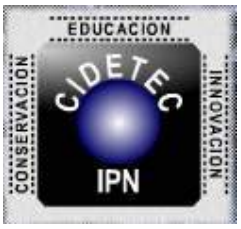
INSTITUTO POLITÉCNICO NACIONAL

Centro de Innovación y Desarrollo Tecnológico en Cómputo



DISEÑO E IMPLEMENTACIÓN DE UN MOTOR DE REALIDAD VIRTUAL ESCALABLE PARA ESCENARIOS 3D

Tesis de Maestría que presenta:

RODRIGO CADENA MARTÍNEZ

Para obtener el grado de:

MAESTRO EN CIENCIAS EN TECNOLOGÍA DE CÓMPUTO

Directores:

M. EN C. ISRAEL RIVERA ZÁRATE

M. EN C. JESÚS ANTONIO ÁLVAREZ CEDILLO

México, D.F., julio 2008

SIP-14

INSTITUTO POLITECNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISION DE TESIS

En la Ciudad de México, D.F. siendo las 13:00 horas del día 19 del mes de junio de 2008 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del CIDETEC para examinar la tesis de grado titulada:

DISEÑO E IMPLEMENTACIÓN DE UN MOTOR DE REALIDAD VIRTUAL ESCALABLE PARA ESCENARIOS TRIDIMENSIONALES

Presentada por el alumno:

CADENA	MARTÍNEZ	RODRIGO
Apellido paterno	materno	nombre(s)

Con registro: B 0 5 0 9 1 2

aspirante al grado de:

MAESTRÍA EN TECNOLOGÍA DE CÓMPUTO

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACION DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISION REVISORA

DR. VICTOR MANUEL SILVA GARCÍA
Presidente

M. EN C. JUAN CARLOS HERRERA LOZADA
Secretario

M. EN C. ISRAEL RIVERA ZARATE
Primer Vocal
(Director de Tesis)

M. EN C JESUS ANTONIO ALVAREZ CEDILLO
Segundo Vocal
(Director de Tesis)

M. EN C. MAURICIO OLGUÍN CARBAJAL
Tercer Vocal

M. EN C. EDUARDO RODRIGUEZ ESCOBAR
Suplente

EL PRESIDENTE DEL COLEGIO

DR. VICTOR MANUEL SILVA GARCÍA

S. E. P.
INSTITUTO POLITECNICO NACIONAL
CENTRO DE INNOVACION Y DESARROLLO TECNOLOGICO EN COMPUTO

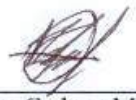
INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México el día 26 del mes Junio del año 2008, el (la) que suscribe Rodrigo Cadena Martínez alumno (a) del Programa de Maestría en Tecnología de Cómputo con número de registro B050912, adscrito a Centro de Innovación y Desarrollo Tecnológico en Cómputo, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de M. en C Israel Rivera Zárate y M. en C. Jesús Álvarez Cedillo y cede los derechos del trabajo intitulado Diseño e implementación de un motor de realidad virtual escalable para escenarios 3D, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección cidetec@ipn.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.



Rodrigo Cadena Martínez
Nombre y firma

Resumen

La primera fase de esta tesis consiste en mostrar que es la realidad virtual y algunas de sus características para posteriormente adentrarnos en visualizar que es un motor de realidad virtual y las ventajas que tiene sobre un sistema tradicional de despliegue de video. La segunda fase consiste en realizar la programación, la cual se divide en dos áreas de programación, la primera es el cliente-servidor, aquí es donde se establece la comunicación entre los dispositivos de despliegue de video y el dispositivo que los controla, los dispositivos de despliegue son computadoras que tienen conectadas un monitor, un cañón o cualquier otro dispositivo de despliegue que soporten y el dispositivo que controla es otra computadora llamada servidor, estas cuatro computadoras deben estar conectadas en red para poder establecer la comunicación entre ellos. Ya que se tiene resuelto el problema de comunicación se empieza a programar el motor, el cual cosiste en introducir escenarios a los clientes y sincronizarlos de tal forma que los tres parezcan un solo mundo y desde el servidor enviar la información de navegación y que los tres dispositivos lo hagan simultáneamente.

Abstract

The first part of this work consists in show what is virtual reality some of its characteristics and then we can visualize what is a virtual reality engine and the advantages that has over other systems of traditional video displaying. The second part consists on do the programation, which is divided in two areas of programation, the first one is about model client-server. Here is where the communications is established between the displaying video devices and the device which, is controlled them, this devices are computers that has connected a monitor, projector or any other device, and the device that control them is another computer called server. These four computers must be connected through a net to established a communication. Once we have solve the problem of communication we start to program the engine, which consist in load scenes to the clients and synchronize in a way that all the clients looks likes were in the same world and from the server send the navigation information and the three clients do it simultaneously.

Agradecimientos

A mi madre, Ilda Martínez Mancilla, por el apoyo que me ha brindado durante toda mi vida y en especial para este proyecto.

A mis hermanas Sandra, Ali y mi padre José, por todo el apoyo brindado.

A mi amigo Eloy al que agradezco su amistad y su ayuda en la realización de esta tesis.

Al M. en C. Mauricio Olguín Carbajal, por las ideas, conocimiento y ayuda brindada para la realización de esta tesis.

Al M. en C. Israel Rivera Zárate, por el tiempo, el conocimiento y el apoyo dedicados para la realización de esta tesis.

Al M. en C. Jesús Antonio Álvarez Cedillo, por la dedicación y apoyo vertidos en esta tesis.

Al M. en C. Juan Carlos Herrera Lozada, por brindarme su apoyo, ayuda y conocimientos en este proceso.

Al Dr. Victor M. Silva García, por su preocupación y apoyo para la culminación satisfactoria de esta tesis.

A todas los familiares, amigos y personas que me ayudaron, que me escucharon y apoyaron, gracias totales.

Gracias infinitas a la mejor institución a la cual he tenido el privilegio de pertenecer. Instituto Politécnico Nacional.

La realización de un proyecto no infiere que lo mas importante es tener mucha información o conocimiento, porque por si mismos no sirven de nada, toman significado hasta que alguien los aplica para diseñar, implementar, crear, desarrollar o inventar cualquier cosa que le sea de utilidad a la humanidad.

Índice General

Índice de Figuras

Índice de Tablas

Glosario

Applet. Aplicación en forma de ventana que utiliza Java.

Bidireccional. En ambas direcciones.

CAVE. Computer aumented virtual environment – computadora aumentada para escenarios vituales

Colisión. Choque de dos objetos

Consola. Dispositivo que contiene los instrumentos necesarios para su control y operación enfocado al entretenimiento.

Displays. Modo de despliegue por lo general en una pantalla.

HMD. Head Mounted Display – Casco de despliegue de video.

Joysticks. Periférico de forma de palanca que permite controlar el movimiento de los objetos o el cursor en la pantalla de una computadora.

Localhost. Forma de referirse a si misma, una computadora en un entorno de red.

Renderizado. Actualizar las imagines en un mundo virtual

SpaceBall. Dispositivo que contiene una bola, la cual funciona para operar el cursor en la pantalla de una computadora.

Capítulo I

INTRODUCCIÓN

1.1 Panorama general de la Realidad virtual

La realidad virtual es un sistema que genera entornos sintéticos en tiempo real, es decir, es el proceso de simular lo que para nuestros sentidos es real, en especial el sentido de la vista.
La realidad virtual puede ser de dos tipos: inmersiva y no inmersiva.

La realidad virtual inmersiva liga a un ambiente tridimensional creado por un programa tal como JAVA3D o VRML con un dispositivo físico que pueda lograr una inmersión, como son: cascos, guantes u otros dispositivos que capturan la posición y rotación de diferentes partes del cuerpo humano, con la intención de enviar esta información a la computadora y que esta realice una interacción entre el usuario y el mundo virtual.

La no inmersiva también utiliza a estos lenguajes de programación enfocados a la realidad virtual, la diferencia con respecto a los inmersivos es que no existen dispositivos adicionales a la computadora donde se esta visualizando.

Una característica muy importante que se debe tomar en cuenta es que los mundos simulados no necesariamente tienen que adaptarse a las leyes físicas naturales, razón por la cual la Realidad Virtual se presta muy fácilmente para ser aplicada en cualquier campo de la actividad humana aunque, sin duda alguna, habrá algunas aplicaciones mucho más apropiadas que otras, áreas del conocimiento que se interesen primero y mas profundamente en ella y otras que tarden un tiempo en asimilarla, pero será, sin dudarlo, una de las tecnologías preferidas en un futuro cercano.

La realidad virtual ha ido evolucionando de una manera vertiginosa en los últimos años, esto en gran medida ocasionado por la implementación y desarrollo de consolas de videojuegos que han explotado esta tecnología y que la han hecho llegar, a toda persona que posea una consola de este tipo.

A pesar de que la realidad virtual ha tenido un gran auge la mayoría de los dispositivos que existen actualmente, están basados en presentarle el mundo virtual al usuario utilizando una sola pantalla, esto debido a lo complejo que resulta sincronizar las imágenes en mas de una pantalla, por lo cuál la visualización final esta es a través de una sola pantalla.

Aunque existen dispositivos de despliegue de más de una pantalla estos suelen ser caros y no aplicados a fines didácticos o de investigación.

1.2 Planteamiento del problema

En la actualidad existen dispositivos de despliegue de video en tres pantallas denominados CAVE´s, pero la configuración de estos ya está definida y sólo se utilizan para realizar aplicaciones en ellos y no como una herramienta enfocada a la educación, tampoco se le puedan cambiar elementos que permitan adecuarse a la aplicación que se requiere, además de que no se puede modificar la parte de la programación del motor, ya sea para optimizarla o agregarle código que sea de utilidad, tampoco son escalables y esto es una gran desventaja debido a que si se requieren utilizar solo dos pantallas este sistema no lo permite, por tal motivo se tienen que utilizar las cinco pantallas siempre. Otro aspecto negativo que tiene es que sólo se puede visualizar a través de las pantallas de la cabina de inmersión y no a través de monitores o algún otro dispositivo de video.

1.3 Justificación

Los motores de realidad virtual en la actualidad son caros y no proporcionan las características necesarias para la cabina de inmersión que se esta construyendo en el CIDETEC, dadas estas circunstancias, es un proyecto viable debido a que puede ser implementado utilizando los recursos que tiene el Centro, puede ser escalable y versátil debido a que puede ocuparse para cualquiera aplicación que se desee.

Dentro de esta implementación el sistema de realidad virtual será capaz de sincronizar imágenes, evitar colisiones de objetos, lo cuál son dos de los elementos principales para poder tener una inmersión en los mundos virtuales, otro punto relevante para tener una mayor inmersión es tener un ángulo de visión similar al humano el cual es de 160°, por este motivo se pretende visualizar al mundo en tres pantallas.

Este proyecto puede servir a trabajos posteriores a través del laboratorio de realidad virtual, ya sea ampliando su capacidad, es decir, introduciendo mas dispositivos de despliegue o bien agregando funcionalidad por medio de dispositivos especiales como brazos robóticos, caminadoras, guantes, etc.

También puede ser utilizado para el desarrollo y programación de aplicaciones visuales más complejas. Además de un aspecto muy importante que es la enseñanza puesto que en el laboratorio se podrá ver en la práctica los conocimientos adquiridos en clase para los alumnos que lleven la materia de realidad virtual.

1.4 Objetivos

Desarrollar un motor de realidad virtual que permita sincronizar múltiples dispositivos de despliegue de video a través de una conexión en red.

- Desarrollar una aplicación de mundos virtuales los cuales se emplearán en el uso del motor de realidad virtual.
- Desarrollar una aplicación de comunicación en red para el procesamiento de la información que se desea transmitir.
- Diseñar la aplicación que sincronice y evite las colisiones de los objetos que se visualizarán a través de los dispositivos de despliegue de video.

1.5 Descripción del proyecto

Este proyecto consiste en un motor de realidad virtual, el cuál simula la visualización de un entorno virtual en tres dispositivos de despliegue de video, con la finalidad de que el usuario pueda sentirse inmerso en el escenario virtual y poder navegar en este, con la característica de tener un campo de visión de 120°, que se aproxima al rango de visión del humano que es en promedio de 160°, que es lo que se puede captar con el ojo cuando visualizamos el mundo que nos rodea. La figura 1 muestra una aproximación de cómo se debe ver el motor de realidad virtual.

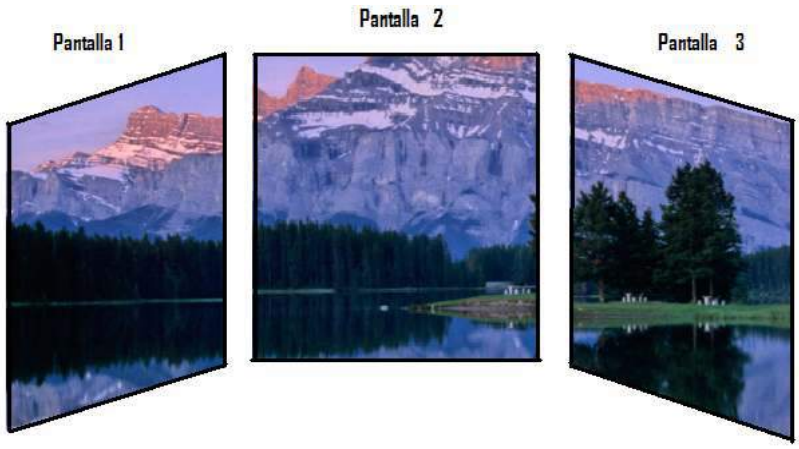


Figura 1. Visualización de motor de realidad virtual

La estructura general de este motor esta basada en un modelo cliente-servidor, donde el cliente y el servidor tendrán las siguientes características:

Características del Servidor.

- Sincronización de las imágenes
- Control de colisión de las imágenes
- Comunicación con los tres clientes, los cuales serán los visualizadores del mundo virtual

Características del Cliente.

- Contendrá el mundo virtual
- Comunicación con el servidor

La figura 2 muestra como se realiza la comunicación entre los clientes y el servidor.

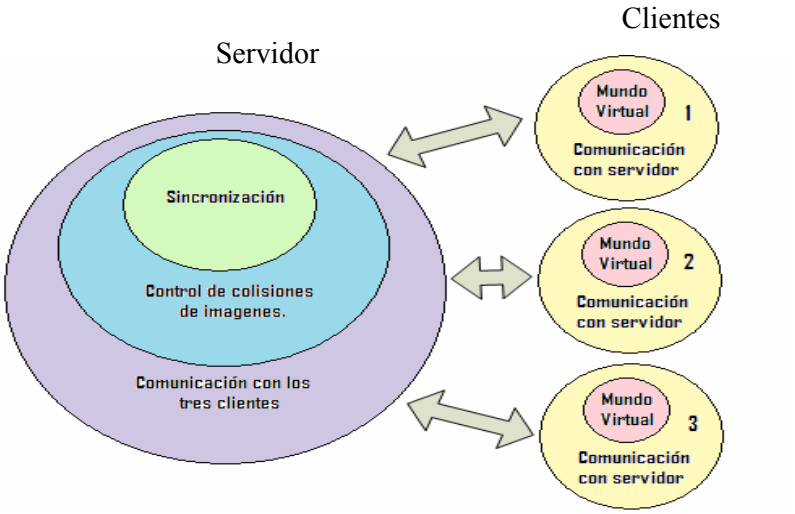


Figura 2. Diagrama de la comunicación entre los clientes y el servidor

El modelo cliente-servidor que se propone consta de tres clientes, los cuales se encargarán de la visualización del mundo virtual, uno tendrá la perspectiva derecha del mundo, otro la perspectiva central y el último la perspectiva izquierda, con la finalidad de abarcar el mayor rango de visión del usuario, debido a que con una sola pantalla se tiene un ángulo de visión de 40° [1]. El servidor se encargará de mandar los comandos de navegación, es decir, moverse hacia delante, atrás, izquierda y derecha, además se encargará de la sincronización, control de colisiones y la comunicación con los tres clientes, todo este conjunto de instrucciones es el motor de realidad virtual.

En términos de programación lo que va a realizar el modelo cliente servidor es lo siguiente:

- El servidor entra en la etapa de espera de la solicitud de los clientes.
- Posteriormente cuando esta se realice, el servidor comenzará a realizar procesos para poder establecer la comunicación.
- Cuando se establece la comunicación, se comienza el envío y recepción de la información, del servidor a los clientes, la cual podrá ser bidireccional.
- Finalmente cuando se termina de enviar y recibir la información, se procede a cerrar la comunicación.

En la figura 3 se muestra el diagrama a bloques.

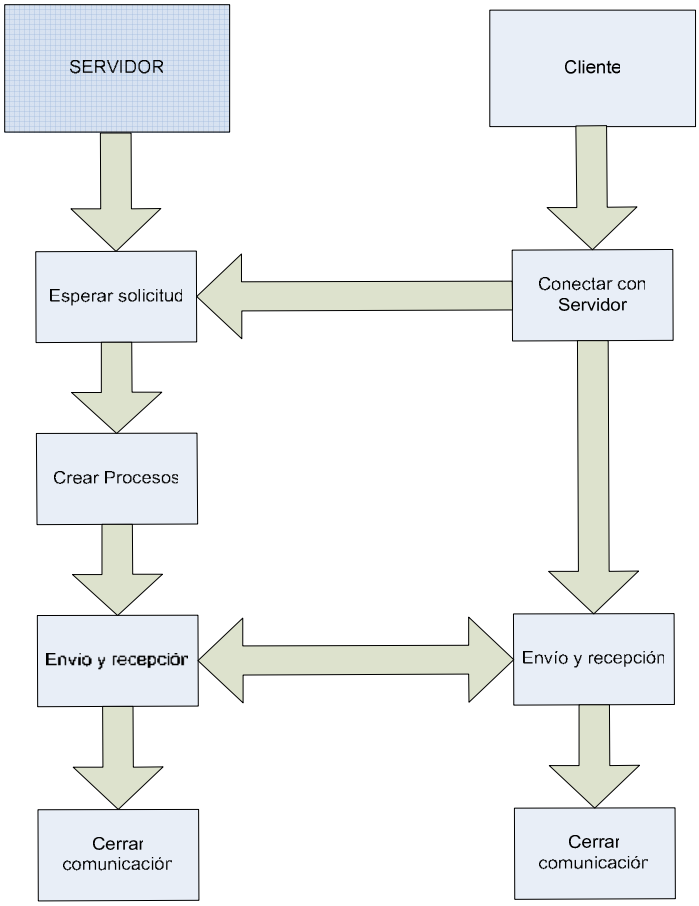


Figura 3. Diagrama a bloques detallado cliente-servidor

1.6 Estado del Arte

En esta sección se contemplan desarrollos hechos en otras universidades o países con la intención de ver que se esta haciendo en otras latitudes y no repetir un proyecto ya diseñado o mejorar alguno que ya exista.

1.6.1 Cabina de Inmersión

Actualmente el Centro de Innovación y Desarrollo Tecnológico en Cómputo CIDETEC, cuenta con una cabina de inmersión ubicada en el laboratorio de realidad virtual, debido a esto surge la necesidad de implementar un motor de realidad virtual para ser probado en dicha cabina, esta idea surge gracias a la propuesta del M. en C. Mauricio Olguín Carvajal con la finalidad de impulsar el desarrollo del laboratorio.

1.6.2 CaveUT

Existe en la Universidad de Illinois una cabina de inmersión llamada CaveUT [2], la cuál tiene cómo función principal lograr una inmersión total para el usuario. Para lograr un completo sentido de inmersión se pueden complementar varios proyectores en domos o semi-esferas (similares al cine panorámico o I-Max), a lo cual se le llamo "cueva" (CAVE: computer aumented virtual environment) iniciado por la investigadora venezolana Carolina Cruz Neira. Este sistema involucra proyectores en la parte posterior, izquierda y derecha que conforman un cubo de aprox. 2x2x2 metros. en que se sitúan uno o más personas. Esto implica una gran cantidad de espacio físico, de hecho algunos proyectores deben utilizarse con espejos. También es muy importante controlar la coordinación y brillo de las imágenes en las esquinas, para difuminarla y no percibir el cubo real sino un entorno continuo[3], En la figura 4 se muestra la cabina de inmersión CAVE.

Figura 4. Cabina de inmersión CAVE

Existen diversas aplicaciones que utilizan a esta cabina como un dispositivo ideal para lograr presentar al espectador una visión real de sus mundos, por ejemplo existe una plataforma de realidad virtual en la cuál las imágenes se consideran como arte en movimiento[4].

Figura 5. Visualización en una cabina de inmersión

1.6.3 Pabellón de Realidad Virtual

En el parque de diversiones Six Flags (Ciudad de México) se ubica una atracción llamada pabellón de la Realidad Virtual.

Esta atracción permite a una sala entera con 36 personas interactuar con el mundo virtual. Este proyecto tiene tres características:

- **Sala de inmersión.** Es una sala panorámica en la que un sistema de 3 vídeo proyectores de alta definición crean una imagen de 160° de apertura (el campo de visión humano), en un diámetro de 8 metros. También se realizo la instalación sonora, que contribuye a espacializar el sonido. Un sistema de infrasonidos instalado en cada asiento reproduce las sensaciones de choque y vibraciones, durante todo el recorrido. A los telespectadores se les proporcionan gafas para que obtengan una visión en relieve de muy alta calidad.

- **Interactividad.** Los asientos están dotados de joysticks que permiten a todos los espectadores convertirse en actores durante la experiencia. Se instala una verdadera dinámica de grupo. Cada espectador puede seleccionar, activar, navegar según su preferencia.

- **El motor.** El cual permite conseguir resoluciones de visualización muy altas de 4000 por 1000 puntos por pantalla, de 60 imágenes por segundo, siempre conservando una impresionante potencia de cálculo, Esta plataforma permite crear mundos virtuales de una riqueza y complejidad muy altos.

1.6.4 Cabina de Realidad Virtual enfocada a la vigilancia y seguridad

Es un proyecto realizado por universidades de Estados Unidos y Mexicanas en el se utiliza una cabina de inmersión en la cual los dispositivos de despliegue muestran todas las cámaras de seguridad de tal forma que cuando alguien enfoca la mirada en una determinada cámara, esta se maximiza para su mejor visualización[5].

1.6.5 Navegación Interactiva en escenarios complejos usando colisiones.

Es un proyecto desarrollado en la universidad de carolina del Norte en el cual diseñan escenarios complejos y navegan a través de ellos utilizando planeación de rutas y detección de colisiones[6].

1.6.6 Gráficas por computadora desplegadas en tres pantallas.

Este es un prototipo desarrollado en el Instituto Internacional de Investigación Tecnológica del Cairo, el cual consiste en desplegar un juego de video en tres pantallas simultáneamente[7].

1.7 Organización de la tesis.

Esta tesis esta dividida en cinco capítulos.

El capítulo primero habla brevemente de lo que es realidad virtual también se aborda el planteamiento del problema y el porque es viable su implementación, además de los objetivos y la descripción del proyecto, por último se mencionan los trabajos relacionados o similares a éste en otras instituciones y/o países.

El capítulo segundo es el marco teórico, abarca temas como son: la evolución, las características, tipos y dispositivos de la realidad virtual, además de especificar el modelo cliente – servidor y ver sus aplicaciones. Otro punto importante es el software que se utilizó, el cual fue Java3d y algunas de sus características mas sobresalientes.

El capitulo tercero explica la programación del cliente servidor y las diferentes versiones que se fueron realizando.

El capitulo cuarto trata sobre la programación final del motor de realidad virtual, incluyendo las versiones con el mouse y con el teclado, también muestra los segmentos de código mas representativos, la visualización del motor y los ejecutables de las versiones.

El capitulo quinto muestra las conclusiones que se determinaron en la realización de esta tesis.

Capítulo II

MARCO TEÓRICO

2.1 Evolución de la realidad virtual

En 1962 Morton Heilig, inventa el “Sensorama”, un videojuego que utilizaba sonidos, imágenes, movimiento, e incluso brisas artificiales para convencer al usuario de que estaba conduciendo una motocicleta por la ciudad de New York. Algunos años después en 1965 surge el concepto de Realidad Virtual. con la publicación del artículo titulado "The Ultimate Display", en el cual se describía el concepto básico de la Realidad Virtual.

En 1968 Ivan Sutherland y David Evans crean el primer generador de escenarios con imágenes tridimensionales, datos almacenados y aceleradores. En este año se funda también la sociedad Evans & Sutherland enfocada ala realidad virtual. Tres años después en 1971 se comienza a fabricar en el Reino Unido simuladores de vuelo con pantallas gráficas. Para el año siguiente General Electric, bajo mandato de la Armada Norteamericana, desarrolla el primer simulador computarizado de vuelo.

En 1977 Dan Sandin y Richard Sayre inventan un guante sensitivo a la flexión el cual era capaz de captar los movimientos de la mano al ser ésta flexionada.

En 1979 se presenta “Polhemus”, un sistema de posicionamiento basado en campos magnéticos diseñado por Eric Howlett. Jaron Lanier es uno de los primeros generadores de aparatos de interfaz sensorial, también colaboró en el desarrollo de aparatos de interface deRealidad Virtual, como guantes y visores.

En 1981 Thomas Furness desarrolló la "Cabina Virtual". Un año después presentó el simulador más avanzado que existe para su época, el VCASS (Visually Coupled Airborne Systems Simulator), contenido en su totalidad en un casco.

En 1983 Mark Callahan construyó un casco con una pantalla en su interior, en el Instituto Tecnológico de Massachusetts. La realidad virtual dejaba de ser desarrollada solo por compañias dando lugar alas universidades.

En 1984 Mike Mc Greevy y Jim Humphries desarrollaron el sistema VIVED (Representación de un Ambiente Virtual, Virtual Visual Environment Display) para los futuros astronautas en la NASA.

En 1985 Jaron Lanier y Jean-Jacques Grimaud fundan la institución VPL Research. Comienza la fabricación del DataGlove y en 1988 comercializarán el Eyephone HMD, los primeros sistemas de Realidad Virtual comerciales. Para el año de 1989 FakeSpace Labs comercializa a BOOM (Binocular Omni Oriented Monitor), una caja pequeña con dos monitores en el interior. Esta caja esta sujeta por un brazo mecánico que proporciona la información de posición y orientación, y puede ser movida por el usuario para navegar así por el mundo virtual.

En 1990 Surge la primera compañía comercial de software de Realidad Virtual, Sense8, fundada por Pat Gelband. Ofrece las primeras herramientas de software, portables a sistemas SUN. En 1991 Industrias W venden su primer sistema virtual. Richard Holmes, asignado por Industrias W, patento un guante de retroalimentación tangible.

En 1992 Un equipo de la Universidad de Illinois hizo una demostración del sistema CAVE(computer aumented- virtual environment).

En 1993 SGI anunció un motor de Realidad Virtual.

En 1995 Primera formulación del VRML uno de los lenguajes de programación para Realidad virtual mas populares. En 1998 Se crea el Consorcio Web3D para gráficos tridimensionales por Internet.

A partir del año 2000 hasta nuestros días la realidad virtual ha tenido un auge impresionante, particularmente en la rama del entretenimiento mas específicamente en el diseño de consolas para videojuegos como son:

- Play station 3
- X box
- Nintendo Wii

Este último con controles enfocados completamente a la realidad virtual.

Figura 6. Control Wii

2.2 Características de la realidad virtual

Las características de un sistema de realidad virtual, que lo distinguen de otros sistemas informáticos son:

- La inmersión, propiedad mediante la cual el usuario tiene la sensación de encontrarse dentro de un mundo tridimensional.
- Existencia de un punto de observación o referencia, que permite determinar la ubicación y posición de observación del usuario dentro del mundo artificial o virtual.
- Navegación, propiedad que permite al usuario cambiar su posición de observación.
- Manipulación, característica que posibilita la interacción y transformación del medio ambiente virtual.

2.3 Tipos de realidad virtual

Existe más de un tipo de Realidad Virtual y existen diferentes esquemas para clasificar los tipos de Realidad Virtual, pero el que se asemeja mas a la realidad es el esquema de clasificación fue descrito por Brill. El modelo de Brill tiene siete tipos diferentes de Realidad Virtual.

- **Inmersión en primera persona:** la Realidad Virtual inmersiva proporciona una experiencia inmediata en primera persona. El usuario se encuentra en el medio de una imagen que actúa como si fuera real en términos de percepción visual, auditiva o táctil. Una variación de la Realidad Virtual es la Realidad Aumentada donde se superpone una capa transparente de gráficos por computadora al mundo real para resaltar ciertas características y aumentar el conocimiento

- **A través de la ventana**: con este tipo de sistema, también conocido como Realidad Virtual de Escritorio, el usuario ve el mundo 3D a través de una "ventana" en la pantalla de su computadora y navega por el espacio con un dispositivo de control. Proporciona una experiencia en primera persona.

- **Mundos espejo**: en contraposición a los sistemas en primera persona descritos antes, los Mundos Espejo (Realidades Proyectadas) proporcionan una experiencia en segunda persona donde el usuario se encuentra fuera del mundo imaginario, pero es capaz de comunicarse con los objetos y personajes que están dentro de el.

- **Mundo Waldo**: este tipo de aplicación de Realidad Virtual es una forma de manejar animaciones digitales llevando puesto un traje con sensores que detectan movimiento.

- **Mundo de cámara**: se trata de un pequeño escenario de proyección de Realidad Virtual controlado por varios ordenadores que le da al usuario la sensación de mayor libertad de movimiento en un mundo virtual que con los sistemas inmersivos. Las imágenes son proyectadas en las paredes de la cámara y pueden ser vistas en 3D con lentes.

- **Entorno simulado en cabina**: este es otro tipo de tecnología en primera persona que es esencialmente una extensión del simulador tradicional, la diferencia radica en que se utilizan mas una pantalla para la visualización.

- **Ciberespacio:** es una realidad global artificial que puede ser visitada por mucha gente vía ordenadores en red, un ejemplo es la aplicación "Second Life", la cual consiste en simular la vida de una persona pero en un mudo virtual, en esta aplicación se conectan miles de personas en todo el mundo.

2.4 Dispositivos de realidad virtual

Los dispositivos de realidad virtual son todos aquellos que sirven como herramienta para facilitar la inmersión en un mundo virtual, éstos se enfocan principalmente en el sentido de la vista, pero también los hay para el tacto y el oído principalmente a continuación se mencionan los mas significativos.

- **Sistemas de posicionamiento y orientación.** La mínima información que se requiere en un sistema de Realidad Virtual inmersivo es conocer la posición y la orientación de la cabeza del usuario para mostrar las imágenes con el punto de vista correcto. Además se pueden monitorizar otras partes del cuerpo como las manos. Estos sistemas tienen seis grados de libertad, tres direcciones espaciales con sus correspondientes rotaciones. Las características más relevantes para tomar una desición con respecto a un sistema de este tipo son: la tasa de transferencia, la latencia, la precisión, la resolución y el rango.

- **Sistemas de seguimiento de ojos.** Los sistemas de posicionamiento permiten que las imágenes se dibujen con el punto de vista correcto. Sin embargo la agudeza visual del ojo no es la misma dependiendo de la distancia a la que esté el objeto, las imágenes lejanas no tienen por que tener la misma resolución y calidad que las más cercanas.

- **Dispositivos de entrada 3D.** Existen dispositivos que permiten el movimiento libre tridimensional más natural. Por ejemplo los joysticks con 6 grados de libertad.

- **Dispositivos de escritorio.** No ofrecen las capacidades de movimiento 3D y la sensación de inmersión de los anteriores pero son cómodos, sencillos y relativamente baratos. Productos de este tipo son el SpaceBall, los ratones clásicos, etc.

- **Dispositivos visuales.** La calidad del sistema de representación juega un importante referente para crear una sensación de inmersión. El sistema ideal debería tener una elevada resolución, alta frecuencia de actualización, amplio campo de visión y mucho brillo y contraste. Como sistemas principales se tienen las gafas 3D, los displays envolventes y los HMD's.

Figura 7. Lentes HMD

- **Dispositivos táctiles.** El ser humano puede percibir dos tipos de sensaciones táctiles: realimentación cinética, que se siente en los músculos y tendones o realimentación táctil que se siente en la piel (sensación de cambio de temperatura, texturas). La primera es más fácil de implementar que la segunda, en la que se puede emplear tecnologías como los nódulos vibratorios, burbujas inflables o fluidos que alteran su viscosidad con la acción de corrientes eléctricas.

- **Dispositivos de audio.** El sonido incrementa considerablemente la capacidad perceptiva del hombre. Aporta: percepción de objetos fuera del campo de visión, orientación espacial y la posibilidad de captar señales de alerta. Para esto el sonido ha de ser tridimensional.

2.5 Modelo Cliente-Servidor

Este modelo se utiliza para descentralizar los procesos o la información de una maquina, esto lo realiza mediante la implementación de los llamados servidores, los cuales tienen funciones muy específicas y los clientes hacen la petición de algún servicio en lugar de hacerlo ellos mismos, esto con la ventaja de optimizar velocidad y recursos de procesamiento. Este modo de procesamiento esta remplazando a gran velocidad tanto a los métodos de procesamiento basados en computadores centrales, como al proceso centralizado y otras formas alternativas del proceso distribuido de datos.

La definición de cliente y servidor son:

Cliente: El que solicita algún tipo de servicio o de información del servidor.

Servidor: Por lo general una computadora muy potente, dependiendo el servicio que ofrezca, que contiene información para que los clientes de red puedan manipularla.

2.5.1 Aplicaciones Cliente-Servidor

La característica central de la arquitectura cliente-servidor es la ubicación de las tareas del nivel de aplicación entre clientes y servidores. Tanto en el cliente como en el servidor el software básico es un sistema operativo que se ejecuta en la plataforma del hardware. Las plataformas y los sistemas operativos del cliente y el servidor pueden ser diferentes. En tanto un cliente particular y un servidor compartan los mismos protocolos de comunicación, y soporten las mismas aplicaciones.

El software de comunicaciones es el que permite operar al cliente y al servidor. El ejemplo principal es el TCP/IP.

El objeto de todo este software de soporte (comunicaciones y sistema operativo) es proporcionar una base para las aplicaciones distribuidas. Las funciones reales de la aplicación pueden repartirse entre cliente y servidor de forma que se optimen los recursos de la red y de la plataforma, así como la capacidad de los usuarios para realizar varias tareas y cooperar el uno con el otro en el uso de recursos compartidos. En algunos casos, estos requisitos dictan que el grueso del software de la aplicación se ejecute en el servidor y, en otros casos, la mayor parte de la lógica de la aplicación se ubica en el cliente.

Un factor esencial para el éxito de un entorno cliente / servidor es la manera en que el usuario interactúa con el sistema como un todo. De esta forma, el diseño de la interfaz de usuario de la máquina es vital. En la mayoría de los sistemas cliente-servidor, se da prioridad en ofrecer una interfaz de usuario gráfico que sea fácil de utilizar y de aprender, pero potente y flexible. Así pues, se puede pensar en un módulo de servicios de presentación en el puesto de trabajo del cliente, responsable de ofrecer una interfaz fácil de usar a las aplicaciones distribuidas disponibles en el entorno.

La siguiente figura 8 muestra un esquema básico del modelo cliente-servidor enfocado a un motor de realidad virtual.

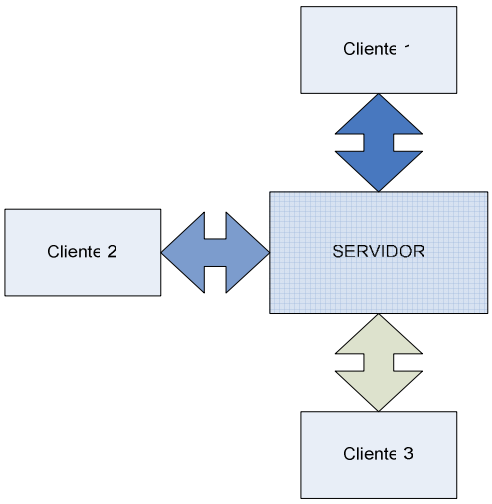


Figura 8. Diagrama a bloques cliente-servidor para el motor de realidad virtual

2.6 Características de software para realidad virtual

Las características que debe tener un software para la óptima programación de mundos virtuales deben ser:

- **Importación de modelos.** Capacidad de importar formas 3D para incorporarlas en una determinada aplicación.

- **Bibliotecas.** La mayoría de los programas de RV están provistos de bibliotecas 3D, con formas básicas o primitivas tales como cajas, esferas, conos, pirámides, etc. que sirven para generar formas compuestas. También cuentan con librerías de objetos complejos, texturas, etc. Es útil mencionar que estas librerías permiten que el diseñador reutilice muchas formas que simplemente decoran el ambiente virtual.

- **Operaciones Geométricas.** Consiste en las capacidades de manipular los objetos creados en una posición definida, y a partir de éstos se puede trasladar, rotar o escalar a otra posición. Eliminando la forma original o duplicándola. Se incluyen ocasionalmente operaciones booleanas y agrupamiento de formas, de modo que se puedan crear objetos
  compuestos operando o asociando distintas formas simples.

- **Nivel de Detalle.** Permite la optimización de la visualización de una escena virtual, al cambiar una forma con un alto nivel de detalle, por otra más simple, dependiendo de la distancia del punto de vista. De este modo, el objeto es reemplazado o se hace invisible si el observador está en movimiento o muy distante, y cuando está quieto y cercano, se despliega la forma más compleja.

- **Animación** Corresponde a la asignación de una traslación o rotación a un objeto en un periodo de tiempo, sincronizado con la navegación por el ambiente virtual. Estos movimientos generalmente equivalen a comportamientos del mundo virtual y pueden ser automáticos u originados por algún evento (interacción con el usuario o con otro objeto).

- **Articulado.** Se refiere a que los objetos puedan ser organizados en jerarquías; es decir, que partes componentes de un objeto posean propiedades de movimiento distintas a otras partes, pero supeditadas al total. Un ejemplo de esto son las ruedas de un automóvil, que pueden girar en un sentido, pero a su vez; debe desplazarse en la dirección del vehículo completo.

- **Detección de colisiones.** Es una característica que permite identificar cuando un objeto intersecta a otros, de modo que pueda ser obstaculizado el movimiento del objeto, de manera similar a si éstos fueran sólidos, como en el mundo real.

- **Propiedades físicas.** Adicionalmente se presenta una serie de atributos relacionados con características físicas, como masa o roce, reconocimiento de gravedad (movimiento vertical acelerado en proporción al tamaño o peso) e incluso de ascensión (salto sobre el objeto).

- **Color y texturización.** Asignación de colores a las superficies y utilización de texturas digitalizadas. Incluyendo propiedades de transparencia.

- **Fuentes de luz.** Definición de iluminación ambiental y focos de luz con cierta posición, orientación, intensidad e incluso colores propios.

- **Incorporación de audio.** Es la propiedad de asociarle a los objetos, de la aplicación virtual, un sonido que les corresponda en el mundo real. Una característica importante es controlar el volumen en relación a la distancia existente entre el objeto y el navegador.

- **Lenguajes de programación.** Esta propiedad corresponde a que el software disponga de comandos de control que dicten comportamientos de los objetos y manejen datos de entradas y salidas.

- **Manipulación de eventos.** Refleja la capacidad de activar un comportamiento al interactuar con un determinado objeto. Esto implica reconocer la posición y acción del usuario, interpretar una programación y modificar la geometría consecuentemente.

- **Configuración de dispositivos múltiples.** Consiste en permitir la incorporación de distintos dispositivos de entrada y salida de datos, como elementos de visualización o interacción del usuario (por ejemplo, cascos y guantes).

- **Mundos paralelos.** Se refiere a la generación de ambientes virtuales constituidos por sub-mundos, en los cuales el navegante puede interactuar al momento que ingrese a cada uno de ellos. Esto con el fin de disminuir la complejidad que se tendría si fuera un solo mundo completo y por ende optimizar el procesamiento.

- **Conectividad en red.** Permite que el mundo virtual pueda ser utilizado en una red, a través de diversos dispositivos o señales de entrada y salida, y además que permita la interacción de diversos usuarios en una misma aplicación. Un ejemplo de esto son los juegos multiusuarios.

2.7 Características de software para el motor de realidad virtual.

Específicamente para el motor de realidad virtual se debe de contar con las siguientes características con la finalidad de tener un desarrollo óptimo.

- **Importación de modelos.** Para el motor de realidad virtual se cargan objetos.
- **Detección de colisiones.** Es fundamental para el motor detectar colisiones debido a que es importante saber donde se puede o no avanzar.
- **Fuentes de luz.** Debe existir luz, de lo contrario no se visualiza nada en el mundo virtual.
- **Manipulación de eventos.** Se tienen que manipular los eventos del mouse y del teclado.
- **Configuración de dispositivos múltiples.** Se tiene que configurar los puertos por los cuales se envía y recibe la información.
- **Conectividad en red.** Es indispensable debido a que se utiliza un modelo basado en la conexión de red.

2.8 Software a utilizar para el motor de realidad virtual

Dentro de los lenguajes de programación enfocados a la realidad virtual, el mas utilizado es VRML, debido a su facilidad de programación y que los mundos programados se asemejan mucho a la realidad, este fue la primera opción, pero debido a que no cumple con algunos de los requisitos para la programación del motor, se decide utilizar **Java3D**, el cual cumple con todas los requerimientos de programación del motor, además de la potencia y portabilidad de Java.

2.9 Java 3D

La **API** (*Application Program Interface*) Java3D es una interfaz de programación utilizada para realizar aplicaciones y applets con gráficos en tres dimensiones. También, se integra correctamente con Internet ya que tanto los applets como las aplicaciones escritas utilizando Java3D tienen acceso al conjunto completo de clases de Java.

2.9.1 Objetivos de Java 3D

Los objetivos fundamentales de esta API son:

- Proporcionar un amplio conjunto de utilidades que permitan crear mundos en 3D.
- Proporcionar un paradigma de programación orientado a objetos de alto nivel para permitir a los desarrolladores generar sofisticadas aplicaciones y applets de forma rápida.
- Proporcionar soporte a cargadores en tiempo de ejecución. Esto permite que Java3D se adapte a un gran número de formatos, como pueden ser formatos específicos de distintos fabricantes o formatos de VRML.

2.10 Desarrollo de mundos virtuales con Java 3D

Un programa Java3D crea instancias de objetos y los sitúa en una estructura de datos denominada “Grafo de escena”. Este escenario gráfico es una composición de objetos 3D en una estructura de árbol que especifica completamente el contenido de un universo virtual, y cómo va ha ser renderizado. El API está diseñado con flexibilidad para crear universos virtuales precisos de una amplia variedad de tamaños, desde astronómicos a subatómicos. Java3D introduce además algunos conceptos que no se consideran habitualmente como parte de los entornos gráficos, como el sonido espacial 3D. Las posibilidades de sonido permiten proporcionar una experiencia más realista al usuario.

Un programa escrito en Java 3D se puede ejecutar como una aplicación o un applet, siempre y cuando se tenga un navegador capaz de desplegar programas en Java 3D.

El término de objeto visual se utiliza para referir a un objeto en el grafo de la escena (un cubo o una esfera por ejemplo). Por el contrario, el término objeto se utiliza únicamente para cuando se hacen instancias de una clase. El término contenedor se utiliza para aquellos objetos visuales que pertenecen a un grafo de escena El universo virtual se crea utilizando un grafo de escena, que es una estructura de datos, representada por medio de nodos y arcos. Un nodo es un dato y un arco es la relación que existe entre los datos. Los nodos en un grafo de escena definen a las instancias que se utilizan en las clases de Java3D. Los arcos representan a las dos posibles relaciones que existen entre las instancias de Java 3D.

La relación más común es la de padre-hijo. Un nodo grupo puede tener una gran cantidad de hijos pero únicamente un padre. Un nodo hoja puede tener un padre, pero ningún hijo. La otra relación es la referencia, que asocia a un objeto NodoComponente con un Nodo de la gráfica de escena. Un objeto NodoComponente define los atributos de la geometría así como los atributos utilizados para dibujar a los objetos visuales. Un grafo de escena se va construyendo como un grafo acíclico-directo, compuesto de nodos y arcos. En un grafo directo los arcos tienen una sola dirección, por lo que un grafo acíclico-directo tiene un solo sentido y, por tanto, los ciclos no están permitidos. Se comienza con un nodo y la dirección del grafo no puede regresar al mismo nodo. Existe una sola ruta desde la raíz del DAG hacia cada una de las hojas, a esta ruta se le conoce con el nombre de ruta del grafo de escena. El significado de cada ruta radica en que indica el estado en que se encuentra cada elemento de la escena, representado por medio de una hoja. Este estado se refiere a ubicación, orientación y tamaño de cada objeto. Cuando se plasman los objetos en la imagen Java 3D puede tomar ventaja de esta jerarquía, lo que permite que el despliegue de los objetos sea más ordenado. Por eso hay que señalar que el programador no tiene control sobre el orden en que se dibujarán los objetos.

En la figura 9 se muestra un ejemplo de la notación de grafos de escena y el ejecutable que se produce.

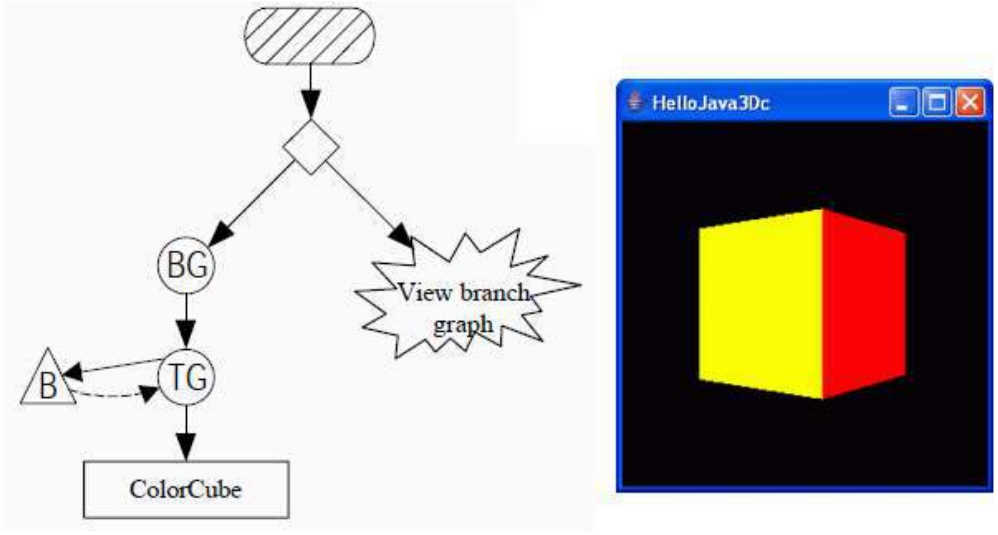


Figura 9. Grafo de escena y su ejecutable.

Capítulo III

PROGRAMACIÓN DEL CLIENTE-SERVIDOR

Se implemento el modelo cliente – servidor por su característica principal de ser un sistema de procesamiento distribuido, por recomendación del M. en C. Mauricio Olguín Carbajal, debido a su experiencia en el área de las redes de cómputo.

La definición de cliente y servidor para un motor de realidad virtual, cambian en su estructuración pero no en su esencia cuando se implementan.

Clientes. Existen tres clientes, cada uno de ellos con un escenario virtual cargado y cada cliente solicita la comunicación con el servidor a través puertos de comunicación.

Servidor. Se encuentra en espera, hasta que los tres clientes establecen una solicitud y posteriormente se realiza la comunicación.

Donde el servidor tendrá la función principal de sincronizar a los tres clientes y mandarles las instrucciones de navegación y los clientes recibir estas instrucciones y ejecutarlas en sus respectivos mundos virtuales.

El modelo cliente- servidor se basa en otro modelo más general llamado TCP/IP, el cual está enfocado a la comunicación entre redes de cómputo y se conforma por cuatro capas: Aplicación, Transporte, Internet y Acceso a red.

Cuando se hace referencia a un puerto de comunicación para poder enviar o recibir datos se ocupa la capa de Aplicación, cuando desea establecer la comunicación con una dirección IP, en este caso la del servidor, se realiza en la capa de Internet.

En el modelo cliente - servidor se utilizan las tres capas superiores, pero principalmente dos: Aplicación e Internet como se muestra en la figura 10.

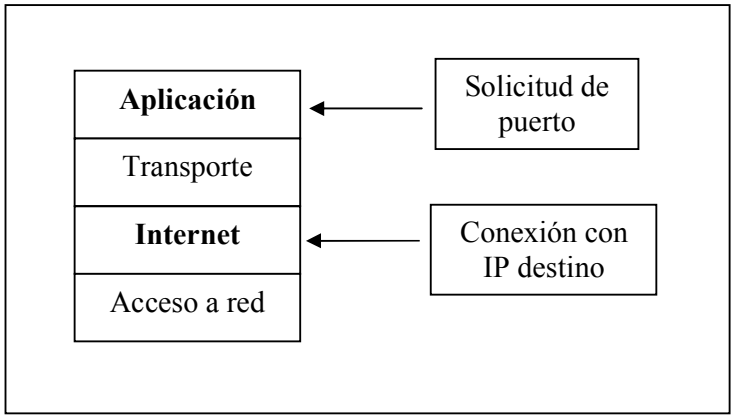


Figura 10. Modelo general TCP/IP y modelo cliente-servidor

Para lograr la programación del motor de realidad virtual es necesario establecer la comunicación entre todos las computadoras, para lograrlo se implementó la programación del modelo **cliente-servidor**.

El primer paso es realizar un cliente-servidor elemental, en el cual el servidor estará escuchando por un puerto hasta que le haga la solicitud el cliente y posteriormente se establece la comunicación y se envía un texto.

3.1 Servidor Versión 1.

A continuación se muestra la implementación de la clase del servidor versión 1:

```
public class servidor_v1 {
    static final int PUERTO=5000;
    public servidor_v1( ) {
        try {
            ServerSocket Servidor = new ServerSocket(PUERTO);
            System.out.println("\n\tEscucho el puerto " + PUERTO );
            Socket Cliente = Servidor.accept(); // Espera conexión  del cliente
            System.out.println("\n\tSirvo al cliente que se comunica por el puerto " + PUERTO);
            OutputStream aux = Cliente.getOutputStream();
            DataOutputStream flujo= new DataOutputStream( aux );
            flujo.writeUTF( "\n\t********* Se estableció comunicación entre el cliente y servidor **********\n ");
            Cliente.close();
            Servidor.close();
            System.out.println("\n\tCerre conexión con el cliente\n");
        } catch( Exception e ) {
            System.out.println( "No hay conexión con el cliente" );
           }
    }

    public static void main( String[] arg ) {
        new servidor_v1();
       }
}
```

Se declara una constante llamada PUERTO, en donde se guarda el valor del puerto por el cual el servidor atenderá al cliente, el puerto es el 5000, aunque se puede utilizar cualquiera de los puertos disponibles, mayores a 1024 debido a que los menores son utilizados por el sistema operativo.

```
static final int PUERTO=5000;
```

En el lenguaje de programación Java para al realizar operaciones de entrada salida (I/O), se utilizan las instrucciones try y catch. Lo que indica que intenta realizar el código que esta entre las llaves del try, pero si ocurre un error o eventualidad (llamada excepción), se realiza lo que esta en el catch. En este caso imprimiría que no hay comunicación con el cliente.

```
catch( Exception e ) {
  System.out.println( "No hay conexión con el cliente" );
 }
```

El uso de Sockets es relevante en la programación del motor, un socket es una instrucción que tiene como objetivo abrir la comunicación entre dos dispositivos mediante software, existen dos tipos de socket el del cliente que realiza esta tarea a través de un puerto desocupado de la computadora y una dirección IP del servidor con el que se requiere establecer la comunicación. El otro es el socket servidor, este permanece escuchando por un puerto hasta que una solicitud de un cliente sea recibida.

En el código se declara una variable **Servidor** de tipo *ServerSocket* y se asigna como parámetro el puerto por donde se requiere que el servidor establezca la comunicación. Se declara otra variable de tipo *Socket* llamada **Cliente**, su función es estar en un loop hasta que el cliente requiera comunicarse con el servidor.

```
ServerSocket Servidor = new ServerSocket(PUERTO);
System.out.println("\n\tEscucho el puerto " + PUERTO );
Socket Cliente = Servidor.accept(); // Espera conexión  del cliente
```

En cuanto se establezca la comunicación sale del loop e imprime que sirve al cliente y manda un flujo de salida, que posteriormente se convierte en un flujo de datos, que es lo que se envía al cliente para que éste imprima en su pantalla la cadena: "se estableció comunicación entre el cliente y servidor".

```
System.out.println("\n\tSirvo al cliente que se comunica por el puerto " + PUERTO);
OutputStream aux = Cliente.getOutputStream();
DataOutputStream flujo= new DataOutputStream( aux );
flujo.writeUTF( "\n\t********* Se estableció comunicación entre el cliente y servidor **********\n ");
```

Al utilizar sockets es necesario especificar que puertos se ocupan y cuando se termine la comunicación hay que especificar que ya no se utilizan para al aplicación en curso, para realizar éste proceso se tienen que cerrar los dos sockets que declaramos **Cliente** y **Servidor** con la siguiente instrucción**.**

```
Cliente.close();
Servidor.close();
```

En el programa se manda llamar la clase servidor_v1 desde la función principal llamada **main** para que pueda ser ejecutada.

```
public static void main( String[] arg ) {
    new servidor_v1();
   }
```

3.2 Cliente Versión 1.

Implementado el servidor versión 1 es necesario realizar lo mismo con el cliente, a continuación se muestra la implementación de la clase del cliente versión 1:

```
public class cliente_v1 {
    static final String HOST = "localhost";
    static final int PUERTO=5000;
    public cliente_v1( ) {
        try{
            Socket Cliente = new Socket( HOST , PUERTO );
            InputStream aux = Cliente.getInputStream();
            DataInputStream flujo = new DataInputStream( aux );
            System.out.println( flujo.readUTF() );
            Cliente.close();
            }catch( Exception e ) {
                  System.out.println( e.getMessage() );
                 }
    }

    public static void main( String[] arg ) {
        new cliente_v1();
    }
}
```

Para el cliente se tienen que declarar dos constantes PUERTO al cual se le asigna el puerto por el cual se comunica con el servidor y la otra es una constante tipo cadena llamada HOST a la cuál se le asigna la **dirección IP** del servidor, en este caso se le asigna la cadena "localhost" debido a que la prueba es en la misma maquina.

```
static final String HOST = "localhost";
static final int PUERTO=5000;
```

Se declara una variable de tipo *Socket* llamada **Cliente** y se le envía como parámetro el Host IP del servidor y el puerto por donde se comunicaran.

```
Socket Cliente = new Socket( HOST , PUERTO );
```

Cuando se establece la comunicación con el servidor, se utiliza la instrucción *getInputStream* para adquirir del puerto 5000 un flujo de entrada enviado por el servidor, el cual se convierte en datos y se imprime en pantalla. Para el envío de datos se utiliza el protocolo UDP porque no se requiere que la información sea validada y corregida, si existe algún problema se solicita que se envíe de nuevo la información.

```
InputStream aux = Cliente.getInputStream();
DataInputStream flujo = new DataInputStream( aux );
System.out.println( flujo.readUTF() );
```

Por último se cierra el puerto para liberarlo, debido a que si otra aplicación desea utilizarlo no lo podrá hacer aunque ya no se esté utilizando para esta aplicación. Para desocuparlo se utiliza la instrucción *close.*

```
Cliente.close();
```

Al ejecutar el servidor muestra primero que escucha el puerto 5000, cuando se ejecuta el cliente automáticamente se despliega el texto “”sirvo al cliente que se comunica por el puerto 5000” y por último se cierra la conexión con el cliente.

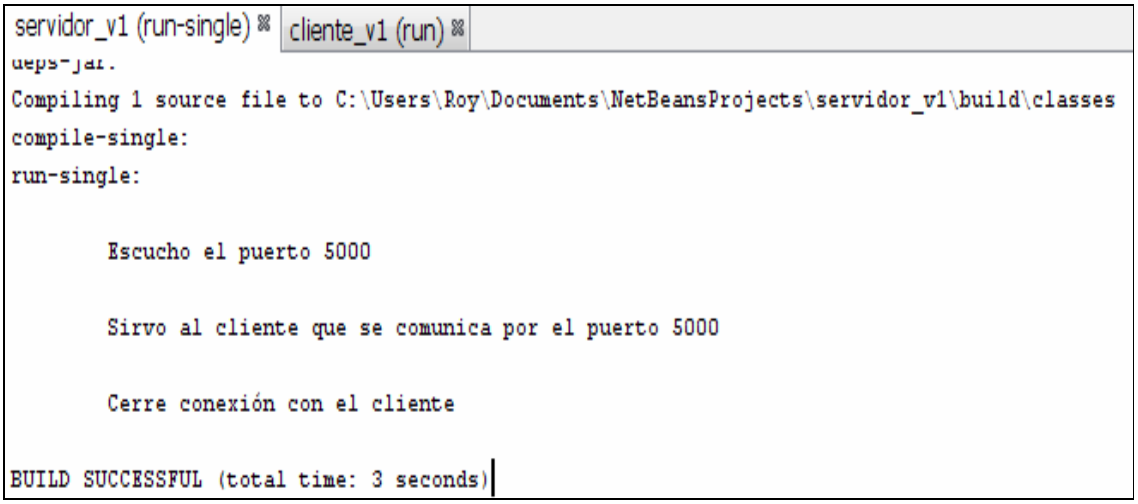


Figura 11. Ejecutable servidor_v1.

Al ejecutar el cliente se comunica con el servidor y este le envía el texto que se imprime en pantalla.

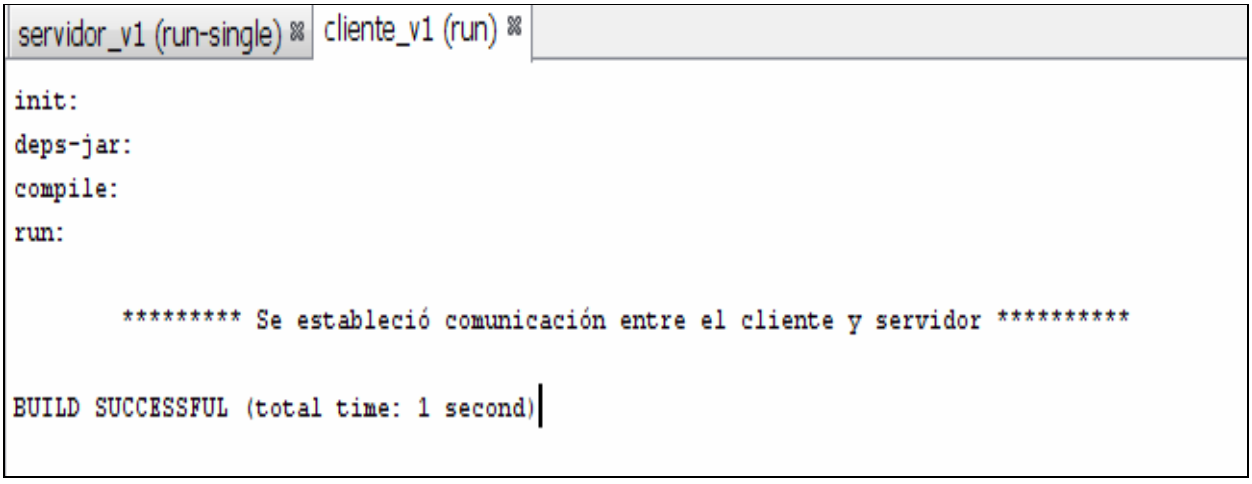


Figura 12. Ejecutable cliente_v1

3.3 Servidor Versión 2.

Esta versión con respecto a la anterior, establece la comunicación entre el servidor y tres clientes, la segunda versión del cliente lo que realiza es la comunicación del servidor con el cliente1 por el puerto 5000, con el cliente2 por el puerto 5001 y con el cliente3 por el puerto 5002

.

Esta es la implementación del código.

```
public class servidor_v2{
public servidor_v2( ) {
try {
        ServerSocket Servidor = new ServerSocket(5000);
        System.out.println("Escucho el puerto 5000" );
        Socket Cliente1 = Servidor.accept(); //  Espera cliente 1
        System.out.println("Sirvo al cliente 1");
        OutputStream aux = Cliente1.getOutputStream();
        DataOutputStream flujo= new DataOutputStream( aux );
        flujo.writeUTF( "Hola cliente 1" );

        ServerSocket Servidor2 = new ServerSocket(5001);
        System.out.println("Escucho el puerto 5001" );
        Socket Cliente2 = Servidor2.accept(); // Espera cliente 2
        System.out.println("Sirvo al cliente 3");
        OutputStream aux2 = Cliente2.getOutputStream();
        DataOutputStream flujo2= new DataOutputStream( aux2 );
        flujo2.writeUTF( "Hola cliente 2");

        ServerSocket Servidor3 = new ServerSocket(5002);
        System.out.println("Escucho el puerto 5002" );
        Socket Cliente3 = Servidor3.accept(); // Espera cliente 3
        System.out.println("Sirvo al cliente 3");
        OutputStream aux3 = Cliente3.getOutputStream();
        DataOutputStream flujo3= new DataOutputStream( aux3 );
        flujo3.writeUTF( "Hola cliente 3");
       int a=0;
       System.out.println("\nFinalizar, teclea enter? ");
       while(a != '\n'){
          a = System.in.read();
         }
       Cliente1.close();
       Cliente2.close();
       Cliente3.close();
       Servidor.close();
    }catch( Exception e ) {
            System.out.println( "No conexión" );
           }
}

    public static void main(String[] args) {
        new servidor_v2();
    }

}
```

En esta versión ya no se declara una constante para el valor del puerto debido a que se requieren tres puertos diferentes, aunque pudo haber sido la comunicación por un solo puerto, pero para realizar pruebas en una misma maquina es necesario utilizar tres puertos, así que para generalizar se utilizan los tres puertos.

En el código primero se declara una variable Servidor de tipo *ServerSocket* y como parámetro se le especifica el puerto 5000, también se declara una variable **Cliente1** de tipo *Socket* la cual estar esperando a que se comunique el primer cliente, en cuanto lo haga imprimirá en pantalla el servidor "Sirvo al cliente 1", después le enviara al cliente un flujo de datos para que e imprima en su pantalla "Hola cliente 1".

```
ServerSocket Servidor = new ServerSocket(5000);
System.out.println("Escucho el puerto 5000" );
Socket Cliente1 = Servidor.accept(); //  Espera cliente 1
System.out.println("Sirvo al cliente 1");
OutputStream aux = Cliente1.getOutputStream();
DataOutputStream flujo= new DataOutputStream( aux );
flujo.writeUTF( "Hola cliente 1" );
```

Este procedimiento se repite para el **cliente2** y el **cliente3**, la diferencia radica en el número de puerto a utilizarse, en el caso del cliente2 la comunicación es por el puerto 5001 y para el cliente2 es por el puerto 5002.

```
ServerSocket Servidor2 = new ServerSocket(5001);
System.out.println("Escucho el puerto 5001" );
Socket Cliente2 = Servidor2.accept(); // Espera cliente 2
```

```
ServerSocket Servidor3 = new ServerSocket(5002);
System.out.println("Escucho el puerto 5002" );
Socket Cliente3 = Servidor3.accept(); // Espera cliente 3
```

Por último se pide al usuario introduzca la tecla "enter" o salto de línea (\n), en cuanto lo haga se terminará la conexión con los tres clientes.

```
System.out.println("\nFinalizar, teclea enter? ");
while(a != '\n'){
   a = System.in.read();
  }
Cliente1.close();
Cliente2.close();
Cliente3.close();
Servidor.close();
```

3.4 Cliente Versión 2.

Esta versión consta de tres programas, uno por cada cliente, el primero es el siguiente.

```
class Cliente1 {
static final String HOST = "localhost";//dirección IP del servidor
static final int PUERTO=5000;
public Cliente1( ) {
    try{
        Socket Cliente = new Socket( HOST , PUERTO );
        InputStream aux = Cliente.getInputStream();
        DataInputStream flujo = new DataInputStream( aux );
        System.out.println( flujo.readUTF() );
        int a = 0;
        System.out.println("\nFinalizar, teclea enter? ");
        while(a != '\n')
           a = System.in.read();
        Cliente.close();
       } catch( Exception e ) {
            System.out.println( e.getMessage() );
           }
}

    public static void main(String[] args) {
        new Cliente1();
    }

}
```

En la clase **Cliente1** se declara un variable **Cliente** de tipo *Socket* a la cual se el envía cómo parámetro el la dirección IP del servidor y el puerto por el cual se establece la comunicación, después recibe el texto que le envía el servidor y lo imprime en pantalla, por último le pide al usuario que introduzca la tecla "enter" para terminar la conexión y el programa.

Para el **Cliente2** y **Cliente3** es el mismo código a excepción del puerto de comunicación con el servidor.

```
class Cliente2 {
static final String HOST = "localhost";//dirección IP del servidor
static final int PUERTO=5001;
```

```
class Cliente3 {
static final String HOST = "localhost"; //dirección IP del servidor
static final int PUERTO=5002;
```

Al ejecutar el **servidor_v2** se imprime en pantalla el puerto por el cual se encuentra escuchando y cuando el cliente establece la comunicación, imprime en pantalla a que cliente sirve y escucha por el siguiente puerto hasta que el segundo cliente establezca comunicación, cuando el segundo cliente se comunica, escucha por el último puerto y espera a que se termine la conexión al introducir la tecla "enter".

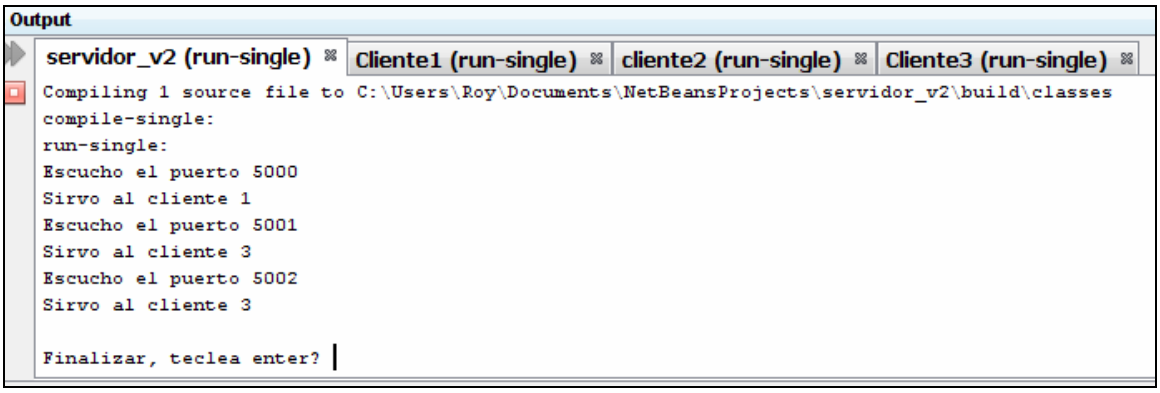


Figura 13. Ejecutable servidor_v2

Al ejecutar Cliente1 se imprime en pantalla el texto que le envía el servidor "Hola cliente 1".

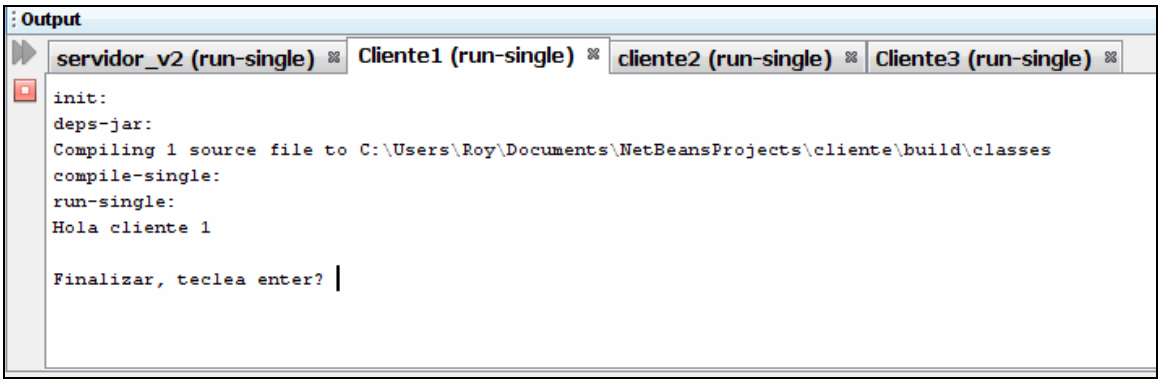


Figura 14. Ejecutable Cliente1

Al ejecutar Cliente2 se imprime en pantalla el texto que le envía el servidor "Hola cliente 2".

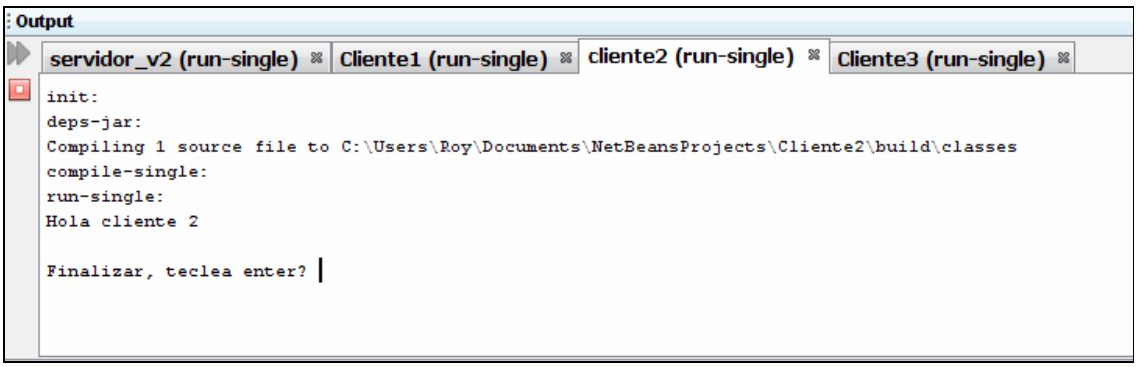


Figura 15. Ejecutable Cliente2

Al ejecutar Cliente3 se imprime en pantalla el texto que le envía el servidor “Hola cliente 3”.

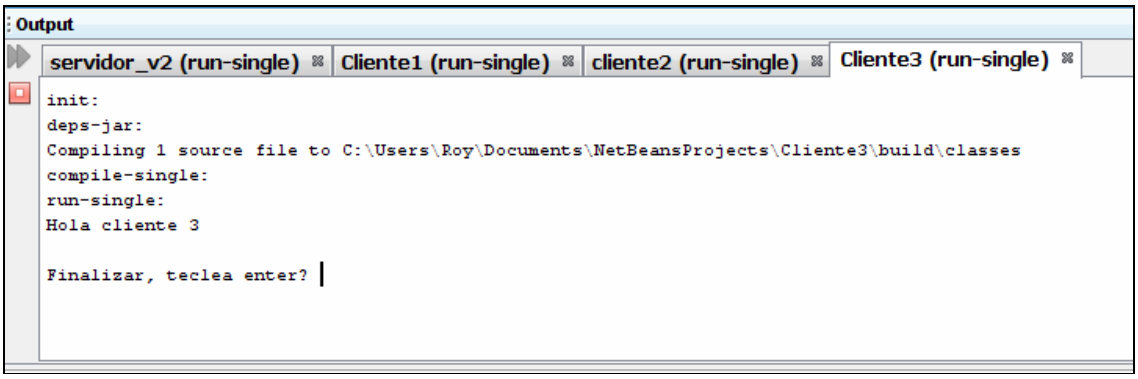


Figura 16. Ejecutable Cliente3

Lo que se logra con esta versión es la comunicación entre el servidor y los tres clientes, esta acción es fundamental para la programación del motor de realidad virtual.

3.5 Servidor Versión 3.

El siguiente paso es poder enviar información desde el servidor a los clientes, esta versión se realizó con un solo cliente debido a que en la versión 2 ya se comprobó que existe comunicación con los tres clientes.

El servidor permanece escuchando por el puerto 5000, hasta que el cliente establece la comunicación, cuando esto ocurre el servidor espera que el usuario teclee caracteres para poder enviárselos al cliente, para lograr esto se tiene que introducir el carácter y después la tecla “enter”, para finalizar la conexión se tiene que pulsar el carácter ‘s’.

La implementación del código es la siguiente.

```
class Servidor_v3{
   int v = 0;
   public Servidor_v3( ) {
       try {
            ServerSocket Servidor = new ServerSocket(5000);   //El servidor escucha por el puerto 5000
            System.out.println("Espero conexión del cliente \n" );
            Socket Cliente = Servidor.accept(); // Espera  a que se conecte el cliente
            System.out.println("Teclea: " );
            OutputStream aux = Cliente.getOutputStream();  // aux es una variable de tipo flujo de salida
            DataOutputStream flujo= new DataOutputStream( aux );  // flujo es una variable de dato de flujo
           while (v != 's'){
               v = System.in.read();                              // que obtiene de aux s
               flujo.write(v);
            }
            Cliente.close();
            Servidor.close();   //Al finalizar se tienen que cerrar los sockets
         }catch( Exception e ) {
               System.out.println( "No existe comunicación con el cliente" );
            }
    }
   public static void main(String[] args) {
         new Servidor_v3();
    }
}
```

Esta versión como las anteriores declara una variable de tipo *ServerSocket* en la cual se establece la comunicación por el puerto y después se declara una tipo *Socket* la cual específicamente establece la comunicación con el cliente, la variante con las versiones anteriores radica en que se envían caracteres, a través de un ciclo en el cual se escribe desde teclado un carácter e inmediatamente después se manda al cliente con la instrucción *write(v),* donde *v* es la variable donde se guardó el valor tecleado. Cuando el valor tecleado es una 's' el ciclo termina y se cierra la conexión con el cliente. Se utiliza este carácter para simbolizar la salida, pero puede ser cualquier otro.

```
while (v != 's'){
    v = System.in.read();
    flujo.write(v);
  }
  Cliente.close();
  Servidor.close();   //Al finalizar se tienen que cerrar los sockets
```

3.6 Cliente Versión 3.

Esta versión en lugar de recibir una cadena de caracteres e imprimirla en pantalla, lo que hace es leer el puerto y recibir carácter por carácter y posteriormente los imprime. Esta versión es de gran utilidad para la realización de los mundos virtuales remotos, debido a que estos se navegan en un principio a través del tecleado, y si ya es posible que el servidor envíe valores del teclado, estos datos serán los que reciba el mundo virtual para que empiece a navegar.

```
public class cliente_v3 {
    static final String HOST = "localhost";
    static final int PUERTO=5000;
    public char val=0;
    public BufferedReader in;

    public cliente_v3( ) {
        try{
            Socket Cliente = new Socket( HOST , PUERTO );
            InputStream aux = Cliente.getInputStream();
            DataInputStream flujo = new DataInputStream( aux );
            while(val != 's'){
                in = new BufferedReader(new InputStreamReader(Cliente.getInputStream()));
                val = (char)in.read();
                if(val != '\n')
                    System.out.println( val );
            }
            Cliente.close();
          }catch( Exception e ) {
                System.out.println( e.getMessage() );
                }
    }

    public static void main( String[] arg ) {
        new cliente_v3();
    }
}
```

Lo diferente de esta versión con respecto a las anteriores esta en que existe un ciclo el cual esta leyendo valores del puerto 5000 , después como estos valores que recibe son de tipo entero los convierte a carácter y verifica que no sea una tecla “enter” (\n), en caso de que no lo sean imprime el carácter y se queda en este ciclo hasta que el carácter que recibe es una ‘s’, cuando esto pasa se sale del ciclo y cierra la conexión con el servidor.

```
while(val != 's'){
    in = new BufferedReader(new InputStreamReader(Cliente.getInputStream()));
    val = (char)in.read();
    if(val != '\n')
        System.out.println( val );
}
Cliente.close();
```

Al ejecutar la versión **servidor_v3** lo primero que realiza es escuchar el puerto 5000 y cuando se conecta el cliente imprime en pantalla la palabra “Teclea:”, esto indica que el servidor ya puede comenzar a enviar los datos al cliente, los datos se verán tanto en la pantalla del servidor como en la del cliente.

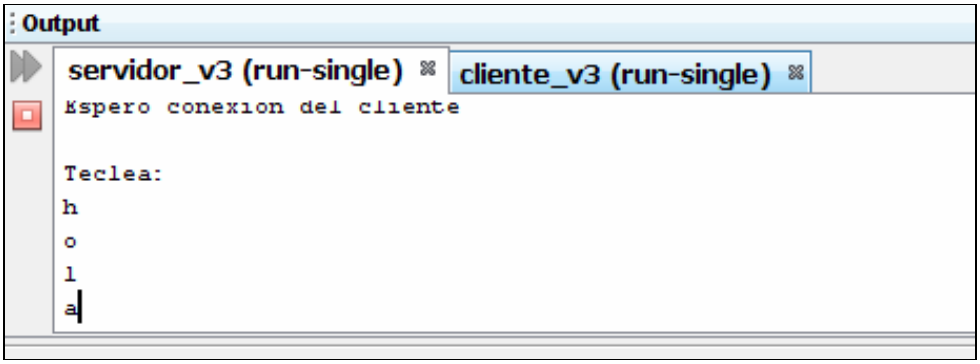


Figura 17. Ejecutable servidor_v3

Al ejecutar el **cliente_v3** establece la conexión con el servidor y espera recibir los valores que se tecleen en el servidor y estos se imprimen en la pantalla

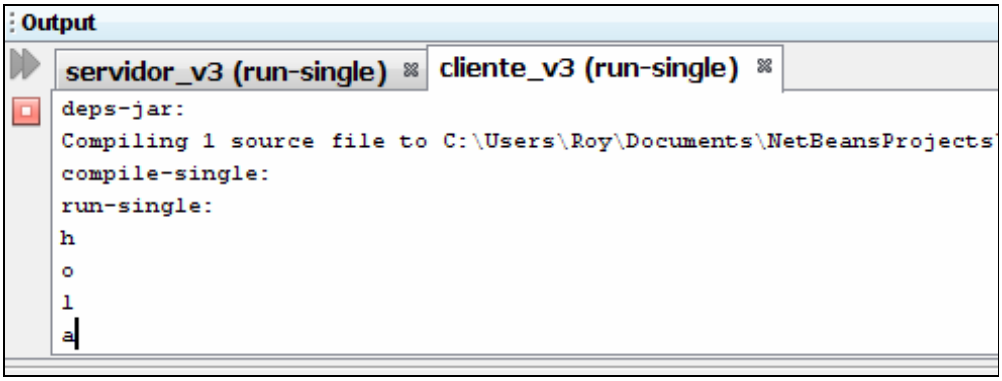


Figura 18. Ejecutable cliente_v3

Estas tres versiones constituyen la primera fase del motor de realidad virtual el cual es la comunicación del servidor con los tres clientes y poderles enviarles datos a los tres simultáneamente. En la siguiente fase el objetivo es poder montar en los clientes los escenarios virtuales y desde el servidor mandar las instrucciones de navegación y los tres mundos reciban estos datos y empiecen la navegación en cada escenario logrando un efecto visual como si fuera un solo mundo.

Capítulo IV

PROGRAMACIÓN DEL MOTOR DE REALIDAD VIRTUAL

Con el modulo de la comunicación integrado, el siguiente paso es empezar a programar el motor de realidad virtual, lo primero que se tiene que realizar es montar un escenario en cada uno de los clientes y ver si responden a los valores enviados por el servidor.

Para realizar esto se utiliza el Servidor versión 3, lo que se modifica es el cliente al que se le cargó un mundo virtual básico conformado por cubos instalados en frente, atrás, izquierda, derecha, arriba y abajo con respecto a la posición original del espectador

4.1 Cliente Versión 4.

Debido a que la implementación del código es extensa se explicaran las secciones mas relevantes y el código completo se anexará en la sección de apéndices.

La primera parte del código es la siguiente.

```
public Cliente3d() {
    Vector3f translate = new Vector3f();
    SimpleUniverse universe = new SimpleUniverse();
    BranchGroup objRoot = new BranchGroup();
    TransformGroup vpTrans = new TransformGroup();
    Transform3D T3D = new Transform3D();
```

En esta parte del código se declaran todas las variables que se van a utilizar en la clase **Cliente3d**, la primera se llama **translate** y es de tipo *Vector3f* la cual sirve para posicionar a un objeto en tres coordenadas (***X,Y,Z***), donde ***X*** representa el poderse mover de izquierda a derecha o viceversa. ***Y*** representa moverse arriba o abajo y ***Z*** representa el movimiento atrás y adelante. La segunda variable **universe** representa el crear el universo en donde se montará el escenario virtual. La tercera **objRoot** representa la rama principal, la programación en Java3D es por jerarquía, es decir, el universo es la raíz la cual tendrá ramas o hijos en el cual estará el escenario y a su vez el escenario tendrá hijos los cuales serán los objetos contenidos en el mundo en este caso se trata de cubos. **vpTrans** y **T3D** se utilizan para poderle hacer modificaciones a estos cubos en cuanto a rotación o translación.

```
TransformGroup cuboTG1 = new TransformGroup();
TransformGroup cuboTG2 = new TransformGroup();
TransformGroup cuboTG3 = new TransformGroup();
TransformGroup cuboTG4 = new TransformGroup();
TransformGroup cuboTG5 = new TransformGroup();
TransformGroup cuboTG6 = new TransformGroup();
```

En esta sección de código se declaran los seis cubos que se van a utilizar, estos son de tipo **TransformGroup** debido a que sufrirán modificaciones en ele programa. El motivo de utilizar estos cubos radica en la posición que tienen en el mundo y se estructuran de la siguiente forma: Partiendo del punto de vista del espectador, hay un cubo a la derecha, izquierda, enfrente, atrás, arriba y abajo. Como se muestra en la figura 19.

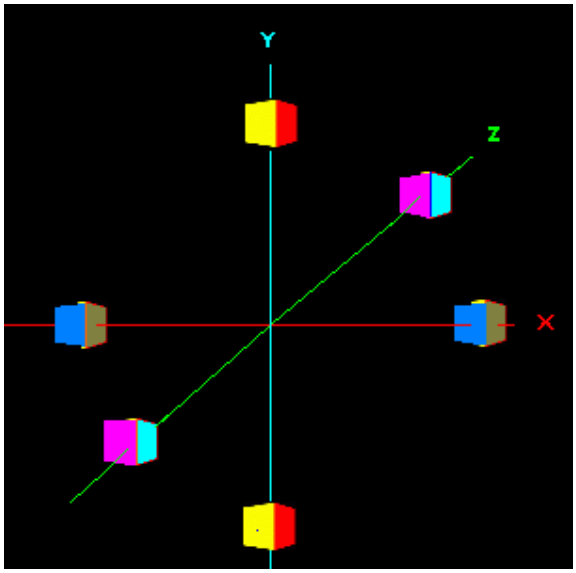


Figura 19. Posición de los cubos en el cliente_v3

```
try {
     J3dSocket = new Socket("localhost", 5000);
     in = new BufferedReader(new InputStreamReader(J3dSocket.getInputStream()));
    } catch (UnknownHostException e) {
         System.err.println("No encuentro el cliente");
         System.exit(1);
    } catch (IOException e) {
         System.err.println("No tengo comunicacion I/O" );
         System.exit(1);
    }
```

En la sección de código anterior, se realiza la comunicación con el servidor a través del puerto 5000 y se establece la comunicación, si no existiera comunicación se contemplan dos posibles fallos, el primero referente con el cliente y el segundo con el canal de comunicación.

```
translate.set(0.0f, 0.0f, -5.0f);
T3D.setTranslation(translate);
cuboTG1.setTransform(T3D);
cuboTG1.addChild(new ColorCube(0.5));
objRoot.addChild(cuboTG1);

translate.set(0.0f, 0.0f, 5.0f);
T3D.setTranslation(translate);
cuboTG2.setTransform(T3D);
cuboTG2.addChild(new ColorCube(0.3));
objRoot.addChild(cuboTG2);

translate.set(0.0f, -5.0f, 0.0f);
T3D.setTranslation(translate);
cuboTG3.setTransform(T3D);
cuboTG3.addChild(new ColorCube(0.4));
objRoot.addChild(cuboTG3);
```

```
translate.set(0.0f, 5.0f, 0.0f);
T3D.setTranslation(translate);
cuboTG4.setTransform(T3D);
cuboTG4.addChild(new ColorCube(0.5));
objRoot.addChild(cuboTG4);

translate.set(-5.0f, 0.0f, 0.0f);
T3D.setTranslation(translate);
cuboTG5.setTransform(T3D);
cuboTG5.addChild(new ColorCube(0.5));
objRoot.addChild(cuboTG5);

translate.set(5.0f, 0.0f, 0.0f);
T3D.setTranslation(translate);
cuboTG6.setTransform(T3D);
cuboTG6.addChild(new ColorCube(0.5));
objRoot.addChild(cuboTG6);
```

En el fragmento anterior de código, se realiza la implementación de los objetos del mundo, primero se les asigna el lugar el cual ocuparan en el mundo con las coordenadas (X,Y,Z), después se aplica esta translación a los objetos, ahora se cargan los cubos, los cuales tendrán un color diferente en cada cara para poder identificarlos a la hora de rotarlos o trasladarlos y por último se cargan a la rama principal para que puedan visualizarse.

```
public void processStimulus(Enumeration criteria){
 try {
       val = in.read();
     } catch (IOException ex) {
           System.out.print("No recibi valor\n");
         }
```

En Java3D existe una clase llamada **processStimulus** y es activada cuando ocurre algún evento en este caso va a ser cuando pase determinado tiempo, lo primero que realiza es la lectura de un buffer a través de la variable **in** y se le asigna a **var**, si ocurre alguna eventualidad imprimirá en pantalla “No recibí valor”.

```
if(val == 's'){
      try {
           J3dSocket.close();
           System.out.print("Cierro socket\n");
           System.exit(0);
       } catch (IOException ex) {
             System.out.print("No se pudo cerrar el socket\n");
          }
  }
```

Si se recibe el caracter ‘s’ esto indica que el servidor desea terminar la comunicación, lo que se hace es cerrar el socket con la instrucción *close* y se sale del programa con la instrucción *System.exit(0)*.

```
if(val == '4'){
     angleY -= 0.1;
     rotation.rotY(angleY);
    }
else if (val == '6'){
          angleY += 0.1;
          rotation.rotY(angleY);
       }
     else if(val == '8'){
            angleX -= 0.1;
            rotation.rotX(angleX);
        }
        else if (val == '2'){
                angleX += 0.1;
                rotation.rotX(angleX);
             }
            else if(val == '5'){
                dist += 0.1;
                traslacion.set( 0.0f, 0.0f, dist);
                T3D.setTranslation(traslacion); // set as translation
                targetTG.setTransform(T3D);
               }
               else if (val == '0'){
                    dist -= 0.1;
                    traslacion.set( 0.0f, 0.0f, dist);
                    T3D.setTranslation(traslacion); // set as translation
                    targetTG.setTransform(T3D);
                  }
if(val == '4' || val == '8' ||val == '6' ||val == '2'){
        targetTG.setTransform(rotation);
      }
```

En esta sección de código se describe como se moverán los cubos, utilizando el sistema de referencia numérico para poder indicar los movimientos. El ‘4’ indica movimiento a a la izquierda, el ‘6’ derecha, ‘8’ arriba, ‘2’ abajo, ‘5’ adelante y ‘0’ atrás.

En el código la implementación es un poco más compleja ya que cuando recibe los valores ‘4’ o ‘6’ en realidad lo que se hace es rotar en el eje Y la perspectiva del observador. Cuando se reciben los valores ‘8’ y ’2’ la rotación ocurre en el eje X. Cuando se reciben los valores ‘5’ y ’0’ se efectúa una translación en el eje Z.

```
T3D.setTranslation(traslacion);
targetTG.setTransform(T3D);          targetTG.setTransform(rotation);
```

Una parte importante es aplicar la rotación o translación al objeto que se va a transformar en este caso **targetTG.**

```
this.wakeupOn(new WakeupOnElapsedTime(50));
```

Esta línea indica el estímulo que requiere la clase para que sea ejecutada la cual es un lapso de tiempo de 50 milisegundos.

Al ejecutar el servidor_v3 le se tiene que teclear los valores en los cuales se quiere que se muevan los objetos, al teclear el '4' se mueve el cubo a la derecha..

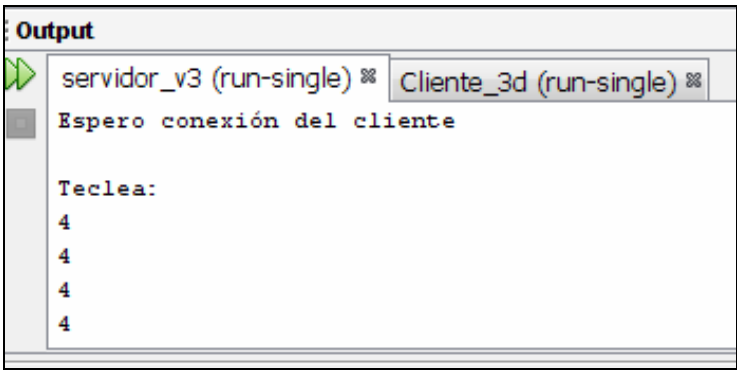


Figura 20. Ejecutable servidor_v3

Al ejecutar el cliente3d se visualiza un cubo y espera instrucciones para empezar a moverse.

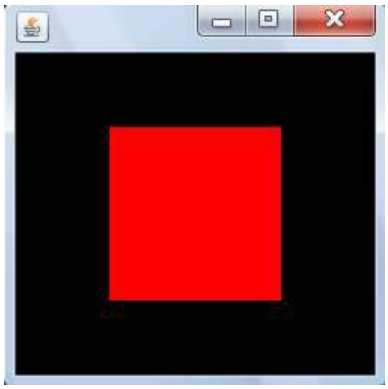
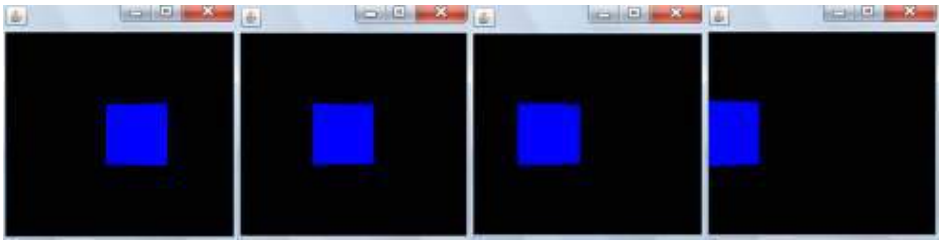


Figura 21. Visualización del cliente3d

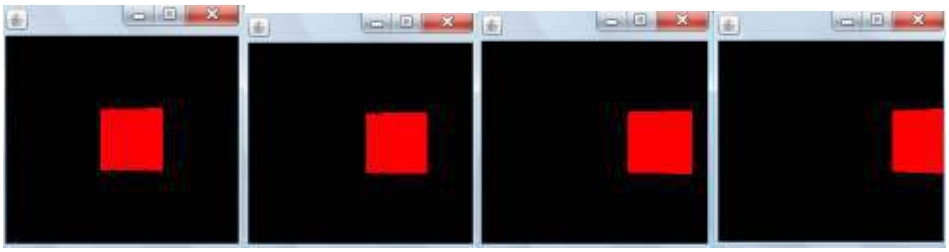
La figura 22.a muestra el cubo en la posición inicial, si se teclea '4' en el servidor, se moverá a la izquierda el cubo en el cliente como se muestra en la figura 22.b, c, d.



(a) (b) (c) (d)

Figura 22. Secuencia de movimiento del cubo a la izquierda

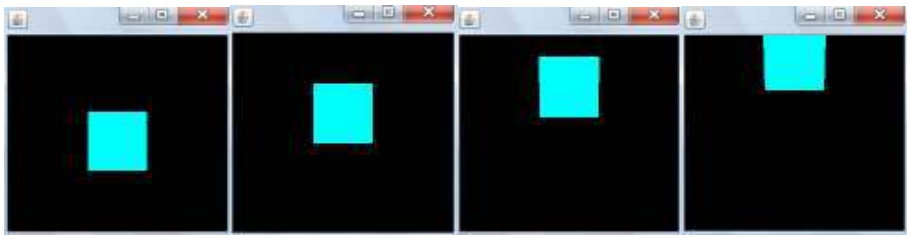
La figura 23.a muestra el cubo en la posición inicial, si se teclea ‘6’ en el servidor, se moverá a la derecha el cubo en el cliente como se muestra en la figura 23.b, c, d.



(a) (b) (c) (d)

Figura 23. Secuencia de movimiento del cubo a la derecha

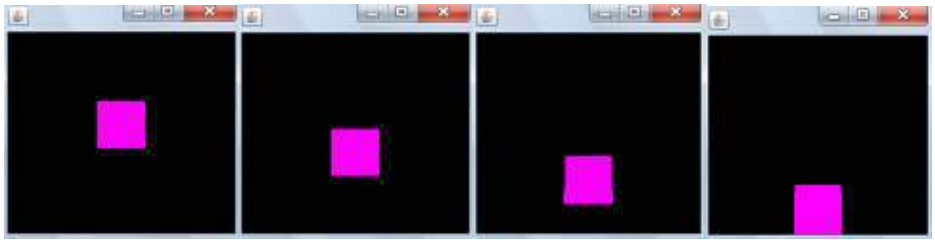
La figura 24.a muestra el cubo en la posición inicial, si se teclea ‘8’ en el servidor, se moverá hacia arriba el cubo en el cliente como se muestra en la figura 24.b, c, d.



(a) (b) (c) (d)

Figura 24. Secuencia de movimiento del cubo hacia arriba

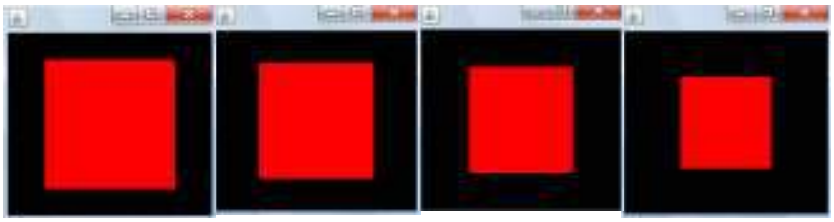
La figura 25.a muestra el cubo en la posición inicial, si se teclea ‘2’ en el servidor, se moverá hacia abajo el cubo en el cliente como se muestra en la figura 25.b, c, d.



(a) (b) (c) (d)

Figura 25. Secuencia de movimiento del cubo hacia abajo

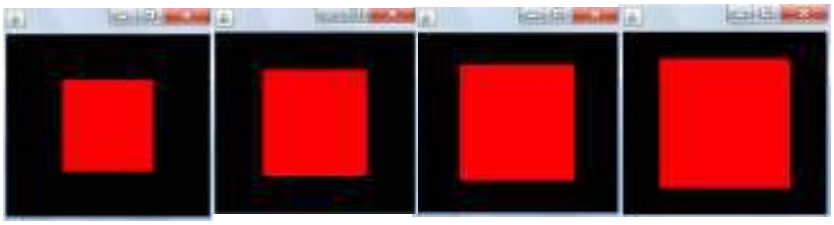
La figura 26.a muestra el cubo en la posición inicial, si se teclea ‘5’ en el servidor, se moverá hacia atrás el cubo en el cliente como se muestra en la figura 26.b, c, d.



(a) (b) (c) (d)

Figura 26. Secuencia de movimiento del cubo hacia atrás

La figura 27.a muestra el cubo en la posición inicial, si se teclea ‘0’ en el servidor, se moverá hacia adelante el cubo en el cliente como se muestra en la figura 27.b, c, d.



(a) (b) (c) (d)

Figura 27. Secuencia de movimiento del cubo hacia adelante

4.2 Servidor Versión 4.

En la versión 4 del servidor se implementa la comunicación con los tres clientes, los cuales ya tienen cargados los mundos virtuales. En el código se declaran tres variables de tipo *ServerSocket* una por cada cliente y también tres variables tipo *Socket* para el envío de los datos.

```
class Servidor_v4{
   int v = 0;
   //public BufferedWriter out;
   public Servidor_v4( ) {
       try {
            ServerSocket Servidor1 = new ServerSocket(5000);   //El servidor escucha por el puerto 5000
            System.out.println("Espero conexión del cliente 1 \n" );
            Socket Cliente1 = Servidor1.accept(); // Espera  a que se conecte el cliente
            OutputStream aux1 = Cliente1.getOutputStream();  // aux es una variable de tipo flujo de salida
            DataOutputStream flujo1= new DataOutputStream( aux1 );  // flujo es una variable de dato de flujo

            ServerSocket Servidor2 = new ServerSocket(5001);   //El servidor escucha por el puerto 5001
            System.out.println("Espero conexión del cliente 2\n" );
            Socket Cliente2 = Servidor2.accept(); // Espera  a que se conecte el cliente
            OutputStream aux2 = Cliente2.getOutputStream();  // aux es una variable de tipo flujo de salida
            DataOutputStream flujo2= new DataOutputStream( aux2 );  // flujo es una variable de dato de flujo

            ServerSocket Servidor3 = new ServerSocket(5002);   //El servidor escucha por el puerto 5002
            System.out.println("Espero conexión del cliente 3\n" );
            Socket Cliente3 = Servidor3.accept(); // Espera  a que se conecte el cliente
            OutputStream aux3 = Cliente3.getOutputStream();  // aux es una variable de tipo flujo de salida
            DataOutputStream flujo3= new DataOutputStream( aux3 );  // flujo es una variable de dato de flujo
```

Es importante recalcar que los clientes deben estar conectados en el siguiente orden, primero el cliente1 que se comunica por el puerto 5000, después el cliente2 por el puerto 5001 y finalmente el cliente3 por el puerto 5002. Si no se respeta este orden no se realizará la comunicación adecuada entre el servidor y los tres clientes. Esta especificación es debido a que en el código se implementó que se conectaran los clientes en este orden.

Para el envío de los datos se realiza a través de un ciclo el cuál estará mandado los datos tecleados a los tres clientes, hasta que el dato sea una 's', cuando ocurre esto se sale del ciclo y cierra la conexión con los tres clientes, en el orden en que fueron abiertos son cerrados las conexiones.

```
            System.out.println("Teclea: " );
            while (v != 's'){
                v = System.in.read();
                flujo1.write(v);
                flujo2.write(v);
                flujo3.write(v);
             }
             Cliente1.close();
             Servidor1.close();   //Al finalizar se tienen que cerrar los sockets

             Cliente2.close();
             Servidor2.close();   //Al finalizar se tienen que cerrar los sockets

             Cliente3.close();
             Servidor3.close();   //Al finalizar se tienen que cerrar los sockets
          }catch( Exception e ) {
               System.out.println( "No existe comunicación con los clientes" );
            }
    }
```

4.3 Cliente Versión 5.

En esta versión se modificó la forma de diseñar a los clientes, en lugar de llamarse **cliente1** se llama **Navega1**, **cliente2** se llama **Navega2** y **cliente3** se llama **Navega3**.

4.3.1 Clase: Navega1

Esta clase define lo necesario para implementar el mundo virtual, en esta sección es donde se deben incluir los escenarios para probar el motor de realidad virtual desde los objetos propios del lenguaje, como objetos cargados, iluminación y forma de navegación.

La clase Navega1 utiliza el puerto **5000** para establecer la conexión con el servidor.

```
public Navega_1() {
     try {
          Servidor = new Socket("localhost", 5000);
          in = new BufferedReader(new InputStreamReader(Servidor.getInputStream()));
```

La implementación de esta clase es extensa, por este motivo se explicarán los segmentos de código mas relevantes, el código de esta clase se muestra en los anexos.

4.3.2 Función: CargarObjetos( ).

```
public TransformGroup CargarObjetos() {
      TransformGroup objTrans = new TransformGroup();
      String arbolURL = "/MustangAirplane.obj";
      objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
      objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

      Background bg = new Background(new Color3f(0.0f,0.5f,1.0f));//Color del fondo
      bg.setApplicationBounds(new BoundingSphere(new Point3d(),1000.0));//Area que cubre,
                                                                        //radio de 1000
      int flags = ObjectFile.RESIZE;
      ObjectFile f = new ObjectFile(flags);
      Scene s = null;
      try {
          s = f.load(arbolURL);
      }catch (Exception e) {
          System.err.println(e);
          System.exit(1);
      }
      Hashtable namedObjects = s.getNamedObjects();
      Enumeration e =  namedObjects.keys();
      while (e.hasMoreElements()) {
          String name = (String) e.nextElement();
          Shape3D shape = (Shape3D) namedObjects.get(name);
          shape.setAppearance(app());
      }
      objTrans.addChild(bg);
      objTrans.addChild(s.getSceneGroup());
      return objTrans;
  }
```

Esta función llamada **CargarObjetos,** como su nombre lo indica sirve para cargar objetos en el mundo virtual, en este caso se cargan objetos de tipo *.obj, los cuales son sencillos de montar.

Un punto a destacar es que se tiene que especificar la ruta del objeto a cargar en este caso como el archivo *MustangAirplane.obj* se encuentra en la misma carpeta donde esta el archivo del programa se pone de esta manera:

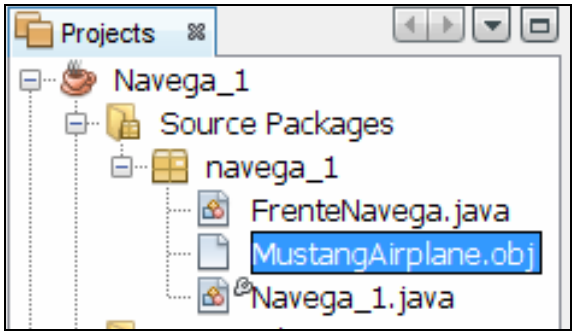


Figura 28. Proyecto Navega_1

```
String arbolURL = "/MustangAirplane.obj";
```

Pero en el caso en que se encuentre en otra carpeta el archivo obj, se debe especificar la ruta completa, separando las carpetas con diagonales '/'.

```
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
```

Para poder realizar alguna modificación a un objeto se necesita especificar a través de dos instrucciones, la primera nos indica que se puede modificar el objeto a la hora de estar corriendo el programa y la segunda nos especifica que podemos tener acceso a este objeto.

```
Background bg = new Background(new Color3f(0.0f,0.5f,1.0f));
```

Esta instrucción indica que se pondrá un fondo al mundo virtual y el color de este fondo se especifica con tres valores (RGB), en este caso es un color azul cielo. Rojo = 0, Verde = 0.5 y Azul = 1. En conversión a escala de 255 colores queda de la siguiente manera:

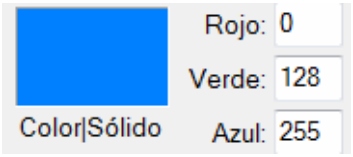


Figura 29. Color del fondo

```
bg.setApplicationBounds(new BoundingSphere(new Point3d(),1000.0));//Area que cubre
```

Esta instrucción determina el área que abarca el fondo, este fondo es una esfera y el cual especifica un diámetro de mil unidades.

```
ObjectFile f = new ObjectFile(flags);
Scene s = null;
try {
    s = f.load(arbolURL);
}catch (Exception e) {
    System.err.println(e);
    System.exit(1);
}
```

En este segmento de código es cuando se carga el objeto, primero se declara una variable de tipo *ObjectFile* llamada **f**, también se declara otra variable de tipo *Scene* llamada **s**, en el **try** lo que sucede es que se carga el archivo **arbolURL** el cual contiene el objeto *.obj* en **f** y posteriormente se asigna a **s.** Si no se puede cargar el objeto manda un mensaje de error y se sale del programa, debido a la instrucción *System.exit(1),* la cual concluye el programa e imprime en rojo el mensaje de error que se genera por no poder cargar al objeto.

```
while (e.hasMoreElements()) {
    String name = (String) e.nextElement();
    Shape3D shape = (Shape3D) namedObjects.get(name);
    shape.setAppearance(app());
}
```

En la sección de código anterior, se asigna a cada objeto su apariencia mediante la función **app( ).**

```
objTrans.addChild(bg);
objTrans.addChild(s.getSceneGroup());
return objTrans;
```

Por último se le agrega a **objTRans** el fondo y el objeto que se cargó con la variable **s**, y se regresa **objTRans** ya con todas las modificaciones, la primera es cargar el objeto satisfactoriamente en la escena, después ponerle atributos de apariencia y al último cargar un fondo.

4.3.3 Función: app( ).

```
private Appearance app() {
        Appearance app=new Appearance();
        PolygonAttributes polyAtt=new PolygonAttributes();
        polyAtt.setCullFace(PolygonAttributes.CULL_NONE);
        polyAtt.setPolygonMode(PolygonAttributes.POLYGON_FILL);
        polyAtt.setPolygonOffset(1.0f);
        polyAtt.setBackFaceNormalFlip(true);
        app.setPolygonAttributes(polyAtt);
        Material mat=new Material();
        app.setMaterial(mat);
        return app;
        }
```

Lo que realiza esta función es delimitar la apariencia del objeto, si se requiere un objeto de estructura consistente se utiliza el atributo **POLYGON_FILL** su función es visualizar a un objeto de manera sólida, es decir, que no se pueda ver el interior del objeto, esto lo realiza la siguiente línea:

```
polyAtt.setPolygonMode(PolygonAttributes.POLYGON_FILL);
```

Si se utiliza el atributo **POLYGON_FILL**, genera que el objeto cargado tenga una apariencia sólida como la de la figura 30.

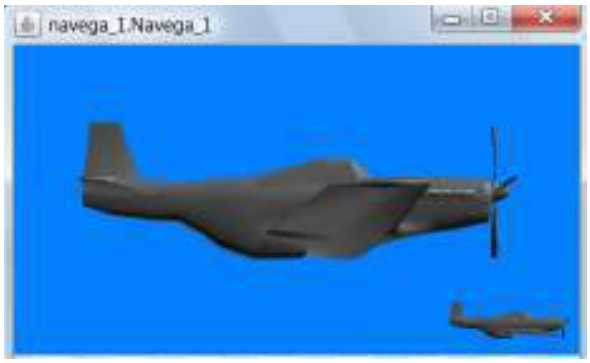


Figura 30. Objeto sólido

Si se utiliza el atributo **POLYGON_LINE**, genera que el objeto cargado se vea estructurado a través de líneas como la de la figura 31.

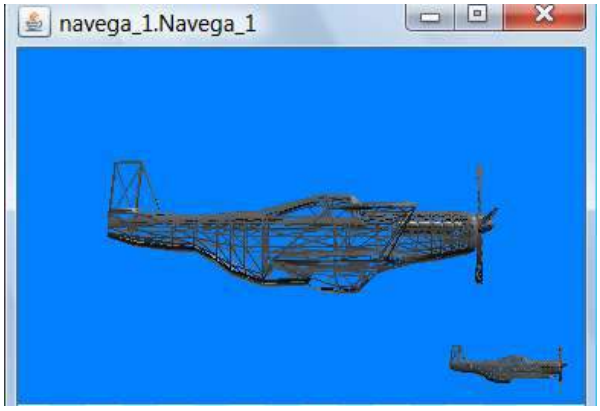


Figura 31. Objeto trazado con líneas

Si se utiliza el atributo **POLYGON_POINT**, genera que el objeto cargado se vea estructurado a través de puntos como la de la figura 32.

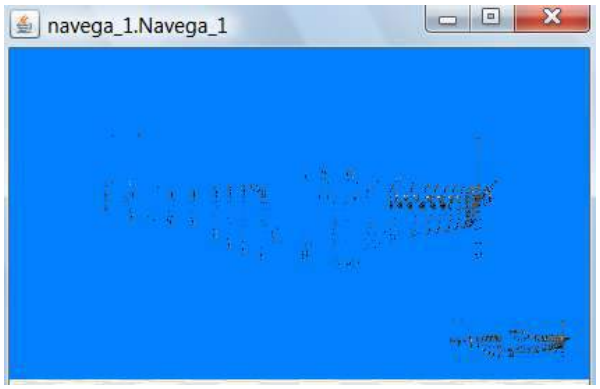


Figura 32. Objeto trazado con puntos

4.3.4 Función: crearSuelo( ).

Esta función sirve para crear el suelo del escenario, este suelo es de forma cuadriculada debido a que es mas fácil su visualización cuando se carguen los escenarios en las tres pantallas.

```
private TransformGroup crearSuelo() {
  TransformGroup sueloTransf = new TransformGroup();
  int tamaño = 40;
  Point3f[] vertices = new Point3f[tamaño * tamaño];

  float inicio = -20.0f;
  float x = inicio;
  float z = inicio;
  float salto = 1.0f;
  int[] indices = new int[(tamaño - 1)*(tamaño - 1) * 4];
  int n = 0;
  Color3f blanco = new Color3f(1.0f, 1.0f, 1.0f);
  Color3f gris = new Color3f(0.90f, 0.90f, 0.9f);
  Color3f[] colors = { blanco, gris };
  int[] colorindices = new int[indices.length];

  for (int i=0; i<tamaño; i++) {
    for (int j=0; j<tamaño; j++) {
        vertices[i*tamaño + j] = new Point3f(x, -1.0f, z);
        z += salto;
        if (i<(tamaño - 1) && j<(tamaño - 1)) {
          int cindex = (i % 2 + j) % 2;
          colorindices[n] = cindex; indices[n++] = i * tamaño + j;
          colorindices[n] = cindex; indices[n++] = i * tamaño + (j + 1);
          colorindices[n] = cindex; indices[n++] = (i + 1) * tamaño + (j + 1);
          colorindices[n] = cindex; indices[n++] = (i + 1) * tamaño + j;
         }
    }
    z = inicio;
    x += salto;
  }

  IndexedQuadArray geom = new IndexedQuadArray( vertices.length,
                                                GeometryArray.COORDINATES |
                                                GeometryArray.COLOR_3,
                                                indices.length );
  geom.setCoordinates(0, vertices);
  geom.setCoordinateIndices(0, indices);
  geom.setColors(0, colors);
  geom.setColorIndices(0, colorindices);
  Shape3D suelo = new Shape3D(geom);
  sueloTransf.addChild(suelo);
  return sueloTransf;
}
```

Parecería sencillo dibujar un suelo cuadriculado, pero la función **crearSuelo** tiene su complejidad, debido a que primero se tienen que crear todos los vértices, es decir, las esquinas de todos los cuadros, después hay que unirlos y asignarles color.

Lo primero que se realiza en el programa es inicializar la variable **tamaño** con un valor de 40, esto nos indica de cuantos cuadros por lado constará el suelo, después se declara el arreglo **vértices** con un tamaño de 40 x40 =1600, este número representa todos los puntos que existen en el suelo y se declara la variable **inicio** con un valor de -20, este valor nos indica la posición inicial de los vértices.

El vértice[0] tiene las coordenadas (-20,-1,-20), el vértice[1] = (-20,-1,-19), primero se empieza a incrementar el valor del eje Z hasta llegar a 20 positivo, esto pasa en el vértice[39] = (-20,-1,20), para el vértice[40] ya se incrementa el eje X y se inicializa en -20 el eje Z de tal forma que queda vértice[40] = (-19,-1,-20) y así hasta llegar al último vértice, el cual es el vértice[1599] = (20,-1,20), como se muestra en la figura 33.

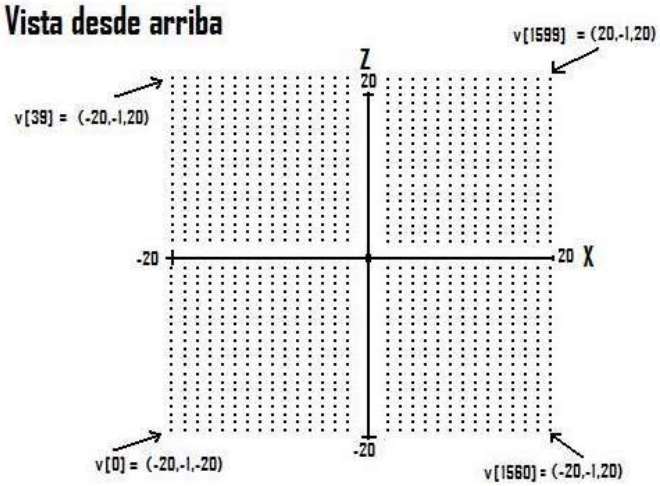


Figura 33. Construcción de vértices

Una vez que se han creado todos los vértices se lleva a cabo un enlace ordenado que permite conformar los cuadros de suelo, se comienza por los vértices 0, 1 41 y 40, se unen y posteriormente se le asigna el color blanco, los siguientes vértices en unirse son el 1,2 42 y 41 y se les asigna el color gris y así con los demás vértices, como lo muestra la figura 34.

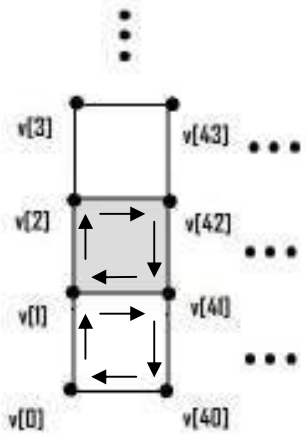


Figura 34. Construcción de suelo cuadriculado

4.3.5 Función: createSceneGraph( )

En esta función es donde se conjunta todo el mundo virtual, primero se carga el suelo posteriormente los objetos, después se define como va a realizarse la navegación y por último la iluminación del mundo virtual.

```
public BranchGroup createSceneGraph(SimpleUniverse s) {
    TransformGroup vistaTransf = null;
    BranchGroup objRoot = new BranchGroup();
    TransformGroup TransfGrp = null;
    Vector3f translate = new Vector3f();
    Transform3D SimpleTrans = new Transform3D();
        objRoot.addChild(crearSuelo());
                              //{   X  ,   Y   ,  Z   }  {   X  ,   Y   ,  Z   }
    float[][] arbolesPosicion = {{  0.0f, 0.5f,  -3.0f} , {  5.0f,  0.0f,  0.0f},
                                 {  18.0f,  3.5f,  0.0f}, {  13.0f,  2.0f,  -8.0f},
                                 {  -5.0f, 1.0f,  5.0f},  {  -15.0f,0.5f,  -15.0f},
                                 {  -15.0f,0.5f,  15.0f}, {  3.0f,  0.0f,  -10.0f},
                                 { 14.0f,  1.0f,  9.0f},  {-8.0f,  4.0f,  -7.0f},
                                 {1.0f,  0.0f,  6.0f},    { -13.0f,  2.0f,  6.0f}};

    for (int i = 0; i < arbolesPosicion.length; i++){
            translate.set(arbolesPosicion[i]);
            SimpleTrans.setTranslation(translate);
            TransfGrp = new TransformGroup(SimpleTrans);
            TransfGrp.addChild(CargarObjetos());
            objRoot.addChild(TransfGrp);
    }

    vistaTransf = s.getViewingPlatform().getViewPlatformTransform();
    translate.set( 0.1f, 0.2f, 0.3f);
    SimpleTrans.setTranslation(translate);
    vistaTransf.setTransform(SimpleTrans);

    FrenteNavega keyNavigator = new FrenteNavega(vistaTransf,val,in,Servidor);
    keyNavigator.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
    objRoot.addChild(keyNavigator);

    BoundingSphere bounds = new BoundingSphere(new Point3d(), 100.0);
    DirectionalLight directLight = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
    new Vector3f(-1.0f, -1.0f, -1.0f) );
    directLight.setInfluencingBounds(bounds);
    DirectionalLight directLight2 = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
    new Vector3f(1.0f, -1.0f, 1.0f) );
    directLight2.setInfluencingBounds(bounds);
    objRoot.addChild(directLight);
    objRoot.addChild(directLight2);

    objRoot.compile();
    return objRoot;
}
```

La siguiente instrucción es importante debido a que si no se escribe, aunque este creada la función que define el suelo, no se dibujará en el mundo.

```
objRoot.addChild(crearSuelo());
```

En el siguiente fragmento de código se describen las coordenadas en donde se cargarán los objetos, estas se encuentran especificadas por los ejes (X,Y,Z) y se almacenan en un arreglo bidimensional, esto debido a que cuando se cargan los objetos se les tiene que especificar las coordenadas donde estarán y se realiza mediante el arreglo bidimensional llamado **arbolesPosicion**.

```
                              //{   X   ,   Y   ,  Z   } {   X   ,   Y   ,  Z   }
float[][] arbolesPosicion = {{  0.0f, 0.5f,  -3.0f} , {  5.0f,  0.0f,  0.0f},
                             {  18.0f,  3.5f,  0.0f}, {  13.0f,  2.0f,  -8.0f},
                             {  -5.0f, 1.0f,  5.0f},  {  -15.0f,0.5f,  -15.0f},
                             {  -15.0f,0.5f,  15.0f}, {  3.0f,  0.0f,  -10.0f},
                             { 14.0f,  1.0f,  9.0f},  {-8.0f,  4.0f,  -7.0f},
                             {1.0f,  0.0f,  6.0f},    { -13.0f,  2.0f,  6.0f}};
```

Cuando se saben las posiciones, lo siguiente es comenzar a cargar los objetos en cada una de esta posiciones a través de un ciclo que comienza en cero hasta el tamaño de **arbolesPosicion**.

```
for (int i = 0; i < arbolesPosicion.length; i++){
        translate.set(arbolesPosicion[i]);
        SimpleTrans.setTranslation(translate);
        TransfGrp = new TransformGroup(SimpleTrans);
        TransfGrp.addChild(CargarObjetos());
        objRoot.addChild(TransfGrp);
}
```

La siguiente instrucción es muy importante debido a que se manda a llamar la clase que determina la navegación por el mundo virtual, se le envían como parámetros, el escenario, la variable donde se almacena el valor enviado por el servidor, la variable de lectura del puerto y el socket. También se determina el rango de manipulación de la clase, el cual es de un diámetro de 1000 unidades, y también se agrega como hijo de **objRoot**, si no se realizara esto no se podría navegar por el mundo.

```
FrenteNavega keyNavigator = new FrenteNavega(vistaTransf,val,in,Servidor);
keyNavigator.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
objRoot.addChild(keyNavigator);
```

Por último se asigna dos luces para iluminar el mundo con un campo de inferencia de 100 unidades y se especifican las posiciones de las luces con la instrucción *Vectro3f*

```
BoundingSphere bounds = new BoundingSphere(new Point3d(), 100.0);
DirectionalLight directLight = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
new Vector3f(-1.0f, -1.0f, -1.0f) );
directLight.setInfluencingBounds(bounds);
DirectionalLight directLight2 = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
new Vector3f(1.0f, -1.0f, 1.0f) );
directLight2.setInfluencingBounds(bounds);
objRoot.addChild(directLight);
objRoot.addChild(directLight2);
```

4.3.6 Clase: FrenteNavega.

Esta clase es llamada por **Navega1** para realizar la navegación por el mundo virtual lo hace en la siguiente línea.

```
FrenteNavega keyNavigator = new FrenteNavega(vistaTransf,val,in,Servidor);
```

Esta clase recibe este nombre precisamente porque es el escenario que esta en la pantalla de en medio, por lo que su ubicación inicial es el centro del escenario con un desplazamiento hacia arriba de 0.3, esto se define en la línea siguiente.

```
trans.setTranslation(new Vector3f(0.0f,0.0f,0.3f));
tg.setTransform(trans);
```

Esta clase se encarga de recibir un carácter del servidor y posteriormente realizar una desplazamiento si esta carácter es válido, donde se realiza todo esto es en la función **processStimulus**. Lo primero que se realiza es la lectura de un carácter que es enviado por el servidor y se le asigna a la variable **val.**

```
public void processStimulus(Enumeration criteria){
   try {
       val = in.read();
       } catch (IOException ex) {
            System.out.print("No recibi valor\n");
         }
```

Después compara la variable **val** con cada carácter numérico desde el 0 hasta el 9, además de la letra ‘i’ y ‘d’, de la siguiente forma.

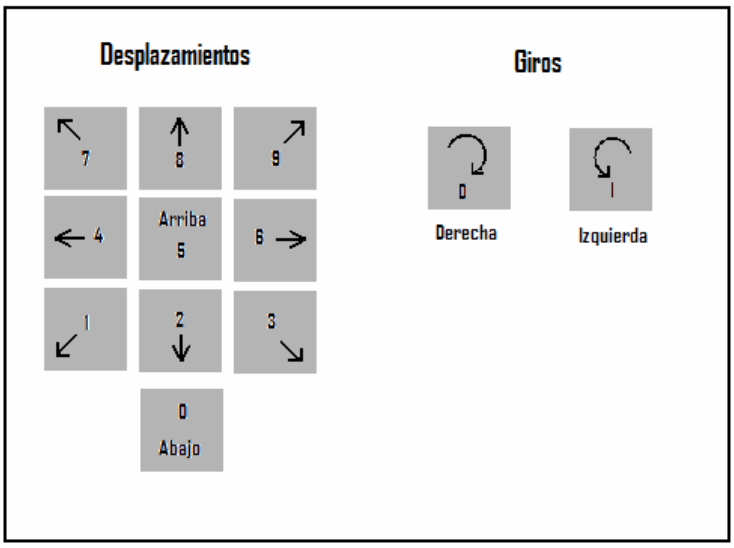


Figura 35. Teclas de Navegación

Si **val** es igual a ‘1’ se desplaza a la izquierda y hacia atrás.

```
if(val == '1'){
     distz = distz + 0.05f;
     distx = distx - 0.05f;
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
   }
```

Si **val** es igual a ‘3’ se desplaza a la derecha y hacia atrás.

```
if(val=='3'){
     distz = distz + 0.05f;
     distx = distx + 0.05f;
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
   }
```

Si **val** es igual a ‘9’ se desplaza a la derecha y hacia adelante.

```
if(val == '9'){
     distz = distz - 0.05f;
     distx = distx + 0.05f;
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
   }
```

Si **val** es igual a ‘7’ se desplaza a la izquierda y hacia adelante.

```
if(val == '7'){
    distz = distz - 0.05f;
    distx = distx - 0.05f;
    traslacion.set(distx,disty,distz);
    trans.setTranslation(traslacion);
    tg.setTransform(trans);
  }
```

Si **val** es igual a ‘2’ se desplaza hacia atrás.

```
if(val=='2'){
    distz = distz + 0.05f;          //
    traslacion.set(distx,disty,distz);
    trans.setTranslation(traslacion);
    tg.setTransform(trans);
  }
```

Si **val** es igual a ‘8’ se desplaza hacia adelante.

```
if(val=='8'){
     distz = distz - 0.05f;          //
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
  }
```

Si **val** es igual a ‘5’ se desplaza hacia arriba.

```
if(val=='5'){
     disty = disty + 0.05f;          //
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
  }
```

Si **val** es igual a ‘0’ se desplaza hacia abajo.

```
if(val=='0'){
     disty = disty - 0.05f;          //
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
  }
```

Si **val** es igual a ‘6’ se desplaza a la derecha.

```
if(val=='6'){
     distx = distx + 0.05f;          //
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
  }
```

Si **val** es igual a '4' se desplaza a la izquierda.

```
if(val=='4'){
     distx = distx - 0.05f;          //
     traslacion.set(distx,disty,distz);
     trans.setTranslation(traslacion);
     tg.setTransform(trans);
 }
```

Si **val** es igual a 'd' gira a la derecha.

```
if(val=='d'){
      yangle = yangle - angle;
      ty.rotY(-angle);
      trans.mul(ty);
      tg.setTransform(trans);
    }
```

Si **val** es igual a 'i' gira a la izquierda.

```
if(val=='i'){
      yangle = yangle + angle;
      ty.rotY(angle);
      trans.mul(ty);
      tg.setTransform(trans);
   }
```

Esta última instrucción nos indica el evento que tiene que pasar para que se active el **processStimulus,** el cual es el que transcurran 50 milisegundos.

```
this.wakeupOn(new WakeupOnElapsedTime(50));
```

4.3.7 Clase: Navega2

Esta clase es igual que Navega1, la única diferencia radica en el puerto por el cual se establece la comunicación el cual es el **5001** debido a que el 5000 lo utiliza Navega1.

```
public Navega_2() {
    try {
        Servidor = new Socket("localhost", 5001);
        in = new BufferedReader(new InputStreamReader(Servidor.getInputStream()));
```

4.3.8 Clase: IzquierdaNavega.

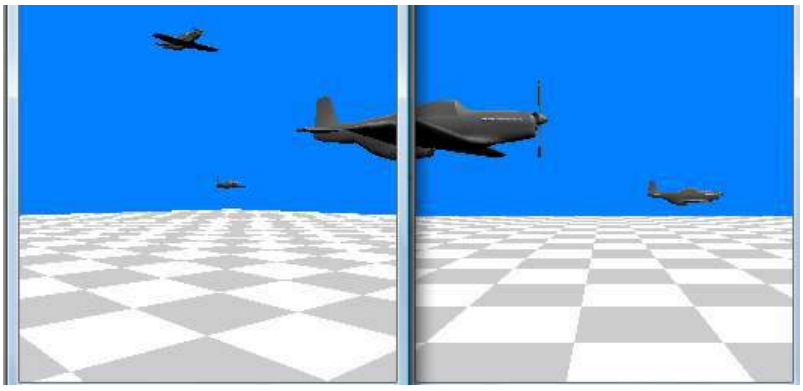
Esta clase es llamada por **Navega2** para realizar la navegación por el mundo virtual lo hace en la siguiente línea.

```
IzquierdaNavega keyNavigator = new IzquierdaNavega(vistaTransf,val,in,Servidor);
```

En este caso el escenario es el de la pantalla izquierda, por lo que su ubicación inicial es modificada **50° en contra de las manecillas del reloj**, esto se define en el siguiente fragmento de código.

```
trans.setTranslation(new Vector3f(0.0f,0.0f,0.3f));
ty.rotY(Math.PI/3.6);
trans.mul(ty);
tg.setTransform(trans);
```

Se realizó una rotación del escenario 50° debido a que de esta forma se establece continuidad entre la pantalla frontal y la pantalla izquierda, como se ve en la figura 36.



Pantalla Izquierda **Pantalla Central**

Figura 36. Pantalla Izquierda y pantalla Central

El cálculo se basa realizando una regla de tres para estimar el valor, la función *Math.PI* devuelve el valor 3.141592654 lo cual en el mundo representa 180° por tanto:

3.1416 ⟶ **180°**

3.1416 / 3.6 = 0.872664 ⟶ **X**

X = 50°

Debido a que el escenario esta rotado a **50°,** el dispositivo de despliegue de la pantalla izquierda debe estar a **130°** como se indica en la figura 37.

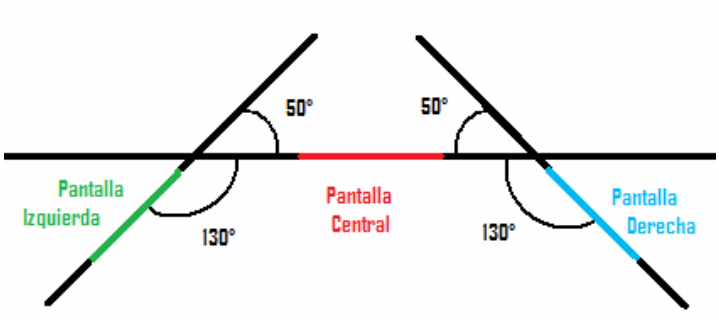


Figura 37. Ángulos de ubicación de las pantallas

4.3.9 Clase: Navega3

Esta clase es igual que N**avega1 y Navega2,** la única diferencia radica en el puerto por el cual se establece la comunicación el cual es el **5002** debido a que el 5000 lo utliza N**avega1** y el 5001 lo usa **Navega2.**

```
public Navega_3() {
    try {
        Servidor = new Socket("localhost", 5002);
        in = new BufferedReader(new InputStreamReader(Servidor.getInputStream()));
```

4.3.10 Clase: DerechaNavega.

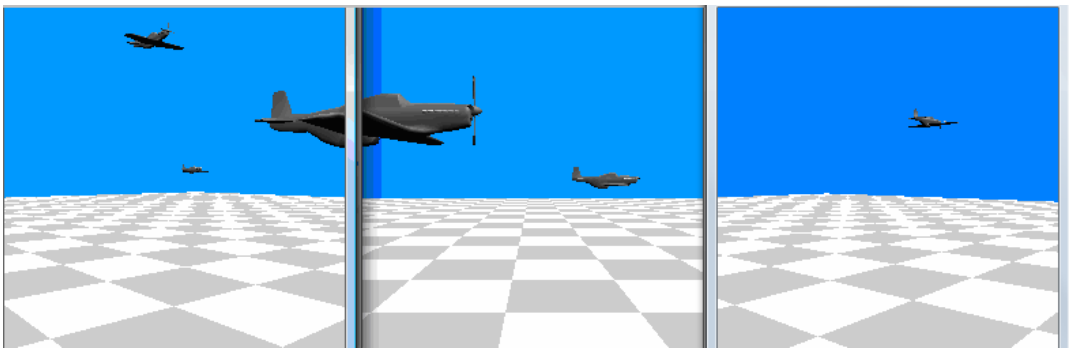
Esta clase es llamada por **Navega3** para realizar la navegación por el mundo virtual lo hace en la siguiente línea.

```
DerechaNavega keyNavigator = new DerechaNavega(vistaTransf,val,in,Servidor);
```

En este caso el escenario es el de la pantalla izquierda, por lo que su ubicación inicial es rotada **50° en el eje Y, con dirección de las manecillas del reloj**, esto se define en el fragmento de código siguiente.

```
trans.setTranslation(new Vector3f(0.0f,0.0f,0.3f));
ty.rotY(-Math.PI/3.6);
trans.mul(ty);
tg.setTransform(trans);
```

En el caso de la pantalla derecha, al igual que en la pantalla izquierda se realiza una rotación de **50° en el eje Y, en contra de las manecillas del reloj,** para que existiera una congruencia entre las tres imágenes.



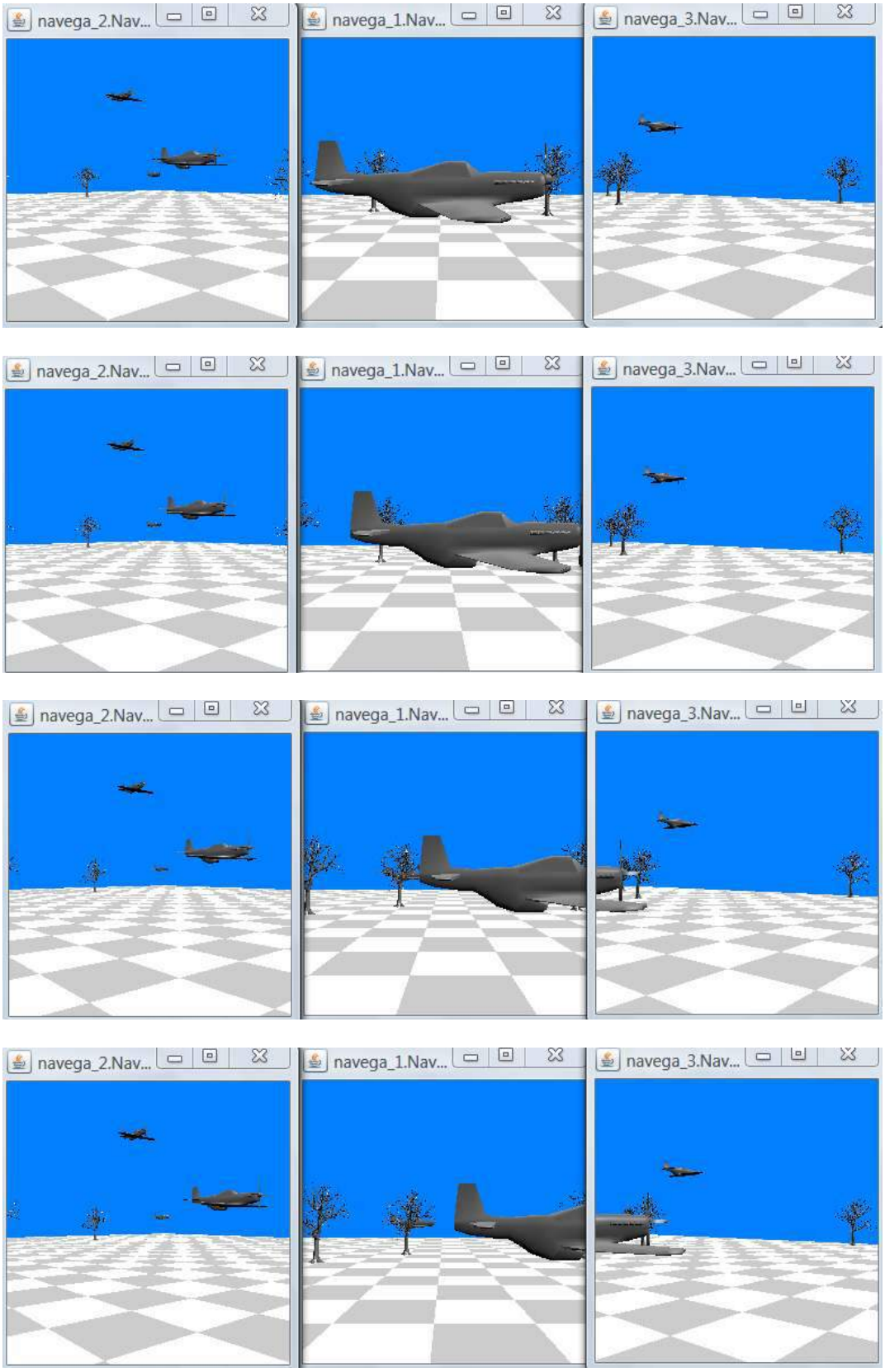
Pantalla Izquierda **Pantalla Central** **Pantalla Derecha**

Figura 38. Pantallas del motor de realidad virtual

La regla de tres es la misma que para la pantalla izquierda la única diferencia es que se le agrega el signo negativo para poder retarla en sentido contrario.

Al ejecutar el código quedan las tres pantallas sincronizadas y listas para recibir información del servidor la cual se les enviará a los tres clientes y navegarán simultáneamente por cada uno de sus mundos y haciendo el efecto óptico como si se tratara de uno solo.

Cuando se corre la aplicación en las tres pantallas y se manda desde el servidor la tecla '4', lo que se realiza en los mundos de los clientes es recorrer a la derecha la visión como lo mustran las siguientes figuras cronológicas.


navega_2.Nav...
navega_1.Nav...
navega_3.Nav...
navega_2.Nav...
navega_1.Nav...
navega_3.Nav...
navega_2.Nav...
navega_1.Nav...
navega_3.Nav...
navega_2.Nav...
navega_1.Nav...
navega_3.Nav...

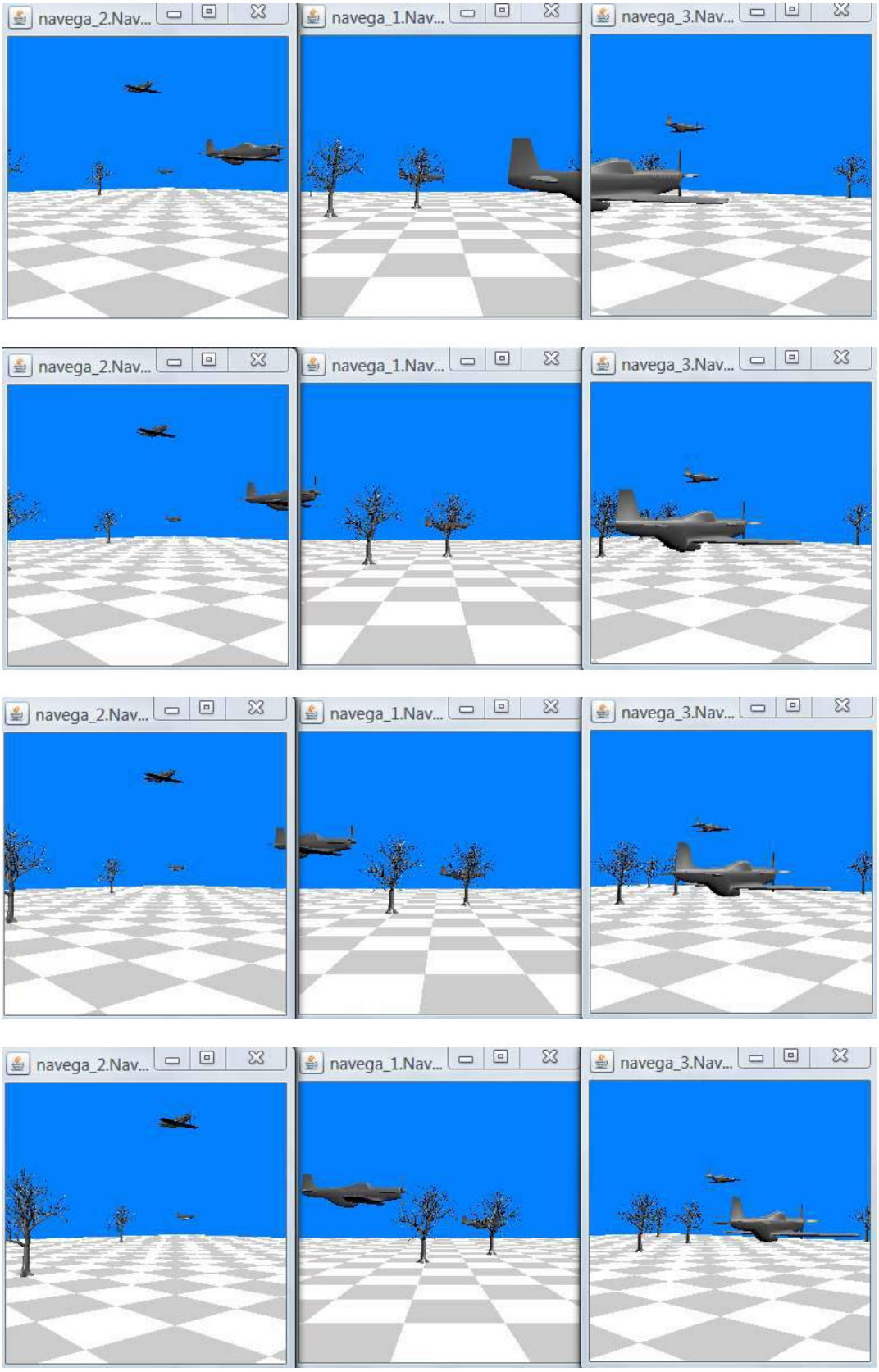


Figura 39. Secuencia de visualización del motor de realidad virtual navegado por teclado

4.4 Servidor Versión 5.

El servidor versión 5 es una adecuación al anterior, la cual consiste en poder navegar con el ratón desde el servidor cargando la pantalla central y navegando desde este y los clientes reciben la información y comienzan a navegar simultáneamente.

El código de **Servidor_v5** es muy similar al de **Navega1,** las dos diferencias radican la primera en que se realiza la conexión con los tres clientes, como se indica en el código.

```
Servidor = new ServerSocket(5000);   //El servidor escucha por el puerto 5000
System.out.println("Espero conexión del cliente \n" );
Cliente = Servidor.accept(); // Espera  a que se conecte el cliente
Servidor2 = new ServerSocket(5001);   //El servidor escucha por el puerto 5001
System.out.println("Espero conexión del cliente 2 \n" );
Cliente2 = Servidor2.accept(); // Espera  a que se conecte el cliente
Servidor3 = new ServerSocket(5002);   //El servidor escucha por el puerto 5002
System.out.println("Espero conexión del cliente 3\n" );
Cliente3 = Servidor3.accept(); // Espera  a que se conecte el cliente
```

La segunda es la manera de navegar la cual la define la clase **Escenario** que recibe como parámetros los tres sockets de conexión con los clientes, en la siguiente línea se visualiza el cambio.

```
Escenario keyNavigator = new Escenario(vistaTransf,Cliente,Cliente2,Cliente3);
```

4.4.1 Clase: Escenario

Esta clase recibe el movimiento del mouse y navega por el mundo, pero además envía esta información a los tres clientes para que realicen lo mismo en sus respectivos mundos.

El primer punto a destacar es que se va a utilizar como evento el movimiento del mouse, para definir esto se utiliza la siguiente línea.

```
AWTEventCondition = new WakeupOnAWTEvent(MouseEvent.MOUSE_MOVED);
```

Después se declaran las variables requeridas para poderles enviar información a cada uno de los clientes, esto se realiza de la siguiente forma.

```
OutputStream aux = Client.getOutputStream();  // aux es
DataOutputStream flujo= new DataOutputStream( aux );
OutputStream aux2 = Client2.getOutputStream();  // aux
DataOutputStream flujo2= new DataOutputStream( aux2 );
OutputStream aux3 = Client3.getOutputStream();  // aux
DataOutputStream flujo3= new DataOutputStream( aux3 );
```

Lo primero que hay que hacer para este tipo de navegación con mouse es dividir la pantalla, la cual se divide en ocho cuadrantes y cada uno determinaría una dirección a seguir, como lo muestra la figura 40.

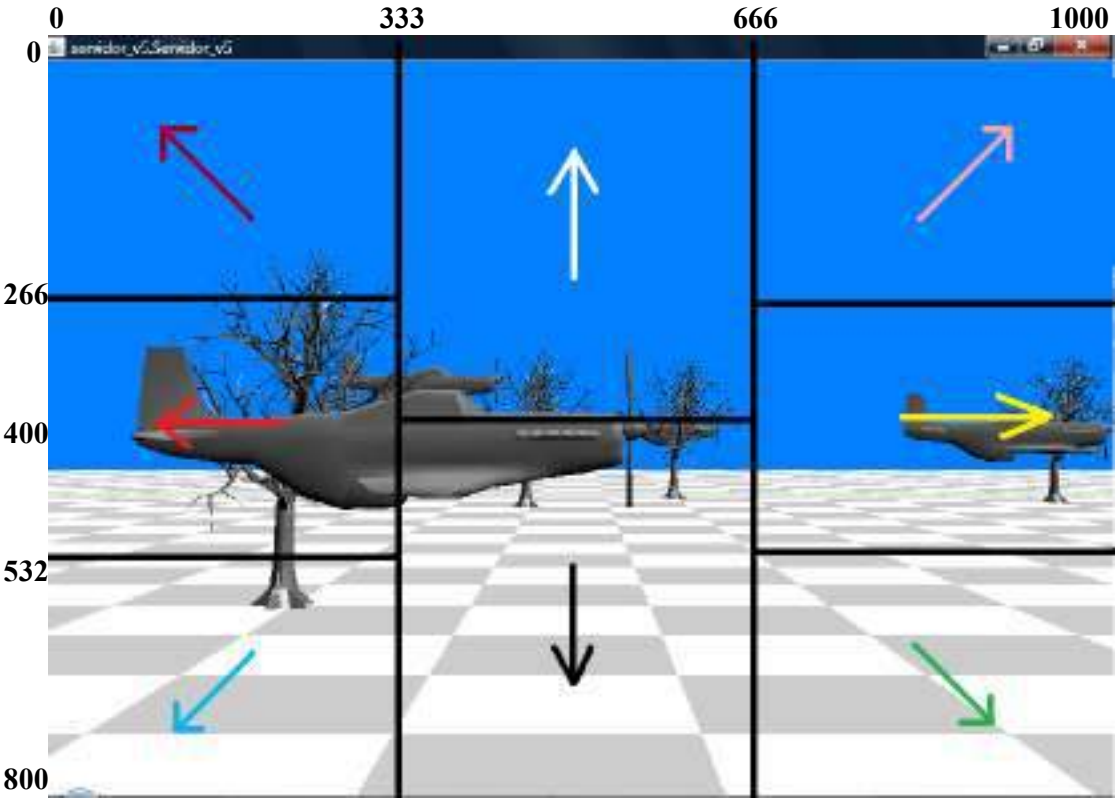


Figura 40. Segmentación de pantalla para navegación con mouse

La dirección de cada una de las flechas se indica en la siguiente tabla.

CUADRANTE	FLECHA	DIRECCIÓN
1	Vino	Izquierda y Adelante
2	Blanca	Adelante
3	Rosa	Derecha y Adelante
4	Roja	Izquierda
5	Amarilla	Derecha
6	Azul	Izquierda y Atrás
7	Negra	Atrás
8	Verde	Derecha y Atrás

Tabla 1. Dirección de los cuadrantes

Como la pantalla tiene una medida de 1000 x 800, el **1000** representa el eje X y el **800** el eje Y, se dividió de la siguiente manera.

El primer cuadrante (Flecha vino), se delimitó en el eje X mayor a **0** y menor de **333**, y en el eje Y mayor a **0** y menor a **266**, esto se implementó con un *if,* si se cumple la condición envía el valor **'7'** el cual representa el movimiento **Izquierda y Adelante**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 0)&& (((MouseEvent)evt[i]).getXOnScreen() < 333)
 && (((MouseEvent)evt[i]).getYOnScreen() < 266) && (((MouseEvent)evt[i]).getYOnScreen() >= 0))
  {
   v = '7';                                  // que obtiene de aux s
     flujo.write(v);
     flujo2.write(v);
     flujo3.write(v);
   distz = distz - 0.05f;                 //desplazamiento: izquierda y adelante
   distx = distx - 0.05f;
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

El segundo cuadrante (Flecha blanca), se delimitó en el eje X mayor a **333** y menor de **666**, y en el eje Y mayor a **0** y menor a **400**, esto se implementó con un *if,* si se cumple la condición envía el valor **'8'** el cual representa el movimiento **Adelante**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 333) && (((MouseEvent)evt[i]).getXOnScreen() < 666)
 && (((MouseEvent)evt[i]).getYOnScreen() < 400) && (((MouseEvent)evt[i]).getYOnScreen() >= 0))
  {
   v = '8';                                  // que obtiene de aux s
   flujo.write(v);
   flujo2.write(v);
   flujo3.write(v);
   distz = distz - 0.05f;                 //desplazamiento: adelante
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
  }
```

El tercer cuadrante (Flecha rosa), se delimitó en el eje X mayor a **666** y menor de **1000**, y en el eje Y mayor a **0** y menor a **266**, esto se implementó con un *if,* si se cumple la condición envía el valor **'9'** el cual representa el movimiento **Derecha y Adelante**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 666) && (((MouseEvent)evt[i]).getXOnScreen() < 1000)
 && (((MouseEvent)evt[i]).getYOnScreen() < 266) && (((MouseEvent)evt[i]).getYOnScreen() >= 0))
  {
   v = '9';                                  // que obtiene de aux s
      flujo.write(v);
      flujo2.write(v);
      flujo3.write(v);
   distz = distz - 0.05f;                 //desplazamiento: derecha y adelante
   distx = distx + 0.05f;
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

El cuarto cuadrante (Flecha roja), se delimitó en el eje X mayor a **0** y menor de **333**, y en el eje Y mayor a **266** y menor a **532**, esto se implementó con un *if,* si se cumple la condición envía el valor **'4'** el cual representa el movimiento **Izquierda**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 0)&& (((MouseEvent)evt[i]).getXOnScreen() < 333)
 && (((MouseEvent)evt[i]).getYOnScreen() < 532) && (((MouseEvent)evt[i]).getYOnScreen() >= 266))
  {
   v = '4';                                  // que obtiene de aux s
     flujo.write(v);
     flujo2.write(v);
     flujo3.write(v);
   distx = distx - 0.05f;                 //desplazamiento: izquierda
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

El quinto cuadrante (Flecha amarilla), se delimitó en el eje X mayor a **666** y menor de **1000**, y en el eje Y mayor a **266** y menor a **532**, esto se implementó con un *if,* si se cumple la condición envía el valor **'6'** el cual representa el movimiento **Derecha**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 666) && (((MouseEvent)evt[i]).getXOnScreen() < 1000)
 && (((MouseEvent)evt[i]).getYOnScreen() < 532) && (((MouseEvent)evt[i]).getYOnScreen() >= 266))
  {
   v = '6';                                  // que obtiene de aux s
      flujo.write(v);
      flujo2.write(v);
      flujo3.write(v);
   distx = distx + 0.05f;                 //desplazamiento: derecha
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

El sexto cuadrante (Flecha azul), se delimitó en el eje X mayor a **0** y menor de **333**, y en el eje Y mayor a **532** y menor a **800**, esto se implementó con un *if,* si se cumple la condición envía el valor **'1'** el cual representa el movimiento **Izquierda y Atrás**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 0) && (((MouseEvent)evt[i]).getXOnScreen() < 333)
 && (((MouseEvent)evt[i]).getYOnScreen() < 800) && (((MouseEvent)evt[i]).getYOnScreen() >= 532))
   {
     v = '1';                                  // que obtiene de aux s
    flujo.write(v);
    flujo2.write(v);
    flujo3.write(v);
    distz = distz + 0.05f;                 //desplazamiento: derecha y adelante
    distx = distx - 0.05f;
    traslacion.set(distx,disty,distz);
    trans.setTranslation(traslacion);
    tg.setTransform(trans);
   }
```

El séptimo cuadrante (Flecha negra), se delimitó en el eje X mayor a **333** y menor de **666**, y en el eje Y mayor a **400** y menor a **800**, esto se implementó con un *if,* si se cumple la condición envía el valor **'2'** el cual representa el movimiento **Atrás**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 333) && (((MouseEvent)evt[i]).getXOnScreen() < 666)
 && (((MouseEvent)evt[i]).getYOnScreen() < 800) && (((MouseEvent)evt[i]).getYOnScreen() >= 400))
  {
   v = '2';                                  // que obtiene de aux s
    flujo.write(v);
    flujo2.write(v);
    flujo3.write(v);
   distz = distz + 0.05f;                 //desplazamiento: atras
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

El octavo cuadrante (Flecha verde), se delimitó en el eje X mayor a **666** y menor de **1000**, y en el eje Y mayor a **532** y menor a **800**, esto se implementó con un *if,* si se cumple la condición envía el valor **'3'** el cual representa el movimiento **Derecha y Atrás**, este valor se envía a los tres clientes y posteriormente el servidor realiza su movimiento, como lo muestra el código siguiente.

```
if( (((MouseEvent)evt[i]).getXOnScreen() >= 666) && (((MouseEvent)evt[i]).getXOnScreen() < 1000)
 && (((MouseEvent)evt[i]).getYOnScreen() < 800) && (((MouseEvent)evt[i]).getYOnScreen() >= 532))
  {
   v = '3';                                  // que obtiene de aux s
    flujo.write(v);
    flujo2.write(v);
    flujo3.write(v);
   distz = distz + 0.05f;                 //desplazamiento: derecha y atras
   distx = distx + 0.05f;
   traslacion.set(distx,disty,distz);
   trans.setTranslation(traslacion);
   tg.setTransform(trans);
 }
```

Por último se tiene que indicar que sigue en espera del evento del Mouse esto se especifica con la siguiente línea de código.

```
this.wakeupOn(AWTEventCondition);
```

Al ejecutar el **Servidor_v5** con **Navega1, Navega2 y Navega3** se obtiene el siguiente resultado, donde el servidor tiene la misma pantalla que **Navega1** y dependiendo el desplazamiento del servidor se desplazaran los otros tres mundos.

La figura 41 muestra los tres clientes ejecutándose y el servidor_v5 que recibe los movimientos del Mouse y estos se envían a los clientes para que naveguen en sus respectivos mundos pero simulando como si fuera uno solo.

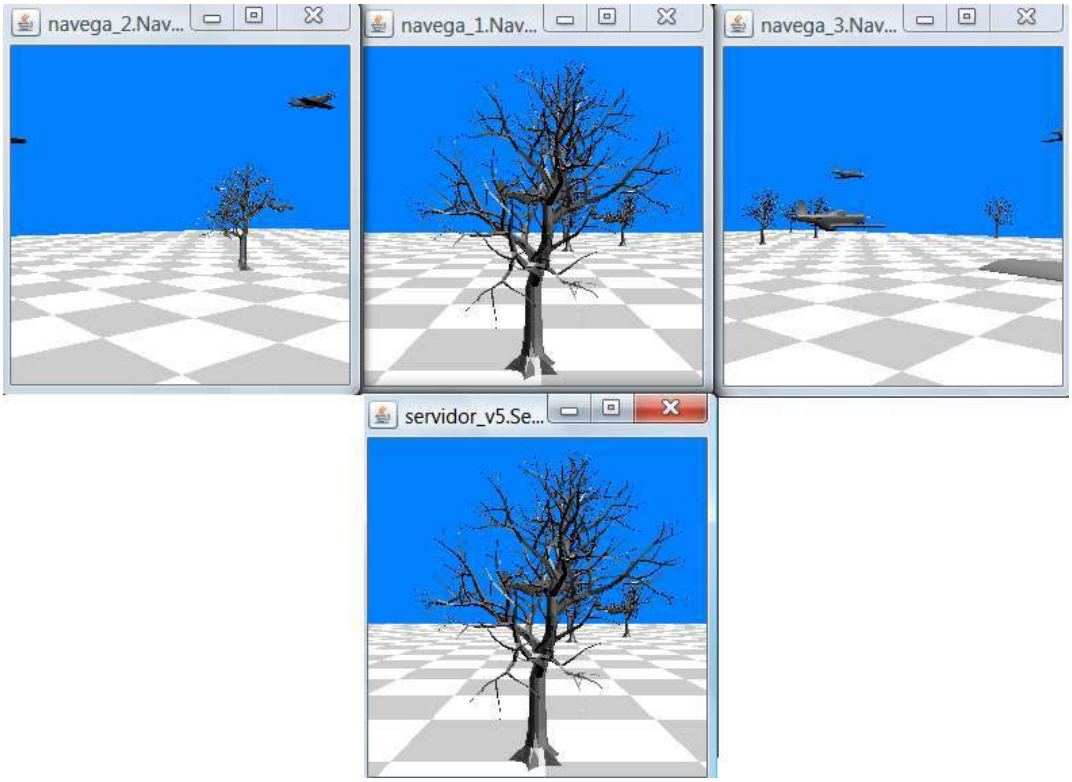


Figura 41. Visualización del motor de realidad virtual navegado por mouse

4.5 Visualización

La visualización del mundo virtual a través de las tres pantallas es en tiempo real y de una forma sincronizada por lo que genera la sensación de estar en un solo mundo virtual, logrando el objetivo principal de la realidad virtual la inmersión del usuario.

La navegación es sencilla para el usuario en cualquiera de las dos versiones, tanto la del teclado como la del mouse, el teclado es menos interactivo pero tiene mayor posibilidad de movimientos en el mundo, por otro lado la navegación por mouse es más inmersita y practica de utilizar.

Capítulo V

CONCLUSIONES

Se desarrollo una aplicación de mundos virtuales los cuales se emplearon en el motor de realidad virtual.

Se desarrollo una aplicación de comunicación en red para el procesamiento de la información utilizando el modelo cliente-servidor.

Se diseño la aplicación que sincronice y evite las colisiones de los objetos que se visualizarán a través de los dispositivos de despliegue de video.

El resultado conjunto es una aplicación general llamada motor de realidad virtual, la cual permita sincronizar escenarios virtuales en múltiples dispositivos de despliegue de video a través de una conexión en red.

El modelo cliente-servidor es muy versátil, debido a que se puede emplear en diferentes aplicaciones, para el caso particular del motor de realidad virtual es muy útil y se puede implementar con el lenguajes de programación Java.

La programación es una importante herramienta de desarrollo, que va desde realizar operaciones muy sencillas hasta llegar a programar mundos virtuales sincronizados en diferentes pantallas, existen diferentes lenguajes de programación y cada uno de estos tiene fortalezas y debilidades, lo importante es encontrar el adecuado para las necesidades que se tengan, en el caso de esta tesis el lenguaje de programación Java fue el mas indicado debido a su versatilidad, como son sus prestaciones en programación en red o sus múltiples API´s, una de estas es Java3D la cual sirvió para programar los mundos virtuales ya descritos en la presente tesis.

Este trabajo puede ser enfocado a diferentes áreas, la principal por la cual se diseño es la educativa, debido a que en el CIDETEC esta poniéndose en marcha un laboratorio de realidad virtual y el motor de realidad virtual puede ser implementado en la cabina de realidad virtual, pudiendo ser utilizado con otras aplicaciones simultáneamente como el caso de la caminadora. Otro aspecto educativo podría ser la reutilización de código ya que se puede utilizar la programación del motor de realidad virtual e implementarle nuevos escenarios mas complejos como parte de alguna tarea o proyecto en la materia de realidad virtual y poderlo probar en la cabina de realidad virtual y que esta tesis sirva como una guía educativa.

Esta tesis describe la programación para establecer una conexión cliente servidor-sencilla, después una conexión entre un servidor y tres clientes y después poder enviar información del servidor al cliente. Por esta razón es versátil esta tesis porque explica los pasos que se siguen para programar un motor de realidad virtual, pero los pasos intermedios pueden ayudar a alguien que desee realizar otro proyecto de comunicación y realidad virtual.

Un punto muy importante para el diseño de un documento es poder mostrar los resultados de lo que se va realizando, por este motivo en esta tesis se ponen los segmentos de código mas representativos, figuras y los resultados que se dan al correr los programas, para tener un enfoque mas completo de lo que se esta realizando.

El sistema de desarrollo que se planteó al principio de esta tesis el cual describe que primero se tiene que realizar el proyecto físicamente y posteriormente ya con el proyecto terminado realizar la escritura de todas las especificaciones que se fueron realizando a lo largo de todo el proceso, fue de gran utilidad porque es mas sencillo escribir la tesis cuando ya el proyecto se ha culminado.

5.1 Resultados

Los resultados obtenidos en esta tesis son:

- Tres versiones de cliente-servidor, las cuales pueden ser utilizadas por otras aplicaciones que necesiten comunicar más de dos computadoras.

- Un motor de realidad virtual nevegado por teclado conformado por cuatro clases principales:
    - Navega_1 (Pantalla Central)
    - Navega_2 (Pantalla Izquierda)
    - Navega_3 (Pantalla Derecha)
    - servidor_v4 (Navegación por teclado)

- Un motor de realidad virtual nevegado por mouse, el cual utiliza las mismas clases de pantallas que el servidor_v4:
    - Navega_1 (Pantalla Central)
    - Navega_2 (Pantalla Izquierda)
    - Navega_3 (Pantalla Derecha)
    - Servidor_v5 (Navegación por mouse)

5.2 TRABAJOS FUTUROS

Existen algunas mejoras que se le puden realizar a este trabajo:

- Implementación con otro modelo de conexión de red distribuida o paralela utlizando el cluster del CIDETEC.
- Desarrollo de un algoritmo de detección de colisiones más óptimo o que se implemente de una forma general para cualquier objeto.

Otro aspecto que se le podría dar es visualizarlo como una aplicación o herramienta para otro trabajo que utilice realidad virtual, algunos ejemplos serían: una cabina de inmersión, un brazo robótico, una caminadora, entre otros.

El motor de realidad virtual puede ser enfocado a simuladores aéreos, de automóviles o cualquier otro simulador que tenga mas de una pantalla, precisamente por el diseño del motor, lo que se tendría que hacer serían adecuacuaciones en cuanto a los ángulos de la pantallas y algunas otras modificaciones menores.

REFERENCIAS

[1] David Garrosa Sastre, “Entendiendo la visión humana”, Articulo de divulgación, Octubre 1998.

[2] Jeffrey Jacobson, Marc Le Renard, Jean-Luc Lugrin, Marc Cavaza, “The CaveUT system: immersive entertainment based on a game engine”, Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, 2005.

[3] Cristina Díaz Busch, José Luis Paz Moscoso, “Realidad virtual: Visión Artificial”, 2006.

[4] Marc Cavazza, Jean-Luc Lugrin, Simon Hartley, Paolo Libardi, Matthew J. Barnes, Mikael Le Bras, Marc Le Renard, Louis Bec, Alok Nandi, “New ways of worldmaking: the Alterne platform for VR art”, Proceedings of the 12th annual ACM international conference on Multimedia MULTIMEDIA '04, October 2004.

[5] Renaud Ott, Mario Gutiérrez, Daniel Thalmann, Frédéric Vexo, “ Advanced Virtual Reality Technologies for Surveillance and Security Applications”, 2006

[6] Brian Salomon, Maxim Garber, Ming C. Lin, Dinesh Manocha, “Interactive Navigation in Complex Environments Using Path Planning and collision detection”, Department of Computer Science University of North Carolina, 2004

[7] Kiran Varanasi,” Computer Graphics as a Space Journey displayed in three screens”, International Institute of Technological Research of the Cairo,2005

ANEXO A

Código del Cliente Versión 4

```
package navega_1;
/*
 * @author Roy
 */
import java.applet.Applet;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.*;
import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.awt.*;
import java.util.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.universe.SimpleUniverse;
import java.io.*;
import java.net.*;

public class Navega_1 extends Applet {
   public int val;
   public BufferedReader in;
   Socket Servidor = null;
  public Navega_1() {
      try {
          Servidor = new Socket("localhost", 5000);
          in = new BufferedReader(new InputStreamReader(Servidor.getInputStream()));
         } catch (IOException e) {
              System.err.println("No tengo comunicacion con el servidor" );
              System.exit(0);
             }
      setLayout(new BorderLayout());
      GraphicsConfiguration config =
      SimpleUniverse.getPreferredConfiguration();
      Canvas3D canvas3D = new Canvas3D(config);
      add("Center", canvas3D);
      SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
      BranchGroup scene = createSceneGraph(simpleU);
      simpleU.addBranchGraph(scene);
   }

  public TransformGroup CargarObjetos() {
            TransformGroup objTrans = new TransformGroup();
            String arbolURL = "/MustangAirplane.obj";
            objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
            objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

            Background bg = new Background(new Color3f(0.0f,0.5f,1.0f));//Color del fondo
            bg.setApplicationBounds(new BoundingSphere(new Point3d(),1000.0));//Area que cubre
            int flags = ObjectFile.RESIZE;
           ObjectFile f = new ObjectFile(flags);
          Scene s = null;
       try {
          s = f.load(arbolURL);
            }catch (Exception e) {
                  System.err.println(e);
                  System.exit(1);
                }
```

```
        Hashtable namedObjects = s.getNamedObjects();
        Enumeration e =  namedObjects.keys();
        while (e.hasMoreElements()) {
                String name = (String) e.nextElement();
                Shape3D shape = (Shape3D) namedObjects.get(name);
                shape.setAppearance(app());
            }

        objTrans.addChild(bg);
            objTrans.addChild(s.getSceneGroup());
        return objTrans;
    }

    public TransformGroup CargarObjetos2() {
            TransformGroup objTrans = new TransformGroup();
            String arbolURL = "C:/Arbol.obj";
        objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

            Background bg = new Background(new Color3f(0.0f,0.5f,1.0f));//Color del fondo
            bg.setApplicationBounds(new BoundingSphere(new Point3d(),1000.0));//Area que cubre,
                                                    //radio de 1000
            int flags = ObjectFile.RESIZE;
        ObjectFile f = new ObjectFile(flags);
        Scene s = null;
        try {
           s = f.load(arbolURL);
        }catch (Exception e) {
           System.err.println(e);
           System.exit(1);
        }
            Hashtable namedObjects = s.getNamedObjects();
        Enumeration e =  namedObjects.keys();
        while (e.hasMoreElements()) {
                String name = (String) e.nextElement();
                Shape3D shape = (Shape3D) namedObjects.get(name);
                shape.setAppearance(app());
            }

        objTrans.addChild(bg);
            objTrans.addChild(s.getSceneGroup());
        return objTrans;
    }

private Appearance app() {
            Appearance app=new Appearance();
            PolygonAttributes polyAtt=new PolygonAttributes();
            polyAtt.setCullFace(PolygonAttributes.CULL_NONE);
            polyAtt.setPolygonMode(PolygonAttributes.POLYGON_FILL);
            polyAtt.setPolygonOffset(1.0f);
            polyAtt.setBackFaceNormalFlip(true);
            app.setPolygonAttributes(polyAtt);
            Material mat=new Material();
            app.setMaterial(mat);
            return app;
            }

  private TransformGroup crearSuelo() {
    TransformGroup sueloTransf = new TransformGroup();
    int tamaño = 40;
    Point3f[] vertices = new Point3f[tamaño * tamaño];
```

```
float inicio = -20.0f;
float x = inicio;
float z = inicio;
float salto = 1.0f;
int[] indices = new int[(tamaño - 1)*(tamaño - 1) * 4];
int n = 0;
Color3f blanco = new Color3f(1.0f, 1.0f, 1.0f);
Color3f gris = new Color3f(0.8f, 0.8f, 0.8f);
Color3f[] colors = { blanco, gris };
int[] colorindices = new int[indices.length];

for (int i=0; i<tamaño; i++) {
  for (int j=0; j<tamaño; j++) {
          vertices[i*tamaño + j] = new Point3f(x, -1.0f, z);
          z += salto;
          if (i<(tamaño - 1) && j<(tamaño - 1)) {
       int cindex = (i % 2 + j) % 2;
       colorindices[n] = cindex; indices[n++] = i * tamaño + j;
       colorindices[n] = cindex; indices[n++] = i * tamaño + (j + 1);
       colorindices[n] = cindex; indices[n++] = (i + 1) * tamaño + (j + 1);
       colorindices[n] = cindex; indices[n++] = (i + 1) * tamaño + j;


          }
  }
  z = inicio;
  x += salto;
}

IndexedQuadArray geom = new IndexedQuadArray( vertices.length,
                                                GeometryArray.COORDINATES |
                                                GeometryArray.COLOR_3,
                                                indices.length );
geom.setCoordinates(0, vertices);
geom.setCoordinateIndices(0, indices);
geom.setColors(0, colors);
geom.setColorIndices(0, colorindices);
Shape3D suelo = new Shape3D(geom);
sueloTransf.addChild(suelo);
return sueloTransf;
}

public BranchGroup createSceneGraph(SimpleUniverse s) {
   TransformGroup vistaTransf = null;
   BranchGroup objRoot = new BranchGroup();
       TransformGroup TransfGrp = null;
   TransformGroup TransfGrp2 = null;
   Vector3f translate = new Vector3f();
    Vector3f translate2 = new Vector3f();
   Transform3D SimpleTrans = new Transform3D();
   Transform3D SimpleTrans2 = new Transform3D();
     objRoot.addChild(crearSuelo());

          //{  X ,  Y  , Z  } {  X ,  Y  , Z  }
   float[][] arbolesPosicion = {{  0.0f, 0.5f,  -3.0f} , {  5.0f,  0.0f,  0.0f},
                        {  18.0f,  3.5f,  0.0f}, {  13.0f,  2.0f,  -8.0f},
                        {  -5.0f, 1.0f,  5.0f},  {  -15.0f,0.5f,  -15.0f},
                        {  -15.0f,0.5f,  15.0f}, {  3.0f,  0.0f,  -10.0f},
                        { 14.0f,  1.0f,  9.0f},  {-8.0f,  4.0f,  -7.0f},
                        {1.0f,  0.0f,  6.0f},    { -13.0f,  2.0f,  6.0f}};
```

```
        float[][] arbolesPosicion2 = {{  0.0f, 0.0f,  3.0f} ,{  -5.0f,  0.0f,  0.0f},
                                      {  18.0f,  0.f,  -3.0f}, {  -13.0f,  0.0f,  -8.0f},
                                      {  5.0f, 0.0f,  5.0f},   {  15.0f, 0.0f,  -15.0f},
                                      {  -10.0f,0.0f,  15.0f}, {  3.0f,  0.0f,  -8.0f},
                                      { 12.0f,  0.0f,  -9.0f}, {8.0f,  0.0f,  -7.0f},
                                      {1.0f,  0.0f,  -6.0f},   { 13.0f,  0.0f,  6.0f}};

        for (int i = 0; i < arbolesPosicion.length; i++){
              translate.set(arbolesPosicion[i]);
              SimpleTrans.setTranslation(translate);
              TransfGrp = new TransformGroup(SimpleTrans);
              TransfGrp.addChild(CargarObjetos());
              objRoot.addChild(TransfGrp);
        }

        for (int i = 0; i < arbolesPosicion2.length; i++){
              translate2.set(arbolesPosicion2[i]);
              SimpleTrans2.setTranslation(translate2);
              TransfGrp2 = new TransformGroup(SimpleTrans2);
              TransfGrp2.addChild(CargarObjetos2());
              objRoot.addChild(TransfGrp2);
        }

        vistaTransf = s.getViewingPlatform().getViewPlatformTransform();
        translate.set( 0.1f, 0.2f, 0.3f);
        SimpleTrans.setTranslation(translate);
        vistaTransf.setTransform(SimpleTrans);

        FrenteNavega keyNavigator = new FrenteNavega(vistaTransf,val,in,Servidor);
        keyNavigator.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        objRoot.addChild(keyNavigator);

            BoundingSphere bounds = new BoundingSphere(new Point3d(), 100.0);
            DirectionalLight directLight = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
            new Vector3f(-1.0f, -1.0f, -1.0f) );
            directLight.setInfluencingBounds(bounds);
        DirectionalLight directLight2 = new DirectionalLight( new Color3f(0.5f,0.5f,0.5f),
            new Vector3f(1.0f, -1.0f, 1.0f) );
            directLight2.setInfluencingBounds(bounds);
        objRoot.addChild(directLight);
        objRoot.addChild(directLight2);

        objRoot.compile();
        return objRoot;
    }

    public static void main(String[] args) {
        new MainFrame(new Navega_1(), 250, 250);
    }
}
```

ANEXO B

Código del Servidor Versión 4

```
package servidor_v4;   //Paquete del proyecto al que pertenece

import java.io.* ;
import java.net.* ;
/*
 * @author Roy
 */
class Servidor_v4{
  int v = 0;
  //public BufferedWriter out;
  public Servidor_v4( ) {
     try {
         ServerSocket Servidor1 = new ServerSocket(5000);   //El servidor escucha por el puerto 5000
         System.out.println("Espero conexión del cliente 1 \n" );
         Socket Cliente1 = Servidor1.accept(); // Espera  a que se conecte el cliente
         OutputStream aux1 = Cliente1.getOutputStream();  // aux es una variable de tipo flujo de salida
         DataOutputStream flujo1= new DataOutputStream( aux1 );

         ServerSocket Servidor2 = new ServerSocket(5001);   //El servidor escucha por el puerto 5001
         System.out.println("Espero conexión del cliente 2\n" );
         Socket Cliente2 = Servidor2.accept(); // Espera  a que se conecte el cliente
         OutputStream aux2 = Cliente2.getOutputStream();  // aux es una variable de tipo flujo de salida
         DataOutputStream flujo2= new DataOutputStream( aux2 );

         ServerSocket Servidor3 = new ServerSocket(5002);   //El servidor escucha por el puerto 5002
         System.out.println("Espero conexión del cliente 3\n" );
         Socket Cliente3 = Servidor3.accept(); // Espera  a que se conecte el cliente
         OutputStream aux3 = Cliente3.getOutputStream();  // aux es una variable de tipo flujo de salida
         DataOutputStream flujo3= new DataOutputStream( aux3 );

         System.out.println("Teclea: " );
         while (v != 's'){
             v = System.in.read();
             flujo1.write(v);
             flujo2.write(v);
             flujo3.write(v);
          }
          Cliente1.close();
          Servidor1.close();

          Cliente2.close();
          Servidor2.close();

          Cliente3.close();
          Servidor3.close();   //Al finalizar se tienen que cerrar los sockets

       }catch( Exception e ) {
           System.out.println( "No existe comunicación con los clientes" );
         }
   }
  public static void main(String[] args) {
       new Servidor_v4();
   }
}
```