Electric Image Animation System

EIAS Version 9.0
Image File Format Specification

This is description of the format of the Image file which is used by the Electric Image Animation System to store both single frame images and multi-frame animations. The same format is also used to store texture and reflection maps (which can also be multi-frame). Following the file format description there is a discussion of the method by which the alpha byte in a 32 bit 1 color image is applied to the red green and blue channels to form a correct 24 bit image.

The image file header is as follows:

Name	Type	Size	Typical
Version	short	2	5
Frame Count	SInt32	4	>0

This is followed by the frames in the image file Each frame has a frame header as follows:

Frame Time	float	4	0.0
Frame Rect Top	short	2	0
Frame Rect Left	short	2	0
Frame Rect Bottom	short	2	480
Frame Rect Right	short	2	640
Frame Bit Depth	Byte	1	8
Frame Frame Type	Byte	1	1
Frame Pack Rect Top	short	2	0
Frame Pack Rect Left	short	2	0
Frame Pack Rect Bottom	short	2	480
Frame Pack Rect Right	short	2	640
Frame Packing Type	Byte	1	1
Frame Alpha Bits	Byte	1	0
Frame Size	SInt32	4	>=0
Frame Palette Entries	short	2	256
Frame Background Index	short	2	0

Note: There is at least 1 palette color for the background color even in direct color images.

Palette Color List

For each entry in the list:

Red	Byte	1	0...255
Green	Byte	1	0...255
Blue	Byte	1	0...255

The frame header is followed by either raw or run length encoded pixels.

Image Pixel Storage and Run Length Encoding:

Count is stored as a byte.
If 0<=Count<=127 the next value in the file is repeated Count+1 times.
If 128<=Count<=255 there follows a block of Count-127 values.

The value size is calculated as follows:

If the Bits Per Pixel value is <=8 then one or more pixels are stored for each 8 bit value. Otherwise, the value size is equal to Bits Per Pixel + Alpha Bits. Currently only 1, 2, 4, 8, 16 and 24 Bits Per Pixel are supported. The Alpha Bits must always be 0 except when the Bits Per Pixel is 24, in which case the Alpha Bits may be either 0 or 8. The last byte in each scanline of 1, 2 and 4 bit is padded with zero bits so that each scanline ends on a byte boundary.

Most images generated by EIAS are 32 bits deep (including an 8 bit alpha). Each pixel is stored as ARGB with 8 bits per channel.

The Frame Types are 0 for direct colors, 1 for indexed colors or 2 for Z-Buffer depth values. Color depths of 1, 2, 4 and 8 bits per pixel are always pixel type 1 and color depths 16 and 24 are always pixel type 0. 32 bit pixel depths are only used for floating point Z-Buffer depth values.

The Packing Type must be 0 for raw pixels or 1 for run length encoded pixels. Both modes are supported, however EIAS only creates run length encoded images.

Z-Buffers always have 4 palette entries. Since the palette is defined as three byte RGB values, four entries are used to store three 32 bit floating point values. These values are background depth, minimum depth and maximum depth in that order. The background depth is always INF.
The background is not used to compute the maximum depth.

Alpha Plane Graphics Discussion:

In 1 color graphics, the red green and blue components of the image pixels each require one byte of memory. Together they take up 24 bits of a 32 bit long word. In most graphics systems, four pixels are packed into 3 long words for efficiency. A better approach is to reserve a fourth byte for use as an alpha plane:

```
31                  24            16                  8                   0
|      alpha   |      red     |      green   |      blue    |
```

In C, a color is declared as follows: (This record is really declared in FACTStuff.h)

```
typedef struct {
       uchar a, r, g, b;
} ARGB
```

The alpha byte is used to mix foreground and background image planes. By using several layers of image planes, a composite image is formed. The layers can be animated to produce a 2 1/2 dimensional movie. In addition to simply specifying the holes in an image plane, the alpha byte can be used to simulate transparencies and high quality anti-aliasing. The formula for placing a foreground ARGB plane over a background RGB plane is quite simple:

```
back.r = (fore.a * fore.r + (255 - fore.a) * back.r) / 255;
back.g = (fore.a * fore.g + (255 - fore.a) * back.g) / 255;
back.b = (fore.a * fore.b + (255 - fore.a) * back.b) / 255;
```

When the alpha is 0, none of the foreground values are used and all of the background values are used. When the alpha is 255, all of the foreground values are used and none of the background values are used. For values between 0 and 255, a mixing of foreground and background values occurs. It is this mixing of values that is important for simulation of transparencies and anti-aliasing of image planes.

Only slightly more complicated is the formula by which two ARGB planes are combined to form a single ARGB plane. In this formula, the source plane is matted onto the destination plane and the alpha values are combined and preserved. Even if repeated composites of the same source image onto the destination image is performed, alpha buildup will not occur.

```
temp = source.a + dest.a;
if (temp > 255) {
        alpha = 255 - source.a;
        temp = 255;
} else
        alpha = dest.a;
if (source.a > dest.a)
        dest.a = source.a;
dest.r = ((SInt32)source.a * source.r + alpha * dest.r) / temp;
dest.g = ((SInt32)source.a * source.g + alpha * dest.g) / temp;
dest.b = ((SInt32)source.a * source.b + alpha * dest.b) / temp;
```

For compositing of two images in which the alpha is used to represent antialiasing, the destination alpha channel is generated as follows:

```
if (source.a > dest.a)
        dest.a = source.a;
```

For layering of two images in which the alpha is used to represent transprency, the destination alpha channel is generated as follows:

```
alpha = source.a + dest.a;
if (alpha >= 255)
        dest.a = 255;
else
        dest.a = alpha;
```

In a 1 color painting program that uses alpha plane graphics, the user may edit the alpha plane directly by using a 256 level mask. The mask is automatically created when the user paints the image. The brush's alpha value would default to 255 but could also be set to any other value. When an image is complete, the user could then modify the alpha plane with the same tools he used to paint the original image. This will provide a simple method for generating images with transparent areas.

The alpha plane may also be used to provide anti-aliased edges around the image planes. If an image contains a solid triangle, for example, the alpha plane values inside the triangle will be 255. Outside of the triangle, the values will all be 0. If, however, the alpha values along the edges of the triangle are chosen with intermediate values, the triangles edges will appear to blend evenly into the background without any noticeable aliasing.

The simplest method of producing the correct alpha values to achieve good anti-aliased edges is to draw the

image at 16 times the visible resolution. For every pixel in the destination image, sixteen pixels are drawn on an off-screen image in a 4x4 matrix. When the entire image has been constructed, the final image is calculated from the off-screen image with the following algorithm:

```
for (x = 0; x < xRes; x++)
        for (y = 0; y < yRes; y++) {
                aSum = 0;
                rSum = 0;
                gSum = 0;
                bSum = 0;
                for (i = 0; i < 4; i++)
                        for (j = 0; j < 4; j++) {
                                pixPtr = &offScrPixel[x * 4 + i, y * 4 + j];
                                aSum += a;
                                rSum += r * a;
                                gSum += g * a;
                                bSum += b * a;
                        }
                pixPtr = &imagePixel[x, y];
                                a = aSum / 16;
                                r = rSum / aSum;
                                g = gSum / aSum;
                                b = bSum / aSum;
                                }
```

A more complicated but more efficient method is to produce the alpha plane on the fly as the user is painting with lines, circles, polygons and other shapes. The program can draw these types of objects and calculate exactly how much of any pixel that the object covers. This value is then placed into the alpha plane. This method is better suited to a drawing or CAD program where all of the pictures are stored as a series of primatives.

There are many other uses for alpha planes. Some graphics cards that provide genlock abilities can mix 256 levels of the foreground graphics with the background video. An alpha plane would eliminate any matte lines from the resulting video and produce a better signal.

Look in FACTStuff.h for the following definitions and structures:

imageCreator
imageType
ImageHeader
ImageFrame