reduces. If the resolution is stated as a real number – eg 6.35, it is always rounded down.

RADIUS: is more powerful than RESOL. It gives you 3 RESOL commands in a single line. Some objects such as a curved handrail have two curves to resolve, the curve of the handrail and the curve of the tubing. You state RADIUS in the form:

RADIUS rmin, rmax

Any curve equal to or less than 'rmin' will render with 6 faces. Any curve equal to or more than 'rmax' will render with 36 faces. Any curve between the two will render with a variable number of faces, on a sliding scale between 6 and 36.

RADIUS cannot be higher than 36 faces, so for very large curves combined with small ones, use TOLER.

TOLER: is the best way of setting Resolution of curvature, and the most powerful. TOLER is like your 'tolerable error', the distance from the actual arc of the curve, and the distance you are willing to allow the chord of the curve to exist away from the curve. GDL will recalculate the resolution to match the object. For example, a TOLER of 1 mm (1/25") is ideal for tubular furniture and does a good job on curves. It's even better on very large structures where a maximum of 36 needs to be exceeded.

TOLER could be set using variables or IF statements, just like RESOL or RADIUS. TOLER takes, perhaps more processor time than RADIUS and RESOL but gives the best result visually.

Note of Warning – RESOL, TOLER and RADIUS cancel each other out if used, so decide which one you want to use for a model, or for a component of a model.

Smart Programming – For me the 4th weapon in the war against polygons is to be a smart programmer. You can have a parameter for the level of detail, e.g. 0=2D only, 1=Simple 3D, 2=High 3D. Most Graphisoft objects have this (parameter is "treed"). If the user selects simple, you leave out all bolts and joints, set TOLER to 0.01 or 3/8" and perhaps use flat POLYs instead of PRISMs (to avoid edge polygons). If the user selects option 2, they are aware that the drawing time will be longer.

Open GL and faster processors have made it possible to have more complex objects with more polygons, but you should still apply a rigorous approach to the Level of Detail (**LOD**)

Elbow: the curved Cylinder

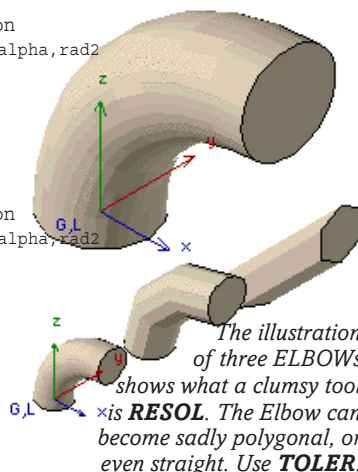
ELBOW builds a curved tubular bend object. It always grows up the Z-Axis, curving towards the X direction. The alpha angle can be from 0 to 360.

The number of polygons can be a problem – the curvature can be controlled with the RESOL or the TOLER statements. Curvature is an important issue – if you have a low resol, the curvature of the tube looks unfortunately angular. However, if you set a high resol, you have too many surfaces on the elbow, and it will take too long to render. TOLER is best for controlling curvature on small and large curves simultaneously. (see later pages for more examples).

!ELBOW Demonstration
!Syntax:- ELBOW rad1,alpha,rad2
ELBOW 0.12,90,0.03

'alpha' is the sweep angle of the Elbow.

!ELBOW Demonstration
!Syntax:- ELBOW rad1,alpha,rad2
RESOL 36
ELBOW 0.12,90,0.03
RESOL 12
ADDY 0.1
ELBOW 0.16,90,0.04
RESOL 6
ADDY 0.1
ELBOW 0.20,90,0.05
DEL 2



Curvy shapes: Tubular Handrail

EARLY on, it's good to take a break from the tyranny of the rectangle, and try something a bit more elegant. Here, we look at:

- Handling cylindrical entities – **CYLIND** and **ELBOW**
- Using **TOLER** to control curve smoothness
- Using **HOTSPOTS** in 2D

Parameters

Ask the user to enter small dimensions for tubing in the form of Diameter, not in Radius. People always think of handrails and other round things in 'diameter'. Your Master Script can convert Diameter to Radius. As the handrail dimensions can be altered, A and B are not used in this script or in the parameters box – we specify the length of each tube.

Angle – You can think of the angle in terms of the angle formed by two lines. But a tube bender, or GDL prefers to think of the 'sweep angle' formed by the tube, from the straight.

TOLER – We want the main curve to be as smooth as possible, but don't want the surface of the tubing to have 36 faces as it will take too long to render. The TOLER command allows you to control the surfaces of small and large curves, and to relate it to the actual tube radius. 2mm is good for most purposes and will work well even for very small or large tubes.

PEN – we use the parameter name 'gs_cont_pen' as this is one of the GS recommended names.

If you want to 'hand' the finished handrail, i.e. Mirror it, then you can do that in the object settings box, later.

3D Script

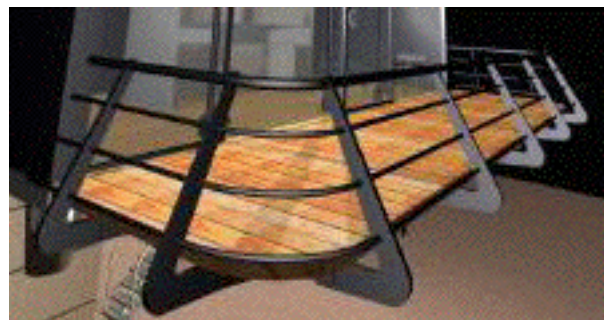
Set the Material and Pen colours.

Starting from the end of the first tube, use a ROTx -90 to lie the handrail down horizontally (or it will grow upwards). Do the first Rail by pushing out a cylinder horizontally, then move to its end by adding along the Z axis – which is now horizontal. You can push the ELBOW round the corner, as it grows along the Z axis, which is already in the right direction. Then go round the bend – go to the centre of the curve, rotate by the angle, and move to the start of the second tube. This trick means that the cursor will be pointing in the correct direction for you to push out the final cylinder.

This script is short, but you should still use the DEL command to bring your 3D cursor back to the origin.

Using the principles of the script for this simple library object, you can assemble handrail and tubing structures of great complexity.

Notice how the cursor movement commands are indented. When you have a long run of them, it makes it easier to count them up.



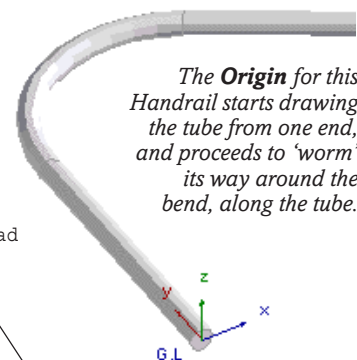
The handrails can be grouped together to form useful structures which would be difficult to do in any other way.

Variable	Type	Name	Value
A			1000
B			1000
ZZYZX		Z Dimension	1000
AC_show2DH...		Show 2D Hotspots in 3D	On
diam		Tubing Diameter	50
len1		Length One	600
len2		Length Two	500
radc		Radius of Bend	300
angb		Angle Bent (from Straight)	120.00
HandRailTu...		Materials & Pens	
tub_mat		Material of tubing	11
gs_cont_pen		Pen Colour	94

```
!Master Script
trad=diam*0.5 !TubeRadius
```

```
!3D Script
MATERIAL tub_mat
PEN gs_cont_pen
TOLER 0.002
```

```
!Build the tube
ROTx -90
CYLIND len1,trad
ADDz len1
ELBOW radc,angb,trad
ADDx radc
ROTy angb
ADDx -radc
CYLIND len2,trad
DEL 5
```



The **Origin** for this Handrail starts drawing the tube from one end, and proceeds to 'worm' its way around the bend, along the tube.

This is the trick for getting round the corner

Making it stretchy?

If you used A and B to make it stretchy, you would have to work backwards from the curve radius to decide what the straight tube lengths would be. For angles other than 90degrees, it would make no sense. This object should be developed further with custom Graphical Hotspots, because the restrictions of an A,B,zzzyx cuboid are not helpful. See the Bendy bar exercise.

2D Script

As the user can change the handrail details, you cannot use a standard symbol. You need to write a 2D Script. For quick-to-draw objects, issue the PROJECT2 command as here.

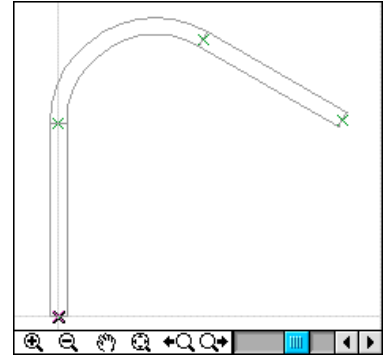
Hotspots: This handrail is a slippery object to pick up because the 'Bounding box' hotspots are going to be positioned well away from the actual rail. You need to plant HOTSPOTS in the 2D Script. Turn off Bounding boxes. The 2D cursor can be moved about in a similar way to the 3D Script, using the ADD2 and ROT2 commands, until you find the end of the tube. This object is not stretchy.

```
!Handrail 2D Script
PROJECT2 3,270,2
HOTSPOT2 0,0
```

```
!First Tube
ADD2 0,len1
HOTSPOT2 0,0

!Round the bend
ADD2 radc,0
ROT2 -angb
ADD2 -radc,0
HOTSPOT2 0,0

!Second tube
ADD2 0,len2
HOTSPOT2 0,0
DEL 5
```



Syntax: TEXT depth,0,string

3D Text: Depth is in metres, zero is always zero, the string is in quote marks unless it's a string parameter.

The height is defined earlier with **DEFINE STYLE** and must be in millimetres. So if you want the lettering to be 0.6 metres high (2'-0") you write in 600 for the height – or multiply a length parameter by 1000.

This text can be for laying flat or standing upright, so you apply an optional ROTx 90 to make it stand up. The problem is that 'B' and 'zzyzx' perform differently according to the object's attitude, so in this case, you need some tricks in the Master Script to define the style accordingly. Because B and zzyzx are in metres when it gets to GDL, you have to convert them to millimetres, multiply by 1000. As we only want to make only one DEFINE STYLE statement, we use a temporary variable like 'hit' to do the work.

For the width of the overall text object, you cannot use 'A' directly because the Font determines its own length. However, the function STW(string) returns a value of the length of the string in millimetres, so divide it by 1000 to get the length in metres. You could use MULx to 'squash or stretch' it by the ratio of A divided by actual length.

When you use ROTx 90 to make it stand up, you need a little offset to make it fit the 2D hotspots.

The final touch is that you have to write a 2D Script with stretchy Hotspots to make it work correctly.

Apply a RESOL or TOLER command before writing 3D text, or your model may be crippled with too many polygons. RESOL 9 is the lowest practical (for legible text), but TOLER is better for all sizes.

3D Text

```
!Master Script
```

```
VALUES 'font' 'Times','Arial','Courier'
```

You can make a small dropdown menu of the fonts on your system, and one for the attitude of the object.

Use 'hit' for the height, depending on the attitude of the object.

For the Define Style, 7 is the best position for the text.

```
!3D Text : Example of 3D Text
```

```
PEN gs_cont_pen
```

```
MATERIAL text_mat
```

```
TOLER 0.002
```

```
SET STYLE '3dtextyl'
```

```
!Make it Stretchy
```

```
MULx A/(STW(message)/1000)
```

```
IF atti=atv1 THEN
```

```
TEXT zzyzx,0,message
```

```
ENDIF
```

```
IF atti=atv2 THEN
```

```
ADDy B
```

```
ROTx 90
```

```
TEXT B,0,message
```

```
DEL 2
```

```
ENDIF
```

```
DEL 1
```

```
!2D Script for 3D text
```

```
HOTSPOT2 0,0
```

```
HOTSPOT2 0,B
```

```
HOTSPOT2 A,0
```

```
HOTSPOT2 A,B
```

```
HOTSPOT2 A/2,B/2
```

```
PROJECT2 3,270,2
```

Set up Pen and Material.

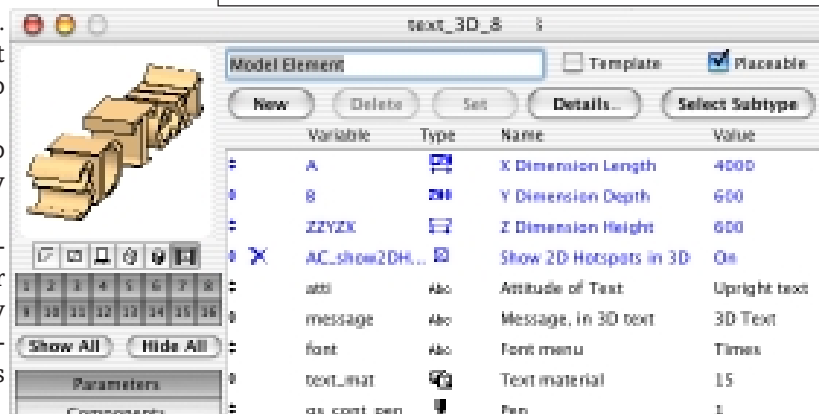
Set the Style for the text.

It is vital to apply a RESOL or TOLER command before writing 3D text, or your model may be crippled with too many polygons. RESOL 9 is the lowest practical (for legible text), but TOLER is better.

You can write the TEXT statement on its own, but with this MUL command, you can make it stretchy. Divide A by the actual length of the string in Metres to get the correct size.

Plant Hotspots at each corner and one in the centre.

MODEL WIRE is a neat trick to make the lettering 3D, but transparent.



Tubular Steel Chair

HERE is a different chair – but rendered in round tubes – like the original Breuer chair.

This chair is a useful starting exercise in

- Handling straight and curved tubular sections
- Keeping track of many 3D cursor moves
- Structuring the program in the form of subroutines
- Using Symmetry and Mirroring
- Soft look upholstery
- Some Trigonometry

Parameters

We will make the origin in the centre front of the chair, draw one leg, all in subroutine 100. Then we'll repeat that on the other side (mirroring the first). Then we'll finish off with the seat and back, all in subroutines 200 and 300. The steel of this tubular chair meets at the bottom. It could also do at the top, but I am trying to keep this simple! For this chair we will use 'zzyzx' to denote the height of the seat.

```
!Master Script
trad = fsec/2 !Tube radius
crad = trad*4 !Curv radius
sinr = SIN(recang)
```

Master Script

Use this for parameters etc. that are needed by both 2D and 3D Scripts.

'Crad' and 'trad' are examples of 'internal parameters'. This means a parameter that is not changeable by the user in the normal dialog box. They have been prescribed by the chair designer / manufacturer. These go into the Master Script. The author of the script can tweak the value of 'crad'.

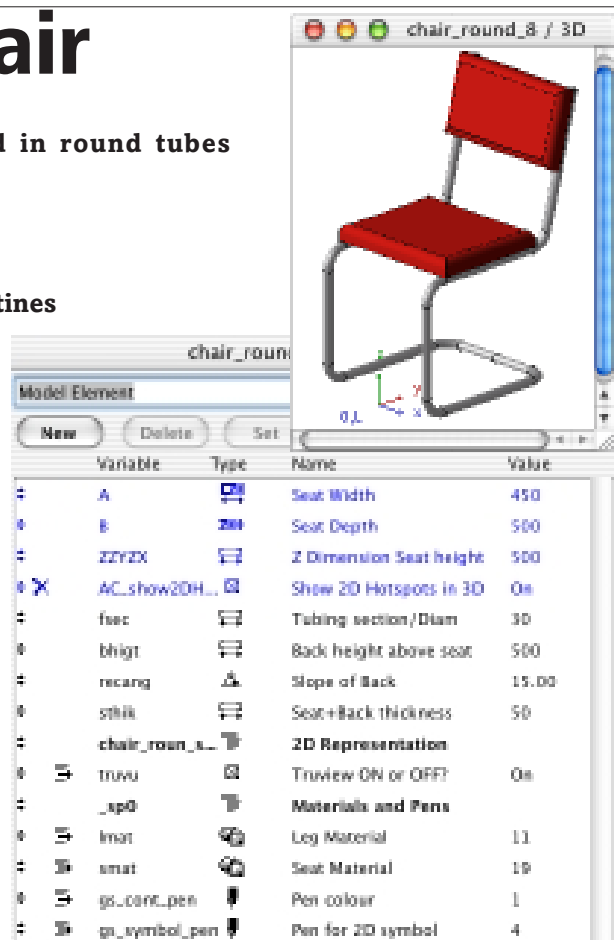
3D Script

We will try the RADIUS command to control the two curvatures of **tube bend** and **tube surface**. Imagine a lecture room or café filled with such chairs if we didn't control the polygons – it would render very slowly!

Axial Symmetry: Because the round tube chair is symmetrical, there is great advantage of starting from the Centre Line, doing only one leg, and then mirroring that whole leg. The MUL command multiplies everything in the direction of the axis. So MULx -1 makes the leg draw itself.



If you are a bit adventurous, you could try making a larger curve radius, leaning the front tubes of the frame, curving the seat and back, and closing the frame at the top of the back.



```
!Tubular steel chair
!3D Script
```

```
PEN gs_cont_pen
RADIUS trad,trad*4
GOSUB 100: !Leg
MULx -1
GOSUB 100: !Leg
DEL 1
GOSUB 200: !Seat
GOSUB 300: !Back
```

```
END!-----
```

The executive script is very short. If your origin is on the centreline, a MULx -1 is enough to mirror the leg symmetrically.

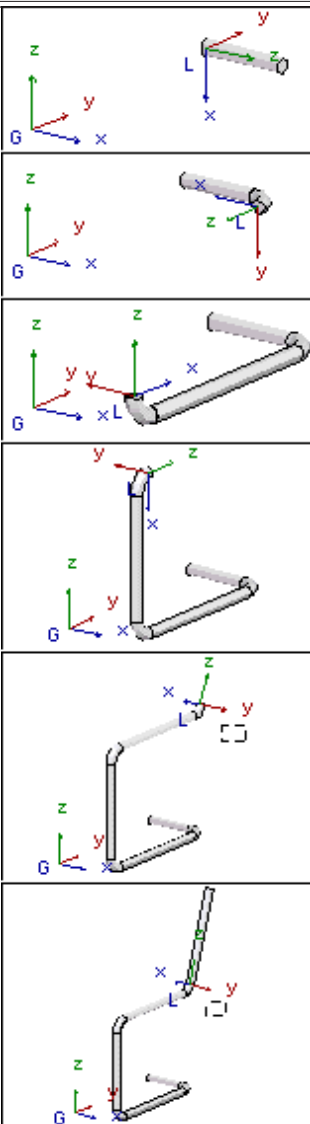
The Leg Tubes

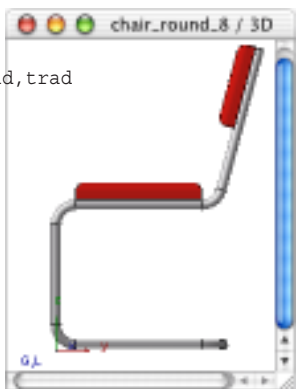
100: Starting from the centre back of the chair, you 'worm' your way round the tubular frame, issuing Cylinder and Elbow commands as you go. Frequently, you have to issue a ROTz to get the next Elbow to grow in the right direction – it always wants to grow in the direction of the X-axis. Use the trick we used on the Handrail to turn the corners (even though it is simply 90° and one could do an easy ADD crad,0,crad jump).

The lengths of the horizontal and vertical leg cylinders are tricky, you have to deduct curve radii and tube radii to get them right, so that the finished height of the seat is as required.

Upholstering the chair

200: and 300: The Seat is easy to place but the Back is more difficult. Maneuvring up to the right place for the Back of the chair proves to be a bit tough, you just have to think out the additions and deductions for curve and tube radii.

<pre> 100:!Cylindrical Leg MATERIAL lmat ADD 0,B,trad ROTy 90 CYLIND A/2-crad-trad,trad ADDz A/2-crad-trad ROTz -90 ELBOW crad,90,trad ADD crad,0,crad ROTy 90 CYLIND B-crad*2,trad ADDz B-crad*2 ROTz -90 ELBOW crad,90,trad ADD crad ROTy 90 ADDx -crad !(Work out cylinder height) clen=zzyzx-sthik-crad*2-trad*2 CYLIND clen,trad ADDz clen ELBOW crad,90,trad ADDx crad ROTy 90 ADDx -crad CYLIND B-crad*2,trad ADDz B-crad*2 ROTz 180 ELBOW crad,90-recang,trad ADDx crad ROTy 90-recang ADDx -crad CYLIND bhigt-crad,trad DEL 20 RETURN </pre>	<p><i>When doing a long tunnelling job as this Worm is having to do you just have to keep your head and your sense of direction.</i></p> <p><i>'clen' is used here just to shorten the line length. It's a short term parameter.</i></p> <p><i>Successive indenting and grouping of blocks of code helps you keep track of what you are doing.</i></p> <p><i>At the end, count up all the moves and DEL them.</i></p>		<pre> 200:!Seat for Cyl chair MATERIAL smat ADDz zzyzx-sthik ADDy crad HPRISM_ smat,smat,smat,smat, 5,sthik,0,sthik/2,1, -A/2,0,15, A/2,0,15, A/2,B-crad*2,15, -A/2,B-crad*2,15, -A/2,0,-1 DEL 2 RETURN For the seat we can give a soft edged impression by using HPRISM_ and making the lines on the hill edge invisible (code 1). For the back its a little more tricky, allowing for a user variable reclining angle for the back. 300:!Back for Cyl chair MATERIAL smat ADDy B-crad ADDz zzyzx-sthik+crad-trad ROTx -recang ADD 0,sthik,bhigt/2-crad ROTx 90 HPRISM_ smat,smat,smat,smat, 5,sthik,0,sthik/2,1, -A/2,0,15, A/2,0,15, A/2,bhigt/2,15, -A/2,bhigt/2,15, -A/2,0,-1 DEL 5 RETURN </pre>
---	---	---	--



2D Script

As the user can change the chair style and size, you need to write a short 2D Script. For quick-to-draw objects, just issue the PROJECT2 command as here. You need to organise a 'nest' of Hotspots, one at each corner and one in the middle to make sure the chair will stretch correctly. This is a classic stretchy hotspots script for any object with the origin at the front centre.

For objects with many uses, it pays to write a 2D Script which will draw a cleaner symbol, and draw it faster. The user can opt for a True view or a scripted view with a Boolean tick box.

For the scripted one, we can use the same line labels as the 3D, but do not need to use subroutines. The legs and seat are easy enough. For the back, it requires a little trigonometry trick if the reclining angle is to be parametric.

When you get further, I advocate the use of POLY2 instead of RECT2, so you can define fill pattern and make the chair opaque in plan.

we want this dimension: it is $bhigt \cdot \sin(\text{recang})$



!Tubular steel chair
!2D Script

```

PEN gs_symbol_pen
HOTSPOT2 -A/2,0 !Stretchy
HOTSPOT2 A/2,0 !hotspots
HOTSPOT2 -A/2,B
HOTSPOT2 A/2,B
HOTSPOT2 0,B/2 !Pickup
!PROJECT2 3,270,2 !Test

```

```

IF truvu THEN
  PROJECT2 3,270,2
ELSE

```

100:!Legs****

```

RECT2 -A/2,0,-A/2+fsec,
      B+bhigt*sinr
MUL2 -1,1
RECT2 -A/2,0,-A/2+fsec,
      B+bhigt*sinr
DEL 1

```

200:!Seat****

```

RECT2 -A/2,crad,
      A/2,B-crad

```

300:!Back****

```

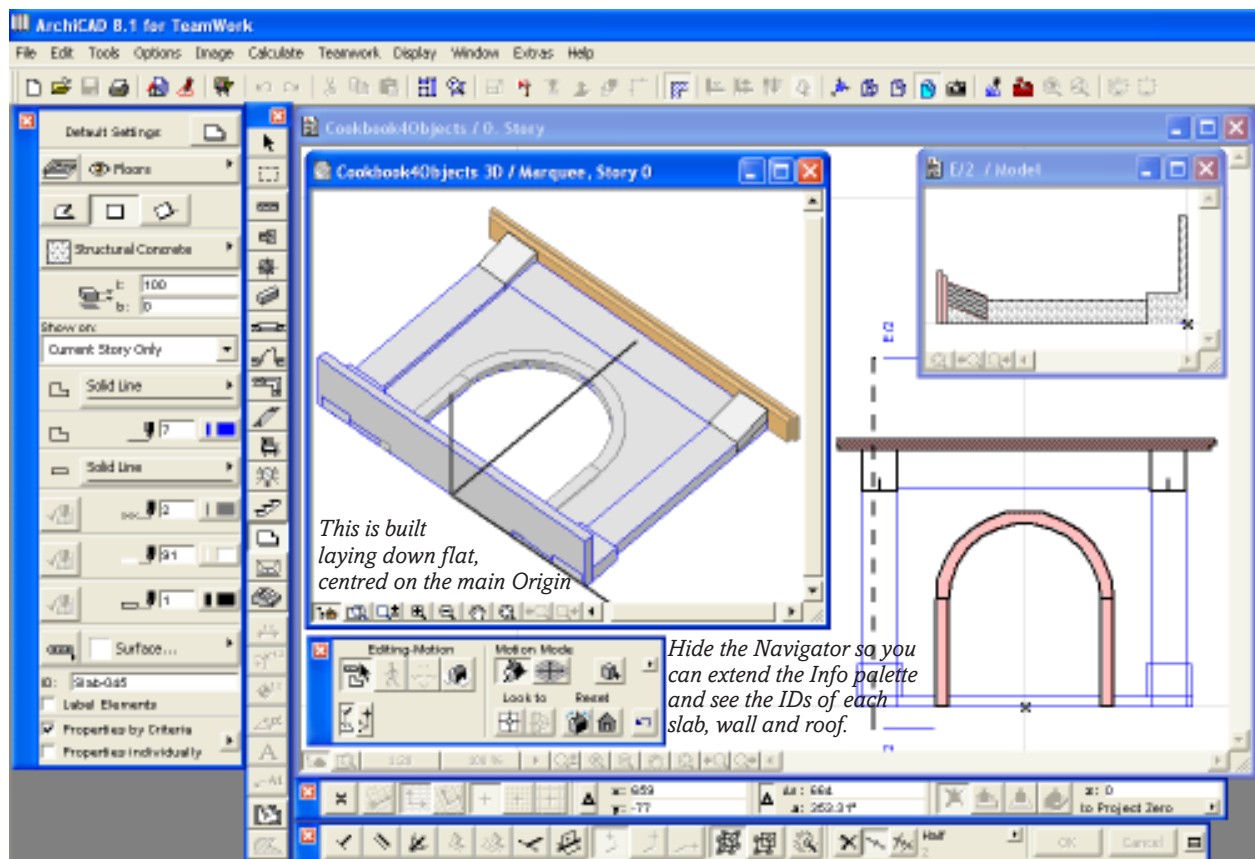
RECT2 -A/2,B-sthik+bhigt*sinr/2,
      A/2,B+bhigt*sinr
ENDIF

```



*True view and
scripted symbols*

Modify Library Object



THIS exercise is a demonstration of how to modify a library object autoscripted by ArchiCAD.

First, using slab, wall and roof tool, you knock out a model, like this. I haven't space to cover this stage in detail because it's not GDL – but you can make use of Profiler to get a better finish, and perhaps make the mantel corbels with the slab tool, saved as objects. My recommendation is to build it precisely ON the origin of the main model. Better still, set the small grid to 1mm, use 'snap-to-grid'. That reduces all the million millionths to 3 decimal places.

Saving it from the 3D view

For plan objects, position the camera using 3D Projection Settings so that it views it in parallel view in Plan with the camera at 270 degrees. Note that this fireplace is built on flat on the floor so you are required to position the camera as if looking from a 90° direction, in Elevational view.

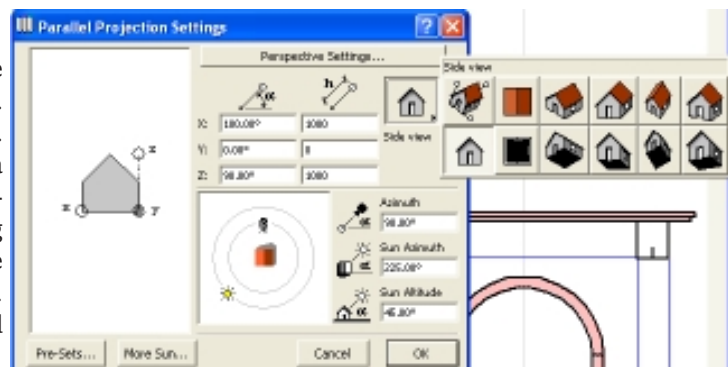
While viewing the 3D image, use the File>GDL Objects to do a Save 3D Model As... and save it as a ArchiCAD Object File (.GSM). You will see a box asking if it is an Object, a Window or a Door. Don't worry if the final destiny of the object is to be a Lamp or a Seating or Roofing object – you can change the subtype later. Tick the 'Remove redundant lines' box. We only want the visible top lines to be saved in the 2D symbol.

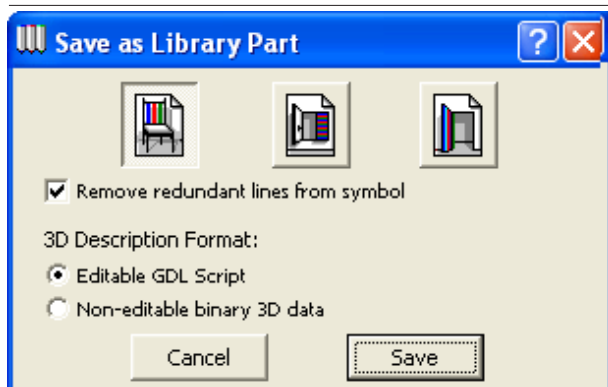
You are asked if you want to save as '**Editable**' or as '**non-Editable** Binary data. Never save as binary if you want to edit the object. The file will lose all parametric qualities except for size. A binary file has all been compressed into one indecipherable lump of data. This can be advantageous for complex objects as they will render more reliably and carry all macros with them. But that isn't the case here. Save it as **Editable**, into a **Loaded Library**.

Saving it from the Floor Plan

If the object is built flat on its backface on the ground, like a window or this fireplace, or built sideways on, you must use the 3D View procedure.

If the object is the right way up like a table should be, select the group of elements; from the File>GDL Objects Menu, Save Selection As... an 'ArchiCAD object' (.GSM). You save it to your loaded library.

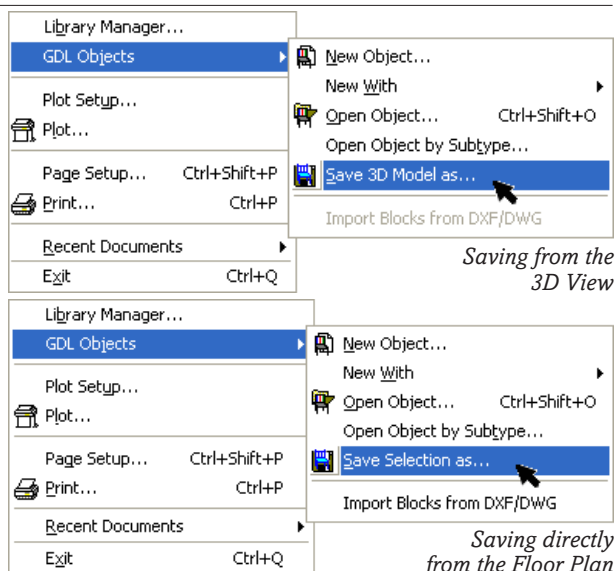




*Saving from the 3D – you are asked if it's an **Object**, a **Window** or a **Door**. If it's a window or door, viewed in 270° plan, GDL will save it with a 90° elevational view and apply a window or door Subtype.*

Now Place it and Open it

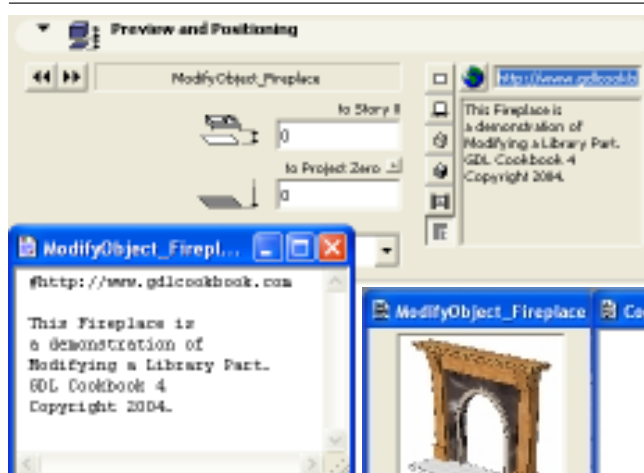
If you just saved it, you can click in the object's icon in the Tools Palette and if you are lucky, the last thing you saved is now available as an object (if it doesn't, you should have saved it into a Loaded Library). You can place it in the Floor Plan. Select it. You can then open it from the File>GDL Objects Menu with Open GDL Object... and you are ready to modify it or just gently tweak it.



First look at the scripts

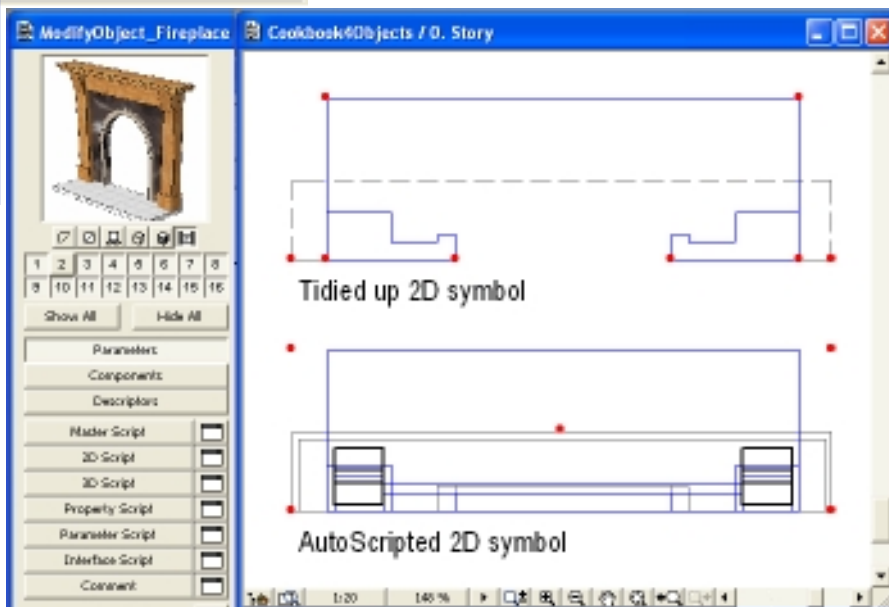
The 3D GDL will be more or less the same whichever method you saved by.

The 2D is different. If you used the Floor Plan method, the object is fully 2D Scripted with POLY2 and HOTSPOTS. If you use the 3D View method the object draws a 2D Symbol instead.



2D Symbol

This should be findable in the 2D Symbol window, not the 2D View window. This is a jumble of lines just as it came from the 3D wireline view. You can do yourself a favour by tidying it up – change Display Options to 'True Line weight' to see how the lines will look. Make the pens thin to avoid your object looking heavier than walls.



Hotspots in the 2D

The hotspots will also be at the corners, bounding box style, and you could now plant your own hotspots from the 2D toolbox. Keep hotspots at the extremities if you wish to retain stretchiness. If your object is NOT required to be stretchy, deliberately plant the new

Tidy up the object

Preview and Comment

First thing you can do to a new object is give it a nice **Preview Image** and write something into the **Comment field**., including perhaps your Website URL, name and date, using a hash # symbol before the URL

hotspots slightly inboard of the corners by zooming in really close – avoid the A,B corners. If you get this wrong, it is possible to have a 3D object with a non stretchy condition, but a 2D symbol that appears to be stretchy – or vice versa. Very confusing!

2D View & 2D Script : and FRAGMENT

If you have an object that you are going to modify in the 3D, or which got saved with a complicated but not so helpful 2D Script (as if saved from the Floor Plan) then you might consider erasing the 2D Script and simply putting in an easy PROJECT2 statement. You know what you are doing with this, it's safe country.

Any working 2D Script, however humble will take control and prevent the 2D symbol from displaying. The scripted version takes over. There are occasions when you write a 2D Script, change your mind, and then erase it, but the previous 2D symbol refuses to reappear (this is a bug). Or, perhaps you want to combine a 2D Script with something you have drawn in the 2D Symbol window. In this unlikely event, simply write FRAGMENT2 ALL,1 into the 2D Script and it forces the 2D Symbol to appear in the 2D View.

FRAGMENT2 is a very useful command that justifies a whole section to itself, later in the book.

3D Script

Now open the 3D Script and have a good look. At first you will be a bit surprised at all the housekeeping at the start of the script. There are an awful lot of lines with GLOB and BODY in them!

This long list of Global Variables (the GLOB statements) are simply a housekeeping record of the conditions prevailing at the time the object was made – current drawing scale, eye position etc. These can all go. BODY commands can go too. So can PENs.

In case you built the object some distance from the origin, go to the very end of the script and write in DEL TOP. The object will look better in the 3D View window. Better still, resave the object over the Origin.

The MUL statements allow the object to be stretchy. If your object is fixed in size and not stretchy, delete all the MUL statements. Leave the ADD commands in place, unless you built your object over the Project Origin.

You will also notice that each Slab or Wall has an ID field attached to it. This makes it easy to work through the 3D Script working out what each piece in the script relates to in the model.

Making Parameters

When you get further you can do a lot of manipulation of the GDL in the 3D Script, but for starters the two most useful things are PEN and MATERIALs. If the object was made with slabs and roof and walls, like the fireplace, it has disparate pens and materials. Make one Pen parameter for the 3D, 'gs_cont_pen' and make a few for Materials. Make a Section Fill parameter so that the fireplace looks solid in section views.

In the case of the Fireplace, you need material parameters for the shelf, the pilasters and capital, the back panel material, the hearth, and the moulding round the arch opening. You also get the AC_show2D Hotspots parameter which is in every library part and cannot be deleted. Leave it ON, only tick it OFF if you plan to write custom 3D hotspots.

Variable	Type	Name	Value
A		X Dimension	1500
B		Y Dimension	450
ZZYZX		Z Dimension	1050
AC_show2DHot.		Show 2D Hotspots in 3D	On
gs_cont_pen		3D Contour Pen	1
shelf_mat		Shelf material	52
pilas_mat		Pilasters + Beam material	52
backp_mat		Back Panel material	38
harth_mat		Hearth material	21
mould_mat		Moulding material	5
stretchy		Stretchy ON/OFF?	Off
secfil		Set Section Fill in 3D	24

Master Script

There's nothing to do here unless, in the example, you wish to prevent the object from being accidentally stretched in the 2D symbol or in 3D. Use the LOCK command, put the parameter names in quotes.

Revisiting Autoscripted objects

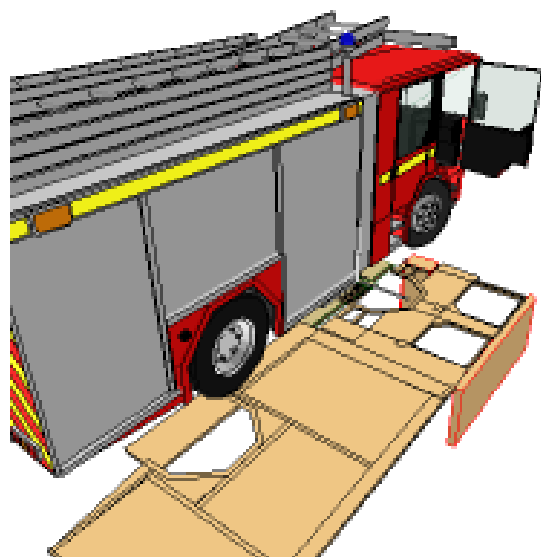
If you regularly make objects using ArchiCAD instead of GDL, what do you do with all those bits of slab and wall that you used? Do you keep them all in one monster .PLN file? It's impractical.



One good idea is to select the elements you have used to build an object and save them as a MODULE. This keeps them together in a tidy compact file, can be stored in a named folder, and ensures that you can easily recover them when you need to make another of the same or similar.

The Golden Rule: use the Origin

ONE of the most important rules to remember is to try to build the original object AT PROJECT ZERO ON THE MASTER ORIGIN of the Project. Moving the origin somewhere else doesn't work. You have to do it on the original origin. To move a temporary origin back to the original, click once on the temporary origin, hit delete, and it will restore the main origin.



This powerful British Fire Engine model was built in a day using the technique in this section: stealing prisms from ArchiCAD. A little bit of GDL knowledge is used to organise details like the glass and the doors into subroutines. Here you see it sitting on its pile of constructional prisms.

Back to the the 3D Script

On this side we have the script as it arrives from ArchiCAD (just the first column). On the other we have it tidied up.

```
! Document name:
! Name : Fireplace.gsm
! Date : February 2004
! Version : 8.10
! Written by ArchiCAD
!
MULX A/1.499717354774
MULY B/0.4499999880791
MULZ ZZYX/1.049610733986
ADDX 0.7544884085655
ROTX 270
ROTZ 180
BODY -1
MODEL SOLID
RESOL 36
GLOB_SCRIPT_TYPE = 3
GLOB_CONTEXT = 3
GLOB_SCALE = 20
GLOB_NORTH_DIR = 90
GLOB_DRAWING_BGD_PEN = 91
GLOB_FRAME_NR = -1
GLOB_EYEPOS_X = 19.2944
GLOB_EYEPOS_Y = -6.2127
GLOB_EYEPOS_Z = 1.5
GLOB_TARGPOS_X = 23.7703
GLOB_TARGPOS_Y = 6.04545
GLOB_TARGPOS_Z = 4.30919
GLOB_SUN_AZIMUTH = 225
GLOB_SUN_ALTITUDE = 45
BODY -1
BODY -1
PEN 1
SET MATERIAL "Paving-Asphalt, smooth"
CUTFORM 4, 1, 57,
    0, 0, 1, 0,
    -0.8150207352017, -0.142237848387, 15,
    0.7740288475367, -0.142237848387, 15,
    0.7740288475367, 1.402534810993, 15,
    -0.8150207352017, 1.402534810993, 15
GLOB_HSTORY_HEIGHT = 0.65
!!Wall-030
PEN 2
GLOB_LAYER="Exterior walls"
GLOB_ID = "Wall-030"
GLOB_INTID = 54
BODY -1
ADD 0.7366109980948, 1.029690951693, 0
ROTZ 180
xWALL_{2} "Oak", "Oak", "Oak", "Oak",
    0.2, 0, 1.465714637969, 1.465714637969, -
2.449212707645E-018, 0, -4.255225907027E-017,
0.02, 0.02, 0.02, 0,
    0, 0,
    15, 15, 15, 15,
    0,
    0
DEL 2
BODY -1
!!Wall-030
GLOB_INTID = 59
BODY -1
ADD 0.7544884238836, 1.049610751387, 0
ROTZ 180
xWALL_{2} "Oak", "Oak", "Oak", "Oak",
    0.22, 0, 1.499717385614, 1.499717385614, -
2.449212707645E-018, 0, -3.8388260989E-017, 0.02,
0.02, 0.02, 0,
    0, 0,
    15, 15, 15, 15,
    0,
    0
DEL 2
BODY -1
!the rest is omitted
```

The title can be shortened and renamed if you are putting work into this.

The 3 MUL commands are there to make the object stretchy. At the moment the Fireplace is 1.5m wide, and if A is stretched to a width of 1.8, the X-values are multiplied by 1.8/1.5. the others work in the same way. Delete them if the object is not stretchy.

There are no ADD commands because the object was built over the Origin.

The GLOBs can all be deleted, and so can the BODY and PEN and MODEL and RESOL statements at this point.

The CUTFORM statement is here because the original model was set in a Marquee. Delete CUTFORM and the CUTEND at the end.

Note, each element in the script has its own ID.

These GLOBS and Layer statements can also be deleted. Retain RESOLs if they occur later in the 3D Script.

Do not interfere with the xWALL statements, and retain the ADD and DELs either side of them.

Autoscripts are 'unstructured', meaning that each element is dealt with one at a time.

We have to delete all BODY statements, because we want to control the Textures in the object and we must be the one to issue BODY -1 statements. THE REST OF THE SCRIPT IS OMITTED

```
! ModifyObject_Fireplace.gsm
! Date : February 2004, Version : 8.10
! Written by ArchiCAD, Modified for GDL CB4
IF stretchy THEN
    MULX A/1.499717354774
    MULY B/0.4499999880791
    MULZ ZZYX/1.049610733986
ENDIF
ADDX 0.7544884085655
ROTX 270
ROTZ 180
PEN gs_cont_pen
SECT_FILL secfil,91,gs_cont_pen,gs_cont_pen
!Mantelshelf
ADD 0.7366109980948,1.029690951693,0
ROTZ 180
xWALL_{2} shelf_mat, shelf_mat, shelf_mat,
shelf_mat,
    0.2, 0, 1.46571, 1.46571, -0, 0, 0, 0.02,
    0.02, 0.02, 0,
    0, 0,
    15, 15, 15, 15,
    0,0
DEL 2
ADD 0.7544884,1.04961,0
ROTZ 180
xWALL_{2} shelf_mat, shelf_mat, shelf_mat,
shelf_mat,
    0.22, 0, 1.499717385614, 1.499717385614, -0,
0, 0, 0.02, 0.02, 0.02, 0,
    0, 0,
    15, 15, 15, 15,
    0,0
DEL 2
GOSUB 999:!Texture
!Remove all BODY statements in the script.
!Moulding round fireplace- Uprights
ADD -0.3, 0, 0
ROTZ 90
xWALL_{2} mould_mat, mould_mat, mould_mat,
mould_mat,
    0.07, 0, 0.424132503479, 0.424132503479, -0,
0, 0, 0.05, 0.05, 0.05, 0,
    0, 0,
    15, 15, 15, 15,
    0,0
DEL 2
ADD 0.3, 0.424132503479, 0
ROTZ 270
xWALL_{2} mould_mat, mould_mat, mould_mat,
mould_mat,
    0.07, 0, 0.424132503479, 0.424132503479, -0,
0, 0, 0.05, 0.05, 0.05, 0,
    0, 0,
    15, 15, 15, 15,
    0,0
DEL 2
ROTZ 90
GOSUB 999:!Texture
DEL 1
!Moulding round fireplace- Curved section
RESOL 37
ADD 0.35, 0.424132503479, 0
ROTZ -269.2410240448
xWALL_{2} mould_mat, mould_mat, mould_mat,
mould_mat,
    0.07, 0, 1.090380453073, 1.090380453073, -0,
0, 0, 0.05, 0.05, 0.05, 0.3500307100535,
    0, 0,
    15, 15, 15, 15,
    0,0
DEL 2
GOSUB 999:!Texture
```

Put an IF... ENDIF routine round the MULs.

Leave the ADD and the ROTs. Set the contour PEN and delete all later pens. Add in the SECT_FILL

Identify each part of the script and put a title over it. Work each part out (wall, roof or slab) by logic or by ID field, and how thick each one was.

Put a 999:!Texture subroutine at the end of the script, copied from the Chair object, and you can texture map every part of the fireplace.

Replace all the material statements with Parameters, NOT in quotes.

Remember with the Texture subroutine that the object was built flat on the floor, so you ROTz to make vertical texture.

There is more of this on the next page.

How do you identify which parts (walls etc) are which? Easy, they all have ID numbers, and so do the sections of the script.

Continued from previous page

!Main Backpanel**RESOL 36**

```
cPRISM_ backp_mat, backp_mat, backp_mat,
27, 0.05,
-0.5, 1.009690951693, 15,
-0.5, -7.549694427109E-017, 15,
-0.3, -7.549694427109E-017, 15,
-0.3, 0.424132503479, 79,
-0.3003476993072, 0.442430660001, 79,
-0.2919171850616, 0.494247123188, 79,
-0.274616914865, 0.543812434433, 79,
-0.2489725486731, 0.589620576834, 79,
-0.215763277576, 0.6302796931654, 79,
-0.1759981484712, 0.664554376751, 79,
-0.1308854046845, 0.691403208675, 79,
-0.08179577410767, 0.710010400076, 79,
-0.03022082032523, 0.71981058290, 79,
0.02227237779032, 0.720505981518, 79,
0.07408884097709, 0.712075467273, 79,
0.1236541522215, 0.6947751970765, 79,
0.1694622946227, 0.6691308308846, 79,
0.2101214109539, 0.6359215597875, 79,
0.2443960945398, 0.5961564306828, 79,
0.2712449264634, 0.551043686896, 79,
0.2898521185531, 0.5019540563192, 79,
0.2996523006928, 0.4503791025367, 79,
0.3, 0.424132503479, 79,
0.3, 0, 15, !E-minus number
0.5, 0, 15, !changed to Zero
0.5, 1.009690951693, 15,
-0.5, 1.009690951693, -1
GOSUB 999:!Texture Horizontal
```

!Pilasters

```
cPRISM_ pilas_mat, pilas_mat, pilas_mat,
5, 0.1,
0.5, 1.009690951693, 15,
0.6364136155509, 1.009690951693, 15,
0.6364136155509, 0, 15,
0.5, 0, 15,
0.5, 1.009690951693, -1
cPRISM_ pilas_mat, pilas_mat, pilas_mat,
5, 0.1,
-0.6364136155509, 1.009690951693, 15,
-0.5, 1.009690951693, 15,
-0.5, 0, 15,
-0.6364136155509, 0, 15,
-0.6364136155509, 1.009690951693, -1
```

!Base blocks

```
cPRISM_ pilas_mat, pilas_mat, pilas_mat,
5, 0.13,
-0.65584, 0.1695, 15,
-0.47905, 0.1695, 15,
-0.47905, 0.0377, 15,
-0.65584, 0.0377, 15,
-0.65584, 0.1695, -1
```

```
cPRISM_ pilas_mat, pilas_mat, pilas_mat,
5, 0.13,
0.48036, 0.16950, 15,
0.65715, 0.16950, 15,
0.65715, 0.03774, 15,
0.48036, 0.03774, 15,
0.48036, 0.16950, -1
```

ROTz 90

```
GOSUB 999:!Texture Horizontal
DEL 1
```

!Hearth slab

```
cPRISM_ harth_mat, harth_mat, harth_mat,
5, 0.45,
-0.655842, 0.037741, 15,
0.6591421, 0.037741, 15,
0.6591421, 0, 15,
-0.655842, 0, 15,
-0.655842, 0.03774123, -1
```

GOSUB 999:!Texture Horizontal

Sorry, space doesn't permit me to do a more sophisticated object with curly corbels and profiled mantel edges.

Do not delete RESOLs that occur later in the script.

This curved arch is converted to a series of Polypoints, with a 79 (15+64) smoothing masking value.

Because the locations are stored in million millionths of a metre accuracy, the ArchiCAD parser must spend longer reading through all that data and building the object.

On this page, the E-minus numbers are approximating to Zero and have been changed to Zero.

If you are a fanatical pruner, you can even truncate the XY

coordinates to 4 or 5 decimal places – this speeds up the reading of the data. Make sure the start and end points are the same.

This hearth was 'eyeballed' in ArchiCAD.

Now you can see that the X values could be replaced safely with 0.66 and -0.66, and the Y with 0.038, and would be more precise.

!Beam

```
cPRISM_ pilas_mat, pilas_mat, pilas_mat,
5, 0.08,
-0.5, 1.009690951693, 15,
-0.5, 0.856795635428, 15,
0.5, 0.856795635428, 15,
0.5, 1.009690951693, 15,
-0.5, 1.009690951693, -1
```

GOSUB 999:!Texture Horizontal**!Capitals atop the Pilasters**

```
cSLAB_ pilas_mat, pilas_mat, pilas_mat,
5, 0.096447260198,
-0.6364136, 0.844527895, 0.0169, 15,
-0.5, 0.84452789571, 0.0169, 15,
-0.5, 1.00969095169, 0.0770, 15,
-0.6364136, 1.00969095, 0.0770, 15,
-0.6364136, 0.844527895, 0.0169, -1
```

```
cSLAB_ pilas_mat, pilas_mat, pilas_mat,
5, 0.09644726,
0.5, 0.8445278957155, 0.0169, 15,
0.63641361555, 0.8445279, 0.0169, 15,
0.63641361555, 1.009691, 0.0770, 15,
0.5, 1.0096909516, 0.0770, 15,
0.5, 0.8445278957, 0.0169, -1
```

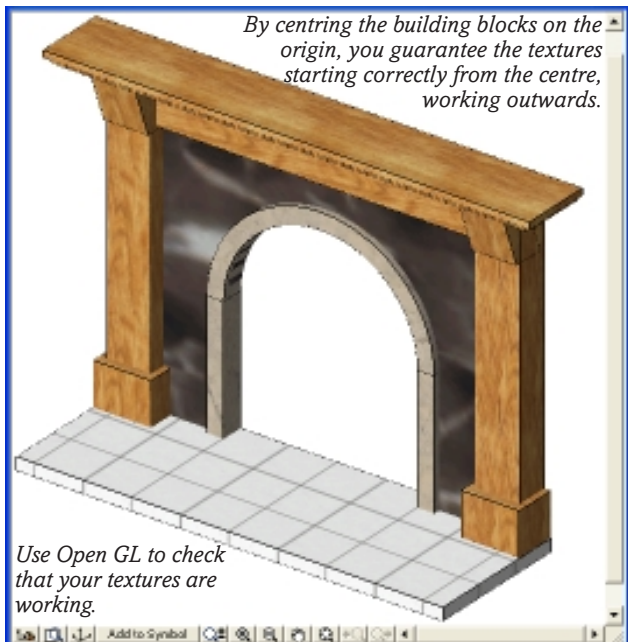
ROTz 90**GOSUB 999:!Texture Horizontal****DEL 1****DEL TOP****END:!****999:!Texture Horizontal****BASE**

```
VERT 0,0,0
VERT 0.1,0,0
VERT 0,0.1,0
VERT 0,0,0.1
COOR 256+2,-1,-2,-3,-4
BODY -1
RETURN
```

Roof elements become cSLAB in autoscripts

This is the routine that you can apply here, there and everywhere. Because the fireplace is built on the floor, don't forget that 'vertical texture requires ROT Z and not Y'.

*If you apply **Align Texture** while building in ArchiCAD, the Coor Vert routine may be included in the autoscript!*



3D ArchiCAD or Open GL?

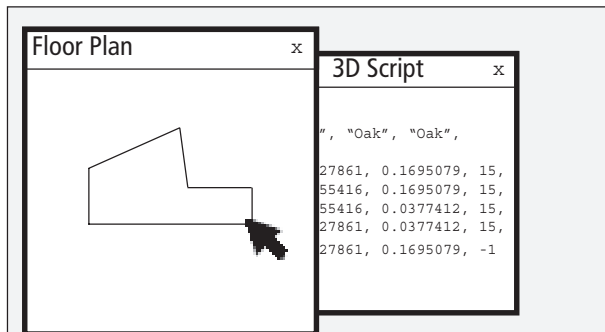
WHEN working in GDL, which 3D drawing mode should one use? Speed is not an issue for small models – so what else? Quality?

ArchiCAD's 3D engine gives sharper and brighter 3D, with more visible polygons... but if you are working with Textures there is only one winner! Open GL is a real time saver, it saves the bother of using Photorendering to check out the direction of your wood grain.

Instant GDL!

EVERY element in ArchiCAD can be expressed in GDL because it is the underlying language of ArchiCAD. If you want to see what anything looks like in GDL, try this and be amazed.

Arrange your project window so it occupies half the screen. Open a New GDL object file and arrange the empty floating 3D Script window so it occupies the other half (click the white script button to make it float).



Grab the 3D element (Slab, Roof, Wall, Mesh) by an **edge** and just drag it **smartly** into the 3D Script which should be in an overlapping window. If the object is a 2D element (Fill, Lines), drag it into the Master or 2D Script.

Steal this chunk of ArchiCAD!

Now select any part of the Project, grip the part(s) by an edge and Drag'n'Drop across and into the 3D Script as quick as you can! **Instant GDL!!**

Tidy up the garbage at the head and tail and you now have some clean usable GDL for the object you dragged over.

You can do the same with any group of 2D elements, Drag'n'Drop into the 2D Script window. It's brilliant! Most complex 3D items such as EXTRUDE, REVOLVE and the profiles for TUBE and SWEEP use the same syntax as a POLY2. So you can copy the *X_Y_mask* data from a Fill and paste it into the 3D Script for a complex 3D command.

Now it's in a script, put a number label at the top and 'RETURN' at the bottom and you have a perfect subroutine!

This technique only works satisfactorily if the objects to be dragged are correctly located relative to the main origin. If somewhere else, then you will have crazy coordinates, many metres or feet away.

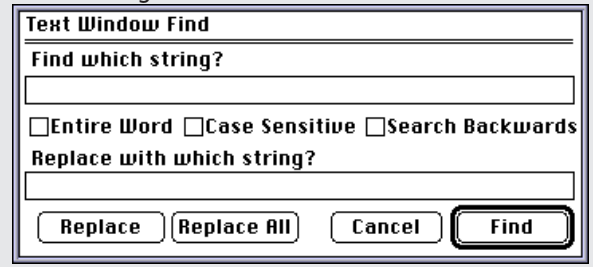


This authentic, parametric Helicopter started out as a set of Fills in ArchiCAD, dragged into GDL.

Find and Replace

If the script is really very long, then you can use the **Find and Replace** feature quite effectively. It speeds up reading of the script if you can remove a lot of the clutter for you and for the machine. Search and replace all instances of 'GLOB_' and 'PEN' with '!'....., or a double space with a single (to compact the code), and more besides.

In more advanced work, e.g. converting an imported file from Rhino, I have found that you can replace all 'LIN_' with 'PUT' and thus put all the values into memory. You can then write a GET routine to get all the values out and do something useful with them.



Experts use this method

This procedure is quite convenient even for hardened GDL users. It is good for knocking out complicated shapes, without having to work out the XY locations yourself. Look at the Helicopter!

Place a 2D symbol of the object you want to make (like the Bus or Truck from the ArchiCAD Library), explode it, and trace over the main body panels with the slab tool. Drag each slab into the 3D Script window, label each one, and then reorganise each prism with the correct thickness and material.

Once you have got the hang of this technique, you can 'steal' the X,Y locations in Slab/Prisms to define many other things, such as the pathway of a tube, or the locations of glazing bars. It's very useful for creating the outline for a shape that you want to lathe around an axis using REVOLVE.

If you want to use the numbers later, delete the statement header (e.g. cPRISM_ etc, leaving just the X,Y,15 numbers, put PUT in front of them and you can store them in memory. That's an expert technique that is explained fully later in the book.

A Use for the Patch Tool

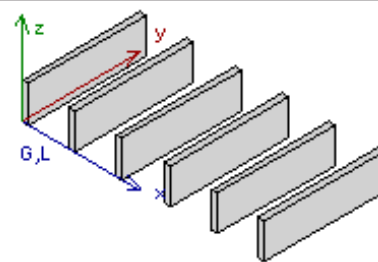
If you have something complex to convert to 2D and it's not realistic to move to the origin, the PATCH tool is a great helper! Make a patch of the bit you want, save it, place the patch over the origin, explode, and your 2D is now in the right place, without disturbing the original.

Mac OSX Workaround

This Drag'n'Drop technique, sadly, doesn't work in OSX on the Mac, something to do with OSX, not the fault of ArchiCAD. But you can click on the object you want to drag over, 'Save Selection as' something like *thing.gsm*, place it in the plan, open GDL Object, and then you can copy chunks of script and paste them into the main object you are inserting them to – more long winded, but perfectly doable.

FOR... NEXT Loops

The inexhaustible workhorse of GDL



FOR k=1 TO n: NEXT k

FOR non-programmers, the FOR... NEXT Loop seems at first a challenge. It is in reality your strongest friend, because it has a tremendous capacity for work. With a few economical lines of script to make a FOR... NEXT loop, huge lattices or structures can be created.

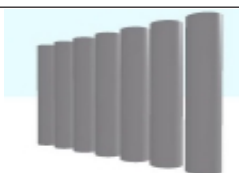
When you want to repeat an operation, there will be more than one way in which you can repeat it. Think of music. On your guitar, you can strum a chord ten times, at your own speed – so NUMBER is the controlling parameter. On the other hand you could strum the chord metronomically until the conductor tells you to stop. Try changing the interval of the strum. In this case TIME is the parameter – the frequency of each chord, and duration of the total will determine how many chords get played.

In 3D, there are three main ways to organise a FOR NEXT loop: Distance, Number and Angle.

- **Distance:** where total distance could decide how many objects get laid – keep laying them stepping at a set spacing until you reach the target distance.

Example: you lay joists at 400mm spacing across a room 4000m wide.

```
FOR k=1 TO 7
CYLIND 2,0.2 !Pole
ADDx 0.5
NEXT k
DEL 7
```



Let's plant 7 poles at fixed spacings

NUMBER: For this script we nominate 'k' to be a counter – the value changes from 1 to 7. Having moved 7 times, you issue a DEL 7 command after NEXT to get back to the origin before your next operation. This is not a good idea as you have a DEL 'hanging off the end'.

It's better to do both the move and DEL inside the Loop. Do the move inside the loop by moving one distance, DEL to the origin, move a longer distance and DEL, and so on.

The first time you want to move zero distance, the second time you want move one, the third you move two. So multiply the 'spacing' by (counter-1).

```
FOR k=1 TO 7
ADDx 0.5 * (k-1)
CYLIND 2,0.2 !Pole
DEL 1
NEXT k
```

As a matter of discipline, always try to use i, j, k, m, or n for loops based on round numbers (integers). Never use these variables for anything else, especially dimensions.

- **Number:** where you are told to lay a fixed number of objects at a set spacing, stepping one at a time.

Example: you are given 10 joists and told to lay them at 400mm spacing.

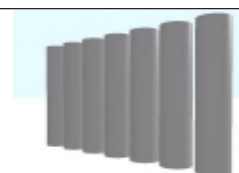
- **Angle:** You are laying out objects at a stepping angular spacing, until you reach your target angle.

Example: you are placing columns to enclose an octagon at a spacing of 45° (relative to the centre) until you reach 360°.

In the **FOR** statement on the lower left, read it to yourself in slow, elongated english. *Using a counter called 'k', for values of 'k' starting with 1, Do What Has to be Done. When you get to NEXT k, if the value of k isn't 4 then go round again, increment 'k' by 1 and Do What Has to be Done again. Keep cycling between FOR and NEXT until it reaches a value of 4. When 'k' reaches 4, you have finished, so execution continue to the rest of the script.*

NEXT: In every case, NEXT must be followed by the name of the variable. In a more complex model, you could have some FOR...NEXT loops nested inside each other, and it is important that GDL knows which loop you are referring to.

```
FOR dist=0 TO 3.0 STEP 0.5
ADDx dist
CYLIND 2,0.2 !Pole
DEL 1
NEXT dist
```

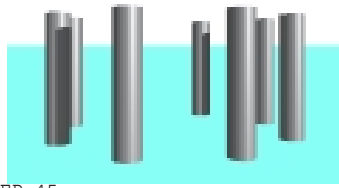


Plant an unknown number of poles

DISTANCE: We don't know how many poles there will be – we let GDL work that out for us – until we run out of road. We have three metres of road to plant. Here, 'dist' acts as a distance dimension. Do not use 'k' here. It's best to associate 'k' in your mind with integer counters, not with 'real numbers'. In the first run through the loop, 'dist' is zero, so the cursor moves zero, plants a pole, then returns to the origin – and then does it again with a new value of 'dist'. For this, the loop needs to be told the 'stepping' rate of each cycle in the loop. If no stepping rate is given, it assumes you mean a value of One. Here, you want to step half a metre each time.

As you have been using DEL inside the pole planting loop, you do not need to finish with a DEL. Moreover, you do not need to worry about how many poles to plant. When it has planted 7 poles and reached a distance of 3.0 metres, it realises that if it plants another, it will go to 3.3 metres, which is too far. Therefore, it stops at 3.0 metres run.


```
FOR ang=0 TO 315+1 STEP 45
  ROTz ang
  ADDx 2.0
  CYLIND 2,0.2
  DEL 2
NEXT ang
```



!Trig Method

```
rad=2.0
FOR ang=0 TO 315+1 STEP 45
  ADD rad*COS(ang),rad*SIN(ang),0
  CYLIND 2,0.2
  DEL 1
NEXT ang
```

In both of these, the principle is that we return the cursor to zero within the loop at the end of every cycle. We usually add a single spare degree to the end value to counteract any adding up error – in this case there is a risk the succession of 45s might add up to 314.99999. The extra degree will not affect the number of poles planted. If you forget to put in a stepping value, it will be assumed to be 1 degree.

Let's plant poles at angles until we have a circle

ANGLE: In this case we can use 'ang' as the counter. We rotate around z, go out to the edge of the circle, plant a pole and then DEL back to the centre. We want to make an octagon, so we use an angle of 45° for the spacing. We do not need to go all the way to 360° because we already planted one when the value of 'ang' was zero.

There's another way to go round a circle, using trigonometry. At first sight it seems more difficult, but it's an essential technique e.g. for tracing out the points for the edge of a prism or the pathpoints along a tube. Look ahead to the Maths Primer for more detailed explanation. At every point on the circle, the radius is actually the hypotenuse of a right angle, and the XY location can be calculated with a COS/SIN calculation.

!Master Script

```
numbay=INT(A/spac) !Number of bays
```

!3D Script

```
FOR k=1 TO numbay+1
  ADDx spac*(k-1)
  CYLIND 2.0,0.2
  DEL 1
NEXT k
```

This little master script routine calculates the number of spaces created. When we lay the poles, we have to remember to include the last pole in the sequence.

!2D Script

```
HOTSPOT2 0,0 !Stretch
HOTSPOT2 A,0 !-spots
FOR k=1 TO numbay+1
  ADD2 spac*(k-1),0
  CIRCLE2 0,0,0.2
  HOTSPOT2 0,0
  DEL 1
NEXT k
```

As an example, we show that by copying and pasting the 3D Script into the 2D, we can amend it to work efficiently in 2D. Circle and Hotspot have XY locations, so we could shorten this:

```
FOR k=1 TO numbay+1
  CIRCLE2 spac*(k-1),0,0.2
  HOTSPOT2 spac*(k-1),0
NEXT k
```

!Master Script

```
numbay=INT(0.5+A/spac) !Number of bays, rounded
spa=A/numbay !Actual spacing of poles
```

!3D Script

```
FOR k=1 TO numbay+1
  ADDx spa*(k-1)
  CYLIND 2.0,0.2
  DEL 1
NEXT k
```

Pen and material parameters have been omitted to keep this routine simple.

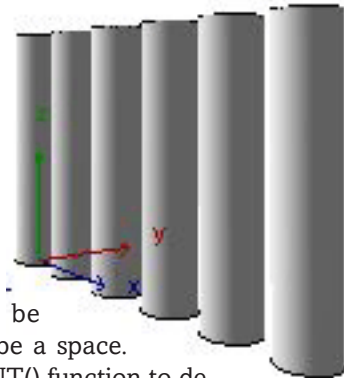
!2D Script

```
HOTSPOT2 0,0 !Stretch
HOTSPOT2 A,0 !-spots
FOR k=1 TO numbay+1
  CIRCLE2 spa*(k-1),0,0.2
  HOTSPOT2 spa*(k-1),0
NEXT k
```

We calculate a value for 'spa' the ACTUAL spacing, and then lay the poles. The last pole will be exactly on the A-spot.

What do we do when we know distance and spacing, but want to know number?

Assume that the required distance is 'A'. We only need one parameter, that of 'spac' for spacing. This routine will space the poles out exactly and if there is remaining distance left over, no additional poles will be planted – there will be a space. We use the Integer INT() function to devise an exact number of poles.



What if we want the poles to fit exactly to 'A' with the last pole perfectly on the A-spot?

We now have to assume that 'spac' is the user's optimum but not the actual spacing. Use the Master Script to work out the most likely number of poles, rounded up or down, and then work out the exact spacing 'spa' to achieve a result. Adding 0.5 in the Integer statement results in it rounding up or down to the nearest whole number.

```
!Parameters rad=5.0 , spac=0.6 , endangl=270
```

!Master Script

```
circum=2 * PI * rad !circumference
spangl= 360*spac/circum !spacing angle
numbay= INT(0.5+endangl/spangl)
spang = endangl/numbay
```

!3D Script

```
FOR ang=0 TO endangl+1 STEP spang
  ROTz ang
  ADDx rad
  CYLIND 2,0.2
  DEL 2
NEXT ang
```

Optimise the spacing angle: work out the optimum number of bays between posts, work out the required spacing angle to meet the endangle exactly. Add the 1 degree as a protection against a cumulative adding up error

Does the above trick work for angles?

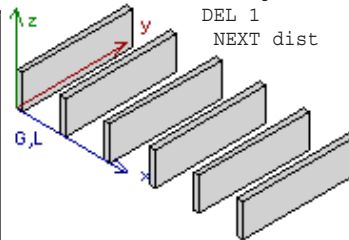
Yes! Try an adaptation, using an angular difference instead of a linear difference.

But how about when you want a **linear** difference around the edge of an arc or circle? e.g. distribute posts at 600mm spacing around the perimeter for 270 degrees sweep: and arrive sweetly on the end angle. Here one divides the spacing by the circumference and multiply by 360 to arrive at the optimum spacing angle 'spangl'. Use this trick to finalise an exact spacing angle, and go!

Quick Loop Object with FOR NEXT : Stretchy Joist Tool

TRY this simple joist tool – just a FOR NEXT loop with Blocks in the 3D, Rect2s in the 2D and some Hotspots – easy!

A	X Dimension	2100
B	Y Dimension	1000
ZZYZX	Z Dimension	300
AC_show2DH..	Show 2D Hotspots in 3D On	<input checked="" type="checkbox"/>
gs_cont_pen	Pen colour	1
bmat	Material	18
spa	Spacing	400
jwid	Joist Width	50



```
!Simple joistool
!3D Script
```

```
PEN gs_cont_pen
MATERIAL bmat
For dist=0 TO A STEP spa
  ADDx dist
  BLOCK jwid,B,zzyzx
  DEL 1
  NEXT dist
```

```
!Simple beamtool
!2D Script
```

```
PEN gs_cont_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A,B
HOTSPOT2 0,B
HOTSPOT2 A/2,B/2
FOR dist=0 TO A STEP spa
  ADD2 dist,0
  RECT2 0,0, jwid,B
  HOTSPOT2 0,0
  DEL 1
  NEXT dist
```

Other types of Loop: REPEAT and DO

REPEAT...UNTIL / DO WHILE

OTHER commands related to loops are **REPEAT...UNTIL**, and **DO WHILE** and **WHILE... DO... ENDWHILE**.

REPEAT... UNTIL and DO WHILE are the best ways to run loops where only a trial-and-error approach will work.

In a REPEAT Loop, there may be an unknown number of occurrences, and you can keep a tally of these. You keep it going until the tally exceeds a value, or falls lower than zero.

Example of REPEAT UNTIL: In Music, you might be playing something very modern where you strummed the guitar at random intervals – and finished after the total number of intervals add up to twenty minutes. This is a REPEAT UNTIL loop (until time=20). Everybody will be very grateful when 20 mins is up. Every time you played the piece, the total number of chords played would be different and could be counted.

Staying with the Music, WHILE... DO occurs while a value remains known (e.g. n>0). If something happens that changes that condition, then you can escape from the loop. For example, while people stayed to listen, you could play. You can't forecast the rate of walk out. But you DO play: WHILE audience>0.

```
!DO WHILE Loop
k = 1
DO
  CYLIND 2.0,0.2
  ADDx 1.0
  k = k + 1
WHILE k <= 5

!REPEAT UNTIL Loop
k = 1
REPEAT
  CYLIND 2.0,0.2
  ADDx 1.0
  k = k + 1
UNTIL k >= 5

!WHILE DO Loop
k = 1
WHILE k <= 5 DO
  CYLIND 2.0,0.2
  ADDx 1.0
  k = k + 1
ENDWHILE
```

Note that with DO and REPEAT, you must ensure that the loop is 'legal', e.g. no division by zero, and no risk that the UNTIL or WHILE conditions will not be invalid.

Some people never use WHILE or REPEAT loops. Sooner or later, as you use GDL, you will come across a situation where you realise that this is where you need to use one of them.

Two occasions I use it for are: when clearing the memory buffer in case you have anything in it after some PUT statements (see later in the book), or applying CUTEND statements after an unknown number of CUTPLANES.

LOOPS: Some Golden Rules

- Loops can be based on Distance, Angle or Number.
- Do not change the value of the variable that is being used as the counter or the end of the loop while in the loop. For this reason, you cannot write FOR k=1 TO NSP.
- If you use a STEP value in FOR... NEXT and forget to give it a real value (other than zero), the loop will be stepping zero each time, for infinity. ArchiCAD may never recover. Alt-Ctrl-Del will be your friend!
- If you REPEAT towards something that is unattainable, then GDL will never stop. So continue until a value is greater or less than, not equalled. For example in a decrementing value, REPEAT UNTIL K<=0.
- Once you are in a loop, you must never branch out of it, except to use a Subroutine. GOSUBs are perfectly safe. GOTO commands are quite forbidden, unless you go to a label inside the loop.
- If you don't know exactly how long the loop will be, use REPEAT... UNTIL, or WHILE ... DO... ENDWHILE.
- If the contents of the loop could cause an error, use WHILE DO, not DO WHILE or REPEAT UNTIL. If the condition does not apply, GDL will not enter the loop.
- To avoid an adding up error add a little deliberate overshoot to the end value in a FOR_NEXT e.g. 361° instead of 360° in a circle. This has no effect on the 3D result.
- It is possible to 'Nest' one loop inside another. Just make sure that you do not use the same counter. Get the NEXT statements in the right sequence.

3D Elements : Planar

PLANAR 3D elements are very useful because they only generate two polygons, 1 top, 1 bottom. When it comes to rendering speed, the 'polygon count' of a model is all important (because AC has to calculate the line outlines and the shading values for each polygon.)

CIRCLE is simply defined by its radius. Use it instead of thin cylinders.

ARC draws a curved line sector shape. The rotation angle goes in an anti clockwise direction starting from horizontal. Here, the start angle is 45 degrees, and it finishes with 100 degrees – anti clockwise.

LIN_ draws a straight line in 3D from one XYZ location to another; but only shows up in 3D views with Best contours.

RECT draws a rectangle of width X and depth Y.

POLY the syntax is identical to that of PRISM, without a thickness. No holes or curved edges or masking codes are possible. POLY must always be laid on the X-Y plane, so if you want a poly in an odd place and angle, you have to move the cursor there first.

POLY_ (Poly underscore) extends the power of POLY – it supports holes and curved edges and masking codes. With POLY_ you only need to know if the line is to be drawn or omitted – so a code of one or zero is enough. In addition, by adding 1000 or other polylines, you get curved lines or holes. As a bit of adventure, I have shown you an example of a hole drilled in the Poly – this is similar to the procedure for holes in PRISM.

Look ahead in the book for more on Polylines and Masking Codes.

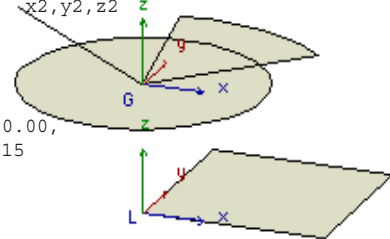
HOTSPOT x,y,z is a hotspot that can be at any 3D location defined by X, Y and Z. Of which more will be said later in the book

CIRCLE, POLY, PLANE and RECT all appear in renderings – but LIN_ and HOTSPOT disappear. Whenever you would normally use a PRISM or a CYLIND, stop to think whether the same job can be done with a PLANE or a POLY or a CIRCLE.

!Planar Three-D elements

```
!Syntax:-
!CIRCLE radius
!ARC radius, startangl,endangl
!LIN_ x1,y1,z1, x2,y2,z2
!RECT x,y
```

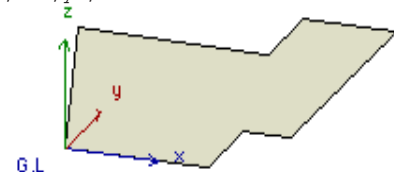
```
CIRCLE 0.2
ARC 0.3,45,100
LIN_ 0.00,0.00,0.00,
-0.15,-0.15,0.15
ADDz -0.2
RECT 0.3,0.3
```



!POLY Demo

```
!Syntax:- POLY number,
!          x1,y1, x2,y2, etc
```

```
POLY 9,
0.00,0.00,
-0.10,0.30,
0.10,0.30,
0.10,0.40,
0.20,0.40,
0.20,0.10,
0.15,0.10,
0.15,0.00,
0.00,0.00
```



Example of **POLY** – a simple outline with straight edges.

!POLY_ Demo w' hole

```
!Syntax:- POLY_ number,
!          x1,y1,mask,
!          x2,y2,mask, etc
```

```
POLY_ 9+2,
0.00,0.00,1,
-0.10,0.30,1,
0.10,0.30,0,
0.10,0.40,1,
0.20,0.40,1,
0.20,0.10,1001,
0.15,0.10,0,
0.15,0.00,1,
0.00,0.00,-1,
0.05,0.1,900, !Hole centre
0.05,360,4000 !Hole drawn
```



Example of **POLY_** – an outline with the opportunity for curved & straight edges and drilled holes (round or polygonal).

PLANE is most useful: you can specify XYZ locations of any points in 3D space and GDL will endeavour to join them up with a surface.

With three points, you can do anything anywhere. With more than 3, you must ensure that the surface will be planar. If they are not planar, the whole surface may appear as empty.

You can position Plane anywhere: it does not require ADDs and ROTs to get to its location. It pays to have a good grasp of trigonometry to get the best out of Plane.

PLANE_ (Plane underscore) allows you to decide if edge lines are to be drawn or not.

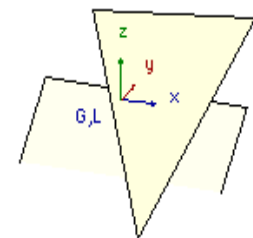
!PLANE Demonstration

```
!Syntax:- PLANE number,
!          x1,y1,z1,
!          x2,y2,z2,
!          x3,y3,z3, etc..
```

```
PLANE 3,
0.20,-0.4,-0.2,
-0.20, 0.3, 0.1,
0.25, 0.4, 0.1
```

```
!Syntax:- PLANE_ number,
!          x1,y1,z1,s, etc.
```

```
PLANE_ 4,
-0.3,0.0,-0.20,0,
0.3,0.0,-0.20,1,
0.3,0.2,-0.05,1,
-0.3,0.2,-0.05,1
```



Imported 3D DXF files often get converted to a large number of PLANEs when read in by GDL.

3D Elements : SLAB

SLAB

SLAB is very similar to PRISM, but far more flexible. You only need to define the XYZ points of the nodes – like PLANE. As with Prism, you define a thickness, but depending on the pitch, this is really a vertical height quantity, not true thickness. The XYZ points are actually the points on the lower surface unless you specify a Negative Height.

Plain SLAB and SLAB_ are all one material.

With SLAB is that you need to be sure that the surface will be planar – or it may not render properly. It does not check for you.

SLAB retains its sides perpendicular to the ground, or to the current XY plane – making it ideal for ramps, roof slabs etc. If the pitch gets too steep, the ridge and eaves detail get very distorted.

SLAB is good because you do not have to move the cursor to its location, or Rotate the XY plane, as you do with a Prism.

SLAB_

SLAB_ is similar, but you can control the pen lines and edges in 3D drawings. (However, all the edges in the illustration here still get shown in photo render.) SLAB_ does not appear to support the drilling of holes – you have to use CROOF, or PRISM_. SLABS do not support PolyLines.

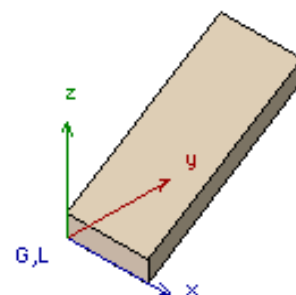
CSLAB_

CSLAB_ is **SLAB_** with the ability to set different materials for top, bottom and edge surfaces. CSLABS are what you get when you make an autoscripted library part using hipped roofs. The Materials can be defined by name, or by index number, or with variables.

This is analogous to cPRISM. Because of the way it handles the edges, and because you do not have to lift or rotate the cursor into position, the SLAB family can be more useful than Prism.

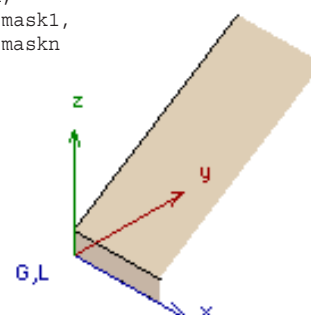
```
!SLAB Demonstration
!Syntax:- SLAB n,h,
!         x1,y1,z1,
!         ...xn,yn,zn
```

```
SLAB 4,0.2,
0.0,0.0,0,
0.8,0.0,0,
0.8,1.5,1,
0.0,1.5,1
```



```
!SLAB_ Demonstration
!Syntax:- SLAB_ n,h,
!         x1,y1,z1,mask1,
!         ...xn,yn,zn,maskn
```

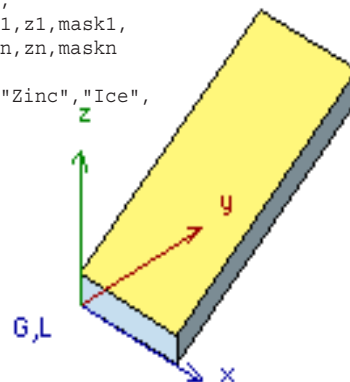
```
SLAB_ 4,0.2,
0.0,0.0,0,15,
0.8,0.0,0, 0,
0.8,1.5,1, 8,
0.0,1.5,1,15
```



With SLAB_, you can control the display of edges and edgelines, as with Prisms.

```
!cSLAB_ Demonstration
!Syntax:- CSLAB_ topm,botm,sidm,
!         n, h,
!         x1,y1,z1,mask1,
!         ...xn,yn,zn,maskn
```

```
CSLAB_ "Gold","Zinc","Ice",
4, 0.2,
0.0,0.0,0,15,
0.8,0.0,0,15,
0.8,1.5,1,15,
0.0,1.5,1,15
```



Typographical format for all XY based 3D solids

- Write the keywords PRISM, SLAB etc. in upper case. Only use an underscore form of prism or a more complex 3D form if you need to make use of masking, polyline curves or holes.
- Indent the XY numbers – it is much easier to read and repair. Tabulate the XY numbers using commas to line up the list of numbers. Express numbers in the 0.00 format, so that all the commas line up. Even if they do not line up, force the commas to line up, using your spacebar.
- When you have a problem with Prism or another XY based solid, it is often because you have made an error with the commas. That is where you should first look for

errors, especially if you live in a country where commas are used as a decimal point – in GDL you must always use dot (period) as a decimal point.

- If you use a -1 masking value for the end of an XY outline, make sure the start and end points have the same XY coordinates.
- Prisms and most solids close themselves. I advise you to get into the habit of joining the first point to the last, e.g. making 9 point XY coordinates in an 8 sides object. When you insert holes or openings, or write out XY lists for other forms such as EXTRUDE, you need to close all prisms and other 3D forms by return to the starting point.

Stretchy Rafter / Joist Tool

This Rafter/Joist tool is useful in practice and the exercise demonstrates :

- Making objects stretchy
- An OBJECT becomes a TOOL
- Using a FOR NEXT Loop to convert stretchy object into a multiple stretchy object
- Popdown menus and Hide / Lock Parameters.
- Trigonometry thinking
- CUTPLANE and CUTPOLY
- Texture mapping correctly

THE ArchiCAD Toolset provides floor slabs and roof tools, but these appear too 'concrete' in sections – you want to see more constructional detail. This rafter/joist tool object is for people who want their model to display visible structure. This is a development of the BeamTool and simple JoistTool that was in an earlier exercise.

Parameters

Think about the parameters that you would want your object to offer to the user. If you want the object to be stretchy, the overall dimension is the primary consideration; the actual number of joists will be worked out by your script.

A and B are used for the plan area. We can use zzyzx for the height of the high end of the joists, but we can offer the user a choice: Pitch or Height – if the Pitch is chosen we calculate Height, and if Height is chosen, we calculate Pitch. The timber details are also entered for width, depth and spacing.

We can plan some extra 3D options e.g. you can offer to provide a final rafter/joist if the space to be filled is not an exact multiple of the joist spacing; a variable eaves overhang.

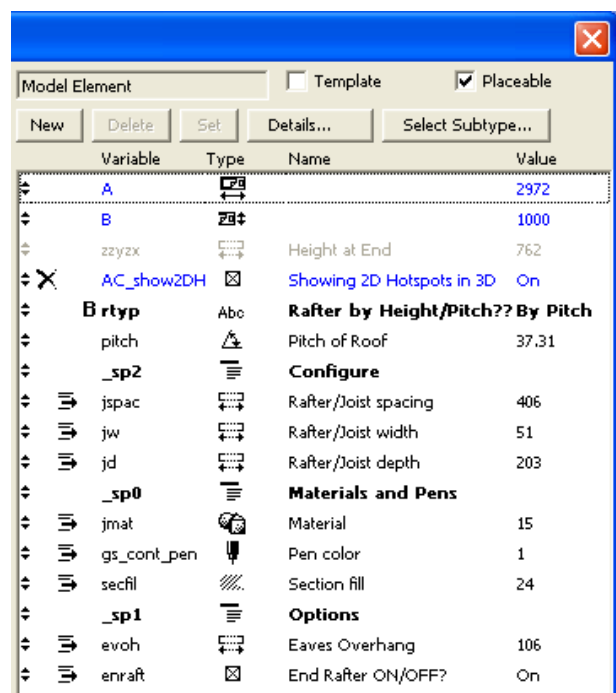
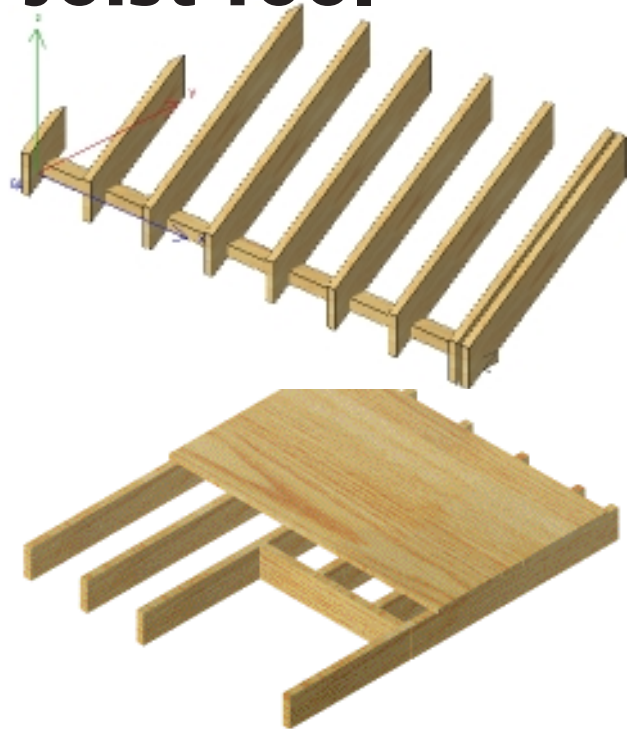
When you are further in, you can set the Subtype to Roof structure. (This gives you a load of ArchiFM parameters that you can hide later). For the moment, treat this as a simple 3D object.

3D Script

At the beginning of a new object, I tend to just knock a prototype that works before I introduce complexity and structure. So this is a first stab at a script, where you get a few timbers working, then develop it further. If we want this to be a rafter tool also, we shall have to use SLAB which is like a PRISM but it allows height coordinates at each point.

Master Script

Some 'Internal' parameters are needed. We need the actual number of joists in the object. We need to find out how much space is left after you have put in a 'round' number of joists, in case there is room to put in the final joist. This is simply 'A' minus the number of bays times the joist_spacing. Joists and rafters usually have fixed spacings because they have to work



More parameters can be added later

with plasterboard, battens and floorboards so we don't need to calculate a variable spacing (as we might do for venetian blinds). We make the ValueList here, too.

3D Script: Use a FOR... NEXT Loop

The joist we first put in can now be 'wrapped up' in a FOR-NEXT loop which spaces out the joists at the required spacing. The joist/rafter can be stored in a sub-routine. Each joist is drawn, and the cursor returns to the origin before drawing the next joist – the Loop is stepping in Distance.


```
!Rafter Joisting tool
!Master Script
rp0="By Height"
rp1="By Pitch"
VALUES 'rtyp' rp0,rp1
```

```
IF rtyp=rp0 THEN
  pitch=ATN(zzyzx/B)
  LOCK 'pitch'
ENDIF
```

```
IF rtyp=rp1 THEN
  zzyzx=B*TAN(pitch)
  LOCK 'zzyzx'
ENDIF
```

```
!Timber sizing and spacing
IF jspac<=jw THEN jspac=jw !Spacing results.
```

```
jdep=jd/COS(pitch) !Joist/rafter depth
evdrop=evoh*TAN(pitch)!Eaves overhang height
```

```
!End Rafter
numbay=INT((A-jw)/jspac)
lenj=(numbay*jspac+jw)
IF A-lenj<=jw THEN enraft=0
PARAMETERS jspac=jspac,pitch=pitch,
            zzyzx=zzyzx
```

At the end of all this, we display the results in the parameter table.

```
!Rafter Joisting tool
!3D Script
PEN gs_cont_pen
MATERIAL jmat
SECT_FILL secfil,gs_cont_pen,91,gs_cont_pen
```

```
!--Loop rafters by distance
FOR dist=0 TO A STEP jspac
  ADDx dist
  GOSUB 100:!rafters
  DEL 1
  NEXT dist
```

```
!End rafter
IF enraft THEN
  ADDx A-jw
  GOSUB 100:!End rafter
  DEL 1
ENDIF
```

```
END!
```

```
100:!rafters
LIN_ -jw,0,0, jw*2,0,0
SLAB 5,jdep,
  0, 0-evoh, -evdrop,
  jw, 0-evoh, -evdrop,
  jw, B, zzyzx,
  0, B, zzyzx,
  0, 0-evoh, -evdrop
HOTSPOT 0,0,0
HOTSPOT 0,B,hit
ROTz -90
ROTy pitch
GOSUB 999:!Texture
DEL 2
RETURN
```

```
999:!Texture Horizontal
BASE
VERT 0,0,0
VERT 0.1,0,0
VERT 0,0.1,0
VERT 0,0,0.1
COORD 256+2,-1,-2,-3,-4
BODY -1
RETURN
```

Build the Popdown menu: do it in Master Script so that rp0 and rp1 can be used in other scripts.

Depending on the result we use a bit of Trigonometry to work out the Pitch, and then lock the Pitch parameter... or we work out the Height and then lock the Height parameter.

We lock these so that the user can see what they are going to get, but cannot be confused into thinking they can edit the calculated results.

These calculations are for the Eaves overhang and the End rafter. If there is not enough space at the end, then the end rafter will not be built.

After the Title and Date, we start with Pen. There is one material in the whole object so it can go here.

We would like a nice fill pattern if the tool is cut in section: SECT_FILL provides this using the current pen & white pen 91.

The Loop is based on Distance and we make the timber as a subroutine to keep things tidy.

The End rafter is added if the user asked for it. It goes to the end 'A' and steps back one rafter thickness, to fit.

For the Timber subroutine, we assume that the zero,zero point is where the timber touches the wall. The Eaves overhang is in the negative Y zone, and its Z value is 'evdrop' downwards.

SLAB is very like PRISM with the Z value added. The LIN_ command draws a 3D line which helps the user visually to attach the tool to a wall.

We now use the Texture mapping trick that was explained in the Timber chair earlier to paint the Texture along the pitch of the Rafter/ Joist. Make it GOSUB 999 and you won't forget it.

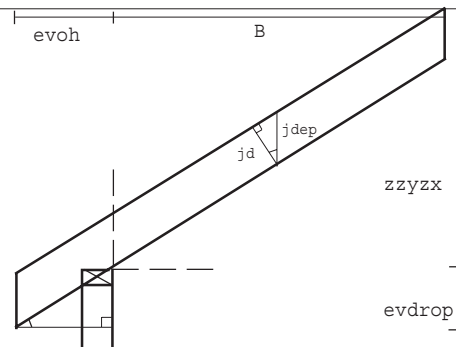


Diagram showing how the 'jdep' and 'evdrop' parameters are calculated using TAN and COS of Pitch.

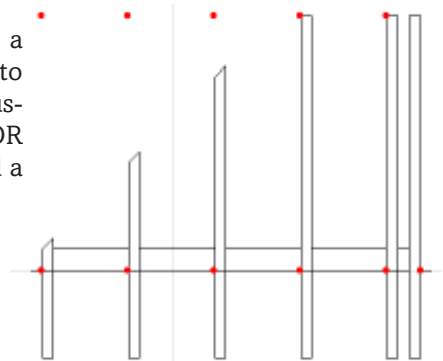
It is useful!

With this tool you can assemble quite complex groupings, and the RafterJoistTool can also be used for floorboarding – in fact for any parallel, linear assemblies. Single joists, such as the trimmer in the floor example can be created simply by squeezing the Tool to the thickness of a single joist.

If you used the tool for Studwork you would get incorrect texture mapping, but now you know how, you could make a similar tool that stood the timbers upright with the correct upwards texture.

2D Script

This follows a similar pattern to the 3D Script, using the same FOR NEXT loop and a subroutine 100.



```
!Rafter Joisting tool
!2D Script
PEN gs_cont_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
!PROJECT2 3,270,2
FOR dist=0 TO A STEP jspac
  GOSUB 100:!rafters
  GOSUB 110:!hotspots
  NEXT dist
IF enraft THEN !End Rafter
dist=A
ADD2 -jw,0
GOSUB 100:!rafters
GOSUB 110:!hotspots
DEL 1
ENDIF
```

```
END:-----
```

```
100:!rafter
RECT2 dist+0,-evoh,
      dist+jw,B
RETURN
```

```
110:!hotspots
HOTSPOT2 dist,0
HOTSPOT2 dist,B
LINE2 dist-jw/2,0,dist+jw*3/2,the wall line is to be.
RETURN
```

Start with Title & Pen. Stretchy Hotspots must come early, for the Width 'A'.

use PROJECT2 as an overlay to check the accuracy of your scripted loop.

We use a similar FOR NEXT loop and the Hotspots for each rafter's zero and 'B' values are included with each rafter.

Instead of using ADD2, we can use 'dist' directly. So we need a little dist=A trick for the end rafter.

The rafter itself is a simple transparent RECT2 (but with a POLY2 we could provide a fill pattern).

The LIN_ from the 3D is provided here as a LINE2 so that the user knows where

Extend the Rafter / Joist Tool

More 3D Options

NOW you have got the Tool working, you could add more 3D options:

- Cutplanes to improve the eaves end detail.
- Mitring at the end for hip/valley conditions
- Birdsmouthing and wall plate for the wall line.
- To make it better still, you could add graphical hotspots for the eaves overhang and spacing.

Eaves end cut

Starting with this, we need a Cutplane hovering just under the Rafter/Joist Tool that can trim off the sharp lower corner of the eaves. The eaves cut is always working so if you don't want it, you can lower it a great distance so that it cuts nothing.

Hip Mitre

We need a vertical Cutplane that can allow user to form Jack rafters around a corner. Use a CIRCLE again to check that your rotations work.

Wallplate and Birdsmouth

These two follow the same line, one is solid, the other is a cutpoly through the rafters. The purpose in using both would be to get a nice inkle at the joint, in 3D.

Alterations to the Scripts

Cutplanes and Polys are always added before building the 3D, so insert these extra routines as shown here. Finish with the CUTENDS. For the 2D, it would be too complex to work out the 2D Script for a mitred raftertool, so reform the 2D into an IF...ENDIF routine. The Subroutines are unchanged.

```
!Rafter Joisting tool
!2D Script
PEN gs_cont_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
IF hipm OR walp THEN
  PROJECT2 3,270,2
  FOR dist=0 TO A STEP jspac
    GOSUB 110:!hotspots
  NEXT dist
ELSE
  FOR dist=0 TO A STEP jspac
    GOSUB 100:!rafters
    GOSUB 110:!hotspots
  NEXT dist
IF enraft THEN !End Rafter
dist=A
ADD2 -jw,0
GOSUB 100:!rafters
GOSUB 110:!hotspots
DEL 1
ENDIF
ENDIF
END:!
```

If either the wallplate or hip mitring are chose, we now converet this to an IF statement, providing an option for a PROJECT2 and a small loop to draw the Hotspots for the rafters where they meet the wall.

	_sp1		Options	
+	evoh		Eaves Overhang	106
+	enraft	<input checked="" type="checkbox"/>	End Rafter ON/OFF?	On
+	eavcut		Eaves Cut below Wallplate	254
+	hipm	<input checked="" type="checkbox"/>	Hip Mitre ON./OFF??	On
+	mitr		Mitring angle	45.00
+	mitrx		Mitring offset	-100
+	walp	<input checked="" type="checkbox"/>	Wall plate ON/OFF?	On
+	birds	<input checked="" type="checkbox"/>	Birdsmouth	Off
+	walpw		Wall plate Width	100
+	walph		Wall Plate height	75

We can add these all at once, or a few at a time. We can assume that the birdsmouth will have the same dimensions as the Wallplate.

```
!Rafter Joisting tool
!3D Script
PEN gs_cont_pen
MATERIAL jmat
SECT_FILL secfil,gs_cont_pen,91,gs_cont_pen

!Cutting routines
ADDz -eavcut
CUTPLANE 180
!CIRCLE 1
DEL 1
IF hipm THEN
  ADDx mitrx
  ROTz -mitr
  ROTy -90
  CUTPLANE
!CIRCLE 1
DEL 3
ENDIF

!Wallplate and Birdsmouth
ROTy 90
IF walp THEN
  PRISM 5,A,
  0,0,
  0,walpw,
  -walph,walpw,
  -walph,0,
  0,0
ENDIF
IF birds THEN
  CUTPOLY 5,
  0,0,
  0,walpw,
  -walph,walpw,
  -walph,0,
  0,0
ENDIF
DEL 1
GOSUB 999:!Texture
```

As Cutplane cuts away everything above it, we need to flip the Cutplane over with a 180° turn. Lower it by 'eavcut'.

With the CIRCLE command, you can temporarily see where the 'bacon slicer' is sitting.

See nearby page for explanation of CUTPLANE.

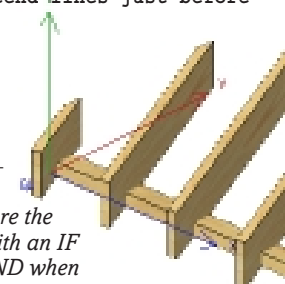
The PRISM and the CUTPOLY can both be build pointing upwards and then bent over with a ROTy to their final positions.

With Graphical Hotspots you could provide independent end offset dimensions for the wall plate. At the moment, it starts from 0,0. We also need GHs because hotspots based on A and B are annoying – they stretch too freely in all directions. GHs can be strictly controlled.

Use the texture subroutine to make sure that the woodgrain is correct.

```
!-----Loop rafters by distance-----
!Repeat everything that is already here
!and add the following cutend lines just before
!the end
CUTEND !Eaves cutting
IF hipm THEN CUTEND
IF birds THEN CUTEND
END!
```

Finally, add the CUTENDs before the END. They must be protected with an IF statement – if you try to CUTEND when none is needed you will get an error.



Cutplane/Cutend

CUTPLANE is one of my favourite commands. It is much easier to make something larger than it needs to be and cut it down to size, than to build it up positively. The more complicated the object, the more useful Cutplane becomes. Its greatest use comes when you need to form mitres at the end of extrusions. Without Cutplane, it could not be done.

I mostly use and teach the Angle method. The other method, Vectors, is more difficult to use. X,Y,Z describe a point, making a vector from 0,0,0 for the direction the Cutplane points to. The 'side' values are:

0 for normal cutting, throw away stuff above the plane, and,

1 to flip the Cutplane and throw away the stuff below.

CUTPLANE can be likened to a large spinning bacon slicer. You move the slicer into position, rotate the blade to the right angle, issue the CUTPLANE command; now you must use DEL to get back to the origin. Next, build the model that needs cutting. If your model disappears, you may have forgotten to DEL correctly.

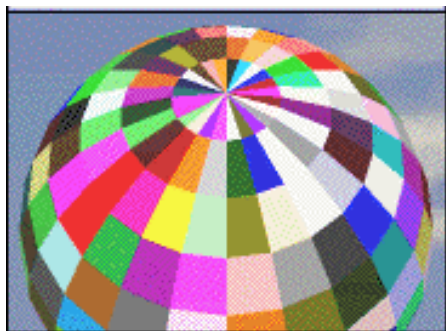
CUTPLANE always cuts away what is above it, and leaves behind what is below. Sometimes, you wish to do the opposite, and cut away everything below the ground plane. For this, it is handy to spin the blade through 180 degrees with a 'CUTPLANE 180' command to leave behind everything above the plane.

CUTPLANE needs to know the Material that you are using. If you do not issue a Material command before Cutplane, it will leave the cut face to the current PEN colour or an earlier stated material, not to the material of the cut object.

DEL is essential. After you write the Cutplane command you have to get back to where you want to start building your part of the 3D model.

Multi coloured models

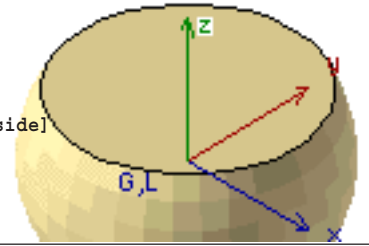
You could issue a Material command, issue a Cutplane, then write another Material statement, issue another Cutplane... and so on. Then build the object to be cut. In this way, you can get multi coloured models. Cutplane uses a lot of memory, and if you issue a large number of them, the model rendering will slow down, or you may even be told that ArchiCAD is out of memory.



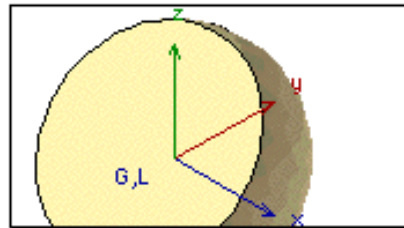
!CUTPLANE Demonstration
!Syntax: - CUTPLANE angle

```
ADDz 0.5
CUTPLANE
DEL 1
SPHERE 1
CUTEND
```

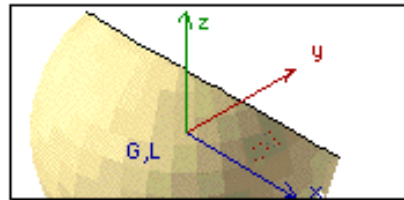
There is also:
CUTPLANE x,y,z [,side]



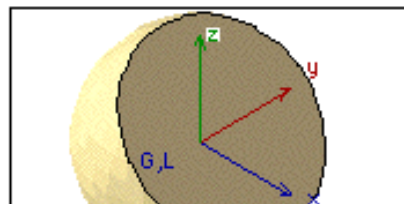
If an angle is included, the Cutplane rotates around the X-Axis.



```
ADDz 0.5
CUTPLANE 45
DEL 1
SPHERE 1
CUTEND
```



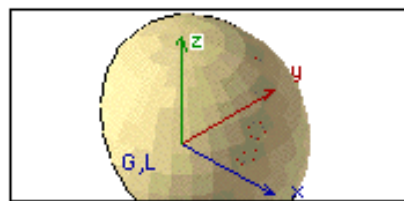
```
ADDz 0.5
CUTPLANE -45
DEL 1
SPHERE 1
CUTEND
```



```
ADDz 0.5
ROTy 45
CUTPLANE
DEL 2
SPHERE 1
CUTEND
```

For other angles (i.e. not around X axis), you do the swivelling yourself with ROT commands and issue a CUTPLANE.

Remember to do the correct number of DELs before building the object that is to be cut.



```
ADDz 0.5
ROTy 45
CUTPLANE 180
DEL 2
SPHERE 1
CUTEND
```

CUTEND At the end of the cutting process, the **CUTEND** command is required to stop the blade spinning. If CUTEND is not stated, the Cutplane command will continue till the end of the object and you will get an Error report.

Some models use many Cuts, and for EVERY Cutplane, you must have the right number of CUTENDs. It is annoying to GDL user that there is not some sort of CUT TOP command that has a similar function to DEL TOP – i.e. it would stop all cutting, and if there were no cutplanes in action it would not result in an error.

More Cutting commands

CUTPOLY operates like a cookie cutter through the model. You define the profile of CUTPOLY with similar syntax to a common PRISM. It grows up from the X-Y plane. The direction of the cut can be modified by manoeuvring it into position (like I advise for CUTPLANE, and just like you would do if you were trying to manoeuvre a PRISM to the same location); or it can be defined with a final X,Y,Z statement at the end of the CUTPOLY statement which defines a vector (from 0,0,0).

Prior to AC8, the limitation was that the profiles had to be CONVEX. Concave or more complex shapes were built from a succession of convex CUTPOLY statements. Now you can use complex shapes.

A helpful idea is to build a PRISM in exactly the same place as you wish to have a CUTPOLY, and when it is in the right place and working, change it to a CUTPOLY.

Remember that CUTPOLY cuts for an infinite distance in both directions, until you issue a CUTEND.

CUTPOLYA: If you want holes with Polylines enabled, you can use a variation called **CUTPOLYA** which is analogous to PRISM underscore. **Masking and Polylines:** Every XY location is followed by a masking value, usually 15. The masking values also permit the use of Polylines, curved edges and round holes.

Status: You can decide if the cut faces adopt the existing material setting or the material of the object being cut – important if cutting through several layers of different material. '1' adopts the material of the cut object, '2' uses an aforementioned material statement.

Depth: A big benefit is that you are able to point CUTPOLYA in a certain direction and fire it off from a defined starting point like a gun, but for an infinite distance in that direction. The Depth is the starting point. In earlier AC, you had to start outside an object. With AC8.x you can start inside as in this example.

Vector: Cutpoly and CutpolyA can also accept an extra XYZ coordinate set at the end of the statement to indicate a vector, and with this you can skew the direction of the cutter.

CUTSHAPE: This is now archaic since CUTPOLY does the job better. CUTSHAPE made a block shaped cutpoly with an infinite z-height and y-depth, you only have to specify the location and x-width. CUTSHAPE does not work with AC 8.1, it just cuts an infinite plane, like CUTPLANE.

CUTFORM: This is a powerful cutting capability that extends CUTPOLYA to include Pyramidal and Wedge shaped 3D cutting. This is advanced and deserves a whole section later in the book.

!CUTPOLY demonstration

PEN 1

MATERIAL 18

RESOL 16

We are just using standard pen and material here.

!Cutpoly Demo

!Syntax: CUTPOLY number,

! x1,y1, x2,y2,...xn,yn

! (define a poly cut plane)

! [,xv,yv,zv] Optional

! Defines the vector direction of cutting.

ADDz 2

ROTx 90

CUTPOLY 4,

-2, 0,

2, 0,

2, 2,

-2, 2

DEL 2

Cutpoly can be lifted, rotated and pointed just like a prism.

!Cutpolya Demo

!Syntax: CUTPOLYA number,status,depth,

! x1,y1,15, etc.

! [,xv,yv,zv] Optional Vector

! Status - 1 body's polygons, 2 normal

CUTPOLYA 5,1,1,

-4, 0, 15,

0, -4, 15,

4, 0, 15,

0, 4, 15,

-4, 0, -1,

0, 0, 1 !Vector

In this case the vector is directly upwards, but you can have fun changing the values of this vector XYZ. The cutpoly is using the material of the body, not the whitewash.

!Now for the object!

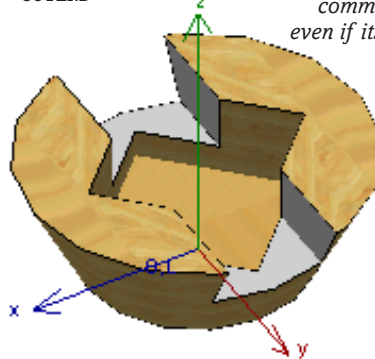
MATERIAL 15

CONE 4,4,6,90,90

CUTEND

CUTEND

The Object to be cut is always made after the cutting commands have been issued. Remember to issue the CUTEND commands after the cutting, even if its the end of the script.



SOLID Geometry Commands allow you to add and subtract and intersect objects from each other. They provide vast opportunities for more realistic modelling in 3D GDL. In many cases, SGC replace the drilling of holes in Prisms or the use of Cutpoly/Cutplane.

!Show the Blade!

ADDx A/2

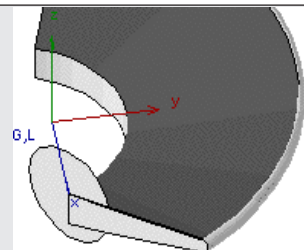
ROTz 90-angl+ABS(mitr)

ROTx 90

!circle 0.5

CUTPLANE

DEL 3



Show the Blade!

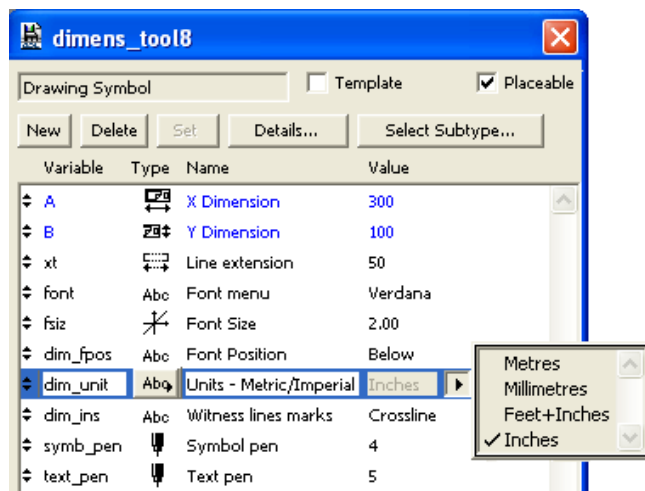
CUTPLANE is most easily done with the 'angle method' – manoeuvre the imaginary cutting blade into position, then issue the command. Sometimes, you get very confused with the pluses and minuses of the angles. **Put an ACTUAL cutting disc at the same place so you can see exactly which way it is facing.** Then comment it out when it is working.

Steps in 2D

Dimension Tool

YOU might think the GDL Cookbook was only about 3D modelling – not so! 2D scripting can be useful for practical work.

This dimension tool is a stretchy dimension line with two witness lines (stretch it, or type in the dimension) and it allows Metric and Imperial, and a choice of tick marks. The Subtype is 'Drawing Symbol'. It can be used after a survey.



Parameters

The Font size in millimetres is based *absolutely* on the plotting or printing size, whatever the scale of the drawing. Later in the Cookbook you can see how to adapt this script to make autosizing fonts.

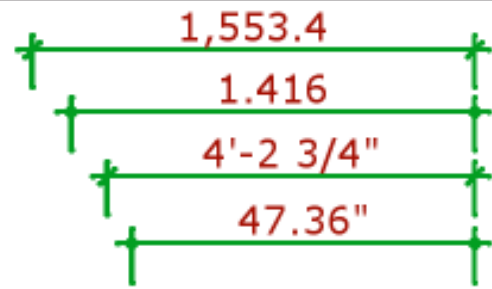
Here, we make popdown menus for the font, the font units, text position, and the witness line marks at the intersections. These have to be converted to flags. We offer a choice of Pens.

2D Script

The Hotspots are key to making it stretchy. Put them in first. Using A and B ensures that both the dimension and witness lines are stretchy.

Finally, for the TEXT2 display, it is *always* necessary to Define the Style of the text. Having done that, if you wonder why it doesn't work, it's because you have to remember to Set the Style to your defined style.

TEXT2 wants to print a string, so we convert the value of A to a string. STR(...) forces TEXT2 to display a number (signified by the '%' sign) as a string expressed in Metres, Millimetres, Feet+Inches or Decimal Inches, and the 0.8 forces it to a precision of 1/8", 0.3 to 3 decimal places and so on. The IF... statements decide on the alternative STR formats for the dimensions. Other formats are available in the GDL manual if you want to add them to this tool.



!Master Script

```
dun0='Metres'
dun1='Millimetres'
dun2='Feet+Inches'
dun3='Inches'
VALUES 'dim_unit' dun0,dun1,dun2,dun3
VALUES 'dim_fpos' 'Above','Centre','Below'
IF dim_fpos='Above' THEN fp=8
IF dim_fpos='Centre' THEN fp=5
IF dim_fpos='Below' THEN fp=2
ch0='Circle'
ch1='Crossline'
VALUES 'dim_ins' ch0,ch1
VALUES 'font' 'Times New Roman Western',
'Verdana','Arial','Courier'
```

The results of the popdown menu are converted to flags.

These position numbers are as on the telephone keypad

!Dimensioning Tool

!2D Script

!Hotspots - Stretch & Pickup

```
PEN symb_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A/2,0
HOTSPOT2 0,-B
HOTSPOT2 A,-B
```

Put the Hotspots first.

!Dim_Line & Witness Lines

```
LINE2 -xt,0, A+xt,0 !Dim_line
HOTLINE2 -xt,0, A+xt,0 !AC9
LINE2 0,-B, 0,xt
LINE2 A,-B, A,xt
```

The dimension lines and witness lines are stretchy using A.

!Dimension markers

```
IF dim_ins=ch0 THEN
LINE2 -xt,0,xt/4
CIRCLE2 0,0,xt/4
CIRCLE2 A,0,xt/4
ELSE
LINE2 -xt/2,-xt/2, xt/2, xt/2
LINE2 A-xt/2,-xt/2, A+xt/2, xt/2
ENDIF
```

We use the extension line length to calculate the size of the crosshairs and circles.

!Text

```
PEN text_pen
DEFINE STYLE "diasty" font,fsiz,fp,0
SET STYLE "diasty"
IF dim_unit=dun0 THEN string=STR('%0.3 m',A)
IF dim_unit=dun1 THEN string=STR('%0.1 mm',A)
IF dim_unit=dun2 THEN string=STR('%0.8 ffi',A)
IF dim_unit=dun3 THEN string=STR('%0.2 di',A)
TEXT2 A/2,0,string
```

STR() formats the dimension.

DEFINE STYLE

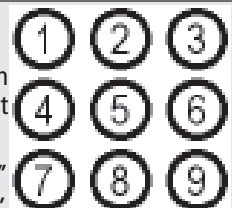
Before we write anything with TEXT2, we need to define style: font etc. The syntax is:

```
DEFINE STYLE "name of style"
font, fontsize_in_mm, position,
style.
```

Style is: 0=normal, 1=bold, 2=italic, 3=bold italic.

Position is as on the numbers on the keypad of a mobile phone. The text 'grows' from that number, so 8 means grow 'upwards', 5 means 'centre'.

AC9 vastly extends DEFINE STYLE, allowing rich text ... varying colour, size, style within sentences.



Polygons in 2D

POLY2

POLY2 simply draws a polygon – a sequence of lines. It can be filled or empty. You list out the XY coordinates just like you do with PRISM. The framefill 'ffill' codes from 0 to 7 determine the visual quality.

POLY2_

POLY2_ (underscore) is an enhancement of POLY2 – you can control the line drawing of each part of the polygon; you can use 'polylines' (curved elements) and drill holes. POLY2 and POLY2_ can be filled with a fill pattern. The polygon must be *closed* for this to work. For any poly to be filled, you should close the form by specifying the last point the same as the first point. The Framefill attribute is normally left as '1' for unfilled Polygon drawing. However, you can vary this with 7 permutations. If you wish to control Pen Colours of the Fill, then you must use POLY2_A and POLY2_B.

POLY2_A & POLY2_B

Good GDL will always use POLY2_B instead of the weaker versions because it gives the user full control over appearance and opacity. You need additional parameters for the Fill pen and the Background Fill pen – if you use Subtype definitions, GDL will provide these as 'gs_fill_pen' and 'gs_back_pen'.

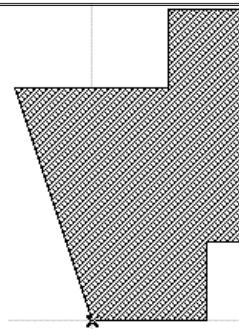
Northpoint

THE NORTHPOINT uses a filled POLY2, and LINE2 to form the cross shape. By using A and B as the determinants of size it is completely stretchy. Hotspots are also positioned at key positions. The Fill pattern and Pen colours are settable by the user.

The script shows how to set a text style in a 2D Script, and how to control font and fontsize. Font height is based on millimetres (1/25th") in the plotted or printed output, regardless of the scale of the drawing. The US version uses Points (1/72th"), so a quick formula allows the user to set the font size in Points.

Drawing Symbol		Template	Placeable
New	Delete	Select Subtype...	
Variable	Type	Name	Value
A		X Dimension	800
B		Y Dimension	1000
gs_symb_pen		Symbol Pen	1
northfil		Fill pattern	24
northpoint_s..		Label	
font	Abc	Font	Times
gs_text_pen		Text Pen	5
funit	Abc	Font Units	Points
fsiz		Font Size	5.00

Note, the SubType is set to **Drawing Symbol**



What is Framefill?

0 means draw nothing, 1 draws the line, 2 draws the fill pattern but no line, 4 means close the Polygon (if not already closed). Any additive combination of these can be used e.g. 3 is the sum of 1 and 2 so you get the lines and the fill pattern, 6 [2+4] results in a closed filled polygon with no line!, 7 [1+2+4] gives you everything.

```
!Syntax POLY2 n, ffill, !Syntax POLY2_ n, ffill,
!x1,y1, ..... xn,yn !x1,y1,s, ..... xn,yn,s
PEN 1 PEN 1
SET FILL 'Earth' SET FILL 'Earth'
POLY2 9,3, POLY2_ 9,7,
  0.00,0.00, 0.00,0.00,1,
 -0.10,0.30, -0.10,0.30,1,
  0.10,0.30, 0.10,0.30,1,
  0.10,0.40, 0.10,0.40,1,
  0.20,0.40, 0.20,0.40,1,
  0.20,0.10, 0.20,0.10,1,
  0.15,0.10, 0.15,0.10,1,
  0.15,0.00, 0.15,0.00,1,
  0.00,0.00 0.00,0.00,-1
```

The 1 status code tells GDL to draw the line to the next point, zero will omit it.

POLY2_B {2}

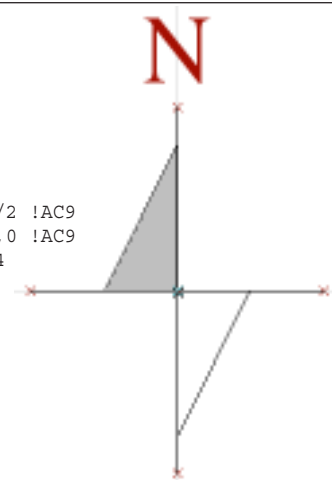
This is the final development of the POLY2 and this includes three additional parameters for XY and angle of the origin of the vectorial fill pattern.

POLY2 and its variants are VERY important in AC9 because objects with a fill can be easily selected.

!North Arrow

!3D Script

```
PEN gs_symb_pen
!Ideal size A=3 B=6
LINE2 0,-B/2, 0,B/2
LINE2 -A/2,0, A/2,0
HOTLINE2 0,-B/2, 0,B/2 !AC9
HOTLINE2 -A/2,0, A/2,0 !AC9
LINE2 A/4,0, 0,-B*0.4
SET FILL northfil
POLY2_ 4,3,
  0,0,1,
  0,B/*0.4,1,
  -A/4,0,1,
  0,0,1
PEN gs_text_pen
IF funit='Millimetres' THEN fs=fsiz
IF funit='Points' THEN fs=fsiz*25.4/72
DEFINE STYLE "north" font, fs,8,0
SET STYLE "north"
TEXT2 0,B/2,"N"
```



This is the formula for Points

!Hotspots

```
HOTSPOT2 0,0
HOTSPOT2 A/2,0
HOTSPOT2 -A/2,0
HOTSPOT2 0,B/2
HOTSPOT2 0,-B/2
```

!Parameter Script

```
VALUES 'funit' 'Millimetres',
        'Points'
VALUES 'font' 'Times','Verdana',
        'Arial','Courier'
```

!Values Lists can be in Parameter or Master Script

Tips and Tricks for 2D

Font size in Millimetres or Points?

Some people prefer to think of text in Points, not mm. Points are 72ths of an inch and go right back to the days of Caxton. If you wish your STYLE to be defined in points, ask the use for points as an Integer Parameter 'fsiz', then adapt your DEFINE STYLE statement above thus:

```
DEFINE STYLE 'mysty' 'Arial', fsiz*25.4/72,5,0
```

...in which you apply the 25.4/72 multiplier to convert points to millimetres.

Real World Autosizing text in 2D

When you 'Define Style' for text, the font height is usually a fixed height that will appear on the printed or plotted sheet at that size, regardless of scale.

More often, you may want the font to appear constant relative to the object that it is part of: so that it is working in real world dimensions, for example it is based on a wall thickness or tube diameter. In the DEFINE STYLE statement, Font size has to be in millimetres. The Object size comes to GDL in metres (even in the non-metric environment). So multiply the object dimension by 1000. Then divide the resulting size by the Global Variable for the scale of the drawing 'A_' or 'GLOB_SCALE'. This finished size will stick faithfully to the size of the object it relates to, whatever the drawing scale.

```
DEFINE STYLE 'mysty' 'Arial', fhht*1000/A_,5,0
```

See elsewhere in the Cookbook for more discussion of Global Variables.

2D Crosshairs

If stretchiness is a big feature of your object, the A.B points may not actually be within the object boundary. To make them visible to the user, plant a **2D crosshair** at the A,B points. e.g. for point A,0, write in the 2D Script:

```
LINE2 A-0.02,0, A+0.02,0
LINE2 0,A-0.02, 0,A+0.02
```

Here the crosshair size is 20mm (3/4"). Don't forget that it is better to declare all your stretchy Hotspots before you issue the Project2 or the rest of the 2D Script.

Standard Hotspots give you round spots and AC 8.1 gives you distinctive diamond spots where they are correctly written Graphical Hotspots designed to be stretchy.

ROT2 bad for your DWGs?

Objects in drawings exported from AC to DWG often behave unexpectedly in the 2D symbol – because ROT2 may cause problems. If you can use Trigonometry, you can often avoid the use of ROT2. Unfortunately it is more difficult to think out. Use COS and SIN*radius for X and Y coordinates.

```
FOR ang=0 TO fan+0.001 STEP stang
  LINE2 0,0,lrad*COS(ang),lrad*SIN(ang)
  ARC2 0,0,arcd1,ang-4,ang+4
  ARC2 0,0,arcd2,ang-3,ang+3
  HOTSPOT2 arcd1*COS(ang),arcd1*SIN(ang)
  HOTSPOT2 arcd2*COS(ang),arcd2*SIN(ang)
NEXT ang
```

See the **Maths primer** for more on Trigonometry.

Cursor control in 2D Scripting

ADD2, ROT2 and MUL2 are the Cursor movement commands in 2D Scripting. ADD2 and MUL2 always have to be specified with X and Y in the same command. e.g. ADD2 1.50,0 will do the equivalent of an ADDx in 3D.

Unfortunately, you do not see a visible cursor as you do in 3D. One trick is to use the CIRCLE2 0,0,0.01 command as a substitute cursor and keep pushing it along just in front of the cursor movement command. e.g.

```
ADD2 1.50,0
CIRCLE2 0,0,0.01
```

Delete the circle when you have finished working, or use it as a hotspot location identifier like the crosshair above.

Hot Tip for rapid 2D Scripting

Leave PROJECT2 in action until you are sure that your script works perfectly. In fact leave it there permanently, with a comment mark to disable it. This is in case you need to edit or check the 2D Script.

• ***In fact, you can copy your 3D Script and paste it into the 2D Script window. Then convert it to 2D.***

Change the ADD x, y, z commands appropriately, to ADD2 x, y (throw away the Z value). ADDx and ADDy can be changed to ADD2. MUL can be changed to MUL2. Moves in the Z direction can be deleted if they will have no effect on the plan view. DELs will have to be recalculated.

As there is no height in 2D, you can change a BLOCK command to a RECT2 command:- for example, BLOCK jwid,jlen,jdep becomes RECT2 0,0,jwid,jlen.

This is a very good technique when there are Loops and subroutines. Loops can use the same counters, and you use DEL to cancel Cursor movements just like you can do in 3D Scripts. Subroutines can retain the same numbers. This is most helpful for object maintenance.

Make a 2D symbol for a non stretchy object

You could do a PROJECT2, place the chair on the plan, Explode it, copy the lines and arcs, and paste into the 2D Symbol window of the chair. To be sure of it working, type FRAGMENT2 ALL,1 into the 2D Script. This is because objects which have had a script and later have a symbol seem to steadfastly ignore this fact. 'Fragment' forces them to pay attention.

DRAWINDEX

DRAWINDEX defines the order in which 2D elements are drawn. This is important if you use opaque objects, so you want, for example, one polygon to be above another polygon, not hidden. You can have just 5 strata of 2D Scripting, DRAWINDEX 10, 20, 30, 40 and 50. Place a Drawindex command in a script and everything after that is in that stratum. The lowest numbered strata get drawn first, so higher numbers will appear ON TOP of the lower numbers.

The normal drawing order without Drawindex is Figures (bitmaps) lowest, then Fills, then Lines, and finally Text on top.

Graphical Hotspots!

GRAPHICAL Hotspots are the most exciting innovation to arrive with ArchiCAD 8. They come early in the GDL Cookbook as they transform the experience of GDL object making from a task into a delight. It may seem early in the book to be tackling these, but the demand for knowledge about Graphical Hotspots is huge. Let's get on with it!

Stretchiness through the ages of GDL

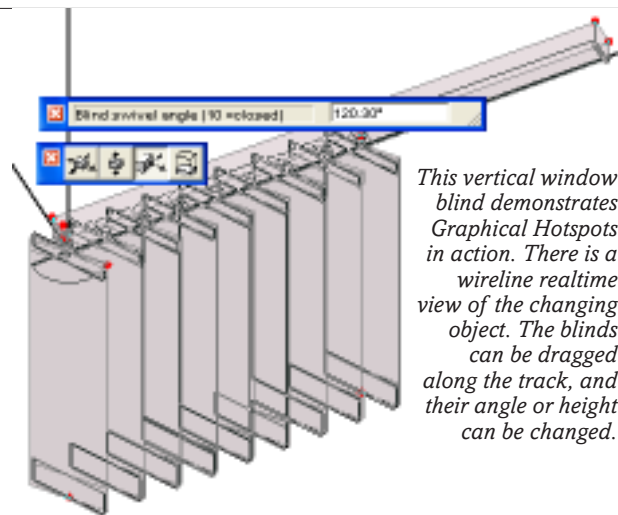
Objects can have Hotspots in their 2D symbol and 3D form which enable them to be selected, moved or stretched. Since the early days of GDL, objects were either fixed in size, or they had stretchiness within a theoretical rectangle based on A and B in the 2D symbol. Autoscripted objects are still stretchy by virtue of a routine using MUL (to shrink or stretch them to the A or B size), but this merely distorts them – such objects are not truly parametric. In recent times, the rectangle was given a height of 'zzyzx' and became a cuboid. Considerable ingenuity has been extended over the years to make sophisticated objects using just A, B and zzyzx for stretching in the 2D or 3D. The holy grail of user friendliness is to extend the tangibility of objects by making hotspots more powerful.

Complex options need a User Interface

If objects are written parametrically, they can have many complex options like swinging doors and drawers whose position can be defined as a percentage or a dimension in the parameters table. These can become extremely tedious to fill in, and are in some cases too complex for users to understand all the interactions. As these are not part of the A/B/zzyzx cuboid, the user cannot be assisted by conventional hotspots.

The User Interface that came with ArchiCAD 6.5 was a major improvement. In the first place, the normal parameter table could be organised into cascading groups such as materials, options, 2D representation. The table could be organised with bold font, separators and titles. Furthermore a new scripting solution was introduced to enable the creation of a graphical User Interface (UI) which was an even greater benefit. The UI can contain many pages, include explanations to the user, include illustrations and pictorial menus. This is good, but you still need to select the object, plough through pages of UI, read all the instructions, fill in the parameters – it still tedious.

At the human level, the very first level of user interface is the object itself as seen in the 2D and 3D windows. Can you touch and manipulate the objects intuitively? This is what we all want to do!



Graphical Hotspots to the rescue!

ArchiCAD 8's Graphical Hotspots (GHs) provide the means to **Push / Pull** and to **Rotate**. With ingenious use of these 2 capabilities in 2D and 3D, it is possible to make GDL objects work like a mechanism; one that you can explore by grasping parts with the mouse pointer, opening doors, sliding open covers, turning levers, pulling drawers open. Once you have grasped the technique it can become addictive, you make one part of the object manipulable, and before you know it, you are making all parts of it manipulable.

It's fair to say that the old fashioned A/B/zzyzx cuboid has had its day. An A, B or zzyzx dimension are always more tidily and accurately handled by the GHs engine, with a visible readout of the dimension achieved, an editable infield to enter accurate dimensions, and best of all, a real-time wireline representation of the object as it moves or rotates.

The user needs to get the hang of the Pet Palette in AC8.x because if the wrong icon is clicked, the hotspots are not accessible.

Graphical Hotspot definitions

The original syntax for a hotspot is HOTSPOT2 x,y in 2D or HOTSPOT x,y,z in 3D. In AC65, we noticed that they could be HOTSPOT2 x,y[,hsid] which is an optional numerically unique hotspot identity, but there was then no function for the ID. In AC8 and AC9, we now see what this ID is for. The internal database of ArchiCAD needs to know what these objects are doing, matching them up to the parameter they relate to and to the function of the individual spot. A hotspot also carries the name of the parameter it is trying to edit, and a numerical code for the function, e.g. push/pulling, or rotating. A GH can even carry a reference to another parameter which it can display while you work the hotspot, e.g. by changing the length of a rod, you can be changing its volume.

Do not worry that your object will suddenly become populated with many spurious hotspots. Many of the new ones can be defined but then hidden from view so that only the ones you want to see and grasp will be visible – as small diamonds.

Graphical Hotspots! the Syntax



The Pet palette for
2D use of GHs



The Pet palette for 3D
use of GHs

Learning Graphical Hotspots (GHs)

FIRST attempt at GHs seems a bit painful and they are somewhat verbose. It takes 6 lines of code to write a single linear hotspot, and up to 10 lines of code to write an angle hotspot. The first one takes the longest, but the remainder are easily created with Copy and Paste. Practice makes perfect very quickly. Soon you will be able to churn them out without access to the Manual or Cookbook!

GHs lend themselves well to use in subroutines. With Parameter Arrays (which come later in the book) you can write incredible numbers of hotspots with just a few lines – 100 or more individual linear hotspots could be created with just 6 lines in a subroutine. With Parameter Arrays, every rafter in the Rafter/Joist Tool could have individual lengths, pitch and eaves overhangs with just three GHs subroutines.

Write them in 2D or 3D?

Do you write GHs in 2D, 3D or in both? The answer is that you should take an intelligently economical view. If the movements are in a plane visible from above, GHs are always best in 2D – faster to write, faster acting for the user, easier to understand. For many of the plan view spots, it's possible to copy the code to the 3D Script, modify them (retain the same subroutine numbers), and you have both.

For true 3D motion, e.g. the pushing and pulling of drawers stacked above each other, or the rotation of a lever, then 3D GHs must be written.

A round of applause for the team in HU

When I work with GDL I am always grateful to Graphisoft for providing this wonderful language that is so creative and practical. But when I have made an object with many GHs and see it working, I positively glow with admiration for those amazingly clever guys and girls who have predicted our needs so accurately and have written the underlying engine for this code. In some ways, GHs create a new middleground between GDL and API in smartness.

'Length' GHs – Push / Pull

GHs have to be written as a group. It takes three to define a single length-based hotspot. GDL must know the Base point from where the action starts, flag **1=Base**; the actual Moving visible point, flag **2=Move**; and a Vector for the direction of pushing or pulling, flag **3=Vector**.

The hotspot ID does not really mind what it is as long as it is an integer and not duplicated anywhere in the current script (you can use duplicate numbers in the 2D that have been used in the 3D). So one never supplies a number, one just supplies an ever-incrementing numerical variable, in this case 'hsid'.

Typical 2D Script to draw a line with dimension 'A':

```
!Syntax: HOTSPOT2 x,y, hsid, param, flag [,disparam]
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid, A, 1
hsid=hsid+1 !Move
HOTSPOT2 A,0, hsid, A, 2
hsid=hsid+1 !Vector
HOTSPOT2 -1,0, hsid, A, 3
LINE2 0,0,A,0 !The Object
```

ID: The hsid=hsid+1 is a neat way to generate new ID numbers without having to write actual numbers.

Comment: For me it's part of my religion to put a comment before each GH to describe its function – essential for object maintenance. It is also valuable to keep the flag order correct, i.e. 1,2,3. Many an object have I seen where the order is jumbled.

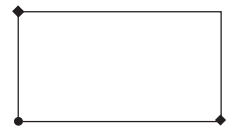
Vector: This Hotspot is invisible – it must be placed in a position opposite to the direction in which the Move-spot is moving. If the Move-spot is going to the right, then anywhere to the left, 1 inch, 1 metre, 100 metres, will be adequate to describe a direction for sliding the spot.

Flag: There are additional attributes which can be added to the 1,2 flags: 128, 256 and 512. GDL recognises by their size that they are attributes, not flags.

+128 in this example makes the Base-spot invisible. You can also hide the Move-spot if you want to, but why?

+256 makes the Base-spot visible and active. Without this a visible Base-spot is static (can be used for picking and moving the object). With 256, the action can be reversed, and the Base-spot will, if grabbed with the mouse pointer, perform like a Move-spot and the Move-spot will perform like a Base-spot.

+512 is used for angles in 2D.



Typical script for a Rectangle:

```
!Hotspot for A
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid, A, 1+128 !invisible
hsid=hsid+1 !Move
HOTSPOT2 A,0, hsid, A, 2
hsid=hsid+1 !Vector
HOTSPOT2 -1,0, hsid, A, 3

!Hotspot for B
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid, B, 1+256 !active
hsid=hsid+1 !Move
HOTSPOT2 0,B, hsid, B, 2
hsid=hsid+1 !Vector
HOTSPOT2 0,-1, hsid, B, 3

RECT2 0,0,A,B !Build the object!
```

You could improve the functionality of this by writing additional hotspot routines for the far corner (A,B) of the rectangle. It's coming soon!

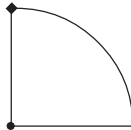
'Angle' GHs – Rotate

GHs have to be written as a group. It takes three to define a single angle-based hotspot in 2D and four to do the same in 3D. GDL must know the Base point from where the action starts, flag **4=Base**; the actual Moving visible point, flag **5=Move**; the Centre of rotation, flag **6=Centre**; and a Vector for the swivel/axle of rotation, flag **7=Vector**. The Vector is not required in 2D, the axle of rotation is assumed to be vertically above the object.

Typical 2D Script for rotating an angle 'angl' using a line of length 'len' :

```
hsid=hsid+1 !Base
HOTSPOT2 len,0, hsid, angl, 4
ROT2 angl
hsid=hsid+1 !Move
HOTSPOT2 len,0, hsid, angl, 5
DEL 1

hsid=hsid+1 !Centre
HOTSPOT2 0,0, hsid, angl, 6
!Build it!
ARC2 0, 0, len, 0,angl
LINE2 0,0,len,0
ROT2 angl
LINE2 0,0,len,0
DEL 1
```



If you are making this, don't forget to make the two parameters, 'angl' and 'len'. GHs only work on parameters, not on variables.

Iteration!: Incredible as it may seem, you can rotate the line with the Move-spot using a ROT2 before it knows the angle. GDL is iterating rapidly, and responding to each tiny change of angle as you turn the Move-spot. The iteration is such that you can even have the Radius varying as the Move-spot rotates, which sounds insane, but it works.

A way to write the routine without rotating the Move-spot with ROT2 is using easy Trigonometry:

```
hsid=hsid+1 !Base
HOTSPOT2 len,0, hsid, angl, 4+256
hsid=hsid+1 !Move
HOTSPOT2 len*COS(angl),len*SIN(angl),hsid,angl,5
hsid=hsid+1 !Centre
HOTSPOT2 0,0, hsid, angl, 6
!Build it!
ARC2 0, 0, len, 0,angl
LINE2 0,0,len,0
ROT2 angl
LINE2 0,0,len,0
DEL 1
```

Try the 256 attribute to allow the Base-spot to work as a Move and rotate spot.

Vector: In the 3D, you must define an axle of rotation, using an imaginary line between the centre of rotation and the vector point. The position should be perpendicular to the plane of rotation, so that positive rotation is anti-clockwise. If you get this wrong, you will see rotation occurring opposite to the direction of the Move-spot, so you will be able to correct it.

Flag: There are additional attributes which can be added to the 4,5 flags: 128, 256 and 512. GDL recognises by their size that they are attributes, not flags.

+128 Invisible.

+256 make the Base-spot active. This can have the useful side effect of rotating the whole object.

+512 can be used in 2D to reverse the direction of rotation. Normally, the anti-clockwise direction is regarded as positive, but you may need to rotate something clockwise, with positive angle increments.

Competition between Hotspots!

You can always tell a graphical hotspot from a normal one as it will be diamond shaped in AC 8.1. But if you get two competing hotspots over the same point you may have problems. If the two spots are both linear, they will work in cooperation (if vectors permit), with ArchiCAD sensing the direction in which the user is pushing his mouse and moving correctly.

If a Linear and Angle Hotspot land on the same place, the Linear Hotspot is the stronger of the two. Avoid this by using 'witness lines' small lines that make it clear to the user if a spot is linear or angular, or which move the angle spot a few centimetres out from the exact position – like making a lever.

Get two working together – now!

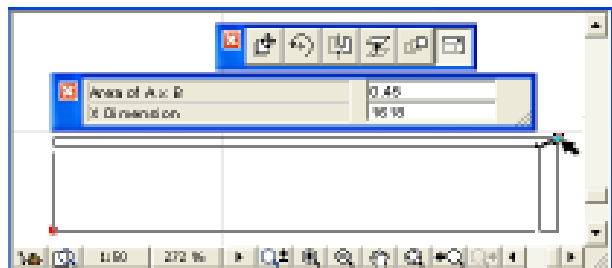
We can work linear ones together and this is an example. In the 0,0,A,B rectangle, only the A,B spot is working. The 0,0 is the bottom left corner, but it is not the Base-spot for either of the A or B stretchers. Note the position of the Base-spots in both cases. This rectangle works better than the earlier example.

While we are here, we can see the use of the extra 'display parameter' that goes after the Hotspot flag. This can enable a different parameter to show in the editable infield while the object is being modified.

```
!Hotspot for A
hsid=hsid+1 !Base
HOTSPOT2 0,B, hsid, A, 1+128 !invisible
hsid=hsid+1 !Move
HOTSPOT2 A,B, hsid, A, 2,area
hsid=hsid+1 !Vector
HOTSPOT2 -1,B, hsid, A, 3

!Hotspot for B
hsid=hsid+1 !Base
HOTSPOT2 A,0, hsid, B, 1+256 !active
hsid=hsid+1 !Move
HOTSPOT2 A,B, hsid, B, 2,A
hsid=hsid+1 !Vector
HOTSPOT2 A,-1, hsid, B, 3

HOTSPOT2 0,0 !simple one for the corner
RECT2 0,0,A,B !Build the object!
```



Make a parameter called 'area' as a Real Number. Calculate a result and send that back to the user with a Parameters command. Add 'area' to the end of the Move-spot statement as the 'display parameter'. This will display in the editable infield while you modify the object.

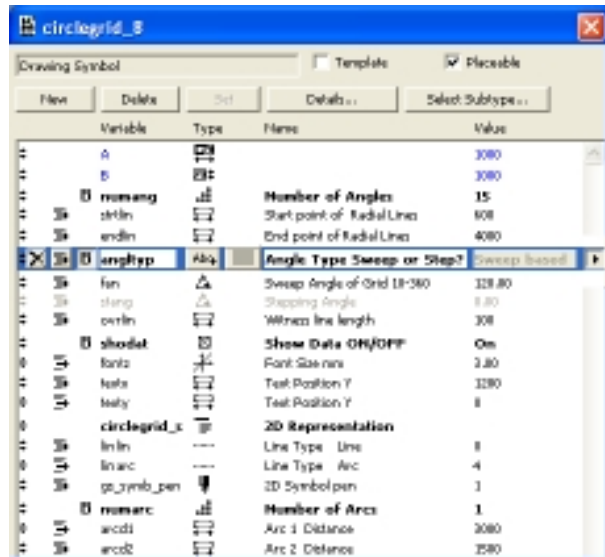
```
!Master Script
PARAMETERS area=A*B
```

Graphical Hotspots!

2D CircleGrid, with Text

GDL can be used to create 2D entities that help you to lay out models or drawings, but which do not have any 3D – let's call them drafting aids. **CircleGrid** is a simple example. It can help you lay out walls in a radiating pattern. You can explode it to turn it into lines, or store it on a ghost storey to help with layout. It also illustrates:

- FOR...NEXT Loop used with angles,
- Pop down menu, PARAMETERS & Master Script,
- Defining Style and using Text and Strings,
- Using a Global Variable
- Graphical Hotspots for everything!



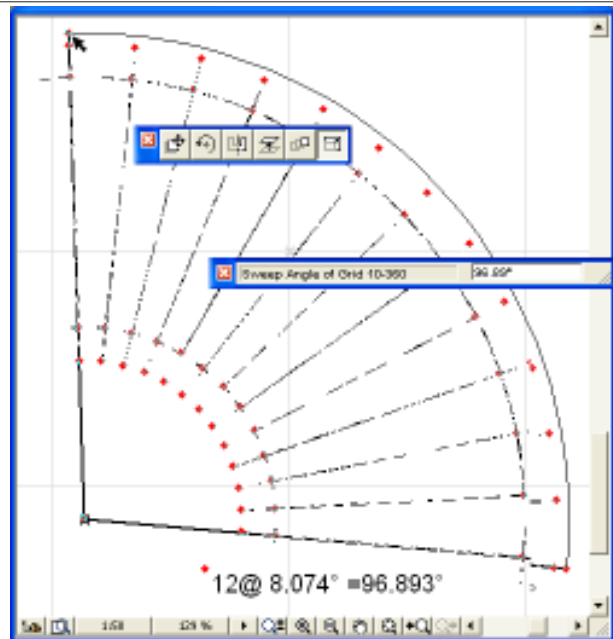
Parameters

In this CircleGrid, there are a user-defined number of radiating lines, and 2 arcs (there could be more if you want). Graphical Hotspots at each point allow the user to rotate and stretch parts of the grid. A choice of 'sweep angle for the whole fan' or 'stepping angle for each segment' is offered through a Popdown menu. The text block gives feedback to the user. The object does not fit a cuboid, so A and B are not used.

Master Script

This sets up the Popdown menu, sets a lower limit on the number of angles, checks the user's choice from the menu and calculates the reciprocal parameters. The PARAMETERS command provides feedback to the user. We can also hide unwanted parameters.

This object could be improved with Parameter Arrays for the arcs, allowing us as many arcs as we want, all equipped with Hotspots. With PAs for the lines, individual lines could have varied lengths.



```
!Circle grid
!Master Script
```

```
at1='Sweep based'
at2='Step angle based'
VALUES 'angltyp' at1,at2
IF numang<=1 THEN numang=1
IF angltyp=at1 THEN
  IF fan<numang THEN fan=numang
  IF fan>=359 THEN fan=360
  stang=fan/numang !Stepping angle
  LOCK 'stang'
ENDIF
```

This Popdown menu must be defined in the Master Script because we use 'at1' and 'at2' in the 2D Script.

If we select sweep we need to know the step angle, and if we select step angle, we need to calculate sweep.

Lock the unused parameter, but do not hide it, it's useful as user feedback with the PARAMETERS statement.

```
IF angltyp=at2 THEN
  fan=numang*stang
  LOCK 'fan'
ENDIF
```

```
PARAMETERS fan=fan,stang=stang,
            numang=numang
```

```
IF numarc=0 THEN HIDEPARAMETER 'arcd1','arcd2'
IF numarc=1 THEN HIDEPARAMETER 'arcd2'
```

2D Script

Title your script, and set the Pen. Decide if you want it made of solid or dotted lines. Hotspots should be decided as early as possible, so these are *always* best placed in a subroutine to avoid clogging up the executive script.

A FOR...NEXT loop is used for the main routine. Rotating around the centre, each line consists of a LINE2 command, with Hotspot subroutines to mark out the line ends and arc intersections. DEL the ROT2 command each time the loop is used. After the Loop we have the ARC2 statements.

The deliberate error of 1/1000 degree for 'fan' is included, or GDL could make tiny rounding down errors and miscount the number of lines to draw. When you ROT2 in 2D the positive direction of rotation is anti clockwise.

We now look at how to label the object, as an added option, and a demo of Text and Strings.

```

!CircleGrid with Hotspots
!2D Script

PEN gs_symb_pen
HOTSPOT2 0,0

IF angltyp=at1 THEN GOSUB 1000:!Hotspots Fan
IF angltyp=at2 THEN GOSUB 1010:!Hotspots Steps

!Draw the actual grid          Hotspots Before the
LINE_TYPE linlin              Lines and Arcs!
FOR ang=0 TO fan+0.001 STEP stang
  ROT2 ang                     Set the Line type.
  GOSUB 1020:!Hotspots Lines
  GOSUB 1030:!Hotspots Arcs
  LINE2 strtlin,0,endlin,0
  DEL 1                        If you Define the
  NEXT ang                    Style, don't forget to
LINE_TYPE linarc              Set the Style next!
IF numarc THEN ARC2 0,0,arcd1,0,fan
IF numarc=2 THEN ARC2 0,0,arcd2,0,fan

!Show angular increment and sweep
IF shodat AND fontz THEN
  GOSUB 1050:!Hotspots for Text
  DEFINE STYLE 'angtext' 'Arial',fontz,1,0
  SET STYLE 'angtext'
  angstr=STR(numang,3,0)+'@'+STR('%6.3dd',stang)+'='+STR('%6.3dd',fan)
  ADD2 textx,texty
  ROT2 -SYMB_ROTANGLE          By rotating it by the negative
  TEXT2 0t0,ngstr              of the object's rotational
  DEL 2                        angle, the label will always
  ENDIF                        remain upright.

END: !=====

```

Text and Strings

We use Text as a form of user feedback. It is essential to DEFINE STYLE first. We keep it simple here and use Arial (although you could use a font Popdown menu). After defining style, you have to SET STYLE.

TEXT2 x,y,string is the command to show data in 2D, but it can only show one thing at a time, a number or a string. If the text element contains more than one item, you have to construct a single string by concatenating all the data. You could use a variable like 'angstr' in which all parts are defined as small strings, then joined by plus signs into one large one.

You can convert 'numang' (the number of angles) as a string using the STR() command and parameters within the brackets decide if it's displayed as an integer, or a decimal number. Add to it an @ symbol, and then, to print the degrees symbol as a string, you can use the '%dd' method with STR() – see the manual to test this further. Try this and experiment with changing the number of digits to be displayed. Add in similar strings for the '≡' and the Sweep angle.

Text position and Global Variables

Because the object is dynamic in many respects, resizing and rotating at the touch of a hotspot, we need to relocate the Text label with some hotspots of its own. We need to enable the label to stay upright, which brings us to our first experience of Global Variables.

GDL Objects can get information from ArchiCAD such as their current storey / layer, drawing scale, position of camera, wall material / thickness, rotational angle, object ID number. These variables are to do with global matters outside the object. SYMB_ROTANGLE tells the object how it has been rotated.

```

1000:!Hotspots Fan angle
LINE2 endlin,0,endlin+ovrlin,0 !Witnessline
hsid=hsid+1 !Base
HOTSPOT2 endlin+ovrlin,0,    hsid,fan,4+256
hsid=hsid+1 !Move
ROT2 fan
HOTSPOT2 endlin+ovrlin,0, hsid,fan,5
LINE2 endlin,0,endlin+ovrlin,0 !Witnessline
DEL 1
hsid=hsid+1 !Centre
HOTSPOT2 0,0, hsid,fan,6
RETURN

1010:!Hotspots Stepping angle
hsid=hsid+1 !Base
HOTSPOT2 endlin+ovrlin,0,    hsid,stang,4+256
hsid=hsid+1 !Move
ROT2 stang
HOTSPOT2 endlin+ovrlin,0,    hsid,stang,5
DEL 1
hsid=hsid+1 !Centre
HOTSPOT2 0,0, hsid,stang,6
RETURN

1020:!Hotspots Lines
hsid=hsid+1 !Base
HOTSPOT2 0,0,    hsid,strtlin,129
hsid=hsid+1 !Move
HOTSPOT2 strtlin,0, hsid,strtlin,2
hsid=hsid+1 !Vect
HOTSPOT2 -1,0,    hsid,strtlin,3

hsid=hsid+1 !Base
HOTSPOT2 0,0,    hsid,endlin,129
hsid=hsid+1 !Move
HOTSPOT2 endlin,0, hsid,endlin,2
hsid=hsid+1 !Vect
HOTSPOT2 -1,0,    hsid,endlin,3
RETURN

1030:!Hotspots Arcs
IF numarc THEN
  hsid=hsid+1 !Base
  HOTSPOT2 0,0,    hsid,arcd1,129
  hsid=hsid+1 !Move
  HOTSPOT2 arcd1,0, hsid,arcd1,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0,    hsid,arcd1,3
ENDIF

IF numarc=2 THEN
  hsid=hsid+1 !Base
  HOTSPOT2 0,0,    hsid,arcd2,129
  hsid=hsid+1 !Move
  HOTSPOT2 arcd2,0, hsid,arcd2,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0,    hsid,arcd2,3
ENDIF
RETURN

1050:!Hotspots for Text
hsid=hsid+1 !Base
HOTSPOT2 0,texty,    hsid,textx,129
hsid=hsid+1 !Move
HOTSPOT2 textx,texty, hsid,textx,2
hsid=hsid+1 !Vect
HOTSPOT2 -1,texty,    hsid,textx,3

hsid=hsid+1 !Base
HOTSPOT2 textx,0,    hsid,texty,129
hsid=hsid+1 !Move
HOTSPOT2 textx,texty, hsid,texty,2
hsid=hsid+1 !Vect
HOTSPOT2 textx,-1,    hsid,texty,3
RETURN

```

Check these out one at a time, look at the real object on the Cookbook CD, and you will find that Graphical Hotspots, if tackled subroutine by subroutine are really very easy to make.

Graphical Hotspots!

3D Vertical Blinds

THE **Vertical Blinds** object demonstrates Loops and 3D Graphical Hotspots. It also uses a technique called 'passing parameters' which uses a subroutine to make a part that we wish was in GDL but isn't: a 'Squilinder'.

This is also a 'smart' object: notice that it does not ask how many blinds we want. Unlike the object in the AC8 library, this one calculates from the length of the rail how many blinds would be possible, and then uses that number, subdivided into the 'drawing length'.

Parameters

The Origin of this object is at the top surface of the rail, so that it can be applied to the head of the window or at the ceiling height. As the blinds hang downwards, we cannot use 'zzyzx': the cuboid misbehaves with a downwards direction for height when working the hotspots in 3D. But why worry!! With Graphical Hotspots, this is not a problem.

We use 'A' for the metal rail, and a GHs for the width and height of the blinds. If this was a manufactured object, you could perhaps hide the rail width and height and blind width parameters if they are unchanging.

For this object, try using a subtype, in this case a building element. You will get extra parameters to do with 2D Representation and ArchiFM. The former are useful, but the latter should be pushed down to the bottom of the table – and hidden if you want to avoid annoying or confusing the user.

It's good to have separate pens for 3D and 2D. We can build a POLY2 for the 2D, so we need the special pen parameters.

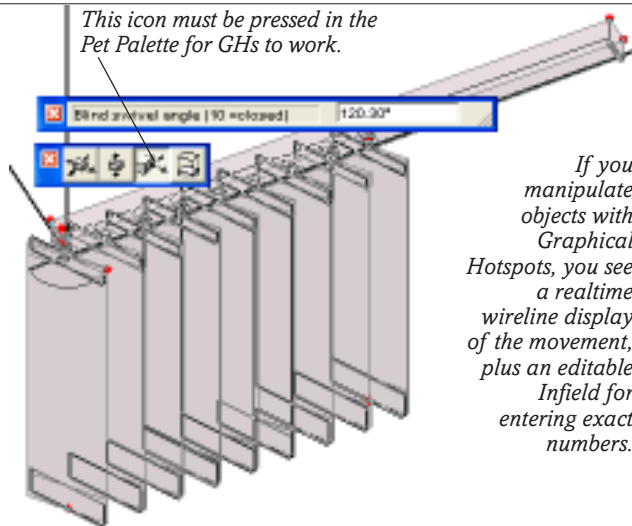
Master Script

This routine works out how many blinds would fit the rail if the blinds were fully stretched and closed. Then it reads the distance to which the blinds have been drawn and works out the dynamic value of blind spacing at this moment in time. For the distance and angle parameters, a ValueList with a RANGE function is a good way to set smallest and largest limits. You can also set Stepping values, so that the object snaps neatly to the required sizes, e.g. the blind angle snaps to 10 degree increments without a host of IF statements. The Parameters statement reports back to the user.

3D Script

This is very short and consists mainly of GOSUBs for the GHs subroutines. The Rail and the Blinds are also carried in tidy subroutines. The Rail is simply a set of blocks. The Blinds are generated by a FOR... NEXT loop. Add one extra blind at the end because our calculation knows the number of bays, not blinds.

This icon must be pressed in the Pet Palette for GHs to work.



If you manipulate objects with Graphical Hotspots, you see a realtime wireline display of the movement, plus an editable Infield for entering exact numbers.

Vertical_Blinds_CB

Building Element ☐ Template ☒ Placeable

New Delete Set Details... Select Subtype...

Variable	Type	Name	Value
A	X Dimension	X Dimension	1800
B	Y Dimension	Y Dimension	1000
ZZYZX	Z Dimension	Z Dimension	0
railthk	Rail thickness	Rail thickness	35
railht	Rail height	Rail height	30
Blinds2_sp1	Blind config	Blind config	
blindwid	Blind width	Blind width	100
blindang	Blind swivel angle (10 =closed)	Blind swivel angle (10 =closed)	45.00
bhang	Blind hanging length	Blind hanging length	1500
blen	Length of Blinds along A	Length of Blinds along A	1200
Blinds2_sp0	Materials	Materials	
railmat	Rail material	Rail material	39
blindmat	Blind Material	Blind Material	23
gs_2D_repr	2D Representation	2D Representation	
gs_cont_pen	Pen Colour in 3D	Pen Colour in 3D	95
gs_symb_pen	Pen Colour in 2D	Pen Colour in 2D	3
gs_fill_type	Fill Type	Fill Type	24
gs_fill_pen	Fill Pen	Fill Pen	4
gs_back_pen	Fill Background Pen	Fill Background Pen	91
AC_show2DH	Showing 2D Hotspots in 3D	Showing 2D Hotspots in 3D	On
gs_list	ArchiFM & Listing Parameters	ArchiFM & Listing Parameters	
gs_list_cost	Cost	Cost	0.00
gs_list_manufa	Manufacturer	Manufacturer	Abc
gs_list_note	Note/Remarks	Note/Remarks	00
gs_list_location	Location	Location	5

!Vertical Blinds: Master Script

```
IF bhang<=railthk+0.13 THEN bhang=railthk+0.130
ovlp=0.01!overlap of blinds
blindspac= (blindwid-ovlp)
blindnum= INT(0.5+(A-0.01)/blindspac)
bspa=blen/blindnum!Dynamic spacing
```

```
VALUES 'blindang' RANGE [10,170] STEP 10,10
```

```
VALUES 'A' RANGE [0.6,2.0] STEP 0.6,0.1
```

```
VALUES 'blen' RANGE(0.025+0.015*blindnum/2,A-0.025)
```

```
PARAMETERS zzyzx=0,
             bhang=bhang,blindang=blindang
```

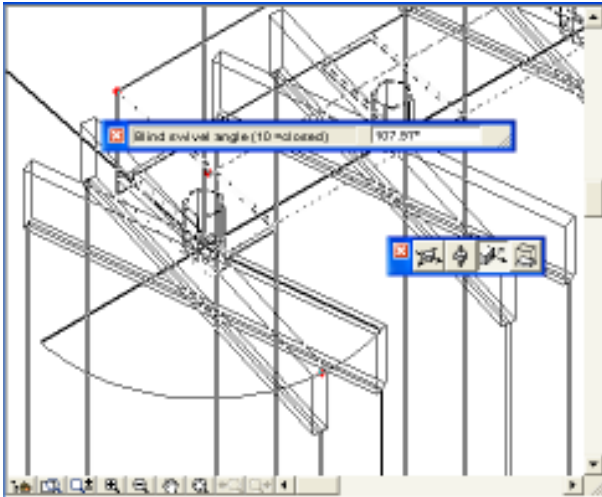
```
HIDEPARAMETER 'zzyzx'
```

We hide parameter 'zzyzx' and make it zero, otherwise it produces an irrelevant cuboid wireline if you try to play with hotspots. VALUES statements with a RANGE and STEP value are an excellent way to eliminate complex IF routines.

```

!Vertical Hanging Blinds
!3D Script
PEN gs_cont_pen
TOLER 0.001
GOSUB 1000:!Rail Hotspots
!Blind drawing hotspots
GOSUB 1010:!top
ADDz -bhang-0.017
GOSUB 1010:!bottom
DEL 1
GOSUB 1020:!Vertical hotspot
GOSUB 1030:!Rotate Blinds angle
GOSUB 100:!Rail
GOSUB 200:!Blinds
END!=====

```



```

1000:!Rail Hotspots
HOTSPOT 0,0,0
HOTSPOT A,0,0
HOTSPOT A,railthk,0
HOTSPOT 0,railthk,0
RETURN
1010:!blind drawing hotspots
ADD 0.0125,railthk/2, -railht
hsid=hsid+1!Base
HOTSPOT 0, 0,0, hsid, blen,1+128
hsid=hsid+1!Move
HOTSPOT blen,0,0, hsid, blen,2
hsid=hsid+1!Vect
HOTSPOT -1, 0,0, hsid, blen,3
DEL 1
RETURN
1020:!Vertical hotspot
ADD 0.0125, railthk/2, -railht-0.017
hsid=hsid+1!Base spot
HOTSPOT 0,0,0, hsid,bhang,1+128
hsid=hsid+1!move spot
HOTSPOT 0,0,-bhang, hsid,bhang,2
hsid=hsid+1!Vect spot
HOTSPOT 0,0,1, hsid,bhang,3
DEL 1
RETURN
1030:!Rotate Blinds angle
ADD 0.0125, railthk/2, -railht-0.017
hsid=hsid+1!Base spot
HOTSPOT -blindwid/2,0,0, hsid,blindang,4+128
ROTz blindang-90
hsid=hsid+1!Move spot
HOTSPOT 0,-blindwid/2,0, hsid,blindang,5
DEL 1
hsid=hsid+1!Centre spot
HOTSPOT 0,0,0, hsid,blindang,6
hsid=hsid+1!Vector spot
HOTSPOT 0,0,1, hsid,blindang,7
DEL 1
RETURN

```

```

100:!---Rail---
MATERIAL railmat
ADDz -railht
BLOCK A, railthk, railht!Main rail
ADDz -0.005
BLOCK 0.005, railthk, 0.005!End block
ADDx A-0.005
BLOCK 0.005, railthk, 0.005!End block
DEL 3
RETURN
200:!---Blinds---
ADD 0.0125, railthk/2,-railht
FOR k=1 TO blindnum+1
ADDx bspa*(k-1)
GOSUB 210:!The Blind
DEL 1
NEXT k
DEL 1
RETURN
210:!---Single Blind---
MATERIAL railmat
ADDz -0.002
x=0.015:y=railthk:z=0.002
GOSUB 220:!Squilinder: Traveller
ADDz -0.015
ROTz blindang-90
CYLIND 0.015, 0.005!swivel
ADDz -bhang
x=0.004:y=blindwid:z=0.04
GOSUB 220:!Squilinder: Lower weight
ADDz 0.04
x=0.001:y=blindwid:z=bhang-0.06
GOSUB 220:!Squilinder: Blind
ADDz bhang-0.06
x=0.004:y=blindwid:z=0.02
GOSUB 220:!Squilinder: Gripper
DEL 6
RETURN

```

The loop uses the (k-1) trick to ensure that the DEL can occur within the loop.

```

220:!Squilinder - quick prisms
PRISM 5,z,
-x/2,-y/2,
x/2,-y/2,
x/2, y/2,
-x/2, y/2,
-x/2,-y/2
RETURN

```

Although I do not like Multistatement lines using the colon, this is one occasion where it seems to be acceptable.

Squilinder: Passing Parameters

For the Blind, we have a mechanism that swivels on a Cylinder and it is very complicated to do all the grippers, blind and weight with BLOCK statements.

We need an Axially oriented cuboid shape that can follow the axis of the swivel. This does not exist in GDL – so we must make one. I call it ‘Squilinder’ – Square Cylinder. This is the subroutine 220 that has only 3 arguments, x, y and z. By passing just those 3 parameters to the subroutine, we get effortless creation of blocks which with the right ADDz commands, all line up perfectly with the axis.

Hotspots can become the new UI!

THE user can change so many parameters with hotspots that it can become positively tedious to open the object settings box. With some ingenuity, it is possible to move more of the parameters to the 2D symbol.

You can build a form of control panel in the 2D symbol, with sliders and levers to control other parameters in the object such as Boolean choices, or opening angles of windows, or even selections from Popdown menus. These will be explored later in the book!

2D Script for the Vertical Blinds

```
!Vertical Blinds
!2D Script
```

```
PEN gs_symb_pen
```

```
GOSUB 1000:!Main rail hotspots
GOSUB 1010:!Blind drawing hotspots
GOSUB 1030:!Blind angle hotspots
```

```
!PROJECT2-3,270,3
```

```
1000:!Main rail
```

```
DRAWINDEX 20
```

```
FILL gs_fill_type
```

```
POLY2_B 5,7,gs_fill_pen,gs_back_pen,
0,0,1,
A,0,1,
A,railthk,1,
0,railthk,1,
0,0,-1
```

*Do the Hotspots first.
Retain PROJECT2
until you are sure that
the new scripted
symbol fits perfectly.*

```
200:!Blinds
```

```
DRAWINDEX 10
```

```
ADD2 0.005,0
```

```
FOR k=1 TO blindnum+1
```

```
ADD2 bspa*(k-1),0
```

```
GOSUB 210:
```

```
DEL 1
```

```
NEXT k
```

```
DEL 1
```

*Normally the lines of
the blinds will
overdraw the Polygon
of the rail. Use
DRAWINDEX
statements to ensure
that the lines are
under the rail.*

```
END!=====
```

```
210: !Single Blind
```

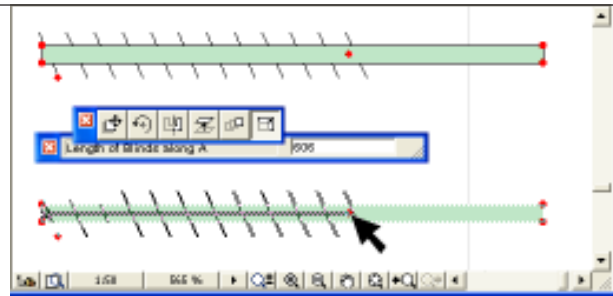
```
ADD2 0.0075,railthk/2
```

```
ROT2 blindang-90
```

```
LINE2 0,-blindwid/2,0,blindwid/2
```

```
DEL 2
```

```
RETURN
```



```
1000:!Main rail hotspots
```

```
HOTSPOT2 0,0
```

```
HOTSPOT2 A,0
```

```
HOTSPOT2 A,railthk
```

```
HOTSPOT2 0,railthk
```

```
RETURN
```

```
1010:!Blind drawing hotspots
```

```
hsid=hsid+1!Base
```

```
HOTSPOT2 0.005,railthk/2, hsid, blen,1+128
```

```
hsid=hsid+1!Move
```

```
HOTSPOT2 blen,railthk/2, hsid, blen,2
```

```
hsid=hsid+1!Vect
```

```
HOTSPOT2 -1,railthk/2, hsid, blen,3
```

```
RETURN
```

```
1030:!Blind angle hotspots
```

```
ADD2 0.0125,railthk/2
```

```
hsid=hsid+1!Base
```

```
HOTSPOT2 -blindwid/2,0, hsid, blindang,4+128
```

```
hsid=hsid+1!Move
```

```
ROT2 blindang-90
```

```
HOTSPOT2 0,-blindwid/2, hsid, blindang,5
```

```
DEL 1
```

```
hsid=hsid+1!Centre
```

```
HOTSPOT2 0,0, hsid, blindang,6
```

```
DEL 1
```

```
RETURN
```

*The 2D Hotspots closely
resemble the code in the 3D –
indeed they are best
prototyped in the 2D first.
Use the same subroutine
numbers.*

Graphical Hotspots!

Add them to the RafterJoist Tool

WITH very little extra effort, we can transform the **RafterJoist Tool** into a more powerful and user friendly object. As with the previous objects, we can place some GOSUBs at the start of the 2D and 3D Scripts, and append the GHs subroutines at the end of the script.

In this case we can provide GHs for the Eaves overhang, the Height and the Pitch of the roof. We can make the wallplate independently extendible which will make corner detailing easier. We can even make the rafter spacing stretchy, but the user might be advised to write this in accurately for a real project.

Parameters

The parameter table has been tidied up with more cascading groups, but the only significant change is the addition of start and end points for the wallplate, the timber section on which the rafters or joists rest.

Variable	Type	Name	Value
A			2972
B			1000
zzyzx		Height at End	762
B rtyp		Rafter by Height/Pitch?? By Pitch	
pitch		Pitch of Roof	37.31
_sp2		Configure	
jspac		Rafter/Joist spacing	406
jw		Rafter/Joist width	51
jd		Rafter/Joist depth	203
_sp0		Materials and 2D	
jmat		Material	15
gs_cont_pen		Pen color	1
secfil		Section fill	24
AC_show2DHot		Showing 2D Hotspots in 3D	Off
_sp1		3D Options	
evoh		Eaves Overhang	106
enraft		End Rafter ON/OFF?	On
eavcut		Eaves Cut below Wallplate	254
rafterjois_sp1		Mitring	
hipm		Hip Mitre ON./OFF??	On
mitr		Mitring angle	45.00
mitrx		Mitring offset	-100
rafterjois_sp0		Wall Plate	
walp		Wall plate ON/OFF?	On
birds		Birdsmouth	Off
walpw		Wall plate Width	100
walph		Wall Plate Height	75
walpst		Wall Plate Start	100
walpen		Wall Plate End	3000

Which Script do we convert?

Most of the changes except GHs for Pitch are best done in 2D, so we do them all in 2D first, and these can later be adapted to work in the 3D. The Pitch and the Height option and Eaves cut are required in 3D.

```

!Rafter Joisting tool
!2D Script

PEN gs_cont_pen

GOSUB 1000:!Hotspots Main width
GOSUB 1010:!Hotspots Eaves Overhang
GOSUB 1020:!Hotspots Hip mitring X
GOSUB 1030:!Hotspots Hip mitring ang
GOSUB 1040:!Hotspots Wallplate length

!-----
!THE EXISTING SCRIPT GOES IN HERE.
!-----

END:!=!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

100:!rafter
!THIS REMAINS UNCHANGED

110:!Hotspots for Depth
  HOTSPOT2 dist,0
  HOTSPOT2 dist,B
  LINE2 dist-jw/2,0,dist+jw*3/2,0
RETURN

1000:!Hotspots Main width+joistspacing
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, A,1+256
  hsid=hsid+1 !Move
  HOTSPOT2 A,0, hsid, A,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0, hsid, A,3
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, jspac,1+128
  hsid=hsid+1 !Move
  HOTSPOT2 0,jspac, hsid, jspac,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0, hsid, jspac,3
RETURN

1010:!Hotspots Eaves Overhang
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, evoh,129
  hsid=hsid+1 !Move
  HOTSPOT2 0,-evoh, hsid, evoh,2
  hsid=hsid+1 !Vect
  HOTSPOT2 0,1, hsid, evoh,3
RETURN

1020:!Hotspots Hip mitring X-position
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, mitrx,129
  hsid=hsid+1 !Move
  HOTSPOT2 mitrx,0, hsid, mitrx,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0, hsid, mitrx,3
RETURN

1030:!Hotspots Hip mitring angle
  hsid=hsid+1 !Base
  HOTSPOT2 mitrx,jd, hsid, mitr,4+128
  ADD2 mitrx,0
  ROT2 -mitr
  LINE2 0,0,0,jd !witness line
  hsid=hsid+1 !Move
  HOTSPOT2 0,jd,hsid, mitr,5
  DEL 2
  hsid=hsid+1 !Centre
  HOTSPOT2 mitrx,0, hsid, mitr,6+512
RETURN

1040:!Hotspots Wallplate length
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, walpst,1
  hsid=hsid+1 !Move
  HOTSPOT2 walpst,0, hsid, walpst,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0, hsid, walpst,3
  hsid=hsid+1 !Base
  HOTSPOT2 0,0, hsid, walpen,1
  hsid=hsid+1 !Move
  HOTSPOT2 walpen,0, hsid, walpen,2
  hsid=hsid+1 !Vect
  HOTSPOT2 -1,0, hsid, walpen,3
RETURN

```

Insert these GOSUBs at the start, and write the subroutines at the bottom. As a matter of habit, I apply a special numbering system to them, here it is based on 1000 plus, with the primary stretching function written first to replace the old A,B stretching.

It is too complex to put the Hip Mitring routine in the 3D. Do it in 2D only.

It is a pity that there is not a method for 'passing parameters' to a standard routine for Graphical Hotspots like you can do with Infields in the User Interface (UI) design. Only the name of a valid parameter from the parameter table is acceptable.

```

!Rafter Joisting tool
!3D Script

PEN gs_cont_pen
MATERIAL jmat
SECT_FILL secfil,gs_cont_pen,91,gs_cont_pen

GOSUB 1000:!Hotspots Main width
GOSUB 1010:!Hotspots Height/Pitch
GOSUB 1080:!Hotspots for Eaves Cutting

!-----
!THE EXISTING SCRIPT GOES IN HERE.
!-----

END:!=!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

100:!rafters
!THIS REMAINS UNCHANGED

1000:!Hotspots Main width and joist spacing
These are the same as the 2D routine, suitably adapted by
changing HOTSPOT2 to HOTSPOT and adding a Z height.
RETURN

1010:!Hotspots Height/Pitch
IF rtyp=rp0 THEN !Height
  hsid=hsid+1 !Base
  HOTSPOT 0,B,0, hsid, zzyzx,1+128
  hsid=hsid+1 !Move
  HOTSPOT 0,B,zzyzx, hsid, zzyzx,2
  hsid=hsid+1 !Vect
  HOTSPOT -1,B,0, hsid, zzyzx,3
ENDIF
IF rtyp=rp1 THEN !Pitch angle
  jlen=SQR(B^2 + zzyzx^2)
  hsid=hsid+1 !Base
  HOTSPOT 0,B,0, hsid, pitch,4+128
  ROTx pitch
  LIN_ 0,jlen-jw,0, 0,jlen,0 !witness line
  hsid=hsid+1 !Move
  HOTSPOT 0,jlen,0, hsid, pitch,5
  DEL 1
  hsid=hsid+1 !Centre
  HOTSPOT 0,0,0, hsid, pitch,6
  hsid=hsid+1 !Swivel
  HOTSPOT 1,0,0, hsid, pitch,7
ENDIF
RETURN

1080:!Hotspots for Eaves Cutting
  hsid=hsid+1 !Base
  HOTSPOT 0,-evoh,0, hsid, eavcut,1+128
  hsid=hsid+1 !Move
  HOTSPOT 0,-evoh,-eavcut, hsid, eavcut,2
  hsid=hsid+1 !Vect
  HOTSPOT 0,-evoh,1, hsid, eavcut,3
RETURN

```

Add the GOSUBs to the start of the 3D Script

For the Pitch to work, we need to calculate the length of the rafter, using Pythagoras theorem. this is calculated on the fly many times a second as the rafter is being tilted.

We need a small extension line so we know where the point is in space if the end has been mitred.

Manufacturing concepts

Make a Cabinet with Drawers

GDL can be a potent tool in furniture making. A freely stretchy object in the Floor Plan? How can something that whimsical be made into a real object?

But suppose we can place a GDL furniture object that has been supplied by a manufacturer; stretch it to the required size; know that it obeys manufacturing rules on jointing and thicknesses; and then retrieve the dimensions of side and back panels, tops, drawer sides and everything; pass the data to CNC machines which can churn out the panels correctly – **GDL could be a powerful tool for product configuration.**

I discovered the power of this when teaching a furniture manufacturer a bit of advanced GDL. He was already doing much of the above, his two CNC machines were churning out panels for beautiful custom-made kitchen, bedroom and office furniture – all designed in ArchiCAD with GDL objects, with cutting outlines saved to DXF. He needed to know better ways to work with macros and textures, and how to output dimensions and lists to spreadsheet format! Every component down to the humble drawer back was itemised accurately and cut on his machines. I came away having learnt as much as I taught!

G-Code is the solution – & the problem

Let's not make it sound easy. It's not, indeed it is a personal 'Holy Grail' of mine to get this link working better, to get a direct output from GDL to CNC usable command language (G-Code) without having to go through too many intermediate stages. This may never happen fully because variations to G-Code are proliferating as fast as new machine manufacturers introduce new models. But there may be an intermediate level of data that can be output by GDL and converted to G-Code by dedicated CNC software like MasterCAM. I am still looking for it!



CNC machine = Computer Numerically Controlled machine. Look somewhere like <http://www.cnczone.com/> for more information on CNC manufacturing.



WE are going to make a small cabinet that can include a variable number of drawers. We will work with veneered board, and butt jointed construction with joints, i.e. not mortice & tenoned or rebated.

This is a good exercise in:

- Learning about manufacturing ideas
- Texture mapping on veneers
- Macros - bringing in other GDL files
- Graphical Hotspots (of course!)
- Parameter Arrays
- Smart use of flags ('ts' & 'txtmdir')

Parameters

The cuboid shape will be governed by A, B and zzyzx. We need to establish the board thicknesses. We can offer some 3D options, e.g. on the jointing style for the top panel & drawers ('inset' and 'overhang' detailing) and plinth detailing. Select Subtypes to make this a 'Storage' object. Your new parameters will appear below the ArchiFM stuff and will have to be repositioned.

Master Script

This sets up the menus and defines some dimensional limits – stepping values for the width and depth.

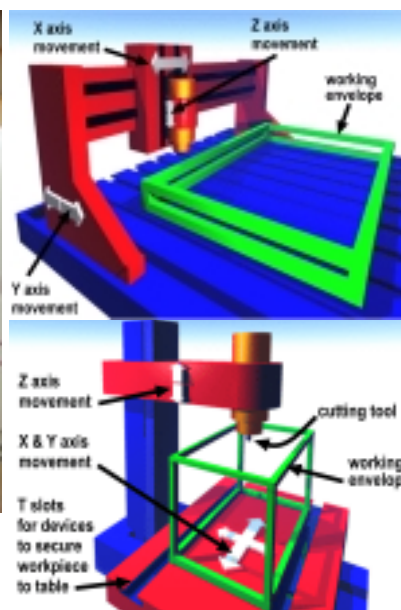
3D Script




















We start by building the essential outer box of the cabinet. We can use the elementary BLOCK command here for most of the components. Note how the Flag 'ts' for timber detailing style is used to vary the size of the side and top panels. We can use the Texture mapping routine that has already been used in the chair and other objects. Make it a subroutine 999, and use it as you need. Get the cabinet working, then move on to the drawers.

Large CNC Machines for manufactured joinery have flatbeds with a moving bridge above them which carries the mechanism that does the drilling and cutting. The drill head has interchangeable tools, based on commands from the computer.

Smaller CNC machines are used for drilling and routing and shaping small objects.

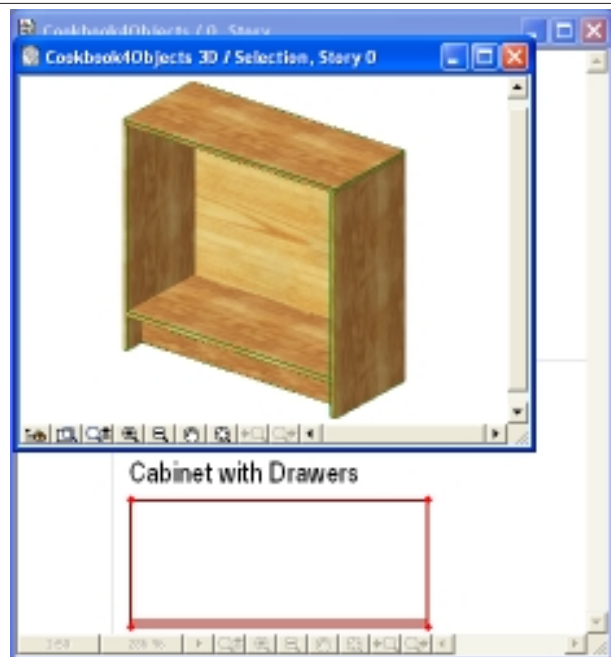
CNC machines work from G-Code. The software converts data (e.g. DXF) into G-Code which instructs the working head.



Storage					<input type="checkbox"/> Template	<input checked="" type="checkbox"/> Placeable
New		Delete	Set	Details...	Select Subtype...	
Variable	Type	Name	Value			
+	A		Cabinet Width	1200		
+	B		Cabinet front-back depth	600		
+	ZZYZX		Z Dimension	1000		
+	AC_show2DHot.		Showing 2D Hotspots In 3D	On		
+	bthk1		Primary Board thickness	20		
+	bthk2		Secondary Board thickness	12		
+	plinh		Plinth Height	150		
+	plind		Plinth Depth	100		
+	cab_mat		Cabinet Exterior material	52		
+	con_mat		Inner construction material	15		
+	timstyl	Abc	Timber Detailing	Office style, ove...		
+	numdraw		Number of Drawers	4		
+	gs_2D_repre..		2D Representation			
+	gs_cont_pen		3D Contour Pen	7		
+	gs_symb_pen		2D Symbol Pen	5		
+	gs_fill_type		Fill Type	24		
+	gs_fill_pen		Fill Pen	91		
+	gs_back_pen		Fill Background Pen	91		
+	gs_list		ArchiFM & Listing Param			
+	gs_list_cost		Cost	0.00		
+	gs_list_manufact.	Abc	Manufacturer			
+	gs_list_note	Abc	Note/Remarks			

Make this a 'Storage' subtype

Make this a 'Storage' subtype



!Cabinet with Drawers We use 'ts' to vary the height and depth of the Side panels. If 'ts' is zero, the panels are shorter. 'ts' alternates between 1 and zero.

```
!3D Script
PEN gs_cont_pen
!Side Panels
MATERIAL cab_mat
BLOCK bthk1,B-ts*bthk1, zzyzx-ts*bthk1 !Left
ADDx A-bthk1
BLOCK bthk1,B-ts*bthk1, zzyzx-ts*bthk1 !Right
DEL 1
txdir=2:GOSUB 999:!Texture
```

!Back panel Depending on the state of the flag 'ts' the Floor and Top panels are also modified.

```
MATERIAL con_mat
ADD bthk1,0,plinh
BLOCK A-bthk1*2, bthk2, zzyzx-bthk1-plinh
DEL 1
!Roof/Worktop
MATERIAL cab_mat
ADD bthk1*(1-ts),0, zzyzx-bthk1
BLOCK A-bthk1*2*(1-ts),B,bthk1
DEL 1
```

!Floor For the Top panel, the 'ts' helps to determine the final width.

```
MATERIAL cab_mat
ADD bthk1,bthk2, plinh
BLOCK A-bthk1*2,B-bthk2-ts*bthk1,bthk1
DEL 1
```

!Plinth We can direct the texture as we please with each panel, and with Open GL, we get an immediate result.

```
MATERIAL cab_mat
ADD bthk1,B-plind-bthk1, 0
BLOCK A-bthk1*2,bthk1,plinh
DEL 1
txdir=0:GOSUB 999:!Texture
```

END: !=====

999:!Texture Horizontal X

```
IF txdir=0 THEN ROTz 0 !Horizontal X
IF txdir=1 THEN ROTz 90 !Horizontal Y
IF txdir=2 THEN ROTy 90 !Vertical Z
BASE
VERT 0,0,0
VERT 0.1,0,0
VERT 0,0.1,0
VERT 0,0,0.1
COORD 256+2,-1,-2,-3,-4
BODY -1
DEL 1
RETURN
```

Note: another use of a Flag, 'txdir' which tells the texture which way to point, and requires only ONE Texture subroutine of the type we used earlier with the Chair and the Rafter – smart and economical.

The two timber styles will become clearer to you when the drawers fit. The 'Office' cabinet drawers overlap the front, and the 'Bedroom' style ones are inset.

!Cabinet with Drawers
!Master Script

There are two ways of designing the fitting of the Top panel and we set a Flag, 'ts' to deal with it in the script.

```
ts0='Bedroom style, inset'
ts1='Office style, overhang'
VALUES 'timstyl' ts0,ts1
IF timstyl=ts0 THEN ts=0 !Flag for timber
IF timstyl=ts1 THEN ts=1 !Detailing style
VALUES 'numdraw' 0,1,2,3,4,5,6,7,8,9,10
```

!Set Values for Width and Depth

```
VALUES 'A' RANGE [0.4,1.5] STEP 0.4,0.1
VALUES 'B' RANGE [0.3,0.9] STEP 0.3,0.05
VALUES 'zzyzx' RANGE [0.3,2.1] STEP 0.3,0.05
```

In ValueList, the RANGE function with Stepping allows the object to 'Snap' to convenient sizes.

!Cabinet with Drawers
!2D Script

Do the Hotspots first, including a pickup hotspot in the centre.

```
GOSUB 1000:!Hotspots for outline
HOTSPOT2 A/2,B/2 !Pickup spot
PEN gs_symb_pen
FILL gs_fill_type
POLY2 5,7,
0,0,1,
A,0,1,
A,B,1,
0,B,1,
0,0,-1
LINE2 0,bthk1,A,bthk1
```

END: !=====

1000:!Hotspots for outline

```
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,A,1+256
hsid=hsid+1 !Move
HOTSPOT2 A,0, hsid,A,2
hsid=hsid+1 !Vector
HOTSPOT2 -1,0, hsid,A,3
GOSUB 1010:!Hotspots Depth
ADD2 A,0
GOSUB 1010:!Hotspots Depth
DEL 1
RETURN
```

1010:!Hotspots Depth

```
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,B,1
hsid=hsid+1 !Move
HOTSPOT2 0,B, hsid,B,2
hsid=hsid+1 !Vector
HOTSPOT2 0,-1, hsid,B,3
RETURN
```

We want a 'filled' 2D symbol, so use a POLY2_ instead of a RECT2. The LINE2 helps to establish which is the back panel.

We want to permit free editing in A, but limit B to forward direction only. A stretches in 100mm increments, and B in 50mm increments.

Extend the Cabinet:

Subroutines, Drawers

WE need to develop the cabinet further. First, make the cabinet itself into a subroutine.

More Master Script work

We want to fit drawers into the cabinet so we need to calculate how much space there is in the internal volume. We need to subdivide this according to the number of drawers. Let's assume they are held by sliders and do not need separating wood bars. The smart object could also have more idiotproofing.

Later, we could have an option whereby if the user could choose zero drawers and opt for doors instead. There are future options to think about, such as drawer handles.

You need to think about small dimensions like drawer sliders, and a tolerance around each drawer.

```
!-----
!Additions to Master Script
!Calculate Drawer sizes
IF numdraw THEN
  drw_x=A-bthk1*2 !Drawer housing X width
  drw_y=B-bthk2 !Drawer housing Y Depth
  !Drawer housing Z Height
  drawrz=zzyzx-bthk1*2-plinh !Total height
  drw_z=drawrz/numdraw !Single drawer height
ENDIF

dt=0.002 !Tolerance of free space around drawers
ds=0.015 !Space for Drawer Sliders
```

Let your drawers float

The drawers have to be made as subroutines, and each of these is a smaller version of the cabinet – a box. The front is of the same wood veneer as the exterior, but the back, sides and bottom are different wood and thinner. The nominal volume of the space for each drawer can be sent to the subroutine which can then decide exactly how big the drawer is when constructed.

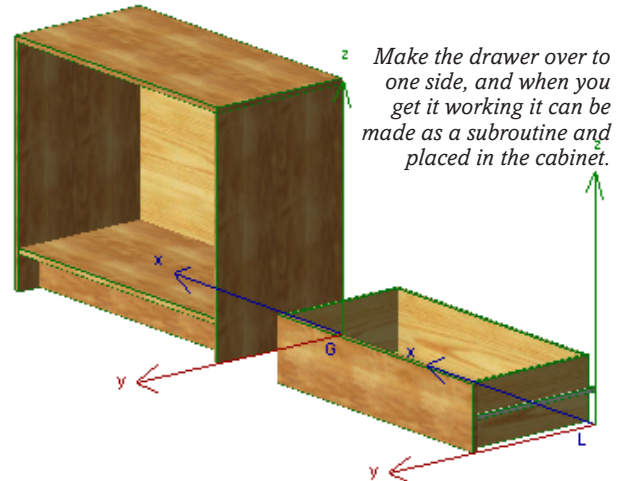
The drawers cannot touch each other or the cabinet, they must 'float in space' with at least 2 mm of tolerance around them – but the fronts need to be flush. It's a similar concept to Bricks *Actual* size versus Brick *Nominal* size (allowing for mortar joints). In addition, the sides must be inset by enough space to accommodate metal sliders. These are manufacturer's dimensions, so need not be parameters for the user, they can be in the Master Script. You need to use 'ts' again to vary the width of the drawer fronts.

Later we should convert the drawers to macros, in case the same manufacturer has other items of furniture which also use drawers (a certainty!).

3D Script

Build a prototype drawer unit a metre and a bit to the left of the cabinet. When it is working you can then use a loop to place them into the cabinet interior.

The drawer does not relate to A and B, it relates only to the drw_x, drw_y and drw_z that you have calculated for it in the Master Script.



```
!Cabinet with Drawers
!3D Script
PEN gs_cont_pen
GOSUB 100:!Cabinet main shape

!Prototype of single drawer
ADDx -1.5 !Temporary position

!Single Drawer
!Side Panels
MATERIAL con_mat
ADD dt+ds,dt+ds,bthk2
BLOCK bthk2,drw_y-bthk1-ds-dt,
      drw_z-bthk2-dt
DEL 1
ADD drw_x-dt-ds-bthk2,dt+ds,bthk2
BLOCK bthk2,drw_y-bthk1-ds-dt,
      drw_z-bthk2-dt
DEL 1
txdir=1:GOSUB 999:!Texture

!Back Panels
ADD dt+ds+bthk2,dt+ds,bthk2
BLOCK drw_x-2*(bthk2+ds+dt),
      bthk2,drw_z-bthk2-dt
DEL 1

!Bottom Panel
ADD dt+ds+bthk2,dt+ds+bthk2,bthk2
BLOCK drw_x-2*(bthk2+ds+dt),
      drw_y-bthk1-bthk2-dt-ds,bthk2
DEL 1

!Front Panel
MATERIAL cab_mat
ADD dt-bthk1*ts,drw_y-bthk1,dt
BLOCK drw_x-dt*2+bthk1*2*ts,
      bthk1,drw_z-dt*2
DEL 1
txdir=0:GOSUB 999:!Texture

!Sliders
MATERIAL met_mat
ADD dt,dt,(drw_z-ds)/2
BLOCK ds,drw_y-dt-bthk1,ds
DEL 1
ADD drw_x-dt-ds,dt,(drw_z-ds)/2
BLOCK ds,drw_y-dt-bthk1,ds
DEL 1

END:-----

100:!Cabinet main shape
!Side Panels
MATERIAL cab_mat
BLOCK bthk1,B,zzyzx-ts*bthk1 !left
ADDx A-bthk1
BLOCK bthk1,B,zzyzx-ts*bthk1 !Right
DEL 1 !etc. etc etc.
!THE REST OF THE CABINET SUBROUTINE IS
!THE SAME AS IN THE PREVIOUS PAGE
```

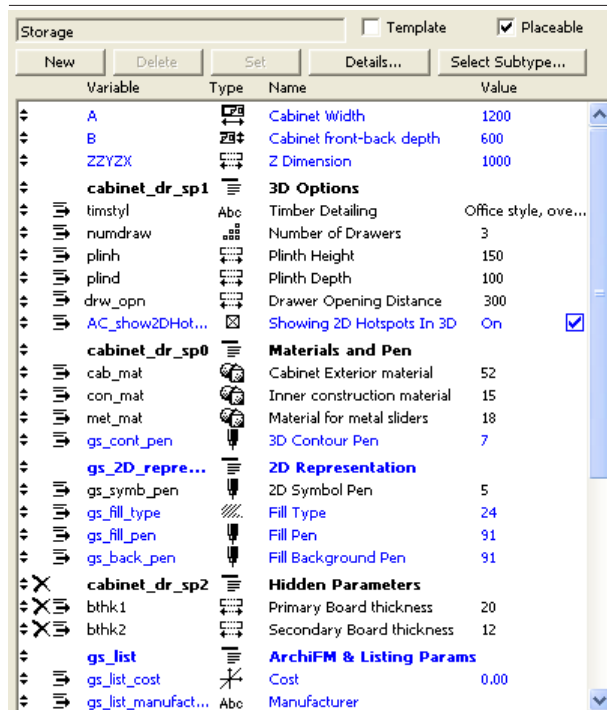
With the help of the sketch plan, you can bend your brain to work out all the 'dt' and 'ds' values to make sure everything fits.

It can be prototyped here, and then pushed down below to be a subroutine (reserve the label of 200 for the whole pack of drawers.)

Add in the Texture GOSUBs for the true authentic woodgrain look. Note that 'ts' makes the drawer front wider.

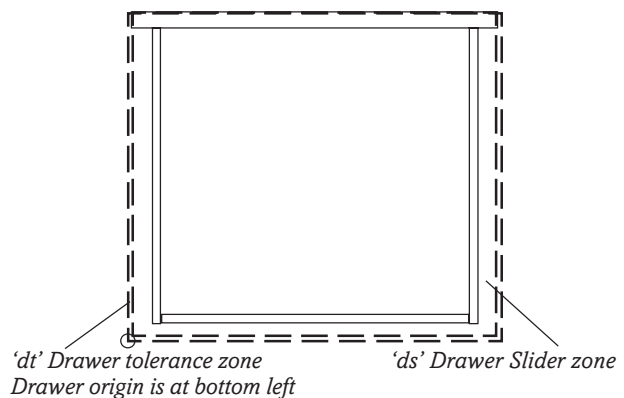
You can break the line (carriage return) if you have a comma in the statement.

Sliders do not need texture if they are of one colour.



We add in also an Drawer Opening Distance 'drw_opn' at this stage, for future use.

Drawer Front



Draw out the drawer!

You MUST draw out a sketch plan of the drawer to understand how the spaces around it for the tolerance and the sliders will work.

Organise the parameters

We need to improve the organisation of the Parameters Table. Make the board thicknesses 'hidden' so the user cannot change them. Add a material for sliders, and make some titles so that parameters can be organised into cascading groups.

Put the drawers in

WE can write a **FOR...NEXT** loop to position the drawers in the cabinet. Because each drawer 'floats' in its own tolerance space, we can safely work to the exact interior corner point of the cabinet. We put the drawer into a subroutine before writing the loop.

In the long run, you may be wanting to generate drawings of the panels with texture orientation, so we can smarten up the texture subroutine in the 3D Script using the texture-direction flag 'txdir', 0 for X-horizontal, 1 for Y-horizontal, 2 for Z-vertical. The rotation of the texture is done inside Subroutine 999.

Master Script

```

!-----
!more Additions to Master Script
IF drw_opn>=drw_y*0.8 THEN drw_opn=drw_y*0.8
IF drw_opn<=0 THEN drw_opn=0
PARAMETERS drw_opn=drw_opn
Limit the drawer
travel to 8/10ths of
the depth

!Autosizing drawers
IF drawrz<0.05 THEN numdraw=0
IF numdraw THEN !Only do it if there are drawers
WHILE drw_z<0.05 DO
numdraw=numdraw+1
IF numdraw THEN drw_z=drawrz/numdraw
ENDWHILE
ENDIF
PARAMETERS numdraw=numdraw

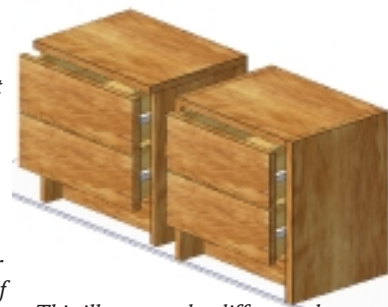
```

We put in limitations to the movement of the drawers. We could use a ValueList here as we did for the cabinet dimensions, but this would not work so well when we use Parameter Arrays for the drawers.

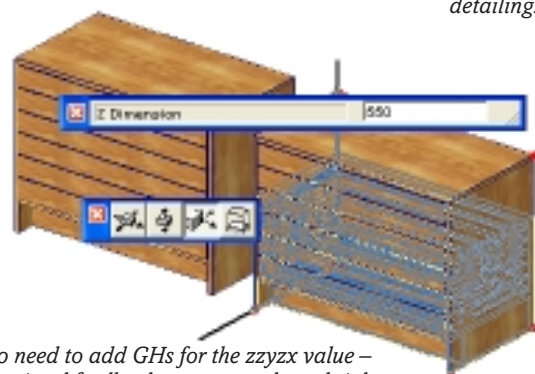
Autosizing Drawers with WHILE DO

We also want to prevent the drawers being built if they will be less than 50mm (2") high. We can autosize the

The drawers sit perfectly within their tolerance space (give them 3mm initially just to make sure it's working correctly). However, having all the same opening distance is a bit ludicrous. How does one make each drawer individually capable of being pulled out? It's time to think about **Parameter Arrays**.



This illustrates the difference between the 'Office' and the 'Bedroom' styles – it's 'overlap' versus 'flush' detailing.



We also need to add GHs for the zzyzx value – gives us visual feedback as we stretch or shrink. **WHILE DO** gives us a routine to calculate whether to reduce the number of drawers if the cabinet height is reduced.

drawers and reduce the number if we see that they are going to be smaller than 50mm. This is an ideal demonstration of a **WHILE... DO... ENDWHILE** loop. If the number of drawers is reduced by one each time, we must not allow a 'divide-by-zero' error, so 'numdraw' is checked **TWICE** in the loop. **WHILE** Loops are dangerous, there's a chance that GDL will never escape the loop – when you try this, there's a fair chance you will be Alt-Ctrl-Dell ing before long!

Extend the Cabinet:

Loop & Hotspots

3D Script

PUT the Drawer into a subroutine, and it's an easy job now to make a FOR NEXT loop based on the number of drawers – the size and spacing of each have been calculated in the Master Script. Add in some hotspots for the vertical Z-heights.

We need to add push-pull hotspots for the drawers, but we need a knob otherwise there is nothing to grab on to – a hotspot hovering in the centre of a flat drawer front would be too intangible. For the moment we put a simple cylindrical knob there – once it's in a subroutine (220) it can be replaced later with a parade of nobly elegant or knobbly handles. The hotspot routine has to be lifted to the right height for each drawer,



but it does not move in the Y direction like the drawers and knobs do – the Base spot remains at the inside face of the back panel.

The routine for the drawers is given a number 200. Guess what! it's going to be a subroutine too!

```
!Cabinet with Drawers
!3D Script
PEN gs_cont_pen
GOSUB 1000!!Hotspots Vertical
GOSUB 100!!Cabinet main shape
```

```
200!!All Drawers
IF numdraw THEN
ADD bthk1,bthk2,plinh+bthk1
FOR k=1 TO numdraw
ADDz drw_z*(k-1)
GOSUB 1020!!Hotspots Drawer knobs
ADDy drw_opn
GOSUB 210!!Single Drawer
GOSUB 220!!Knob for Drawer
DEL 2
NEXT k
DEL 1
ENDIF
```

```
END:!-------
```

```
100!!Cabinet main shape
RETURN

210!!Single Drawer
RETURN

220!!Knob for Drawer
TOLER 0.002
MATERIAL cab_mat
ADD drw_x/2,drw_y,drw_z/2
ROTx -90
CYLIND bthk1,bthk1
DEL 2
RETURN
```

Both of the Z-hotspots can be stored in a subroutine.

Move to the inside back corner of the Cabinet first before the loop starts.

There is one hotspot routine for all drawers at the moment, each one is a subroutine inside the loop.

The reason for giving the loop a label of 200: is that it can also become a subroutine.

```
1000!!Hotspots Vertical
GOSUB 1010!!Hotspots Vertical
ADDx A
GOSUB 1010!!Hotspots Vertical
DEL 1
RETURN
```

```
1010!!Hotspots Vertical
hsid=hsid+1 !Base
HOTSPOT 0,0,0, hsid,zzyzx,1
hsid=hsid+1 !Move
HOTSPOT 0,0,zzyzx, hsid,zzyzx,2
hsid=hsid+1 !Vector
HOTSPOT 0,0,-1, hsid,zzyzx,3
RETURN
```

```
1020!!Hotspots for Drawer knobs
ADD drw_x/2,drw_y,drw_z/2
hsid=hsid+1 !Base
HOTSPOT 0,0,0, hsid,drw_opn,1+128
hsid=hsid+1 !Move
HOTSPOT 0,drw_opn,0, hsid,drw_opn,2
hsid=hsid+1 !Vector
HOTSPOT 0,-1,0, hsid,drw_opn,3
DEL 1
RETURN
```

The Knob subroutine is a temporary fix. Other, more exotic knobs can be provided later.

Notice that we economize by writing only one vertical hotspots routine, and move it to the two rear corners of the cabinet.

The drawer opening hotspots work from a base at the back inside face of the cabinet.

Because we are using drw_opn for each drawer, they all pull out together – it's time for Parameter Arrays.

Deliberate Errors

COMPUTERS work in Machine Code, we work in the world of elementary applied Maths. Humble earthlings use feet and inches, millimetres or metres. But between human and machine, there may be conversion stages like Decimal and Hexadecimal and Binary. There is an extra stage if you are converting from imperial to decimal dimensions.

Sometimes when you divide two numbers, or use the INT() command, you get tiny errors as a result of this conversion. For example 1/3 produces a recurring number in decimal. INT(9/3) could produce a result of 2 instead of 3 due to tiny conversion errors. FOR k=0 TO 10 STEP 5/2 should produce 4 intervals and 5 locations; but this could

result in one of them being lost due to a tiny error in the division, resulting in underflow: not meeting the target.

Using a tiny unit like a millimetre or 1/25", build in a deliberate error to minimise the risk of underflow. If you write FOR k=0 TO 10+0.001 STEP 2.5, you will eliminate the conversion error, but you will not alter the model at all. I noticed recently that if you divide 6'-0" by 3'-0" you get an error. If you divide by 6'-0" by 2'-11.9999" (yes you can use decimals) you guarantee getting a result of 2. If a piece of GDL simply doesn't work when it should, look for the opportunity to feed in a deliberate error.

With rotational loops, add a single degree to prevent underflow, e.g., FOR angl=0 TO 331 STEP 30: NEXT angl.

Extend the Cabinet: Parameter Arrays

Hotspots without Arrays: nightmare!

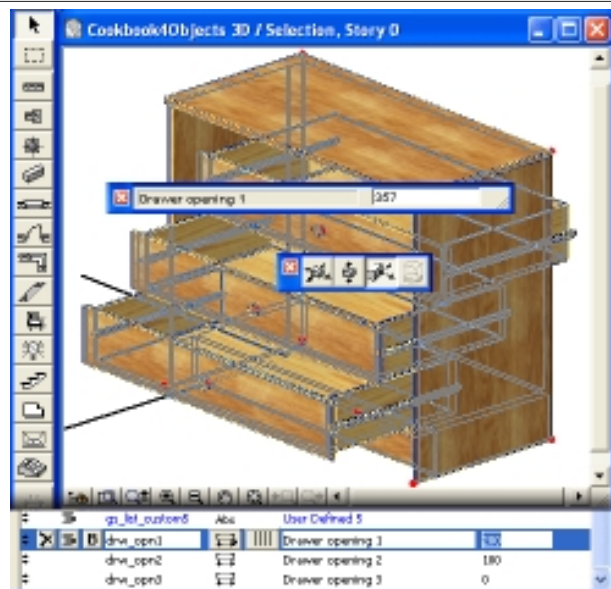
WITHOUT Arrays, the only way to write hotspots for the ten drawers in this cabinet is to have some very verbose code. You will need to make a parameter of opening distance for EACH one, and write a complete hotspot routine for EACH one. Instead of a neat loop, you have to write a cluster of IF statements deciding whether to GOSUB each of those routines. Very tedious. Actually, I encourage you to try writing this way for say the first 3 drawers, just to see how it might be done. The script on this page shows you the solution for the first 3 drawers. You have to make 10 more parameters like `drw_opn1`, `drw_opn2`, `drw_opn3` and so on. You need 10 lengthy IF statements, and you cannot use a nice friendly efficient Loop. It's tedious to write an individual routine stopping the drawers coming out into free space or pushing through the back panel. Supposing you have an object with 50 or 100 movable objects, like curtain wall mullions – how many IF statements are you prepared to write?

Explaining Parameter Arrays

Parameter Arrays were thought so complex and unusual requirements that they didn't rate a mention in GDL Cookbook 1, 2 or 3. However if you have an object that combines **Loops** with AC8's **Graphical Hotspots**, it has become clear to me that **Parameter Arrays become transcendently important** – as the example on this page proves. They make an elementary object like this cabinet behave like something made by an expert.

With Parameter Arrays, we can write ALL the drawer-opening hotspots for any number of drawers in ONE 6 line routine! We can initiate PAs by hitting that little square button next to the parameter icon. Look at the picture below – every newcomer to GDL hits that pesky little button at some time by accident, and reverses out rapidly in a panic!

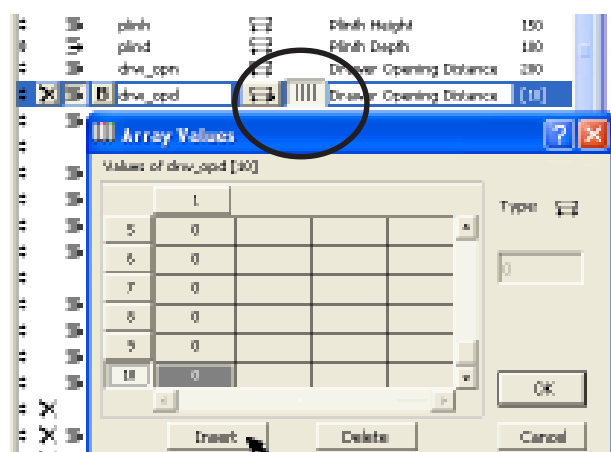
Arrays, in a nutshell, are Lists, or they can be 2D Grids like a spreadsheet. They live with a single name for the whole array, so that any cell in the array can be recalled with an easy grid reference. They can contain numbers (mostly) or strings (occasionally) – whichever of these, they are always referenced by a number. If the array is called 'dd' the 7th one in the list is a parameter called `dd[7]` – we use square brackets for arrays. The prime benefit of arrays is that with a numbering system for the parameter, we can now write FOR NEXT loops to make use of them, using a counter like 'k'. Thus if we want to do something with `dd[k]`, it works! 'Parameter Arrays' are ones that start life in



```
!Drawers without a loop
ADD bthk1,bthk2,plinh+bthk1
IF numdraw>=1 THEN
  ADDz drw_z*0
  GOSUB 1101:!Hotspot
  ADDy drw_opn1
  GOSUB 210:!Single Drawer
  GOSUB 220:!Knob for Drawer
  DEL 2
  ENDIF
IF numdraw>=2 THEN
  ADDz drw_z*1
  GOSUB 1102:!Hotspot
  ADDy drw_opn2
  GOSUB 210:!Single Drawer
  GOSUB 220:!Knob for Drawer
  DEL 2
  ENDIF
IF numdraw>=3 THEN
  ADDz drw_z*2
  GOSUB 1103:!Hotspot
  ADDy drw_opn3
  GOSUB 210:!Single Drawer
  GOSUB 220:!Knob for Drawer
  DEL 2
  ENDIF
DEL 1
END:!----
1101:!First drawer, the rest are similar
ADD drw_x/2,drw_y,drw_z/2
hsid=hsid+1 !Base
HOTSPOT 0,0,0,hsid,drw_opn1,1+128
hsid=hsid+1 !Move
HOTSPOT 0,drw_opn1,0,hsid,drw_opn1,2
hsid=hsid+1 !Vector
HOTSPOT 0,-1,0,hsid,drw_opn1,3
DEL 1
RETURN
```

Do not attempt to type this in if you are following the exercise: this is another example of BAD GDL: a complex sequence of IF statements and repetitive Hotspot subroutines.

If this convinces you to use Parameter Arrays, then read on.....



Keep hitting the insert button till you have ten

Parameter Arrays, continued

the Parameter Table, as in this example. It is also possible to have 'Dynamic Arrays', ones that are initiated and defined in the Master Script, not in the Parameter Table. These will be covered later in the book. When you use a practical example, as in this cabinet, you will rapidly see how to use them, and appreciate how powerful and simple they are.

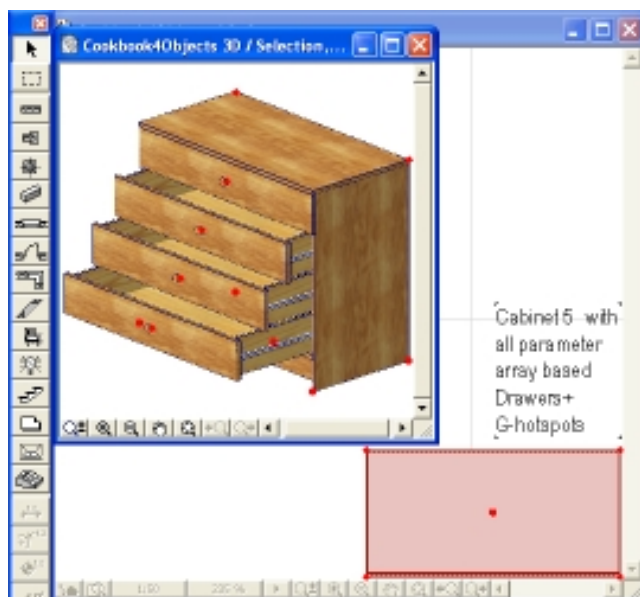
I never found a use for them in earlier ArchiCADs but now find that almost every object with Loops and GHs demands them. Hence their inclusion in this 'beginners-level' section of the Cookbook.

Make a Parameter Array

So where we had 'drw_opn', let's make a new parameter called 'drw_opd' for the opening distance. Position it below the old parameter, as above. Hit the *square button* and when the next dialog appears, click first on one of the *side buttons*. Then hit the *Insert* button till you get 10 items in the list. They will all have a starting value of Zero, but don't let that bother you – they are all values for the drawer-opening distance. Close the dialog, and you return to the Parameter Table. You now have 10 items in the Array called 'drw_opd'. You can no longer use the name 'drw_opd' on its own. From now on, you have to refer to all mentions of this parameter as drw_opd[1], drw_opd[2] and so on. Once we have modified the Master and 3D Scripts we can delete the obsolete parameter 'drw_opn'.

Use the 'Set' Button

If you wish to open the object in GDL to edit the contents of the array grid, you can only do so by hitting the 'Set' button. You could enter default values for the drawers. But for now, you can leave them all at zero and let the user to play with them using the Hotspots. When the object is in ArchiCAD, the user can easily change array values by clicking on the parameter.



For now, you can write in the Master Script a routine that will write in the values for you to this grid using the PARAMETERS command. Use this routine to replace the former simple routine to limit the extent of the drawer pulling.

```
!Modify existing Master Script
!Limit extent of drawers - collision control
FOR k=1 TO 10
IF drw_opd[k]>=drw_y*0.8 THEN drw_opd[k]=drw_y*0.8
IF drw_opd[k]<=0 THEN drw_opd[k]=0
PARAMETERS drw_opd[k]=drw_opd[k]
NEXT k
```

3D Script

The modifications to the 3D Script are stunningly simple and effective. The Drawer routine now becomes a subroutine, the drawer opening distance is simply replaced by an array parameter 'drw_opd[k]' in the drawer subroutine and in the Hotspot sub routine.

```
!Cabinet with Drawers
!3D Script
PEN gs_cont_pen
GOSUB 1000:!:Hotspots Vertical
GOSUB 100:!:Cabinet main shape
GOSUB 200:!:All Drawers

END:!-------

100:!:Cabinet main shape
RETURN

200:!:All Drawers
IF numdraw THEN
ADD bthk1,bthk2,plinh+bthk1
FOR k=1 TO numdraw
ADDz drw_z*(k-1)
GOSUB 1020:!:Hotspots for Drawer knobs
ADDy drw_opd[k]
GOSUB 210:!:Single Drawer
GOSUB 220:!:Knob for Drawer
DEL 2
NEXT k
DEL 1
ENDIF
RETURN

210:!:Single Drawer
RETURN

220:!:Knob for Drawer
RETURN

999:!:Texture
RETURN

1000:!:Hotspots Vertical
RETURN

1010:!:Hotspots Vertical
RETURN

1020:!:Hotspots for Drawer knobs
ADD drw_x/2,drw_y,drw_z/2
hsid=hsid+1 !Base
HOTSPOT 0,0,0,hsid,drw_opd[k],1+128
hsid=hsid+1 !Move
HOTSPOT 0,drw_opd[k],0,hsid,drw_opd[k],2
hsid=hsid+1 !Vector
HOTSPOT 0,-1,0,hsid,drw_opd[k],3
DEL 1
RETURN
```

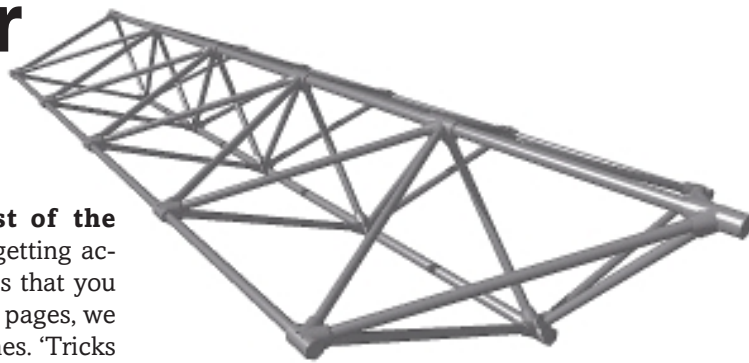
The drawer subroutine is now tidily put in a safe place after the END statement.

Change the parameter for the drawer opening to the new array parameter in this routine and in the Graphical Hotspot routine below.

Parameter Arrays are a general term for the whole thing, a single one is an Array Parameter'.

Maths Primer

Functions



THE Master Script is where most of the 'thinking' takes place. It is worth getting acquainted with some of the Maths functions that you need to help your thinking. On the next few pages, we also look at Trigonometry, Circles and Arches. 'Tricks of the Trade' is perhaps the most often visited section (by me) in the GDL Cookbook.

ABS(number) returns a positive value of a number, regardless of whether it is negative or positive. You can prevent the user entering a negative parameter with a statement like: `item=ABS(item)`.

SGN(number) returns a 1 if the number is positive, or a -1 if the number is negative. Small token numbers like this which signify the status of a number are called 'flags', in programming jargon.

RND(number) returns a random number value between 0.0 and 1.0 and is multiplied by the number in the bracket, e.g. `RND(50)` gives you numbers from 0 to 50, `RND(0.1)` is from 0.0 to 0.1. RND numbers are Real Numbers, so can only be used to generate Dimensions if you are using Metres, unless you convert them.

Unfortunately, GDL's random number generator is weak, and generates the same sequence each time from a fixed list, with a preponderance of low numbers. The only way to get a more nearly random number is to get the list to start from anywhere but the start. In animations, you could use the Frame number as a randomisation seed. One method that seems to work well is: `nul=RND(GLOB_INTID)`.

INT(number) returns the integer of the number – an integer is a round number (without decimals). The INT command ALWAYS rounds downwards, so you can do `x=INT(number + 0.5)` to get it to round up OR downward correctly.

Example: `INT(8.56)=8` ! (rounds down)
`INT(8.56+0.5)=9` ! (rounds up)

FRA (number) is the other side of INT(). It returns the fraction remaining when you remove the integers. Unfortunately it behaves always in one direction, so `FRA(6.3)` returns 0.3, but `FRA(-6.3)` returns 0.7.

CEIL (number) is similar to INT, collects the nearest integer number larger than the number.

SQR(number) returns the square root of the number. This is useful in matters like the use of Pythagoras Theorem in Triangles, and Circle geometry. If you need Cube and Quad roots, use the 'Power' symbol – e.g. `c=length**(1/3)`: `q=width**(1/4)`.

PI is useful in circle geometry – it is exact, and therefore better than trying to put in your own number, such as 3.1416, and incurring errors in a calculation.

MIN() and MAX() are useful – they simply tell you the highest or lowest of a series of numbers or variables that you print in the brackets, using commas. For example: `ADDz MAX(leftht, rightht)` raises the cursor to the level of the highest of the two heights of a furniture monopitch end panel.

NOT() is useful in IF statements. You could write `IF thing=0 THEN GOSUB 100`, but it is more like english to write `IF NOT(thing) THEN GOSUB 100`. Also because NOT() either returns a value of 1 or zero, you could use it as a flag, as in:

`LET rotation_angle= 30*NOT(noturn).`

When 'noturn' is valid, the angle would be zero and if 'noturn' is zero, then the angle will be at 30degrees.

AND (&) and OR (|) and EXOR (@) are used as arguments in IF statements.

For example, you could say:

`IF item>4 AND item<6 THEN GOSUB 500.`

This would certainly identify a floating point number in the region of five. With OR, either or both statements being true will produce an action, e.g.

`IF item<4 OR item>64 THEN GOSUB 500.`

Usually, if you say it to yourself slowly in English, and digest the natural logic of the AND and the OR argument, you will get it right.

EXOR is an exclusive OR statement, meaning that if one or other are true, do the action, but if both are simultaneously true or false, do not do it. EXOR is less often used, but mostly because people do not understand it. Circumlocution is possible if you do not wish to use it.

If you are comparing something with zero, you can use a shortened form not requiring the 'equals' sign, e.g. `IF circlr THEN GOSUB 500 ELSE GOSUB 550`. 'circlr' is a flag with a value of 1 or zero. In effect, do the action if 'circlr' is any value but zero.

EXP(), LGT(), LOG() are 'transcendental' functions, to do with 'e' and logarithms. These are rarely used, e.g. in calculating Catenary curves. `EXP(x)` calculates the value of e^x . ('e' is a magic number 2.7182818). `LOG()` return the natural logarithm and `LGT()` the logarithm to base 10.

MOD is a way of finding the remainder after a division e.g. `8 MOD 2` equals 0 because 2 divides perfectly into 8. `8 MOD 3` equals 2 because if you divide 8 by 3 it goes twice and leaves a remainder of 2.

Trigonometrical functions

SIN(), COS() and TAN() – Trigonometry functions that return you the Sine, Cosine and Tan of the angle. **ATN(), ASN(), ACS()** mean respectively ArcTan, ArcSin and ArcCos. This means that if you know the sides of a right angle triangle, you can easily work out the angles. For ArcTan: ATN(0.231) will return you the Angle which, if you calculated TAN(angle) would give you a TAN of 0.231. This is essential if you are building lattice trusses.

Functions like COSH, SINH, SEC, COSEC can be derived from these and from EXP().

Common maths symbols

<>	Not Equal to (you can also use #)
>	Greater than, < Smaller than
>=	Great than or Equal to
<=	Smaller than or Equal to
^	To the power of... (also **)

Declaring variables : number or string

All numbers in GDL are 'Floating Point', you cannot force a variable to be an integer.

You can force a variable to be a string – this contains text. Once a variable is a string, you cannot not change it back to a number and *vice versa*. The script below is likely to produce a syntax error that will baffle you – but the error has been caused by a previous use of 'n' as a string.

```
LET n="washbasin"
FOR n=1 TO 5
  GOSUB 100
NEXT n
```

Eats, shoots and leaves? Use Brackets

Multiply and Divide stick things together, Plus and Minus separate. What does 300/5*6+10 equal? If in any doubt, use Brackets.

e.g. SPHERE_VOLUME = (4/3)*PI*(R^3)

Strings and things

WHAT is a String? It is a sequence of alphanumeric characters which have no numerical value – so it's in quote marks.

In GDL, a set of alphanumeric characters on their own (not in quotes) are either a **Parameter** or a **Command** (not a string) and have to be doing something, or you get an error. A set of alphanumeric characters behind an exclamation mark are a **Comment**.

A set of alphanumeric characters in quote marks are a **String**. '123.456' is a string because it is in quotes.

Materials (and other attributes) can be named by a string – you can say MATERIAL 'Whitewash'.

Strings can be used as a parameter or variable, e.g. the ABC parameter type contains a string, frequently used for Popdown menus. e.g. timstyl= 'Office style'.

TEXT2 in 2D scripting can print a string or a number but not both. If you wish to combine strings and numbers, then you have to convert the numbers to strings using STR().

STR() is very useful in 2D labelling – when you need to combine many numbers together to make a long string of letters – it often happens. For example, you can use it when you wish to make an object self labelling. If you have a stretchy staircase, you can get it to display a string like: 'Pitch= 38.462 degrees'. We did it for the CircleGrid earlier.

STR() needs to have an idea of how many digits will be needed to represent an entire number, and how many of those digits will occur after the decimal point. For example: item=123.456

string=STR(item,6,3) gives you '123.456'

string=STR(item,6,1) gives you '123.5'

string=STR(item,4,3) gives you '123.456'

string=STR(item,7,2) gives you '123.46'

To concatenate strings and numbers, you might use the addition symbol and join the strings, as in:

```
lenstr="Length=" + STR(item,6,2)
TEXT2 0,0, lenstr
```

and get a string variable called lenstr.

x=STRLEN('hello') returns a value of 5, the number of letters.

x=STW('hello') gives you a dimension in metres equal to the length of the word, in its present text style. Allowing you to build a 2D box round it, or to get a good results if labelling an object where you want the label to fit neatly to the object, whatever the drawing scale.

SPLIT(): This is an interesting command that convert part of a string back into numbers – complicated to use, but sometimes necessary. See the 'Bendibar' for an example. It half-does what VAL() does in BASIC. To me, its most useful purpose has been rapid parsing of Value List answers – reducing the need for many IF statements.

```
!Value List definition for a Stair Handrail
VALUES 'handtyp' '0-None', '1-Parallel rails',
        '2-Rail + spindles'
```

```
!Parse the type of Handrail
x=SPLIT(handtyp, '%n',htyp)
!htyp is the flag for the rail
```

As a result of this, htyp is equal to either 0, 1 or 2.

STRSTR(): Returns the value of the position of one string inside another.

e.g. x=STRSTR('Whitewash', 'wash') returns a value for x of 6. If nothing is found, x is zero.

STRSUB(): returns a string, extracted from a longer one. Use this for 'parsing' strings. e.g.

```
myname= 'David'
FOR k=1 TO STW(myname)
  TEXT2 0,k*0.3, STRSUB(myname,k,1)
NEXT k
```

This comes up with each letter of my name arranged vertically.

Maths Primer: Triangles & Circles

GDL requires a 3D spatial awareness, and an essential minimal knowledge of Triangle and Circle Geometry. Architects usually have no problems with the former, but may need help with the latter.

Let us assume that you are familiar with basic Arithmetic and Algebra. At this stage in your GDL learning curve, you have a knowledge of BASIC. However, there are 3D problems which require considerable analysis before you begin, and I find these pages useful for myself when tackling new problems. Sketch onto paper the essential geometry of the problem. Then plan the solution.

Nostalgia trip

Suddenly all those distant memories of Maths you did back at school can be dusted off and made useful.

Isn't it marvellous!!

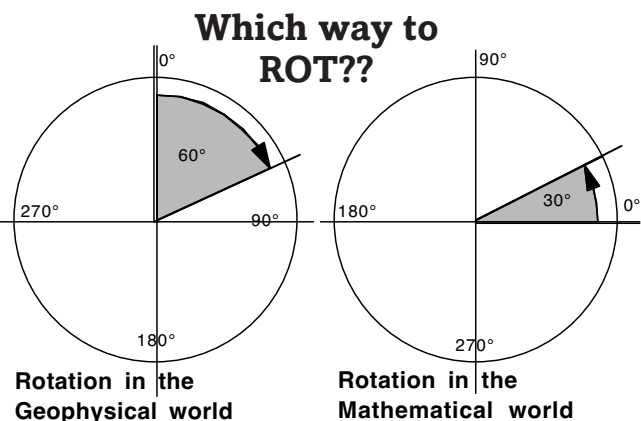
Rotation Directions

THE Geophysical world works with the **Vertical** being Zero° North at the top; East (90°) is in a Positive direction clockwise. West is either -90° or +270°, depending on how you travel there. The hands of a clock work that way.

In Mathematics, the **horizontal** is Zero°, and an angle grows in the anti clockwise direction, so that 90° is facing up the page. This is how the diagram for the camera works in 3D Projection Settings, in ArchiCAD.

In GDL, You can use angles in either the mathematical or geophysical sense, as long as you remain consistent in your interpretation but the Maths method works best.

When you talk about the azimuth of the Sun relative to the site and entering site survey information, this causes confusion. Tread Carefully.



Triangles

TRIGONOMETRY can be easy: you need a basic knowledge of the right angle triangle – in fact all you need to know is the motto **SOH CAH TOA** so that you can always calculate the characteristics of a triangle from minimal information. If you don't know this already, learn it by saying it to yourself as a mantra whilst in traffic queues.

The angles of any triangle add up to 180°.

Coordinate Geometry is based on Pythagoras. The distance between any two points in 2D is simply $\text{SQR}((x1-x2)^2 + (y1-y2)^2)$. This works even when points have negative values. The gradient 'm' of a line is $=(y1-y2)/(x1-x2)$ – same as the TAN of its angle.

The formula for a straight line is $y=mx+c$, or $ax+by+c=0$. If two lines intersect (straight or curved), you can derive a simultaneous equation – both 'y' values equal each other.

The gradient 'm2' perpendicular to a line of gradient 'm1' equals $(-1)/(m1)$ – and you can say that $m1*m2=(-1)$.

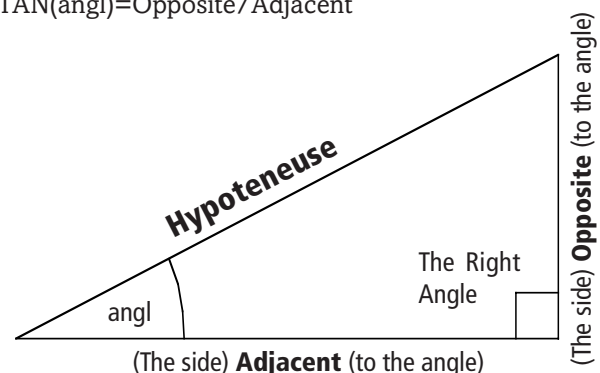
Right Angle Triangles

Remember: 'SOH CAH TOA'

$\text{SIN}(\text{angl}) = \text{Opposite} / \text{Hypoteneuse}$

$\text{COS}(\text{angl}) = \text{Adjacent} / \text{Hypoteneuse}$

$\text{TAN}(\text{angl}) = \text{Opposite} / \text{Adjacent}$



Pythagoras him say (many years ago): Hypoteneuse squared equals sum of the squares of the other two sides. In other words, $\text{Hypoteneuse} = \text{SQR}(\text{Adjacent}^2 + \text{Opposite}^2)$

Consult these pages! You are now looking at the set of pages that I use myself most frequently. The Maths Tips and Tricks are useful, and some of the formulae for circular and parabolic curves are only found in this book.

Irregular Triangles

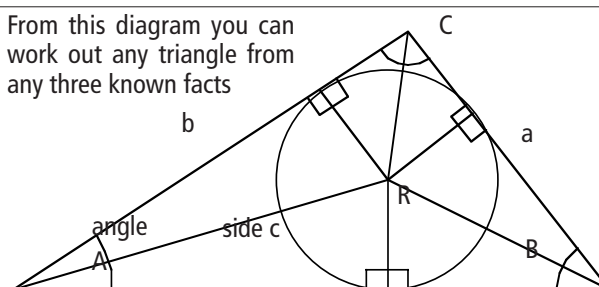
IRREGULAR triangles can be analysed by breaking them into small right angled ones and using SOH CAH TOA. The SIN and COS rules enable you to derive all the facts from any three pieces of information.

Powers can be written with two asterisks e.g. num**2, or a caret as num^2 and square root is SQR(num) or num^0.5. Cube and other roots would be written as num^(1/3).

Any three points in space, no matter how irregularly disposed can be joined up to form a planar (flat) triangle; a planar circle can be drawn through the points. ArchiCAD's circle tool provides a means of finding this centre. The calculation is more difficult.

An Isosceles triangle has two equal sides and two equal angles. The angles of any triangle add up to 180°.

From this diagram you can work out any triangle from any three known facts



A circle can always be inscribed in an irregular triangle.

From this, several right angle triangles can be derived.

SIN Rule: $a/\sin(A)=b/\sin(B)=c/\sin(C)=2*R$

COS Rule: $a^2=b^2+c^2-2*b*c*\cos(A)$

Hence, if you know the sides,

Angle $A=\arccos((b^2+c^2-a^2)/(2*b*c))$

Area=(length of any side)*(height relative to that side)/2

3D Pythagoras

ONE fact that I worked out for myself (it must be in a book somewhere, but which?) and which I have used for some time, is Pythagoras' Theorem applied to 3D. The diagonal of a cuboid shape is the Square Root of the sum of the squares of the Length, Width and Height. $\text{diaglen}=\text{SQR}(x^2+y^2+z^2)$.

The script here demonstrates it. Thus if you have a linear object arcing around in 3D space (like the arm of a crane) you know its length and its declination angle from the simple formulae demonstrated here. (Declination is the elevational angle from vertical).

Reverse engineer

From this theorem, you can work out the reverse of the 3D Pythagoras, using Trig. If you know the crane arm's length, its Rotational angle, and its Elevational angle (from horizontal), you can know its X,Y,Z location in space. In this diagram, 'rotang' is the rotational angle of the arm from the Y axis, and 'elvang' is the elevational angle from flat on the floor. 'diaglen' is the length of the Cylinder. Hence, these quick formulae:

3D Pythagoras Demonstration

MODEL wire

PEN 1

x=1: y=2: z=3

BLOCK x,y,z

!Calculate length of diagonal
 $\text{diaglen}=\text{SQR}(x^2+y^2+z^2)$

!Azimuth=ATN(y/x)

!Elevation=ACS(z/diaglen)

ROTz ATN(y/x)

ROTy ACS(z/diaglen)

CYLIND diaglen,0.02

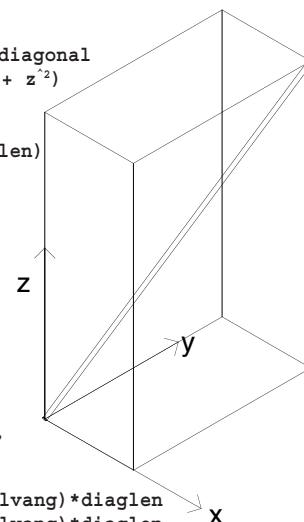
DEL 2

Warning: Azimuth calculation doesn't always work well in the negative quadrants. Objects may become mirrored by ArchiCAD, and disobey the calculation.

- $X=\sin(\text{rotang})*\cos(\text{elvang})*\text{diaglen}$
- $Y=\cos(\text{rotang})*\cos(\text{elvang})*\text{diaglen}$
- $Z=\sin(\text{elvang})*\text{diaglen}$

Distance between any two points in space:

$\text{dist}=\text{SQR}((x1-x2)^2+(y1-y2)^2+(z1-z2)^2)$



Maths Primer: Tricks of the Trade

YOU frequently need to 'toggle' or convert numbers and flags especially when Boolean choices are involved.

- To find out if a number is odd or even :-
IF INT(item/2)=item/2 THEN even=1 ELSE even=0
or IF item MOD 2=0 (it is even)
- To find out if number is Negative or Positive (& set a flag):
IF item<>0 THEN flag=item/ABS(item) ELSE flag=0
(makes 1 or -1, or 0 if item is zero)
- To force a number to be positive: item=ABS(item)
- To force a number to be nearest even number
item=2*INT(1+item/2)
- To generate 1 & zero alternately in a loop (stepping 1)
flag=2*FRA(k/2)
- To make any number into 1 (if real) or 0 if zero.
IF item<>0 THEN flag=item/item ELSE flag=0
- To round a number Up or Down to nearest integer,
newvalu=INT(0.5+item)

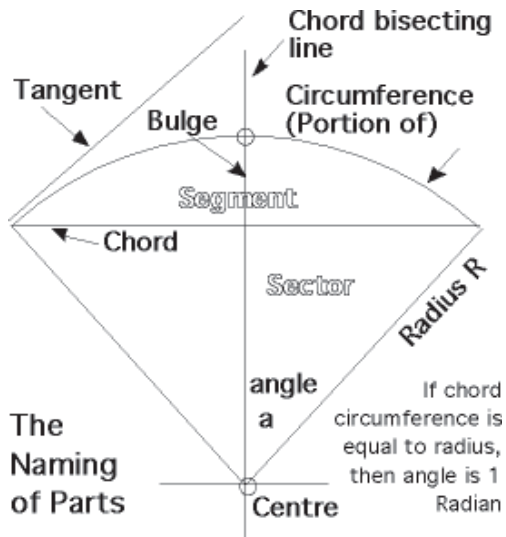
Fiddling with Flags

- To turn 1 & 2 into 0 & 1 (item-1)
- To turn 1 & 2 into -1 & +1 (item-1)*2-1
- To turn 0 & 1 into -1 & +1 item*2-1
- To turn 0 & 1 into +1 & -1 (item*2-1)*(-1)
- To toggle values of -1 & +1 item*(-1)
- To generate 1 or 0 randomly INT(RND(2))
- To toggle values of 0 & 1 (1-item)
- To make 0=0, and 1,2,3 etc=1 INT(x/(x-0.1))
- To make 1 & -1 into 0 & 1 -(item-1)/2
- To make -1 & 1 into 0 & 1 (item+1)/2

- To round a long number to a set no. of decimals:
number=INT(0.5+number*(10^decmls))/(10^decmls)
- Make an angle that is below zero (e.g. -26°) show as minus 26°, not 336°:
IF angl>=180 AND angl<360 THEN angl=angl-360
PARAMETERS angl=angl

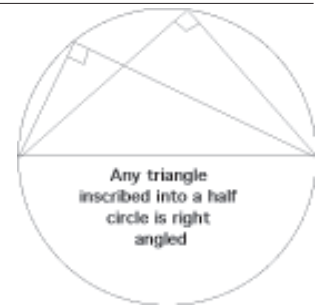
Circles

THERE will be many situations where the object you are making is circular (curved truss), or parts of it trace a circular path through space (a swinging arm). In coordinate geometry, the Formula for a Circle is $R^2 = x^2 + y^2$. There are 360 degrees in a circle. But you can safely issue commands like SIN(angl) when angl is a number like 535° (greater than 360) and GDL will not turn a hair (it will return a result of SIN(175)).



The diagram displays the terminology of parts of circles used in many of the worksheets in the CookBook.

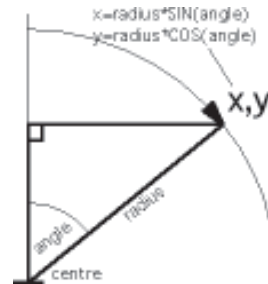
All triangles can be drawn into a circle. It is useful to know that in a right angle triangle, the hypotenuse is also the diameter of an enclosing circle, and its half way point is the Circle centre.



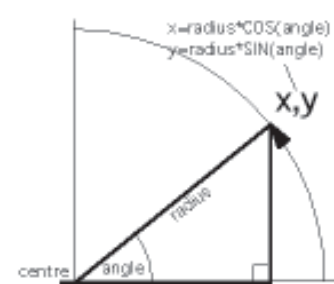
A circle's circumference is $2\pi R$, or $\pi \times \text{diameter}$.

A circle's area is πR^2 .

Sphere's volume is $(4/3)\pi R^3$



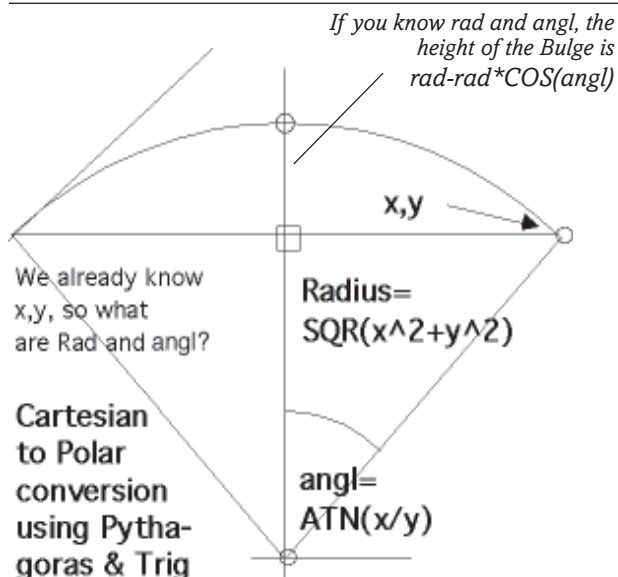
If you are starting from the vertical, moving clockwise



If you are starting from the horizontal, moving anti-clockwise

Polar to Cartesian conversion

If you are distributing things in a circle (e.g. mullions in a bay window), you usually know the centre, the radius and the angle in which you are pointing.



Above: You may know the X and Y location, but need to know the Radius and Angle – perhaps you have a diagonal cylindrical tube which must connect the two points. Only works if 0,0 is the centre.

These formulae above can be juggled about.

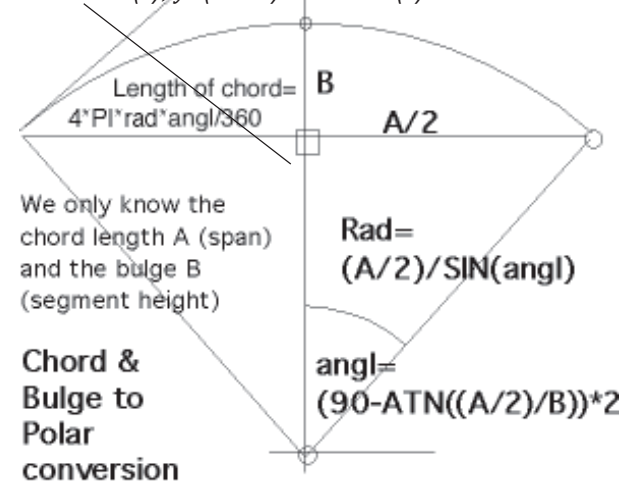
If you already know Radius and A then:

$$\text{angl} = \text{ASN}((A/2)/\text{radius})$$

In which case, B, the Bulge can be found:

$$B = (A/2) / \text{TAN}(90 - \text{angl}/2) \text{ OR } B = \text{rad} - \text{rad} * \text{SIN}(\text{angl})$$

If your origin is at the centre-base of this chord, the locations for x and y at angle k from the vertical are: $x = \text{rad} * \text{SIN}(k)$, $y = (B - \text{rad}) + \text{rad} * \text{COS}(k)$.



Above: On many occasions, you need to derive the factors of Circles and their inscribed triangles when you only know the CHORD length, A, and the BULGE (segment height), B.

If you know Bulg and angl, then radius is found:

$$\text{rad} = \text{Bulg} / (1 - \text{SIN}(\text{angl}))$$

Distance along an Arc whose angle is angl:

$$\text{dist} = 2 * \text{PI} * \text{rad} * \text{angl} / 360$$

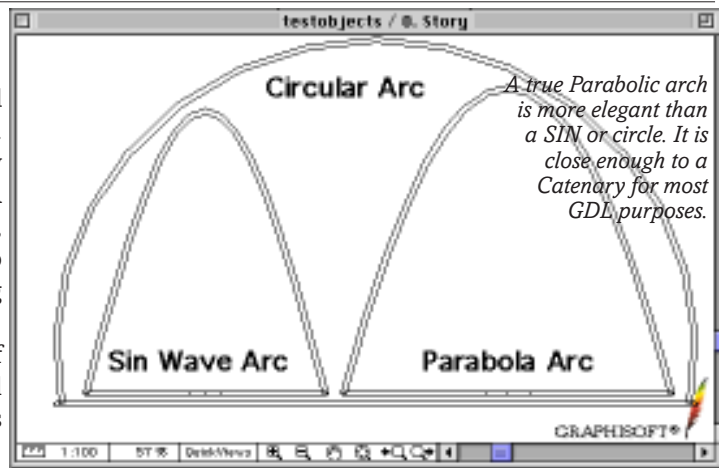
Distance between points whose angular separation is angl and radial distance is rad = $2 * \text{rad} * \text{SIN}(\text{angl}/2)$



Arch Forms

I have developed formulae which can be used for raised arches or hanging cables in GDL. The Parabola and the Catenary forms display the effects of gravity on a linear pathway. A weightless arch or cable would form a Parabola, but with self weight (e.g. chain) it is formed into a Catenary, which is based on hyperbolic trig (COSH & SINH). See the Catenary exercise.

The formulae below describe several ways of building arches and forms using mathematical curves. You might use TUBE or SWEEP to pass through the curve.



Flattish Round Arch

(less than 180 degrees of sweep):

$\text{radius} = (\text{span}/2) / \sin(2 * (90 - \text{ATN}((\text{span}/2)/\text{height})))$

$Z = \text{radius} * \cos(\text{ASN}(x/\text{radius})) - (\text{radius} - \text{height})$

(Circles and ellipses are much better done by a radial method. Ellipses can use the above method, but add, LET Z = Z*height/(span/2)).

The cartesian formula for a Circle is $R^2 = x^2 + y^2$ so you can use $y = \text{SQR}(R^2 - x^2)$ for flattish circular forms.

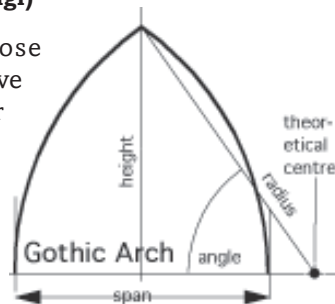
Gothic Arch

For each side, you can run a profile of the doorway with REVOLVE and use a Cutplane to cut it at the vertical axial centre. Or by using TUBE you can do it in one sequence.

$\text{angle} = (90 - \text{ATN}(\text{height}/(\text{span}/2))) * 2$

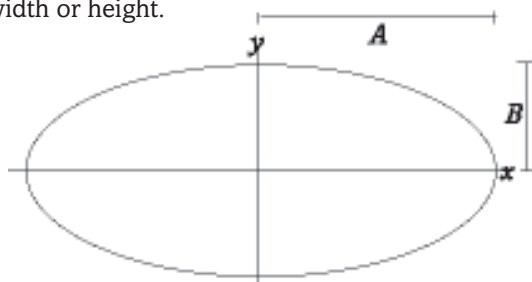
$\text{radius} = \text{height} / \sin(\text{angl})$

This is for arches whose springing point for the curve is tangential to the door style, as in this diagram.



Swing the Cat!*

This technique is explained in some exercises (e.g. Long Handrail, Helix). Trace out a circular path, use COS and SIN to work out the X,Y points, store them and then send the list of points to a TUBE. This can also be an elliptical path if you apply a multiplier to the width or height.



* 'Swing the Cat' has several possible derivations, but for me it comes from the days when one lived in a student bedsitter in which there "wasn't room to swing a cat".

SIN Wave Arch

$Z = \text{height} * \sin(X * 180/\text{span})$. This can be surprisingly useful. I used it to calculate the declination of the sun on a day by day basis through the year.

Parabolic Arch

$Z = \text{height} - (4 * \text{height} / (\text{span}^2)) * X^2$

$\text{dydx} = (8 * \text{height} / (\text{span}^2)) * X$ ('dydx' is TAN of the slope at point X – if you want to attach panels.)

For this, you start a FOR...NEXT loop starting from span/2 one side of zero and finish up span/2 the other side of zero – using PUT to store all the XYZ values. Then feed them to a TUBE command or to whatever you are building.

Inversely, $X = \text{SQR}((\text{hit} - Z) / (4 * \text{hit} / (\text{span}^2)))$ works in the Z direction but you get an over flow if you go close to the peak of the parabola.

Catenary Arch/Cable

This is a parabola for something that has weight, e.g a chain, or an arch composed of stone or concrete with some self weight. There is no guaranteed formula, there is a procedure based on $\text{SINH}(x)$. You test this formula $y = (\text{EXP}(k*x) + \text{EXP}(-(k*x))) / (2*k)$ with a changing value of a constant 'k', until the chain is the height you want it to be. See the exercise on **Catenary**.

Elliptical shapes

... are done by applying MUL to a Circular form, but if it is a 3D solid, this action may distort the section of the tubing _ but this can be helpful, e.g. this Handle:

```
!Drawer handle 3D
MULx 2 !the distortion value
ELBOW 0.1,180,0.02
DEL 1
!Makes a nice drawer handle
```



```
!Outline of an ellipse 2D
FOR ang=0 TO 361 STEP 15
x=rad*cos(ang) * 2 !the distortion value
y=rad*sin(ang)
HOTSPOT2 x,y
NEXT ang !Produces hotspots round an ellipse
```

Useful things for ellipses:

- Locus for an Ellipse: $x^2/A^2 + y^2/B^2 = 1$
- Area of an Ellipse: $\text{PI} * A * B$
- Perimeter of an Ellipse: $\text{PI} * (A+B)$

Global Variables

GLOBAL Variables are current conditions and settings of your main project. They are known by ArchiCAD. GDL objects can, at any time, interrogate them and make use of the information.

Each new version of ArchiCAD introduces more Global variables, and you need to study the Appendix in the AC Help files and Manual to see the entire list. Many of the newer ones are not usable for 2D and 3D model building (they are for the Calculate menu). Meanwhile the following are some of the GV's you

could find most useful, with a minimum of commentary. See some of the exercises to see these in active use.

In early versions of ArchiCAD, Global Variables were known by a single letter and a symbol, e.g. A_ or T~. However, this has fallen into disuse and you are advised to use the long names in case the old single letters (which still work in AC8) get dropped in future versions, or are not recognised by GDL engines like the Web browser, GDL Adaptor, ArchiFM, Black Turtle or something yet to come.

General Environment information

GLOB_SCALE : Drawing Scale (A_). Use this to determine how complex a 2D Scripted symbol will be. For example, at scales up to 1/20, you show rebates in door jambs, at 1/50 you can show them as rectangles, and beyond that, they are rendered as simple door swings. If you use True Line widths, this is essential.

GLOB_SCRIPT_TYPE : Type of current script (T~). Tells the object which script is currently working. See the Help file for the results 1,2,3,5,6.

GLOB_CONTEXT : Context of appearance. Similar to but more comprehensive than the Script_type, this lets the object know it is being viewed, say in a Section/Elevation or in its 2D symbol and it can modify the level of detail accordingly. A 3D Script can know if it is operating a PROJECT2 command and can simplify the model with planar 3D forms instead of solid 3D. See the Help file for the many results 1,2,3,4,5,6,7, 22,23,24, 43,44,46.

GLOB_DRAWING_BGD_PEN : Nearest match of Pen colour to the background colour of the drawing.

GLOB_MODPAR_NAME : Most recently altered parameter. This is incredibly useful in User Interface design. This adds intelligence to the object as it uncanonically seems to know the last parameter you changed or boolean box you ticked. It's most useful function is in User interface design where you can use it to make Boolean checkboxes function like Radio buttons.

GLOB_LAYER : The layer to which the object has been assigned.

GLOB_INTID : The internal ID created by ArchiCAD (you can't edit this).

GLOB_ID : The ID provided by the user in the object settings box, in the Listing option.

GLOB_ELEVATION : Base height relative to the current storey (not for windows/doors, they have their own GVs).

Storey Information

GLOB_HSTORY_ELEV : Elevation of the Home storey (B_) that the object is currently placed on. This is important if you have a staircase, and want it to show on the home storey as the start of a flight, and on the next as the end of the same flight.

GLOB_HSTORY_HEIGHT : Height of the Home storey. (Q_).

GLOB_CSTORY_HEIGHT : Height of the Current storey in the Floor Plan window. (R~).

GLOB_CSTORY_ELEV : Elevation of the Current storey in the Floor Plan window. (Q~).

GLOB_CH_STORY_DIST : Relative position of the Current storey to the Home storey. (S~).

Flythrough Information

GLOB_FRAME_NR : Current frame number in the animation. (N_)

GLOB_FIRST_FRAME : First frame number in the animation. (O_)

GLOB_LAST_FRAME : Last frame number in the animation. (P_). Quickest way to know if an model is experiencing a flythrough is :

```
IF GLOB_LAST_FRAME>0 THEN .....
```

Symbol Attributes

SYMB_VIEW_PEN : Default pen of the library part, (L_) if you do not use object's own pen.

SYMB_MAT : Default material of the library part, (M_) if you do not use object's own materials.

SYMB_LINETYPE : Default Line Type of the object (if you didn't use object own Line Type).

SYMB_FILL : Default Fill of the object (if you didn't use object's own Fill) when cut in Section, and similar considerations apply to the following GVs:

SYMB_FILL_PEN : Fill Pen. **SYMB_FBGD_PEN** : Fill Background Pen. **SYMB_SECT_PEN** : Section Pen on contours.

SYMB_VIEW_PEN : Default pen of the library part, (L_) if you do not use object's own pen.

More Global Variables

Camera Information

GLOB_EYEPOS_X, **GLOB_EYEPOS_Y**, & **GLOB_EYEPOS_Z** (K~, L~ and M~) tell the object where the most recent camera is located.

GLOB_TARGPOS_X, **GLOB_TARGPOS_Y**, & **GLOB_TARGPOS_Z** (N~, O~, P~) tell the object what the camera is looking at – the ‘target position’.

SYMB_POS_X, **SYMB_POS_Y**, **SYMB_POS_Z** (X~, Y~, Z~) tell the object where the object itself is in the overall project.

Location Awareness : These GVs are useful for getting objects to face the camera in a static perspective view. They can change position as the camera approaches during a flythrough by recalculating their position and that of the camera.

Limitations to Globals

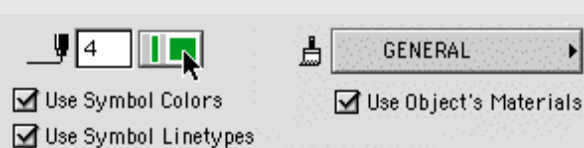
Library Parts are only that, they cannot see other objects. Thanks to APIs in ArchiCAD, they may appear to be able to do this, e.g. the way Rooflights find and cut the roof; Window and Doors cut walls; objects can use gravity to find a Mesh, Roof or Slab surface. In most cases, these are things you can only influence by your choice of Subtype.

Certain Certain things cannot be found from Globals and may have to be found using the REQ and the REQUEST commands.

Most globals are numbers; REQUESTs are for renumbers or strings. An example of REQUEST is current System Time and Date, or the RGB of the Pen.

MATERIAL SYMB_MAT : PEN SYMB_VIEW_PEN

MATERIAL M_ : PEN L_



Rotation of Object

SYMB_MIRRORED : Flag to indicate if D symbol is mirrored in plan. (V~). This is useful if you have a scripted 2D symbol, and want to ensure that Hotspots and text in objects still appear the right way round and in the right place, even when the symbol is mirrored. When you tick the ‘mirrored’ checkbox in the Object settings dialog, the 2D symbol and the hotspots will be mirrored for you. However, you might want the Properties script to know that the object has been mirrored – for example in specifying right or left handed desking units in interior layouts.

SYMB_ROTANGLE : Rotational angle of the object. (W~). No matter how you rotate the object, you can straighten it out or straighten its text label – see the Circle Grid.

Orientation Information

(if you set the Sun position in the Sun Dialog).

GLOB_NORTH_DIR : Project North direction (U~). Force a North point to face North; set a solar panel or shading device to animate during a flythrough.

GLOB_SUN_AZIMUTH : Direction of the Sun.

GLOB_SUN_ALTITUDE : Altitude of the Sun.

Windows / Doors

There are a HUGE number of Global Variables to do with Windows and Doors, so that the windows or doors can know the Wall thickness, materials, fill patterns and pens, sill height and many more conditions. These mostly begin with the prefix **WIDO**. They make the GDL for windows and doors perhaps the most complex GDL of all to write. They require further discussion in the windows/doors section of the GDL Cookbook.

Global Variables: Location Aware Capability

IT is possible to give a certain amount of decision making capacity to the object. If you run a flight through the model, objects can react to the camera position – if you ‘Rebuild each Frame’.

The location awareness algorithm can be applied to:

- 2D trees and human figures rotate to look full bodied to the camera.
- The density of branches & leaves on trees can vary. People and car objects can simplify hugely when camera is far. Complex objects like glazing fittings and louvres decide not to draw, or turn off Shadows.
- Lights come on in rooms.
- Curved objects change their RESOL/TOLER value to show more polygons when the camera is near, and reduce when it is far.

Do it with Global Variables!

The Location awareness routine uses Global Variables. ArchiCAD tells GDL the Camera location and the Object’s own location. By comparing the distance differences in X, Y and Z (a simple subtraction that works whichever sector the model or camera is in), you use a 3D Pythagoras calculation to reveal the real distance *dd* from camera. *dkx* is the difference between the GVs ‘K~’ and ‘X~’ – the object and camera positions in the X direction. The same idea applied in the Y & Z direction generates *dly* and *dmz*. See the example here.

An ArcTan (ATN) calculation reveals the Azimuth (or direction) by dividing the X distance into the Y distance.

Location Aware Capability, continued

When does it not work?

If there is no camera, e.g. in an axonometric or an elevation, you need a 'get-out-clause' to set a false distance from the camera. This is to avoid a 'divide by zero' error when calculating the azimuth. Thus the variable *dkx* is given a tiny value of 1 mm.

The routine works with Sun Studies and Flythroughs. Doors and Windows can open during an animation, but must do so based on frame number. They do not work correctly with 'location awareness' because they subordinate themselves to the wall they are attached to and you cannot write the routine for the wall.

Animate based on Distance/Angle or Time?

Other aspects of animation, such as wind turbines whose blades turn, buildings stacking themselves up, cars moving, human figures moving, sea undulating, yachts heeling, Doors and Windows etc are better done by using the global variable **GLOB_FRAME_NR** (frame number in the animation) to determine the action – which is equivalent to TIME.

Steal this routine!

This location awareness routine can be used again – keep it in your Scrapbook. I have done a model where EVERY GDL component was location aware to keep the rendering workload under control. If the model is a mile or more in size, you can completely hide objects too distant to be worth drawing.

Rebuild each Frame

This routine will work for still views with different camera distances (after a 'Rebuild'). To ensure that this works during a movie flythrough, you need a REBUILD EACH FRAME to happen during animation. This can be enabled in the dialog box just prior to starting a Flythrough.

Which Way is it?

Azimuth is a difficult calculation. Because objects could be in positive and negative quadrants, this calculation could result in inversion. There is an interesting and complicated procedure for 100% accurate azimuth. Applied to people and tree objects, you do not mind if objects flip 180° as that merely helps the randomness

```
!Test of Frank's Azimuth algorithm
!2D Script
PEN ink
HOTSPOT2 0,0
HOTSPOT2 x1,y1
HOTSPOT2 x2,y2
CIRCLE2 0,0,0.1 !Origin
LINE2 x1,y1,x2,y2 !Line itself
CIRCLE2 x1,y1,0.02 !Start
CIRCLE2 x2,y2,0.01 !End
DEFINE STYLE 'angtst' 'Arial',4,8,0
SET STYLE 'angtst'
TEXT2 0,0,angla !Label
ARC2 x1,y1,hypo,0,angla !AngleArc
```

```
!Location Awareness Routine
!with camera global variables
!using the old letter system.

dkx=K~ - X~:IF dkx=0 THEN dkx=0.001
dly=L~ - Y~
dmz=M~ - Z~

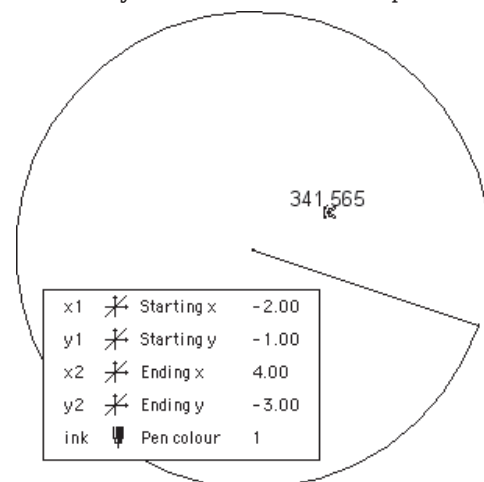
dr =SQR(dkx^2 + dly^2) !Plan distance
dd =SQR(dkx^2 +dly^2 +dmz^2) !Distance
azi=ATN(dly/dkx) !Azimuth

IF facecam THEN ROTz 90+azi-W~
```

The Eye positions K~, L~, & M~ can also be called GLOB_EYEPOS_X, Y & Z, and the Object positions X~, Y~ & Z~ can be called SYMB_POS_X, Y & Z. The adjustment to dkx avoids a divide by zero error. If azimuth is critical there is more to it than this, but this routine is good enough for trees and people.

to look more authentic. But if you need accurate rotation, this routine from Frank Chin of Brunei provides a fix. It is more complicated than the ArcTan of the quick and easy method – but it works!

Frank writes: Below is the routine that I used for finding the bearing of a line from Point 1 (x1, y1) to Point 2 (x2, y2). This can be adapted to be a subroutine to calculate many different lines in a shape.



```
!FINDING BEARING OF LINE - Master Script
dx=x2-x1
dy=y2-y1
hypo=SQR(dy^2+dx^2) !Hypoteneuse
sinq=dy/hypo
cosq=dx/hypo

!angla is the angle the line
!makes with the horizontal X-axis
IF dx=0 THEN
  IF dy>0 THEN angla=90 ELSE angla=270
ENDIF
IF dy=0 THEN
  IF dx>0 THEN angla=0 ELSE angla=180
ENDIF
IF dx<>0 AND dy<>0 THEN GOTO 121
GOTO 131

121:
angla=SGN(ASN(sinq))*ASN(sinq)
!to get angle between 0 & 90 degree only
IF SGN(sinq)= 1 AND SGN(cosq)= 1 THEN angla=angla
IF SGN(sinq)= 1 AND SGN(cosq)=-1 THEN angla=180-angla
IF SGN(sinq)=-1 AND SGN(cosq)=-1 THEN angla=180+angla
IF SGN(sinq)=-1 AND SGN(cosq)= 1 THEN angla=360-angla
131:
```


Global Variables:

Storey Sensitivity

OFTEN an object needs to appear on other storeys¹, such as main beam lines, staircases, ventilation ducts, drainage lines, lighting fittings etc. – but you don't want it to appear solid. Here is an example of Storey Sensitivity, using a simple stretchy round table.

GLOBAL_CH_STORY_DIST: it's a Global!

On its own storey, the Global variable S~ (also known as GLOBAL_CH_STORY_DIST) is zero. On other storeys it's a number e.g. 1 for the storey above, -1 for the storey below. So the object can be displayed on the home storey in all its glorious detail, and on other storeys it will show more simply using a linetype of the user's choosing.

Learn how to Define LineType

One thing you can introduce here is a 'Calculated Line type'. This could be based on a fraction of the circumference of the table, in this case 1/36 of it – this gives you a consistent line spacing regardless of the drawing scale – the dotted line pattern will be the same at 1/50 or 1/200 scale. DEFINE LINE_TYPE is tricky but the example here works. We define a Solid Line first, then the more difficult dotted line. Of course the Line weight is still subject to the scale of the drawing.

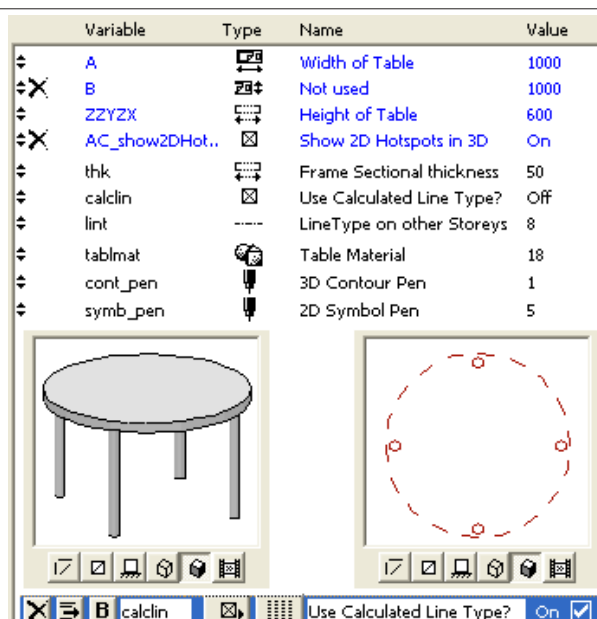
```
!Simple Table Object
!3D Script

MATERIAL tablmat
PEN cont_pen

!Legs
RESOL 8
FOR k=1 TO 4
  ROTz k*90
  ADDx A/2-thk
  CYLIND zzyzx-thk,thk/2
  DEL 2
NEXT k

!Table Top
RESOL 36
ADDz zzyzx-thk
CYLIND thk,A/2
DEL 1
```

¹ In UK spelling, 'stories' (pl. from story) are what you tell your children at bedtime, and 'storeys' are what you call levels in buildings.



```
!Simple Table Object
!2D Script

PEN symb_pen
HOTSPOT2 0,0
HOTSPOT2 -A/2,0
HOTSPOT2 A/2,0
HOTSPOT2 0,-A/2
HOTSPOT2 0,A/2

DEFINE LINE_TYPE 'solid' 1,1,1
SET LINE_TYPE 'solid'
```

This routine will guarantee that the segments are resized to fit your object.

This dotted circle has 36 segments and gaps, but at the end point, you always get a double segment. The spacing is the reciprocal of the scale so it looks right whatever the drawing scale.

```
!Legs
IF GLOBAL_CH_STORY_DIST=0 THEN
!We are 'on home storey'
FOR k=1 TO 4
  ROT2 k*90
  CIRCLE2 A/2-thk,0,thk/2
  DEL 1
NEXT k
ENDIF

!Table Top
IF GLOBAL_CH_STORY_DIST THEN
!We are 'Not at home'
IF calclin THEN
  circum=PI*A !Circumference
  DEFINE LINE_TYPE 'tabl_lin' 1/GLOBAL_SCALE,
    2, circum/36,circum/36
  tabl_lin=IND(LINE_TYPE,'tabl_lin')
  SET LINE_TYPE tabl_lin
ELSE
  SET LINE_TYPE lint
ENDIF
ENDIF

CIRCLE2 0,0,A/2 !Draw the Tabletop
```

Here we convert the defined Linetype to a numerical value (an 'index') with the IND() statement.

Special line types do not work unless 'Use Object's own Line Types' is ticked

Global Variables:

Un-rotatable Object

THIS simple little object could be developed into a Northpoint or Text box, where it cannot be rotated by a normal Ctrl-E command.

Only the Angle Parameter 'angl' or its hotspots will rotate it.

```
!The Un rotatable Object
!2D Script
!Can only be rotated by the hotspot angle changer
HOTSPOT2 0,0
```

```
ROT2 -SYMB_ROTANGLE
HOTSPOT2 A*0.6,0, 1000,angl,4+128!Base
ROT2 angl
HOTSPOT2 A*0.6,0, 1002,angl,5!Move
LINE2 A/2,0,A*0.6,0
DEL 1
HOTSPOT2 0,0, 1003,angl,6!Centre
DEL 1
ROT2 -SYMB_ROTANGLE+angl
POLY2 5,7,
-A/2,-B/2,
A/2,-B/2,
A/2,B/2,
-A/2,B/2,
-A/2,-B/2
DEL 1
```



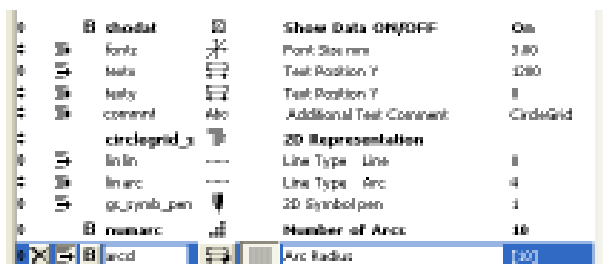
CircleGrid

Parameter Arrays & Font scaling

CIRCLEGRID object can now be updated with Parameter Arrays for the Arcs, and (as it was for the Cabinet) this is very easily done with only small changes to make. Then you can add hotspots for each of the arcs, using a single subroutine for all.

Parameters and Master Script

Add in the new parameter of 'arcd' and hide or delete the old 'arcd1' and 'arcd2'. Add a text parameter of 'commnt'



!Add to Master Script for Parameter Arrays

```
IF numarc=0 THEN HIDEPARAMETER 'arcd'
```

```
FOR k=1 TO 10
```

```
IF arcd[k]<=0.01 THEN arcd[k]=0.01
```

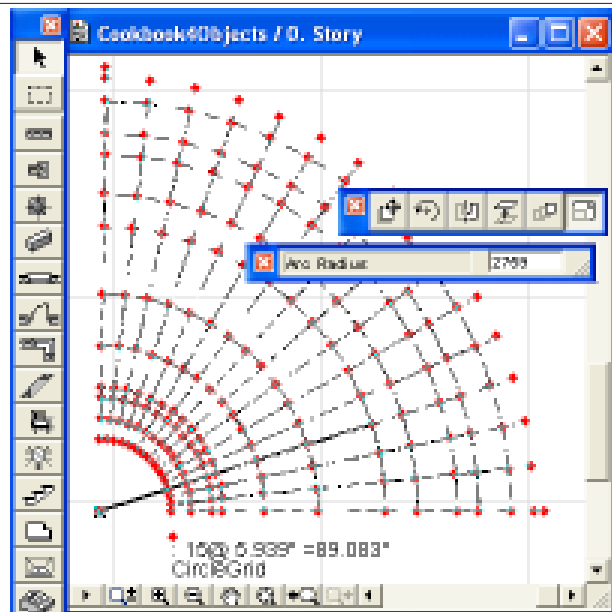
```
NEXT k
```

We can hide the parameter 'arcd' if no arcs are required. More importantly, we need a routine to prevent GDL trying to draw arcs with a radius of zero. When you make new parameter arrays, zero is their value. This routine protects the object from errors.

2D Script

We now have an easy FOR NEXT loop to draw all or any of the arcs. If you are editing the old CircleGrid, delete the previous arc drawing routine in favour of this one. The Hotspot subroutines for the arcs can be replaced by this simple one that is sufficient for *all* the hotspots on *all* the arcs.

There is no change to the first part of the original Text routine which used the GV: SYMB_ROTANGLE. We have added one more parameter, 'commnt' which allows the user to display a title. This gives an opportunity to show another use of GVs. The font size is given in terms of plotting millimeters, and we need to calculate the leading – a safe downward distance for the second text line.



!CircleGrid with Hotspots
!2D Script additions

```
IF numarc THEN
```

```
LINE_TYPE linarc
```

```
FOR k=1 TO numarc
```

```
ARC2 0,0, arcd[k],0,fan
```

```
NEXT k
```

```
ENDIF
```

```
!Show angular increment and sweep
```

```
IF shodat AND fontz THEN
```

```
GOSUB 1050: !Hotspots for Text
```

```
DEFINE STYLE 'angtext' 'Arial',fontz,1,0
```

```
SET STYLE 'angtext'
```

```
angstr=':'+STR(numang,3,0)+'@'+STR('%6.3dd',stang)+'='+STR('%6.3dd',fan)
```

```
ADD2 textx,texty
```

```
ROT2 -SYMB_ROTANGLE
```

```
TEXT2 0,0, angstr
```

```
!Additional comment!
```

```
IF commnt>" THEN
```

```
TEXT2 0,-1.3*fontz*GLOB_SCALE/1000,commnt
```

```
ENDIF
```

```
DEL 2
```

```
ENDIF
```

Multiply the millimetre font height by the drawing scale. Dividing this by 1000 gives us a final height in Metres. Multiply this by 1.3 to get a comfortable leading between each line.

```
END: !
```

```
!-----Replacement subroutine-----
```

```
1030: !Hotspots Arcs
```

```
FOR k=1 TO numarc
```

```
hsid=hsid+1 !Base
```

```
HOTSPOT2 0,0, hsid,arcd[k],129
```

```
hsid=hsid+1 !Move
```

```
HOTSPOT2 arcd[k],0, hsid,arcd[k],2
```

```
hsid=hsid+1 !Vect
```

```
HOTSPOT2 -1,0, hsid,arcd[k],3
```

```
NEXT k
```

```
RETURN
```

The Cabinet

Enhance the 2D: Text & Hotspots

THIS Cabinet can be a more serious object with some text labelling, and 2D Hotspots for the opening of the drawers. Although one might not need to invest such a humble item of furniture with all this capability it's a good vehicle for learning GDL basics. Read this and use these routines for any object you want to add labelling to.

Later we can add dimensions and highly varied forms of 2D representation, to help in the design and manufacturing process – configuration using GDL!

Parameters

Add a whole raft of extra parameters. Amazingly, nothing more is required in the Master Script – we can use these directly in the 2D Script. If you make this a subtype (I recommend 'Storage') then you will get a lot of Archi-FM parameters. For now you can click the little X next to them and hide all. We can put a small ValueList for the Fonts in the Parameter Script.

```
!Parameter Script
VALUES 'font' 'Arial', 'Verdana', 'Times'
```

User-editable Font Sizing

In the CircleGrid we used Plotting size for the text, which is in millimetres. For the Cabinet we can use another way (hints of AutoCAD...) – sizing the text to real world dimensions, so they stay with the cabinet. We have a safeguard to hide the text if the scale gets smaller than 1/100, using the Global Variable GLOB_SCALE. We also want a Hotspot routine to make the text height and positions user-editable. The Hotspots are on the first lines of the script. The parameters for the text position are hidden in the Parameter Table so that Hotspots are the only way to relocate the text.

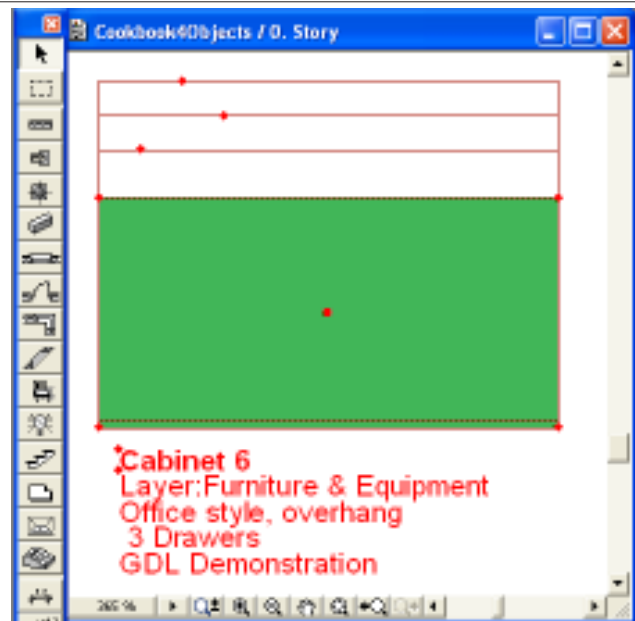
We need two Define Style routines for the main title and the minor lines of text. In this case, the display of the current Layer and timber detailing style could be very useful, and we also have an Optional Label for you to add an extra comment.

Text options

As the User can select a number of items to display in the label, we use Boolean parameters to select them, and then a routine in the 2D Script to set out the lines of text so that there are no gaps if one of the lines is omitted. We also want to be able to rotate the Cabinet to place it anywhere in the plan, but still have the lines of text horizontally oriented – so we must always rotate the text to counter the object's rotation angle.

2D Enhancement

We can also show the drawers opening and shutting. This is not necessary for such a cabinet, but it's another lesson in GDL 2D scripting and hotspots. Indicator lines over the Poly can indicate the timber styling of the cabinet.



The distance of the Hotspots from the top left corner tells you if it's a lower, middle or higher drawer. Try the object on the CD. No matter how you rotate the object, the text always remains upright.

Variable	Type	Name	Value
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	B labl	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Font	Font	On
<input checked="" type="checkbox"/>	Font	Font	Arial
<input checked="" type="checkbox"/>	fsiz	Font size in Real World dims	100
<input checked="" type="checkbox"/>	gs_text_pen	Pen for Text	10
<input checked="" type="checkbox"/>	globid	Show Global ID of object	On
<input checked="" type="checkbox"/>	globlayer	Show Layer	On
<input checked="" type="checkbox"/>	labdetl	Show Timber style	On
<input checked="" type="checkbox"/>	labdraw	Show Drawers	On
<input checked="" type="checkbox"/>	labopt	Show optional Label	On
<input checked="" type="checkbox"/>	optlabl	Text for Optional label	Opt label
<input checked="" type="checkbox"/>	txtx	Text Location X	100
<input checked="" type="checkbox"/>	txty	Text Location Y	100
<input checked="" type="checkbox"/>	cabinet_dr_sp2	Hidden Parameters	
<input checked="" type="checkbox"/>	bthk1	Primary Board thickness	20
<input checked="" type="checkbox"/>	bthk2	Secondary Board thickness	12
<input checked="" type="checkbox"/>	gs_list	ArchiFM & Listing Params	
<input checked="" type="checkbox"/>	gs_list_cost	Cost	0.00
<input checked="" type="checkbox"/>	gs_list_manufact..	Manufacturer	
<input checked="" type="checkbox"/>	gs_list_note	Note/Remarks	
<input checked="" type="checkbox"/>	gs_list_location	Location	
<input checked="" type="checkbox"/>	gs_list_accessories	Accessories	
<input checked="" type="checkbox"/>	FM_Type	Group Type	Furniture
<input checked="" type="checkbox"/>	FM_InventoryN..	Inventory Number	
<input checked="" type="checkbox"/>	FM_SerialNumber	Serial Number	
<input checked="" type="checkbox"/>	FM_Production...	Production Year	

```
!Cabinet with Drawers
!2D Script
```

```
GOSUB 1000:!Hotspots for outline
GOSUB 1020:!Hotspots for Drawers
```

Add in the Hotspot subroutine for the drawers.

```
HOTSPOT2 A/2,B/2
PEN gs_symb_pen
FILL gs_fill_type
POLY2_B 5,7,gs_back_pen,gs_fill_pen,
0,0,1,
A,0,1,
A,B,1,
0,B,1,
0,0,-1
```

The indicator lines just overlay the Fill showing if it's Office or Bedroom style – using the flag 'ts'.

```
IF ts THEN !Indicator lines
LINE2 0,bthk1,A,bthk1
ELSE
LINE2 bthk1,0,bthk1,B
LINE2 A-bthk1,0,A-bthk1,B
ENDIF
```

```

!Drawer lines
FOR k=1 TO numdraw
ADD2 0,B
POLY2 4,1,
  0,0,
  0,drw_opd[k],
  A,drw_opd[k],
  A,0
DEL 1
NEXT k

!Labelling Routine
IF lab1 AND GLOB_SCALE<101 THEN
PEN gs_text_pen
  ADD2 0,-fsiz*0.35
  GOSUB 1050:!Hotspots for Text
  GOSUB 1060:!Hotspots for Fontheight
DEL 1
DEFINE STYLE 'txtbold' font,
  fsiz*1000/GLOB_SCALE,1,1 !Bold
DEFINE STYLE 'txtnorm' font,
  fsiz*1000/GLOB_SCALE,1,0 !Normal
SET STYLE 'txtbold'
ledg=1.2 !leading distance
ln=-ledg !Starting value:line number
ADD2 txtx,txty
ROT2 -SYMB_ROTANGLE

IF globid THEN !ID
ln=ln+ledg
string=GLOB_ID
TEXT2 0,txty-fsiz*ln,string
ENDIF

SET STYLE 'txtnorm'
IF globlayer THEN !Layer
ln=ln+ledg
string='Layer: '+GLOB_LAYER
TEXT2 0,-fsiz*ln,string
ENDIF

IF labdet1 THEN !Detailing
ln=ln+ledg
string=timstyl
TEXT2 0,-fsiz*ln,string
ENDIF

IF labdraw THEN !How many drawers
ln=ln+ledg
string=STR(numdraw,2,0)+' Drawers'
TEXT2 0,-fsiz*ln,string
ENDIF

IF labopt THEN !Optional Comment
ln=ln+ledg
string=optlab1
TEXT2 0,-fsiz*ln,string
ENDIF
DEL 2
ENDIF
END: !=====

```

*These are open (unclosed)
Polygons to indicate each
drawer.*

*The IF statement includes a
safeguard to stop the Cabinet
printing out the text if it's in a
general building layout plan –
this could be a user definable
limit.*

*Include a
routine for the
Font sizing
Hotspots.*

*Two similar
text styles are
defined,
normal and
Bold.*

*We are using
a leading of
1.2. Each
option
increments a
flag 'ln' for
line number.*

*If the option
is not
required, the
lines appear
to close up
tidily.*

*Some of the
strings are
created by
concatenation
before they
are sent to the
TEXT2
statement.*

```

!Add these Subroutines to the 2D Script
1020:!Hotspots for Drawers
FOR k=1 TO numdraw
ADD2 k*A/11,B
  hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,drw_opd[k],129
  hsid=hsid+1 !Move
HOTSPOT2 0,drw_opd[k], hsid,drw_opd[k],2
  hsid=hsid+1 !Vector
HOTSPOT2 0,-1, hsid,drw_opd[k],3
DEL 1
NEXT k
RETURN

1050:!Hotspots for Text Location
  hsid=hsid+1 !Base
HOTSPOT2 0,txty, hsid,txtx,129
  hsid=hsid+1 !Move
HOTSPOT2 txtx,txty, hsid,txtx,2
  hsid=hsid+1 !Vector
HOTSPOT2 -1,txty, hsid,txtx,3
  hsid=hsid+1 !Base
HOTSPOT2 txtx,0, hsid,txty,129
  hsid=hsid+1 !Move
HOTSPOT2 txtx,txty, hsid,txty,2
  hsid=hsid+1 !Vector
HOTSPOT2 txtx,-1, hsid,txty,3
RETURN

1060:!Hotspots for Fontheight
ADD2 txtx,txty
  hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,fsiz,129
  hsid=hsid+1 !Move
HOTSPOT2 0,-fsiz, hsid,fsiz,2
  hsid=hsid+1 !Vector
HOTSPOT2 0,1, hsid,fsiz,3
DEL 1
RETURN

```

*As there are 10
drawers, we can't
go wrong by
basing the hotspot
spacing on 11ths.*

*With parameter
arrays, the entire
set is done in one
subroutine.*

*The Text location
is the same as for
the CircleGrid.*

*The Font sizing
routine is very
useful, copy and
use it on many of
your objects.*

Multiply, don't Divide!

PROCESSORS find it much easier to multiply than to divide. To divide once can take perhaps 32 cycles of a processors time, whereas a multiplication can take 1, 2 or 4 cycles, depending on which processor we are talking about, and whether it is optimised for floating point or integer. So although it may be easier on the eye to write an expression like 'LET len=wid*3/4', it may process faster in the form 'LET len=wid*0.75'. If you have to divide by 3, then you should trust the machine, I would always write prime number division in the form 'wid/3'.

Seriously, the time difference would only matter on extremely long scripts or slow machines. You are better off writing efficient code, by using polylines and clear structure, and by reducing the polygon count on surfaces.

Rendering – Speed

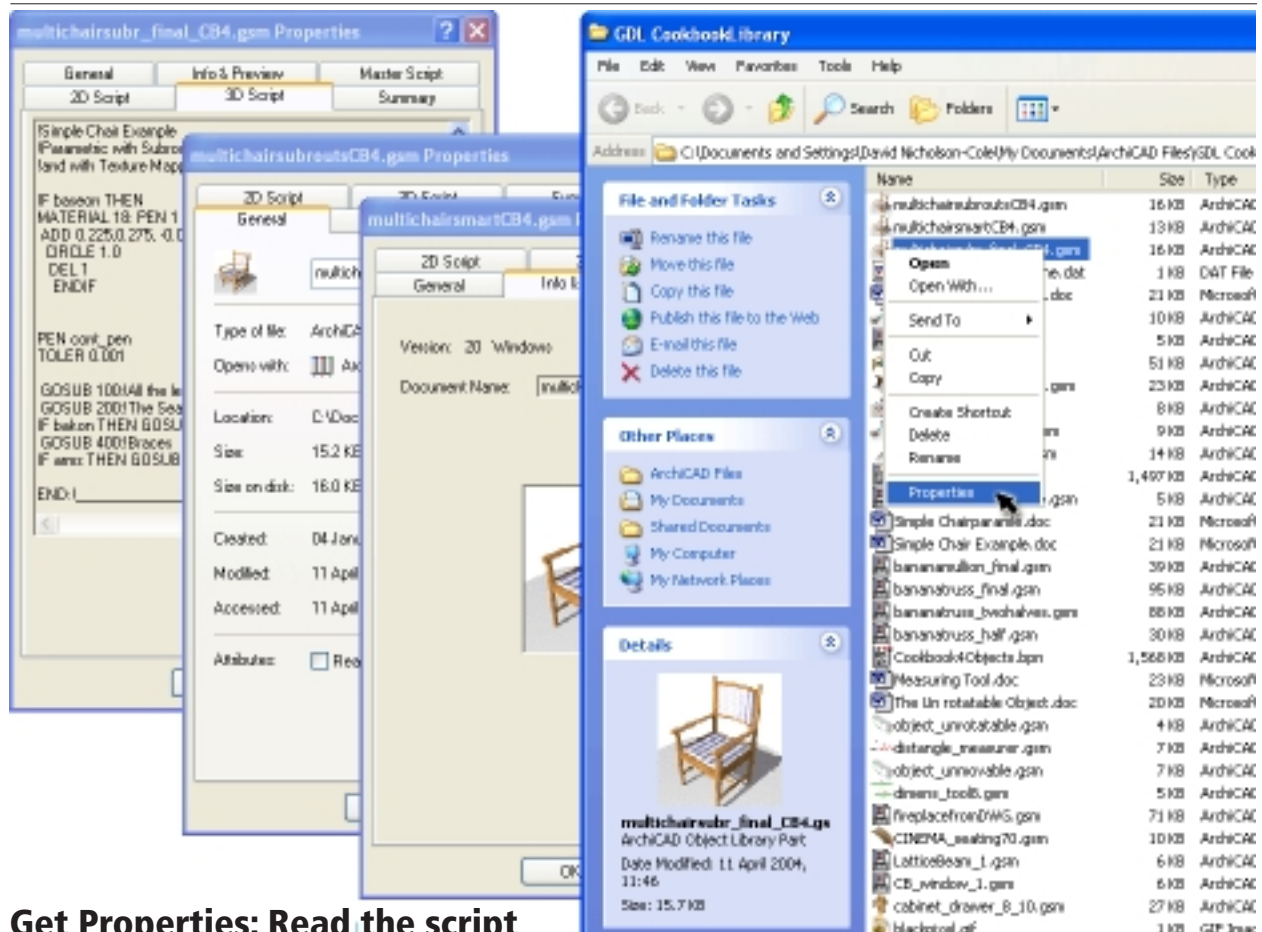
IF the project is large, photorendering often appears faster than 3D drawing, analytic or raster. Before embarking on a VR walkabout or a flythrough animation or series of pictures, ArchiCAD tries to draw a 3D of the most recent camera used – which can take ages. It does this for good reasons – it is, in effect, performing an integrity check on the whole model to make sure that there are no objects, pictures or textures missing.

Therefore, set up one 3D camera facing **away** from the model in Raster mode (so that it draws nothing), and immediately after, proceed with the VR or Flythrough. It will start with the rendering almost immediately.

Subroutines in the Master Script

THIS is supposed to be impossible – because to have a subroutine you need to have the END statement to prevent the program execution running into the subroutine. You can't do this because the Master Script is so powerful that it will do what you say! Well if you are a believer in the idea of structured programming, it is possible. If you have a line at the very end of the script like 999999: !End, you can have a GOTO 999999 in the Master Script which will behave like an END statement.

One drawback of this is that your line numbers must not be ones that are likely to be used in any of the other scripts. So you should number your subroutines with high numbers like 100011:, 100021:, 100031: etc.



Get Properties: Read the script

ONE useful trick available to Windows users is the **Get Properties** function – available to right clickers on GSM icons. You get the preview and basic info, as you would expect. More amazingly, because the scripts are contained in resources, Windows will tab up each resource and reveal the contents of the scripts, and you can drag a mouse through them and copy script to the clipboard. Unfortunately, Mac OSX users can only envy this, the most they can do is view the preview image.

If this doesn't work, it may be that your GSM files are thought to be a form of Video, which shares the same suffix. With right-click **Open As...** you can tell it which application you want the file to belong to and it will remember for later.

It would be great if GS and MS could collaborate so that the 'Get Properties' also included a 3D object browser, so that objects appear as they do on the web.

You could use the Master Script to build some material definitions if you wanted clean primary coloured materials.

```
!Master Script
IF NOT(stretchy) THEN LOCK 'A','B','ZZYZX'
DEFINE MATERIAL 'blak' 4, 0.1,0.1,0.1
DEFINE MATERIAL 'whit' 4, 1,1,1
```

We do not need special materials for the Fireplace, but here is an early glimpse of a Material definition – **DEFINE MATERIAL** name, type, Red, Green Blue.

Attribute Definitions

DEFINE LINE_TYPE

LINE_TYPE sizes are like Font height in defining style. The dimensions in the **DEFINE** statement are the actual segment length as drawn by your plotter, nothing to do with the dimensions of the objects they are drawing..

Syntax: **DEFINE LINE_TYPE** "name" spacing factor, number of line parts, length1, length2, etc etc

DEFINE LINE_TYPE is tricky because of drawing scale issues – so make the spacing factor the reciprocal of the Global Variable for Drawing Scale.

Please consult the GDL Help pages for more detail on **LINE_TYPE**. A quick definition for a Solid line is **DEFINE LINE_TYPE 'solid' 1,1,1** because it only needs one line part.

Attribute Definitions



DEFINE

PARAMETERS for Fills, Materials and Lines etc can be provided by the user in the Parameter Table. But there are times when the GDL writer has to use **DEFINE**. Only a Masochist could possibly enjoy **DEFINE FILL** and **DEFINE LINE_TYPE**; but **DEFINE STYLE** and **MATERIAL** are essential and desirable parts of your GDL. Once defined, you can command GDL to use a Style, Material etc, by the command **SET**. In fact, you do not need **SET**; just saying **MATERIAL** followed by the name will be sufficient.

DEFINE STYLE is essential if any **TEXT2** is to be included in a 2D symbol – followed by the **SET STYLE** command.

It is even more important if you have more than one line of text, and if you intend to print or plot the plans, because you can control the position of the text, and the font plotting and printing height.

Once you start 2D scripting you will write **DEFINE STYLE** statements so often that you will be able to remember the syntax without difficulty.

Size: Most dimensions in GDL are assumed to be in Metres, but this is one odd case where font height produced by the plotter has to be defined in millimetres (even in the USA) For this reason, when you state the font size in the parameter box, you must use parameter type ‘real number’ or ‘integer’, and not use the ‘length’, because ‘length’ could be changed by the dimensional settings of the project. If font size is not stated, the **TEXT** or **TEXT2** will not appear.

Anchor defines the position where the text starts writing. If you use ‘1’, it will be left justified, and above the point of text entry. If you use 2, it will be centred, and if you use 3 it will be right justified. Larger numbers change the vertical position of the text. It’s like the keypad of a touchtone phone.

FaceCode is simply whether the text is plain, italic, bold etc. Codes are:

0=Normal, 1=**Bold**, 2=*Italic*, 4=Underline, 8=**Outline**, 16=Shadow. You can combine the numbers e.g. 3=**Bold Italic**.

DEFINE MATERIAL enables you to carry material qualities with your GDL object so that, when dropped into someone else’s model in another country, it will still have the correct colour and appearance. People often tweak their materials in AC and then default material index numbers change. ‘Whitewash’ and ‘Stainless Steel’ may have disappeared... Mercy!

In my Motorist object, it is vital that red tail lamps and indicators look right, and that the white bodywork of the police car does not look grey. The tail-lamps change from reflective red in daylight, to brightly emitting red when the lights are turned on.

GDL has long highly detailed ways to define materials. For now, we can look at the short definition, which is based on RGB values. You can make use of predefined surface qualities, like ‘Plastic’(4) so try this simpler definition:

- 2: Matte: non reflective, horrid.
- 3: Metal: shiny but non reflective, too dark.
- 4: Plastic: not shiny, but reflective – the **Best!**
- 5: Glass: transparent (with colour) – **OK!**
- 6: Glowing: with colour (doesn’t work properly).
- 7: Constant: utterly horrid! no shading

!Example of a long definition, type 0

```
DEFINE MATERIAL "Bright_Metal" 0,
0.84, 0.83, 0.85, !Surface RGB
0.95, 0.10, !ambient, diffuse
0.95, 0.0, !specular, transparent
57, !shining
4, !transparency attenuation
1,1,1, !Specular RGB
0.0, 0.0, 0.0, !Emission RGB
0.0 !Emission attenuation
shmat=IND(MATERIAL, "Bright_Metal")
```

!Example of a short definition, type 4

```
DEFINE MATERIAL "anodiseframe" 4,
0.84, 0.83, 0.85 !Surface RGB
```

!The “don’t bother” definition

```
MATERIAL 0: PEN ccol
```

If you don’t bother, and just make the material zero, the pen colour becomes the material, but it’s a horrible matte (type 2) quality.

Fill Patterns

Quick fill patterns for bitmaps are as follows:

```
DEFINE FILL 'whit' 0, 0, 0, 0, 0, 0, 0, 0,
0,0,0 ! totally white fill, note it's 11 zeroes
DEFINE FILL 'blak' 255, 255, 255, 255, 255, 255, 255, 255,
1,0,0 ! totally filled fill
DEFINE FILL 'grey50' 85, 170, 85, 170, 85, 170, 85, 170,
1,0,0 ! 50% filled fill
DEFINE FILL 'grey25' 85, 0, 170, 0, 85, 0, 170, 0,
1,0,0 ! 25% filled fill
DEFINE FILL 'tiled' 64, 192, 192, 192, 192, 192, 255, 255,
1,0,0 ! 3D tiled effect
```

DEFINE FILL can be a nightmare as you have to define Fill patterns in binary, valuing each dot, and then in lines. The same command also includes routines to define the fill by vectors. But then if you don’t have a fill, all your objects could have transparent 2D symbols and look wrong. So you may have to get by with the standard fills.

But it’s very useful to be able to script a perfect solid and white fill. Here are a few useful routines for making white, black and grey fills. In other cases you can have a user parameter, asking the user to specify the fill for the 2D symbol. For this definition, the number of zeroes is very important.



ZZYZX

Vertically Stretchy Parameter

ONE innovation since ArchiCAD 6 was to permit GDL users to provide a quick way to enter heights of objects without having to open the settings box. 'zzyzx' is the parameter for height. If you use zzyzx as the magic height parameter, ArchiCAD will give special treatment to it including making it stretchy in the 3D view, and extending the 2D hotspots to the height of zzyzx.

I wrote to Laszlo Vertesi of Graphisoft to ask how 'zzyzx' came about because I guessed that the inspiration may have come from the famous computer game 'Colossal Cave'?? Maybe it was the password to get from the Sapphire room back to the House in the Forest??? (and back down again) Laszlo Sparing of R&D at Graphisoft-HU replied,

"Dear David, I was the fool who suggested this stupid name for this standard parameter. Sorry, no computer game, only a short visit to the City of Casinos after a conference. I looked for a unique name which doesn't make any old library part obsolete. Names like Z, ZZ, ZVALUE, ZPARAM etc. can be used anywhere in the ArchiCAD world, but I hope that zzyzx is strange enough, and that's why ideal for the above goal."

What is ZZZZX? – it's a small town in California halfway between Las Vegas and Baker; close to the Nevada border in the Mojave Desert about 3 miles south of the main highway – on the edge of the dried up Soda lake, next to the very dry Soda Mountains. Its main street is "the Boulevard of Dreams".

I have actually driven a car through the Mojave desert from LA to LV, and although I didn't see that sign, I noted that there is a lot of weirdness out there – and it doesn't surprise me if there's someone trying to make their town the last in the alphabet. You may see ZZZZX from 35000 feet if you fly from London Heathrow to LAX on a clear day, about ten minutes after you pass over Las Vegas.

You can enter a VR panorama of the Zzyzx road at:

<http://www.virtualguidebooks.com/SouthCalif/MojaveDesert/MojaveRiver/ZzyzxSpringsRoad.html>

You can see and read more about Zzyzx on the GDL Alliance website at: http://www.gdlalliance.com/GDL_Rocks/blvd_of_dreams.html

Why not Zala?: Having travelled in Hungary, I am somewhat astonished that the Hungarian team should have chosen ZZZZX, as Hungary is graced with dozens of placenames starting with Z, including the river Zala, and towns like Zador, Zabar, Zajk, Zajta, Zanka, Zirk, Zok, Zsira, Zsurk, Zics, Zomba, Ziliz, Zsana, and Zavod.



Do you need zzyzx?

Since Graphical Hotspots came in with ArchiCAD 8.x, there is less need for 'zzyzx' unless you want a single easy number to update in the settings box. If you have a more complex object with many parts, you will want to write a separate graphical hotspot routine for many of the components of the object which have a height value. In this case, I often write a routine in the Master Script of:

PARAMETERS zzyzx=0: HIDEPARAMETER 'zzyzx'

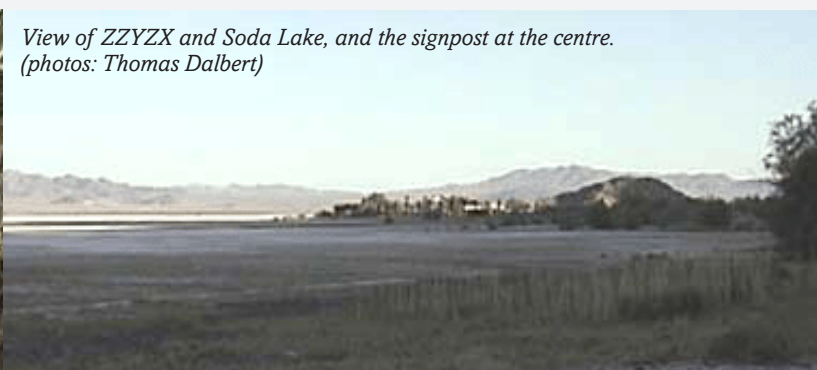
to make sure that it's clear that it cannot be used, and to remove irrelevant hotspots from the object's 3D view.



I want to go there sometime

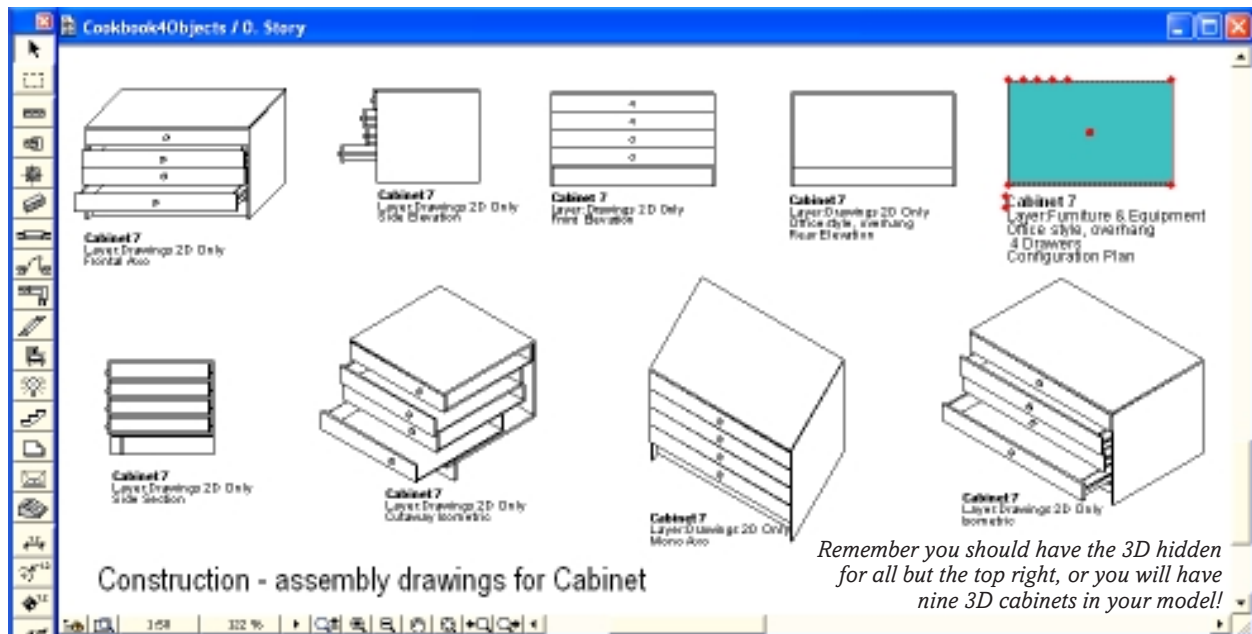


View of ZZZZX and Soda Lake, and the signpost at the centre.
(photos: Thomas Dalbert)

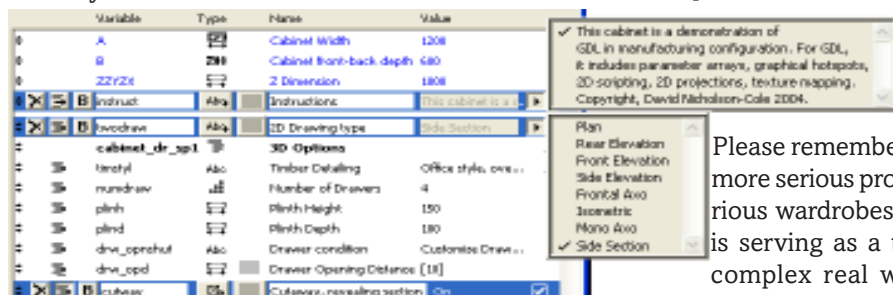


The Cabinet

GDL as a configuration & assembly assistant



WITH a minimum of coding, you can transform an object like our Cabinet to provide a range of construction drawings using the power of the PROJECT2 command, with its interesting parameters. So we can add four more parameters: for instructions, for 2D view type, drawer opening condition and 3D cutaway. Make ValueLists for the first three.



Short Instructions Manual / User Information

The first, 'instruct' is an example of providing instructions or information to the user in an easy way. Apart from writing good clear parameter descriptions, we thought the only way to get information to the user is to write a full blown User Interface? Well no, this method is a middle way. It can be quite wordy, and it is easily edited if you think of more to add later. Moreover, it's a good way to provide instructions/info even as part of a scripted User Interface, since it avoids all the problems of lengthy Outfield statements and the idiosyncratic word-wrapping and font-sizing and platform issues. As this isn't going to be used in other scripts, it's OK to put in the Parameter Script, along with the fontmenu.

```
!Parameter Script
VALUES 'font' 'Arial','Verdana','Times New Western'
VALUES 'instruct' 'This cabinet is a demonstration of',
  'GDL in manufacturing configuration. For GDL,',
  'it includes parameter arrays, graphical hotspots,',
  '2D scripting, 2D projections, texture mapping.',
  'Copyright, David Nicholson-Cole 2004.'
```

2D Viewing options

Think of the views you might want in an assembly or construction drawing. PROJECT2 gives you a good range of orthographic and parallel 3D views. It's a pity that it doesn't include a perspective view. The above graphic shows how it might look. The self labelling in the previous version of the Cabinet comes in perfectly here.

The idea is that the user configures their perfect furniture solution in Floor Plan and 3D. Please remember that we are really talking about more serious products like kitchen furniture, luxurious wardrobes, office suites. This little cabinet is serving as a teacher of techniques for more complex real world problem solving. Having configured the perfect solution, the user does a 'multiply' of the object alongside for as many views as they want. They move the duplicates to a 2D layer (so as not to confuse a scheduler into thinking the customer wants eight more cabinets). Change each cabinet to the 2D view of their choice. Each object labels itself with the correct view (neat feature) without any work by the user. We can include a Sectional view by providing a Cutplane command in the 3D Script. In fact with the 'cutway' parameter, any of these projections can be of cutaway views. If we had space, additional views might be the underside view, the alternative side elevation (if the object is non symmetrical), a section in the other direction, and a front elevation with a choice of doors open or closed (if it has doors).

Can it get better than this?

For a more systematic manufacturer, a single GDL object could make all these 2D assembly drawings in the one object and distribute them tidily on the sheet.


```

!-----
!Additions to Master Script Cab 7
!working out different drawings for construction
tw0='Plan'
tw1='Rear Elevation'
tw2='Front Elevation'
tw3='Side Elevation'
tw4='Frontal Axo'
tw5='Isometric'
tw6='Mono Axo'
tw7='Side Section'
VALUES 'twodraw' tw0,tw1,tw2,tw3,
          tw4,tw5,tw6,tw7
IF twodraw=tw7 THEN cutway=1

VALUES 'optlabl' CUSTOM, twodraw !Self Labelling

!Temporarily shut the drawers
cd0='Customize Drawers'
cd1='Close Drawers'
VALUES 'drw_opnshut' cd0,cd1
IF drw_opnshut=cd1 THEN
  FOR k=1 TO 10
    drw_opd[k]=0
    PARAMETERS drw_opd[k]=0
  NEXT k
ENDIF
PARAMETERS drw_opnshut=cd0 !Restore to 'Customize'

```

*Set the 'cutway'
flag if the chosen
view is a
Sectional view.*

*Neat trick for
labelling!*

*Shut the drawers,
then return the
option to
'Customize'.*

Master Script

Create the first ValueList as shown; it must be in the Master Script so that the 3D and 2D can make use of the results. The remaining two could be in the Parameter or the Master scripts.

If the view is to be a sectional cut, we set the 'cutway' Flag for the cutting, so that it would only happen in the 2D symbol or in GDL, but not in the 3D. For GLOB_CONTEXT, 1 is the GDL editing environment and 2 is for the 2D symbol drawing.

A very neat trick with 'optlabl' is that if you put the result of the 'twodraw' into a ValueList, it becomes the default value. The simple act of placing the object and configuring the 2D makes it self labelling, but still allows the user to adapt the label.

In some views, the user would like to have the drawers shut, so the 'drw_opnshut' ValueList gives them that option, and a little loop runs to shut all the drawers. Then the parameter is restored to 'customize'.

```

!Cabinet with Drawers
!2D Script

HOTSPOT2 0,0
IF twodraw=tw0 THEN GOSUB 100:!Plan shape
IF twodraw=tw1 THEN PROJECT2 4,270,2
IF twodraw=tw2 THEN PROJECT2 4,90,2
IF twodraw=tw3 THEN PROJECT2 4,180,2
IF twodraw=tw4 THEN PROJECT2 6,90,2
IF twodraw=tw5 THEN PROJECT2 7,90,2
IF twodraw=tw6 THEN PROJECT2 8,90,2
IF twodraw=tw7 THEN PROJECT2 4,180,2

GOSUB 110:!Labels

END:=====

```

2D Script: short and sweet

Put almost all of the previous routines of the 2D Scripts into two subroutines. We only want the plan if plan view is selected. We only want one hotspot now, for the origin-corner. Now we can enable all of the remaining views using easy one line IF-statements.

3D Script: even shorter!

All we need to do is to provide a Cutplane routine that will move inboard by just more than the thickness of the drawer sides and display a 3D Cutaway of the cabinet.

The executive script is now as short as you can get it – the cutting routine is best left up here, not as a subroutine. Subroutines are meant to be stable 'quasi-objects', but a Cutplane routine is like a lighted match – waiting for the Cutend.

```

!Cabinet with Drawers
!3D Script

PEN gs_cont_pen
IF cutway THEN
  ADDx MAX(bthk1+bthk2*3,A/3)
  ROTy -90
  CUTPLANE
  DEL 2
ENDIF

GOSUB 1000::Hotspots Vertical
GOSUB 100:!Cabinet main shape
GOSUB 200:!All Drawers

IF cutway THEN CUTEND

END:=====

```

*If there is a
sectional view, the
CUTPLANE is
activated, as is the
CUTEND.*

Attribute Definitions: IND()

IND() provides an index number for a Material, Text Style, Texture, Line_Type or Fill. Mostly these can be specified by name in the form of a string, but there are times when they have to be specified by number. IND() does this conversion for you.

In a MATERIAL statement, it helps to specify a material by its index number, not name. In a Material definition you need to specify a previously defined Fill or Texture by its index number – so you need IND().

It is most useful where you think that users have tweaked their materials library and changed some of the default numbers. For example, IND(MATERIAL, "mypaint") will guarantee getting the 'mypaint' you de-

fined, whereas MATERIAL 18 might get a quite different result if someone has re-used 18 for a type of brick. You can use it to do the same if users have tweaked their **Linetype** and **Fill** numbers.

When you make your own material in GDL using the DEFINE command the new material gets allocated a number by ArchiCAD. To be sure, type in:

```
MATERIAL IND(MATERIAL, 'dnc_brick')
```

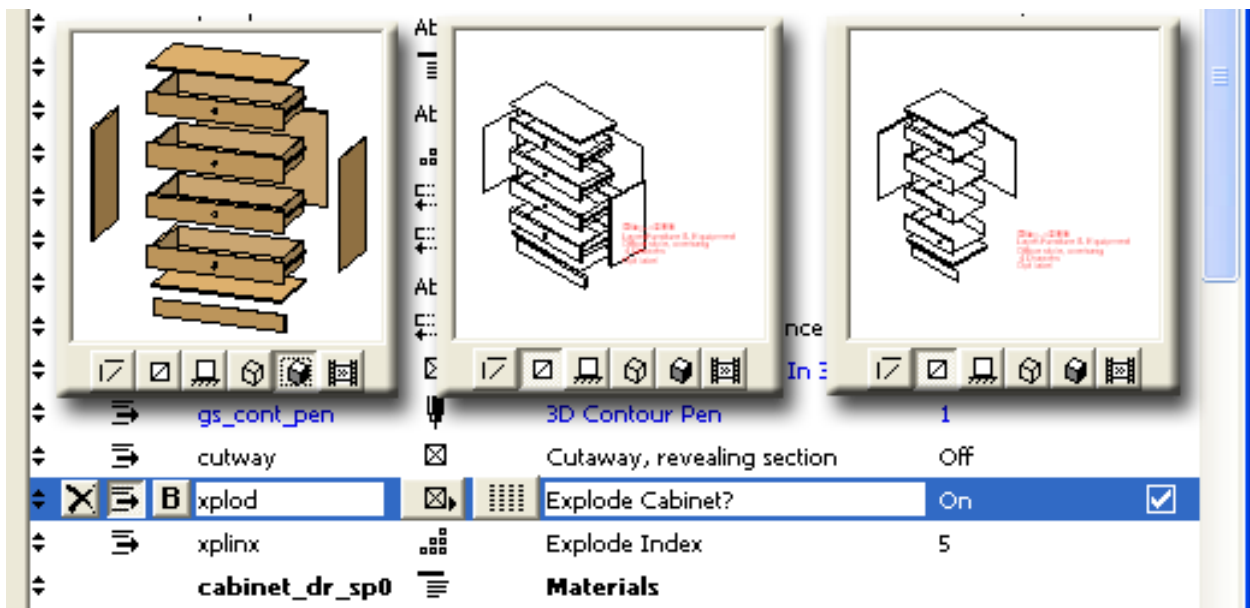
A new Material definition using a Texture called 'woolly_carpetTX' might be written:

```

DEFINE MATERIAL 'woolly_carpet' 22,
  red, green, blue, 0,61,
  IND(TEXTURE, 'woolly_carpetTX')

```

The Cabinet: Explode the 3D GDL!



Simulating the explosion

ONE does not do this for fun! It has a serious use, extending the power of the GDL to generate construction and assembly drawings from a custom configured object. This makes the ideal drawing for flatpack quick assembly constructors – all we need now is some tiny GDL screws to complete the package.

The 3D and all the 2D drawings in the previous section can be exploded at the will of the user. All you need is an ‘explode index’ as above, and then to apply that to all the parts. We could also improve the Hotspots on the various 2D representations.

Parameter and Master Scripts

Add the two parameters, and add these lines to the scripts:

```
!Master Script : Cabinet 8 additions
xpld=xplod*xplinx !Set degree of explosion
```

This sends a value to the 3D so must be in the Master Script, but the statements below are purely about parameters and can be in the Parameter Script.

```
!Parameter Script : Cabinet 8 additions
!Set or Hide Explode index
VALUES 'xplinx' 0,1,2,3,4,5,6,7,8,9,10
IF xpld=0 THEN HIDEPARAMETER 'xplinx'
```

3D Script

The changes all occur in the Subroutine 100 to make the cabinet. We could also explode the drawers, if we wanted to take it further.

We insert ‘xpld’ values in all directions outwards, including downwards for the floor and plinth. Most of the movements are easy, but we have to do some tricks with the roof to make sure there’s enough to explode all the drawers too.

By setting 11 levels of explosion, we make the best looking explosion as value 4, and allow the user to vary the degree expansion from zero to 10, either side of 4. The decisions on the mini-distance values for

```
!Modify the Cabinet 3D Subroutine
100:!Cabinet main shape
!Side Panels
MATERIAL cab_mat
ADDx -xpld*0.1
BLOCK bthk1,B-ts*bthk1,zzyzx-ts*bthk1 !Left
DEL 1
ADDx A-bthk1 +xpld*0.1
BLOCK bthk1,B-ts*bthk1,zzyzx-ts*bthk1 !Right
DEL 1
txdir=2:GOSUB 999:!Texture

!Back panel
MATERIAL con_mat
ADD bthk1, -xpld*0.1,plinh
BLOCK A-bthk1*2, bthk2, zzyzx-bthk1-plinh
DEL 1

!Roof/Worktop
MATERIAL cab_mat
ADD bthk1*(1-ts),0,
zzyzx-bthk1 -xpld*0.08+xpld*0.05*numdraw
BLOCK A-bthk1*2*(1-ts),B,bthk1
DEL 1

!Floor
MATERIAL cab_mat
ADD bthk1,bthk2, plinh -xpld*0.1
BLOCK A-bthk1*2,B-bthk2-ts*bthk1,bthk1
DEL 1

!Plinth
MATERIAL cab_mat
ADD bthk1,B-plind-bthk1, -xpld*0.125
BLOCK A-bthk1*2,bthk1,plinh
DEL 1
txdir=0:GOSUB 999:!Texture
RETURN
```

The roof height is dependent on the number of drawers. The roof goes down to the floor, then up by the number of drawers (plus a bit).

explosion are a matter of judgement. If they look OK with index 4, just check that they look OK with level 1 and 10 before saving the object.

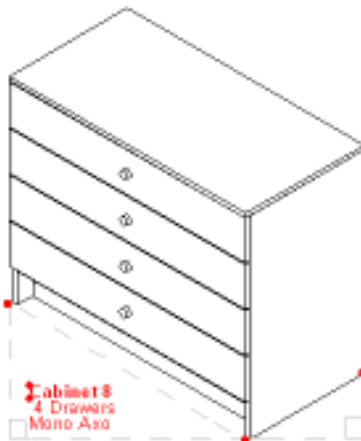
The Cabinet: Explode the 3D!

Improve the 2D Hotspots

We have to elongate the IF statements in the 2D Script to include some additional Hotspots. This is easy enough for the orthographic views, but requires some trigonometry to work the hotspots into the 3D projections. The Axonometrics and Isometrics work on angles of 45° and 30/60° so it doesn't take long to get the Hotspots in the right place.

For the different 2D projections, the text might be in an inconvenient place for an AC7 GDL writer. Now, with Graphical Hotspots, you can allow the user to reposition and resize the text without difficulty.

The diagram shows the 60/30/90 triangles which help you work out the hotspot locations.



Anything else?

Self dimensioning, and output of the panels to 2D for further output to DWG or G-Code would complete the whole trick!

```
!Cabinet with Drawers
!2D Script

HOTSPOT2 0,0 !They all have this one
IF twodraw=tw0 THEN GOSUB 100:!Plan shape
IF twodraw=tw1 THEN !Rear Elev
PROJECT2 4,270,2
HOTSPOT2 A,0
ENDIF
IF twodraw=tw2 THEN !Front Elev
PROJECT2 4,90,2
HOTSPOT2 -A,0
ENDIF
IF twodraw=tw3 THEN !Side elev
PROJECT2 4,180,2
HOTSPOT2 -B,0
ENDIF
IF twodraw=tw4 THEN !Frontal axo
PROJECT2 6,90,2
HOTSPOT2 -B*COS(45)/2, -B*COS(45)/2
HOTSPOT2 -B*COS(45)/2-A, -B*COS(45)/2
ENDIF
IF twodraw=tw5 THEN !Isometric
PROJECT2 7,90,2
HOTSPOT2 -B*COS(30), -B*SIN(30)
HOTSPOT2 -(A+B)*COS(30), -(B-A)*SIN(30)
ENDIF
IF twodraw=tw6 THEN !Monometric Axo
PROJECT2 8,90,2
HOTSPOT2 -B*SIN(30), -B*COS(30)
HOTSPOT2 -B*SIN(30)-A*COS(30),
-B*COS(30)+A*SIN(30)
ENDIF
IF twodraw=tw7 THEN !Section
PROJECT2 4,180,2
HOTSPOT2 -B,0
ENDIF

GOSUB 110:!Labels

END: !=====
! The rest remains unchanged....
```

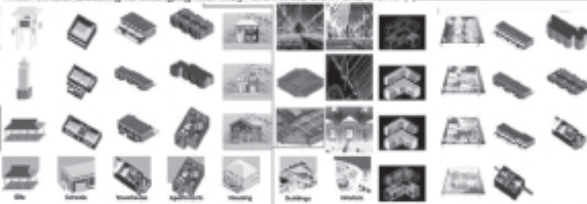
Reflections on GDL: the Genome Object Project

KIMON Onuma, president of the GDL Alliance promotes the idea of GDL as analogous to the genome: DNA is a compact (but complex) molecule that contains all the data and rules necessary to construct the organism it represents – in nature this is for constructing living creatures. The human genome has found a way to make 6 billion people all different, but mostly following similar rules (number of arms and legs, head location, reproductive mechanisms).

In the artificial human world we can see examples of growth and form and evolution following rules. Greek Architecture is an example: from a simple set of construction and decorative rules originating from prehistoric timber construction, it went on to be used, extended and re-interpreted: first by the Greeks, then the Romans, the Byzantines, the Islamic world, with reviv-



The Virtual Building is changing the way architects think and work.



als at regular intervals, such as the Renaissance, Baroque, Rococo, Soanian, Greek Revival, Edwardian, Imperialist neo classical and even Post-Modern!

In this world we try to find better and better ways of encoding construction information to construct anything from microchip to ships and skyscrapers.

GDL has the ability to provide data and rules for numerous permutations of essentially the same object. Smart Object Technology even includes an evolutionary cycle: as more tools are added, the writers of GDL get smarter and so do the objects they make.

With a UI and Graphical Hotspots, one single GDL object can contain all the Kitchen Cabinets of a manufacturer's range, including configurations that do not exist but can be user-customised. The GDL object can permit additional small variations – the result being millions of permutations.

In nature, starting with the code in a single cell, a reproductive mechanism starts the process of cell division, and by taking on nutrients and oxygen and surviving in an environment becomes a vastly larger complex organism – like a human being. In GDL, 10 or 50k of smart code can flourish in the environment of ArchiCAD using the GDL drawing and rendering engine to construct virtual models of infinite variety.

The Cabinet: Dimension and Reflection

LET'S just dimension the Plan view so that when the cabinet is configured, the user gets direct feedback, and their construction drawing is even more useful. Why put the user through dimensioning the drawing if the GDL can do it?

Reflection: GDL as a configurator

This GDL object is in a sense a prototype for a whole production technology in furniture making. If we take the object further and make more use of macros (external GDL files called by the script), the subroutines 100 and 200 could contain different cabinet and drawer designs, we could add subroutines for doors and handles and slider designs.

The GDL Genome idea

We could make this object a sort of DNA molecule for a whole family of furniture. The routines which make the menus for 2D representation, configuration and dimensioning and labelling would be need very little alteration to accommodate designs for a Bed or a Table or anything that works with A, B or zzyzx. This approach can also be applied to windows and doors which demand a high level of modularity due to the many variations of user choice – I have worked on a window design that allows a zillion variations based on one procedure for configuring the casements.

Adding the Dimensions

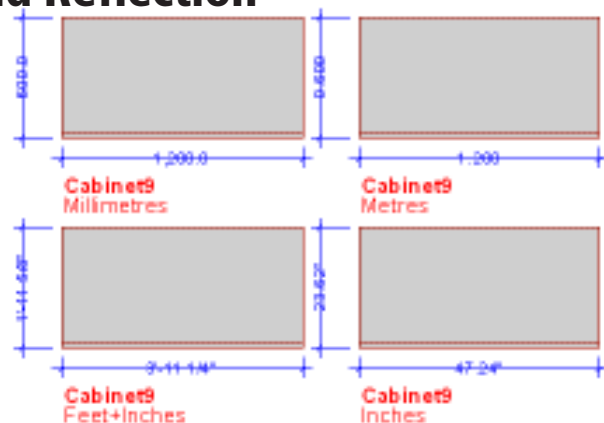
First of all, in the Master Script we should add an extra option for the 2D view, a 'true view plan', so that we can have a view of the exploded object that is in keeping with the other exploded drawings. We will apply dimensions to the plan and side elevations only. We could dimension subsidiary parts of the elevations and sections (plinth, drawers) if space permitted.

Dimension lines, witness lines

When the dimensions are shown, they all need to be a certain distance from the object and must fit the drawing well – so the user must be able to position the dimension, and pull them into shape with GHs. Make a new parameter called 'dimdis' for dimension distance. As this could be used for plan and elevations, we can make a Parameter Array, so that all dimensions are stored in one parameter – even if they are applied to elevations and sections. Taking it further, you might consider linetype and pen, and the design of the crosshair. We'll use a small overshoot, and circles – if space permitted I could add more intersection styles.

Text considerations

You should also consider parameters for dimensioning pen and text size. A fully developed product could also have a routine to rotate dimension text so that it is always the right way up on the drawing. It also needs to truncate the dimension text to a controlled number of decimal places and offer a choice of metric or imperial. Dimensions in GDL are always in metres, but if we are to offer the user the range of choice that Archi-



You can dimension something after it is made using AC's dimensioning tools. But smart objects should do it for you!

CAD offers, we need to borrow the techniques from the earlier Dimensioning Tool and use String formatting to make the texts suit the user's requirements.

Master Script and Parameters

Create the parameters we have discussed above. The font size will be a real world dimension, not a plotting height. In the Parameter Script, we can hide the parameters that are not needed if there are no dimensions to be shown. This is optional, we didn't do it with the labels. The ValueLists need to be written into the Master Script because we make use of the variables and flags such as 'dun0' and 'fp'.

cabinet_drawer_8_9

Storage ☐ Template ☒ Placeable

New Delete Set Details... Select Subtype...

Variable	Type	Name	Value
optlabl	Abc	Text for Optional Label	Opt label
B shodim	<input checked="" type="checkbox"/>	Show Dimensions ON/OFF? On	
dim_pen		Dimension Lines+Text pen	35
dim_ins	Abc	Dimensioning Intersection style	Crosshair
dim_ovr		Dimension Overshoot	50
dim_unit	Abc	Units - Metric/Imperial	Millimetres
dim_fpos	Abc	Dimension Text Position	Centre
dim_fsiz		Dimensions Font Height	60
cabinet_dr		Hidden Parameters	
B dimdis		Dimensioning Distance	[8]
txtx		Text Location X	100
txty		Text Location Y	100
bthk1		Primary Board thickness	20

```

!-- Cabinet 9 additions to Parameter Script --
IF NOT(shodim) THEN HIDEPARAMETER 'dim_pen',
'dim_ins','dim_fsiz',
'dim_ovr','dim_fpos','dim_unit'

!-- Cabinet 9 additions to Master Script --
ch0='Circles'
ch1='Crosshair'
VALUES 'dim_ins' ch0,ch1
chd=dim_ovr*0.3 !Crosshair half dimension
dun0='Metres'
dun1='Millimetres'
dun2='Feet+Inches'
dun3='Inches'
VALUES 'dim_unit' dun0,dun1,dun2,dun3
VALUES 'dim_fpos' 'Above','Centre','Below'
IF dim_fpos='Above' THEN fp=8
IF dim_fpos='Centre' THEN fp=5
IF dim_fpos='Below' THEN fp=2
  
```


The Cabinet: Dimensions

2D Script

This whole routine is only carried out if dimensions are chosen. Although we are only doing the Plan dimensions here, we need to think ahead. We can write a general routine that will cover any dimension and any parameter in the X direction, and do the same for a dimension and parameter in the Y direction. Any dimension in the Z direction in 3D is also going in the Y direction in the 2D. We can use a 'd' counter to decide which dimension is being stretched by the hotspots.

We also want to ensure that the dimension is readable no matter how the object is rotated so use a Global Variable to straighten out the text.

We use a general variable 'paradm' so that the routine can be used for any dimensions in the X or Y direction for the text and line lengths and intersections.

```
!Cabinet with Drawers
!2D Script

HOTSPOT2 0,0
IF twodraw=tw0 THEN
  GOSUB 100:!Plan shape
  GOSUB 500:!Dimensioning Plan
ENDIF

IF twodraw=tw00 THEN !Plan Trueview
  PROJECT2 3,270,2
  HOTSPOT2 A,0
  HOTSPOT2 A/2,B/2
  GOSUB 500:!Dimensioning Plan
ENDIF

!-----
The remainder of the script as far as the END
statement is unchanged. Subroutine 500 governs the
dimensions.

END:!======

500:!Dimensioning Plan
IF shodim THEN
  PEN dim_pen
  DEFINE STYLE "diasty" font,
    dim_fsiz*1000/GLOB_SCALE,fp,0
  SET STYLE "diasty"
  d=1:paradm=A
  GOSUB 550:!Dim lines X
  GOSUB 540:!Format string
  ADD2 A/2,dimdis[d]
  ROT2 -SYMB_ROTANGLE
  TEXT2 0,0,string
  DEL 2
  d=2:paradm=B
  GOSUB 551:!Dim lines Y
  GOSUB 540:!Format string
  ADD2 dimdis[d],B/2
  ROT2 90-SYMB_ROTANGLE
  TEXT2 0,0,string
  DEL 2
ENDIF
RETURN
```

```
540:!Convert dimension to correct units
IF dim_unit=dun0 THEN string=STR('%0.3 m', paradm)
IF dim_unit=dun1 THEN string=STR('%0.1 mm', paradm)
IF dim_unit=dun2 THEN string=STR('%0.8 ffi', paradm)
IF dim_unit=dun3 THEN string=STR('%0.2 di', paradm)
RETURN
```

```
550:!Dim lines X
GOSUB 1070:!Hotspot dimension distance
  LINE2 -dim_ovr,dimdis[d],
    paradm+dim_ovr,dimdis[d]
  LINE2 0,-dim_ovr, 0,dimdis[d]-dim_ovr
  LINE2 paradm,-dim_ovr,paradm,dimdis[d]-dim_ovr
  GOSUB 580:!Crosshairs etc
RETURN
```

```
551:!Dim lines Y
GOSUB 1071:!Hotspot dimension distance
  LINE2 dimdis[d],-dim_ovr,
    dimdis[d],paradm+dim_ovr
  LINE2 -dim_ovr, 0,dimdis[d]-dim_ovr, 0
  LINE2 -dim_ovr,paradm,dimdis[d]-dim_ovr,paradm
  GOSUB 581:!Crosshairs etc
RETURN
```

```
580:!Crosshairs etc for dimension X direction
IF dim_ins=ch0 THEN
  CIRCLE2 0,dimdis[d],chd
  CIRCLE2 paradm,dimdis[d],chd
ELSE
  LINE2 -chd,dimdis[d]-chd,chd,dimdis[d]+chd
  LINE2 paradm-chd,dimdis[d]-chd,
    paradm+chd,dimdis[d]+chd
ENDIF
RETURN
```

```
581:!Crosshairs etc for dimension Y direction
IF dim_ins=ch0 THEN
  CIRCLE2 dimdis[d],0,chd
  CIRCLE2 dimdis[d],paradm,chd
ELSE
  LINE2 dimdis[d]-chd,-chd,dimdis[d]+chd,chd
  LINE2 dimdis[d]-chd,paradm-chd,
    dimdis[d]+chd,paradm+chd
ENDIF
RETURN
```

```
!Add these Graphical Hotspot routines
1070:!Hotspots:Dimension distances Y direction
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,dimdis[d],129
hsid=hsid+1 !Move
HOTSPOT2 0,dimdis[d], hsid,dimdis[d],2
hsid=hsid+1 !Vector
HOTSPOT2 0,-1, hsid,dimdis[d],3
RETURN
```

```
1071:!Hotspots:Dimension distances X direction
hsid=hsid+1 !Base
HOTSPOT2 0,0, hsid,dimdis[d],129
hsid=hsid+1 !Move
HOTSPOT2 dimdis[d],0, hsid,dimdis[d],2
hsid=hsid+1 !Vector
HOTSPOT2 -1,0, hsid,dimdis[d],3
RETURN
```

What are IFCs?

INDUSTRY Foundation Classes are a system of classification of information about components and element in the building industry – designed to improve the building and design and manufacturing processes by defining a common language for construction between software applications. The project is driven by the International Association for Inter-operability (IAI). Graphisoft are fully committed to IFCs, because at its heart the IFC idea revolves around the principle of the Virtual Building or BIM (Building Information Model). GS have pioneered this for over 20 years and can only benefit by supporting the movement – although, at the time of writing, it seems Autodesk are ignoring IFCs. GS have been developing Archi-FM for several years, and IFCs are essential to managing facilities based on a project model. Correct use of IFC classification for objects reduces the risk/uncertainty factor in design and management of buildings. GDL technology gives GS a head start in the IFC movement, because it has been an object-oriented approach from its beginning.

What do you need to do to ensure that your GDL library parts are IFC compatible? At this stage you must ensure that you select the correct subtype for your object. If you are making Windows or Skylights, you have no choice – but it is possible to apply the correct subtype even if you are only making general 3D objects, such as a chair, table or storage unit. If you select a subtype, GDL will place into the parameters table a number of pre-organised parameters (in blue font) which you are advised to use. For example, if you and everybody else uses 'gs_frame_mat' for the frame material in a Window, then you are giving your object a longer life in the future by providing compatibility.

It is most valuable in this multilingual world to have a IFC, a system which crosses the boundaries of country and language. For example, an object defined as 'Seating' will be seen as such by every regional version of ArchiCAD, even if it is a library object called 'sedia' or 'chaise'. Some APIs have to operate on libraries, and they will now recognise subtypes, whatever their language.

If you are into serious GDL, for manufacturing, you can define your own subtypes, using the 'template' checkbox in GDL.

If you wish to know more, just GOOGLE with "Industry Foundation Classes".

Quality Control: Finishing

WHEN you have made an object or a library for others to use, put it through a series of tests to ensure that it meets good standards. If you wish to transfer an object or a library of objects to another user, the safest way is to save it as an PLA Archive. If your object has macros and associated image files, they will be included. You should tick the **Options** button to see what else is included. Avoid including all of the loaded libraries or your archive file could exceed 100megabytes!!! Only include Textures if you have special ones. If your objects use normal ArchiCAD textures, untick the textures checkbox.

It's annoying that you can only save an archive in the version you are in. If you wish to send someone an object library say in version 8.0 from 8.1, save the PLN file as an 8.0, open it in 8.0, then save as an archive from 8.0.

If you use the student version of ArchiCAD you cannot send someone (outside the world of education) an planfile or an archive, so put the .GSM files and any other little macros or GIFs into a ZIP file and send that.

Let's assume that you have saved the floor plan correctly with the library into an archive. Transfer it by floppy or zip or email to a different platform, Mac or PC, to make sure that the whole thing unpacks correctly on the other machine. The library objects should have been saved in the correctly named folders and they should all show correctly in the newly created floor plan.

- View the floor plan. Check that the objects are there. Look for coloured dots (missing objects). You can write an instruction text file into the floor plan on the use of the library. You could label each object, and put boxes round groups of related objects. Objects which have more than one 2D representation or 3D configuration should be placed enough times to show all the versions.
- Check the 'Missing Library Parts' list to make sure that you haven't lost any. Check that you haven't included library parts that you didn't want included.
- Ensure that your Plan file is clean (not having wierd personalised materials or pens). You could upset your end user because they will get your settings.
- Check the settings of each object to reassure yourself that the parameter descriptions look right, are in the right language and that the 3D view displays correctly.
- If you have edited the object's GDL a lot, delete it and place it again in the floor plan so that its database entry in ArchiCAD is refreshed, before you archive it.
- Check that all objects have an icon in the browser (if not, they need a Preview image).
- Make sure you have saved each object with sensible default values.
- Delete 'cache' and other unwanted binary files like .gs& files. If your library objects have to work in listing, then run some listings to check that they display correctly.
- Delete section/elevation views from the floor plan. These inflate file size alarmingly.
- Don't send .GSM files directly in email, they often lose their preview image or subtype definition – very confusing! On PCs, they may be treated as movie files!

If you are quite happy, you could now re-archive the file, put it in a ZIP archive, and it is ready for sending to the recipient.

Professional Standards in GDL

Can GDL be a profession?

IT'S a good point at which to ask how far we (or others who learn it) can take GDL. The normal architectural work you do may not be demanding enough to need all these GDL skills, but you may still see a purpose in continuing with GDL – to build investment objects for your office, to sell your objects to others, or to work with a manufacturer on a product range.

Manufactured products – this is a major area where GDL should go to. ArchiCAD users and Graphisoft would like GDL to be a world platform for 3D object description. Every manufacturer who now thinks that it is essential to have their products in DXF should now be thinking about having the same range of products available to ArchiCAD and AutoCAD users in the form of GDL. This will help to get their product specified. When they see the reality of Smart, Parametric objects, they would benefit from using GDL. But they may only see this if there is pressure from Users. Don't wait for Graphisoft to do this, do the asking yourself! Many of their existing DWG/DXF libraries can easily be adapted to GDL with a bit of smartness and modularity on the part of the GDL author.

A single GDL object can contain dozens, or even

millions of configurations of a product, just as we have explored in a modest way with the GDL cabinet with drawers. The power of GDL as a **Configurator for custom manufacturing** is perhaps the most significant change that GDL can bring.

GDL needs a new class of user – GDL should be available in a form separate from ArchiCAD. It should be available to lower paid technical staff who just want to do GDL. Graphisoft cannot expect a new class of dedicated GDL technicians to appear from nowhere, all capable of paying thousands of dollars for the whole of ArchiCAD when they just need a development environment. Something like the student version would achieve this. Trained GDL writers could write better and more professional products than inspired architectural dabblers.

The GDL Alliance, formed in 2000, is committed to developing GDL as a professional activity. The definition of achievement levels, the promotion of GDL as a programming language, the enabling of the creation of a new technician class of GDL writer – these are all part of that mission.

Developing GDL Professionally

IF you are a developer these are what you should aim to achieve. If you are a customer, these are what you should expect.

- **Select the correct Sub-type** classification and use standard GS-recommended parameter names.
- **Structured scripting**: scripts should be debuggable and logical, capable of being repaired, extended, updated by you or by others.
- **Signed and dated**: scripts should identify the author (if you take responsibility for your work), date of creation and all subsequent amendments.
- **Maintainable in future**: should be documented with comments for each stage of the script.
- **Economical LOD**: object renders quickly with fewest polygons, and it's quicker to write.
- **Scale sensitive LOD**: in 2D and 3D – smart.
- **Consider 3D wireline view**: should draw correctly on a pen plotter, not just look good in 3D render.
- **Stretchy**: stretchy objects are more user friendly, but they should NOT stretch if the objects are required to have fixed size.
- **Appropriate Hotspots**: intelligent 2D and 3D hotspots make it easy for the user to pick up, identify, snap and stretch (don't rely on the bounding box!).
- **Graphical editing hotspots**: allow GDL objects to be manipulable, e.g. opening doors, turning knobs.

- **Cascading grouped parameters**: Group them into logical groups: Materials, Pens, 3D config., 2D etc.
- **User Interface**: your object will look more like an API with its extra level of quality – but this quality will take a lot of time to design and write.
- **Preview window**: a 128x128 pixel image in the Preview window box will give you a nice browser icon.
- **Comment**: provides your object with a URL.
- **2D Scripted**: objects must not only be good in 3D, they must be quick to draw, look good in the floor plan, and be scale sensitive, have good hotspots.
- **Correct 2D and sectional lines, fills etc.**: looks correct in plan view, and when viewed in section.
- **Self scaling, self orienting labelling**: versatile.
- **Error correcting**: prevent crashes or non appearance of the object. Correct errors, if the user enters bad data, or use ValueLists to guarantee good data.
- **Include Material definitions**: so your objects are portable between countries and versions.
- **Property**: descriptor and components – essential for schedules. The manufacturer needs this.
- **Dimensional environment**: objects must work when used in any measurement system.
- **Libraries organised**: so you can find objects – group windows, doors, stairs, macros etc. logically in their own folders. Use sensible names that do not replicate existing library parts.

Advanced 3D

TUBE

TUBE is a profoundly useful command. You can draw a profile through a series of X-Y-Z points in 3D space, thus creating complex frameworks without having to use Trigonometry. TUBE does its best to work out a mitring solution at each change in the pathway (sometimes not very well).

TUBE Profile

With TUBE, you define the 2D outline/profile, just as you do in **Extrude** or **Poly2**.

TUBE Pathways

You define a number of path points that the section must pass through – as in **Sweep**. At each joint, GDL mitres the junction.

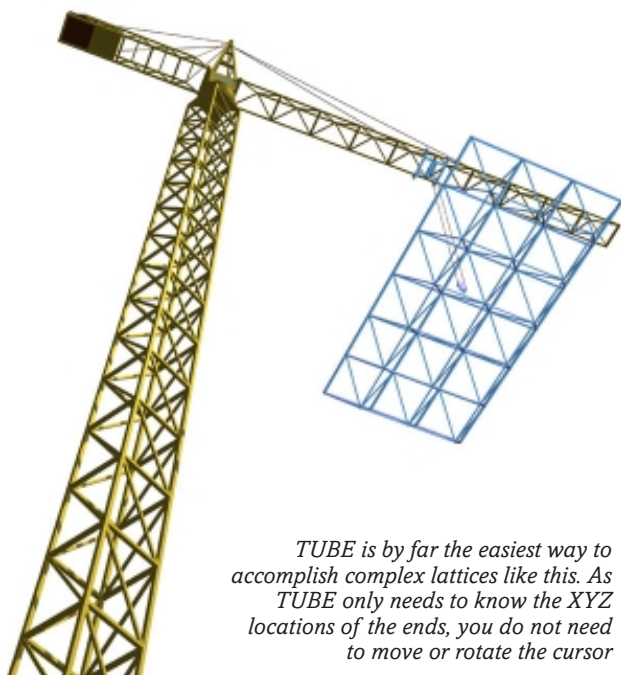
Design Problems

Because the Mitring and Rotational angle can be set with each joint, TUBE requires a Starting and Ending Direction, so it knows how to treat the start and end surfaces.

So you must position a 'Phantom Start' and 'Phantom End' point. This will correctly mitre the start and end of the tube object. You need to type in some examples to appreciate quite how it works. The example here puts a black ball at each phantom point.

The Pathway is defined in XYZ locations – a bit like in Sweep. In Sweep, the section changes by the same scale and angle change, and moves from XYZ point to XYZ point. In Tube, the section remains constant, but you can have a different twist angle, at each mitring opportunity.

TUBE works best if the section is kept simple, and is axially centred on the origin – like a cylindrical or rectangular profile.



TUBE is by far the easiest way to accomplish complex lattices like this. As TUBE only needs to know the XYZ locations of the ends, you do not need to move or rotate the cursor

!TUBE Demonstration

```
!Syntax:- TUBE n,m,mask,
! section-> u1,v1,s1,...un,vn,sn,
! pathway-> x1,y1,z1,twistangle1,
!           ->...xn,yn,zn,anglen
LET p=0.05 !one pixel
```

BODY -1

```
TUBE 4,5+2,63,
0,0,0,
0,p,0,
p,p,0,
p,0,0,
0,0,-p,0,!phantom start
0,0,0,0,!real start
-p,0,p*3,0,
p*2,0,p*4,0,
0,0,p*5,0,
0,0,p*6,0,!real end
0,0,p*7,0,!phantom end
```

Note, here, the use of 'p' to save having to type in many zeroes.

BODY -1

TUBE works out its own mitres, and wierd things happen if you start trying to twist the section. For safety, always use a mitre/twist value of zero.

In this example, I have used the mysterious command called BODY -1 which does an integrity check on the 3D form. It's a lifesaver with TUBE which can often display irrational errors.

!Display Phantom points

```
MATERIAL 0
ADD 0,0,p*7
SPHERE 0.01
DEL 1
ADD 0,0,-p
SPHERE 0.01
DEL 1
```

Phantom Start and End

In this script, a little ball has been placed at the Phantom Start & Phantom End points to illustrate the principle.

Masking: 1=Base surface, 2=End surface, 16=Base edges, 32=End edges, 64=Cross section edges visible

One trick you can do is to lay out a Fill in plan, save it as a temporary 2D library object, and then steal the XY locations, copying and pasting them to the 3D Script to form the profile points for a TUBE command. You could also use a Line tool dragged into the 2D window to give you the path points for the Tube. This is how the Add-On 'Profiler' works.

TUBE+TUBE Mental Health Warning!

TUBE causes more grief than most – because if you don't get it right, you get error messages. Once you understand it, though, you may find yourself typing in TUBE scripts with confidence. My solution is to use a series of PUT statements first to ensure that one major source of syntax errors is removed.

However, the way that the sectional outline is drawn in TUBE is verging on the irrational. It is supposed to follow the 'looking from above' rules of the other 3D elements. But then it jumps about the origin once the location points are entered. It works more reliably if the direction of the tube is mainly horizontal. A lot of trial and error may be involved to get it right. Circular tube profiles always work well.

A BODY -1 command before and after the TUBE is advisable.



Illegal status value.
at line 309 of file GDL_Cmds.

Stop

Continue

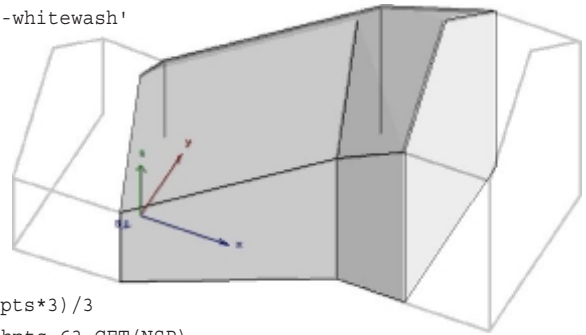
TUBE

ARCHICAD has a **TUBE** command. It's different from TUBE, in that there is no Twist Angle involved at each junction; it is not better than TUBE, just different.

The section defined is drawn onto the YZ plane using the same syntax as REVOLVE (which has to be drawn onto the XY plane), and is 'extruded' along the X axis. The section defined is the section at the mitres, so there seems to be a danger of the actual linear section changing erratically along its length if the pathway is moving a lot.

A good feature is that if you leave the section description unclosed, it will force the section to 'grow' downwards to meet the XY plane, like a wall of changing height.

```
PEN 1
MATERIAL 'surface-whitewash'
PUT -1.0,1.4,0,
    0.0,2.8,1,
    1.2,1.5,1,
    1.2,1.0,1
profpts=NSP/3
PUT -1,0,0,
    0,0,0,
    2, 1, 1,
    3, 1, 1.6,
    4, 1, 1.6
pathpts=(NSP-profpts*3)/3
TUBE profpts,pathpts,63,GET(NSP)
```



TUBE needs phantom points just like TUBE, so the example here shows the finished object, and the phantom points in wireline. Phantom start and stop points tell the object how to mitre its ends.

The routine uses PUT & GET, *profpts* and *pathpts* and is a classic way of doing TUBE and TUBE with the lowest risk of errors. This method ensures that you know where the definition of the profile ends and where the pathpoints begin.

Masking: 1=Base surface, 2=End surface, 16=Base edges, 32=End edges, 64=Cross section edges visible

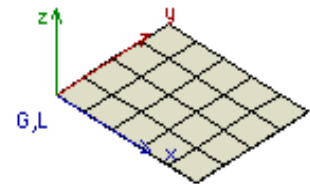
MESH

MESH – is limited by being applicable to rectangular entities only.

However, it is easy to use, as you only need to enter the Z -height of each point. Write the mesh out with all points zero at first. Set them out with the same grid as the 'n' and 'm' factors. Then try tweaking some points upwards to reshape the terrain. Although almost obsolete as a 3D shape, it is an ideal use for parameter arrays.

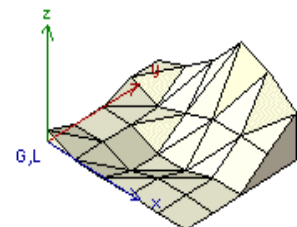
As you view it in the script & then in the 3D View, the positions of the meshes appear inverted. The bottom left on the mesh is the top left in the script. The top right of the mesh is the bottom right in the script. It is as if you were viewing it from upside down.

```
!MESH Demonstration
!Syntax: MESH a_width,b_length,
!         m,n,mask,
!         z11,z12,z13,z14,..z1m,
!         z21,z22,z23,z24,..z2m,
!         ..zn1,zn2,zn3,zn4,..znm
MESH 15,12,6,5,63,
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0
```



Start off with a flat mesh (all the heights at zero), then tweak the points up and down.

```
!MESH Demonstration
MESH 15,12,6,5,63,
    1,0, 0, 0,0,0,
    0,0,-1, 0,0,0,
    0,0,-1,-1,0,0,
    1,0, 1, 2,2,2,
    1,2, 4, 7,6,4
```



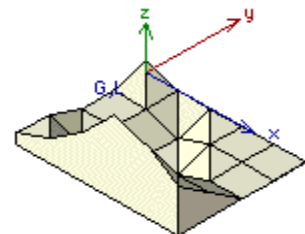
Masking: 1=Base surface, 4=Side surface, 16=Base & side edges, 32=Top edges, 64=Top edges & surface rough.

Above – it appears inverted
Below – it now appears corrected with the MULy -1 command

Hot MESH Tip!

If you precede the MESH command with a MULy -1 then the script and the object will be the same way up, and it is easy to enter heights (even though the top left corner is now at the origin).

```
!MESH Demonstration
MULy -1
MESH 15,12,6,5,63,
    1,0, 0, 0,0,0,
    0,0,-1, 0,0,0,
    0,0,-1,-1,0,0,
    1,0, 1, 2,2,2,
    1,2, 4, 7,6,4
DEL 1
```



Notice that the grid is rectangular, but if a surface within a rectangle of the mesh is not planar, it is divided into two triangles, and thus can be rendered as two planar surfaces. With triangular subdivisions, ANY surface, however complex can be rendered. It could be quite useful for doing a site, but you cannot drill holes in it, or increase local detail, or vary its outline. The MASS command is more useful for sites, and permits irregular shapes and allows holes to be drilled.

COONS

Freeform surface making

NOW we come to one of the most intriguing 3D Commands of all – **COONS**. I used to theorise that this meant something like Coordinated Network Surface, but when I asked Graphisoft, it turned out to be named after a mathematician called **Robert Coons** who is admired by the programming team!

By defining the **XYZs** of the **corners** and **edgepoints** of a **rectangular object**, GDL will try to work out all the in-between locations; the whole surface will be rendered as a myriad of small triangles. It can therefore adopt any shape.

The overall object need not be an exact Rectangle – it just requires 4 corners (so that it is ‘quasi-rectangular’), and almost anything can happen in between. GDL interpolates and smooths all the points over the surface.

More complex surfaces can be made up with tiles of COONS elements. Sometimes the edge points are determined by mathematical procedure, not by guesswork, so for this you **MUST** use the PUT statement.

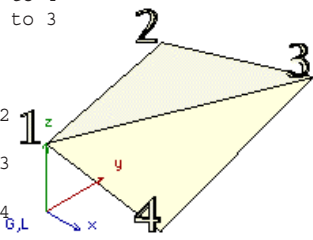
Use this method!

I have evolved a fairly unburstable routine for doing COONS successfully, and very rarely get an error using this method. Try this one out.

!COONS Demonstration

```
!Syntax:- COONS n,m,mask,
! all XYZs from Point 1 to 2
! all XYZs from Point 4 to 3
! all XYZs from Point 1 to 4
! all XYZs from Point 2 to 3
```

```
COONS 2,2,63+64,
0.0, 0.0, 1.0, ! 1 to 2
0.0, 2.0, 1.5,
2.0, 0.0, 0.7, ! 4 to 3
2.2, 2.4, 1.9,
0.0, 0.0, 1.0, ! 1 to 4
2.0, 0.0, 0.7,
0.0, 2.0, 1.5, ! 2 to 3
2.2, 2.4, 1.9
```



63 is the best mask, 127 reveals the triangles, and can be written as 63+64

Masking: 4=1st boundary edge, 8=2nd boundary edge, 16=3rd boundary edge, 32=4th boundary edge, 64=Top edges & surface rough. Even in AC8, none of these maskings seem to work correctly except 64. It seems impossible to control 3D line representation. If you make a complex surface from several COONS tiles, the viewer can see how you have tiled it.

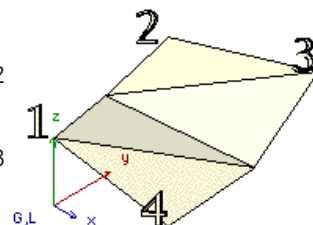
Start with the corners only:

If you follow the rigorous discipline of Points 1 to 2, Points 4 to 3, Points 1 to 4, Points 2 to 3, then you will be able to use COONS successfully.

Once it works, then insert the interpolated points along the edge. Don't forget to increase the stated number of points on the first line of the statement.

!COONS Demonstration

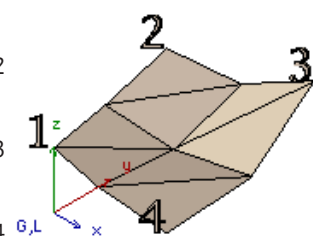
```
COONS 3,2,63+64,
0.0, 0.0, 1.0, ! 1 to 2
-0.1, 1.0, 1.1,
0.0, 2.0, 1.5,
2.0, 0.0, 0.7, ! 4 to 3
2.3, 1.2, 1.1,
2.2, 2.4, 1.9,
0.0, 0.0, 1.0, ! 1 to 4
2.0, 0.0, 0.7,
0.0, 2.0, 1.5, ! 2 to 3
2.2, 2.4, 1.9
```



The example here happens to have an XY location of 0,0. You don't have to move the cursor to the start of the command.

!COONS Demonstration

```
COONS 3,3,63+64,
0.0, 0.0, 1.0, ! 1 to 2
-0.1, 1.0, 1.1,
0.0, 2.0, 1.5,
2.0, 0.0, 0.7, ! 4 to 3
2.3, 1.2, 1.1,
2.2, 2.4, 1.9,
0.0, 0.0, 1.0, ! 1 to 4
1.0, -0.2, 1.0,
2.0, 0.0, 0.7,
0.0, 2.0, 1.5, ! 2 to 3
1.1, 2.2, 1.4,
2.2, 2.4, 1.9
```



Inserting additional pairs of XYZ points in the COONS makes the mesh more shapely – the more you put in, the smoother the result.

Hot Tip!

Do it with PUT & GET

COONS statements can get very long and increasingly complex. The same applies to **MASS**. Once this happens the chance of errors increases.

By using a series of **PUT** statements to ‘build’ the coons or mass, and by using an **NSP** statement to define the number of points on the surface, you will have greater freedom to interpolate edge points.

I also use PUT and GET for most Prisms and Tubes – it reduces syntax errors.

OK! so **COONS** looks terrifying. But if you take the gradual systematic approach demonstrated here, you can get COONS to be useful for you. It is the only way to do smooth interpolated surfaces in ArchiCAD.

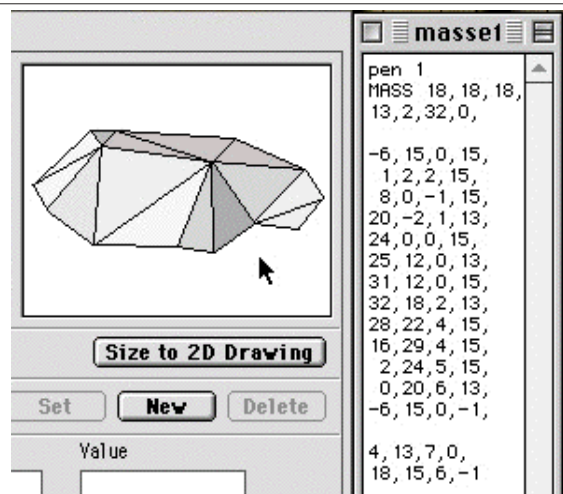
MASS

MASS is good. **MASS** is very good. It is the GDL equivalent of the **Mesh** tool, and forms a surface from a number of X-Y-Z node locations by resolving the resulting surface into triangles. It is a great help for surface and site making. Triangulation is calculated automatically by GDL. Like the Meshtool, it can only make surfaces that can be seen from above, i.e. has no tucks or curls in it.

A neat trick is to use the mesh tool direct in AC to get the basic shape worked out, then you can edit the details of the resulting **MASS** script in GDL. With some thought, you could generate new shapes with **MASS** such as trigonometrical or random surfaces or points with graphical hotspots, an interesting adventure to try some time.

After the opening lines of materials and masking, you first define the XYZ nodes in the boundary. Then, if you have any points within the surface, you define them as a series of small 'ridges' or single points, each ending in a -1 masking code. As the ridges are added, the whole surface responds with yet more triangles to ensure that the whole surface will shade. By playing with the object mask, you can also show or omit the side wall (skirt) or the base, and invoke object smoothing.

```
MASS topmat, botmat, sidemat,  
numbextpoints, numbintpoints, mask, skirtheight,  
xel,yel,zel,...xen,yen,zen,sen, list of external points,  
xil,yil,zil,sil,...xin,yin,zin,sin, list of internal points
```



The XYZ's of **Holes** are considered to be 'external points', but they need a minus 1 after the external shape, as in **PRISM_holes**.

The end of the definition of the perimeter is marked by a masking code of -1, and the later XYZ definitions are ridges or points – also each ending in a -1. The overall object masking code (here, 32) can decide if the sides or bottom are to be displayed, or if the object is to appear smooth in renderings.

Object Masking: 1=Show base, 4=Show skirt, 16=Show edge lines for base and skirt, 32=Edge lines for top are visible, 64=Top edges visible, top surface is not smooth.

The difference between COONS and MASS in a Nutshell

COONS makes a skin, and requires you only to describe the edges of a surface and it tries to interpolate all the other points – sometimes producing excellent results – but sometimes it smooths the surface in a way that you do not like. If the surface is too complex in shape to tolerate smooth interpolation because it has more complex detail (pointy bits), **MASS** will allow you to be more prescriptive about the exact height of points, in addition to making skins or solid objects – but then you will not get the smooth interpolation. Both are difficult to write.

MASS has the advantage that you can build it in ArchiCAD's normal project window with the **MESH** tool, then copy it to GDL and edit to its final form. **COONS** can only be built the hard way. **MASS** permits holes to be drilled, or plateaux to be formed with ease.

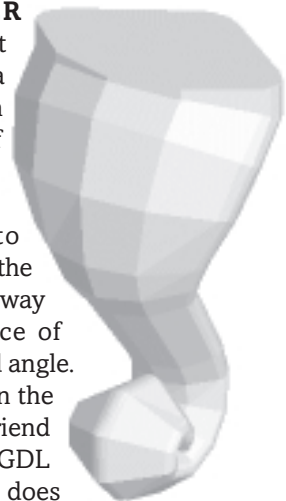
MESH: How does **MESH** compare with these? Don't even think about it! Very few things in the 3D world are perfectly rectangular, but worse, you have to calculate every blessed point height in the **MESH**.

LOFTER

No, sorry, LOFTER does not exist. But it ought to. If you had a **LOFTER** command which was a combination of **RULED**, **TUBE** and **SWEEP**, how happy we would be.

The syntax would be to state the number of points in the section, the number of pathway points, and then a sequence of Skin polygon definitions, and angle. The function would then skin the results to form a shape. My friend Oleg Shmidt has written a GDL Macro called **LOFTER** that does just this, although it is complex to use.

Let's hope Graphisoft adopt something like **LOFTER** in a future edition.



More 3D GDL Elements

EVERY element in ArchiCAD's toolbox has its 2D or 3D equivalent in GDL. You can find out which by laying out elements in the ArchiCAD Floor Plan and dragging them to a GDL Script window.

- Walls become XWALL_{2}
- Roofs become CSLAB_
- Floors become CPRISM_
- Beams become BEAM
- Columns become CPRISM_
- Mesh becomes MASS
- Fills and Polylines become POLY2_B{2}
- Line becomes LINE2
- Spline lines become SPLINE2A
- Texts become TEXT2
- Circles become CIRCLE2
- Hotspots become HOTSPOT2
- Objects, Lamps and stairs dragged into GDL become CALLs. Some of these are already covered.

What of the WALL and related 3D GDL?

My view is that these are near impractical to write in creative GDL scripting, even though it might be worth the work of tweaking them if they appear as part of editable script. If you need to make anything with XWall, for example, I would draw something as near to what I wanted first in ArchiCAD – including roof and window cuts – then tweak the result.

Most parametric objects, however complex, can more easily be done with **workarounds** using prisms, cutpolys and SGC. Boolean operations (Solid Geometry Commands SGC) make more complex 3D possible. For this reason, the GDL CB does not go into detail on the walls, nor on the CROOF. If you have read this far in the GDL CB, you will have enough courage to read the sections on the manual with a more analytical approach.

Strangely, GS spend pages detailing the use of additional GDL commands that are only likely to be used in autoscripting – VECT, PIPG, NODE for example. As The manual has the tendency to fall open at these pages (like buttered toast landing on the carpet), it's no wonder that some newcomers to GDL snap the book shut again!

CWALL_ and bWALL_ and XWALL_

These are commands to generate wall shapes with cuts and holes. BWALL adds parametric curvature to the CWALL. XWALL and its extended version XWALL_{2} allows greater complexity altogether – including parametric Log walling capability, varied planform, chamfered sills etc.

VERT, COOR, BASE, BODY

These are very important commands in Texture mapping and deserve their own section in the GDL CB.

EDGE, PGON, PIPG & TEVE

These commands are vital to imported Autoscripted objects, but of no use to creative GDL – they are not even friendly to those who like to tweak. If you import DXF files, you get the code condensed into pages of scripted garbage using these commands. So while it's useful to have a way of interpreting DXF imports, I very much doubt if you will use them for making new objects.

ARMC, ARME

These derive from the earliest days of GDL – which started out as a piping design tool for early designs of nuclear powerstations.

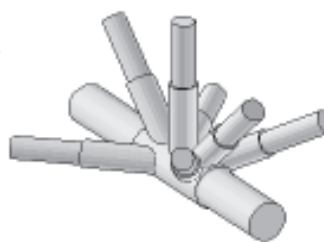
ARMC allows you to make a cylinder with a concave curved end that can fit cleanly against another pipe of the same or larger radius. The cylinder can be in line with the bigger pipe axis or off centre. It is ideal for simulating sleeve joints in a lattice structure.

ARME is designed to fit cylinders to an ellipsoid. You will be unlucky if you ever need to do this in a lifetime of GDL writing! A circular chimney on a dome?

I have never used either of these in practical GDL work, perhaps out of laziness – models of tubular space structures etc. looked adequate even when pipes passed through each other because they look OK in photo renders.

If your GDL model requires attention to closeup detail and line quality, the pro-minded GDL author needs to make the effort to use these. The major benefit seems to be that if you get intersecting pipes, you will get cleaner end cutting, leading to clean wireline views. With Boolean operations, they may be obsolete.

It would be much easier for the GDL writer if ARMC was more like CONE or ELBOW, in that the primary axis runs vertically, and the l,h,alpha parameters determine how the top end is to be cut.




```

!ARMC Tester
!3D Script
MATERIAL 18
PEN 1
TOLER 0.001

ROTY -90
!Main Pipe
CYLIND 1.4, 0.09

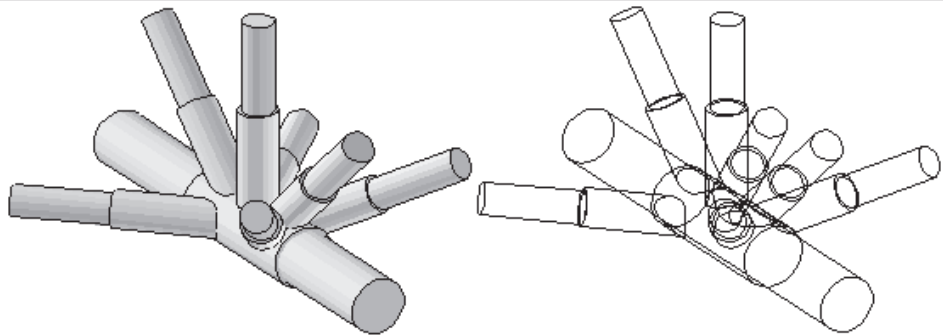
!Main sleeve
ADDz 0.4
CYLIND 0.5, 0.1

!First pipetree
ADDz 0.3
alph=45
FOR ang=-90 TO 90 STEP 90
  ROTz ang
  GOSUB 10
  DEL 1
  NEXT ang

!Centre Pipe
ADDz -0.05
alph=90
GOSUB 10

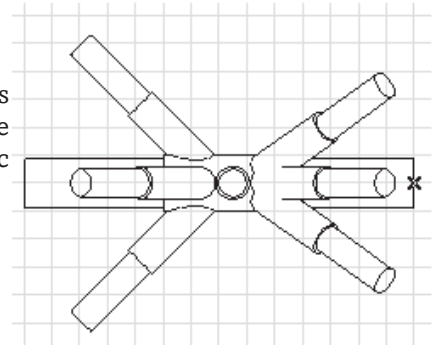
!Second pipetree
ADDz -0.05
alph=135
FOR ang=-45 TO 45 STEP 45
  ROTz ang
  GOSUB 10
  DEL 1
  NEXT ang
DEL 5

```



ARMC Syntax:

ARMC largepipe radius, armc's radius, length, offset from centre of large pipe, alpha angle of armc to large pipe around Y.



```

END: !=====

10: !Sleeve and pipe
ARMC 0.1, 0.06, 0.4, 0, 0, alph
  ROTy alph
  ADDz 0.4
  CYLIND 0.3, 0.05
  DEL 2
RETURN

```

```

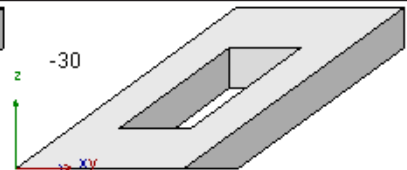
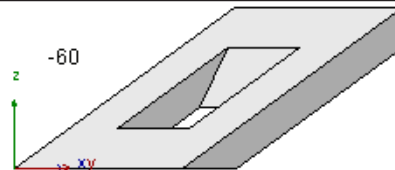
!CROOF tester
!3D Script
PEN 1

CROOF_ 18, 18, 18, ! materials
5+5,
0, 0, !Start X,Y
1, 0, !End X,Y
0.0, !Height
-30, !Angle
0.5, !Thickness

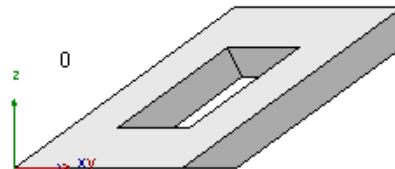
!XY List Outline
0.0, 0.0, -60, 15,
3.5, 0.0, 0, 15,
3.5, 4.0, -30, 15,
0.0, 4.0, 0, 15,
0.0, 0.0, 0, -1,

!XY List Window
1.0, 1.0, -60, 15,
2.5, 1.0, 0, 15,
2.5, 3.0, -30, 15, !<----
1.0, 3.0, 0, 15,
1.0, 1.0, 0, -1

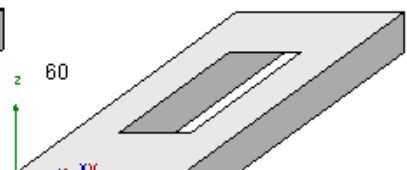
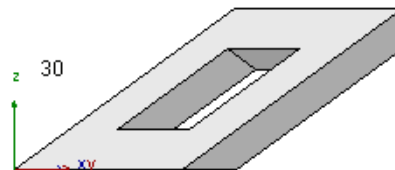
```



CROOF: Window reveal angles



We are changing the cut angle of the upper surface of the rooflight hole.



CROOF_ Syntax:

CROOF_ Material, Material, Material,
 Number of points,
 start X, start Y, end X, end Y,
 Height above reference plane,
 Angle of pitch,
 Thickness,
 X,Y, cutangle, masking code,
 X,Y, cutangle, masking code,
 X,Y, cutangle, masking code,
 X,Y, cutangle, masking code,
 etc, etc, etc

cROOF

CROOF is less difficult than the walls for someone really determined to get custom shapes that CSLAB cannot form. CSLAB limits you to vertical cuts on upper and lower faces – and you have to know the XYZ of every point on the outline. If the resulting surface is non-planar then it will disappear!

CROOF allows you to define the hingeing line (as you do when making a monopitch roof in ArchiCAD), then the Height, Pitch and Thickness. The list of XY locations (looking at it vertically) no longer asks for Z heights (as with CSLAB), it asks for edge cutting angles and a visibility code. Very powerful.

Lattice Truss



This model is a useful exercise in:

- Using Trigonometry to determine lengths and angles
- FOR... NEXT loops
- 2D Script writing
- Using 2D image to display information
- Defining text styles

THIS is kind of structure could be made by placing, arranging and grouping the cylinders from the ArchiCAD Library, but it would not be parametric or smart. We need GDL for a beam that is adaptable to different spans and depths.

Parameters

The model is not stretchy, as the lattice is built from manufacturer's standard sized modules.

- The Truss is to be done here using only CYLIND to form the tubes.
- The user is offered a choice of width, height and any number of modules, and tubing diameter. This example does not worry about joints, but it would be easy to add balls or cones, which could be a user option.
- The user can ask the truss to 'Show Data' – this could be a print out of its span or weight.
- As the truss could in fact be a vertical structure (column) or could be leaning (over an atrium) the lean angle option is offered.

3D Script

Even for a small object, a strictly structured approach is adopted from the start. Resolve the model into a series of subroutines, one to draw every element of the model. The executive script is short in the extreme, just calling subroutines.

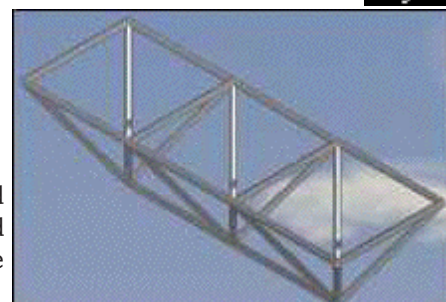
The technique is to make all the upper elements, in a FOR...NEXT loop, then the bottom tube, and finally, the diagonal tubes. Some trigonometry is required to work out the angles of the diagonal tubes. As this is an important part of the script, a special subroutine 100: is proposed for working out the trigonometry and setting up other parameters.

100: First set up the usual things like RESOL, Material and Pen. Then convert diameter to radius.

Now work out the angle of the diagonal tubes. As the truss module as seen in plan is a perfect square, the diagonal across the square is the 'truss module times $\sqrt{2}$ '; the angle of lean of the cylinder will be the ArcTan of half that diagonal and the truss depth.

Finally, the Length of the diagonal tube can be found as the hypotenuse of the right angle triangle formed by the tube – the depth divided by the COS of the angle. Alternatively, you could work out the tube length by Pythagoras. In later exercises (winged truss) you will see that it is possible almost to dispense with trigonometry and use the TUBE command to go from XYZ point to XYZ point.

Variable	Type	Name	Value
tmod		Truss Module plan size	1200
tdep		Depth of Truss	1000
tdiam		Diameter of tubing m	60
tnum		Number of Truss Modules	3
shodata		Show Data Yes=1 No=0 On	
fheight		Font height mm	2
leanang		Leaning Angle (0=horiz)	0.00
tmatl		Tubing material	11
tpen		Pen Colour	1



```
!LATTICE TRUSS
!3D Script

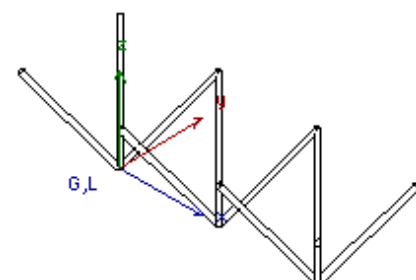
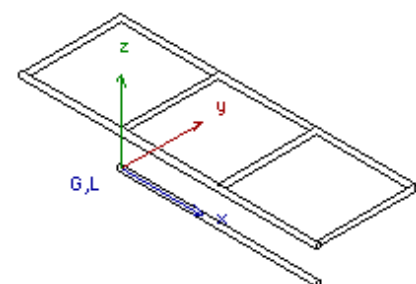
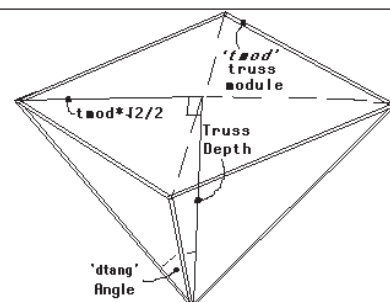
!---- THE PROGRAM -----
GOSUB 100 !check parameters

    ROTy -Leanang
GOSUB 200 !Upper elements
GOSUB 300 !Bottom tube
GOSUB 400 !Diagonal tubes
    DEL 1 !undo Lean angle

END !-----
```

```
100: !parameter set up
!or put into Master Script
```

```
RESOL 8
MATERIAL tmatl
PEN tpen
trad =tdiam/2 !tube radius
dtang =ATN((tmod*sqr(2)/2)/tdep)
dtlen =tdep/cos(dtang)
RETURN
```



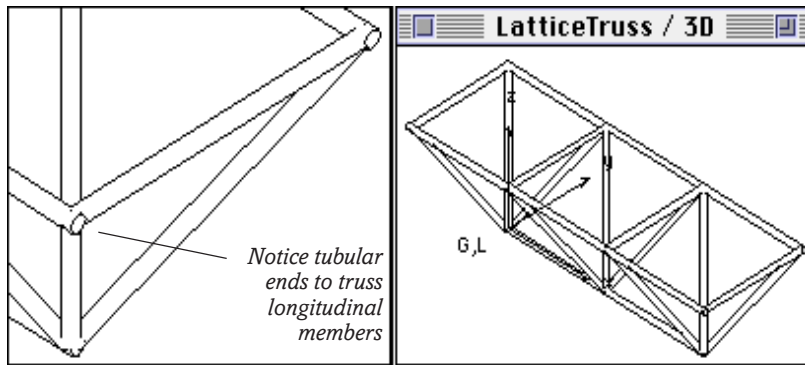
200: The Upper longitudinal tubes are easy enough : a matter of getting to the right position, and pumping out a CYLIND. However, note that one steps backwards by one tube radius, and makes the tube a tiny bit longer to give the impression of a butt end to the tube. The same is done for the bottom member.

The FOR...NEXT Loop is needed to generate the lateral tubes. By using a loop, you can use the number of modules as a counter to decide how many to draw. Notice in this example, that we use K as a counter in the loop and also as a way of calculating how far the cursor travels before generating the cylinder. For the first tube, it moves 'tmod*(k-1)' which is, effectively, zero. When it gets to the 4th tube, it is moving 'tmod*(4-1)' which gets it to the right place. Using this method, you can do the DEL command INSIDE the FOR... NEXT loop, which is much tidier.

300: Bottom member of the truss is very easy. One long Cylinder, slightly elongated to improve the junction detail.

400: The Diagonals – well, the routine to draw the 4 diagonal tubes is a bit complicated, so it uses subroutine 450 for each truss module, so that the main FOR...NEXT loop looks tidier.

450: Rotate 45° then issue 4 leaning-over cylinders rotating 90° each time. At the end, put DEL 4+1 instead of 5. It reminds you not to forget the ROTz 45 with which you started drawing the tubes.



2D Script

It is important to have a good 2D Script, as the Project2 command will take too long to display, with all those cylinders.

If you wish to tilt the truss so that it's a leanto or a Column, the 2D Script would not work, so you need to use Project2.

The 2D Script here takes the same parameters as the 3D Script, works out a few locations of the nodes, and draws quick lines and rectangles to draw the truss in plan. It also plants Hotspots in the right places.

Finally, you might like to try getting the 2D symbol to give you information. In this case, the span of the truss. For this, you need to define a text style – the size of the text is parametric. This 2D display ability could be used to print out structural or weight data. It is also useful in circular objects in displaying Radius or span.

The truss can also be used as a Walling element. A way to make the truss draw very quickly in the project plan is to 'encapsulate' your Truss: make your parametrically defined truss a library object AGAIN, so that it has a standard object symbol – providing you have no further intention of changing it. You can use this method to make an entire PORTAL.

```
200: !Truss upper element
!Longitudinal
ADD -tmod/2-trad,-tmod/2,tdep
ROTy 90
CYLIND tmod*tnum+tdiam,trad
ADDy tmod
CYLIND tmod*tnum+tdiam,trad
DEL 3

!Truss upper tube - Lateral
FOR k = 1 TO tnum+1
ADD -tmod/2,-tmod/2,tdep
ROTx -90
ADDx tmod*(k-1)
CYLIND tmod,trad
DEL 3
NEXT k
RETURN

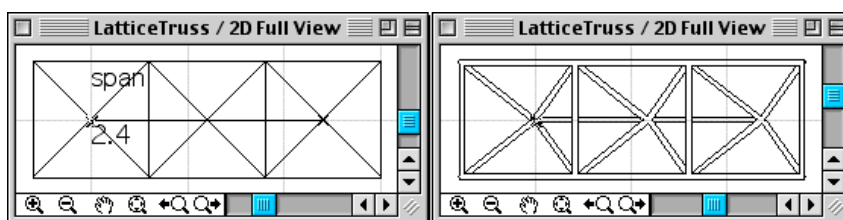
300: !Bottom boom
ROTy 90
ADDz -trad !slightly longer
CYLIND tmod*(tnum-1)+tdiam,trad
DEL 2
RETURN

400: !Diagonal Truss element
FOR k = 1 TO tnum
ADDx tmod*(k-1)
GOSUB 450
DEL 1
NEXT k
RETURN

450: !Diagonals
ROTz 45: !first tube angle
FOR n=1 to 4
ROTx dtang
CYLIND dtlen,trad
DEL 1
ROTz 90
NEXT n
DEL 4+1
RETURN
```

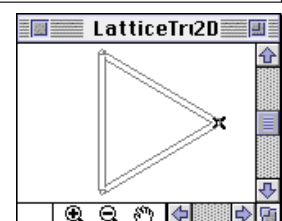
!2D Script

```
IF Leanang THEN
PROJECT2 3,270,2
ELSE !if Truss is Flat
LET w2 = tmod/2
!Top and bottom lengths
tenx= tmod*tnum-tmod/2
benx= tmod*(tnum-1)
!-----
RECT2 -w2,w2, tenx,-w2
FOR k=1 to tnum
ADD2 tmod*(k-1),0
LINE2 -w2,-w2,w2,w2
LINE2 -w2,w2,w2,-w2
LINE2 w2,w2,w2,-w2
DEL 1
NEXT k
LINE2 0,0, benx,0
!-----
HOTSPOT2 0,0
HOTSPOT2 benx,0
!-----
IF shodata THEN
DEFINE STYLE "display"
"geneva", fhigt,1,0
STYLE "display"
TEXT2 0,w2,"span"
TEXT2 0,0,tmod*(tnum-1)
ENDIF
```



Scripted 2D symbol if the truss is flat

Use PROJECT2 if the truss is tilted



Truss as a wall element

GDL Macros

MANY OF GRAPHISOFT's Library objects – windows, kitchenware etc – make use of macros. A macro is a script or object that is used so often that it is stored as a special component.

An example is a window sash, that could be called from the Library to join a whole variety of window designs. It saves having to write the code for the sash again and again. In fact, the 'Window Sash' could itself be calling macros for each 'Window Section' that makes up the sash. The same applies to a kitchen tap – which can be called to a variety of sink unit configurations. A door handle might be applicable to a whole range of doors, so you would not want to have to paste the same door handle subroutine into every one of the doors. I think of the macro being a child and the major object being the parent – calling the child, and passing on its characteristics to the child ('inheritance').

Those who use subroutines a lot are more than half way aware of the power of macros. With subroutines, you wrap up a chunk of code that you intend to call frequently. If well written, the subroutine is nicely logi-

cal, neatly encapsulating a sub-object. Macros are similar in concept only the code for the sub-object is contained in a quite separate .GSM file. As these macros can have parameters of their own, they can be much more powerful and richly complex than a subroutine. And the primary benefit is that they can be used by several major objects, whereas a subroutine would have to be copied and pasted into the major objects. If you want to do some upgrading on the sub-object, you would have a nightmare doing it to all those subroutines, whereas with a macro, you improve the code in the macro, save, look again at the major object, and Hey! the whole thing is immediately improved.

Macros are activated with the CALL command. They may bring their own parameters (that they were first saved with) or may be told to adopt new values in the current script.

CALLing the Macro from another script/object

Save the left hand routine as an object in its own right with only *hit* and *w* as parameters. Now your current script can get it from the library with a CALL command.

```
!Squilinder macro
PRISM 4,hit,
  w/2,w/2,
  -w/2,w/2,
  -w/2,-w/2,
  w/2,-w/2
```

```
!The major object
MATERIAL 18
CALL squilinder PARAMETERS
  w=12",hit=2"-10"
```

Rules and Tricks for Macros:

- Macros are better as placeable Objects (.GSM files), not as pure scripts, or as Macro Sybtypes – thus they can have their own small parameter names, preview image, 3D and 2D Script, which makes it much easier to write and review them because you can see them in 3D as you make them.
- It is better that many objects (such as a lot of sink units) should call on a few macros (such as taps); this is more efficient than a few objects calling on many macros (such as a lattice repeatedly calling macros for the members).
- If you are certain that you will not need to use the Macro as an independent object, then you could make the object "unplaceable". You can omit PEN and MATERIAL statements. If the macro lacks these, it will simply inherit them from the parent object that calls it. The only reason you might specify materials in the macro is if the macro contains more than one material and cannot simply inherit the parent's materials.
- Write Macros with simpler parameter names, for example, it's not unreasonable to use single letter variables for macros, as in the example here. Another trick is to use the same parameter names as in the parent object so that the CALL command will contain simple Parameter statements like `armlen=armlen, arm_mat=arm_mat`.
- Store macros in a folder in your library that clearly labels them as macros – because they are all .GSM now and you will soon forget which objects are macros.
- Never use **DEL TOP** in a macro. This is so powerful it will delete the stack in the parent object too.
- If you try to use the **GDL Debugger** in the parent object, it will regard the CALL command as a single command – i.e. it will NOT continue execution in the macro (unless there is a BREAKPOINT statement in the macro).

Know when to use Macros

If you are serious about GDL for manufacturing, e.g. your objects are part of a commercial library using standard timber sections and metal fixings, there is no argument about it: Macros are the way to go! the ONLY way to go.

If you are working as a team with others, you can share the same library of macros. Can you imagine how complex it would be to copy and paste chunks of subroutine code between GDL objects on different workstations, at different times over several weeks of a development programme.

By clever organisation and macro-naming tactics, you can make a parent object like a single window have a set of casement configuration options, but have a choice of timber, composite metal-timber and all metal window sections, all of whose code would be safely stored in macros, and not clutter up the main window object.

Know when to avoid Macros

If you are making unified objects, such as ones you wish to place on Objects on Line, or sell on the web, and which are one-off in conception, then macros are a nuisance, because the user will lose the macros. They will get error messages and non-appearing parts of their parent object. In this case, you should use Subroutines.

A similar argument applies to small bitmap images in the UI. If you have a commercial library, you can store images along with the macros, but if you want have a singular object, write a UI that avoids the use of bitmap images.

Twisted Cable

THIS exercise investigates a useful feature of **TUBE** command. The last digit in the Pathway definition of a TUBE command is a Twist angle. (The best stories always have a twist at the end.) If this angle progressively increases, then the section twists as it moves through space. The first exercise shows simply how this is done. The angle is always absolute, not relative. 10, 10, 10, will go straight, 10, 20, 30 will twist.







The 900 and 4000 define a circular section; if these were 901 and 4001, the twisty lines would disappear.

If you have smoothness ON in photorendering this form gets smoothed and the coil is useless – with smoothness OFF, you can clearly see the twists.

Because it's very short and could be used as a macro, it's easier to use single letter variables. As a macro, it could be used almost as easily as CYLIND.

Material and Pen are omitted because they can be applied in the object settings box. If the object is to be used as a macro, it would inherit material and pen from the calling object.

With graphical hotspots, you could make this 'wind' itself realistically.

Variable	Type	Name	Value
A		X Dimension	1.000
B		Y Dimension	1.000
l		Length of Cable	5.000
d		Diameter of Cable	0.800
n		Number of visible strands 3-12	8
t		Twistiness Index 1-10	4

Dynamic Line numbers!

Normally you GOSUB 100 and that's it. But if you are making use of flags, you can use this technique. Where you might normally write:

```
IF hs=1 THEN GOSUB 101
IF hs=2 THEN GOSUB 102
IF hs=3 THEN GOSUB 103
```

You can write all of these on a much shorter line:

```
GOSUB 100+hs
```

The trick of course only works if you organise your subroutines so that the numbering relates to the flag numbers. If you (like me) prefer to follow good practice and separate your line numbers in chunks of ten or more, then replace:

```
IF hs=1 THEN GOSUB 110
IF hs=2 THEN GOSUB 120
IF hs=3 THEN GOSUB 130
```

with:

```
GOSUB 100+hs*10
```

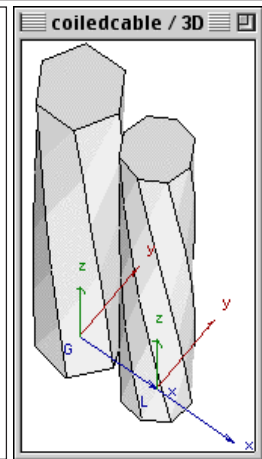
Name your subroutine!

Another idea for subroutines is to find a way to name them directly. (The subroutines still have to have numbered labels, but the name can be part of the Label Comment.)

!Master Script	!3DScript
build_cabinet=100	GOSUB build_cabinet
build_drawers=200	GOSUB build_drawers
dimensioning=500	GOSUB dimensioning

```
!Coiled cable
!Non parametric
RESOL 6
rad=0.5 !Radius
PUT 0,0,-1,0,twist
0,0,0,10,
0,0,1,20,
0,0,2,30,
0,0,3,40,
0,0,4,50,
0,0,5,60,
0,0,6,70

TUBE 2,NSP/4,63,
0,0,900,
rad,360,4000,
GET(NSP)
```



```
!'Coilcabl' 3D Script
!Parametric
!Error checking
IF n<3 THEN n=3
IF n>12 THEN n=12
IF t<1 THEN t=1
IF t>10 THEN t=10
RESOL n
FOR z=-d TO l+d STEP d
tw=tw+t^2
PUT 0,0,z,tw
NEXT z
BODY -1
TUBE 2,NSP/4,63,
0,0,900,
d/2,360,4000,
GET(NSP)
BODY -1
```

As a tiny object, the error checking can be done in the 3D script

The twist angle is progressively increased as the loop progresses.

As a macro, it would not need a 2D script.

```
MATERIAL 'Zinc'
PEN L_
CALL coilcabl PARAMETERS l=5,
d=0.4,n=8,t=6
```

This is how it can be used as a macro

This is a Parametric way of doing the cable so this could be a macro, used like CYLIND, up the Z-axis. BODY -1 makes it more reliable in 3D.

```
!Ultra simple 2D Script
CIRCLE2 0,0,d/2
```

Lattices and Tube Structures

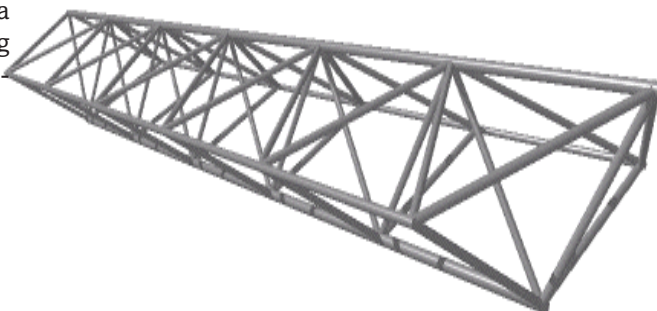


Lattices – Winged Truss – 1

THIS is an interesting truss form, which I call a 'winged truss'. It is here to show a way of building lattices that can be done by one subroutine call and requires no trigonometry.

This is a valuable learning exercise, including:

- Smart stretchy algorithm (recalculates spacing)
- Uses no trigonometry (done by XYZ locations)
- The 2D script is written in the 3D window (?)
- The members are made with TUBE command :-)



This method is incredibly easy for making lattices, because one subroutine does all the work, you only have to tell it the XYZ locations of each end of a tube and the job is done. Even better, you get a totally accurate 2D image, scripted from within the 3D script!! All the 2D script has to supply are hotspots and labels. In the Cookbook, I have referred to this as the 'AngleRod' method, using a simple TUBE from point to point.

Master Script

GLOB_SCRIPT_TYPE is the key to automatic 2D scripting. It tells GDL when it's busy drawing a 2D symbol; if it is then the flag *draw2d* is set to 1. Subroutine 100 does all the work in 3D. Every tube in the 3D can have a matching LINE in 2D also which is performed by subroutine 101. We can then do a project2 of the truss made with either tubes or lines.

3D Script

Establish the subroutine 100 first, because this is what does all the work. Do one test member, which can be the lower tube.

Because this truss is, in effect, a load of cylinders that all collide at node points, you are depending on the ability of shaded views to 'z-buffer' all the cylinders where they meet – i.e. to tidy up the junction. Thus the lower tube is improved in appearance by being extended by its own radius in each direction. The parameters for x_1, y_1, z_1 and x_2, y_2, z_2 and the tube radius are sent to the subroutines. If *draw2d* is positive, the action will be sent to subroutine 101 where a LIN command will draw a line instead of a cylinder.

```
!Winged 3D truss
!Master Script

tnum=INT(1+A/tlen) !Number of Trusses
IF tnum<1 THEN tnum=1
twid=B !Width of Truss
thit=zzyzx !Height of Truss

!Resolution limits
IF resl<4 THEN resl=4
IF resl>12 THEN resl=12

!Drawing in 2D or 3D?
IF GLOB_SCRIPT_TYPE=2 THEN draw2d=1
IF truvu=1 THEN draw2d=0
```

For would-be lattice builders, this is the most useful exercise in the GDL Cookbook.

A		X Dimension	9.000
B		Y Dimension	3.600
zzyzx		Height of Truss	1.200
Title		Configuration	
tlen		Length of one Truss module	2.100
t1diam		Diameter Top+Bottom main tube	0.100
t2diam		Diameter Lateral long tubing	0.070
t3diam		Diameter Upper wing tubing	0.070
t4diam		Diameter Lower wing tubing	0.070
t5diam		Diameter Diagonal bracings	0.080
tmat		Material of Tubing	11
pcol3		Pen colour 3D	1
Title		Options	
iden	Abc	Truss Identification	MainHall
truvu	<input checked="" type="checkbox"/>	Truvu=ON Scripted View=OFF	Off
shownam	<input checked="" type="checkbox"/>	Show Name in 2D symbol ON/O...On	
resl		Resolution of Curvature	6

3D script: continued

The tube structure continues with the upper and the lateral tubes. Then the diagonal tubes are done as a series of FOR NEXT loops. They could all be done as one single FOR NEXT, but for the Cookbook I have done each set of tubes separately so you see how the thing builds up. All you have to do is to work out the relative x1,y1,z1, and x2,y2,z2 locations for each tube and you have won!

```
!Winged 3D truss
!Rewritten for Cookbook3 Nov 2000
```

```
PEN pcol3
MATERIAL tmat
RESOL resl
```

```
!Lower main member
```

```
rad=tldiam/2
x1=-rad: y1=0: z1=0
x2=tlen*(tnum-1)+rad
y2=0: z2=0
GOSUB 100+draw2d
```

```
END: !-----
```

```
100: !Tube
```

```
!Define Phantom Points
```

```
x0=x1+(x1-x2)
x3=x2-(x1-x2)
y0=y1+(y1-y2)
y3=y2-(y1-y2)
z0=z1+(z1-z2)
z3=z2-(z1-z2)
```

This routine works out the position of phantom points no matter how wierd the start and end points. It ensure clean butt ends to all tubes.

```
!Draw it
```

```
TUBE 2,4,63,
0,0,901,
rad,360,4001,
x0,y0,z0,0,
x1,y1,z1,0,
x2,y2,z2,0,
x3,y3,z3,0
```

These two lines are the polylines required to draw a cylinder. The extra 1 gives you tidy cylinders. 900 and 4000 would give you a mass of ink lines for each polygon on the cylinder – horrid!

```
RETURN
```

```
101: !Tube in 2D
```

```
LIN_ x1,y1,z1, x2,y2,z2
RETURN
```

Subroutine 101 only happens if PROJECT2 is working

The 2D symbol with a simple level of detail. Each member is represented by one line

The 2D symbol with a high level of detail – a truvview.

The full PROJECT2 is drawn. A detail study in an architect's drawing might need such detail, but an engineer's layout for a whole roof of trusses would prefer the simpler representation and the faster redraw time.

```
!Winged 3D truss
```

```
!2D Script
```

```
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 tlen*(tnum-1),0
HOTSPOT2 tlen*(tnum)-tlen/2,0
HOTSPOT2 -tlen/2,0
```

```
!Put Hotspots on each truss element
```

```
FOR n=1 TO tnum
HOTSPOT2 tlen*(n-1),0
HOTSPOT2 tlen*(n-1), twid/2
HOTSPOT2 tlen*(n-1), -twid/2
NEXT n
```

```
!Crosshair on stretchy point
```

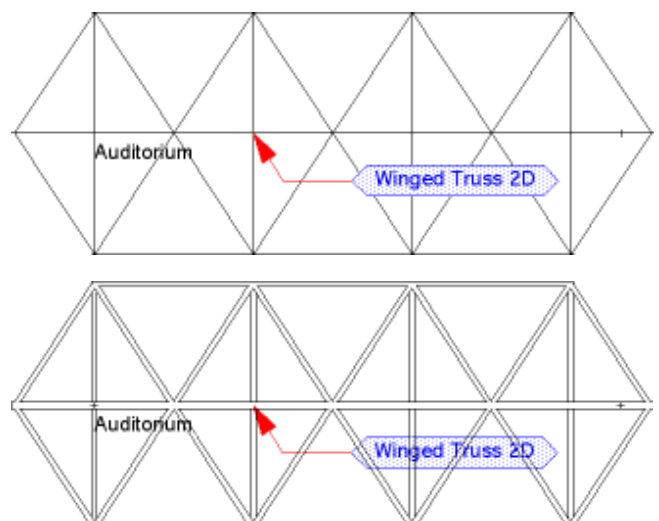
```
p=-tldiam/2
LINE2 -p,0,p,0
LINE2 0,-p,0,p
LINE2 A-p,0,A+p,0
LINE2 A,-p,A,p
```

This is where the real work happens. Normally you are advised to script in 2D instead of using PROJECT2. We have done! ...in the 3D script.

```
PROJECT2 3,270,2
```

```
IF shownam THEN !Annotation
fntz=2*tldiam*1000/A_ !Autosize
DEFINE STYLE 'ttxt' 'Arial',fntz,1,0
SET STYLE 'ttxt'
TEXT2 0,-tldiam,iden
ENDIF
```

these pages are
not yet
formatted



```

!Winged 3D truss
!ReWritten for Cookbook3 Nov 2000

PEN pcol3
MATERIAL tmat
RESOL res1

!Lower main member
rad=tldiam/2
x1=-rad: y1=0: z1=0
x2=tlen*(tnum-1)+rad
      y2=0: z2=0
GOSUB 100+draw2d

!Upper main member
rad=tldiam/2
x1=-tlen/2-rad: y1=0: z1=thit
x2=tlen*(tnum-1)+tlen/2+rad
      y2=0: z2=thit
GOSUB 100+draw2d

!Outer long members
rad=t2diam/2
x1=-rad: y1=-twid/2: z1=thit/2
x2=tlen*(tnum-1)+rad
      y2=-twid/2: z2=thit/2
GOSUB 100+draw2d
y1=-y1: y2=-y2
GOSUB 100+draw2d

!Upper diagonal tubes
FOR n=1 TO tnum
ADDx tlen*(n-1)
rad=t3diam/2
x1=-tlen/2: y1=0: z1=thit
x2=0: y2=-twid/2: z2=thit/2
GOSUB 100+draw2d
y2= twid/2
GOSUB 100+draw2d
x1=tlen/2
GOSUB 100+draw2d
y2=-twid/2
GOSUB 100+draw2d
DEL 1
NEXT n

!Lower diagonal tubes
FOR n=1 TO tnum
ADDx tlen*(n-1)
rad=t4diam/2
x1=0: y1=0: z1=0
x2=0: y2=-twid/2: z2=thit/2
GOSUB 100+draw2d
y2= twid/2
GOSUB 100+draw2d
DEL 1
NEXT n

!Longitudinal diagonal tubes
FOR n=1 TO tnum
ADDx tlen*(n-1)
rad=t5diam/2
x1=0: y1=0: z1=0
x2=-tlen/2: y2=0: z2=thit
GOSUB 100+draw2d
x2=tlen/2
GOSUB 100+draw2d
DEL 1
NEXT n

END:!-------

!Follow this with the
!subroutines 100 and 101

```

All you have to do is send the values of XYZ at each end, and radius, and your TUBE will be built!

2D Script

The Hotspots are important – this truss is stretchy in length (and in height in the 3D window). A loop is used to put one on each truss node. There could be an option to omit this if it is a nuisance with too many hotspots. The Crosshair assists in finding the stretchy 'A' hotspot.

Lattices – Winged Truss – 2



THIS is the 'winged truss' again, but done in a quite different way. This is the more traditional craftsmanlike way to make one, showing a good level of detail. The truss has cast metal sleeved joints to the tubes which we want to display. This requires the use of Trigonometry, to get all the angles of the tubes precisely calculated.

For manufacturing purposes, this would provide accurate feedback – as you stretch the truss, it can calculate and report back on the angles of the joints and the lengths of the tubes – this could be a written report for an engineer/fabricator. For general perspective drawing, or animation, one would like to shorten the drawing time – hence the truss can also be drawn with the joints concealed.

This is a valuable learning exercise, including:

- Using Trigonometry to calculate lattice members
- Outputting data to file to inform the user of details and risks
- Reading the system clock
- Extensive 2D scripting, to save redraw time
- Level of Detail awareness
- Defining and Autosizing text in 2D symbols

Parameters

If you have already built the previous Winged Truss, note that this has an identical parameter window, except for the addition of two parameters, *simcom* and *outdat*. So you could open the previous truss, erase the scripts and resave with a new name to save the effort of rewriting all the parameters.

The truss has four different radii of tubing. Every linear element is parametrically definable. A and B are used to make the truss stretchy. The Master Script converts the A and B into 'tnum' and 'twid', from which the rest of the truss can be drawn.

If you are starting from scratch, leave the optional parameters at the bottom of the list until you need them.

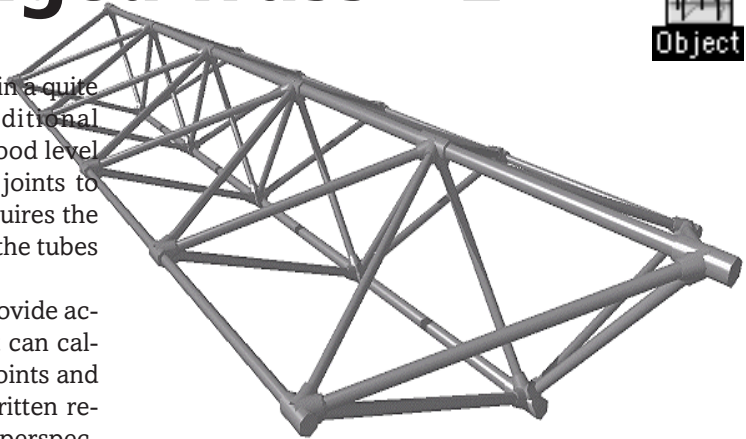
Master Script

If the object is stretchy, make sure that you equate A, B and *zzyzx* to the parameters names you wish to use. the *tnum* routine is a way of counteracting the tendency of INT(...) to round down. (One could use a new function CEIL(...) which is like INT but rounds up.)

The next set of calculations use ArcTan and Pythagoras to work out the angles and lengths of tubing.

3D Script

The housekeeping (like the angle and length of diagonal tubes) are all dealt with in the Master script. The 3D script can be dedicated to 3D. As this is a complex object, set out to structure the script, so put in the END



A		X Dimension	9.000
B		Y Dimension	3.600
zzyzx		Height of Truss	1.200
Title		Configuration	
tlen		Length of one Truss module	2.100
t1diam		Diameter Top+Bottom main tube	0.100
t2diam		Diameter Lateral long tubing	0.070
t3diam		Diameter Upper wing tubing	0.070
t4diam		Diameter Lower wing tubing	0.070
t5diam		Diameter Diagonal bracings	0.080
tmat		Material of Tubing	11
pcol3		Pen colour 3D	1
Title		Options	
iden		Truss Identification	MainHall
truvu		Truview=ON Scripted View=OFF	Off
simcom		Complex=ON Simple=OFF	On
outdat		OutPut DataFile ON/OFF?	On
shownam		Show Name in 2D symbol ON/O...	On
resl		Resolution of Curvature	6

```
!Winged 3D truss
!Master Script

tnum=INT(1+A/tlen) !Number of Trusses
twid=B              !Width of Truss
thit=zzyzx         !Height of Truss
IF tnum<1 THEN tnum=1

!Longitudinal Diagonal Tubing
dtang=ATN(thit/(tlen/2))
dlen =SQR(thit**2+(tlen/2)**2)

!Upper Wing tubing
ang1=ATN(tlen/twid)
ang2=ATN((thit/2)/((tlen/2)/SIN(ang1)))
utlen=(thit/2)/SIN(ang2)

!Lower Wing Tubing
ltang=ATN(thit/twid)
ltlen=SQR(thit*thit+twid*twid)/2

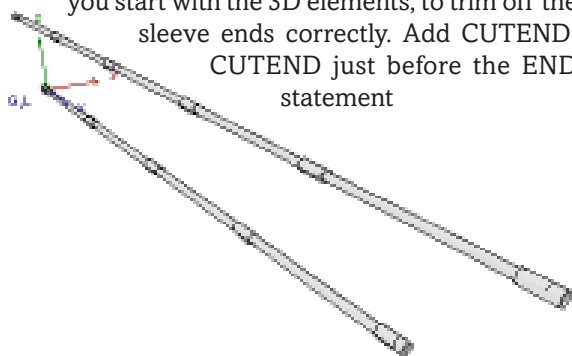
IF resl<4 THEN resl=4
IF resl>12 THEN resl=12
```

In some versions of ArchiCAD 6.5, the presence of trigonometry functions in the Master script causes Stretchy objects to have a spurious 'divide by zero' error – move the routine to the 3D and 2D scripts.

statement early.

The Lower and Upper and outer tube members (300: and 310:) are easy enough to do. Manoeuvre into position, rotate to point Z along the line of the truss and push out your cylinders. If the user has opted for the complex truss, then run the FOR NEXT loops that are sleeves for the joints. Because the sleeves are bigger than the tube, and have to accept diagonal tubes later, they are longer than their diameter. You need to make small adjustments in position to make the sit nicely.

The sleeves sit nicely along the rods. The ones in the middle are the right length because they have to accept a total of 8 tubes arriving into them at different angles. The end tubes overhang too much, so a CUTPLANE routine is required to reduce the size of the end joints. Insert the CUTPLANE routine before you start with the 3D elements, to trim off the sleeve ends correctly. Add CUTEND: CUTEND just before the END statement



These two lattices depend on cylinders meeting exactly at nodes. If you need to trim them back for a perfect wireline view, use the Trig and Loop method, and push circular CUTPOLYAs along the lines of the main tubes to cut the ends of the minor tubes (not illustrated in this example).

!Winged 3D truss: 3D Script

PEN pcol3

MATERIAL tmat

RESOL res1

CUTPLANE routine:

Insert here

GOSUB 300 !Lower Tube

GOSUB 310 !Upper Tube

END: !-----

300: !Lower Tube sleeves

ADDx -tldiam/2

ROTy 90

CYLIND tlen*(tnum-1)+tldiam,tldiam/2

IF simcom THEN

FOR k=1 TO tnum

ADDz tlen*(k-1)-tldiam

CYLIND tldiam*3, tldiam/1.5

DEL 1

NEXT k

ENDIF

DEL 2

RETURN

310: !Upper Tube sleeves

ADD -tlen/2-tldiam/2,0,thit

ROTy 90

CYLIND tlen*tnum+tldiam,tldiam/2

IF simcom THEN

ADDz -tldiam

FOR k=1 TO tnum+1

ADDz tlen*(k-1)

CYLIND tldiam*3,tldiam/1.5

DEL 1

NEXT k

DEL 1

ENDIF

DEL 2

RETURN

The basic Cylinder is the tube and the Loop creates the sleeves

!Cutplane routines for ends

ADDx -tlen/2-tldiam/2

ROTy -90

CUTPLANE

DEL 2

ADDx tlen*(tnum-1)+tlen/2+tldiam/2

ROTy 90

CUTPLANE

DEL 2

CUTPLANE routine: you need this to trim the sleeves

Tips and Tricks Building Lattices

WE CAN easily make non parametric lattices with the slab tool, laying out the shape, and then inserting holes, and finishing up with rectangular sectioned members. But if you stretch them, the sizes of all the members are distorted. Making a serious 3D truss this way is almost out of the question.

Home built Parametric Lattice trusses are one way to impress people who think that everything should be done with AutoCAD. Challenge them to match that with AutoLisp! Here are four methods to think about.

The Prism method: A lattice can be made as a PRISM_ with holes drilled. You can use arithmetic to work out how many holes to draw, PUT to store the outlines of the holes, and GET to draw the prism. You could also make the web of a castellated beam this way. This is somewhat limited to 2D trusses.

The Trigonometry and Loop method: Using the technique of the Lattice truss in the Discovery course, you can use arithmetic to work out how many lattice modules required (or to calculate the module size); then use a For Next Loop. The cursor runs along the length of the truss, building each module.

Along the way, you will have to use trigonometry to work out the angles of the truss members, and either Cylinder, or a Squilinder (prism) to draw the members. (I call this 'knitting'). This needs some serious maths if the truss is curved or irregular in shape. (Winged Truss-2)

CUTPOLY method: First build your lattice as a solid block; then apply a series of CUTPOLYs to drill holes, leaving behind the structural members. This is a laborious method for a 2D lattice, but for a 3D lattice, it could be the easiest! Use Loops and PUT and GET to place them.

The 'AngleRod' method: is useful if the truss departs from regular spacings or layout, and has members in which there are so many angles and geometries that the trigonometry gives you a headache. (I like trig, but you may not). In this case, you just use a macro like AngleRod in the Voyager course. You just specify the start and end points, the sectional diameter, and whether the rod is square or round. You do not actually need to call Anglerod, because the routine using TUBE is so short, it can be copied and pasted as a subroutine into your main script. (Winged Truss-1).

Lateral tubes

The lateral (or wing) tubes are done in the same way. Do one of them, and then use a mirroring technique, using MULy -1, to do the other. The routine in 320 is done in exactly the same way as for the main tubes.

Diagonals

The way to get the diagonals done is to start off by getting ONE module right, and the rest will be done with a FOR... NEXT loop.

These diagonals are more complicated. The main tubes were long single tubes, and the sleeves were cylinders. In the diagonals, the sleeves are conical, and have to point in the same direction as the tubes. This is a good illustration of an object oriented approach – one cannot solve the problem in bits and pieces. One must get to the right place, summon the tube and it should arrive as a whole object, complete with its cones.

Passing parameters

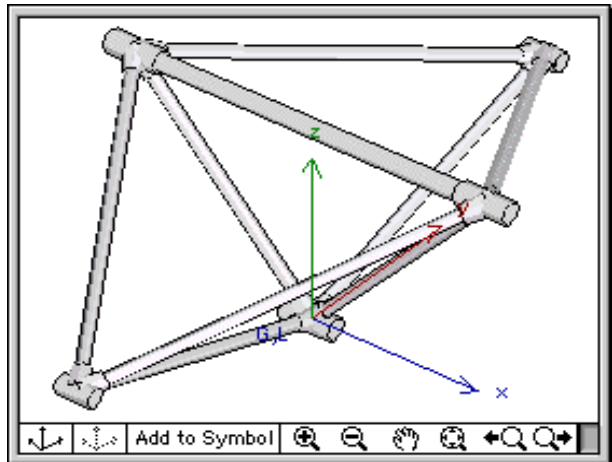
A standard subroutine (240:) has been developed for a Cylinder that has a cone at each end. (This gives the form of the sleeved joints). All you have to do is get the cursor to the right place, point it in the right direction, pass appropriate parameters to the subroutine, and it's done. 260 calls 270 to get the cones. the parameters are *clen* (cylinder length) and *diam* (diameter). The subroutines have built in rules to decide how big the cones should be.

!3D Script, continued

```
GOSUB 320 !Outer Lateral member
MULy -1
GOSUB 320 !Outer Lateral member
DEL 1

END: !-----

300:
310:
320: !Outer Tube
    ADD -t2diam/2,-twid/2,thit/2
    ROTy 90
    CYLIND tlen*(tnum-1)+t2diam,t2diam/2
IF simcom THEN
    FOR k=1 TO tnum
        ADDz -t2diam
        ADDz tlen*(k-1)
        CYLIND t2diam*3, t2diam/1.2
        DEL 2
    NEXT k
ENDIF
DEL 2
RETURN
```



```
!All the diagonal tubes
FOR k=1 TO tnum
    ADDx tlen*(k-1)
    GOSUB 200: !Do Diagonal Tubes
    GOSUB 210: !Upper Wing tubes
    GOSUB 220: !Lower Wing tubes
    DEL 1
NEXT k
CUTEND
CUTEND
END: !-----
```

```
200: !Do Diagonal Tubes
LET clen=dlen: diam=t2diam
    ROTy 90-dtang
    GOSUB 260 !tube
    DEL 1
    ROTy -(90-dtang)
    GOSUB 260 !tube
    DEL 1
RETURN
```

```
210: !Upper wing tubes
LET clen=utlen: diam=t3diam
ADD 0,-twid/2,thit/2 !first upper
    ROTz angl
    ROTx 270 + ang2
    GOSUB 260 !tube
    DEL 2+1
ADD 0,-twid/2,thit/2 !second upper
    ROTz -angl
    ROTx 270 + ang2
    GOSUB 260 !tube
    DEL 2+1
ADD 0, twid/2,thit/2 !third upper
```

These are the end of the cut commands cutting the end joints to the correct length

Subroutine 260 does all the work, you just have to pass parameters to it

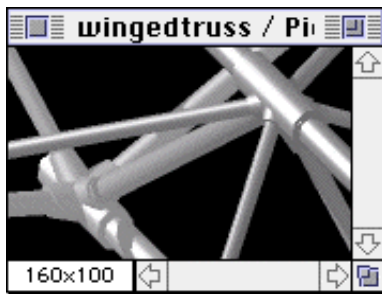
```
    ROTz 180 - angl
    ROTx 270 + ang2
    GOSUB 260 !tube
    DEL 2+1
ADD 0, twid/2,thit/2 !fourth upper
    ROTz 180 + angl
    ROTx 270 + ang2
    GOSUB 260 !tube
    DEL 2+1
RETURN
```

```
220: ! Do Lower Wing tubes
    ROTx 90-ltang
    clen=ltlen: diam=t4diam
    GOSUB 260: !Cylinder lower tube
    DEL 1
    ROTx -(90-ltang)
    GOSUB 260: !Cylinder lower tube
    DEL 1
RETURN
```

```
260: !Cylinder lower tube
IF simcom THEN GOSUB 270 !joint
CYLIND clen,diam/2
IF simcom THEN
    ADDz clen
    MULz -1
    GOSUB 270 !joint
    DEL 2
ENDIF
RETURN
```

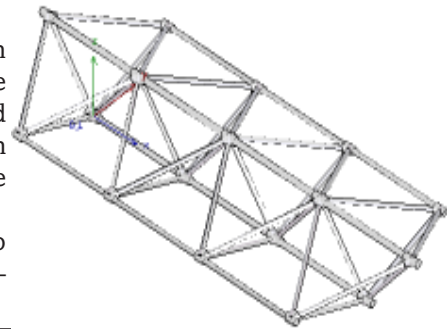
```
270: !Joint using a Cone
CONE diam*2,diam,diam/1.5,90,90
RETURN
```

Winged Truss



The joints in this truss look good even at close inspection. Perhaps the cones at the end of each tube could be a bit longer. From this, one can derive the design of cast steel sleeve joints.

One can get the script to write out to disk a list of the data for each tube – see later in this worksheet.

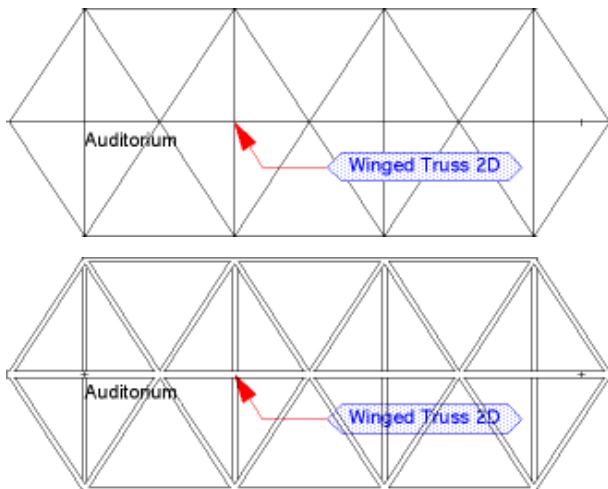


2D Script

Whether you are to use the Truview (using Project2) or a scripted 2D symbol, you will need Hotspots to pick the truss up with. You also need to write the stretchy hotspots before you draw the truss.

If the user prefers a scripted view (many seconds faster especially if you use the one with complex joints), the script is below. The sequence is similar to the 3D in that you do a FOR... NEXT loop for the diagonals in each module, and then put in the main longitudinal tubes. As a rule, 2D scripting is easy if you use a similar structure as for the 3D (loops etc)

As the truss has a name, it is possible to have it display its name in the 2D symbol, using TEXT2. The size of the Text is automatically calculated to be twice the diameter of the main tube.



!2D Script for Winged Truss

```

HOTSPOT2 -tlen/2,0 !Stretchy
HOTSPOT2 A-tlen/2,0 !Hotspots

!Hotspots on each truss element
FOR n=1 to tnum
  ADD2 tlen*(n-1),0
  HOTSPOT2 0,-twid/2
  HOTSPOT2 0, twid/2
  HOTSPOT2 0,0
  DEL 1
NEXT n

IF shownam THEN !Annotation
  fntz=2*tldiam*1000/A !Autosize
  DEFINE STYLE 'ttxt' 'Arial',fntz,1,0
  SET STYLE 'ttxt'
  TEXT2 0,-tldiam,iden
ENDIF

IF truvu THEN
  PROJECT2 3,270,2
ELSE
  RECT2 -tlen/2,tldiam/2,
    tlen*tnum-tlen/2,-tldiam/2
  FOR n=1 to tnum
    ADD2 tlen*(n-1),0
    POLY2 6,1,
      0, twid/2,
      tlen/2,0,
      0,-twid/2,
      -tlen/2,0,
      0,twid/2,
      0,-twid/2
    DEL 1
  NEXT n

!Lateral tubes
RECT2 0, twid/2+t2diam/2,
  (tnum-1)*tlen, twid/2-t2diam/2
RECT2 0, -(twid/2+t2diam/2),
  (tnum-1)*tlen,-twid/2+t2diam/2
ENDIF

```

This differs from the previous Winged truss in that the 2D script is fully scripted, not depending on Project2. As there is still the mysterious bug with the project2 solution, it may be preferable to script it. There is no trigonometry because in the plan view it's possible to describe locations using polygon points.

File Output

You were offered the possibility of outputting to disk a file with the data about the truss. Here it is. You can now add the final parameters.

First, it is important to know WHICH truss it is, as a building project may have several of different lengths.

So we offer the user the chance to name the truss ('iden') and at the same time, we offer them the chance to turn off the feature ('outdat') – we don't want dozens of trusses all simultaneously sending the same data to disk. If the user appoints a truss (the "show-truss") to do the work, the others can remain silent.

By the way, the name 'iden' will determine the name of the file. This file will be placed in the ArchiCAD Data Folder. It takes the name of the truss followed by .TXT.

146

Variable	Type	Name	Value
t2diam		Diameter Longitudinal tubing	0.070
t3diam		Diameter Upper wing tubing	0.070
t4diam		Diameter Lower wing tubing	0.070
truvu		Truview=ON Scripted View=OFF	Off
mat		Material of Tubing	11
simcom		Simple=OFF Complex=OFF	On
outdat		OutPut DataFile ON / OFF?	On
shownam		Show Name in 2D symbol ON/OFF	On

Add 'outdat' and 'shoname' to the parameters box

```
IF outdat THEN GOSUB 500: !File I/O
```

Add this line to the executive script, just above the END statement.

Make sure that different trusses have different names, or later trusses will overwrite the file written by one that was processed earlier.

Date and Time

We want to know the date of the report, in case it was different from the one we wanted. this, along with the name of the truss will appear in the report.

Date and Time are not in the manual, but they are in a Readme file in your Add-ons folder. In the date&time script, one uses a REQUEST which investigates a resource in ArchiCAD called 'DateTime' that knows the system time and date. The strings using percent signs can extract the data.

Actually, the whole time and date can be represented in a single string called '%c', but to show you how the strings work, I have joined several ones together – %A gives the day of the week; %d the day number, &B the month; %Y the year and century (millenium bug proof! ha-ha!); %X gives the time in AM or PM.

The rest of the DateTime strings can be found by reading the ReadMe file in the Add-ons Folder. The whole lot are combined into a text string called 'dstr'.

The Report

FILE INPUT and OUTPUT are covered in the appendix to the main GDL manual. If you are a Voyager, it's time to read that appendix! But then you may feel like reading the Cookbook for light relief.

First, the file must be opened and named, a channel defined, and you must set up the way in which the routine will separate data.

The OUTPUT commands then follow; each time you remind it which channel you are writing to (in case you have more than one file open at once – you could be reading from one file, and writing to the other.)

- The report adopts the name put into the 'iden' field in the parameters box.
- The report has TABs ('\t') between fields, so you could copy and paste this to a spreadsheet – or you could reformat it so that it could be read in by Filemaker.
- It's a good idea to have carriage returns to separate groups of data. A report of this kind would be used as a cutting schedule for the tubing and the design of the steel sleeve joints in the truss.

It is not really on to have it report in detail on structural matters, as you would require to ask the user to enter vastly more data, like Live and Dead loads, K-factors, steel grades, joint characteristics etc. However, a report on slenderness ratio is a good example of providing a warning to the designer. This could also flash up in the 2D symbol.

This report does provide the engineer with essential detail like tube lengths, from which calculations would be done – how else could those dimensions be derived, except by the engineer having to calculate them all again?

```
500:!File Output Routine
!Get the Date and Time
x=REQUEST("DateTime","%A %d %B %Y",dstr)
x=REQUEST("DateTime","%X",tstr)
```

REQUEST results in a number (X) which can be discarded, but as a result of making the request, you obtain the value of 'dstr' and 'tstr'.

```
500:!File Output Routine
!Get the Date and Time
x=REQUEST("DateTime","%A %d %B %Y",dstr)
x=REQUEST("DateTime","%X",tstr)
!Write the Report File
ch1=OPEN("TEXT",iden+".txt","SEPARATOR='\t',MODE=WO")
OUTPUT ch1,1,0,"Winged Truss Data:>",iden
OUTPUT ch1,1,0,"Output Date:",dstr
OUTPUT ch1,1,0,"Output Time:",tstr
OUTPUT ch1,1,0,"Cutting Schedule for Tubes in mm."
OUTPUT ch1,1,0," "
CLOSE ch1
RETURN
```

OPEN: Channel 1 is opened to allow output to a file, in a Write Only mode (WO), using tabs (\t) to separate.

OUTPUT: 'ch1' tells it where to send, '1' tells it to start a new line ('0' appends to the most recent line), the '0' does nothing. After that, you send the data you would like to have in the line, enclosing all text in quote marks.

Winged Truss Data:> MainHall

Output Date: Sunday 02 January 2000

Output Time: 17:33:04

Cutting Schedule for Tubes in mm.

Upper tubes

Number : 4
Length : 2100
Diameter: 100

Bottom tubes

Number : 3
Length : 2100
Diameter: 100

Diagonal tubes

Number : 8
Length : 1594.52
Diameter : 70
Declination: 48.8141

Upper Wing tubes

Number : 8
Length : 2168.52
Diameter : 70
Angle in Plan: 30.2564
Declination : 16.0625

Lower Wing tubes

Number : 4
Length : 1897.37
Diameter : 70
Declination: 18.4349

Upper Tube Sleeves : 5

Bottom Tube Sleeves: 4

Wing Sleeves LeftH : 4

Wing Sleeves RightH: 4

Slenderness Ratios

Upper : 1/ 21
Lower : 1/ 21
Diagonals: 1/ 23
UpperWing: 1/ 31
LowerWing: 1/ 27

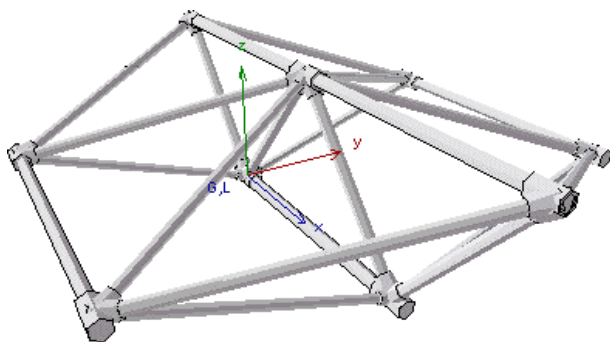
This is the kind of data that you can get a report to produce. It is much neater and more organised than a Components listing from the Properties scripting, but it is more tiresome for people to have to 'dig around' in the ArchiCAD data folder for the result. You should still try to give objects some Descriptor and Components listing in the Property script.

Do not be dismayed by apparent complexity of this script. Like everything else, type in the first few lines, do a 3D view and see if it works by looking at the Data Folder to see what has landed there. Don't forget that you must include the CLOSE statement or the file cannot be read.

As you build up the text, you add in the extra lines. You need to be pretty precise about the format of the report, so each line is precisely worked out, and appropriate labelling text is put in, always enclosed in quote marks. When you have got the above text working, continue, line by line, until you have completed. You do not need to do all of course – by now, you should have got the idea.

By the way, you notice that it is possible to include algebraic expressions in the OUTPUT statements, such as the ones that calculate Slenderness Ratio. It would be possible to express it as a decimal, but you can see that by playing with the layout, you can get it to lay it out in a more readable form.

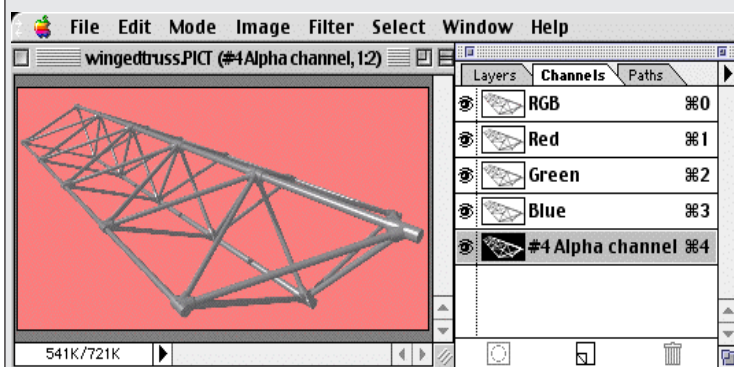
In this case, OUTPUT tells it to send, 'ch1' tells it where to send, '1' tells it to start a new line ('0' should append to the most recent line according to the manual), the final '0' does nothing at all. After all that, you send the data you would like to have in the line, carefully enclosing all text material in quote marks.



```
500:!File Output Routine
!Get the Date and Time
x=REQUEST("DateTime","%A %d %B %Y",dstr)
x=REQUEST("DateTime","%X",tstr)
!Write the Report File
ch1=OPEN("TEXT",iden+".txt","SEPARATOR='\t',MODE=WO")
OUTPUT ch1,1,0,"Winged Truss Data:>",iden
OUTPUT ch1,1,0,"Output Date:",dstr
OUTPUT ch1,1,0,"Output Time:",tstr
OUTPUT ch1,1,0,"Cutting Schedule for Tubes in mm."
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Upper tubes"
OUTPUT ch1,1,0," Number :",tnum
OUTPUT ch1,1,0," Length :",tlen*1000
OUTPUT ch1,1,0," Diameter:",tldiam*1000
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Bottom tubes"
OUTPUT ch1,1,0," Number :",tnum-1
OUTPUT ch1,1,0," Length :",tlen*1000
OUTPUT ch1,1,0," Diameter:",tldiam*1000
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Diagonal tubes"
OUTPUT ch1,1,0," Number :",tnum*2
OUTPUT ch1,1,0," Length :",dlen*1000
OUTPUT ch1,1,0," Diameter :",t2diam*1000
OUTPUT ch1,1,0," Declination:",dtang
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Upper Wing tubes"
OUTPUT ch1,1,0," Number :",tnum*2
OUTPUT ch1,1,0," Length :",utlen*1000
OUTPUT ch1,1,0," Diameter :",t3diam*1000
OUTPUT ch1,1,0," Angle in Plan:",ang1
OUTPUT ch1,1,0," Declination :",ang2
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Lower Wing tubes"
OUTPUT ch1,1,0," Number :",tnum
OUTPUT ch1,1,0," Length :",ltlen*1000
OUTPUT ch1,1,0," Diameter :",t4diam*1000
OUTPUT ch1,1,0," Declination:",ltang
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Upper Tube Sleeves :",tnum+1
OUTPUT ch1,1,0,"Bottom Tube Sleeves:",tnum
OUTPUT ch1,1,0,"Wing Sleeves LeftH :",tnum
OUTPUT ch1,1,0,"Wing Sleeves RightH:",tnum
OUTPUT ch1,1,0," "
OUTPUT ch1,1,0,"Slenderness Ratios"
OUTPUT ch1,1,0,"Upper : 1/",(INT(tlen/tldiam+0.5))
OUTPUT ch1,1,0,"Lower : 1/",(INT(tlen/tldiam+0.5))
OUTPUT ch1,1,0,"Diagonals: 1/",(INT(dlen/t2diam+0.5))
OUTPUT ch1,1,0,"UpperWing: 1/",(INT(utlen/t3diam+0.5))
OUTPUT ch1,1,0,"LowerWing: 1/",(INT(ltlen/t4diam+0.5))
CLOSE ch1
RETURN
```

Tips and Tricks

Doing it with Alpha Channels : think BIG! VERY VERY BIG!



SOMETIMES it is just too difficult to set up the background in ArchiCAD. Do you want to model the entire town or perhaps use a photograph of it as a background and spend hours trying to get it to line up with the model? Yes!.... but No, I haven't got the time.

Much easier still is to use a perfectly white background. When you save the render, ArchiCAD saves the background as an Alpha Channel. The proof is on the left – Photoshop displays the channel perfectly. Save as a PICT or a TIFF with no compression. This can be brought into Photoshop and laid over a photographic background. Make use of the transparency of the alpha channel.

Anti aliasing is good, but for better results for images with alpha channels/transparency, render VERY LARGE with NO Anti-aliasing – like 4000 pixels wide! Remove backgrounds with 100% precision. Then rescale the image to the size you want.

Lattices: More complex shapes

Not as difficult as you may think

SOMETIMES you have to make a compromise between parametric technology, and 'clicking in the plan to steal from ArchiCAD'. We will make a curved space frame where the locations of primary points along its surface are defined by clicking, but other characteristics are parametric. The lattice can be developed further to give a higher level of parametric capability.

In this case, we have a roof already made, and we go to a Section window. Its easiest to clickout the diagonals first using the continuous Linetool (Polyline), so that you get the vertices nicely spaced. Click in the lower lines of the spacedeck, snapping to the vertices. Copy both of these and paste into the floor plan, then click in the lines of the upper members (you could do it earlier but there may be confusing snapping points in the roof fabric that cause you to miss the vertices.) Move the lines to somewhere near the main Origin, preferably the extreme left corner to be on zero.

Use the drag and drop method to drag ONLY THE TOP and BOTTOM lines into the 2D Script window of a new GDL Object. Change the headers so that they become a PUT statement. Cut and Paste these into the Master Script. We are ready to build the frame. The program we write will work for any set of numbers created in this way. You can use this object again and again.

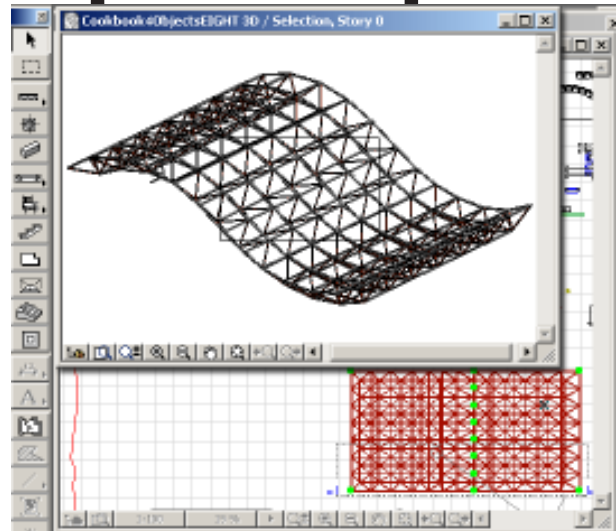
We put the numbers into an array

We are limiting it here to 20 points (you could add more). We need to write routines that will store these numbers in an array. We have mentioned arrays many times earlier, the ones here are dynamic arrays created in the Master Script, not Parameter arrays.

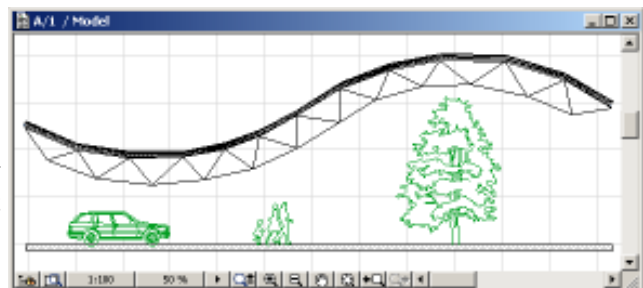
We declare the likely size of the array with a DIM statement, make sure it's safely larger than the number we will actually use. We need arrays for the X and Z values of the points, plus twist angles for each top tube.

```
!Curved spacedeck: Master Script
!Store in an array
DIM tx[20],tz[20] ! Points on the top
DIM bx[20],bz[20] ! Points on the bottom
DIM dx[40],dz[40] ! Points on the diagonals
DIM twt[20] !Twist angle top in cross tubes
DIM twb[20] !Twist angle bottom

!Top level
PUT -0,0, 1,
2.154987149632, -0.9170764513291, 1,
4.327020796939, -1.350285995269, 1,
6.562311181797, -1.314987298444, 1,
8.381395402791, -0.9789795244067, 1,
9.860819677821, -0.4131370856228, 1,
11.42146364404, 0.4413916146407, 1,
13.31680906168, 1.598128616054, 1,
15.30049749881, 2.44317319297, 1,
17.5099790329, 2.877467134283, 1,
20.15874336325, 2.731740204065, 1,
22.63568039847, 2.00658441796, 1,
24.72146266704, 0.839168080994, -1
```



This is what we want: a bendy spacedeck. The profile is fixed, but we can stretch it widthways, and provide hotspots.



In AC7 you have to use the Fill tool to get the points. AC8 and AC9 have a Polyline drawing tool.

We use a WHILE DO loop to run through the list counting the number of points, storing the X and Zs in the array, and throwing away the third number on each line of the Line tool script.

```
k=0
WHILE NSP DO
k=k+1
tx[k]=GET(1)
tz[k]=GET(1)
nothing=GET(1)
ENDWHILE
numt=k
```

'k' is a counter here, but it has to be manually updated

```
!Bottom level
PUT 0.8915248799115, -1.452438081482, 1,
2.960938636525, -2.114626824624, 1,
5.298840429751, -2.390505674063, 1,
7.490314916727, -2.220957359614, 1,
9.545282878069, -1.689359651964, 1,
11.42146364404, -0.8545555737123, 1,
13.14805774461, 0.2451485099629, 1,
14.75332930093, 1.186391085724, 1,
16.67102669916, 1.769052816456, 1,
18.71558942951, 1.843338465032, 1,
21.14286105828, 1.348931363385, 1,
23.05211021206, 0.567871936596, -1
```

```
k=0
WHILE NSP DO
k=k+1
bx[k]=GET(1)
bz[k]=GET(1)
nothing=GET(1)
ENDWHILE
numb=k
```

Don't copy over the diagonals, we can calculate these points by combining the top and bottom points. Using 'k' as a counter, we can find out how many points there actually are.

Master Script and Parameters

We want this to be stretchy widthways, so it's time to make some parameters. We are going to make 'B' as the width, and set a number of frames, 'numf'. Make a few others, for materials, pens etc. Use the master script to do some idiotproofing and to work out some internal parameters, like the width and half width of each frame section.

Variable	Type Name	Value
A	X Dimension	1000
B	Y Dimension	7000
ZZYZX	Z Dimension	1000
Instructions		
numf	Number of frames, grid	5
truvu	Truview 2D ON/OFF?	Off
wid	Truss spacing single width	1400
Ac_show2DHot.	Show 2D Hotspots in 3D	Off
spaceframe_..	Tubing Detail	
tub_shap	Tubing Shape	Square
sect	Main Section size	80
secd	Diagonal Section size	60
spaceframe_..	Material, Pens	
matl_main	Material Main Tubing Top+Bot	18
matl_diag	Material Tubing Diagonal	45
gs_cont_pen	Contour Pen	1
gs symb pen	2D Symbol Pen	5

We run a routine to work out the points of the diagonals. You can do this with a smart routine that alternates between the top and bottom lines, building up a new array.

```
!Curved spacedeck: 3D Script
PEN gs_cont_pen
IF GLOB_CONTEXT=2 THEN PEN gs_symb_pen
TOLER 0.005

!---Space grid-----
!Top and Bottom linear tubes
MATERIAL matl_main
FOR n=1 TO numf+1
  ADDY wid*(n-1)
  GOSUB 200:!Top deck
  DEL 1
NEXT n

FOR n=1 TO numf
  ADDY wid/2+wid*(n-1)
  GOSUB 210:!Bottom deck
  DEL 1
NEXT n

!Diagonal Tubes
MATERIAL matl_diag
FOR n=1 TO numf
  ADDY wid/2+wid*(n-1)
  m=1
  GOSUB 220:!diagonals
  m=-1
  GOSUB 220:!diagonals
  DEL 1
NEXT n

!Cross Tubes
MATERIAL matl_main
GOSUB 250:!Top deck cross tubing single
IF numf>1 THEN
  GOSUB 260:!Bot deck cross tubing single
ENDIF

!Hotspots on edges
GOSUB 300:!Hotspots on one top deck line
ADDY B
GOSUB 300:!Hotspots on one top deck line
DEL 1
GOSUB 310:!Hotspots on edges
END:=====
```

This flag 'm' switches the directions of the diagonals. Run the same subroutine, but with a negative 'm' and the diagonals all lean over the other way from a positive 'm'.

Continued next page

```
!Continuing the Master Script
!----Space Grid calculations-----
wid=B/numf !width of each frame
w2=wid/2 !Half Width
IF numf<=1 THEN numf=1
IF secd>=sect THEN secd=sect
PARAMETERS numf=numf, wid=wid, secd=secd
LOCK 'wid'

!----Calculate the Diagonals----
FOR k=1 TO numt !Alternating top and bottom
  dx[k*2-1]=tx[k]
  dz[k*2-1]=tz[k]
  dx[k*2]=bx[k]
  dz[k*2]=bz[k]
NEXT k
numd=numt*2-1

!--Twist angle calculation for the cross tubes--
!--Top tubes
FOR k=1 TO numt-1
  twt[k]=ATN((tz[k+1]-tz[k])/(tx[k+1]-tx[k]))
NEXT k
twt[numt]=twt[numt-1] !last tube

!--Bottom tubes
FOR k=1 TO numd-1
  twb[k]=ATN((bz[k+1]-bz[k])/(bx[k+1]-bx[k]))
NEXT k
twb[numd]=twb[numd-1] !last tube

!Write in the ideal place for pick up hotspots
!at middle and end
spt=tx[1] !Start point
mpt=tx[INT(1+numt/2)] !Middle Pickup point
ept=tx[numt] !End point

!Select shape and set flag
VALUES 'tub_shap' 'Square','Round'
IF tub_shap='Square' THEN ts=0
IF tub_shap='Round' THEN ts=1
```

This takes some thinking out, but it's better than requiring the user to copy and paste in an separate and extra set of points for the diagonals.

```
!Curved spacedeck: 2D Script
PEN gs_symb_pen
HOTSPOT2 spt,0
HOTSPOT2 spt, B
HOTSPOT2 ept,0
HOTSPOT2 ept, B

FOR n=1 TO numf+1
  HOTSPOT2 mpt, wid*(n-1)
NEXT n

PROJECT2 3,270,1
```

We also need a routine to rotate the cross tubes of the spacedeck – this is the kind of thing one thinks of and adds later when one has got the main frame working correctly.

To round off the Master Script, calculate some X dimensions for positioning pickup hotspots in the 2D and 3D – the start, middle and end points.

2D Script

We should write a full 2D script to make the object draw itself very quickly in the plan, but for the moment, let's leave it as a PROJECT2.

3D Script

The IF statement allows us to use a different pen colour in the 2D symbol. Set TOLER to control the number of polygons. In constructing the tubes, we delegate the hard work to the subroutines – these loops merely decide how many tubes we will have. The number of parallel frames will be 'numf'.

100: !Square Section outline Main tubes

```

PUT sectn/2, sectn/2, 0
PUT -sectn/2, sectn/2, 0
PUT -sectn/2, -sectn/2, 0
PUT sectn/2, -sectn/2, 0
PUT sectn/2, sectn/2, -1
RETURN

```

We can use these two routines for PUTting the profile coordinates for the small and large tubing.

101: !Round Section outline Main tubes

```

PUT 0,0, 901
PUT sectn/2, 360,4001
RETURN

```

This routine draws the top tube. We have two phantom points either side of the real points on the array. As we don't know the number of points on either the profile or the path, we have to use this trick with NSP.

200: !-----Top deck-----

```

sectn=sect !tube section
GOSUB 100+ts !Section
prpts=NSP/3 !profile points
PUT -100,0,0,0
FOR k=1 TO numt
  PUT tx[k],0,tz[k],0
NEXT k
PUT 100,0,0,0

```

The routine for the bottom tube is the same.

```

ptpts=(NSP-prpts*3)/4
TUBE prpts,ptpts,63,GET(NSP)

```

RETURN

210: !-----Bottom deck-----

```

sectn=sect !tube section
GOSUB 100+ts !Section
prpts=NSP/3 !profile points
PUT -100,0,0,0
FOR k=1 TO numb
  PUT bx[k],0,bz[k],0
NEXT k
PUT 100,0,0,0

```

```

ptpts=(NSP-prpts*3)/4
TUBE prpts,ptpts,63,GET(NSP)

```

RETURN

220: !-----Diagonals-----

```

sectn=sect !tube section
h=0 !top or bottom
GOSUB 100+ts !Section
prpts=NSP/3 !profile points
PUT -1,0,0,0
FOR k=1 TO numd
  h=1-h !switch from 1 to 0
  PUT dx[k],m*h*wid/2,dz[k],0
NEXT k
PUT 100,0,0,0

```

We have the 'm' flag deciding which way to lay, but the 'h' flag is deciding whether the tube is touching the top or bottom main tubes. This is a bit crude, the diagonal is one continuous zigzag line and can result in some spiky mitring details at the joints.

```

ptpts=(NSP-prpts*3)/4
TUBE prpts,ptpts,63,GET(NSP)

```

RETURN

Summary

Frames like this are in the 'quick and dirty' category, i.e. one is working towards the geometric configuration, but is not looking to make beautiful joint details etc. This would require calculation of the directions and lengths of every tube – each line of which is different!

We also need to find a way to improve the detailing at the joints – to make them less 'spiky'. the joints here meet precisely at the node point, but in

250: !--Top deck cross tubing--

```

FOR k=1 TO numt
  GOSUB 100+ts !Profile
  prpts=NSP/3 !profile points
  PUT tx[k],0, tz[k], twt[k]
  PUT tx[k],0, tz[k], twt[k]
  PUT tx[k],B, tz[k], twt[k]
  PUT tx[k],B+1, tz[k], twt[k]
  TUBE prpts,4,63,GET(NSP)
NEXT k
RETURN

```

These could be done with prism, but we have a good thing going here with TUBE so we stay with it. It allows us to use the twist, the fourth parameter on the path points list in a TUBE command.

260: !--Bottom deck cross tubing--

```

FOR k=1 TO numb
  GOSUB 100+ts !Profile
  prpts=NSP/3 !profile points
  PUT bx[k], 0, bz[k], twb[k]
  PUT bx[k], wid/2, bz[k], twb[k]
  PUT bx[k], B-wid/2, bz[k], twb[k]
  PUT bx[k], B, bz[k], twb[k]
  TUBE prpts,4,63,GET(NSP)
NEXT k
RETURN

```

The bottom deck cross tubes are slightly shorter than the top tubes.

300: !3D Hotspots:Top Deck side edges--

```

FOR k=1 TO numt
  HOTSPOT tx[k],0,tz[k]
NEXT k
RETURN

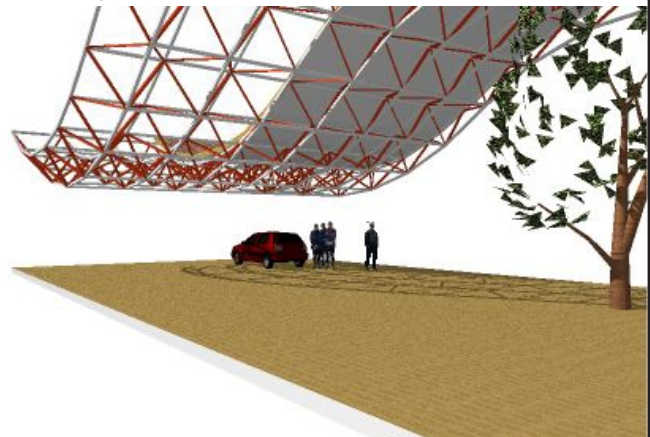
```

310: !Hotspots on start and end edges--

```

FOR k=1 TO numf+1
  HOTSPOT tx[1], wid*(k-1), tz[1]
  HOTSPOT tx[numt],wid*(k-1), tz[numt]
NEXT k
RETURN

```



real frames, there has to be space for the welding of the tubes.

Because GDL is a programming language, it would be easy enough to printout a list of calculated tube lengths for manufacturing purposes – indeed, how else would one calculate all this essential data? It's worth expending this effort on a more parametric version of this framework, which we will try next.

Curved Lattice: make it Parametric!

GRAPHICAL Hotspots have transformed the capabilities of GDL. The curved lattice on the earlier pages rely on a complex list of XY coordinates in the Master Script. There are pitfalls for the unwary, especially those who do not quite understand the way PUT and GET work. It's possible to make a mistake in clicking out the polyline, or in pasting the code into the Master Script. And it's not really parametric.

Let's try to use Parameter Arrays to make a similar curved lattice, but one that offers a staggering level of freedom to configure the framework. This does not actually do calculations for the structure (you would still need this verified by an engineer), but you could provide your engineer with detailed information on tube lengths (and slenderness ratios) and greatly assist the fabrication process. Previously, this kind of frame, irregular in configuration, would be too complex to manufacture. GDL can make the impossible possible! The way to make it configurable is not to make the user type in vast numbers of coordinates (this can be a fall-back method); the best way is to display a 2D configuration tool using hotspots which makes it easy to shape the lattice to fit a roof outline.

In just playing with this object, I am pretty amazed at its flexibility – there must be a practical use for it, if it can configure frames of a complexity that normal design methods could not tackle.

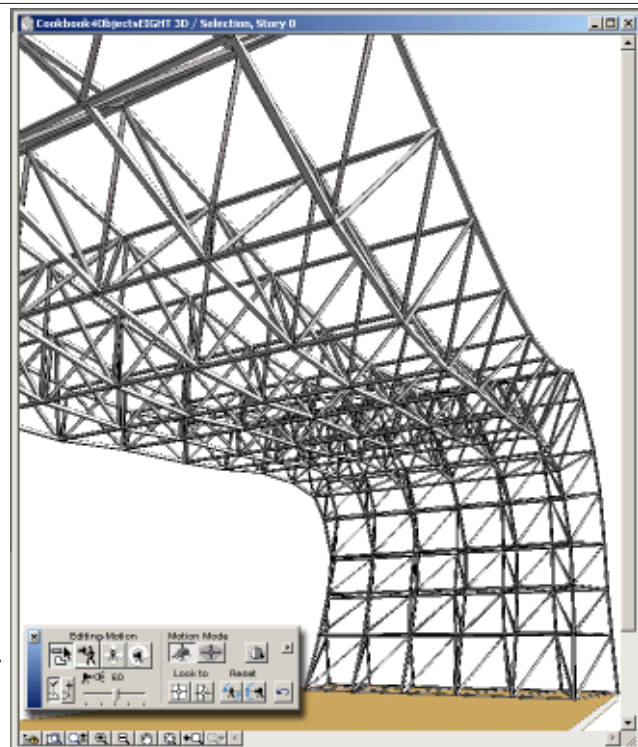
Author's note

This frame is vastly smarter than the one on the previous page. One of the problems I have as an author is to make the more advanced examples complex enough to amuse and educate the advanced GDL users, but short enough to stay within 2-3 pages. Please forgive me for leaving out some of the more complex elaborations that could be added.

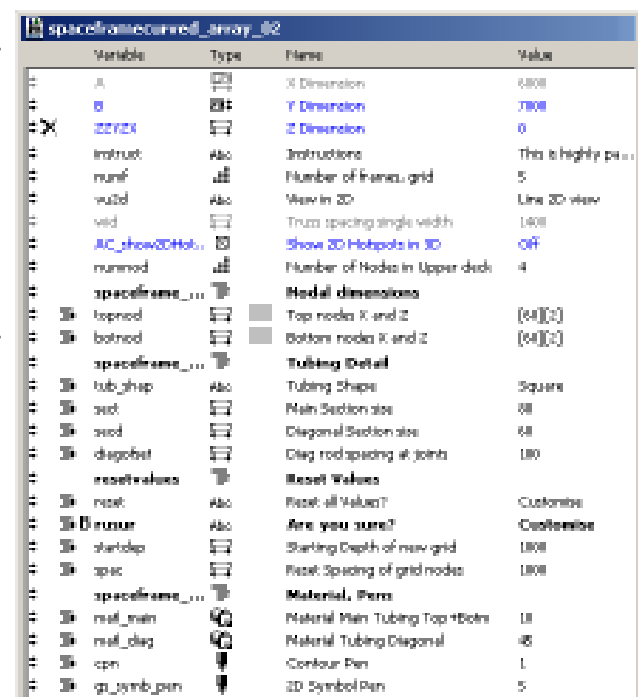
Parameters and Master Script

We need to have parameter arrays for the top and bottom nodes if we want to be able to use Graphical Hotspots. Let's base the overall length of the frame on 'numnod', number of nodes in the top surface, and call the top and bottom nodes 'topnod' and 'botnod' respectively. (The 'startdep' here is for future use, for initialising a new frame.) We need parameters for the Tubing sectional sizes, and a spacing dimension where the small diagonals meet the larger main tubes. The tubing can be round or square in section.

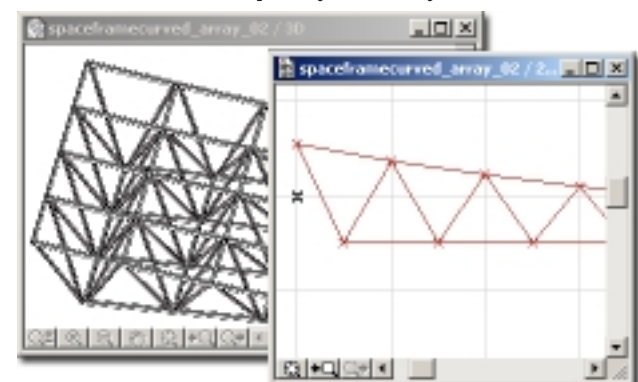
One of the smart features we will give this frame is to enable the user to configure the frame in the 2D view by laying it on its side.



This frame isn't only for roofs. It can be arched over to form an enclosure like this.



This Parameter Table include the 'Reset' routine for initialising the lattice. You can adapt this for other objects.



!Curved spacedeck: Master Script

x=1:z=2 !Reference for arrays

!----Menus-----

VALUES 'tub_shap' 'Square', 'Round'

IF tub_shap='Square' THEN ts=0

IF tub_shap='Round' THEN ts=1

vu0='True view hidline'

vu1='True view solid'

vu2='Line 2D view'

vu3='Sideways configuration view'

VALUES 'vu2d' vu0, vu1, vu2, vu3

!----Space Grid calculations----

!Frame overall is stretchy

!number of bays is parametric

wid=B/numf !width of each section

LOCK 'wid'

IF numf<=1 THEN numf=1

IF secd>=sect THEN secd=sect

w2=wid/2 !Half Width

df2=diagofset*0.5 !Offset for Diagonals

PARAMETERS wid=wid, numf=numf, secd=secd

!Limit Number of points for the Array version--

IF numnod<=2 THEN numnod=2

IF numnod>=60 THEN numnod=60

PARAMETERS numnod=numnod

numd=numnod*2-1 !Number of Diagonal points

!-----

!Initialise Points in the arrays: Reset routine

resetval=0 !Customise not reset

rv0='Customise'

rv1='Reset all values'

VALUES 'reset' rv0, rv1

VALUES 'rusur' rv0, rv1

IF reset=rv0 THEN

PARAMETERS rusur=rv0

HIDEPARAMETER 'rusur'

ENDIF

IF reset=rv1 THEN

IF rusur=rv1 THEN resetval=1

ENDIF

IF resetval THEN

FOR k=1 TO 60

PARAMETERS topnod[k] [x]=spac*(k-1) !X Values

PARAMETERS topnod[k] [z]=0 !Z Values

PARAMETERS botnod[k] [x]=spac*(k-0.5) !X Values

PARAMETERS botnod[k] [z]=-startdep !Z Values

NEXT k

HIDEPARAMETER 'rusur'

PARAMETERS reset=rv0

resetval=0 !reset flag to zero

ENDIF

!-----

!Store Twist values in an array

DIM twt[] !Twist angle top in cross tubes

DIM twb[] !Twist angle bottom

!--Twist angle calculation for the cross tubes--

!--Top tubes

twt[1]=ATN((topnod[2] [z]-topnod[1] [z])/(topnod[2] [x]-topnod[1] [x]))

FOR k=2 TO numnod-1

twt[k]=ATN((topnod[k+1] [z]-topnod[k-1] [z])/(topnod[k+1] [x]-topnod[k-1] [x]))

IF twt[k]>=180 AND twt[k]<=360 THEN twt[k]=twt[k]-360

NEXT k

twt[numnod]=twt[numnod-1] !last node

!--Bottom tubes

twb[1]=ATN((botnod[2] [z]-botnod[1] [z])/(botnod[2] [x]-botnod[1] [x]))

FOR k=2 TO numnod-1

twb[k]=ATN((botnod[k+1] [z]-botnod[k-1] [z])/(botnod[k+1] [x]-botnod[k-1] [x]))

IF twb[k]>=180 AND twb[k]<=360 THEN twb[k]=twb[k]-360

NEXT k

twb[numnod-1]=twb[numnod-2] !last one

Note that when lines are too long to fit, you can use a \ symbol and continue on the next line.

Master Script

This appears complex, but real objects of this sophistication have even longer Master Scripts as the author thinks of more features.

In this case we set up some of the main menus for tubing sections and 2D representation. We calculate some of the essential parameters for the space grid width, and do some idiotproofing to limit the size of the frame. We could have 60 nodes, we could choose to have 600! But here, we have limited it to 60.

Note that when we have idiotproofed or corrected parameters, we do not forget to notify the user with a PARAMETERS command, and apply a LOCK if the parameter is not to be edited.

'Reset' – smart Initialisation routine

In this script I have included a very interesting routine, which is an Initialisation routine – this straightens out the frame in case it is an insane shape, or it can be used to set the frame to near-enough the required shape then tweak it to its final shape – you can set a starting value of spacing of the nodes and depth of the frame.

What makes it smart is the way it has a 'conversation' with the user. If the user decides to reset the structure, they may regret this, so the parameters come up with the question "Are you sure?" and the user has to issue the reset command again to make it happen.

The advanced GDL user should play with this object and explore its capabilities.

This reset routine can be used again with other objects where you can devise a way of initialising the configuration to a default value. It may not matter with a chair or table, but if you have spend hours configuring a complex lattice, you don't want to straighten it out by accident.

Twist Values, and Welding details

One smart feature of this frame is to twist the cross tubes of the frame so that they are not always horizontal. We did this for the previous object, but this is more accurate – it averages the angle between the preceding and the next nodes to the one in question, and the twist angle also govern the spacing of the diagonals. Because it unrealistic for all those tubes to come precisely together, we have a parameter for a welding offset for the diagonals. It has made this object much more complex than I would have liked, but without it, the sharp eyed readers of this book would be crying out that I had ignored an important feature. It gets very complicated if the frame is distorted greatly.

Finally, we calculate the optimum position of the midpoints, and the location of the ends (to improve the 2D Hotspots) and send a value of 'A' to the parameter table.

!Start and End Points

spt=topnod[1] [x] !Start

ept=topnod[numnod] [x] !End

mpt=topnod[INT(numnod/2)] [x] !Midpoint

PARAMETERS A=ept, zzyzx=0

LOCK 'A'

HIDEPARAMETER 'zzyzx'

!Curved spacedeck: 3D Script

```
PEN cpn
IF GLOB_CONTEXT=2 THEN PEN gs_symb_pen
TOLER 0.005
```

*Change pen
colour if it's
a 2D
PROJECT2.*

!-----Hotspots on edges-----

```
GOSUB 1000:!Hotspots on one top deck line
ADDY B
GOSUB 1000:!Hotspots on one top deck line
DEL 1
GOSUB 1010:!Hotspots on one edge
```

!-----Space grid-----**!Top and Bottom linear tubes**

```
MATERIAL matl_main
FOR n=1 TO numf+1
  ADDY wid*(n-1)
  GOSUB 200:!Top deck
  DEL 1
  NEXT n
FOR n=1 TO numf
  ADDY wid/2+wid*(n-1)
  GOSUB 210:!Bottom deck
  DEL 1
  NEXT n
```

*The main routines are in
compact routines with the
heavy work delegated to
subroutines. The loops here
just repeat the modules in
the Y direction.*

*The M value swaps the
direction of the tubes so that
the 'vees' get repeated both
sides of the bottom tubes.*

!Diagonal Tubes

```
MATERIAL matl_diag
FOR n=1 TO numf
  ADDY wid/2+wid*(n-1)
  m=1
  GOSUB 220:!diagonals
  m=-1
  GOSUB 220:!diagonals
  DEL 1
  NEXT n
```

!Cross Tubes

```
MATERIAL matl_main
GOSUB 250:!Top deck cross tubing single
IF numf>1 THEN
  GOSUB 260:!Bot deck cross tubing single
ENDIF
```

END:!

100:!Square Section outline Main tubes

```
PUT sectn/2, sectn/2, 0
PUT -sectn/2, sectn/2, 0
PUT -sectn/2, -sectn/2, 0
PUT sectn/2, -sectn/2, 0
PUT sectn/2, sectn/2, -1
```

*We have two choices of
section for the tubing,
round or square. The
subroutines decide
which size section to
use.*

101:!Round Section outline Main tubes

```
PUT 0,0, 901
PUT sectn/2, 360,4001
RETURN
```

200:!-----Top deck Array-----

```
sectn=sect !tube section
GOSUB 100+ts !Section
prpts=NSP/3 !profile points
PUT -10000,0,0,0
FOR k=1 TO numnod
  PUT topnod[k] [x], 0, topnod[k] [z], 0
  NEXT k
  PUT 10000,0,0,0
ptpts=(NSP-prpts*3)/4
TUBE prpts,ptpts,63,GET(NSP)
RETURN
```

*This top tube just
follows the array of
nodes. The Phantom
points are a long way
away!*

210:!--Bottom deck-----

```
sectn=sect !tube section
GOSUB 100+ts !Section
prpts=NSP/3 !profile points
adjb1x=(df2+secd)*COS(twb[1])
adjb1z=(df2+secd)*SIN(twb[1])
adjbnx=(df2+secd)*COS(twb[numnod-1])
adjbnz=(df2+secd)*SIN(twb[numnod-1])
PUT -10000,0,0,0
PUT botnod[1] [x]-adjb1x,0,botnod[1] [z]-adjb1z,0
FOR k=1 TO numnod-1
  PUT botnod[k] [x], 0, botnod[k] [z], 0
  NEXT k
  PUT botnod[numnod-1] [x]+adjbnx,0,
    botnod[numnod-1] [z]+adjbnz,0
  PUT 10000,0,0,0
  ptpts=(NSP-prpts*3)/4
  TUBE prpts,ptpts,63,GET(NSP)
RETURN
```

*The bottom tube
is more
complicated.
Because we want
to improve the
look of the weld
of the diagonal
tubing we
introduce a
spacing
parameter.*

220:!--Diagonals-----

```
sectn=secd !tube section
FOR k=1 TO numnod-1
  !!Do each Rod in the Vee separately
  adjtx=df2*COS(twt[k]) !Adjustments
  adjtz=df2*SIN(twt[k])
  adjbx=df2*COS(twb[k])
  adjbz=df2*SIN(twb[k])
  x1=topnod[k] [x]+adjtx
  y1=m*wid/2
  z1=topnod[k] [z]+adjtz
  x2=botnod[k] [x]-adjbx
  y2=0
  z2=botnod[k] [z]-adjbz
  GOSUB 900:!Build a tube
  x1=botnod[k] [x]+adjbx
  y1=0
  z1=botnod[k] [z]+adjbz
  x2=topnod[k+1] [x]-adjtx
  y2=m*wid/2
  z2=topnod[k+1] [z]-adjtz
  GOSUB 900:!Build a tube
  NEXT k
RETURN
```

*This gets very
complicated. We
have twisted the
cross tubes, but
we have to apply
a similar twist to
the locations of
the welds for the
diagonals.
Hard GDL-case
will appreciate
this. It's done with
separate tubes
with butt-ends
and a small
Adjustment in X
and Z for the top
and bottom
points.*

250:!Top deck cross tubing single

```
sectn=sect !tube section
FOR k=1 TO numnod
  GOSUB 100+ts !Profile
  prpts=NSP/3 !profile points
  PUT topnod[k] [x], -1, topnod[k] [z], twt[k]
  PUT topnod[k] [x], 0, topnod[k] [z], twt[k]
  PUT topnod[k] [x], B, topnod[k] [z], twt[k]
  PUT topnod[k] [x], B+1, topnod[k] [z], twt[k]
  TUBE prpts,4,63,GET(NSP)
  NEXT k
RETURN
```

*Having calculated
the twist angles,
we can apply
them in the cross
tubes.*

!continued next page

3D Script

This is a complex object, and is best looked at on the CD. It demonstrates how important it is to document every subroutine with a title. It contains most of the heavy GDL artillery except for UI, Surfaces and File I/O. Surfaces could easily be added once the coordinates are in the array.

!Flexible Spacegrid: Parameter Script

```
VALUES 'instruct' 'This is highly parametric, best done by dragging vertices with',
'smart hotspots. The 3D ones are a bit dodgy, the safest method is using',
'the 2D one in the sideways elevation view. You can copy and paste the',
'object into a Section/Elevation, then configure',
'the spacegrid very accurately in that view, following the profile of the roof,',
'then copy and paste back to the Plan view and then lay it flat again!',
'Beware of using the <solid> view in the 2D symbol, its very slow.'
```

*As usual, I like to have
some instructions with
every object, and a
popdown text is the
easiest way to do this.*

!3D Script continued**260:!--Bottom deck cross tubing--**

```
sectn=sect !tube section
FOR k=1 TO numnod-1
  GOSUB 100+ts !Profile
  prpts=NSP/3 !profile points
  PUT botnod[k][x],0, botnod[k][z], twb[k]
  PUT botnod[k][x],wid/2, botnod[k][z], twb[k]
  PUT botnod[k][x],B-wid/2,botnod[k][z], twb[k]
  PUT botnod[k][x], B, botnod[k][z], twb[k]
  TUBE prpts,4,63,GET(NSP)
NEXT k
RETURN
```

900:!--Tubes for Diagonals-----

```
x0=x1+(x1-x2)
x3=x2-(x1-x2)
y0=y1+(y1-y2)
y3=y2-(y1-y2)
z0=z1+(z1-z2)
z3=z2-(z1-z2)

GOSUB 100+ts !Profile
TUBE NSP/3,4,63,
  GET(NSP),
  x0,y0,z0,0,
  x1,y1,z1,0,
  x2,y2,z2,0,
  x3,y3,z3,0
RETURN
```

1000:!--3D Hotspots on Top Deck-----

```
FOR k=1 TO numnod
  hsid=hsid+1
  HOTSPOT topnod[k][x],0,0, hsid,topnod[k][z],1+128
  hsid=hsid+1
  HOTSPOT topnod[k][x],0,topnod[k][z],hsid,topnod[k][z],2
  hsid=hsid+1
  HOTSPOT topnod[k][x],0,-100, hsid,topnod[k][z],3
NEXT k
RETURN
```

1010:!--Hotspots on one edge

```
FOR n=1 TO numf+1
  HOTSPOT topnod[1][x],wid*(n-1),topnod[1][z]
  HOTSPOT topnod[numnod][x],wid*(n-1),topnod[numnod][z]
NEXT n
RETURN
```

!Curve spacedeck: 2D Script

```
PEN gs_symb_pen
IF vu2d=vu0 THEN !Plan in true view line
  GOSUB 1000:!--Hotspots Plan view
PROJECT2 3,270,2
ENDIF

IF vu2d=vu1 THEN !Plan in true view solid
  GOSUB 1000:!--Hotspots Plan view
PROJECT2 3,270,3 !solid
ENDIF
```

!2D Fully scripted version Plan

```
IF vu2d=vu2 THEN !Plan in 2D
  GOSUB 1000:!--Hotspots Plan view
```

!Top Deck

```
FOR n=1 TO numf+1
  FOR k=1 TO numnod-1
    LINE2 topnod[k][x], wid*(n-1),
      topnod[k+1][x], wid*(n-1)
  NEXT k
NEXT n
```

!Bottom Deck

```
FOR n=1 TO numf
  FOR k=1 TO numnod-2
    LINE2 botnod[k][x], w2+wid*(n-1),
      botnod[k+1][x], w2+wid*(n-1)
  NEXT k
NEXT n
```

!Top cross tubes

```
FOR k=1 TO numnod
  LINE2 topnod[k][x],0, topnod[k][x], B
NEXT k
```

!Bottom cross tubes

```
IF numf>=1 THEN
  FOR k=1 TO numnod-1
    LINE2 botnod[k][x], w2, botnod[k][x], B-w2
  NEXT k
ENDIF
```

!!Diagonals

```
m=1
FOR n=1 TO numf
  ADD2 0, wid*(n-1)
  FOR k=1 TO numnod-1
    LINE2 topnod[k][x],0, botnod[k][x],wid/2
    LINE2 topnod[k][x],wid,botnod[k][x],wid/2
    LINE2 topnod[k+1][x],0,botnod[k][x],wid/2
    LINE2 topnod[k+1][x],wid,botnod[k][x],wid/2
  NEXT k
  DEL 1
  NEXT n
ENDIF
```

!2D Fully scripted version

```
IF vu2d=vu3 THEN !Elevation in 2D
  GOSUB 1010:!--Hotspots Elevation view
```

!Top Deck

```
FOR k=1 TO numnod-1
  LINE2 topnod[k][x], topnod[k][z],
  topnod[k+1][x], topnod[k+1][z]
NEXT k
```

!Bottom Deck

```
FOR k=1 TO numnod-2
  LINE2 botnod[k][x], botnod[k][z],
  botnod[k+1][x], botnod[k+1][z]
NEXT k
```

!Diagonals

```
FOR k=1 TO numnod-1
  LINE2 topnod[k][x],topnod[k][z],
  botnod[k][x],botnod[k][z]
  LINE2 topnod[k+1][x],topnod[k+1][z],
  botnod[k][x],botnod[k][z]
NEXT k
ENDIF
```

END:=====

1000:!--Hotspots Plan view

```
!Plan Hotspots
FOR n=1 TO numf+1
  HOTSPOT2 ept, wid*(n-1)
  HOTSPOT2 spt, wid*(n-1)
  HOTSPOT2 mpt, wid*(n-1)
NEXT n
RETURN
```

1010:!--Hotspots for elevation view

```
FOR k=1 TO numnod
  hsid=hsid+1
  HOTSPOT2 0,topnod[k][z], hsid,topnod[k][x],1+128
  hsid=hsid+1
  HOTSPOT2 topnod[k][x],topnod[k][z], hsid, topnod[k][x],2
  hsid=hsid+1
  HOTSPOT2 -1,topnod[k][z], hsid,topnod[k][x],3
  hsid=hsid+1
  HOTSPOT2 topnod[k][x], 0, hsid,topnod[k][z],1+128
  hsid=hsid+1
  HOTSPOT2 topnod[k][x],topnod[k][z], hsid, topnod[k][z],2
  hsid=hsid+1
  HOTSPOT2 topnod[k][x],-1, hsid,topnod[k][z],3
NEXT k

FOR k=1 TO numnod-1
  hsid=hsid+1
  HOTSPOT2 0,botnod[k][z], hsid,botnod[k][x],1+128
  hsid=hsid+1
  HOTSPOT2 botnod[k][x],botnod[k][z], hsid,botnod[k][x],2
  hsid=hsid+1
  HOTSPOT2 -1,botnod[k][z], hsid,botnod[k][x],3
  hsid=hsid+1
  HOTSPOT2 botnod[k][x], 0, hsid, botnod[k][z],1+128
  hsid=hsid+1
  HOTSPOT2 botnod[k][x],botnod[k][z], hsid,botnod[k][z],2
  hsid=hsid+1
  HOTSPOT2 botnod[k][x],-1, hsid,botnod[k][z],3
NEXT k
RETURN
```

Tube structures from imported DXF files

SOME 3D structures may come to you as DWG or DXF, and if you are lucky with the format, there is a method of converting them to practical 3D GDL solid structures.

When you import the file into GDL, you have no say on how it will appear, but try it out and see what you get (always ask for editable GDL, not Binary). In the example used here, that of a Geodesic structure, the import filter produces a 3D script full of LIN_ statements. These produce 3D lines in space which look OK, but disappear in a photorender – because lines on their own are not solid. This exercise gives you a standard routine for taking any of these line based 3D structures and making them solid, using TUBE statements.

Formian

The examples used here come from 'Formian' which is a programme (Windows only) developed at the University of Surrey by Hoshyar Nooshin and his team in the Space Structures Research Centre. You can construct 3D structures with Formex Algebra which can then be saved as DXF files. I have also brought line based structures saved as DXF from Rhinoceros into GDL which produced the same line based 3D code.

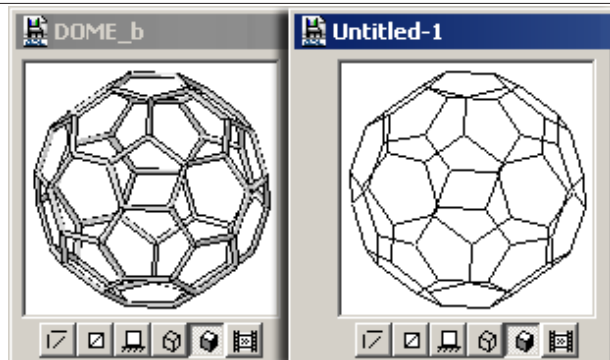
Use Google to look up 'Formian Surrey' and 'Formex Algebra' and you will discover more about it, and will be able to download and use Formian yourself (although it is a hundred times more difficult than GDL!) The demonstration files generate a good number of parametric towers, vaults, domes and scallop domes.

3D Script

When you have opened the DXF file as a GDL Object, look at the 3d Script. If you suspect that the dimensions of the lines are either 1000 times smaller or larger than they should be, bring it back in with a modification to the scale in the DXF Translator. It will look like the code below, hundreds of lines of LIN_ statements.

```
MODEL SURFACE
MUL a, b, ZZZZX
XFORM TR3D_1_1, TR3D_1_2, TR3D_1_3, 0.0,
      TR3D_2_1, TR3D_2_2, TR3D_2_3, 0.0,
      TR3D_3_1, TR3D_3_2, TR3D_3_3, 0.0
MATERIAL 0
LINE TYPE "Solid Line"
PEN 1
LIN_ 0.343279, 0, 0.939234, 0.106079, 0.326477, 0.939234
LIN_ 0.106079, 0.326477, 0.939234, -0.277718, 0.201774, 0.939234
LIN_ -0.277718, 0.201774, 0.939234, -0.277718, -0.201774, 0.939234
LIN_ -0.277718, -0.201774, 0.939234, 0.106079, -0.326477, 0.939234
LIN_ 0.106079, -0.326477, 0.939234, 0.343279, 0, 0.939234
LIN_ 0.343279, 0, 0.939234, 0.686557, 0, 0.727076
LIN_ 0.106079, 0.326477, 0.939234, 0.212158, 0.652955, 0.727076
LIN_ -0.277718, 0.201774, 0.939234, -0.555436, 0.403548, 0.727076
LIN_ -0.277718, -0.201774, 0.939234, -0.555436, -0.403548, 0.727076
etc, etc, etc, hundreds more lines like this.....
```

All these LIN_ statements draw lines from x1, y1, z1 to x2, y2, z2. All we need is to make Tubes that do the same thing!



We receive a DXF of a 3D model like the one on the right, and we want to finish up with the one on the left.

The original DXF file will look something like this – a long chunk of text containing something that appears to be Line statements.

LINE	LINE	LINE
8	8	8
Default	Default	Default
10	10	10
0.343279	0.106079	-0.277718
20	20	20
0.000000	0.326477	0.201774
30	30	30
0.939234	0.939234	0.939234
11	11	11
0.106079	-0.277718	-0.277718
21	21	21
0.326477	0.201774	-0.201774
31	31	31
0.939234	0.939234	0.939234
0	0	0

Variable	Type	Name	Value
A	X Dimension	1000	
B	Y Dimension	1000	
ZZYZX	Vertical scale	1000	
AC_show2DHot...	Show 2D Hotspots in 3D	On	
Derived_from_A...	Abc	Derived from AutoCAD	100
TR3D_1_1	TRAFO3D_1_1	1.00	
TR3D_1_2	TRAFO3D_1_2	0.00	
TR3D_1_3	TRAFO3D_1_3	0.00	
TR3D_2_1	TRAFO3D_2_1	0.00	
TR3D_2_2	TRAFO3D_2_2	1.00	
TR3D_2_3	TRAFO3D_2_3	0.00	
TR3D_3_1	TRAFO3D_3_1	0.00	
TR3D_3_2	TRAFO3D_3_2	0.00	
TR3D_3_3	TRAFO3D_3_3	1.00	

When you import the 3D file into GDL you will get a parameter table looking like this. We need to add some parameters to include details of tubing.

Parameters and Master Script

You will need to set up new parameters for the tubing detail, plus some popdown menus for choices, such as resolution of tubing and 2D representation. We could have a choice of section (round or rectangular) but this has been covered in other exercises in the GDL CB. You can also HIDE the transformation parameters, these can be retained, but we can set permanent values for them. You could also write some instructions for the object's user.

Variable	Type Name	Value
A	X Dimension	2000
B	Y Dimension	2000
ZZYZX	Vertical scale	2000
instruct	Abc Instructions	The structure ca...
mode3d	Abc 3D Mode of Model	Solid tubes
gs_cont_pen	Pen 3D contour	1
_sp1	3D Options	
diam	Tubing Diameter	50
tmatl	Tube Material	18
rsl	Tube Resolution 3-8	5
_sp0	2D Representation	
vu2d	Abc View in 2D	Line representati...
gs_symb_pen	Pen 2D Symbol	1
AC_show2DHot...	Show 2D Hotspots in 3D	Off
TR3D_1_1	Transform in X direction	0.50
TR3D_2_2	Transform in Y direction	0.50
TR3D_3_3	Transform in Z direction	0.50
TR3D_1_2	TRAFO3D_1_2	0.00
TR3D_1_3	TRAFO3D_1_3	0.00
TR3D_2_1	TRAFO3D_2_1	0.00
TR3D_2_3	TRAFO3D_2_3	0.00
TR3D_3_1	TRAFO3D_3_1	0.00
TR3D_3_2	TRAFO3D_3_2	0.00

```
!Dome imported from DXF - Master Script

!Resolution This limits the resolution to
rsl=MAX (3,rsl) between 3 and 8.
rsl=MIN (8,rsl)
PARAMETERS rsl=rsl

!Menus We retain the option to
mv1='Lines' display in line mode, in case
mv2='Solid tubes' the dome is very large and
would be slow to render.
VALUES 'mode3d' mv1,mv2

IF mode3d='Lines' THEN HIDEPARAMETER 'diam',
'rsl','tmatl','truvu','_sp1'

vv1='Line representation'
vv2='3D hidden line' The other TR3D parameters
vv3='3D Solid' are to do with skewing. If
VALUES 'vu2d' vv1,vv2,vv3 you wish to experiment with
skewing, leave the XFORM
statement in the 3D script.

!Parameters
trad=diam/2 !tube radius

!Transformation to make it the right size
PARAMETERS TR3D_1_1=0.5,
TR3D_2_2=0.5,TR3D_3_3=0.5
!Leave this line till last....
```

Tweaking the TR3D_1 parameters

The dome/structure comes from Formian in a specific size. GDL assigns a size of 1m x 1m x 1m to it as a default, but displays it at the imported size. Very confusing. We can make use of the XFORM command in the imported script to adjust the object to its correct size and to be able to use A, B & zzyzx correctly. XFORM is like an extended MUL statement. We can use either trial and error or pure logic to work out the multiplication factor – in this case, I used the reciprocal of the actual size as imported from DXF.

3D Script again – the trick!

The primary trick in this task is to use Global Search and Replace and change all the LIN_ statements to PUT statements. Once they are in memory, you can use a standardised routine which will convert all these numbers either to Lines or Tubes. Then use the script here. In future, use this as a template – you can bring any other objects in, grab that list of LIN_ statements, convert them to PUT, paste them into duplicate of this

```
!Parameter Script
VALUES 'instruct' 'The structure can be in',
'3D lines, or in 3D solid tubes.',
'Warning, use the line representation',
'in the 2D symbol while',
'setting up the object',
'as drawing time in 2D will be slow.',
'If you have the object in',
'Line mode in the 3D View,',
'it will disappear in renders.'
```

It is ALWAYS a good idea to include some instructions to the user, and what better place to put it than the Parameter Script. I tend to use the Master Script for any menus that result in flags like 'mv1' or 'vv1' in this case so that they are known to the 3D and 2D scripts.

!Dome imported from DXF - 3D Script

```
GOSUB 1000:!Hotspots

IF vu2d=vv1 AND GLOB_CONTEXT=2 THEN mode3d=mv1

MUL a, b, ZZZZX
MUL TR3D_1_1, TR3D_1_2, TR3D_1_3
PEN gs_cont_pen

PUT 0.343279, 0, 0.939234, 0.106079, 0.326477, 0.939234
PUT 0.106079, 0.326477, 0.939234, -0.277718, 0.201774, 0.939234
PUT -0.277718, 0.201774, 0.939234, -0.277718, -0.201774, 0.939234
PUT -0.277718, -0.201774, 0.939234, 0.106079, -0.326477, 0.939234
PUT 0.106079, -0.326477, 0.939234, 0.343279, 0, 0.939234
PUT 0.343279, 0, 0.939234, 0.686557, 0, 0.727076
PUT 0.106079, 0.326477, 0.939234, 0.212158, 0.652955, 0.727076
PUT -0.277718, 0.201774, 0.939234, -0.555436, 0.403548, 0.727076
PUT -0.277718, -0.201774, 0.939234, -0.555436, -0.403548, 0.727076
PUT 0.106079, -0.326477, 0.939234, 0.212158, -0.652955, 0.727076
!etc etc etc, hundreds more lines like these
```

!Standard routine to use with all such DXFs

```
IF mode3d=mv1 THEN Do the hotspots before the rest of
WHILE NSP DO the object.
LIN_GET(6) This IF-statement enables us to get
ENDWHILE the right appearance in the 2D
ENDIF symbol. The imported object will
have a 2D Symbol, but if you
represent the 3D in Lines mode, it's
acceptable to use a PROJECT2.
Leave the XFORM statement in
place – or change it to a MUL
statement. This tweaks the object so
that the A,B,zzyzx parameters are
correct.

!---- Tubes -----
IF mode3d=mv2 THEN
MATERIAL tmatl
RESOL rsl
WHILE NSP DO
x1=GET(1)
y1=GET(1)
z1=GET(1)
x2=GET(1)
y2=GET(1)
z2=GET(1)

x0=x1+(x1-x2)
x3=x2-(x1-x2)
y0=y1+(y1-y2)
y3=y2-(y1-y2)
z0=z1+(z1-z2)
z3=z2-(z1-z2)

TUBE 2,4,63,
0,0,901,
trad,360,4001,
x0,y0,z0,0,
x1,y1,z1,0,
x2,y2,z2,0,
x3,y3,z3,0
ENDWHILE
ENDIF
```

For the Tubes, we get the data in bunches of 6 at a time using a WHILE statement. The x0 and x3 routines here work out the theoretical phantom points at the start and end of each tube – it's a standard routine which you can use for all tube structures, giving you a nice butt ended cut on each tube.

In another exercise we can look at ways to give such tubes conical ends.

The Hotspots routine is visible on the example object on the Cookbook CD.

```
END: !=====
```

1000: !Hotspots

```
HOTSPOT 0,0,0
!the remaining hotspots are standard Graphical Hotspot
!routines that you can get in more detail from the
!3D Script of the object on the Cookbook CD.
RETURN
```

object using this 3D script – apply the size adjusting tweak in the Master Script and you have a new 3D structure!

2D Script: See next page.

Print out a parameter list

LASZLO NAGY of Graphisoft posted on ArchiCAD-Talk this very useful method of printing out a list of parameters. The only thing it does not do for the GDL writer is to show the internal parameter names, but for those who wish to list out the details of objects, it's perfect.

The method I have used throughout the GDL Cookbook is to do a screen capture and then use Photoshop to tidy it up – but if you need a text version of the table this is a good method.

It uses a little known capability of ArchiCAD to make an SQL query on its database. To get this to work you use Calculate\SQL\Query. Make sure SQL is enabled from the lowermost drop-down list in 'Options\ Preferences\ Imaging and Calculation', so it appears in the Calculate menu. This example is taken from the Dome in the page on converting from DXF files. Type the following query:

Query

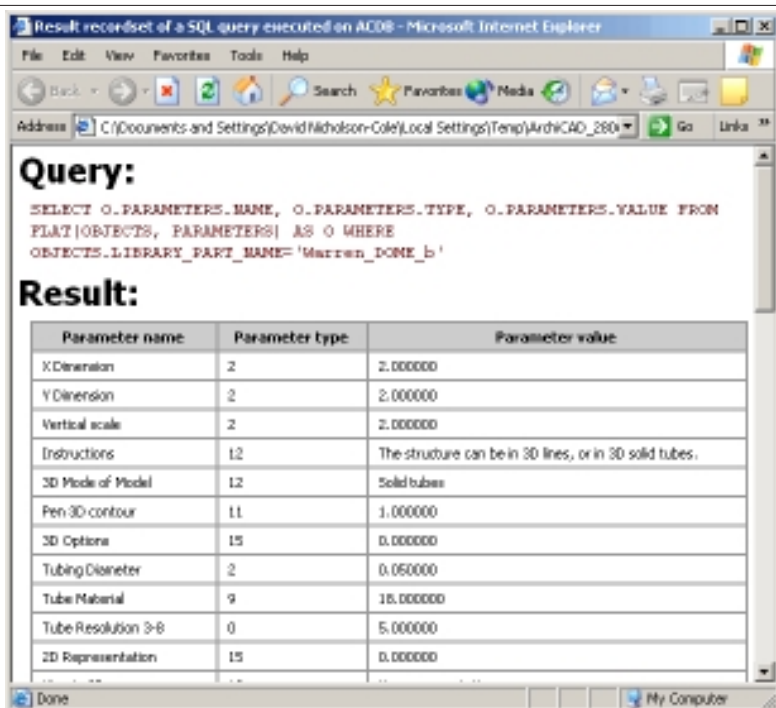
```
SELECT O.PARAMETERS.NAME, O.PARAMETERS.TYPE,
O.PARAMETERS.VALUE FROM FLAT(OBJECTS, PARAMETERS) AS O WHERE
O WHERE OBJECTS.LIBRARY_PART_NAME='DOME_b'
```

Result

Parameter name	Parameter type	Parameter value
X Dimension	2	2.000000
Y Dimension	2	2.000000
Vertical scale	2	2.000000
Instructions	12	The structure can...
3D Mode of Model	12	Solid tubes
Pen 3D contour	11	1.000000
3D Options	15	0.000000
Tubing Diameter	2	0.050000
Tube Material	9	18.000000
Tube Resolution 3-8	0	5.000000
2D Representation	15	0.000000
View in 2D	12	Line representation
Pen 2D Symbol	11	1.000000
Show 2D Hotspots in 3D	13	0.000000
Transform in X direction	4	0.500000
Transform in Y direction	4	0.500000
Transform in Z direction	4	0.500000
TRAF03D_1_2	4	0.000000
TRAF03D_1_3	4	0.000000
TRAF03D_2_1	4	0.000000
TRAF03D_2_3	4	0.000000
TRAF03D_3_1	4	0.000000
TRAF03D_3_2	4	0.000000

The result comes up as a Table in Explorer, and this can be pasted directly into a word processor like Word.

Make sure you have actually placed at least one example of the object into the floor plan of the ArchiCAD file. Only place one example of the object if you want a single list; if you have 3 you will have three listings. You must use single quotes, not double, and it is Case Sensitive.



I don't know the SQL lingo, so I am not in a position to explain in detail how it works – but it does!

Tube structures from imported DXF files

(continued from previous page)

2D Script

We can just write some quick PROJECT2 statements here – which would allow you to develop the object further e.g. including rotation.

!Tube structure from DXF: 2D Script

PEN gs_symb_pen

GOSUB 1000: !Hotspots

IF vu2d=vv1 THEN PROJECT2 3,270,1

IF vu2d=vv2 THEN PROJECT2 3,270,2

IF vu2d=vv3 THEN PROJECT2 3,270,3

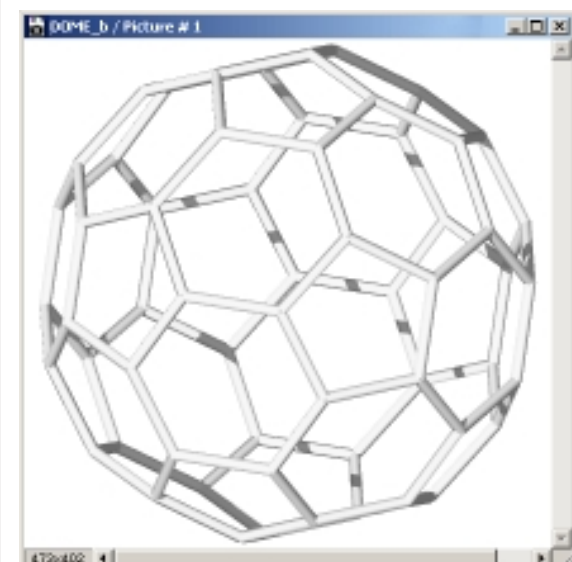
END: !=====

1000: !Hotspots

HOTSPOT2 0,0

!the remaining hotspots are standard Graphical Hotspot
!routines that you can get in more detail from the
!2D Script of the object on the Cookbook CD.

RETURN



Cylindrical Mapping

MOST of the textures we have used so far have been directionally oriented on what are primarily cubic volumes – i.e. lengths or panels of timber. What happens when you want to wrap the texture round a cylinder?

I discovered that cylindrical textures can be applied, but the method defies the normal logic you would expect to apply, and I had to solve it with trial and error until a system emerged. Here it is!

The trick is twofold: to get the Texture definition right, and then to get the COOR and VERT routine working correctly.

There are a number of other useful learning outcomes – defining texture, masking codes etc.

Parameters and Master Script

We can use the A, B, zzyzx dimensions, plus ones for top material and the Logo to be printed on the side of the can. We can also have a hidden parameter for the side material and set the *AC_show2DHotspotsIn3D* parameter to be hidden and ON. The Texture should be defined in its width and height. Width should be $2 \times \pi \times \text{radius}$, but it doesn't work!! Trial and error shows me that setting the width to $\langle \text{twice } \pi \rangle$ on its own gives a reliable index, and if you multiply this by the reciprocal of the number of logos, it works!!

```
!Cylindrical box with logo
!Master Script
PARAMETERS numlog=MAX(1,numlog)
DEFINE TEXTURE "side" logo,
  PI*2*(1/numlog), zzyzx, 1, 0
DEFINE MATERIAL "sidem" 22,
  1, 1, 1,
  0, 61,
  IND(TEXTURE, "side")
PARAMETERS sidemat=IND(MATERIAL, 'sidem')
PARAMETERS AC_show2DHotspotsIn3D=1
```

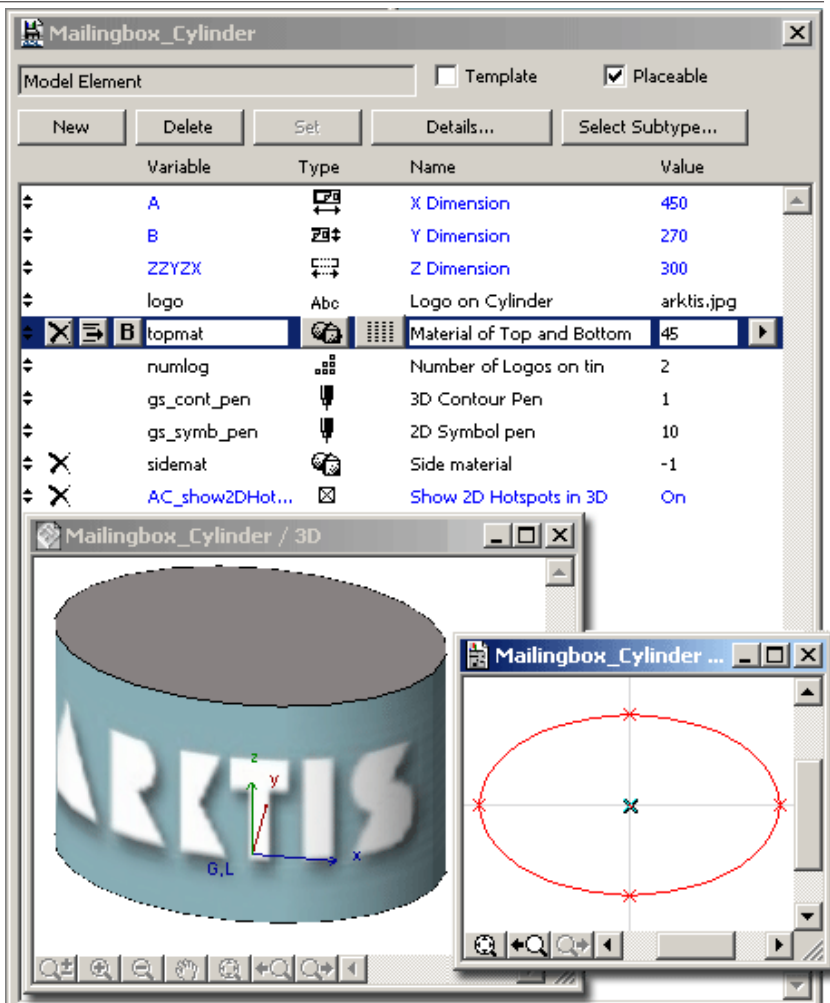
This routine allows you to have 1 or more of repetitions of the 2D jpeg logo.

When you define a texture, convert it to an Index, and make it equal to a hidden material parameter.

It will export to Artlantis more successfully.

```
!2D Script
PEN gs_symb_pen
HOTSPOT2 0,0
HOTSPOT2 A/2,0
HOTSPOT2 -A/2,0
HOTSPOT2 0,B/2
HOTSPOT2 0,-B/2
MUL2 1,B/A
CIRCLE2 0,0,A/2
DEL 1
```

By setting the AC_show2DHotspotsIn3D parameter to ON, we don't need to repeat the ground plane hotspots in the 3D script, only the ones for zzyzx.



```
!Cylindrical box with logo
!3D Script
```

```
TOLER 0.002
PEN gs_cont_pen
GOSUB 1000:!Hotspots
MUL 1,B/A,1
CPRISM_ topmat,topmat,sidemat,
  2,zzyzx,
  0,0,900+13+64,
  A/2,360,4000+13
```

```
!Texture routine
BASE
VERT 0,0,0
VERT 1,0,0
VERT 0,1,0
VERT 0,0,1
COOR 3,-1,-2,-3,-4
BODY -1
DEL 1
```

```
END: !=====
```

```
1000:!Hotspots for upper edge of cyl.
HOTSPOT 0,0,zzyzx
HOTSPOT A/2,0,zzyzx
HOTSPOT -A/2,0,zzyzx
HOTSPOT 0,B/2,zzyzx
HOTSPOT 0,-B/2,zzyzx
RETURN
```

We try to state the hotspots first: it's tider to put them in a subroutine.

Use the MUL to distort the PRISM into an ellipse. The fundamental radius is A/2, the B dimension is obtained with MUL. The masking codes removes unwanted vertical lines and ensure that the curve will look smooth.

The COOR and VERT routine uses value THREE to ensure cylindrical mapping around the centre of the cylinder.

3D Script

This is simply a Cylinder, using the Prism command. You could elaborate this further, to make it hollow, with a parametric lid. By using MUL and some hotspots, we can make the cylinder stretchily elliptical.

Advanced 2D

FRAGMENT2

FRAGMENT2 is a system of layering in the 2D environment of GDL. The little buttons in the Parameter Table allow you to toggle the visibility of as many as 16 of these 'layers'. With all these buttons on, you see all the fragment layers. If you use the power of scripting, you can decide which fragment will be displayed in the 2D View.



All 2D lines, text, circles etc that you would normally draw in the 2D Symbol window are put into the top drawing layer, called Fragment 1. Open the 2D symbol window, click on any of the lines or other 2D elements and see which fragments they are in – the Info box tells you. Now you can use the popdown fragment selector to move it to a different fragment such as 2 (just as, in 3D, you could move a wall object from 'External walls' to 'Site and landscaping' layer).

If you are using a complicated symbol, you might have to repeat this process for all Text, Circle and Fill objects. Put them all into Fragment no. 2.

Once the whole symbol is in Fragment no. 2, you could draw another symbol into Fragment 1, and then move it by the same process to Fragment 3. In this way, you could build up a whole series of quite different drawings or images in each fragment.

You could draw directly in a fragment, but have to 'choose' that as the drawing fragment first. It's advisable to leave fragment 1 as the one that is always the default fragment for new 2D drawing elements.

Turn Fragments on and off

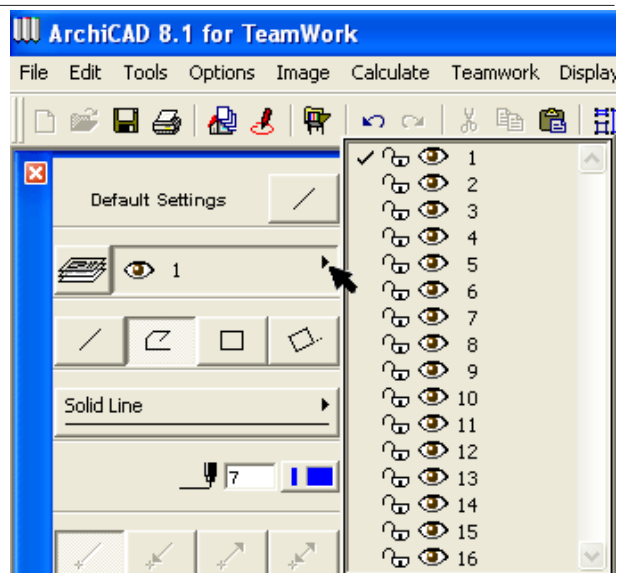
In the Library part dialog box, all fragments are normally displayed, using a series of dark or light numbered buttons. Click on the buttons, and you find that you can select all, or one only, or any group of fragments to be displayed.

Normally when a 2D symbol is drawn, it is immediately disabled when there is an active 2D script. However, the two work together if the FRAGMENT2 command is scripted.

Syntax:

FRAGMENT2 fragment_number, use_attributes

- If the attribute flag is 0, the 2D fragment appears with the pen colours and line types etc as it was drawn.
- If the attribute flag is 1, the 2D fragment appears with the pen colours and line types etc as specified in the script or settings box.



Info box displays the Fragment of lines you click on and with the popdown, you can change their fragment.

FRAGMENT2 in action

For a very simple example, suppose you have a non stretchy object that is intended for use in 3D models, but also in Plans as a 2D object, and in Sections – a Sanitary fitting for example. A 2D symbol might be chosen with a number, or with a ValueList, and your fragments might be organised thus:

1=Plan 3D true view; 2=Plan 2D line; 3=Side Elevation 2D line; 4=Front Elevation 2D line.

Procedure

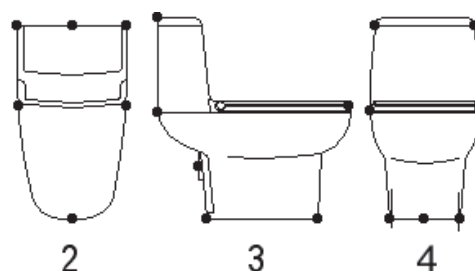
To make your own 2D symbols from the 3D, here is a way. Place the object in the Floor Plan. View it in Plan, or in Section/Elevation, select it, then Explode it. 'Cut' the resulting lines, and paste them into the 2D Symbol window of the GDL object. The pasted result will be put into Fragment 1.

Open the 2D Symbol window, and select all lines in the symbol – move them all to a different Fragment from 1. Put it in 4 if it's the front elevation. Fragment 1 should now be empty. So now do the same for view 3 and so on, down to 2. Leave 1 empty (for further use).

In the 2D script, include a routine like this:

```
IF cstyl=1 THEN PROJECT2 3,270,2
IF cstyl=2 THEN FRAGMENT2 2,1
IF cstyl=3 THEN FRAGMENT2 3,1
IF cstyl=4 THEN FRAGMENT2 4,1
```

The 1 after the command is an attributes flag. Leave it as 1.



Most sanitary products are provided by manufacturers in DXF or DWG. In GDL, you can combine 3 such files into one by copying and pasting each image to the 2D Symbol window. Use a ValueList to choose which to display.

DXF conversion

Convert manufacturers' products

GDL has a big future with ArchiCAD enthusiasts, but that is nothing compared with the status it will achieve if manufacturers of building products recognise it as a suitable language for 3D form – and start offering architects CDs of their product libraries in BOTH DXF and GDL. If ArchiCAD users and AutoCAD users can make equal use of the parametric powers of the GDL products, DXF may become a distant memory, a bit like CPM and MSDOS.

DXF is just so inconvenient to use compared with GDL. A GDL smart object, such as a sanitary fitting can contain the 2D plan, the 2D front elevation, the 2D side elevation and the 3D form, and a 3D wireline form all within the same object; in addition it can contain manufacturer's data on serial numbers, weight, cost, volume, maintenance info and more; database integration that can transform the way we design, build and manage buildings. In fact, if the 3D model isn't too voluminous (and if they come from DXF they usually are!), you can get a whole range of manufacturer's products inside one single smart GDL object.

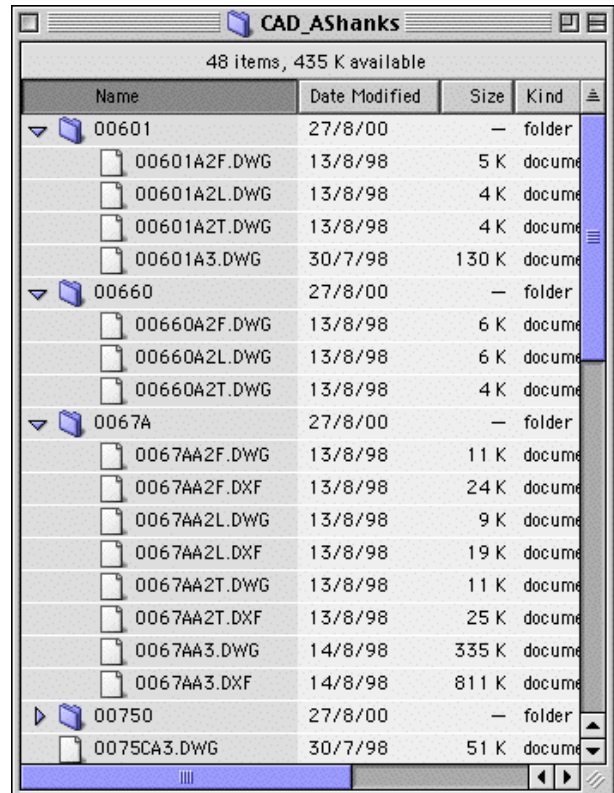
GDL objects are so compact that they can be published on the web, downloaded in seconds, and working in the 3D model or the 2D drawings in a few mouse clicks! Graphisoft have provided the technology needed for potential users to view and test GDL objects before they download. Manufacturers can now offer live libraries on their websites – instead of showing pretty pictures, they can show working models of their products and provide every inducement for designers to specify and instal those same products.

If all goes well, there should be a lot of work for years to come converting DXF drawings of manufactured products to GDL and writing products yet to come in native GDL. Let's work out a typical procedure for doing this economically. Remember that with dozens or perhaps hundreds of products to convert, there's no point in over-elaborating. Just get the job done efficiently. The elaboration will come when we have the leisure time to review converted objects and convert the 3D part of the GDL to native GDL.

There's is a whole manual on the subject of DXF/DWG conversion so there is no intention to reiterate all the details – we will stick strictly to object making.

DXF/DWG files are usually stored separately. A typical object, say a water closet, is represented by at least 4 files: a 3D model, a 2D plan, a 2D front elevation, and a 2D side elevation.

Before we proceed, ask yourself: Do I know enough about the use of FRAGMENT2? If you do not, then mug up FRAGMENT2 quickly, it is vital to the success of this conversion.



DXF files are almost three times the size of the equivalent DWG file. When brought into GDL, there is some reduction, but the example featured here still finishes up as a GDL object of about 380k – unless you choose to rebuild the 3D model entirely, in which case it will reduce to a tiny fraction of the original size.

Let's do a typical file conversion

1. Bring the file into ArchiCAD

DXFs brought into ArchiCAD itself always arrive as 2D files. As your intention is to make library parts, you also have a option to open them as a library part – click this option and bring them in. Then you can copy and paste the contents of 4 such files into one single file with a bit of smart scripting to separate them.

There are issues of scale – it is not uncommon for a brought-in file to be a thousand times too big or too small – depends on the scale of the original drawing. These objects (from Armitage Shanks) all come in 1000 times too big unless you tell GDL to treat 1 Autocad unit as 1.0. It's alarming to imagine a WC that is almost a kilometre long – but thats what you could get!

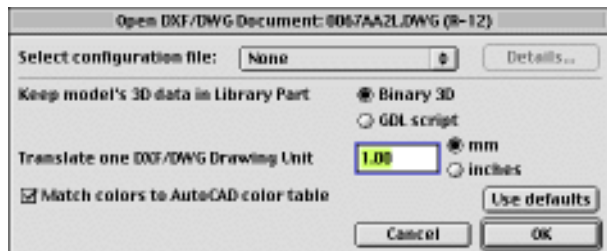
If you wish to bring in a 3D DXF/DWG, you could first open the 3D file as a binary library part, and add the 2D library part symbols to it. But this will be 1.5 times the file size of bringing it in as editable script, and you would only be able to have that one object in the file. If perchance you want your GDL to contain several objects (selected by a value list) then its best to bring them all in as editable GDL and write some

little subroutine structure to organise them.

If you want to bring in dozens or more objects (perhaps an entire library of sanitary fittings), most of them will be on the manufacturer's CD in a baffling jumble of serial or type numbers. I advise to bring them •all• into the ArchiCAD floorplan so you can survey the entire range of symbols to be converted, group them, organise them, decide which one will go together and so on; having thus organised, you may actually do the process by bringing them in as library objects.

2. Organising the files – move fragments

OK, so you may have 4 GDL files open at once! Save

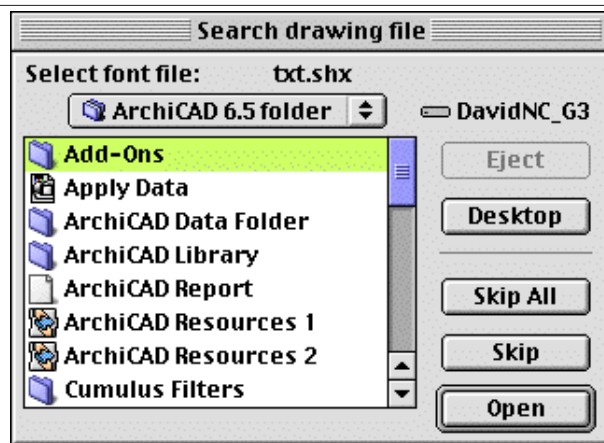


The default conversion may be 1000. Experiment. If your objects come in several kilometres wide, throw them away and reimport with a smaller conversion.

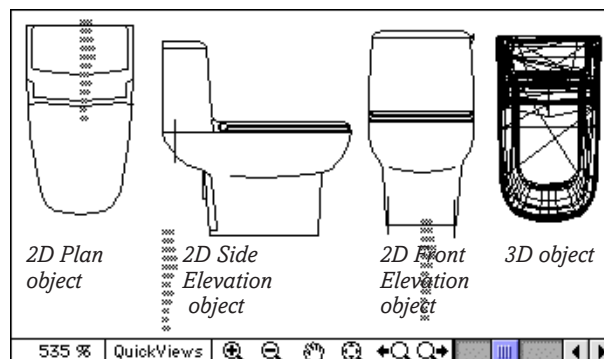
them all to a library, then drop all four of them into a floor plan window. One more file, a fifth one, is the one in this exercise – open this later as a template for building your objects.

The 2D objects (and the 2D symbol of the 3D object) will all be in Fragment number 1, so the task is to copy and paste the symbols into one GDL object without getting the fragments mixed up. This is a most vexing process!! It is explained here.

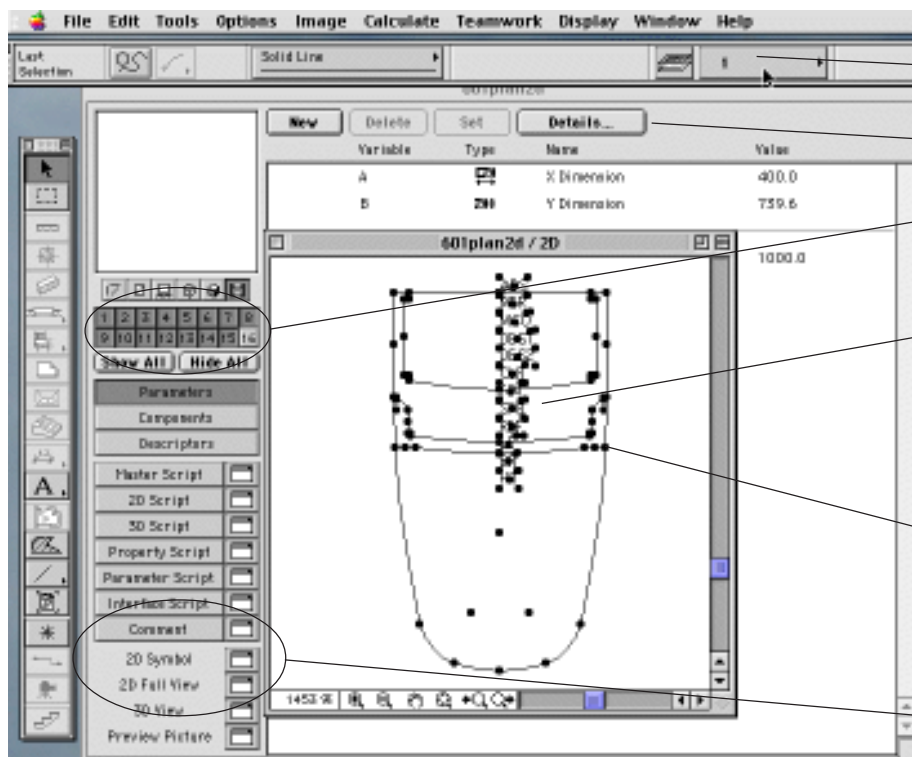
a. Open the GDL object with the 2D Plan. Open the 2D symbol window. We're going to move the plan to



You will be asked to find a font file – you can SKIP it



fragment 16 as temporary parking space. On the fragment button bar, change 16 to Hidden. Now select the whole plan view and move it to fragment 16, using the popdown layer/fragment selector in the Info palette. First the Lines disappear. Do it again. Now the Arcs disappear. Do it again. Now the Text disappears. If nothing else shows, the whole symbol is now in fragment 16.



Click and select here to push the 2D elements to the correct fragment

Hit the Details button to hide the Bounding Box hotspots

These buttons Hide or Display the Fragments, dark=visible, light=hidden

None of these text bits are of interest so they can be deleted. Edit out any bit that you don't want, and use 2D tools to tidy up as appropriate

If you first hide fragment 16, you will see Lines, then Arcs then Text 'disappear' until the window is blank. Now you have space in fragment 1 to paste in the next 2D image

The 2D symbol window actually contains 16 separate layers called Fragments. Use them here!

b. Open the symbol window of the Side Elevation object. Copy (or Cut) the whole symbol; return to the Plan object, open the 2D symbol window and Paste. It will be dumped into fragment 1. You are going to move the drawing to fragment 2. Hide fragment 2 in the fragment button bar, then move the whole fragment from 1 to 2 using the method described above (of repeatedly shifting each set of 2D elements). Close the Side Elevation object.

c. Open the symbol window of the Front Elevation object. Do it all again (now you have got the hang of it!) and put it in fragment 3 of the Plan object. Close the Front Elevation object.

d. Open the 3D script of the 3D object, Copy the whole script, return to the Plan object and Paste the whole 3D script into the 3D script window. Close the 3D object.

e. Return to the Plan object, show fragment 16, and hide fragment 1, 2 and 3. Move the 2D plan from fragment 16 to fragment 1. Go on moving elements until 16 is quite empty. Save the file under a new name,

based on the objects description and number. Put it into a loaded library, one that you have created for the storage of all your newly converted files.

f. Now, leaving the other two hidden, take each fragment in turn, tidy up the image, filling in missing lines, removing bad bits of text, and positioning the image correctly over the origin. Add Hotspots to each image, ensuring beforehand that you double click the hotspot tool and tell it precisely which fragment it is going into. Hotspots should be in significant corners or axes, with one in the middle for picking up. Click on the Details button in the objects title bar, and untick the Bounding Box hotspots. Set all line colours to a single colour, black will do (you can change it later from the settings box).

Learn this technique!

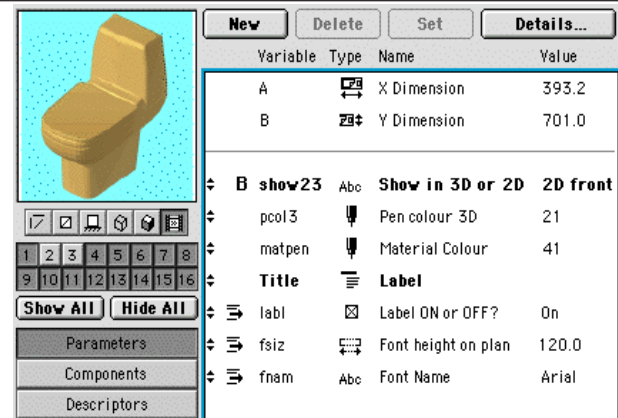
You can have fun with this technique – you can freely download DXF models of the Eiffel Tower or Chrysler building and a host of other wonderful structures off the Internet and convert them to GDL.

3. Convert to a Smart Object

Build a list of parameters for the new object. To make your life easy, use the same variable names on every occasion. You may need to add more if the objects you are making require additional definitions of material or if you are including more than one object in the same file.

Now open the Conversion Template object (which we are building now), copy the significant parts of the Master script, the 2D script, the 3D script, the Parameter script and the Property script. Paste them, one at a time into the script windows of the WC Object. For the 3D bit, paste the template text ABOVE the 3D description of the object.

The side and front elevations are provided for use in sections and elevations, into which you can place 2D objects. A 2D version of the plan can be shown in the 3D drawing if the user does not require a fully solid model. See the 3D script.



In this case, the font size is NOT a plotting size, it remains constant to a real dimension, whatever the scale

Because label is an option, the fine details are hidden inside a cascading set



```
!DXF/DWG Conversion Template
!Master script

VALUES 'show23' '3D model',
      '2D plan','2D side view','2D front'

VALUES 'fnam' 'Geneva','Arial','Times',
      'Verdana','Helvetica','Georgia','Courier'

IF show23= '3D model'      THEN dt=0
IF show23= '2D plan'      THEN dt=20
IF show23= '2D side view' THEN dt=21
IF show23= '2D front'     THEN dt=22

x=REQUEST("Rgb_of_pen",matpen,pred,pgreen,pblue)
DEFINE MATERIAL "Body_Shiny" 4,
                pred,pgreen,pblue
bodymat=IND (MATERIAL,'Body_shiny')

desc_str1='Armitage Shanks WC 601'!for listing
desc_str2='WC 601'                !for 2D symbol

DESCRIPTOR desc_str1
```

Value lists can be in the Parameter script or the Master script. Put them first.

Parse the Value list and convert to flags

Extract a colour. If your manufactured objects have special ranges of colours (say three colours for a lamp, black, white and dark green) you might have to build those materials with Define, and make a Value list to select them. In this example, the Pen colour can be analysed into Red, Green and Blue, and converted into a colour. 'Plastic' usually gives the best results for home-made materials.

Text descriptions, insert one to appear in a listing, and another as an optional label in the 2D symbol.

For the moment this is the minimum you need for the Property script, unless your client wants to go overboard with greater levels of detail.

2D Script

Standard title is pasted in at the top. Insert the title of the particular object you are making.

Pen can be set in the user's settings box, and write the Key hotspot

If the user wants a label, you need to define style. In this case, font size is autosizing, even if the user changes scale. At smaller than 1/100 scale, the label disappears. The label is on two lines, you may want to adapt this to your own needs.

If the object is in 2D in Plan or 3D then it will use Fragment 1. The zero flag allows the user to set the colour from the main object settings box. Use the Project2 to nudge the 3D object to fit the 2D plan symbol.

```
!DXF/DWG Conversion Template
!3D script

!Open Each file first as an object. Unit is 1 millimetre.
!Then open the 2D objects one at a time and drag 2D into
!fragments 1, 2 and 3 of the 2D Plan object. Edit the
!2D images and ensure that they fit. Copy 3D
!script into 3D script of 2D Plan object.
!Resave 2D plan object as final new object.
!Build Parameter table and value lists for new object.
!Copy over Master, 2D and 3D script bits from the
!Conversion Template object. Nudge 3D to fit.
!Untick the box for Bounding Hotspots, plant your own
!Test Object - tune object.

PEN    pcol3
MATERIAL bodymat

ADD -0.393/2,-0.700,0 !nudge into position
IF dt=0 THEN GOSUB 100 ELSE GOSUB 200
DEL TOP

END!-----

100:!3D solid model
PLANE_3,
    0.299818, 0.035215, 0.452023, 0,
    0.320588, 0.045317, 0.452023, 0,
    0.160630, 0.020968, 0.452033, 0
PLANE_3,
    0.299818, 0.035215, 0.452023, 0,
    0.160630, 0.020968, 0.452033, 0,
    0.195927, 0.020968, 0.452035, 1
PLANE_3,
    0.012636, 0.351726, 0.452027, 0,
    0.019182, 0.167628, 0.452024, 0,
    0.384218, 0.443323, 0.452026, 0
PLANE_3,
    0.039468, 0.621380, 0.731770, 0,
    0.048015, 0.560041, 0.746757, 0,
    0.045833, 0.621251, 0.745732, 1
!..... and so on for thousands of lines until it
!      finishes describing every damned polygon!!
RETURN

200:!2D view of plan
PEN    pcol3
LIN_   0.194345, 0.700254, 0, 0.371504, 0.700254, 0
MODEL WIRE
ADDX   0.371504
ADDY   0.680254
ARC    0.020000, -1.832342, 90.000000
DEL    2
LIN_   0.391493, 0.679614, 0, 0.386457, 0.522175, 0
LIN_   0.386457, 0.522175, 0, 0.368389, 0.511478, 0
LIN_   0.368389, 0.511478, 0, 0.365417, 0.466724, 0
ADDX   0.355439
ADDY   0.467387
ARC    0.010000, -86.968877, -3.799346
DEL    2
!.... and so on until every line is included
RETURN
```

```
!DXF/DWG Conversion Template
!2D Script

!Armitage Shanks
!WC range

PEN    L_
HOTSPOT2 0,0

IF lab1 THEN
IF A_<101 THEN
DEFINE STYLE 'lab1' fname,
    fsiz*1000/A_,2,0
SET STYLE 'lab1'
TEXT2 0,0,'ArmSh'
TEXT2 0,-fsiz*1.1,desc_str2
ENDIF
ENDIF

IF dt=0 OR dt=20 THEN !Plan
FRAGMENT2 1,0
ENDIF

IF dt=21 THEN !Elevation Sideview
FRAGMENT2 2,0
ENDIF

IF dt=22 THEN !Elevation Front
FRAGMENT2 3,0
ENDIF

!!!project2 3,270,2
```

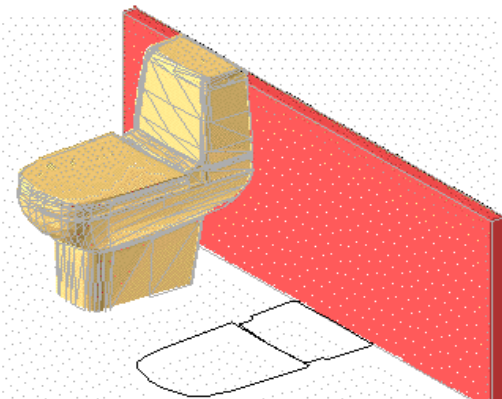
3D Script

As you may not have your Cookbook to hand all the time (shame!) then include a summary of the instructions at the top of the script. You may need them again.

Set the Pen, set the material (that you built in the Master script)

The ADD command is an offset to make the 3D fit over the 2D. Use Project2 in the 2D. The offset here is actually *half the width* of the WC and *the full length*.

As the DXF for this WC kindly provided a 2D version of the WC that would appear in 3D if the solid object was not required, it has been separated into another subroutine 200. If it hadn't provided one, you could place the 2D plan symbol over the origin, explode it, drag into the 2D script, form it into a subroutine and you have made your own 2D image for use in the 3D drawing. Since the 3D object could have thousands of polygons, it's wise to allow the designer to place 2D objects in the 3D environment and only show them as solid when 3D renderings are required.



Billboard Objects – PhotoReality

PHOTOGRAPHY can be more powerful than 3D modelling! There are times when you recognise that an object may be too complex to do, even in GDL – either it will take too long, or will just look too ‘wooden’ for the purpose. People and trees are prime examples of this problem. It’s easier to attach a good photograph to a surface to simulate 3D reality than to build the entire object. Some artists advocate adding the people and trees and backgrounds with Photoshop later. This argument may apply for static photoreal images, but for flythroughs, and for timesaving on static views, there is a good case to use **Billboard objects**, surfaces with phototextures in 3D views.

There are several ways to do these, and several pitfalls to get over. Third party Add-ons like ArchiPaint and ArchiFacade will make billboard objects for you, and the trees in ArchiTerra are also prime demonstrations of the art – the tree objects are accurate standup cutouts of the tree with a texture map matching precisely to the cutout shape.

PICTURE and Cutout Objects

There are two types of object in this category: **Picture based billboards** which use the Alpha channel in the colour image on a rectangular shape; and **Cutout based billboards** which have a texture applied to them, fitting the cutout accurately.

Open GL has made a big difference to the status of billboard objects – previously Picture based ones spent most of their life looking like large grey rectangles until photorendered, and Cutouts just looked like... grey cutouts. Now, they look good most of the time! It is possible with advanced GDL to animate the cutouts and the phototexture on them to simulate motion, RPC style.

Piranesi and Artlantis (third party renderers for ArchiCAD) work well with cutouts, or you can use their own library of billboard objects for people and trees. But they are not *your* people and *your* signs, and you may want to do your renderings in ArchiCAD. The new Lightworks renderer will encourage people to use ArchiCAD for renderings and save you the worry about the wrinkles of exporting to third party renderers.

Anybody who has played computer games knows that the urban environments are mostly very simple in 3D but are made complex by having photographs mapped onto simple block shapes – and trees are usually planar cutout objects.

Existing Billboard objects

There are two objects in the existing ArchiCAD Library, the Carpet and the Picture. The Carpet is a horizontal stretchy rectangle, and the Picture is a vertical stretchy rectangle with a fixed back panel, and options for a picture frame. We can write variants of these which work well.

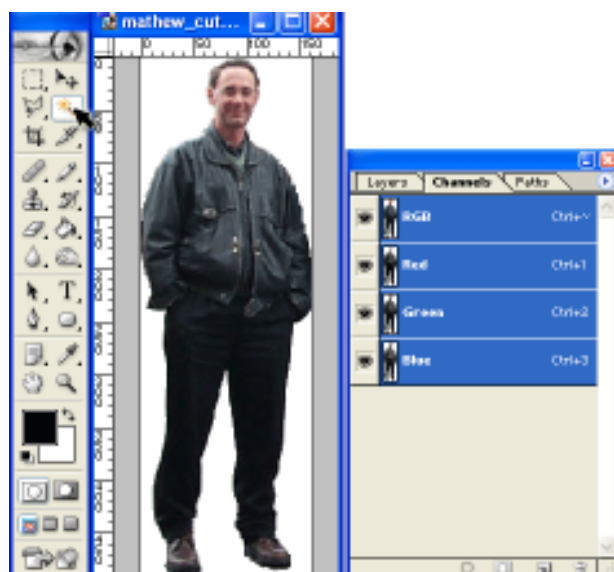


Make your own Billboards using the Alpha Channel with PICTURE

Billboard objects using the PICTURE command are actually GDL rectangles with a picture mapped on to them – for example the ‘carpet’ and the ‘picture’ in the ArchiCAD Library. They become ‘outlined’ objects by virtue of the alpha channels in the bitmap file and only show correctly in photorenderings. In 3D views they remain as rectangles, and cast rectangular shadows. You do not need to make a Material and Texture definition – but you require a quick tutorial in Adobe Photoshop.

A quick run around Photoshop

Adobe Photoshop is an incredibly valuable ‘companion’ to ArchiCAD users, and if you do not have it, see if the other bitmap editing software you have can support channels. Alpha channels are a part of 32 bit colour – whereby 24 bits are taken up with three visible channels of Red, Green and Blue (8 bits each). The remaining 8 bits are available for ‘interesting things’ like transparency, special reflection effects, bump mapping etc. and we call this the Alpha Channel.



We use an image of Matthew Lohden, a prime contributor to ArchiCAD-Talk and to ArchiCAD University events.



Bitmap files with an alpha channel must be rich in data, which means saving them as a .TIF format without compression. If you try to save one of these as a .JPG, the first thing that is omitted to achieve compression is the alpha channel. The PICTURE command in GDL allows you to use alpha channels for transparency. The picture you wish to use must be saved in a loaded library (and then reload).

Let's make the alpha channel. In Photoshop, surround Matthew carefully with the polygon-lasso. From the Select menu, Inverse, and delete (the background will be erased). Re-Crop the picture closely to the edges of the image, but do this: leave at least *ONE* white pixel all around the edges.

Now use the Magic wand to pick up the white background. The whole white area should be shimmering.

Now hit the Select menu>Inverse, and the marquee will now be surrounding the image. Now Save that selection from the Select menu – call it 'alpha' – the job is done, the alpha channel is now created. You can see it in the illustration, added to the list of channels, under the Red, Green and Blue. Save the bitmap file as a .TIF, into a loaded library and reload.

It's time to make the Picture billboard object

If you simply apply a ROTx 90 in the 3D script of a copy of the 'Carpet.gsm' and replace the 2D with LINE2 0,0,A,0, you have a splendid upright billboard object. Better still, you can make a better one with this nice clean script.

```
!!Billboard.gsm
!!2D Script
PEN cont_pen
HOTSPOT2 0,0
HOTSPOT2 A,0
HOTSPOT2 A/2,0
CIRCLE2 0,0,A/100
CIRCLE2 A,0,A/100
CIRCLE2 A/2,0,A/100
LINE2 0,0,A,0 !AC8 and earlier
!AC9 - add HOTLINE2 0,0,A,0
```



Billboard parameters, 3D view and 2D symbol.

The 3D Script is almost as short!

```
!!Billboard.gsm
!!3D Script
PEN cont_pen
IF gs_shadow THEN SHADOW ON
MATERIAL matl
ROTx 90
PICTURE picture_file,A,zzyzx,mask
DEL 1
HOTSPOT 0,0,0,1
HOTSPOT A,0,0,2
HOTSPOT A,0,zzyzx,3
HOTSPOT 0,0,zzyzx,4
HOTSPOT A/2,0,zzyzx/2,5
```

This is an ultra simple version of a Picture billboard object, and it's possible to build in smarter features – for example the ability to turn to face the cameras. See the GDL Cookbook for a more extensive discussion of Billboard objects.

Make your own Billboards as accurate Cutouts

You can make your own cutout shapes easily with the Slab tool, and if you also make a new Material with a Texture which is the correct size, you can make the two fit each other. You make the material in ArchiCAD's Options menu.

First, using Photoshop or a similar editor, crop carefully around the image, leaving only one pixel at most around the person, tree or car. You do NOT need to remove the background (when you make the cutout, any texture that does not rest on the cutout just disappears into oblivion). Save this cropped image into a loaded library and reload libraries. It can be a .JPG, we are not using an alpha channel.

If you haven't got a 'straight-on' view of your subject, you will first have to use Photoshop's Edit>Free Transform>Perspective and Distort to adjust and straighten the image.

Now you can place it as a 'Figure' into the ground plan. Without losing the proportions, resize it till it's the correct size. In this case, it's 590mm x 1800mm (2 ft x 6 ft).

Make the Material

From the Options menu>Materials in ArchiCAD make a new material. Take a material like “Whitewash”, hit the Duplicate button, rename the duplicate – in this case I called it “matthew_cutout”. Use the Texture button to bring in the Figure. Set the size of the Figure used for the texture to the same as the one in the plan – 590mm x 1800mm.

Make the Cutout

Now, back in the plan, select the Slab tool, set it to 10mm (3.8”) high and trace closely around the image – as in the illustration. Trace another rectangular slab over the cropped image. Set the material of both slabs to be “matthew_cutout”.

Initially, you will find the texture will be starting from a random position, but at least you have the comfort of knowing it's the right size – in fact it's coming from the main origin of the project, a long way away. We need to use Align Texture to get this starting from the bottom left corner. Open GL is a great help here because you get instant display to changes of the texture alignment in the 3D View.

Select both your slabs and view in 3D Axo. Use Edit>Align Texture>Set Origin and place the Origin at the bottom left corner of the rectangular slab. Now, try to set the direction: use Edit>Align Texture>Set Direction to go from a point on the cutout slab in a horizontal direction (use shift key for constraint to go rightwards) – if it offers you a choice of graphical or numerical, set the angle numerically, to 0°.

Select your cutout, view it in 3D Elevation view from 90° and save as an Object, Editable. Place it in the floor plan.

Look at the 3D Script

If your cutout fitted the Figure closely (to within one pixel) the 3D Script will have been written so that the bottom left corner is the origin of the new object. So if you set the texture to the bottom left, you will get a perfect alignment.

If the Align Texture routine worked, 3D Script will end with:

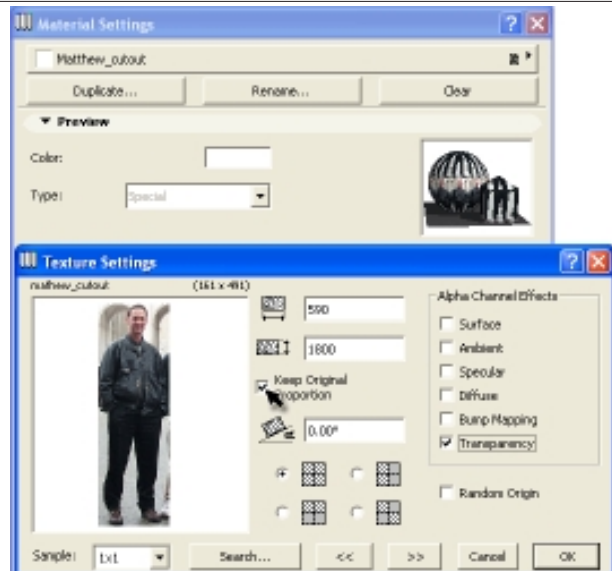
```
! The CPRISM will end, something like this....
-35.36420271267, 12.94639198852, 15,
-35.35731157426, 12.90504515807, 15,
-35.30677655926, 12.88720216694, -1
```

```
BASE
VERT -35.452594159, 12.887202166, -0.00045168882040
VERT -33.640501651, 12.887202166, -0.00045168882040
VERT -35.452594159, 13.533489550, -0.00045168882040
VERT -35.452594159, 12.887202166, 0.0090966223591
COOR 8468, -1, -2, -3, -4
DEL 1
BODY -
1
```

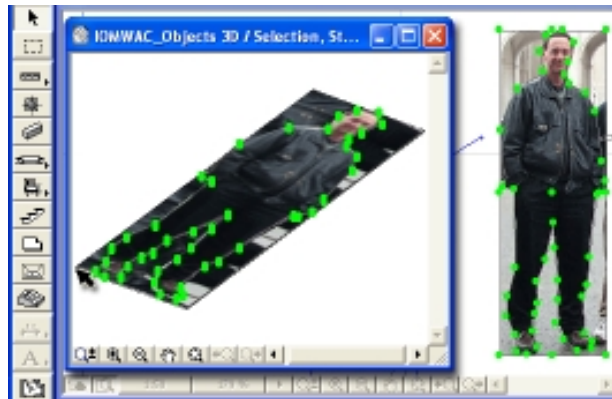
The Align Texture routine just did us a favour! it wrote in a perfect texture mapping routine! If you have a problem with Align Texture, just save the object. Open the 3D Script and have a look....

```
! The CPRISM will end, something like this.....
-35.36420271267, 12.94639198852, 15,
-35.35731157426, 12.90504515807, 15,
-35.30677655926, 12.88720216694, -1
```

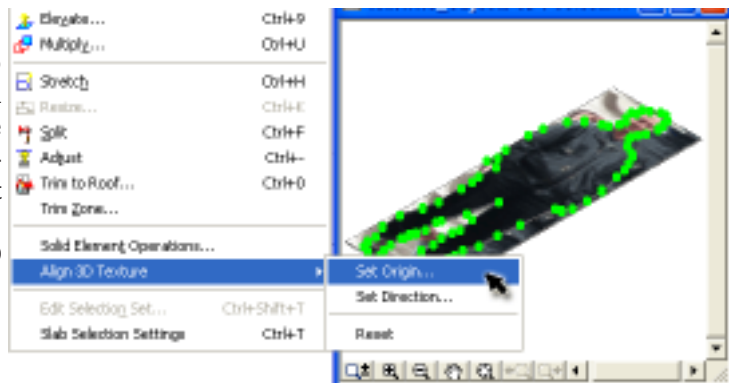
```
DEL 1
BODY -1
```



Make your own material in the Options Menu



Align Texture: do it in the 3D View



Just change this ending with a texture routine as follows:

```
! The CPRISM will end, something like this.....
-35.36420271267, 12.94639198852, 15,
-35.35731157426, 12.90504515807, 15,
-35.30677655926, 12.88720216694, -1
```

```
DEL top
ROTz 180
ROTx 90
BASE
VERT 0,0,0
VERT 1,0,0
VERT 0,1,0
VERT 0,0,1
COOR 258,-1,-2,-3,-4
BODY -1
```

Now, the Open GL view works with the Cutout perfectly and even Shadows are cast correctly.

You can use Ice figures

Cutouts can be used without any texture. People often look better in renderings as cutouts using 'Ice' or 'Water' than using realistic textures. If you start with accurate textures you have to continue. Ice Cutout figures work in ArchiCAD and external renderers, require less effort to make and do not distract from the architecture.

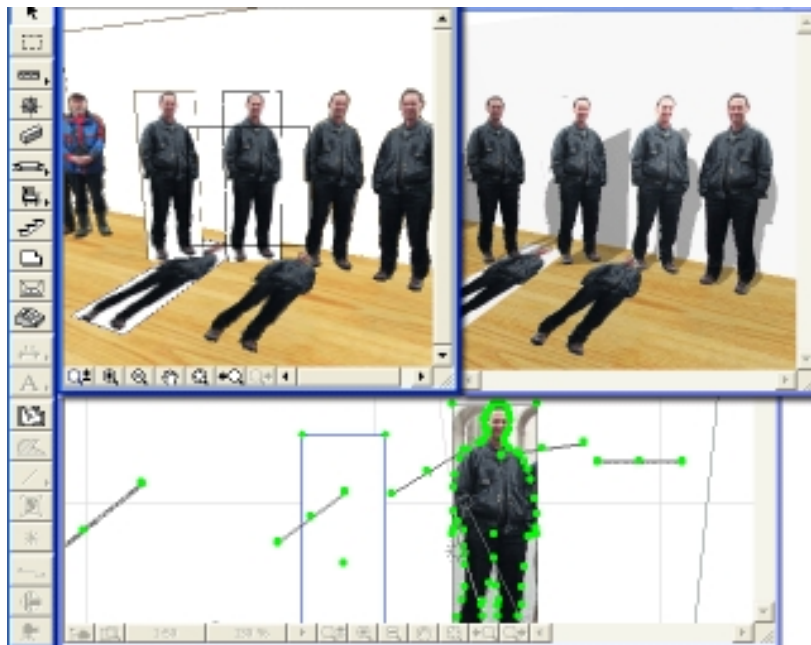
The final version of this object:

... Build in the Texture and Material

It is possible to build in the texture and material so that you do not need to make the material in ArchiCAD each time. Make the object with the material set to something unimportant like 'Whitewash'. Now you can use this easy script – copy and paste it into the Master Script of the object each time with minor modifications.

The idea of this is first to define a Texture based on the picture. You need to tell it the name, size, masking behaviour and rotational angle of the texture. Then define a Material based on that Texture. 24 is the material code for one written using a texture name. 1,1,1 means an RGB of white. The texture reference should be a number not a name, so we use an IND() function to convert the name to a number. 61 is a fill value.

```
!Master Script
!Texture and Material routine
!Write in the name of the Texture file and
!its dimensions in metres
DEFINE TEXTURE 'bilttexture' 'mathew_cutout.jpg',
    0.590,1.80, 1,0
DEFINE MATERIAL 'bilmat' 24, 1, 1, 1,
    0, 61, IND(TEXTURE,'bilttexture')
```



The two pictures above are Open GL (left) and Photorender (right). The PICTURE based objects cast rectangular shadows, the Cutout works perfectly as does the rectangular billboard using Cutout technology.



Ice figures are easy and quick to make, and can be used more frequently in a model without looking repetitive.

Now go to the 3D Script and change the three materials of the CPRISM to **'bilttexture'**, in quote marks. You must also insert a **PROJECT2 3,270,2** into the 2D Script.

The object will now be completely portable, and you will not need to make a new material in ArchiCAD's Options menu. There is no point in making a parameter for the picture file name as the cutout is specific to the figure you traced. But you should be able to copy and paste and modify this Master

A note on Shadows

Alpha channel based material textures, if applied to ArchiCAD elements or objects will cast shadows correctly cut around the outline of the alpha channel. Picture based objects will cast rectangular shadows.

To be sure of the best results, go to Image> Photorendering Effects> Options, 'use transparency in shadow calculation'.

If you want to cast perfect shadows, the Cutout billboard is the way to get the best result in both renders and 3D views – if you are prepared to make a material for each one or write the neat little routine on this page. Open GL and the 3D view do not use transparency in the Picture object.

If you want to control shadow casting fully, make a Boolean parameter 'shad' and start the 3D Script with the line:

```
IF shad THEN SHADOW ON ELSE SHADOW OFF
```

A compromise of the two types is that you can save yourself all that tracing around objects in ArchiCAD and use a rectangular cutout (ie a thin upstanding wall) and apply a texture to it which uses alpha channels. This will look like a rectangle in 3D but look OK in Photorender, and will cast shadows correctly.

Rectangular 'cutout'

The perfect combination object!

THE Rectangular Cutout is not really cutting out, it's a rectangle that uses the technology of the **Cutout** object. This is the principle of mapping a texture to a surface with a texture definition, and using a routine to lock the texture to the bottom left corner. It can have an interface like the **Picture** object – where the user is asked for the name of the picture file – but it produces a stretchy rectangle that will cast shadows correctly.

Master Script

We need to adapt the Master Script from the Cutout object.

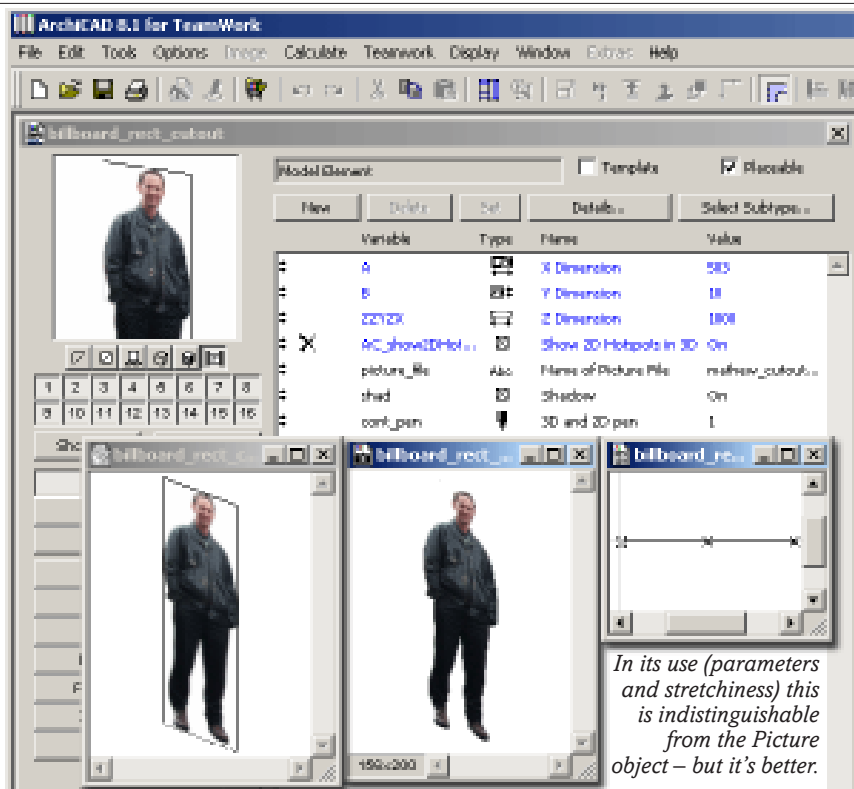
```
!Billboard Object rectangular
!Master Script
DEFINE TEXTURE 'biltexture' picture_file,
  A, zzyzx, 1, 0
DEFINE MATERIAL 'bilmat' 24, 1, 1, 1,
  0, 61, IND(TEXTURE, 'biltexture')
```

The small difference here is that the name of the picture file is now a parameter and the dimensions of the picture are dynamically resized by the 'A' and 'zzyzx' dimensions.

3D Script

We are not using a PRISM here. A POLY_ is one simple surface that will not reveal tell-tale edges showing in a render. We are adapting the texture mapping routine that was used in the previous section. Copy and paste!

```
!Billboard Rectangular.gsm
!3D Script
PEN cont_pen
IF shad THEN SHADOW ON ELSE SHADOW OFF
MATERIAL bilmat
ROTx 90
  POLY_ 5,
    0, 0, 1,
    A, 0, 1,
    A, zzyzx, 1,
    0, zzyzx, 1,
    0, 0, -1
BASE
  VERT 0, 0, 0
  VERT 0.1, 0, 0
  VERT 0, 0.1, 0
  VERT 0, 0, 0.1
  COOR 258, -1, -2, -3, -4
  BODY -1
DEL 1
```



You do not need to write graphical hotspots in the 3D because it works on the ABZzyzx cuboid. But if you do, it will work more pleasantly, showing dimensions in the floating palette.

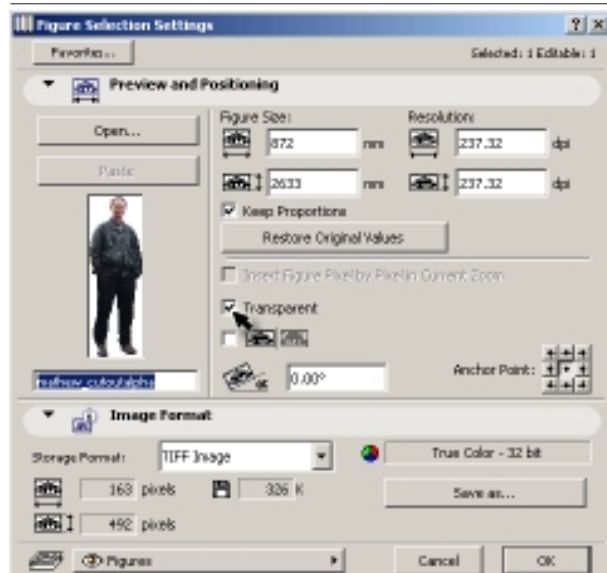
2D Script

Copy the script from the Picture object in the previous section.

```
!!Billboard_rect.gsm
!!2D Script
PEN cont_pen
HOTSPOT2 0, 0
HOTSPOT2 A, 0
HOTSPOT2 A/2, 0
CIRCLE2 0, 0, A/100
CIRCLE2 A, 0, A/100
CIRCLE2 A/2, 0, A/100
  LINE2 0, 0, A, 0 !AC8 and earlier
!AC9 - add HOTLINE2 0, 0, A, 0
```

This view of West Bridgford Central Avenue near the author's home was done by Bite Design in Nottingham. It uses billboard surfaces for the building elevations, 3D street furniture and a few billboard trees to fill in the spaces.



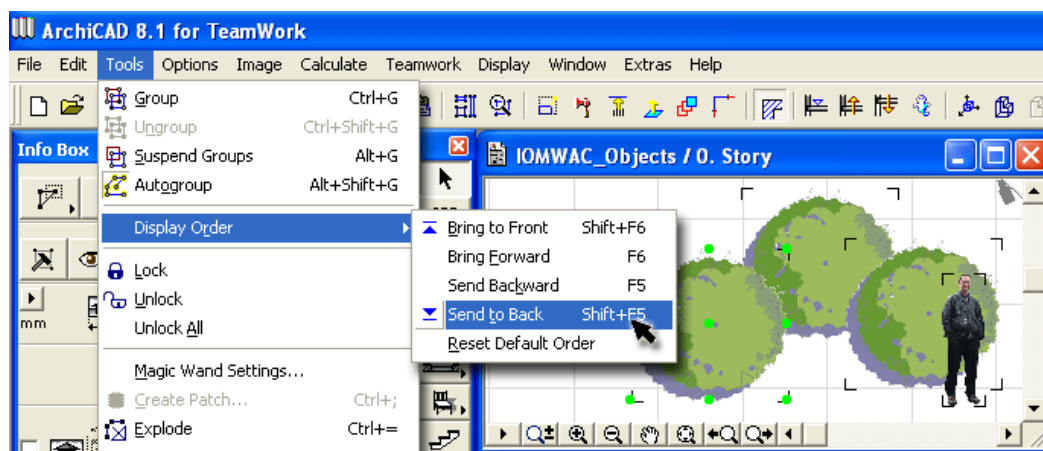


The Billboard idea in 2D?

If you place a bitmap Figure into the floor plan and look at its Settings dialog, you may notice that there is a transparency button. Any completely white parts of the image (which have an RGB value of 255:255:255 or FFFFFFFF) will show as transparent.

This is useful mostly for placing trees and plants into ArchiCAD in the floor plan. You can also place elevations of people, cars and trees into sections and elevations, giving the impression of upright billboard objects – all you need to do is whiten the background.

You can now use the 'Display Order' menu option to organise how the figures will overlap.



Get the Photograph right!

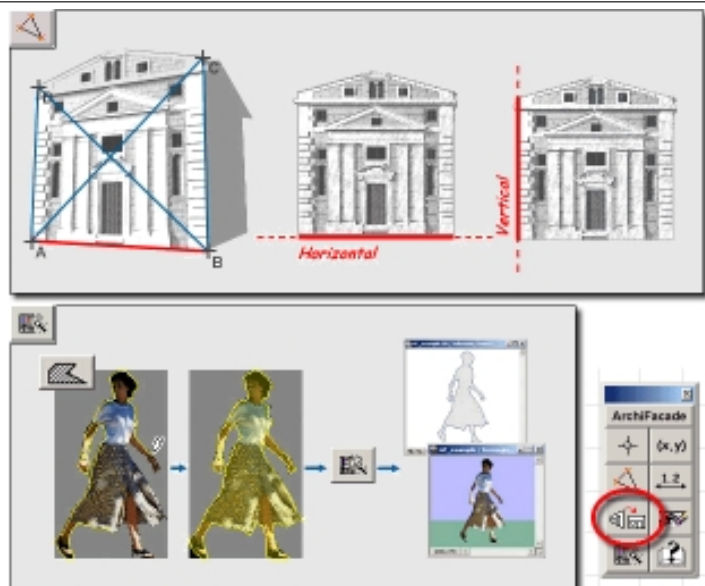
If you apply this technique to building facades, one problem is that you can so rarely get squarely in front of a building. There is either a policeman telling you to move on, the sun is facing you, trees or a tourist bus parked in front, or simply, the building is too tall for a front-on photo. You may have to make do with a perspective looking upwards, or from the side.

If you manage to get the photo in perspective, you can use Photoshop's Edit> Free Transform> Perspective and Distort capability to straighten out a facade. But it is hard work and the proportions and the upper level windows may still be wrong.

You can look at the facade, and if you can get clean images of the entrance and some typical windows and cornice details, you might be able to reconstruct a facade by copying and pasting the bits that you managed to capture – even if the rest was obscured with trees! But this is going to be hard labour!

Addons that help you

The serious billboarder will always carry either Cigraph's **ArchiFacade** for ArchiCAD in their kitbag, or the standalone Abvent's **PhotoCAD**.



ArchiFacade is an API (Add-on) which uses the cut-out method and generates a GDL object for each cut-out. It is good at both the requirements, straightening and regularising the bitmap image. Its talents extend to making the billboard object for you including the GDL texture routines!

Custom Window Tutorial:

without GDL... (almost)

by David Pacifico

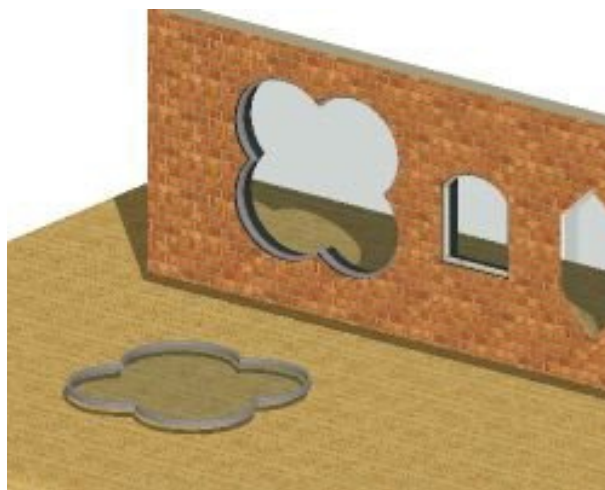
WANT to create your own total custom windows? Round tops, sloped head, or sill etc. Creating your own custom windows in AC 8 is easier than before because the WALLHOLE command in GDL will now cut window shapes with concavities:

1. Draw or build the custom window with slabs, walls and other tools so that the window is face down on the ground. Be sure to use a single slab for the glass that will also act as the WallHole. (So have it go through the Mullions to the jamb.) Build the window so the center of it is the project zero.
2. Select your creation and activate the 3D window (no need to go 3D window settings)
3. Chose File/GDL Objects/Save 3D Model as (Name it and place it in a project specific library) The file type is .gsm (no more .win or .dor)
4. A dialog box will come up with 3 buttons at the top. Select the middle button for a window, check remove redundant lines and choose format as editable gdl script.
5. Draw a wall and place your new window in it.
6. Now if your window has a non-square WallHole we will need to select the window and choose File/GDL Objects/Open Object. (Don't be afraid we just need to copy and paste some GDL)
7. Click on the 3D Script button. Scroll down until you see the script that is your single slab that you used to make the glass. Select and Copy the script that starts with cPRISM_ "Glass", "Glass", "Glass" and ends with BODY -1. (all the stuff between is the coordinates we will be using for the WallHole)
8. Scroll down to the end of the script and paste. Delete the first pasted line that likely looks like:
cPRISM_ "Glass", "Glass", "Glass",

Type in front of the next line type WALLHOLE. (This cuts the wall for the window. Click the button Check Script.

9. Save the object and Rebuild. Extra Credit: If you want to make it a corner window change the object Subtype by opening the object and clicking on the "select subtype" button in the main window and choosing the corner window subtype. Hope this helps.

To get the 4-leafed clover window under previous AC versions, you would make 4 WALLHOLES each spaced so that they overlap – all being convex, they would not cause an error, whereas a single wallhole outline would cause and error.

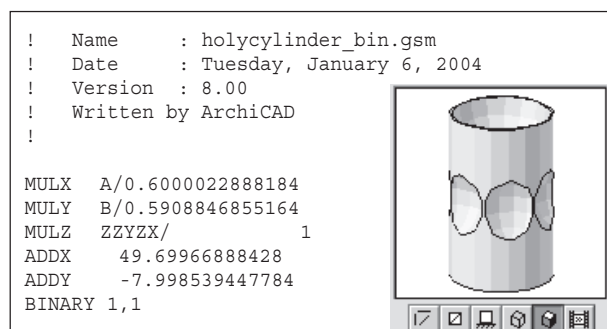


BINARY

THE BINARY statement is what all Binary converted Library objects contain, and have a limited set of parameter values of 3, 2, 1 or 0 (for pens and materials) and an index number of 1. Binaries are just a big chunk of impenetrable machine code that defines the polygons, edges and nodes and materials. The code for the binary is embedded within the GSM object and is not detachable. Binary objects are not editable (except for the odd ADD, MUL or ROT in front of them or the pen parameters, and in the normal sense, there cannot be any further parameters for them, and no hotspot editing.

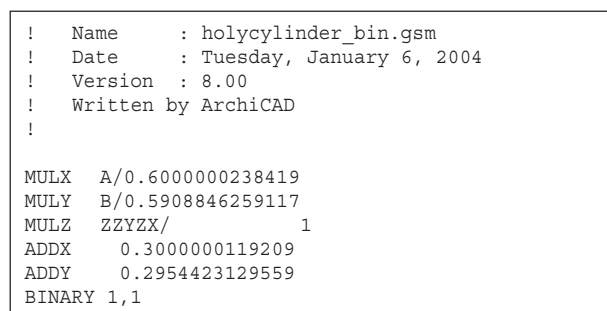
Binary objects contain all the data necessary for 3D display, including materials. They are faster and more reliable than editable objects (which may have hundreds of k of complex script to read through and interpret) and can handle more polygons.

If you wish to combine Binaries, it is surprisingly easy. Place two in the floor plan, arrange them as you wish, view in 3D then save as a new 3D object, and make sure you tick the boxes to make them Binary.



Open that new object, and you have a single binary which now contains both, and has a new 2D symbol. There is no use for the command in your creative GDL, but you can manipulate the Binary to some extent. You can insert ROT or MUL or ADD commands in front of the Binary script, to multiply or move the object. You can write a FOR... NEXT Loop to do something interesting with the Binary that is embedded in your object.

Here is an object (made for another exercise, viewed in 3D in Plan, and saved as a binary object. First of all note that it is preceded by a number of cursor movement commands. Because it was made about 49x8m



from the origin, that 49x8m is embedded into the binary, and ArchiCAD writes offsets to bring it back to its own origin, at the near left bottom corner. Lesson number one is: build objects over the floorplan origin. Next you can see that the MUL commands are there to make it stretchy – because they all refer to the magic parameters of A and B. The object itself is contained in the incredibly brief single line command of BINARY 1,1.

Fun with BINARY

Remake the object over the origin, save it again, and this time the binary itself is centred. Now you can write

!Fun with a Binary

!First, Delete all the MUL and ADD offsets

!Build the 3 in a row

```
BINARY 1,1
ADDx 1.0
BINARY 1,1
ADDx -2.0
BINARY 1,1
DEL 2
```

!Build the circular structure

```
ADDy -2.5
FOR angl=0 TO 331 STEP 30
```

```
ROTz angl
ADDx 1.5
BINARY 1,1
DEL 2
NEXT angl
DEL 1
```

This is efficient, because it doesn't double or quadruple the amount of code in the object, it just reuses the same code again and again.

!Build the vertical structure

```
ADDz 2.4
ROTx 180
BINARY 1,1
ROTx 90
```

!Rotate 4 times horizontally

```
FOR k=1 TO 4
ROTy (k-1)*90
BINARY 1,1
DEL 1
NEXT k
DEL 3
```

!Sadly, CUTPLANE doesn't work

```
ADDx 4.0
ADDz 0.5
CUTPLANE
DEL 1
BINARY 1,1
CUTEND
DEL 1
```



all sorts of moves, loops and other things using that BINARY.

One disappointing discovery is that the Binary is so complete that CUTPLANE and other cutting commands do not work.

Binary Hacking is a cutting trick

However there is a trick that will cut binaries. If you place the object in the ArchiCAD floor plan on its own, and use the Marquee polygonally or 3D cutaway, you can cut it up in various ways, the view the object again, and save it as a new binary object. Bring it back in, put a ROT in front of it, update the 2D to PROJECT2, cut it a different way, save again... and again... and again...

Join Binaries together?

XML Converter!

THERE is no doubt that ArchiCAD is the required and preferred environment within which to produce GDL, but there are other ways to produce objects. Some of you may have tried “Black Turtle” (Abvent) or “3NF” (Webscape) – but Graphisoft themselves provide a small application called **XML Converter**. There is one for AC8.1 and one for AC9.0. Both of these are on the Graphisoft website.

XML is Extensible Mark Up Language, an evolutionary development from HTML that provides a way to store and exchange data on the web. You can store parameters, text and data in an editable text file using chevrons and slashes (<> </>) to separate fields. In this way a converter can read the XML data and change the data into something more complex – in this case, a .GSM object.

XML Converter is a way of converting GSM objects to XML or XML objects to GSM. It's not really a way to make a completely new object (this would be VERY hard work!), but it's a way to edit and convert existing libraries or objects. Every part of the object becomes text – modify it in a text editor, then convert it back to an object. You can do this to individual GSM files, or a a directory containing a whole lot of GSM files – in fact, you could use it to update or edit a whole library.

Actually, XML Converter is a bit of a nightmare to use, it works only in the raw Command line interface of MSDOS or OSX, and nobody really wants to go back to that. Once you have got a GSM object into XML form it's just text, and the best editor is Dreamweaver or GoLive or a similar web page editor. If you want to write XML creatively there are tons of tutorials on the web and you can easily find them with Google.

What do you do with XML Converter?

I see two main uses of **XML Converter** which make it worth learning how to use this, to do things that cannot easily be done in normal GDL.

1. You can edit the parameter descriptions. This is something that is normally very tedious. If you have an object in English that you want to have in German, you have a very hard workload to open each object, write the new description in, then save and close (of course you have to know German too!). By converting the object or the whole library into XML, you can use a smart Global Search and Replace routine to replace all parameter descriptions – this does not replace the parameter name or alter your script. Then convert them back to GDL objects, into a different directory.

XSTEEL can use XML with GDL!

An example of XML with GDL is in Steelwork. There is an XML translator for using XSTEEL (from Tekla) with an ArchiCAD / GDL model. You can assemble an outline frame using the steel sections in the ArchiCAD library. Don't worry about the joints – bring the steel sections close to each other. Select and save the model as an XML model using the XSTEEL translator (another plug-in). XSTEEL can open this file, enabling an engineer to do calculations, perfect the joints. This model can then be saved as an XML file from XSTEEL. ArchiCAD / GDL can then reimport the modified file and show you the final corrected steel frame model.

2. Adding Binaries. We have all made objects with BINARY. We know that we can copy and paste the Binary command, and we will get a repeat of the object inside the GSM file. The code for the binary is locked up inside the GSM file, but is inaccessible using GDL as we know it. We cannot copy a binary from one GSM file to another. Binaries in autoscripted objects always have a section number of 1.

Some of the newer more sophisticated objects in the GDL world such as the people and cars and trucks have not been made with GDL – they have been made with **Cinema 4D** or **Rhino** or **3DS** and brought in to GDL. If you want the objects to animate – e.g. doors open and shut, hide or show people, Lights on and off, you cannot have the car as one single binary. If you have a number of binaries with IF statements, you can make very realistic objects. You can contain up to 16 Binaries within one GDL.

If you look at the 3D scripts of some of the newer cars and trucks you will see a lot of different BINARYs for the body, the doors, the roof, the seats, the passengers etc. These finished objects have all been created with XML Converter, not with GDL.

How to join Binaries together

Each part (door, bodyshell, roof) has been imported from somewhere else as a separate GDL object with a BINARY of section number 1. Each part has been converted to XML with the Converter. They can all be separate files, or if you convert a folder full, you will get one long file.

You can now open each XML file with Dreamweaver or similar. The XML code shows a great mass of hexadecimal text for each Binary, with a header and footer. Copy and paste all the chunks of Binary code into the one object you decide will be the final one. Change the section numbers of the the binaries to a sequential order, e.g. 1, 2, 3 etc. The limit is 16.

Tall stories from Down Under

The Eureka Tower, designed by Fender Katsilides of Melbourne Australia is, in its 92 soaring stories, the tallest residential tower in the world, and Australia's tallest building – and it is an example of the Virtual Building/ BIM approach to design and documentation. **David Sutherland**, Director of Planning for FKA, is a key member of the team and has travelled and lectured on aspects of the Eureka Tower. Here he looks at the contribution of GDL. You can read more in the AEC Bytes website at:
<http://www.aecbytes.com/feature/EurekaTower.htm>

IMAGINE, if you will, an architectural practice on the other side of the world from most of you. While not an overly large practice, we had created large and significant projects in our home city of **Melbourne, Australia**. We had just embraced **ArchiCAD** as the prime tool of architectural process, but because we were rolling it onto new projects and allowing existing projects to continue as they were we had only a few licences in the office, a couple of experienced users and were training our architects to use it.

As we were taking those first few tentative steps in a new way of working, the design development and construction documentation of **Eureka Tower** came on stream. We decided to make something of that process and, in the process, found that we had made something of ourselves.

This little side-article is a melange of anecdotal observations arising from our work on Eureka Tower, discussing GDL, but giving the reader a bit of light relief from all these elegant but intensive code structures in the GDL Cookbook.

Parametric flexibility versus Associativity

In traditional modes of architectural process the design and documentation phases are distinguished from one another by the intended outcome of each stage. Design is concerned with the projection of aspiration whereas documentation is focused on the description of the detail. As a consequence design looks to précis our understanding and, by comparison, documentation seeks to describe the singular and unique and the particular interrelationships between those singularities.

Understanding that difference is critical to an understanding of the context of the use of GDL and parametrics within the virtual building. Because the continuum of design to building description is one of increasing detail, it is also one of decreasing flexibility. Correspondingly, as the depiction of elements allowed by parametric geometric description is increas-

GDL Case Study:

by **David Sutherland**,
Fender Katsilides

ingly more narrowly focused, the non-geometric description should become richer. Simultaneously we find that the description of the interrelationships between elements becomes more important.

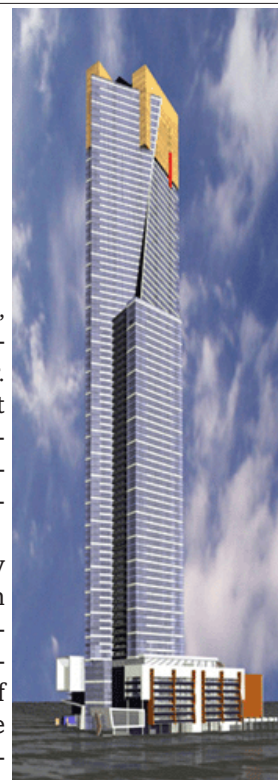
For the GDL community we find therefore that such an approach is one of decreasing use of parametric flexibility and increasing use of associativity. Most of us are familiar with the use of parameters to change the geometric and descriptive characteristics of elements. However we are generally less used to parametrics to describe relationships between components.

Self detailing associative Objects

Let's consider a humble wet area fixture: the basin. During conceptual design we wish to have significant flexibility in that object – perhaps manufacturer, basin type, dimensions, even colour are parameters we may wish to change during design. As design progresses through succeeding development, however, we may wish to turn off some of those variables. Perhaps we have agreement that a single manufacturer will be used. We therefore wish to lock away any temptation for the user to select any other manufacturer. Then the time arrives when we inevitably wish to finally determine the final model of basin, and therefore associated dimensionality, materiality and even order code. To have any other alternative as a parameter is courting the danger of those parameters being inadvertently altered, thereby creating incorrect documentation.

The consideration of associativity creates different opportunities. One of the explorations we undertook during the Eureka Tower project was the self-detailing component. Taking as our test bed a pre-cast panel we considered how we could author that element so as to have it recognise elements adjacent it, and automatically detail its connections to those adjacent elements. The basis of the approach was to create in GDL that detailing.

For example, if the pre-cast panel was situated next to another panel, the detail shown would incorporate the rebates in the panels to accommodate the steel jointing elements. Those elements would include cast-in plates, stitch plates, welds, and the grout used to finish the joints. Waterstops were included, as was



jointing compound. Those elements automatically re-dimensioned themselves depending on the parameter-driven dimensionality of the panels.

In our experiments that detailing was carried out as a 2D exercise, but could just as easily be carried out as a 3D process.

How far could you take this?

This experiment was simplistic in its execution, but gave rise to speculations as to the effectiveness of more sophisticated approaches. That approach allowed those panels to be located in accordance with the design intent, dimensioned in response to construction location and structural requirements, and the detailing would take care of itself. And not just in those panels where we had decided that details should be taken, but in every instance of the panels.

That approach leads us towards the answer for the question: how much should we model in 3D? The apprehension underlying that question is related to the concern about having to model every little aspect of the virtual building – down to the bolts. However that approach is merely manual drafting in 3D guise. Rather than that rather outmoded approach we believe that instead we will be installing assemblies of components that will be self-detailing in a far more sophisticated manner than our humble pre-cast panel.

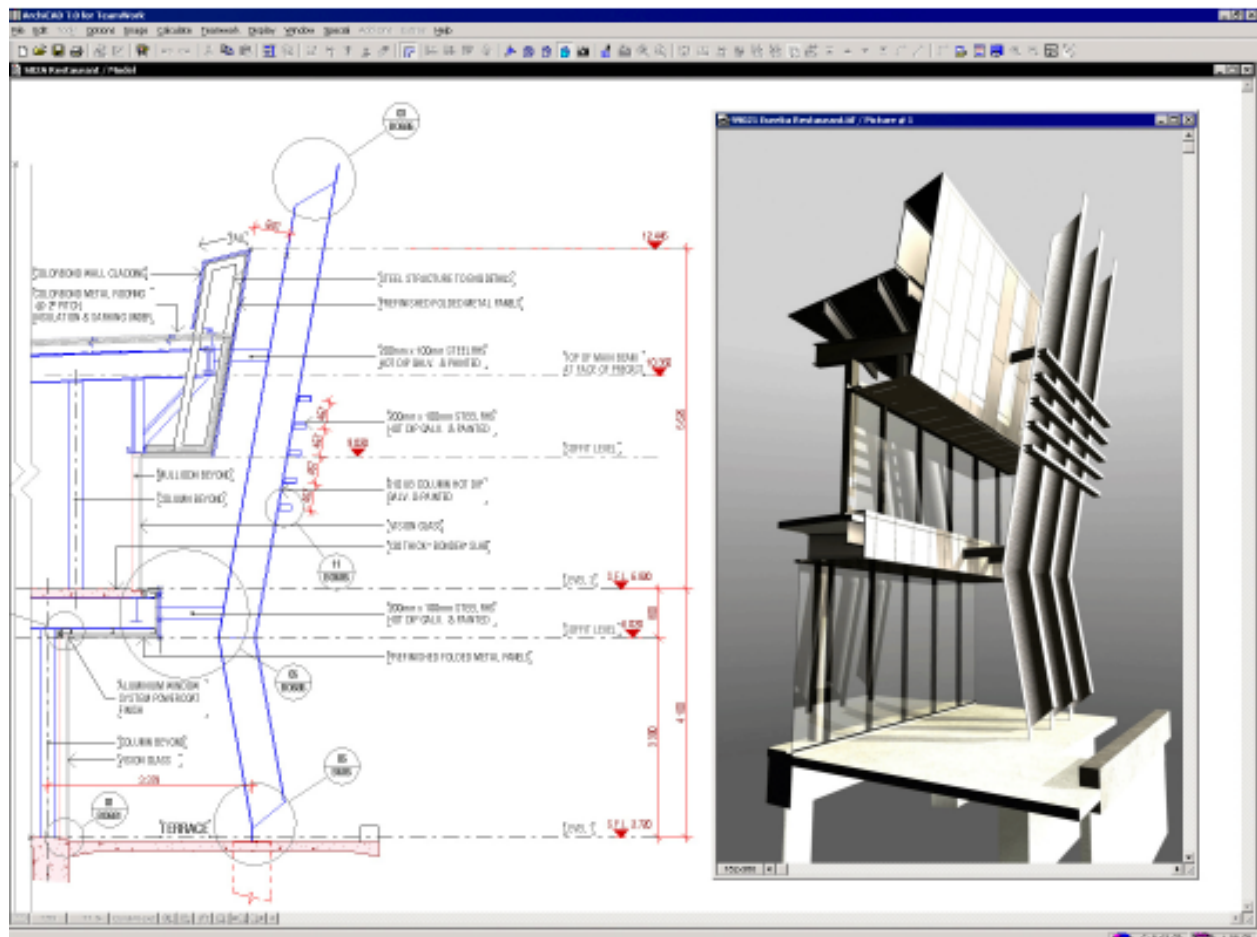
At that time we were toying with that panel we were contemplating the nature of construction documentation for the virtual building. Given the circumstance where we, in our 3D virtual world, were coordinating

with others in the 2D world, could that interface be by mechanisms other than 2D drawings? Understanding that our virtual model was comprised of elements and components, could we schedule the entire building rather than deriving drawings from our 3D model? Accordingly we set up schedule sheets which, using our test-bed pre-cast panel as a prototype, could schedule to scale plans, elevation and section of each element with dimensions, locate that element on a 3D Cartesian grid within the building and provide specification data related to each element. Admittedly to do so for every element in the building would create a stack of A4 paper as high as the building itself, but it does lead to speculations.

Searching for the practical solution

When we started our detailed work on Eureka Tower we had just shifted from ArchiCAD 6 to 6.5. While the rest of the office has moved on through successive upgrades to our current platform of 8.1, Eureka has stayed with 6.5. We just didn't want to take the risk of unforeseen problems occurring through upgrades of software. It was in those days that we formulated our approach of 'by-product' – a process whereby we try as much as possible to derive design or construction documentation automatically as a by-product of our design processes.

A component of that approach was to work very little in Plotmaker which, after all, represents a drawing mode rather than a modelling mode of working. That can be achieved for the drawing component by



creating the live links between ArchiCAD and Plotmaker – which is a typical use of Plotmaker by most ArchiCAD users. We went a step further by linking the title block information on our drawings to an external relational database, which had been used for some years in our practice as a document register/ transmittal/ issues database. That linking was created through an ArchiCAD object which uses the INPUT command of GDL to automatically read information from a text file. We set up our database to automatically create an appropriate text file when required for document creation or revision.

I remember that at the time we contacted the Australian distributor of ArchiCAD (who has consistently over the years been of great support to us) to get some assistance on the integration of our database with GDL. The first response from him was “Why are you reinventing the wheel? There are plenty of proprietary document management software packages around!” I explained that we weren’t reinventing the wheel – at the time of the creation of the database, there were actually no wheels. (I had created the initial version of the document register/ transmittal system in the mid 1980s, using a SQL engine on DOS computers, reproduced it in Filemaker Pro when I fell in love with Macintosh computers in the late 1980s, and subsequently ported the system to Access when I belatedly realised that working in Windows-based offices was going to be an unfortunate reality.)

Despite the initial response a representative was flown from Sydney to our office to assist us in understanding how we could connect our two systems. His comment on looking at our document register? “Mmm, somewhat more sophisticated than we had thought...”

GDL proves itself to be a valuable tool

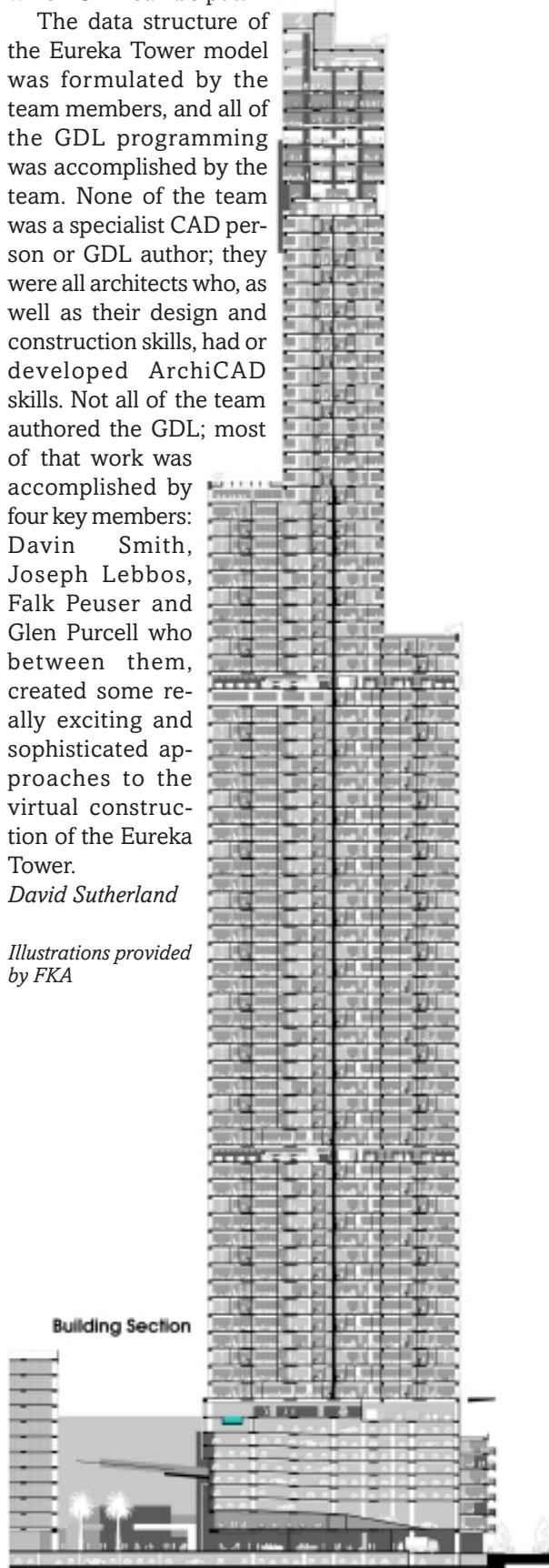
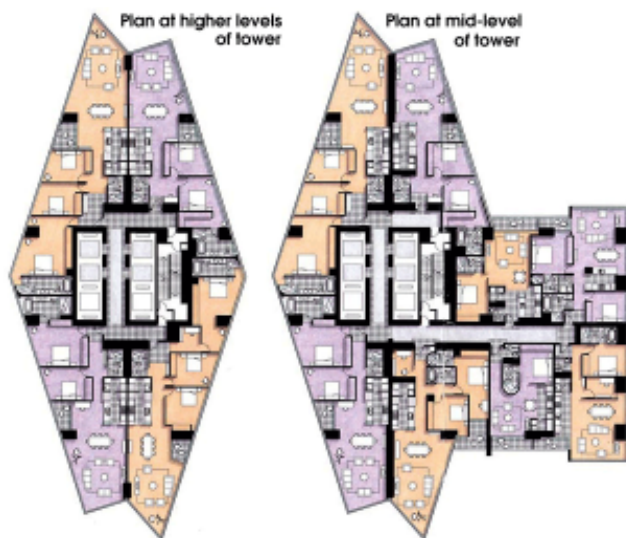
One of the GDL tools developed during Eureka was an object which would enable team members to take any object, rotate it to any angle, and then to project onto succeeding storeys the plan manifestation of that object. In doing so one could specify the height above the floor level the plan cut was to be taken, how much of the object should be seen below that plan cut and

the graphic fill required, if any, in that plan. Hot spots, of course, were provided. This tool enabled correct assessment to be made of the position on plan at any storey of the many raking columns and trims in the Eureka project and is, to my mind, a wonderful example of the creative uses to which GDL can be put.

The data structure of the Eureka Tower model was formulated by the team members, and all of the GDL programming was accomplished by the team. None of the team was a specialist CAD person or GDL author; they were all architects who, as well as their design and construction skills, had or developed ArchiCAD skills. Not all of the team authored the GDL; most of that work was accomplished by four key members: Davin Smith, Joseph Lebbos, Falk Peuser and Glen Purcell who between them, created some really exciting and sophisticated approaches to the virtual construction of the Eureka Tower.

David Sutherland

Illustrations provided by FKA



User's Global Variables

User editable GV: let GDL objects talk to each other

NORMALLY, Global Variables are a way for GDL objects to find out what is going on in the project as a whole – current wall thickness, camera position, sun azimuth, drawing scale, object's ID etc. You cannot write back to these, for example you cannot tell the sun to be at a different altitude or azimuth.

However Global Variables can provide a way for objects to talk to each other. User editable Global Variables are all of the form GLOB_USER_1 - 20. A GDL object can write a value for a user editable GV into ArchiCAD's global memory – the others will take note. The only use for this really is for a family of objects which need to share common data, e.g. a set of furniture object or building components in a manufacturer's library – they might need to share data like the name of files to open to retrieve tables of prices. If one GDL object contains a statement:

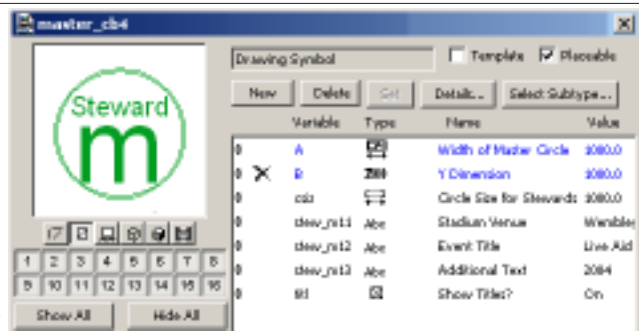
GLOB_USER_12= 'furnitur_data.txt' , it places that information into global memory, thus making it available to all other objects. Thus, the others know the name of the file to open to get furniture prices.

My guess is that you can cause yourself some difficulty if you let ordinary GDL objects do this, as you might have difficulty remembering which one did it. You could have a lot of windows in a building whose level of detail in the 2D symbols were autoscaled. It could be operated so that if you opened one of the windows and set it to a fixed level of detail, it would set a user editable GV, and all the other ones would also change to the fixed level of detail. I am still puzzling about how you would reverse this if you forgot which window started it!

Introducing "Master" GDL Objects

The answer is to use these Globals in a GDL object whose name is prefixed with the word **Master**. Any GDL object starting with this prefix is loaded before other GDL objects and this can put values into ArchiCAD's global memory, and these values can be dictated to other objects. A Master GDL file can quietly exist in the background in a loaded library. A Master file can contain extra material definitions, fill and line definitions. If it is to be highly editable, you could have it visible in the plan somewhere as a sort of control panel object with a simple 2D symbol – with some parameters that you could set, and which could inform other objects in the model. Even if it is not in the plan it will continue to send values to global memory, but these will be the defaults it was saved with – it's best to have it visible if it's to be of any use!

You have 20 user editable GVs, and those from GLOB_USER_1 to GLOB_USER_10 are for numeric



Master GDL files : these get loaded first, in alphabetical order. This one is designed to manage 2000 or more 2D markers.

values (or indices). GVs from GLOB_USER_11 to GLOB_USER_20 are for strings (e.g. the project name, datafile names etc.).

Usually there is quite a delay in getting GDL objects to notice an update in Global Values. If you are changing a value of an editable GV, reloading libraries or the plan file will guarantee that objects will re-read the user GVs.

Try an example!

We can make this little Master object, and give it a simple icon of a circle with an 'm' inside. We can try to experiment with sending strings and number values to global memory and see if other GDL objects can pick them up. We only need to write a short master script.

```
!Master Object : Master Script
GLOB_USER_1 =titl      !Boolean for titles
GLOB_USER_2 =csiz      !Size of Circle markers
GLOB_USER_11=stew_m11 !Stadium Venue
GLOB_USER_12=stew_m12 !Event title
GLOB_USER_13=stew_m13 !Extra line

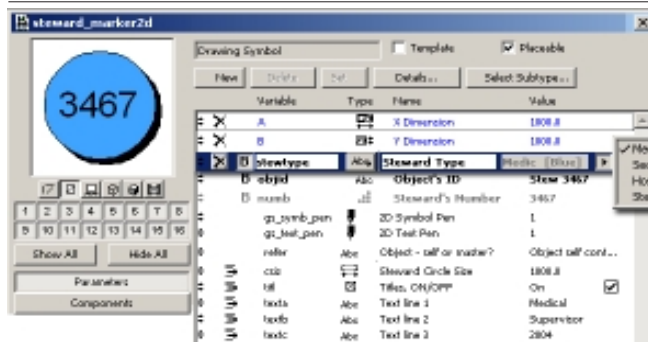
!Master Object :2D Script
HOTSPOT2 -A/2,0, 1001,A,1+256 !Base
HOTSPOT2 A/2,0, 1002,A,2 !Move
HOTSPOT2 -100,0, 1003,A,3 !Vect
HOTSPOT2 0,0

DEFINE STYLE 'txta' 'Arial Baltic',A*600/A_,5,0
STYLE 'txta'
TEXT2 0,-A*0.1,'m'
CIRCLE2 0,0, A*0.50
CIRCLE2 0,0, A*0.48
DEFINE STYLE 'txtb' 'Arial Baltic',A*150/A_,5,0
STYLE 'txtb'
TEXT2 0,A*0.2,'Steward'
```

2D numbered Marker and their Master!

WE can write an object that will make use of globals and some other interesting features in GDL. This object is a numbered circle (with a drop shadow) that is used for marking the locations on a plan of safety stewards in sports grounds.

Major sports grounds may be used for Football, but also for Reserve games, Firework displays, Rock concerts, religious gatherings, film broadcasts – each will have a different size of audience and need different



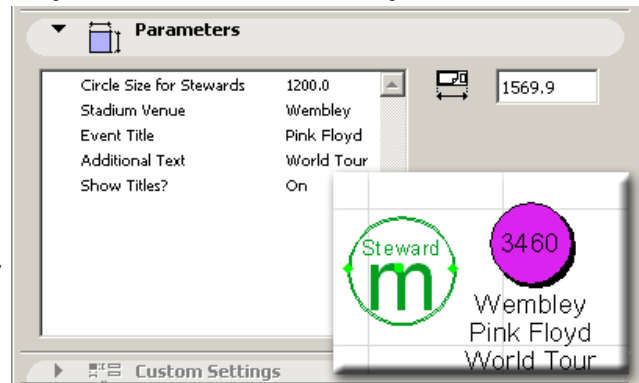
Using the Steward's marker

Place the Master-steward in the plan somewhere and then place one steward in the plan. In the object settings, give it an object ID of something like "Stew:1". If you alt-click that, and proceed to place more, the new ones are all sequentially numbered with that prefix – needs to be 5 digits in my system. Click 2000 of them down if you need that many, then change their colours. Make sure that the first one you alt-clicked was set to obey the master.

safety steward arrangements. Different layouts of stewards for different events can be organised using layer conventions in ArchiCAD.

Some stadia may have 6 or even 12 different levels, so changes to a plan can be very tedious. Select All will only update the ones on that level. A **Master object** can be a **Control Panel** to update all objects in every layer in every storey.

There are four categories, *Medical*, *Security*, *Hosts* and general *Stewards*. Each has a different colour. Each must have a separate number, and there could be up to 2000 stewards in a major venue such as an Olympic stadium or a Wembley Cup final. We can place coloured circular spots for the stewards and set them to obey the Master control panel or to be self contained.



```
!Steward Object - Master Script
sv1='Medic [Blue]'
sv2='Security [Violet]'
sv3='Host [Green]'
sv4='Steward [Yellow]'
VALUES 'stewtype' sv1,sv2,sv3,sv4
VALUES 'objid' GLOB_ID,
"This can only be edited in",
"the Object's ID setting",
"Format: 5 digits followed by Number"
PARAMETERS objid=GLOB_ID
rv0='Object self contained'
rv1='Object governed by Master'
VALUES 'refer' rv0,rv1
IF refer=rv1 THEN
  titl=GLOB_USER_1
  csiz=GLOB_USER_2
  texta=GLOB_USER_11
  textb=GLOB_USER_12
  textc=GLOB_USER_13
  LOCK 'csiz','titl','texta','textb','textc'
ENDIF

!Set up Pen colours for fill
IF stewtype=sv1 THEN stewpen= REQ('Pen_of_RGB 0.2 0.6 1.0')
!Light Blue
IF stewtype=sv2 THEN stewpen= REQ('Pen_of_RGB 0.9 0.2 0.9')
!Violet
IF stewtype=sv3 THEN stewpen= REQ('Pen_of_RGB 0.0 1.0 0.0')
!Green
IF stewtype=sv4 THEN stewpen= REQ('Pen_of_RGB 1.0 1.0 0.0')
!Yellow

IF numb<10000 THEN diyn=240
IF numb<1000 THEN diyn=300
IF numb<100 THEN diyn=400
IF numb<10 THEN diyn=500
DEFINE STYLE 'stewnum' 'Arial Baltic',
csiz*diyn/A,5,0
DEFINE SOLID_FILL 'stewfil'

!Find out the value of 'numb'
string=STRSUB(GLOB_ID, 6,4)
x=SPLIT(string,'%n',numb)
PARAMETERS numb=numb
```

!Steward Object - 2D Script

```
HOTSPOT2 -csiz/2,0,1001,csiz,1+256 !Base
HOTSPOT2 csiz/2,0,1002,csiz,2 !Move
HOTSPOT2 -100,0,1003,csiz,3 !Vect
HOTSPOT2 0,0

STYLE 'stewnum'
PEN gs_text_pen
TEXT2 0,0,numb

!Titles
IF titl THEN !csiz*0.35 spacing
TEXT2 0,-csiz*0.75,texta !Venue
TEXT2 0,-csiz*1.10,textb !Event
TEXT2 0,-csiz*1.45,textc !Extraline
ENDIF

PEN gs_symb_pen
SET FILL stewfil

!Background Shadow
POLY2_A 2,7,gs_cont_pen,
csiz*0.05,-csiz*0.05,901,
csiz*0.5,360,4001

!Actual Steward symbol
POLY2_A 2,7,stewpen,
0,0,901,
csiz*0.5,360,4001
```

Each of the following Stewards are correctly numbered. the GDL Object takes the GLOB_ID of the object and displays a number. It calculates the correct size of font based on the number of digits.

You can apply this idea to other cases where you need to organise markers or objects, e.g. in setting out an exhibition space or a supermarket or a distribution warehouse.

STRSUB and SPLIT

Note the smart routine with STRSUB to read in the substring of the number contained in the ID; SPLIT converts it to a number. Using LOCK and VALUES, the data can be displayed to the user, but it also provides them with instructions for using the object correctly. If you want a single steward to be special, you can set it to be self contained, and enter a special size and titling option for it.

Rich Text in ArchiCAD 9

ONE of the great improvements to AC9 is the rich text in the working environment. On the principle that anything that appears in the floor plan must also work in GDL, there is a way to author Rich Text in GDL – but it's very hard work! It is not simply a matter of using something easy like BB-Code.

BB-Code is easier

When you have tried Rich text in GDL you will wish it was. BB-Code is used in bulletin board systems like ArchiCAD talk as an easy to learn and simple method of formatting text – reminds me of word processing in the early 1980's, but it's applied to HTML style text.

For example, 'The **quick brown fox** jumped over the *lazy dog*' could be written as 'The [B]quick brown fox[/B] [color=red]jumped[/color] over the [I]lazy dog[/I]'. This would give us the 'fox' in bold and the 'dog' in italic and 'jumped' in red and the rest of the sentence in default pen colour. It's very simple – square brackets with a code starts an action, and square brackets slash with the same code ends it.

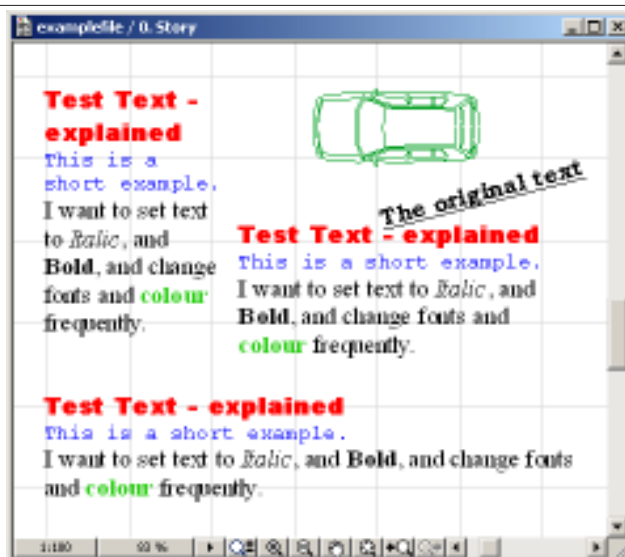
There are masses of tutorials on BB-Code on the internet – it can even include pictures and active URLs. You can change lettering size and colour and style. We should all put pressure on GS to retain the new system (for compatibility) but to make another available for bears of little brain like us to use, using a version of BB-Code.

Rich Text hierarchy

Rich Text in GDL operates a bit more like Solid modelling. You have to build up definitions of styles, assemble paragraphs and text blocks, and then use the RICHTEXT command to finally place the result (a bit like PLACEGROUP). There is a systematic hierarchical sequence to get it right. Hardened GDL writers will like it because of the vastly improved appearance in self labelling and dimensioning objects, but normal GDL users will be content to get by in the way we have for the last few versions of GDL, changing the pen colours perhaps. For investment objects (where your time is rewarded in the number of times you will use the object), it is well worth learning how to do this. Word wrap is dynamic, so if you have graphical hotspots to resize text width, you will get a very powerful looking result.

Try some Rich Text

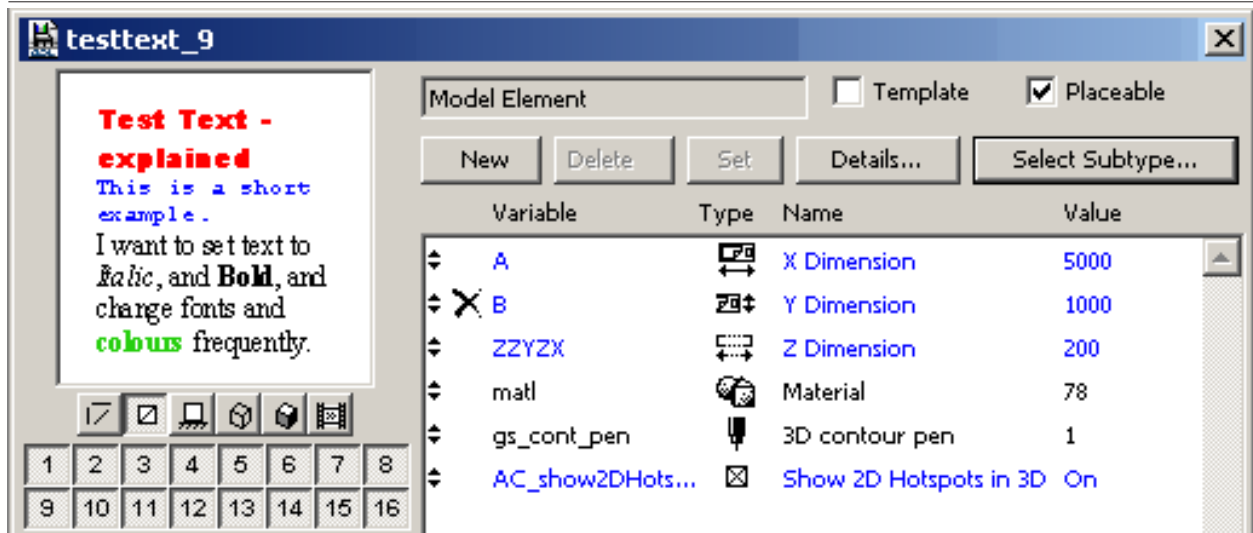
The best way to understand for yourself the way RICHTEXT and RICHTEXT2 work in GDL is to create a sample of rich text in ArchiCAD and drag it from the floor plan into GDL 2D Script and see what it makes as an autoscript. Once you have mastered the 2D, it's very easy to adapt the result to make it into 3D text. As usual, the autoscript gives one very prescriptive



numbers for everything (pens, font sizes, indents, letter spacing etc) but smart parametric routines in your script can make it perform parametrically.

The Sequence is as follows:

- Define the Styles you are likely to be using. If you are making a single GDL object, you will avoid going overboard with an excess of styles. If you are making a library of objects you will be able to copy and paste the same definitions from object to object. Instead of DEFINE STYLE, you are encouraged to use DEFINE STYLE{2} name font, size, facecode because you do not need an 'anchor code' – left, centre or right is set for the paragraph. You do have additional facecodes, for example superscript, subscript and strike out.
- If you want 3D text, you cannot use the same DEFINE STYLE{2} statements, because definitions are in millimetres - 4 mm looks pretty good on the page in 2D, but pretty small in the 3D. This chapter will provide a solution – you write a separate one, or you could use a single DEFINE statement with a 'scaling factor' equal to the scale of the drawing. It's extremely unusual to want Text that is simultaneously in 2D and in 3D at the same size.
- If you want coloured text in 3D, you might have to write a few material definitions too, based on the colours. Otherwise, in the absence of materials, the 3D text will follow the colour of the Pens.
- Next you define Paragraphs, each one with the PARAGRAPH statement. This is a bit like building Groups in Solid Geometry Commands. A paragraph is given a token name, and it governs alignment, indents and linespacing and tabs. Start a paragraph with PARAGRAPH and end it with ENDPARAGRAPH. Within that, you set values for pens, materials and styles as you need. All text will be concatenated continuously so that word wrap works – include spaces, commas etc as required. The way to make a carriage return is to make a new paragraph.
- Next you define a Text block with the TEXTBLOCK statement. This is where you can set the overall width of the block, anchor (left, centre, right or up



or down relative to the zero point). At this point you can also apply width and spacing and height factors to the text to suggest condensed or stretched text. You list the paragraphs that will be included in the text block, and the commas separating the names of the paragraphs are, in effect, carriage returns.

- These can all be done in the Master script, or in the script where the text will occur.
- Finally, you can use RICHTEXT2 to place the text in 2D and RICHTEXT to place it in 3D (along with a height parameter). If you intend to make 3D text, then the PARAGRAPH statement is a bit like a cPRISM_, you must include the material in the paragraph. A general material setting for the script will be ignored.

Note that 3D text generates a lot of polygons, so do not forget to add a TOLER statement. Build in a ZZYX height hotspot, and either do the same for the 3D hotspots for 'A' or rely on the 2D hotspots in the 3D.

```
!Rich Text Object: 2D Script
PEN gs_cont_pen
!Place the text
RICHTEXT2 0, 0, "AC_TEXTBLOCK_2"
HOTSPOT2 0,0, 1001,A,1+256 !Base
HOTSPOT2 A,0, 1002,A,2 !Move
HOTSPOT2 -100,0, 1003,A,3 !Vect

!Check size of Text Block and set Hotspots
nul=REQUEST ('TEXTBLOCK_INFO', "AC_TEXTBLOCK_2", wid, hit)
w=wid*A_/1000
h=hit*A_/1000
HOTSPOT2 w/2,-h/2
HOTSPOT2 w,-h
HOTSPOT2 0,-h

!Rich Text Object: 3D Script
MATERIAL matl
PEN gs_cont_pen
TOLER 0.01
!Place the text
RICHTEXT 0, 0, zzyzx,0,"AC_TEXTBLOCK_3"
HOTSPOT 0,0,0, 1001,zzyzx,1 !Base
HOTSPOT 0,0,zzyzx, 1002,zzyzx,2 !Move
HOTSPOT 0,0,-100, 1003,zzyzx,3 !Vect
```

!Richtext object: Master Script

!Scaling Factor between 3D and 2D
IF GLOB_CONTEXT=3 THEN sf=GLOB_SCALE ELSE sf=1

```
!----- Text Definitions-----
DEFINE STYLE{2} "AC_STYLE_1" "Arial Black", 4*sf,1
DEFINE STYLE{2} "AC_STYLE_2" "Courier New", 3,0
DEFINE STYLE{2} "AC_STYLE_3" "Times New Roman",4,0
DEFINE STYLE{2} "AC_STYLE_4" "Times New Roman",4,2
DEFINE STYLE{2} "AC_STYLE_5" "Times New Roman",4,1
```

```
PARAGRAPH "AC_PRG_5" 1, 0, 0, 0, 1
PEN 10
MATERIAL matl
SET STYLE "AC_STYLE_1"
"Test Text - explained"
ENDPARAGRAPH
PARAGRAPH "AC_PRG_6" 1, 0, 0, 0, 1
PEN 7
SET STYLE "AC_STYLE_2"
"This is a short example."
ENDPARAGRAPH
PARAGRAPH "AC_PRG_7" 1, 0, 0, 0, 1
PEN 1
SET STYLE "AC_STYLE_3"
"I want to set text to "
SET STYLE "AC_STYLE_4"
"Italic"
SET STYLE "AC_STYLE_3"
", and "
SET STYLE "AC_STYLE_5"
"Bold"
SET STYLE "AC_STYLE_3"
", and change fonts and "
PEN 166
SET STYLE "AC_STYLE_5"
"colours "
SET STYLE "AC_STYLE_3"
PEN 1
"frequently."
ENDPARAGRAPH
```

You define each 'paragraph' - as far as the next carriage return.

If it's going to be 3D, then specify material.

Predict all the styles you are going to use, and SET them in the PARAGRAPH statements.

This is a complex task, so you will only use it when you want a distinctive benefit, e.g. clarity or colour of text.

TEXTBLOCK assembles the paragraphs together in sequence, each comma represents a carriage return.

The syntax allows you to set block width, line spacing etc.

```
!Add the paragraphs to make the whole Text Block
TEXTBLOCK "AC_TEXTBLOCK_2" A*1000/GLOB_SCALE,
1, 0, 1, 1, 1,
"AC_PRG_5", "AC_PRG_6", "AC_PRG_7"
```

```
!-----3D Text Definition-----
DEFINE STYLE{2} "AC_STYLE_10" "Arial Black",
4*GLOB_SCALE, 1
```

!or use the scaling factor in the master script

```
PARAGRAPH "AC_PRG_8" 1, 0, 0, 0, 1
PEN 11
SET STYLE "AC_STYLE_1"
"Test Text - explained"
ENDPARAGRAPH
```

In this example, we use Style 1 because it supports 2D & 3D, but you can define styles separately for 3D if you wish.

```
TEXTBLOCK "AC_TEXTBLOCK_3" A*1000,
1, 0, 1, 1, 1, "AC_PRG_8"
```

Bendy Bar!

BENDiBAR is an object that can be put to many uses, but its primary use could be rebars in concrete, or perhaps handrails. It can be used for drainage and plumbing, or for neon lettering. It consists of a bar or tube that can be bent in several ways, with any degree of curvature, bending angle and twisting angle at each bend.

It is a useful exercise in demonstrating the simple but effective use of Parameter ARRAYS, and of making that array visible to the user. It makes a complex object vastly easier to write and user-friendly to manipulate in ArchiCAD. Bendibar goes on to be an outstanding example of graphical Hotspots and User interface design.

```
!Parameter script
sh1= '1_Single bar'
sh2= '2_Two-bar'
sh3= '3_Three-bar'
sh4= '4_Four-bar'
sh5= '5_Five-bar'
sh6= '6_Six-bar'
sh7= '7_Seven-bar'
VALUES 'shape' sh1, sh2, sh3, sh4, sh5, sh6, sh7

v1='You can use the parameter array table in the user'
v2='interface to enter numerical data, but its much'
v3='more fun to view the bendibar in the 3D window'
v4='and explore the function of the 3D Hotspots'
v5='to stretch, bend & twist the bar. Get it near-'
v6='enough right, then straighten and tidy in the'
table'
VALUES 'info' v1, v2, v3, v4, v5, v6
```

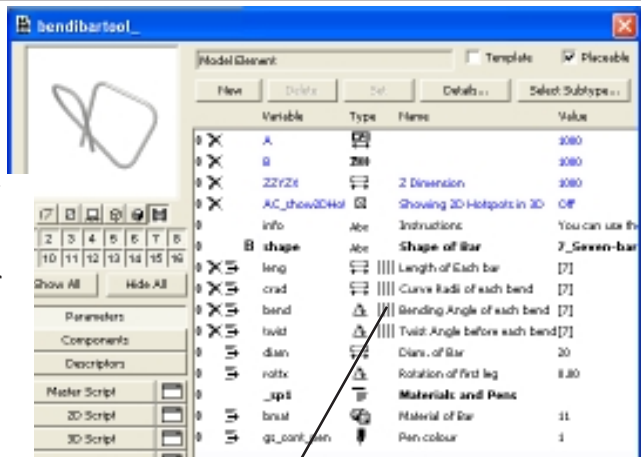
We can use the Parameter Script to build the pop-down menus, one for the choice of Bar and one for Instructions. These appear in the parameter table and in the UI. Use this method of building the Values lists.

Housekeeping before we do the 3D

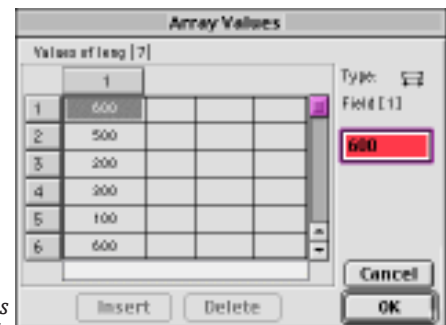
The primary job is to parse the Value List – this is done in the Master Script. I have shown two methods of doing it. The SPLIT() command extracts from the string a single digit value of 'shp' in one single line of smart code – a lot easier than a whole lot of IF statements. The **Parameter checks** build in a number of safeguards, in case of problems such as the curve radius being smaller than the tube radius. I usually write the object first, get it working, then discover the things that can upset it with user-abuse, and write in safeguard routines to catch the snags.

3D Script

Because this uses arrays, the final 3D routine is incredibly short. The FOR... NEXT loop does all the work and the value of 'k' in each run through the loop decides which section of the rod and which bend will be created. Get one run of it working, and then a second. Once that is proven, you can increase the size of the loop. 6 or 7 bends in a reinforcing bar would be as many as you could want, but you could add more if you wish. e.g. for something like neon lettering. The routine follows a similar idea to that used in the first Handrail in the Cookbook – a simple CYLIND and ELBOW. The



You have to click on this button to make the parameter adopt the form of an Array. Then you can use "Set" to set the values of the array, install some typical values to get the object started. do this for bar length, curve radius, bending angle and twist angle. In this table, we click the Hide button for the parameter arrays because we need a User Interface to make a better way to give the user access to the array data.



!Bendibar Master Script

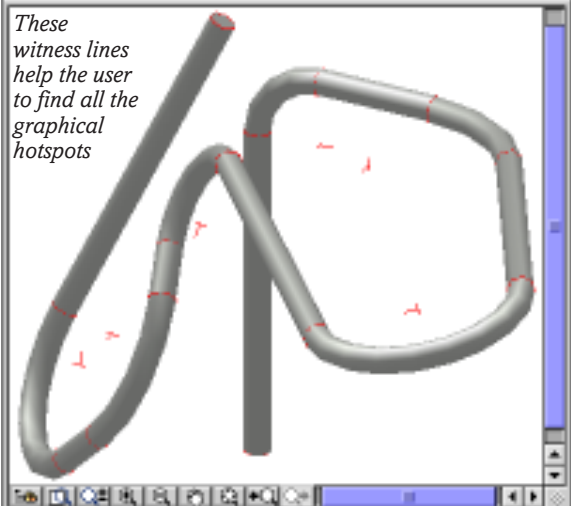
```
!This is the long hard
!way to parse a value list
!IF shape='1_Single bar' THEN shp=1
!IF shape='2_Two-bar' THEN shp=2
!and so on and so on.....

!If all you want is a flag, use SPLIT() instead
nul=SPLIT(shape, "%n", shp) !creates value 'shp'

!Parameter Checks
brad=diam/2 !Bar radius
FOR k=1 TO shp-1
IF crad[k]<brad THEN crad[k]=brad
IF bend[k]<0 THEN
twist[k]=twist[k]+180
bend[k]=ABS(bend[k])
ENDIF
IF leng[k]<0 THEN leng[k]=0
NEXT k
IF leng[shp]<0 THEN leng[shp]=0
```

This is the long way to parse the Valuelist and could be shortened to IF shape=sh1 THEN shp=1: but SPLIT is still shorter.

This is idiotproofing, using a FOR NEXT loop



twist added to the bar routine adds 3 dimensional quality to the bar.

2D Script

This tool could be bent into such wierd shapes that it is not practical to work out a 2D script for it. You will have to depend on PROJECT2 for the 2D. However, as a bunch of these bars could get mighty confusing, it helps a great deal to write a script to place cross hairs at the start of the tool, and to a starting Hotspot. It might help to remove Hotspots on the Bounding box, as these could become a dense forest of hotspots when you built a reinforcement cage. In the GDL, we have the option to hit the “Details” button and ask GDL to give us a Bounding Box in 2D.

!Bendibar 2D Script

```

PEN gs_cont_pen
PROJECT2 3,270,2
br=brad*1.5 !crosshair size
!Crosshair, hotspot at start
HOTSPOT2 0,0
LINE2 br,0,-br,0
LINE2 0,br,0,-br
ARC2 0,0,brad,60,120
ARC2 0,0,brad,150,210
ARC2 0,0,brad,240,300
ARC2 0,0,brad,330,390

```

Because it will be Project2, we will get some small extraneous lines from the 3D Graphical Hotspots. You could write an IF routine in the 3D script to hide these.

!Bendibar 3D Script

```

MATERIAL bmat
PEN gs_cont_pen
TOLER 0.001
GOSUB 120:!Rotation at start
ROTx rotxx
n=1 !set up a small counter
GOSUB 100:!Hotspot routine first tube
CYLIND leng[1],brad
FOR k=1 TO shp-1
  ADDz leng[k]
  GOSUB 140:!Hotspot routine Twist
  ROTz twist[k]
  GOSUB 110:!angle bend
  GOSUB 150:!change radius

  ELBOW crad[k],bend[k],brad
  ADDx crad[k]
  ROTy bend[k]
  ADDx -crad[k]
  n=k+1 !increment 'n'
  GOSUB 100:!Hotspot routine length each tube
CYLIND leng[k+1],brad
NEXT k
DEL 5*(shp-1)
DEL 1 !Remove the ROTxs
END:!-------

```

Set up Material, Pen and Curve tolerance. Subroutines 120 and 100 are graphical hotspot routines, so if you are making this, leave these out till you have got the basic tube working.

The first Cylinder is built. Then use the loop to build the subsequent Elbows. The 'n' counter is always one ahead of the 'k' counter, so you can use Subroutine 100 for both the present elbow and the next cylinder.

Make the Graphical Hotspots

Building the tube is comparatively easy compared with the Hotspots. But once you have built one, there is a lot of Copy and Paste – it gets easier with each one. See the earlier sections on Graphical Hotspots.

```

100:!Hotspot routine length of tubes
hsid=hsid+1 !base
HOTSPOT 0,0,0,hsid,leng[n],1
hsid=hsid+1 !moving
HOTSPOT 0,0,leng[n],hsid,leng[n],2
hsid=hsid+1 !reference
HOTSPOT 0,0,-leng[n],hsid,leng[n],3
RETURN
110:!Hotspot routine angle bend
LIN_ crad[k],-0.005,0,crad[k],0.005,0
ADDx crad[k]
hsid=hsid+1 !base
HOTSPOT -crad[k],0,0,hsid,bend[k],4
ROTy bend[k]
hsid=hsid+1 !moving
HOTSPOT -crad[k],0,0,hsid,bend[k],5
DEL 1
hsid=hsid+1 !centre
HOTSPOT 0,0,0,hsid,bend[k],6
hsid=hsid+1 !swivel
HOTSPOT 0,1,0,hsid,bend[k],7
DEL 1
RETURN
120:!Hotspots Rotation at the start
hsid=hsid+1 !base
HOTSPOT 0,0,leng[1]+crad[1],hsid,rotxx,4+128
ROTx rotxx
hsid=hsid+1 !moving
HOTSPOT 0,0,leng[1]+crad[1],hsid,rotxx,5
DEL 1
hsid=hsid+1 !centre
HOTSPOT 0,0,0,hsid,rotxx,6
hsid=hsid+1 !swivel
HOTSPOT 1,0,0,hsid,rotxx,7
RETURN

```

Distance Hotspots are the easiest.

Rotational hotspots are more complex. Put a 'witness line' in to help you locate the spot in 3D space. The Move spot can be found with a SIN and COS routine, but it's easier just to make it the same as the Base spot, but rotated – easy!

Use the same rotation trick for the move spot

Make the User Interface

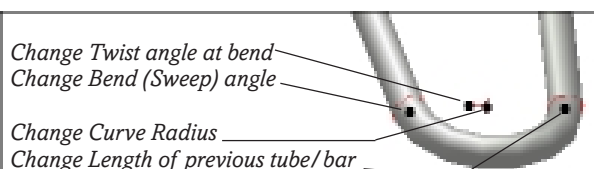
This uses UI_INFIELD{2} to build the spreadsheet like table. This script uses standard subroutines for most of the Outfields and Infields, and variables for indents and field lengths.

```

140:!Hotspot routine Twist angle
p=0.01 !1 3d pixel
hsid=hsid+1 !base
HOTSPOT crad[k]+p,0,0,hsid,twist[k],4+128
ROTx twist[k]
LIN_ crad[k]+p,0,0,crad[k]+0,0,0
hsid=hsid+1 !moving
HOTSPOT crad[k]+p,0,0,hsid,twist[k],5
DEL 1
hsid=hsid+1 !centre
HOTSPOT 0,0,0,hsid,twist[k],6
hsid=hsid+1 !swivel
HOTSPOT 0,0,1,hsid,twist[k],7
RETURN
150:!Hotspot routine change radius
hsid=hsid+1 !base
HOTSPOT 0,0,0,hsid,crad[k],1
hsid=hsid+1 !moving
HOTSPOT crad[k],0,0,hsid,crad[k],2
hsid=hsid+1 !moving
HOTSPOT -crad[k],0,0,hsid,crad[k],3
RETURN

```

How the Hotspots and witness lines work




```

!User Interface script for Bendibar
!This makes a spreadsheet like display

UI_DIALOG 'Bendibar Tool'
UI_SEPARATOR 2,2,442,264
UI_PAGE 1

!---Set up Indents and Field lengths---
stl=0 !Start Line
led=21 !Leading of lines
ln=0 !Line number
ind0=12 !Indent for outfields
ind1=112 !Indent for Infields
ind10=234 !indent for right column outfields
ind11=334 !indent for right column infields
ifl=100 !Infield length
ofl=100 !outfield length
ind00=38 !Indent for first outfield in table
iflt=100 !infield length in table
iflp=36 !infield length for Pen

!----Parameters of user's choosing----
UI_STYLE 0,1
ln=ln+1
string='Number of Tubes': GOSUB 100: !outfield
param='shape': GOSUB 103: !infield
string='Diam of Tubes': GOSUB 110: !outfield
param='diam': GOSUB 113: !infield
UI_STYLE 0,0
ln=ln+1
param='info': GOSUB 102: !infield
string='Rotation,1st leg': GOSUB 110: !outfield
param='rotx': GOSUB 113: !infield
ln=ln+1
string='Tube Material': GOSUB 100: !outfield
param='bmat': GOSUB 103: !infield
string='Pen Color': GOSUB 110: !outfield
param='gs_cont_pen': GOSUB 114: !infield

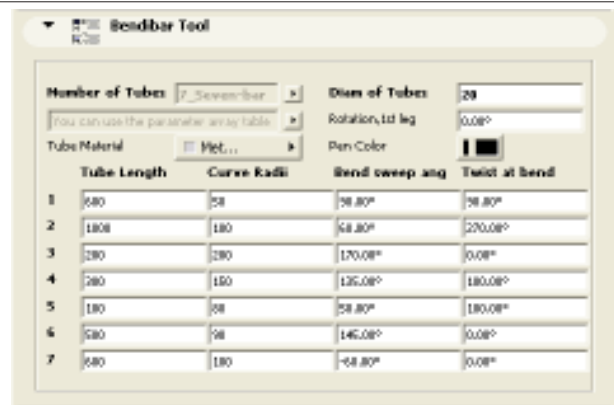
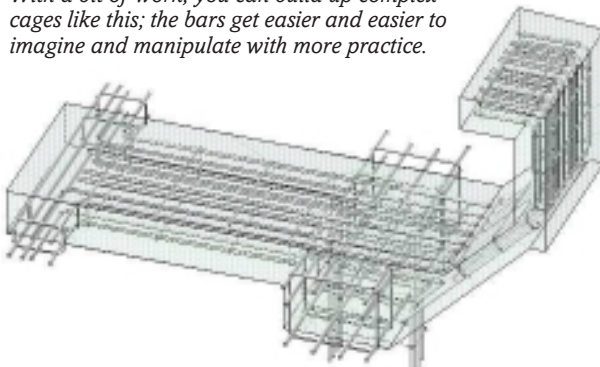
!-----
! Headers for the parameter arrays table !
!-----
ln=ln+1
UI_STYLE 0,1
UI_OUTFIELD 'Tube Length',ind00+iflt*0, led*ln,
            iflt,led
inden=inden+104
UI_OUTFIELD 'Curve Radii',ind00+iflt*1, led*ln,
            iflt,led
inden=inden+104
UI_OUTFIELD 'Bend sweep ang',ind00+iflt*2, led*ln,
            iflt,led
inden=inden+104
UI_OUTFIELD 'Twist at bend',ind00+iflt*3, led*ln,
            iflt,led

!----Titles for each Row----
FOR k=1 TO shp
UI_OUTFIELD STR(k,1,0),ind0,led*ln+led*k,12,led-1
NEXT k
UI_STYLE 0,0

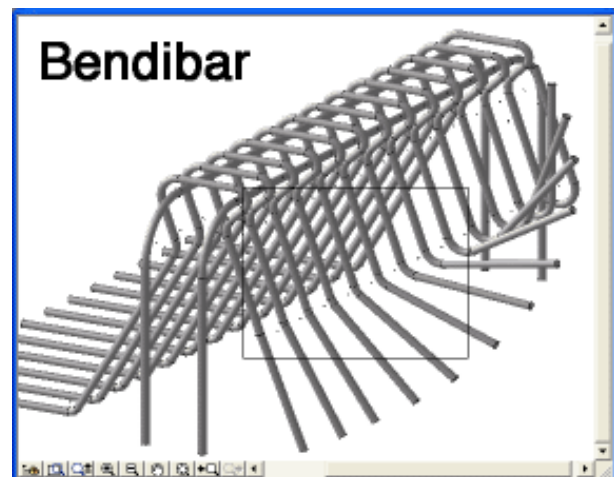
!----Build the actual Table----
FOR k=1 TO shp !Loop for all the table entries
!Tube lengths
UI_INFIELD{2} leng[k], ind00+iflt*0,
            led*ln+led*k-2,iflt,led-1

```

With a bit of work, you can build up complex cages like this; the bars get easier and easier to imagine and manipulate with more practice.



This spreadsheet like user interface is the only practical way that the user can see all the Bar parameters at once and edit them.



```

!Curve Radii
UI_INFIELD{2} crad[k], ind00+iflt*1,
            led*ln+led*k-2,iflt,led-1
!Bend angle at each junction
UI_INFIELD{2} bend[k], ind00+iflt*2,
            led*ln+led*k-2,iflt,led-1
!Twist angle before each junction
UI_INFIELD{2} twist[k], ind00+iflt*3,
            led*ln+led*k-2,iflt,led-1

NEXT k
END:=====
100:!Outfield normal length
UI_OUTFIELD string,ind0,stl+led*ln,ofl,ofh
RETURN
101:!Outfield extra long
UI_OUTFIELD string,ind0,stl+led*ln,ofl*2,ofh
RETURN
102:!Text infield extra long from left edge
UI_INFIELD param,ind0,stl+led*ln-2,iflt*2,ifh
RETURN
103:!Text infield normal length
UI_INFIELD param,ind1,stl+led*ln-2,iflt,ifh
RETURN
104:!Pen infield left
UI_INFIELD param,ind1,stl+led*ln-2,iflp,ifh
RETURN
110:!Outfield normal length
UI_OUTFIELD string,ind10,stl+led*ln,ofl,ofh
RETURN
113:!Text infield normal
UI_INFIELD param,ind11,stl+led*ln-2,iflt,ifh
RETURN
114:!Pen infield right
UI_INFIELD param,ind11,stl+led*ln-2,iflp,ifh
RETURN
115:!Text infield right column, entire space
UI_INFIELD param,ind10,stl+led*ln-2,iflt*2,ifh
RETURN

```