Instituto Politécnico Nacional
Centro de Innovación y Desarrollo
Tecnológico en Cómputo

INTERFAZ PARA EL APRENDIZAJE DE LA REALIDAD VIRTUAL HACIENDO USO DE UN GUANTE DE DATOS

TESIS QUE PARA OBTENER EL GRADO DE

MAESTRÍA EN TECNOLGÍA DE CÓMPUTO

PRESENTA:

Lic. Rodrigo Morales Alvarado

Director:

M. en C. Mauricio Olguín Carbajal

México, D. F.
Noviembre, 2007

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 13 del mes de NOVIEMBRE del 2007 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:

INTERFAZ PARA EL APRENDIZAJE DE LA REALIDAD VIRTUAL HACIENDO USO DE UN GUANTE DE DATOS

Presentada por el alumno:

MORALES	ALVARADO	RODRIGO
Apellido paterno	materno	nombre(s)

Con registro:

A	0	5	0	0	0	9

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. RAMÓN SILVA ORTIGOZA
Presidente

M. EN C. EDUARDO RODRÍGUEZ ESCOBAR
Secretario

M. EN C. MAURICIO OLGUÍN CARBAJAL
Primer Vocal
(Director de Tesis)

M. EN C. JUAN CARLOS GONZÁLEZ ROBLES
Segundo Vocal

M. EN C. EDUARDO VEGA ALVARADO
Tercer Vocal

M. EN C. JUAN CARLOS HERRERA LOZADA
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 POLITECNICO NACIONAL

COORDINACION GENERAL DE POSGRADO E INVESTIGACION

CARTA DE CESIÓN DE DERECHOS

En la Ciudad de México, el día 13 del mes noviembre del año 2007, el que suscribe Rodrigo Morales Alvarado, alumno del Programa de Maestría en Tecnología de Cómputo, con número de registro A050009, adscrito al Centro de Innovación y Desarrollo Tecnológico en Cómputo, manifiesta que es autor intelectual del presente trabajo de Tesis bajo la dirección del M. en C. Mauricio Olguín Carbajal y cede los derechos del trabajo intitulado "*Interfaz para el aprendizaje de la realidad virtual haciendo uso de un guante de datos*" 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 Morales Alvarado

Resumen

En el presente trabajo de tesis el cual lleva por titulo: “Interfaz para el aprendizaje de la realidad virtual haciendo uso de un guante de datos”, se documenta el proceso de desarrollar una interfaz genérica para recibir información de un guante de datos, con el fin de retroalimentar un ambiente en tres dimensiones y aprovechar las características de interacción intuitiva de este dispositivo. Primero se presentan los antecendentes de la realidad virtual desde su origen hasta nuestros días; se analiza la problemática actual de la realidad virtual, y la importancia de desarrollar interfaces genéricas que puedan servir de trampolín para desarrollos actuales y futuros. El objetivo de este proyecto, es que el estudiante aprenda los siguientes conceptos: Creación de figuras geométricas en tres dimensiones, manipulación de figuras geométricas en tres dimensiones, navegación en ambientes de tres dimensiones, concepto de seis grados de libertad (6DOF) y de rotación en tres ángulos (yaw, pitch y roll); todo esto desde el punto de vista de realidad virtual, es decir partiendo de un entorno tridimensional, que interactúe directamente con el usuario; debido a que la mayoría de los cursos de realidad virtual están diseñados en dos dimensiones, y no hacen uso de ningún dispositivo inmersivo para su mejor comprensión.

La interfaz propuesta hace uso de un guante de datos para interactuar con el usuario. El uso de dispositivos inmersivos como lo son guantes de datos permite una navegación más eficiente dentro de los ambientes virtuales y acelera la curva de aprendizaje de los estudiantes y usuarios. Hoy en día existen varios guantes comerciales y su precio oscila de los 59 hasta los 1495 dólares. En éste trabajo se hace un análisis de los guantes comerciales más importantes y se selecciona uno para la aplicación..

Para conseguir la interfaz ideal, primero se estudian los trabajos más recientes y sus aplicaciones. Una vez determinado el problema, se analizan de manera exhaustiva algunas interfaces actuales relativas a guante de datos seleccionado, así como los entornos de programación más utilizados para desarrollos de ambientes en tres dimensiones tales como: Visual C++, Cortona SDK, Freewrl y Java 3D.

Una vez que se ha seleccionado un guante de datos y un ambiente de desarrollo, se presenta el proceso de programación de la interfaz y la forma de interactuar con un ambiente virtual particular. Finalmente se presenta una propuesta de interfaz genérica y se proponen trabajos futuros de investigación.

Abstract

This thesis discusses the development process of a generic interface to receive data from a data glove, in order to feed a three-dimensional virtual environment to take advantage of the glove's features such as intuitive interactivity, ease of use, six -degree of freedom, etc.

First, we present the history of virtual reality to this days; we analyze the current problematic, and the importance of developing generic interfaces that support different environments. The main goal is that the student understand the following terms: geometry shapes's modeling in 3D, handling 3D objects, interaction with the user and navigation in a virtual environment, six-degree of freedom and rotation(pitch, yaw and roll). Most of the virtual reality courses use conventional devices such as Mouse and keyboard, which are not immersive. We focus our efforts in the addition of a data-glove which interact with the user, to navigate in a virtual world. In this way we can improve the learning virtual reality process, from a no immersive environment to an immersive environment.

Second, we study some current works and their applications. Once we have defined the main problem, we analyze the most popular programming environments in virtual reality such as Visual C++, Cortona SDK, Freewrl and Java 3D, and choose the best fit for our goals. After that, we present the whole process of developing our interface and the way of interaction in a particular virtual environment.

Finally, we present our proposal of a generic interface and we mention further work to be done.

Dedicatoria

Este trabajo esta dedicado a:

Mi Madre que me ha brindado su apoyo incondicional para que yo pudiera desarrollarme sin importar sacrificar su propia carrera y posponer sus estudios de posgrado. Que me ha acompañado en infinidad de situaciones siempre con un consejo sabio y una visión de triunfador. Este trabajo es tuyo, porque con cada palabra de aliento me has acercado más al éxito; quiero que sepas que eres una madre ejemplar, que me has procurado, pero además me has enseñado una de las lecciones más importantes de mi vida, como lo es el sobresalir de los demás y pensar más allá de lo convencional. Estoy orgulloso de ser tu hijo y deseo ser un padre responsable como tú, cuando llegue el momento.

Mi hermana Liz, que siempre me ha escuchado y me ha compartido su vida. Espero que este trabajo te sirva como base para que un futuro cercano tú también te desarrolles como una profesionista exitosa. Estoy convencido que eres una mujer muy inteligente, y que puedes lograr todas las metas que te propongas, porque tú al igual que yo querida hermana tienes mucho porque luchar.

Mi tía Maricela que es otra mujer en mi vida que merece todo mi respeto y admiración. Tú me has enseñado que se puede llevar una vida recta y dedicada a Dios. Con tu trabajo y tú entrega me has demostrado fortaleza, y tesón. Te agradezco porque sé que siempre te preocupas por mi familia y siempre me has brindado tu amor.

A Maribel, que me ha brindado su amistad y su interés. Gracias por confiar en mí y darme ánimos para seguir adelante.

A Hilda Alejandrina, con la cuál compartí hermosos momentos y que me ayudó al principio de éste trabajo de tesis.

A Laurita con quien he vivido momentos maravillosos e inolvidables, y que me ha comprendido y aceptado con amor y paciencia.

A Noemi, que ha sido una amiga dedicada, divertida y muy entusiasta.

A todos aquellas personas que estoy por conocer y que formarán una parte importante de mi vida.

A todas aquellas personas que tienen un sueño y no cesan hasta volverlo realidad.

Agradecimientos

A mi Padre Celestial por proporcionarme la capacidad para poder concluir mis estudios de maestría y por su infinita sabiduría y amor al concebir el Plan de Salvación.

A mi Director de tesis M. en C. Mauricio Olguín Carbajal por ayudarme a desarrollar este proyecto. Gracias a sus ideas he podido perfeccionar y enriquecer mi trabajo, su confianza hacia mi; no tengo forma de agradecer el tiempo que ha invertido revisando este trabajo, el material que me ha proporcionado así como su verdadera amistad.

Al M. en C. Juan Carlos Herrera Lozada por invitarme a formar parte de ésta generación y por todo el apoyo incondicional que siempre me ha brindado. Gracias a Ud. se me ha abierto un mundo entero de posibilidades. Le agradezco por ser una persona accesible, que ésta dispuesto a escucharme, comprenderme y que siempre me ha sabido aconsejar.

A Marlon que siempre ha sido mi compañero y amigo. Gracias por todos los grandes momentos que hemos compartido, mi estancia dentro y fuera del CIDETEC ha sido bastante divertida gracias a ti. Me has enseñado que no todo en la vida es estudiar ni trabajar, y que se la puede pasar uno bien en cualquier circunstancia.

Al M. en C. Eduardo Vega, al Dr. Miguel Lindig, al Dr. Victor M. Silva y a todos mis profesores de la maestría que me han enseñado tantas cosas y que además han sido muy amables conmigo.

A la Maestra Silvia Toledo, a Irma Alonso y a todo el personal administrativo del CIDETEC que siempre han sido muy atentos conmigo y me han apoyado en cualquier circunstancia.

A la Lic. Ma. Elena Martínez por creer en mí desde el principio y darme la oportunidad de trabajar a su lado. Quiero decirle que he aprendido mucho de Ud. y siempre le he admirado.

Al Dr. Ramón Silva que me proporciono su formato de Scientific Workplace así como sus valiosas recomendaciones para la elaboración de la presente tesis.

Al Instituto Politécnico Nacional, mi Alma Máter la cuál me ha provisto de las herramientas necesarias para desarrollarme como un profesionista exitoso. Gracias a su labor, miles de jóvenes al igual que yo son nutridos de conocimiento para colaborar en el crecimiento de México y poner la Técnica al servicio de la patria.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Antecedentes

En general la realidad virtual ha sido hasta ahora un nicho de mercado. Su precio siempre ha estado arriba del alcance de las masas, y por lo tanto no ha existido una fuerte competencia en la industria, o algún incentivo para proveer una interfaz de realidad virtual que se adapte a todas las necesidades.

En la medicina, la realidad virtual se presenta como una herramienta muy socorrida, principalmente en la cirugía, la enseñanza y el tratamiento de algunos padecimientos como las fobias. Las interfaces que se utilizan modelan el cuerpo humano ó alguna de sus partes a través de algoritmos que reproducen fotografías tridimensionales; que permiten observar el objeto de estudio a través de diferentes ángulos, y controlar a petición la información presentada en la simulación. Estas interfaces, además, deben presentar la información en formato fácil de entender y utilizar.

En el terreno del arte, las interfaces que se han implementado buscan la conversión en banda sonora de movimientos digitalizados para componer piezas musicales, para la creación de pinturas a través de una computadora, etc.

Para el usuario casero, la realidad virtual esta relacionada principalmente con la industria del entretenimiento. Esté ámbito no ha logrado consolidarse debido principalmente a que un sistema de este tipo es muy costoso y al trasladar esta tecnología a un tamaño aceptable y precio accesible se convierte en un producto de calidad poco atractiva. El número de hogares que cuentan con sistemas de entretenimiento virtual es mínimo. Sin embargo, hoy en día existen sistemas de entretenimiento in situ los cuales son una fusión de un sistema de cine en casa, videojuegos y parque temático con el fin de reproducir de manera virtual, un parque de diversiones, un simulador de baile o de peleas en un espacio reducido. De ahí se deduce que la clave del éxito en las interfaces que pertenecen a la industria del entretenimiento, es la calidad de la experiencia.

1.2. Estado del arte

Dentro del campo de estudio de la realidad virtual, se pueden mencionar varios trabajos relativos al uso de guantes de datos con el fin de proporcionar mayor manejabilidad para controlar ambientes virtuales basados en situaciones reales. A continuación se presentan los trabajos más relevantes en esta área de estudio.

En el ámbito de las misiones tripuladas a otros planetas, la implementación de guantes de datos en sistemas de control basados en gesticulaciones, proveen de métodos más naturales, eficientes y precisos que superan los medios convencionales como el joystick ó palanca de mando [1]. En el caso de los sistemas de manipulación de robots y brazos mecánicos, los controles con botones, teclados, etc., exigen habilidades adicionales por parte de los operadores ya que no existe correlación entre los movimientos del robot y el control, del mismo modo que al usar comandos en un determinado lenguaje de programación. En este caso, los guantes de datos facilitan el manejo del robot y lo hacen más seguro ya que el robot sigue el movimiento del brazo del operador [2]. En la medicina, los guantes de datos se utilizan para el diagnóstico de enfermedades en las extremidades superiores así como para la rehabilitación; dichos sistemas han mejorado la recuperación del paciente debido a que no dependen de la experiencia u objetividad del terapeuta en turno al momento de aplicar los exámenes a los pacientes [3]. Otro ejemplo son las exploraciones por computadora a nivel molecular del sistema cardiovascular, donde la cantidad y la complejidad de la información son todo un reto para los especialistas. Por lo que se utilizan técnicas de visualización y localización, para proporcionar una navegación intuitiva y natural que soporte la percepción inmersiva de las estructuras internas del organismo humano, y que permita definir regiones de interés como tumores o daños en la estructura de las arterias [4].

Para el análisis del desarrollo urbano, los guantes de datos son una herramienta muy eficiente que permite crear y modificar planes y proyectos a través de la manipulación de edificios virtuales y áreas de construcción, así como representaciones tridimensionales de las vías de comunicación, zonas residenciales, distritos o incluso ciudades enteras [5].

En el ámbito del diseño industrial, los diseñadores han encontrado una manera más rápida y atractiva de esbozar los bosquejos para sus nuevos modelos, en tres dimensiones y en tiempo real a través de la realidad virtual. De esta forma se puede migrar del uso del lápiz y papel en la primera etapa (diseño conceptual) de una manera amigable e intuitiva, proporcionando a los diseñadores la posibilidad de transferir sus puntos de vista artísticos en productos funcionales [6].

Para la administración de documentos, el diseño de interfaces que permitan la mejor comprensión, exploración y búsqueda de información al tiempo que estimulen y exploten la capacidad humana del procesamiento visual, ha sido altamente favorecido con la implementación de los guantes de datos. Eso se debe a que a pesar de que dichas interfaces pueden ser controladas por la combinación de teclas y el uso del ratón (mouse), la tarea

de seleccionar y colocar objetos en el espacio tridimensional es complicada y requiere mucho mayor esfuerzo y atención por parte del usuario que debe memorizar dichos comandos [7].

En la educación los guantes de datos han sido utilizados para estimular las habilidades motoras de los niños. A través de bloques virtuales de colores y cilindros para conectarlos, el usuario construye sus propias figuras. Este tipo de sistemas de realidad virtual presentan ventajas sobre los objetos reales, ya que el usuario puede guardar sus modelos en la computadora en estantes y tomar los bloques y cilindros de cajas de forma ilimitada [8].

Incluso en la música, en la creación de sonidos novedosos a través de instrumentos virtuales, los guantes de datos juegan un papel protagónico, ya que por medio de la combinación de diferentes gestos como la flexión de los dedos, se pueden producir cambios en la música, así como tener el control de múltiples parámetros. Éste enfoque es rápido, sencillo, y divertido tanto para el usuario experto como para el principiante [9]. En la reproducción de medios como películas, música, etc., los usuarios que han experimentado con guantes de datos e interfaces virtuales, encuentran a estos dispositivos más cómodos, fáciles de comprender y manipular en comparación con las interfaces convencionales de botones [8].

1.3. Planteamiento del problema

Un sistema de realidad virtual se compone de una serie de dispositivos más elaborados que los de un sistema de cómputo convencional. Debido a que la mayoría de estos dispositivos tienen un precio elevado, se ha generalizado el uso de ambientes virtuales de escritorio que usan simples equipos de cómputo con sus respectivos dispositivos de entrada y salida. Como ejemplo se puede mencionar VRML (Virtual Reality Modeling Language) [11] el cual es un estándar para la creación de ambientes virtuales no inmersivos, que a través de un conjunto de nodos, permite el modelado en tercera dimensión, así como el desarrollo de animaciones y de interactividad con el usuario.

Sin embargo, el aprender Realidad Virtual desde una perspectiva tan limitada como lo son los ambientes virtuales de escritorio, provoca que la curva de aprendizaje del estudiante sea menos eficiente, pues la mayor parte de los conceptos se dejan a la imaginación pues no existe una estimulación apropiada de los sentidos y por ende no se consigue una sensación de inmersión. La implementación de uno o varios dispositivos de realidad virtual a los ambientes virtuales de escritorio, eleva la sensación de inmersión en el estudiante y le permite comprenderla mejor desde dentro para el desarrollo de mejores ambientes virtuales.

Dentro del ámbito de la realidad virtual, los guantes de datos son los dispositivos más populares debido a su bajo costo, y su uso natural e intuitivo, como es el caso del guante de datos P5 [12]. Como se mencionó en el estado del arte, los guantes de datos son una

herramienta esencial para manipular ambientes de forma más natural, y se utilizan en diferentes áreas de conocimiento. El problema de programar una interfaz general para los guantes de datos es que cada programa es por naturaleza diferente y trabaja con diferentes comandos. Por ejemplo: Swift 3D, Maya y 3D Studio son ambientes similares de renderización, pero poseen diferentes características, atajos de teclado, etc. Si todos trabajaran de la misma forma, no habría necesidad de competir con diferentes productos. Esta es una de las razones por las cuales el hardware de realidad virtual tiende a ser propietario y muy costoso.

Por lo tanto, el problema que se desea resolver es, cómo mejorar el aprendizaje de la realidad virtual en un ambiente de escritorio. La propuesta es implementando un guante de datos y desarrollando una interfaz de uso general, debido a que este dispositivo de bajo costo [13] proporciona una interacción de manera natural e intuitiva para el usuario de ambientes virtuales.

1.4. Motivación para desarrollar la interfaz

La realidad virtual es un tema común hoy en día debido a que el crecimiento de la industria de cómputo en hardware, software y comunicaciones busca que la tecnología sea más amigable y que llegue al mayor número de usuarios posible.

Los dispositivos de realidad virtual como los guantes de datos, sistemas de rastreo y sistemas hápticos mejoran la experiencia del usuario, proporcionan una curva de aprendizaje mucho más eficiente que los medios convencionales como el monitor, el teclado, el ratón (Mouse), etc.

El integrar un guante de datos a un Sistema de Realidad Virtual de escritorio, permite que el usuario aproveche las ventajas de usar su mano de forma expresiva natural e intuitiva. Por lo tanto, se propone desarrollar una interfaz de uso general para un guante de datos comercial de modo que los programadores interesados en este tipo de dispositivo no tengan que empezar de cero, y puedan concentrarse en el desarrollo de mejores ambientes virtuales.

1.5. Objetivo

Con base en el problema planteado, se propone desarrollar una interfaz para el aprendizaje de la realidad virtual que permita relacionar un guante de datos comercial con un ambiente de escritorio, con el fin de acelerar la curva de aprendizaje y mejorar la experiencia de estudiantes y usuarios afines. Esto dará como resultado una aplicación de software para interactuar en un ambiente simulado a través de los movimientos de la mano, así como el conocimiento sobre el dispositivo, el entorno de programación y el documento de ésta tesis.

En cuanto a las problemáticas a resolver, se llevará acabo la selección de un guante de datos comercial para programar la interfaz. Así mismo, se escogerá un ambiente de programación flexible y portable que permita conjuntar un ambiente virtual con los movimientos de la mano en términos de traslación, rotación, y doblado de los dedos. Esto implicará desarrollar algunas clases de comportamiento que permitan integrar dispositivos de entrada no convencionales, y que por su naturaleza no estan presentes en las plataformas de desarrollo en tres dimensiones, así como llevar acabo transformaciones sobre objetos virtuales en tiempo de ejecución.

Los medios para la realización de esta tesis serán:

- Una computadora para realizar las pruebas.
- Software para transcribir la tesis (Scientific WorkPlace).
- La adquisición de un guante de datos comercial.
- Software para establecer la comunicación con dicho guante.
- Software para modelar ambientes virtuales.
- Libros y documentación sobre la plataforma de desarrollo.
- Metodología para el modelado en tres dimensiones.

1.6. Metas

Una vez que se ha planteado el problema y el objetivo de ésta tesis, se presentan a continuación las metas:

- Hacer un estudio de la realidad virtual y los guantes de datos.
- Seleccionar un guante de datos comercial para programar una interfaz.
- Seleccionar el entorno de programación más adecuado para el desarrollo de la interfaz guante-máquina.
- Descripción del proyecto, considerando el perfil del usuario e indicando los recursos necesarios para poder establecer el modo de interacción con el usuario.
- Diseño de la mano virtual que representará los movimientos del usuario.
- Desarrollar una interfaz (software) que permita el reconocimiento del guante, y permita capturar la posición, rotación y traslación del guante en tres dimensiones y con 6 grados de libertad.
- Implementación de la interfaz con la mano virtual.

1.7. Contenido de la tesis

Este trabajo está dividido en seis capítulos y un apéndice: Capítulo 1. Introducción, Capítulo 2. Realidad virtual y los guantes de datos, Capítulo 3. Hipótesis, Capítulo 4. El ambiente de desarrollo, Capítulo 5. Desarrollo, Capítulo 6. Conclusiones y perspectivas y el Apéndice A. Manual de usuario.

En el Capítulo 1 se abordan los antecedentes al problema medular de ésta tesis como son las interfaces de realidad virtual. Se comentan algunos trabajos actuales relevantes en la navegación espacial, control de robots y brazos mecánicos, medicina, análisis de información, música, desarrollo urbano, diseño gráfico y educación. También se plantea el problema, la motivación para desarrollar una interfaz para el aprendizaje de la realidad virtual y se establecen las metas.

En el Capítulo 2 se presenta una reseña de la simulación, de la realidad virtual como una rama de la simulación así como sus orígenes y trabajos más representativos hasta la actualidad. A continuación se presentan los componentes de un sistema de realidad virtual y se comparan con un sistema de cómputo convencional. También se presenta el concepto de interfaces humanas, los guantes de datos, su historia y se selecciona un guante comercial para trabajar la interfaz de ésta tesis.

Una vez definido el problema y seleccionado un guante de datos, en el Capítulo 3 se plantea la hipótesis, se proponen diferentes caminos de solución y se efectúa un análisis a fondo de las alternativas, para finalmente seleccionar la opción más adecuada.

En el Capítulo 4 se presenta el ambiente de desarrollo y las herramientas para programar la interfaz. Para esto se profundiza en el guante P5, algunas clases propuestas para establecer comunicación con él, el lenguaje de programación Java 3D su estructura y algunos ejemplos.

En el Capítulo 5 se presenta el desarrollo realizado en Java/Java3D para la realización de la interfaz del guante de datos P5, desde las pruebas con el lenguaje de programación, así como los problemas que fueron surgiendo, las alternativas de solución, su evaluación y la interfaz final.

En el Capítulo 6 se presentan las conclusiones obtenidas durante la tesis, así como algunas propuestas para trabajos futuros de investigación.

En la parte final del trabajo, se presenta el apéndice A el cual contiene el manual de usuario de la interfaz, donde se explican los pasos a seguir para instalar la interfaz, así como su funcionamiento.

Capítulo 2

Realidad virtual y los guantes de datos

En este capítulo se presenta un panorama global de la realidad virtual y los guantes de datos, como dispositivos que estimulan la sensación de inmersión en ambientes virtuales. Primero, se proporciona una reseña cronológica de los desarrollos más importantes hasta nuestros días. A continuación se definen los componentes de un sistema de realidad virtual y se comparan con los de un sistema de cómputo convencional. Sucesivamente se definen las interfaces humanas y se explica el proceso por el cual se aprovecha el conocimiento que se tiene de los sentidos humanos y de las señales que transmiten al cerebro para simularlas y provocar en el usuario una sensación de inmersión. Subsecuentemente se analizan algunos desarrollos actuales que utilizan uno o más dispositivos de realidad virtual. Por último se presentan las características de los guantes de datos, su historia algunos modelos comerciales recientes de los cuales se describen sus ventajas y desventajas para posteriormente seleccionar alguno para la elaboración de la interfaz de la presente tesis.

2.1. Simulación

La simulación es la representación de un sistema real o imaginario, a partir de un modelo, el cuál toma los aspectos más importantes de éste para convencer al usuario común y para experimentar con el sistema como si se experimentara con el sistema que se esta simulando.

En 1953 Fermi creo la primera simulación virtual de la historia, al conectar 64 computadoras en red para el estudio de la entropía en 64 partículas atómicas. Desde entonces, y gracias al rápido avance de la tecnología digital, las simulaciones pueden recrear sistemas reales con una fidelidad impresionante. Es por eso, que el mundo científico se apoya en la simulación para realizar sus investigaciones, y éstas por ende, han llegado al mundo empresarial.

Desde simular un juego de tenis ó el vuelo de un avión hasta el colapso de una arteria, las conexiones químicas de una molécula, un ecosistema ó la reproducción de la vida entre otras.

Para poder construir buenas simulaciones, se requiere una visión completa del sistema que se busca simular. Se dice que la clave de cualquier simulación esta en la traducción de los algoritmos matemáticos de cada una de las relaciones que se producen en el sistema que se quiere imitar [14]. Sin embargo, la representación de la realidad en algoritmos matemáticos no siempre es tarea sencilla, pues por más avanzada que esté la tecnología, sigue siendo finita. Además los costos de poderosos equipos de cómputo estan fuera del presupuesto de muchas universidades y centros de investigación, lo que conduce a reducir el número de variables a intervenir en las simulaciones.

2.2. Realidad Virtual como caso especial de la simulación

Hoy en día, la realidad virtual es un término muy usado, principalmente en el medio del entretenimiento. En términos de funcionalidad, la realidad virtual es una simulación en la cual un conjunto de gráficos generados por computadora, son usados para crear un mundo realista. Más aún, el mundo virtual no es estático, ya que responde a las acciones del usuario (gestos, comandos por voz, etc.). Esto define la clave principal de la realidad virtual, la cuál es interactividad en tiempo real. La interactividad en tiempo real significa que la computadora es capaz de detectar la entrada del usuario y aplicar los cambios necesarios en el mundo virtual de forma instantánea. Esto atrae a los usuarios, que pueden observar como se modifica su entorno en la pantalla en respuesta a sus acciones y por ende quedan cautivados [15]. Se puede decir que la realidad virtual consiste en [16]: "La simulación de medios ambientes y de los mecanismos sensoriales del hombre por computadora, de tal manera que se busca proporcionar al usuario la sensación de inmersión y la capacidad de interacción con medios ambientes artificiales".

La realidad virtual no es una invención nueva, pues tiene su origen hace más de 40 años. Esta surgió como un caso especial de la simulación, al tratar de modelar con computadoras ambientes inaccesibles para el ser humano, como el estudio de moléculas o incluso la atmósfera en otros planetas.

A continuación se presenta una breve reseña del desarrollo de la realidad virtual de forma cronológica:

- En 1962 fue promovida la patente No. 3, 050,860 por Heilig para su invención titulada Simulador Sensorama, el cual fue la primera máquina de video de realidad virtual. Esta estación de trabajo de realidad virtual tenía video en 3D (obtenido a través de un par de cámaras de 35 mm), movimiento, color, sonido estéreo, aromas,

y efectos de viento (usando pequeños ventiladores colocados cerca de la cabeza del usuario), y un asiento que vibraba. Con este invento era posible simular un viaje en motocicleta sobre Nueva York, donde el conductor sentía el viento y los agujeros en el camino al vibrar el asiento. El conductor incluso podía oler comida al pasar por un restaurante. Posteriormente, todavía en la decada de los 60, Sutherland y otros perfeccionaron el casco visor HMD mediante el cual un usuario podía examinar, moviendo la cabeza, un ambiente gráfico.

- En 1969, Krueger creó ambientes interactivos que permitían la participación del cuerpo completo, en eventos apoyados por computadoras. También en ese año, la NASA puso en marcha un programa de investigación con el fin de desarrollar herramientas adecuadas para la formación, con el máximo realismo posible, de posteriores tripulaciones espaciales.

- En 1970, Brooks logra que los usuarios muevan objetos gráficos mediante un manipulador mecánico. También en esos años, Minsky acuña el término "TELEPRESENCIA", para definir la participación física del usuario a distancia. A fines de los años 70, en el Media Lab. del Instituto Tecnológico de Massachusetts (MIT), se obtiene el mapa filmado de Aspen, una simulación de vídeo de un paseo a través de la ciudad de Aspen, Colorado. Un participante puede manejar por una calle, bajarse y hasta explorar edificios.

- En 1980, Gibson publica la novela "Neuromancer"donde la trama se desarrolla con base en aventuras en un mundo generado por computadora al que denomina CIBERESPACIO.

- En 1982, los estudios Disney producen la película "TRON". TRON fue una de las primeras películas en utilizar gráficos generados por ordenador, además de tener su propio estilo visual En el mismo año, Zimmerman inventa el Dataglove y Lanier acuña el término de Realidad Virtual, concretando la variedad de conceptos que se manejaban en esa época.

- En 1984, McGreevy y sus colegas de la NASA desarrollan lentes de datos con los que el usuario puede ahora mirar el interior de un mundo gráfico mostrado en computadora.

- En 1987 la NASA perfecciona la primera realidad sintetizada por computadora mediante la combinación de imágenes estéreo, sonido 3-D, guantes, etc.

- En 1990 surge la primera compañía comercial de software de realidad virtual, Sense8, fundada por Gelband. Esta compañía, ofrece las primeras herramientas de software portables a los sistemas SUN.

- En 1992 Leonard produce la película “El cortador de césped”. Esta película incorpora imágenes generadas por computadora que representaban los entornos virtuales, así como la utilización de cascos de visión y guantes de datos

- En 1994 en Inglaterra se funda la Sociedad de Realidad Virtual.

- En 1995 Sony Pictures produce "Johnny Mnemonic", un thriller de ciencia ficción, que reproduce digitalmente el ciberespacio de una forma viva e imaginativa. La realidad virtual era el medio donde se desarrolla la acción de la película. En el mismo año, Nintendo lanza un novedoso sistema de juego llamado "Virtual Boy", una novedosa consola semiportátil parecida a los cascos de realidad virtual de los salones recreativos que, gracias a un trípode, se situaba encima de una mesa y el jugador tenía que aproximarse a los visores para apreciar un efecto tridimensional en los juegos. Además aparece "Toy Story". Es una película de animación generada por computadora de Pixar y Walt Disney Pictures. Fue el primer largometraje totalmente animado por computadora (77 minutos), y el primer proyecto importante de Pixar en el cine. La renderización de la película corrió por cuenta de Sun Microsystems la cuál utilizó una red de 117 estaciones de trabajo para los 114,000 marcos (frames) que conforman la pelicula.

- En 1999 Silver produce la película "Matrix" .La película parte de una premisa: existen dos realidades, una que consiste en la vida que vivimos cada día, y otra que se encuentra detrás de ella. Una es un sueño, la otra es THE MATRIX; un sistema gigante de computadoras que reproduce el mundo con todos sus habitantes e inyecta un falso sentido de realidad en los seres humanos.

- En el año 2000 Nintendo presenta el "GameCube". Esta nueva consola se concentró bastante en el aspecto gráfico desarrollado por la compañía ATI Technologies, así como la implementación de un potente procesador desarrollado por IBM. Esto da como resultado una capacidad real de procesamiento gráfico: de 6 a 12 millones de polígonos por segundo. (capacidad de procesamiento estimada en un juego real incluyendo modelados complejos, texturas, etc). En el mismo año Sony lanza al mercado el Play Station 2, provisto de características avanzadas incluyendo la reproducción de DVD, soporte para USB y FireWire, y ranuras de expansión. Esta consola posee una transformación Geométrica 3D equivalente a 66 millones de polígonos por segundo sin texturizar, sin efectos; con efectos y texturas maneja unos 12 millones de polígonos por segundo.

- En 2001 Square Pictures produce "Final Fantasy",.primera película animada completamente en 3-D. que ha sido realizada con una sorprendente y avanzada técnica de animación por computadora como el escaneado facial en los personajes que supera la percepción del espectador.

- En el 2002 aparece la "Era de Hielo", una película de animación generada por computadora, dirigida por Forte y la incursión exitosa al mercado de 3D de los estudios Fox.

- En el 2005 sale a la venta la consola Xbox 360, una consola de videojuegos diseñada por Microsoft para reemplazar la Xbox. En cuanto al hardware cuenta con

un CPU desarrollado por IBM y un GPU diseñado por ATI con memoria integrada desarrollada por NEC Corporation, dispone de una interfaz SATA, tarjeta de red, sistema I/O (entrada/salida) creado por SIS y soporte para controles alámbricos e inalámbricos compatibles con la nueva versión de Windows. Reproduce 500 millones de polígonos por segundo superando por 4 veces los 116 millones del primer Xbox.

- En el 2006 sale a la venta el Nintendo Wii. La principal característica de Wii es el control inalámbrico de la consola, bautizado como Wii Remote o también Wiimote por su parecido a un control remoto de TV (Remote Control en Inglés), que es capaz de detectar el movimiento y rotación en un espacio de tres dimensiones. El mando dispone de funciones de vibración, mientras que la última versión del mando integra además un altavoz. Otro aspecto importante de la consola es el modo "stand-by WiiConnect24", que permitirá recibir mensajes y actualizaciones a través de Internet con un consumo de energía muy bajo. Se incluye además el concepto de avatars (representación gráfica mediante un dibujo o fotografía de una persona para su identificación) para personalizar el usuario en los juegos.

2.3. Componentes de un sistema de Realidad Virtual

Para satisfacer las necesidades de un usuario, un sistema de realidad virtual debe realizar un gran número de operaciones en el menor tiempo posible; es decir, requiere hacer uso de diferentes tecnologías para cumplir su cometido. Dentro de las más importantes se encuentran el cómputo en paralelo, las redes estandarizadas y la inteligencia artificial.

Las funciones principales que debe llevar a cabo un sistema de Realidad Virtual son:

- Presentación de objetos en tres dimensiones.
- Posicionamiento y rastreo de los objetos que conforman el mundo virtual, así como del propio usuario de forma interactiva.
- Simulación del comportamiento de los objetos en base al modelo propuesto.
- Interacción del usuario con los elementos del mundo virtual.
- Generación del sonido y ambientación del mundo virtual.

Las diferencias entre un sistema de cómputo convencional y un sistema de realidad virtual son:

- Un sistema de cómputo convencional, esta compuesto de dispositivos de entrada como son el teclado, el ratón y dispositivos de salida como el monitor y las bocinas. Dichos dispositivos están diseñados para recibir órdenes del usuario, pero no para sumergirlo en una realidad diferente de la que tiene enfrente, una máquina inerte que difícilmente responde a sus peticiones de manera natural y atractiva

- El objetivo de un sistema de realidad virtual es la simulación de un mundo virtual regido por normas específicas, que pueden o no, diferir de las normas del mundo real. Los componentes que lo conforman invitan al usuario a olvidarse de la realidad y formar parte del ambiente virtual, ya que éste no puede existir sin la interacción activa del usuario.

- La información que se despliega en un sistema de cómputo convencional difícilmente se presenta en tres dimensiones.

- Un sistema de realidad virtual se compone de una serie de dispositivos de entrada más elaborados como son los guantes de datos, sistemas de rastreo y de retroalimentación de la fuerza (háptica). Para los dispositivos de salida existen los cascos HMD (Head Mounted Display), los lentes LCD (Liquid Crystal Display), etc., para atender a uno o más sentidos del usuario.

- El software que se ejecuta en un sistema de realidad virtual, es más complejo al igual que el poder computacional que se requiere para coordinar los dispositivos mencionados anteriormente.

En la Figura 2.1, se pueden apreciar los elementos que conforman un sistema de realidad virtual.

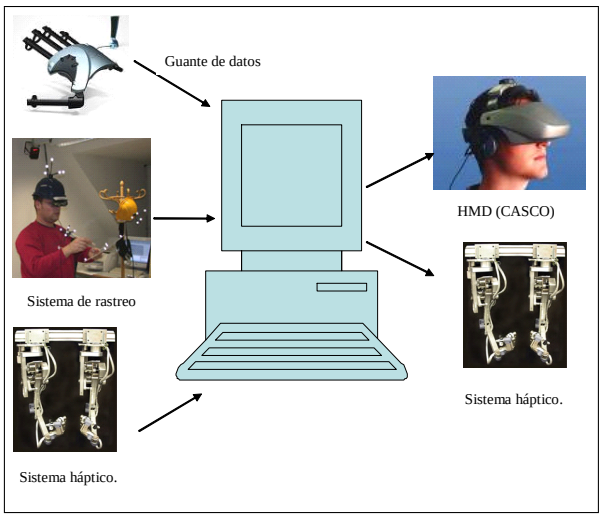

Figura 2.1: Elementos que conforman un sistema de realidad virtual.

2.4. Interfaces humanas

Para que la realidad virtual produzca la sensación de estar inmersos en un mundo tentativamente real, se requiere de estimulaciones en el sentido visual, auditivo y táctil. Las interfaces humanas pretenden explotar la tecnología y el conocimiento que se tiene de los sentidos humanos, para simular las señales y estimulos que se reciben en la vida diaria. A continuación se presenta una breve semblanza de dichos sentidos.

2.4.1. Sentido visual

Generalmente los seres humanos obtienen la mayor parte de su conocimiento del mundo a través de los ojos. El sentido visual procesa la información de dos maneras distintas: consciente e inconscientemente. Cuando se contempla una fotografía, se lee un libro o al consultar un mapa, se requiere procesar concientemente las imágenes lo cuál necesita de ciertas habilidades que se adquieren con el paso del tiempo.

Sin embargo, el procesamiento visual inconsciente describe la capacidad básica de percibir luz, color, forma, profundidad y movimiento. Dicho procesamiento es autónomo, y por lo tanto, no se está consciente de cómo y cuándo sucede.

Físicamente los ojos son órganos sumamente especializados. Células expertas forman estructuras complejas las cuales llevan acabo varias funciones –la pupila actúa como un filtro que controla el flujo de luz que entra en el ojo; el cristalino enfoca la luz a través del uso de músculos que cambian su forma y la retina es el motor que transforma la luz en impulsos eléctricos para ser procesados posteriormente en el cerebro. El cerebro realiza el procesamiento visual al dividir la información neuronal en fragmentos pequeños que pasan a través de muchas neuronas filtradoras. Algunas de estas neuronas sólo detectan cambios drásticos en el color mientras otras detectan sólo ejes verticales u horizontales.

La información completa converge de diferentes maneras. Para obtener una visión estereoscópica, se extrae información relevante como resultado de comparar el ojo izquierdo con el ojo derecho.

La sensación de inmersión visual en realidad virtual viene de varios factores que incluyen el campo de visión, la tasa de actualización de marcos, y el rastreo ocular. Un campo de vista limitado puede dar como resultado la sensación de estar en un túnel. Las tasas de actualización deben ser suficientemente altas para permitir a nuestros ojos mezclar en conjunto los marcos individuales para formar la ilusión de movimiento y limitar la sensación de latencia entre los movimientos de la cabeza y el cuerpo así como la regeneración de la escena.

Al proceso de generar una escena por computadora a partir de un modelo matemático se le denomina renderizar". El rastreo ocular puede ayudar a reducir la carga computacional cuando se renderizan los marcos, dado que se recomienda renderizar en la más alta resolución sólo cuando los ojos están mirando un objeto particular.

La sensación de inmersión virtual es usualmente alcanzada vía algunos medios de posicionamiento y rastreo. Dentro de los medios de rastreo más comunes se incluyen los ópticos, ultrasónicos, electromagnéticos y mecánicos. Todos estos medios han sido usados en varios cascos HMD (Head Mounted Display); estos vienen en tres variedades incluyendo estereoscópicos, monoculares y acoplados en la cabeza.

2.4.2. Sentido auditivo

Los oídos forman la parte más visible del sistema auditivo, guiando ondas sónicas al canal auditivo. El canal por si mismo realza el sonido escuchado y lo dirige al tambor, el cual convierte las ondas de sonido en vibraciones mecánicas. En el oído medio tres pequeños huesos; el martillo, el yunque y el estribo, forman un puente alrededor de un vacio de aire y amplifican el sonido por un factor de 30. El estribo rota hacia fuera y el tambor se endurece para inhibir sonidos fuertes. El oído interno traslada estas vibraciones mecánicas a través de señales electroquímicas para ser procesadas por el cerebro.

En los sistemas de realidad virtual el sonido generado por computadoras llega de diferentes formas. El uso de sonido estéreo añade cierto nivel de retroalimentación al ambiente virtual, pero no se asemeja en su totalidad al mundo real. El sonido estéreo se refiere a los sistemas de audio que usan más de un canal para recrear una experiencia más natural de las fuentes de donde proviene el sonido. Cuando se usan sonidos en 3D, es posible colocar sonidos en el ambiente virtual usando señales de rastreo de manera similar al sistema auditivo. El sistema de sonido en 3D debe reaccionar en tiempo real para desplazar los sonidos en el ambiente virtual.

2.4.3. Sistema táctil

La sensación de tacto es realizada por lo que se conoce como sistema háptico ó táctil. El sistema táctil libera información de la siguiente manera: un receptor mecánico provee información acerca de la forma, textura, y temperatura; la información táctil converge en el cerebro por fibras adaptables que sensan el movimiento o resistencia al movimiento, el peso de un objeto y su firmeza por medio de la interacción de los músculos.

En los sistemas de realidad virtual, los dispositivos táctiles y de retroalimentación de la fuerza buscan emular las señales táctiles que el sistema háptico humano alimenta al cerebro. Los guantes de datos permiten la manipulación de objetos de una manera más natural que el tradicional ratón, debido a que explota los movimientos de los cinco dedos, sus respectivas combinaciones así como la rotación de la mano. El guante VPL (compañía de productos de realidad virtual que se encuentra en Palo Alto California, E.U.), es quizás el más conocido. A través de una serie de fibras ópticas detecta la doblez de los dedos; y usa sensores magnéticos para detectar la posición y orientación de la mano.

El guante Power Glove de Mattel obtuvo una gran popularidad en el círculo de los hackers, debido a su bajo costo. Este guante usa tinta eléctrica resistiva para sensar el doblez de los dedos y sensores ultrasónicos para detectar la orientación y la posición de la mano.

Hoy en día el guante de datos más popular es el P5 de Essential Reality. Este dispositivo diseñado en principio para resaltar la emoción de los juegos de computadora, hoy en día es base de muchos proyectos de fanáticos de la realidad virtual e investigadores. Funciona por medio de sensores infrarrojos para detectar la posición y orientación de la mano, así como de sensores autónomos para detectar la flexión de cada uno de los dedos.

2.4.4. Ejemplos de Interfaces de Realidad virtual

A continuación se presentan tres interfaces de realidad virtual inmersiva con uno o más dispositivos de realidad virtual no convencionales. Se comentan las características principales, las ventajas y desventajas de dichos proyectos. Éstos provienen de México, España y Estados Unidos.

Prototipo de un sistema de captura de Movimiento [31]

El prototipo de éste sistema se divide en tres partes: un conjunto de sensores que detectan el movimiento de las articulaciones de una mano, un circuito que permite digitalizar las señales de estos sensores y una aplicación encargada de adecuar y visualizar la información. Para ello se utiliza una PC Pentium a 450 Mhz, con 128 MB de memoria RAM y una tarjeta de video de 32 MB. El software utilizado para la interfaz es Java 3D y Java Communications. El circuito esta compuesto por 64 sensores y la comunicación se hace de manera serial. La Figura 2.2 muestra la interfaz conectada por un lado a la computadora y por otra al circuito convertidor. La principal ventaja de este prototipo de un sistema es su bajo costo, lo cual lo hace accesible para proyectos de bajo presupuesto. Además, los sensores flexibles se acoplan a las articulaciones y miden en forma directa la flexión. Este proceso no depende del medio ambiente que los rodea, como sucede con otros métodos que utilizan campos magnéticos o procesamiento de imágenes. Las desventajas de este prototipo es que al utilizar el puerto serie la velocidad de transmisión es muy baja (9600 bps.); hasta el momento sólo puede capturar la flexión de los dedos, por lo que falta desarrollar la posición y orientación de la mano con los 6 grados de libertad y finalmente el tamaño y forma del prototipo resulta incómodo.

Hi-Tech PFCS [32]

Es un sistema que manipula el ratón (mouse) a través de un sistema de rastreo de un guante con diodos a través del procesamiento de imágenes obtenidas desde una cámara web. Se compone de dos módulos: el primero presenta al servidor como si fuera un ratón USB(Universal Serial Bus) en vez de una computadora de bolsillo (Zaurus), y un

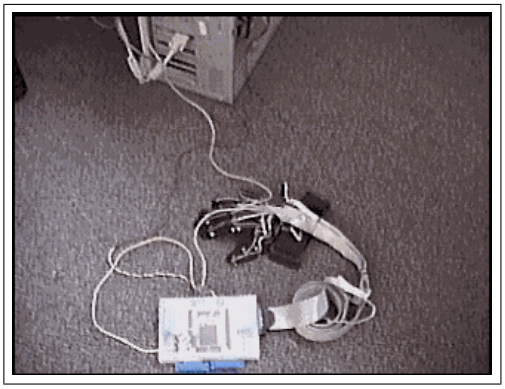

Figura 2.2: Prototipo de un sistema de captura de movimiento.

programa en la capa de usuario que reconoce el movimiento y lo manda al controlador asociado al módulo.

En cuanto al código, se utiliza GPLed y -la parte del proyecto en Linux. La Figura 2.3 muestra el guante de lana con diodos, y la computadora de bolsillo. Esta interfaz provee un método alternativo de localizar la posición del guante; sin embargo la precisión no es del todo aceptable. Por supuesto que tampoco se manejan los seis grados de libertad, ya que el propósito de ésta interfaz es sustituir al tradicional mouse de 2D, y mucho menos las flexiones de los dedos.

Figura 2.3: Hi-Tech PFCS.

Entrenador de football virtual [33]

El objetivo de este sistema es entrenar a un jugador para que analice las diferentes situaciones que pueden ocurrir en un juego, y entrenarlo para responder de la manera más rápida a los movimientos del equipo contrario. Este sistema de software integra las siguientes funciones: modelado y edición de jugadas en dos dimensiones, creación en

tres dimensiones de las jugadas (incluyendo los jugadores), control de la animación en la caverna (CAVE) y distribución de la animación en 3D vía web.

Los elementos que componen el entrenador virtual son: una caverna (cuarto de 10x10x10 pies). Estas cuatro paredes sirven como pantallas de proyección para imágenes estereoscópicas generadas por computadora al exterior de la caverna. Los usuarios entran a la caverna usando unos lentes LCD que se conocen como *shutter glasses* y que proveen de visión en tres dimensiones. La siguiente Figura 2.4 presenta a un usuario dentro de la caverna participando en una jugada programada.

Figura 2.4: Entrenador de football virtual.

Los entrenadores y jugadores pueden estudiar diferentes formaciones simuladas y variaciones en las jugadas desde cualquier punto de vista. Dichas animaciones pueden ser distribuidas por internet o vía disco compacto. La principal desventaja de este sistema es que se basa exclusivamente en el aspecto visual y no incluye retroalimentación háptica de ningún tipo y tampoco existe localización del jugador.

2.5. Guantes de datos

Los guantes de datos son uno de los dispositivos de rastreo y manipulación de ambientes virtuales más populares que existen hoy en día. El usuario se coloca un guante el cual posee sensores para detectar la flexión de los dedos, y además sensores que permiten rastrear la posición de la mano, así como su rotación y traslación. Estas medidas son reflejadas en una mano virtual la cual reproduce los movimientos del usuario en uno o más objetos virtuales [17].

Existen dos tecnologías principales que se utilizan para la detección del doblado que consisten en:

1. El uso de líneas de fibra óptica que recorren los dedos que trabajan en conjunto con unos fotodetectores que detectan la intensidad de luz, la cual tiende a disminuir al doblar un dedo. Ejemplos, Dataglove VPL.

2. El uso de tiras de metal cubiertas con líquido conductor el cual cambia la resistencia eléctrica cuando los dedos son flexionados. Ejemplos, Matel Power Glove y el P5 glove.

Además existen diferentes técnicas para rastrear la posición del guante, así como la rotación del mismo en tres ejes coordenados.

1. Rastreo ultrasónico. Este rastreo consiste en una serie de bocinas las cuales envían señales acústicas a un detector; estas crecen o se atenúan en base a la distancia del guante.

2. Rastreo óptico. Este rastreo se basa en el procesamiento de fotografías tomadas vía una cámara digital del guante, el cual esta provisto de leds que producen luz resaltando ciertas partes del guante, y eso se convierte a coordenadas vía software.

3. Rastreo infrarrojo. Consiste en un guante el cual posee transmisores infrarrojos que son detectados a través de una torre receptora, y ésta a su vez detecta las coordenadas del guante, así como la rotación del mismo.

Al combinar la detección del doblado, con las técnicas de rastreo se obtiene un dispositivo natural e intuitivo para el ser humano con un mundo virtual. Gracias al uso de guantes de datos, es posible brindar una experiencia más atractiva al usuario, ya que la mayoría de los dispositivos convencionales como el Mouse o el joystick, se encuentran limitados para ambientes en dos dimensiones, pues no permiten el movimiento natural del ser humano para manipular objetos.

2.5.1. Historia

A continuación se presenta la Tabla 2.1, que contiene las aportaciones más relevantes en cuanto a la historia de los guantes de datos.

1977	Sandin y Sayre inventan un guante sensitivo a la flexión.
1981	Grimes, asignado a Bell Telephone Laboratories, patentó un guante para introducir datos.
1982	Zimmerman patentó un guante llamado Data Glove, para introducir datos, basado en sensores ópticos, de modo que la refracción interna puede ser correlacionada con la flexión y extensión de un dedo.
1986	VPL saca a la venta el VPL Dataglove – el primer guante comercial con un costo aproximado de $9000 dolares
1987	La NASA utilizando algunos productos comerciales, perfecciona la primera realidad sintetizada por computadora mediante la combinación de imágenes estéreo, sonido 3-D, guantes, etc
1987	Zimmerman et al. Desarrolla un guante interactivo.
1989	Stone y Hennequin coinventaron el guante Teletact I.
1989	Goddard y Davis de Nintendo diseñan el Power Glove
1990	Hennequin y Stone, asignados por ARRL, patentaron un guante de retroalimentación tangible.
1991	Holmes, asignado por Industrias W, patenta un guante de retroalimentación tangible.
1992	Zimmerman, asignado por VPL Research, patentó un guante usando sensores ópticos.
2000	Kappen y Fekete crean el guante P5 para la compañía Nytric. Un año después ganan el premio Solidworks Grand Prize por la creación del mismo.

Tabla 2.1 Historia de los guantes de datos.

Como se puede observar, los guantes de datos fueron uno de los primeros dispositivos de realidad virtual inmersiva. A través de los años han ido evolucionando con el fin de encontrar nuevas alternativas para obtener la información más precisa acerca de la posición de la mano, la rotación y los gestos del usuario.

2.6. Selección de un guante de datos

La elección de un guante de datos para trabajar en ambientes virtuales no es una tarea sencilla debido a qué existen pocos dispositivos y la información que se presenta en la mayoría de los casos es provista por los propios vendedores. Hacen falta más fuentes formales que comparen las características de forma objetiva como en su momento lo hicieran Zeltzer y Stuart [18]. Sin embargo, existen factores que son primordiales que no se pueden ocultar como es el factor económico, las conexiones entre el guante y la computadora, el módulo de alimentación ya sea externo o interno, y las herramientas para programar disponibles.

En la tabla 2.2 se presenta una relación de algunos guantes de datos, una breve descripción y su precio [13]. Dentro de los guantes considerados se decidió seleccionar

el guante P5 glove debido a que reúne las características necesarias para el desarrollo de la interfaz del presente trabajo de tesis. El P5 es el guante de datos más económico en este momento; además posee una conexión USB que supera al puerto serie RS-232 del guante Pinch Glove y DG5-VHand; en cuanto al modulo de alimentación el P5 glove obtiene la energía del mismo puerto; esta provisto de 3 botones programables y uno de encendido/apagado, y a diferencia de los demás puede ser programado por diferentes lenguajes de programación entre ellos Visual Basic, Java, Delphi, y C++.

Por otro lado, el P5 también posee algunas limitantes como la necesidad de cable del guante a la torre receptora, la falta de sensibilidad en las yemas de los dedos, y la necesidad de filtrar los datos referentes a la posición del mismo. Sin embargo, la torre receptora es transportable y viene integrada con el dispositivo a diferencia de los guantes profesionales que requieren equipo adicional para el rastreo y no están al alcance del usuario común [7].

IMAGEN	NOMBRE Y DESCRIPCIÓN	PRECIO
	5DT Data Glove 5 Ultra. Ideal para animación realística en tiempo real.	995 dólares.
	5DT Data Glove Ultra Wireless Kit. Kit tipo Plug-and.play para el guante 5DT Ultra. Con compatibilidad Bluetooth.	1495 dólares.
	DG5-VHand. Guante de datos Profesional Multipropósito.	495 dólares.
	P5 Glove. Guante de datos económico y muy versatil.	59 dólares.
	Pinch Glove. Soporta una gran variedad de gestos con las yemas de los dedos.	1899 dólares

Tabla 2.2 Guantes de datos comerciales

Capítulo 3

Hipótesis

En este capítulo se presenta la formulación de la hipótesis de la presente tesis con base en los capítulos anteriores. Una vez que se define la hipótesis, se presentan los posibles caminos de solución y se selecciona el más adecuado. Para ello se estudia el concepto de interfaz, se analizan diferentes plataformas de programación tales como: Visual C++, Cortona SDK, Freewrl, Java/Java 3D, y de acuerdo a los resultados obtenidos se selecciona la plataforma más adecuada.

3.1. Desarrollo de una interfaz genérica para un guante de datos

Una interfaz es un conjunto de rutinas de software para recibir la información de un dispositivo periférico y hacerle los cambios necesarios para que otro programa pueda aprovechar los recursos de este dispositivo. Una vez que se ha desarrollado una interfaz, ésta puede ser utilizada para diferentes aplicaciones, ya que se establece la comunicación por medio de métodos y atributos de una forma abstracta y sencilla, del mismo modo que los programas de usuario se comunican con la impresora u otros dispositivos a través del sistema operativo. Más aún, el programador no tiene que conocer los detalles técnicos de comunicación del dispositivo con la interfaz, ya que el programador solamente conoce los métodos que pone a su disposición la interfaz para comunicarse con dicho dispositivo y que devuelven datos concretos, que optimizan el desarrollo de nuevas aplicaciones.

Por lo tanto, si se desarrolla una interfaz que permita simular los gestos y movimientos de la mano a través de un guante de datos en un ambiente virtual de escritorio, se acerca a los estudiantes a un sistema de realidad virtual inmersivo, y como consecuencia se les ayudará a comprender mejor los conceptos y acelerar su curva de aprendizaje.

Una vez que se ha seleccionado un guante de datos comercial, el siguiente punto es escoger una plataforma de desarrollo que sea compatible con dicho guante. Como se mencionó en el Capítulo 1, los diferentes ambientes de renderización utilizan comandos distintos y no se apegan a un estándar en común. Por lo tanto se debe seleccionar un programa de desarrollo conocido y accesible, puesto que una de las metas del presente

trabajo de tesis, es que los estudiantes de realidad virtual puedan aprovechar las características de un guante de datos para navegar dentro de sus ambientes virtuales, y no se tengan que preocupar por detalles técnicos de éste dispositivo.

3.2. Presentación de software para programar ambientes virtuales

A continuación se presentaran los distintos entornos de programación que fuerón evaluados para desarrollar la interfaz generica.Los factores a evaluar fueron seleccionados como producto de una ardua investigación en grupos de discusión relativos al guante y diferentes páginas Web en Internet. Esto principalmente porque no existe literatura al respecto debido que el guante fue diseñado con el propósito de entretener. La búsqueda comprendió entornos de programación para la plataforma Windows y Linux, asi como diversos estandares de modelado en 3D como VRML, Active X y Java 3D.

Una vez presentados los entornos de programación más relevantes y conforme se realizaron las pruebas hasta donde fue posible experimentar; se reportan los factores técnicos y se selecciona aquel ambiente de programación que satisface las expectativas del proyecto como son: la portabilidad, documentación para programadores, costo e interoperabilidad con ambientes virtuales.

3.2.1. SDK oficial con Visual C++

Este SDK (*Software Development Kit*) fué diseñado por Essential Reality para invitar a los programadores a realizar aplicaciones y juegos principalmente para el Guante P5 Glove.

En el momento en que se escribió el presente trabajo de tesis, ya no existían enlaces para obtener una versión del mismo de la página oficial de Essential Reality [19] aunque es posible que se pueda encontrar en algunas páginas de aficionados al guante [12].

Este SDK se compone de un archivo de instalación, una copia del manual del guante (igual a la que viene con dicho guante), una carta de agradecimiento para el desarrollador y un folleto informativo bastante pobre.

Una vez que se instala el SDK, se crea una carpeta de nombre P5SDK la cual contiene algunos ejemplos y su respectivo código y lo principal, que es el archivo *CP5DLL.dll* que se copia en la carpeta de \Windows\system32\ y que es la biblioteca de funciones para el guante. Todos los programas que se desarollen con este SDK, deben crear un objeto *CP5DLL()* para poder leer los datos del guante. Más que un ambiente de desarrollo, podemos decir que es una biblioteca con funciones para obtener los datos del guante. Aunque el uso de dicha biblioteca es gratuita, se debe instalar Visual C++ para programar la interfaz y Active X para el diseño del ambiente virtual, los cuales son programas propietarios de Microsoft y funcionan exclusivamente para la plataforma Windows.

Dentro de la carpeta P5SDK se encuentran varias carpetas con aplicaciones hechas en Visual C++ y Direct X. Sin embargo, se requiere tener instalado el SDK de Direct X 8.1 para ejecutarlas todas desde el código fuente, pues al momento de compilar hacen falta algunas bibliotecas. En la Figura 3.1 se puede apreciar una de estas aplicaciones funcionando.

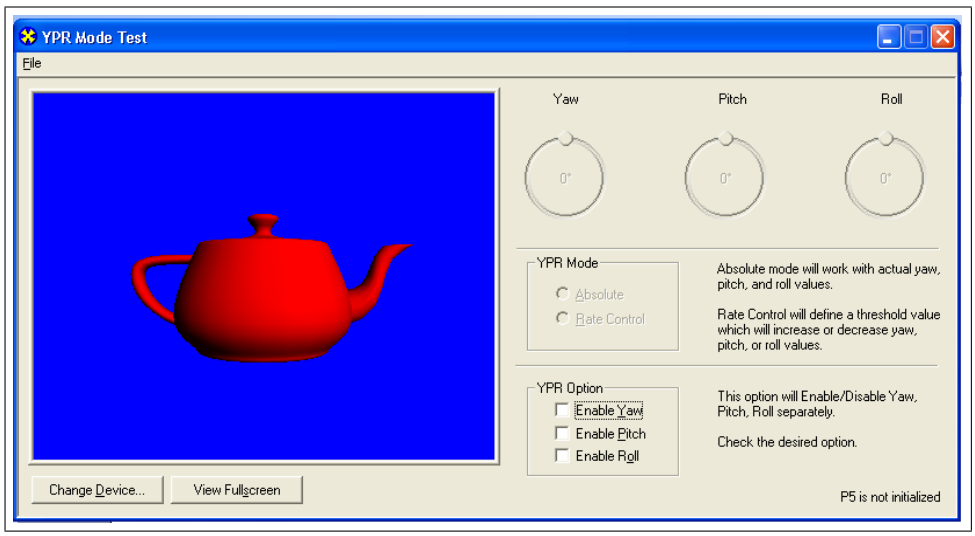

Figura 3.1: YPR mode test. Una de las aplicaciones funcionales del SDK oficial.

Una de las desventajas principales que posee este SDK, es que las posiciones del guante las detecta de modo relativo. A continuación se proporciona información detallada del modo absoluto y el modo relativo.

Modo Relativo.- Significa que no se conoce la posición física del guante en el espacio. Sólo se conoce a grosso modo dónde está de forma relativa al punto donde se encontraba anteriormente. Es decir, cuando se ejecutaba un juego con el guante la posición inicial del guante se reconocía como ($\mathcal{X}$=0, $\mathcal{Y}$=0, $\mathcal{Z}$=0) donde quiera que se encontrara dicho guante. Cualquier movimiento a partir de ese momento modificaría las coordenadas con base en dicha posición. Ésta posición podría ser alterada incorrectamente si movemos al guante lejos de la torre de recepción infrarrojo. Otra desventaja de este modo es que sólo rastrea un led a la vez, por lo que dicho led dominaría la dirección de las rotaciones dependiendo de si este se encuentra al frente o atrás.

Modo Absoluto.- Significa que se conoce la posición física del guante en el mundo real con respecto a la torre de recepción infrarrojo. Por lo tanto, se puede decir con seguridad si el guante se encuentra a 10 cm. a la izquierda de la luz roja del receptor y a 40 cm. enfrente de él. Si se desea utilizar el modo absoluto, se requiere escribir los programas usando controlador de Kenner [20].

El modo de Mouse que viene integrado en el hardware del guante, siempre utiliza el modo relativo y no se puede cambiar debido a la misma razón.

En el caso de las rotaciones, estas son siempre absolutas, sin importar el modo, se puede saber el ángulo del guante en el mundo verdadero.

Una vez explicado lo anterior se descartó el uso de este SDK, pues el modo relativo limita las posibilidades de la interfaz que se desarrolla en el presente trabajo de investigación además de que se limita a la plataforma Windows y utiliza software propietario de Microsoft.

3.2.2. Cortona SDK

VRML (*Virtual Reality Modeling Language*), es un lenguaje basado en texto genérico diseñado para construir mundos virtuales animados [10] . Fue creado por Mark Pesce y Anthony Parisi con la colaboración de Galvin Bell y Brian Behlendorf, con el propósito de diseñar una interfaz tridimensional para Internet. Al igual que HTML(*Hypertext Mark up Language*), los archivos de VRML son archivos ASCII compuestos de nodos que definen formas y atributos de modo que un navegador(browser) pueda presentar. Existen navegadores sólo para VRML, ó plug-ins (programas auxiliares para VRML que se añaden al navegador de Internet de su elección) para navegar en un mundo virtual en tres dimensiones. De este modo se pueden explorar simulaciones que van desde figuras geométricas como un cubo hasta edificios o ciudades enteras.

El estándar más actual de VRML es *VRML97* [11], el cuál dentro de su especificación no menciona la interacción con dispositivos de realidad virtual como guantes de datos, cascos, etc. Sin embargo, permite la inserción de código en Javascript para interactuar con los elementos nativos de VRML.

Uno de los plug-in más populares para el estándar VRML dentro de la plataforma Windows se llama Cortona y pertenece a la compañía ParallelGraphics.

Por lo tanto, una forma de añadir soporte para el guante P5 es escribir una interfaz para un plug-in de VRML tal como Cortona, en el lenguaje de programación Javascript.

Cortona VRML se instala como un plug-in para el navegador de Internet de su preferencia (Internet Explorer, Netscape y Mozilla) y aplicaciones de oficina como Microsoft Office. También provee de nodos y capacidades adicionales para el estándar de VRML 97. En la Figura 3.2 se puede observar un archivo abierto desde el cliente VRML de Cortona.

Cortona SDK es una interfaz para programar aplicaciones que integra la tecnología 3D de la compañía ParallelGraphics [21] en otros ambientes de desarrollo usando Visual C++, Visual Basic, Delphi y páginas Web.

Además cuenta con tutoriales para varios lenguajes de programación como Visual Basic y Delphi. Las plataformas para las que trabaja son Windows PC y Pocket PC respectivamente.

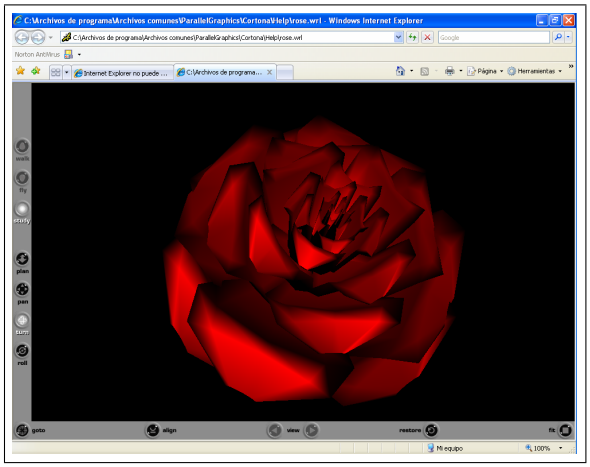

Figura 3.2: Ejemplo de un archivo de VRML visto desde Cortona VRML client.

Este SDK esta integrado de los siguientes elementos:

- Cortona Control.- Control ActiveX que permite a una aplicación desplegar e interactuar con escenas de VRML. Las aplicaciones que se desarrollan con Cortona SDK pueden ser creadas en cualquier ambiente que soporte ActiveX y OLE como Visual Studio, Delphi y Microsof Office.
- VRML Automation Interfaces.- Describe un conjunto de interfaces para que un ambiente externo tenga acceso y manipule varios objetos en una escena VRML cargada con el Cortona Control.
- VRMLSaver. Es un objeto COM (Component Object Model) que provee los medios para guardar escenas VRML o conjuntos de nodos a archivos VRML.
- Native Scripts.- Es una extensión del nodo Script de VRML.

VRML no especifica en ninguno de sus nodos como trabajar con el guante P5 o con cualquier otro dispositivo. Por lo que se tendría que desarrollar una interfaz entre el P5 y su biblioteca CP5DLL.dll y el ambiente virtual en VRML.

Las desventajas de este SDK son que trabaja para software de Microsoft y la plataforma Windows exclusivamente, su código es propietario y tiene un costo de 749 dólares. Por otra parte, no hay documentación de proyectos que usen éste SDK más allá de la información que proporciona el proveedor. Por lo tanto se descarta.

3.2.3. FreeWRL

Freewrl es un navegador de código abierto para VRML y X3D escrito principalmente para las plataformas Linux y OS X. para arquitecturas de 32 y 64 bits. Éste navegador puede correr por si mismo o dentro de un navegador web como Mozilla ó Netscape.

EAI (External Authoring Interface) [22] es una interfaz externa que permite controlar el contenido de un mundo virtual desarrollado en VRML dentro de una página web, por medio de un *applet* (miniaplicación en lenguaje de programación Java). Esto se hace a través de un *plug-in* (programa que extiende las capacidades de un navegador) que permite a objetos empaquetados, comunicarse entre sí.

Cabe mencionar, que EAI es independiente del Nodo Script de la especificación de VRML97. EAI es un anexo informativo a la especificación de VRML. Esto significa que no todos los clientes de VRML lo incluyen.

Freewrl puede ser accesado vía EAI para permitir el control del contenido visual de una escena en VRML si se programa en Java un controlador para el guante de datos P5.

Para poder entender cómo funciona Freewrl y el EAI, se requiere tener una distribución de Linux y bajar el instalable de http://freewrl.sourceforge.net/. Dependiendo de la distribución que se tenga se deben seguir ciertas instrucciones, pues se pueden tener problemas como en el caso de Fedora, Suse y Ubuntu para los cuales existen instalables del tipo RPM (Red Hat Package Manager).

La instalación que se realizó fue a partir del archivo RPM compatible con Fedora. Sin embargo, éste archivo no instala los ejemplos de EAI para empezar a practicar, por lo que se deben descomprimir dichos ejemplo del archivo tar.gz que se puede encontrar en el mismo sitio. Una vez bajado este archivo se descarga al sistema. Los ejemplos se pueden encontrar en la siguiente ruta: /usr/freewrl-1.17.1/tests/.

Dentro de los ejemplos más sobresalientes se probaron EAIAddRemove.java y EAIAddRoute.java. Sin embargo, sólo funcionó el primer ejemplo, el cuál presenta una escena de VRML con figuras geométricas, las cuales van desapareciendo por medio del código escrito en Java.

En la Figura 3.3 se puede observar una imagen del primer ejemplo funcionando.

El segundo ejemplo no funcionó porque requiere de la herramienta llamada *Appletviewer* que viene incluido en Sun Java SDK como se puede corroborar en la Figura 3.4.

Fedora, tal como muchas versiones de Linux proveen en cambio GNU Java, el cuál es una versión mucho más reducida del SDK de Sun Microsystems. Por lo tanto, para poder probar todos los ejemplos de EAI de Freewrl se tiene que desactivar GNU Java, descargar Sun Java SDK e instalarlo. Por obvias razones se requiere un conocimiento de Linux y su administración para poder utilizar Freewrl.

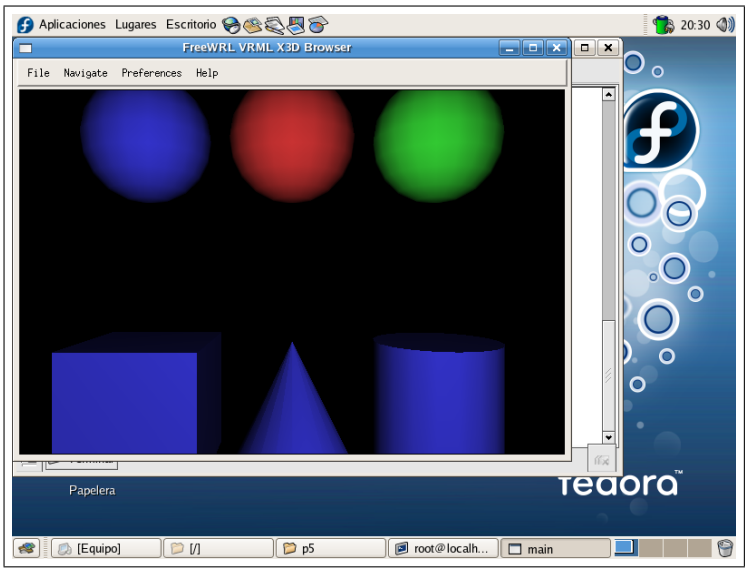

Figura 3.3: AddRemove. Uno de los ejemplos funcionando del directorio tests.

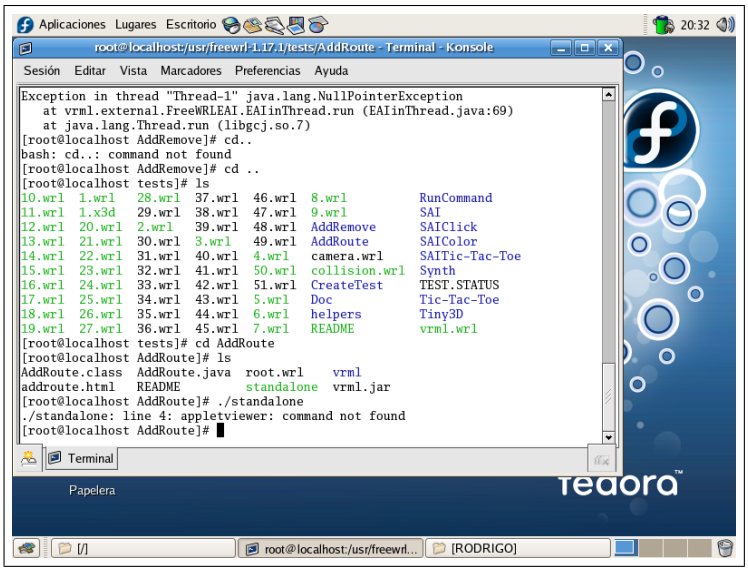

Figura 3.4: AddRoute. No funcionó porque no se encontró la aplicación Appletviewer.

Otro problema al tratar de usar esta herramienta es que el controlador del guante P5 para Linux, esta diseñado para programar con GNU C y no Java, por lo que se requiere que se capturen los datos vía GNU C, y se use código en Java para convertir esos datos en órdenes para el navegador Freewrl. Por ello se deduce inviable utilizar Freewrl para la interfaz que se desarrolla en la presente tesis.

3.2.4. Java/Java 3D

Java es un lenguaje reconocido por las siguientes características: simple, orientado a objetos, distribuido, robusto, seguro, de arquitectura neutra, portable, interpretado, de alto rendimiento, multitarea y dinámico.

Java así como Java 3D [24] son aplicaciones propietarias de Sun Microsystems que no requieren el pago de licencias para su uso.

Java 3D es una extensión de Java para poder programar mundos en tercera dimensión que pueden ser de diversos tamaños y niveles de complejidad, desde lo atómico hasta lo astronómico. Además, posee herramientas para crear y manipular objetos y estructuras en 3D así como importar escenas de otros formatos como VRML, 3D-Studio, AutoCad, Solid Works, etc.

Una de las ventajas de programar usando Java 3D es que es compatible con todos los desarrollos de Java, por ejemplo, se puede utilizar la tecnología de los applets para poder visualizar los mundos virtuales en Internet, así como las clases matemáticas, vectores, redes y comunicación, entre otras.

El controlador de Kenner permite programar aplicaciones para Java, por lo que no se requiere ningún otro programa para comunicarse con el guante; y a diferencia de los demás lenguajes de programación (Visual Basic, Delphi, C, etc.) hay un capitulo en línea sobre como programar el guante usando esta herramienta escrita por Davison [23].

Para corroborar el funcionamiento del controlador de Kenner con Java, se descargó el código fuente de Davison. Este código contiene aplicaciones para leer datos del guante. A continuación se sustituyó el controlador original del guante por el de Kenner en el directorio de \Windows\system32.

Se debe compilar el archivo *CP5DLL.java*, empaquetarlo y copiarlo al directorio de Java para poder hacer uso de sus diversas funciones. Se probó *ShowGlove.java*, un formulario grafico, el cuál despliega los datos que genera el guante en tiempo real, como la posición de los tres ejes (X,Y,Z), el doblez de los dedos, la orientación de la mano (Pitch, Yaw y Roll), etc.

También se probó MotionTest.java, una aplicación en modo texto que despliega caracteres de acuerdo a qué tan cerca esta el guante de la torre de recepción, según la orientación y si el puño esta cerrado o no. Las pruebas realizadas fueron exitosas. La parte

gráfica se llevará acabo con Java 3D. Para ello se debe descargar el programa del sitio oficial de Sun . Una vez instalado, se probó modelando un cubo de colores. Dicha prueba también fue exitosa.

Dado que la parte medular de la interfaz, que es la recepción de datos del guante, así como la parte gráfica pudieron llevarse acabo se decidió utilizar Java/Java 3D con el controlador de Kenner para realizar la interfaz.

En la Figura 3.5 se puede observar una fotografía del programa *Show Glove*, así como los valores que reporta, mientras que en la Figura 3.6 se observa el programa Motion Test, el cuál funciona en modo consola.

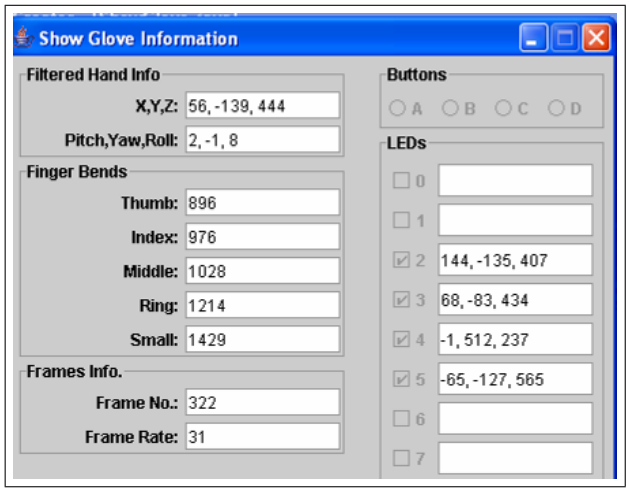

Figura 3.5: Show glove. Una de las aplicaciones de ejemplo de Davison.

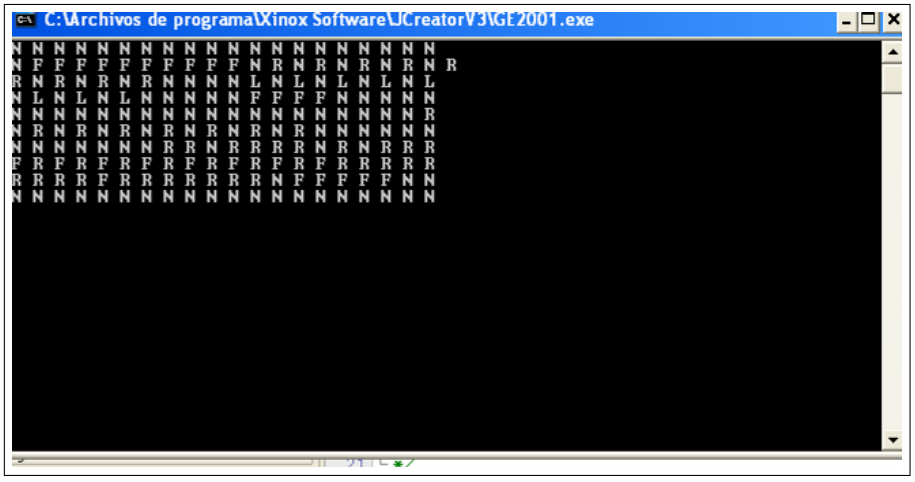

Figura 3.6: Motion test. Otra aplicación de Davison en modo consola.

3.3. Selección de software para desarrollar la interfaz

Una vez revisados los diferentes entornos de programación del guante P5, se concluye que la mejor opción para desarrollar la interfaz de este trabajo de tesis es Java/Java 3D, pues incluye todas las características necesarias como documentación, además de que es un lenguaje muy reconocido por la mayoría de los desarrolladores. Para la parte del modelado en tercera dimensión, su extensión conocida como Java 3D ofrece las bibliotecas necesarias para trabajar de manera similar a Freewrl. Otro factor importante es que existe documentación sobre el uso de ésta herramienta tal como el trabajo de Davison sobre el controlador dual de Kenner para el desarrollo de juegos de video.

Hablando de costos, Java es el ambiente ideal pues no se tiene que pagar para utilizar éste ambiente de programación.

Finalmente, se hace hincapié en la portabilidad de Java, pues uno de los propósitos principales de este trabajo de tesis es que la interfaz pueda ser ejecutada en diferentes equipos y gracias a las bibliotecas de *applets*, es posible implementar la interfaz del proyecto en una página web con todos los beneficios que involucra para los usuarios.

Capítulo 4

Herramientas de desarrollo

Una vez que se ha seleccionado una plataforma de desarrollo para programar la interfaz del presente trabajo de tesis, en este capítulo se describen las características de las herramientas con las que se va a trabajar tales como: el guante de datos P5 [12], el controlador dual [20], la clase FPSGLOVE [23], el lenguaje de programación Java y su extensión para el desarrollo de ambientes virtuales Java 3D [24].

4.1. Guante P5

El guante P5 es un dispositivo de realidad virtual para computadoras personales, económico e ideal para manipular ambientes en tres dimensiones y en general cualquier ambiente que pueda ser controlado con la naturalidad con la que se usa la mano en el mundo real para manipular cualquier objeto o herramienta. En la Figura 4.1 se puede observar una fotografía del guante y la torre de recepción en acción. Para utilizar el guante, éste se conecta a la torre receptora. Después se conecta el cable *USB(Universal Serial Bus)* de la torre receptora a la computadora. El modo de operación del guante es muy sencillo, el usuario mueve su mano enfrente de la torre de recepción de rayos infrarrojos, la cuál sensa una serie de leds, visibles o no, que se encuentran en la palma del guante y convierte esas señales en coordenadas para los ejes X, Y, y Z; así como la orientación en grados para dichos ejes. Además, el guante también posee sensores de doblez en los dedos y cuatro botones en la parte superior de la palma.

Una de las ventajas de éste innovador dispositivo, es que muchas operaciones pueden llevarse acabo al mismo tiempo como rotar la mano y doblar uno o más dedos, así como mover la mano a la izquierda. Más aún, se le puede dar un significado a los gestos de la mano, por ejemplo cerrar el puño.

En términos de compatibilidad, el guante fue diseñado para trabajar en la plataforma Windows; sin embargo puede ser utilizado en otras plataformas con Mac OS ó Linux.

Algunas de las ventajas que presenta este innovador dispositivo son:

Figura 4.1: El guante P5 y la torre en acción.

- Es un dispositivo muy económico en comparación con otros guantes de datos.
- Utiliza un método de rastreo muy eficiente vía infrarrojos por lo que es factible de usar en lugares muy congestionados sin ser afectado a diferencia de los dispositivos ultrasónicos.
- Es fácil de instalar, gracias a que utiliza el puerto USB de la computadora.
- Es compatible con sistemas operativos Windows, Linux y Mac OS.
- No requiere alimentación eléctrica extra en la torre de recepción o el mismo guante.

Es importante mencionar que no existe una interfaz general para controlar al dispositivo en un ambiente virtual, ya que la comercialización de éste se enfocó a la industria de los videojuegos. Tal es el caso que aún las versiones de hoy en día del P5 incluyen tres juegos, un controlador para Windows, pero ningún ambiente de desarrollo para programar el guante.

Essential Reality (la compañía propietaria del P5) liberó un Kit de desarrollo para el guante para Microsoft Visual C++ 6.0 sin costo alguno, para así motivar el desarrollo del mismo sin obtener una respuesta favorable, al menos en el ámbito de los videojuegos, que pudiera motivar la venta del mismo. La Figura 4.2 presenta una fotografía del guante en su empaque original.

Figura 4.2: El guante P5 en su empaque de fábrica.

Otros programadores y aficionados han desarrollado bibliotecas para programar el guante en varios lenguajes además de Visual C++ e incluso para diferentes plataformas superando el software oficial de Essential Reality. Gracias a ellos el guante se ha convertido en el dispositivo de realidad virtual favorito de los estudiantes y programadores de realidad virtual, más que los propios jugadores los cuales han migrado a consolas especializadas de Nintendo, Sony y Microsoft. En la Figura 4.3 se puede observar el guante, la torre de recepción y los juegos que vienen incluidos.

Desafortunadamente, sólo existen aplicaciones propietarias que hacen uso de estas bibliotecas y proveen escasa o nula documentación sobre dichas bibliotecas. De ahí que el usuario interesado en programar el P5 tenga que comenzar desde cero, sin la posibilidad de reutilizar el código de las aplicaciones existentes por falta de conocimiento y de fuentes de información. Existen en Internet algunas fuentes de información acerca del guante, tales como el grupo de discusión del guante P5 en Yahoo [25].

Figura 4.3: El guante P5 y los discos que vienen incluidos.

Otra de las características relevantes del guante P5, es que está provisto de dos interfaces físicas al mismo tiempo, las cuales son:

1. Comportarse como un ratón USB

2. Funcionar como guante virtual.

No obstante, la mayoría de los programadores de ambientes virtuales apagan la función de ratón USB, para que esta no interfiera en sus aplicaciones.

En la Figura 4.4 se pueden apreciar los componentes superiores del guante, tales como: los 3 botones programables, el botón de encendido/apagado y los leds (diodos emisores de luz) transmisores infrarrojos.

Si se usa el guante en modo relativo, los valores de los ejes X, Y, Z son coordenadas relativas a los ejes de la torre receptora y se considera el origen en el centro de la base de la torre. En el modo absoluto, los valores de los ejes mencionados anteriormente, son absolutos a la posición física del guante; así se puede saber con exactitud (en centímetros por ejemplo) que tan cerca ó lejos está el guante de la torre de recepción. En la Figura 4.5 se aprecia la torre de recepción respecto a los ejes X, Y, y Z.

En cuanto a los grados de rotación, los valores que reporta el guante son siempre absolutos. En la Figura 4.6 se puede apreciar el guante respecto a los tres ejes de rotación.

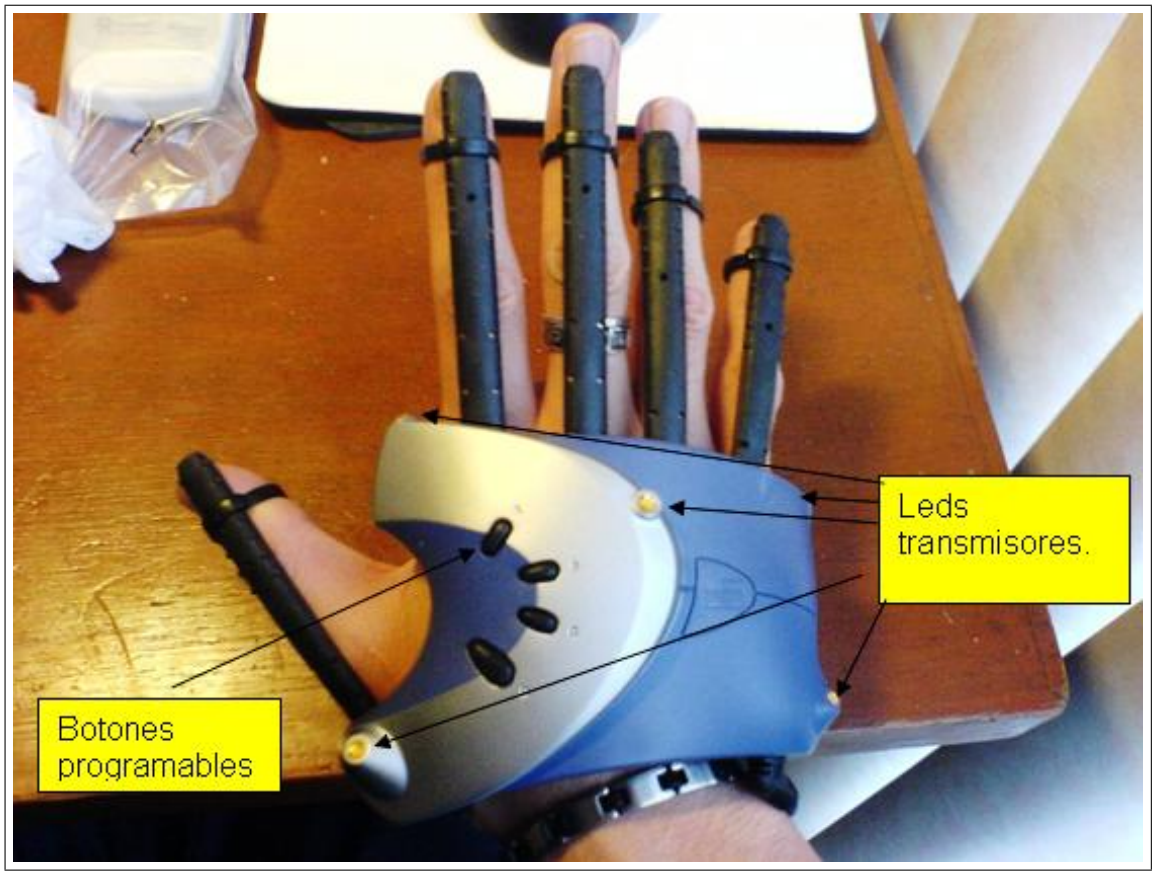

Figura 4.4: Componentes superiores del guante P5.

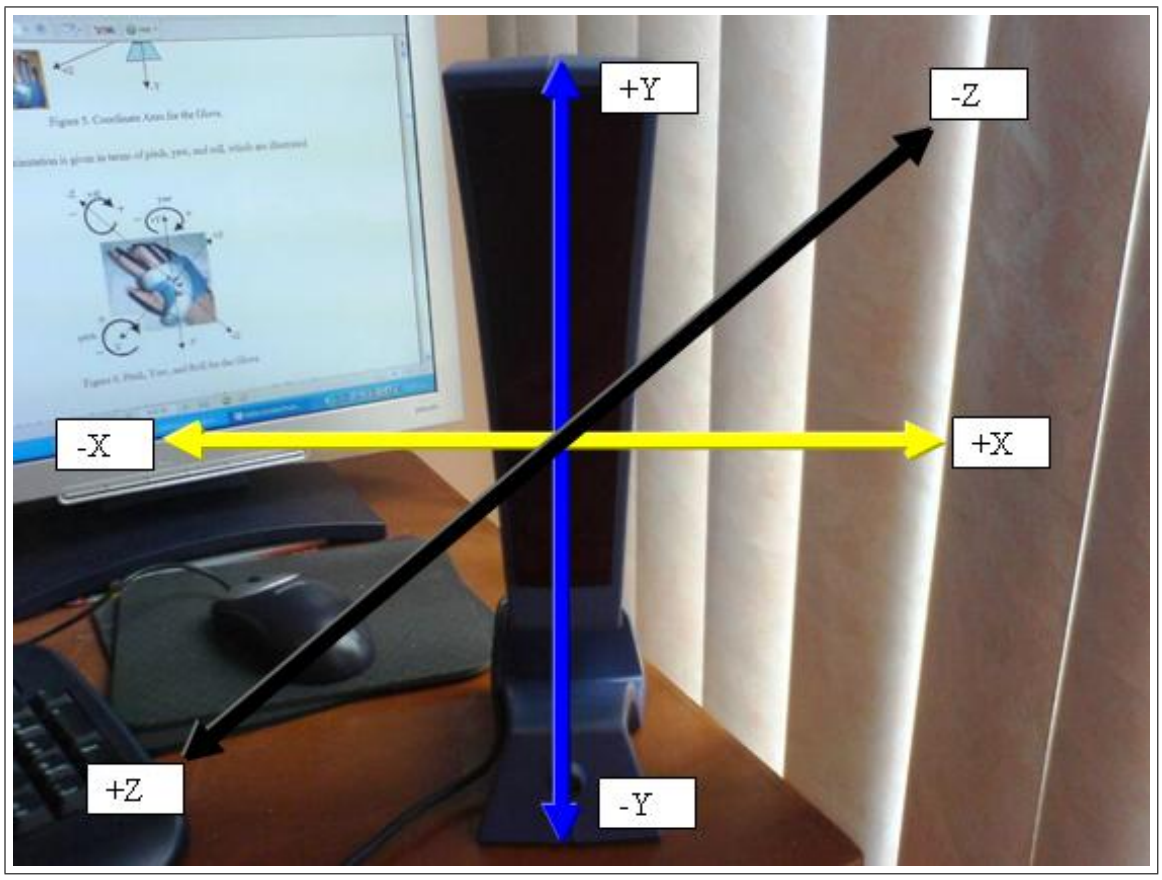

Figura 4.5: La torre de recepción y sus respectivos ejes.

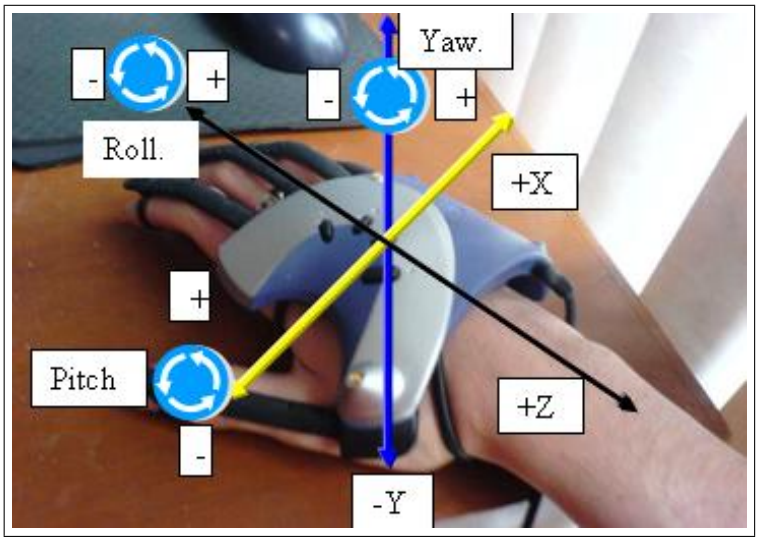

Figura 4.6: Rotación en el guante.

4.2. Controlador dual de Kenner

Este controlador constituye un SDK no oficial para escribir software para el guante P5, está basado en el código oficial de Essential Reality. Es un reemplazo de la biblioteca *P5DLL.DLL* requerido por la mayoría de las aplicaciones que usan el P5. No es un controlador para Windows, debido a que el P5 no requiere controladores adicionales a los del ratón que vienen incluidos en dicho sistema operativo; tampoco tiene efecto en el modo mouse del guante, ya que este viene grabado en el hardware y no se puede modificar. Éste controlador puede ser descargado desde el sitio del programador Kenner [20] de forma gratuita en su versión beta. Es llamado "dual", debido a que provee información acerca de la posición del guante tanto en modo absoluto como relativo. En la Figura 4.7 se puede apreciar una fotografía del sitio web de Kenner.

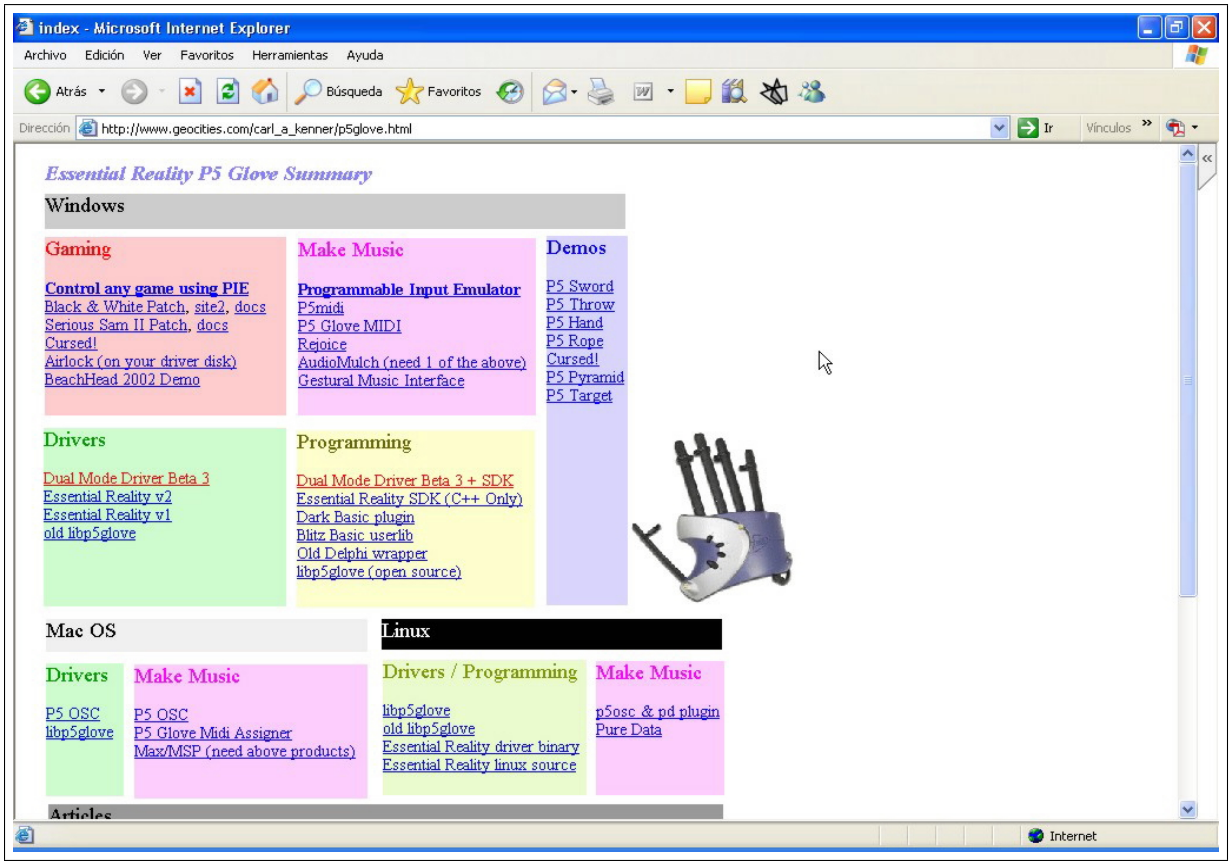

Figura 4.7: El sitio web de Kenner.

Comparado con el controlador oficial del P5, el controlador de Kenner ofrece mayor precisión, filtrado de datos provenientes del guante, mejor acceso a la información de los leds de la palma, etc.; dentro de los lenguajes de programación soportados para dicho controlador, se encuentran Delphi, Visual Basic, C, Visual C++ y Java.

No existe documentación por parte de Kenner acerca del uso del controlador con ninguno de los lenguajes mencionados. Sin embargo, Davison elaboró un capitulo en linea acerca del guante P5 con fines educativos y de promoción, y es la mejor fuente de información acerca del controlador de Kenner y su uso con Java [23].

Instalación del controlador dual de Kenner

Una vez descargado el controlador y descomprimido, los archivos necesarios para programar en Java son: *P5DLL.dll* y *CP5DLL.java*. La biblioteca *P5DLL.dll* es el controlador de Windows para el guante P5, mientras que *CP5DLL* provee las rutinas necesarias para comunicar algún lenguaje de programación (en este caso Java) con la biblioteca *P5DLL.dll*.

Aunque es recomendable copiar la biblioteca *P5DLL.dll* dentro del directorio donde se va a correr la aplicación, para este trabajo de tesis se copio en <Directorio de Windows>\system32, de modo que sea cargada cada vez que se inicia el sistema operativo. Se sugiere hacer una copia de seguridad de la biblioteca original que viene con el guante y que se encuentra en dichodirectorio.

En cuanto al archivo *CP5DLL.java*, se debe compilar y empaquetar primero antes de poder llamar a las diferentes rutinas del mismo. Es importante mencionar, que antes de programar el guante usando el Controlador Dual, se requiere tener instalado Java SDK en el sistema; dicho software se puede descargar gratuitamente de *http://www.java.com/es/download/*.

Para compilar se escribe:
javac –d . CP5DLL.java

Una vez compilado, se procede a empaquetarlo con la siguiente instrucción:
jar cvf CP5DLL .jar com.

De ahí, se procede a copiar el archivo en:
<Directorio de java>\jre\lib\ext y <directorio de la maquina virtual de java>\lib\ext.

Contenido del paquete CP5DLL

La clase *CP5DLL* consta de varias constantes, variables, métodos y de tres clases internas: *P5Data*, *P5Infor* y *P5State;* la clase *P5Data* solo se conserva con fines de compatibilidad con versiones anteriores del controlador. En la Figura 4.8 se presenta un diagrama jerárquico del paquete *CP5DLL*.

Los 40 métodos públicos de *CP5DLL* son usados para inicializar y configurar el o los guantes conectados a la computadora (recordemos que se puede conectar más de un guante). Por ejemplo, es posible ajustar el nivel de filtro de los datos que se reciben del guante. Por su parte, las 40 variables en *P5State* proveen información acerca de la

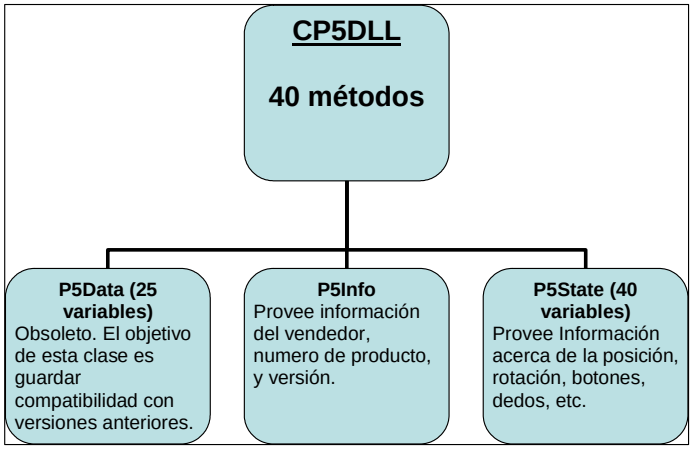

Figura 4.8: Estructura jerárquica de la clase CP5DLL.

posición del guante, rotación, estado de los botones, de los *LEDS (*diodos emisores de luz), etc. Existe un método llamado *update()* el cual se debe llamar para actualizar dichos valores.

Finalmente, las 25 variables públicas de la clase *P5Info* poseen información acerca de los detalles del guante, como el nombre del fabricante, el número del vendedor, numero de serie, etc.

4.3. La clase FPSGLOVE de Davison

FPSGLOVE (First-Person Shooter Glove Class) es una clase que utiliza las funciones de *CP5DLL* y las simplifica. Por ejemplo, cambia los arreglos de punto flotante que se reciben del guante por constantes e implementa algunos métodos de uso cotidiano como *isAPressed*(que nos indica si el botón *A* es presionado), etc. Dicha clase puede ser descargada del sitio web de Davison y es ideal para abstraer la forma en que se reciben los datos del guante P5. En la Figura 4.9 se presenta una fotografía de éste sitio.

Los apuntes de Davison proveen información descriptiva para realizar las tareas más comunes, como actualizar los datos del guante, detectar la posición en el eje Z, la rotación, finalizar el controlador, etc. En la Figura 4.10 se puede apreciar un diagrama jerárquico de dicha clase.

El objetivo de *FPSGLOVE* es acceder exclusivamente a ciertos datos requeridos por otra aplicación de Davison llamada *HandView3D*. *HandView3D* es una aplicación hecha en Java 3D la cuál presenta un escenario compuesto de un tablero de ajedrez y una vaca en el centro. El usuario al mover su mano mueve el punto de vista local alrededor de la vaca. Además hay sonido tridimensional colocado en el centro de la vaca el cuál se puede encender con el botón *A* del guante. Esta aplicación esta compuesta de varias

Figura 4.9: El sitio web de Davison.

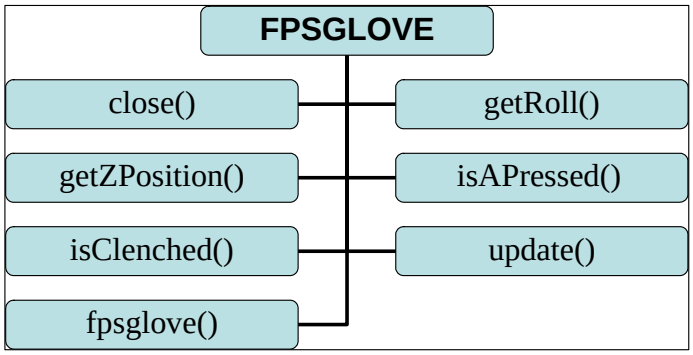

Figura 4.10: La clase FPSGLOVE de Davison.

clases, aparte de *FPSGLOVE()*, sin embargo para fines del presente trabajo de tesis no se requiere profundizar en ellas.

Cabe mencionar que no fue posible ejecutar la aplicación *HandView3D*, ya que se requiere descargar un componente adicional de sonido tridimensional llamado *Joal Sound Manager*. Dicho componente no se pudo instalar, y por lo tanto al momento de compilar se produjeron errores de referencia. En la Figura 4.11 se puede apreciar una fotografia de *HandView3D* tomada del libro de Davison [23].

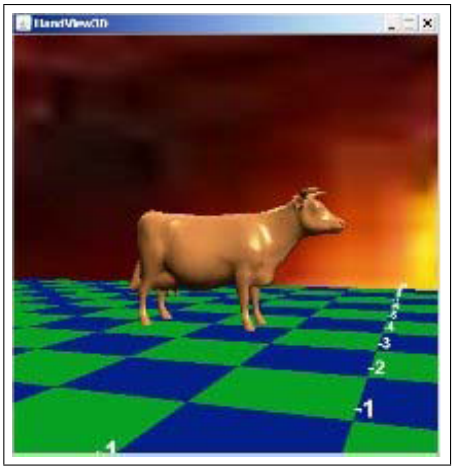

Figura 4.11: HandView3D por Davison.

4.3.1. Funcionamiento de la clase FPSGLOVE y sus métodos internos

Constructor FPSGLOVE

El constructor de la clase FPSGLOVE inicializa el enlace al guante y almacena el estado del mismo para su uso futuro.

```
//Se instancia el objeto gloves de la clase CP5DLL
private CP5DLL gloves;
//Se asigna el atributo P5State a gloveState
private CP5DLL.P5State gloveState;
 public FPSGlove()
  {
   gloves = new CP5DLL();
   //Se inicializa el guante
   if (!gloves.P5_Init()) {
   //si no se detecta el guante despliega un error
     System.out.println("P5 Initialization failed");
```

```
    System.exit(1);
   }
  gloves.P5_SetForwardZ(-1); // cambia Z positivo  hacia el usuario
  gloves.P5_SetMouseState(-1, false); // desactiva el modo mouse
  //Se asegura que solo exista un guante
  int numGloves = gloves.P5_GetCount();
  //Si encontramos mas de un guante enviamos el mensaje correspondiente.
  if (numGloves > 1) {
   System.out.println("Too many gloves detected: " + numGloves);
   System.exit(1);
   }
  // Se almacena una referencia al estado del guante
  gloveState = gloves.state[0];
 }
```

Actualizar y terminar el estado del guante

Los métodos *update()* y *close()* nos permiten actualizar el estado del guante y finalizar la referencia al mismo cuando se termina la aplicación.

```
public void update()
 { gloveState.update(); }
 public void close()
 {// Restaura el modo mouse del hw del guante.
  gloves.P5_RestoreMouse(-1);
  gloves.P5_Close();
 }
```

Obtener la posición del eje Z

El método *getZposition()* recibe la posición filtrada del guante en Z y devuelve una de tres constantes de acuerdo a que tan lejos se encuentra el guante de la torre receptora.

Las constantes devueltas son:
NEAR.- Cerca de la torre.
FAR.- Lejos de la torre.
MIDDLE.- En medio.

Estas constantes son el resultado de una medición de los valores arrojados por el guante, por medio de la aplicación *ShowGlove*, la cuál nos permite observar los datos que devuelve el guante en tiempo real.

A continuación se presenta el código con sus respectivas constantes.

```
// Constantes para definir la posicion del eje Z con respecto a la torre
public final static int NEAR = 0;
public final static int FAR = 1;
public final static int MIDDLE = 2;
 // Constantes de posicion.
private final static int NEAR_MIN = 500;
 //Que tan cerca de la torre se debe estar para que se cuente.
private final static int FAR_MIN = 900;
//Que tan lejos de la torre debe estar el guante para que se cuente.
public int getZPosition()
 /* Se convierten los valores recibidos en constants basado en NEAR_MIN
     y FAR_MIN.*/
  {
   float zPos = gloveState.filterPos[2];
 // posicion filtrada del eje Z
   if (zPos < NEAR_MIN) { // cerca de la torre
 // System.out.println("front (" + zPos + ")");
   return NEAR;
   }
   else if (zPos > FAR_MIN) { // lejos de la torre
 // System.out.println("back (" + zPos + ")");
    return FAR;
   }
   else
   return MIDDLE;
  }
```

Detectar si el botón A del guante ha sido presionado

El método *isAPressed()* nos devuelve verdadero si el botón *A* del guante ha sido presionado. Cabe destacar, que el oprimir el botón sólo puede ser grabado en la variable gloveState si la acción termina al momento en que se invoca al método *update()*. Esto no es un problema si se llama al método *update()* frecuentemente (por ejemplo, cada 100 milisegundos ó menos), debido a que la acción de oprimir un botón toma varias decenas de segundo para llevarse acabo.

A continuación se presenta el código con sus respectivos valores. En la Figura 4.12 se puede apreciar de cerca los 4 botones programables del guante P5.

```
public boolean isAPressed()
 // Devuelve verdadero si el primer boton del guante (A) es presionado.
 { boolean[] buttonPresses = gloveState.button;
  return buttonPresses[0];
 }
```

Figura 4.12: Botones programables del P5.

El último método de la clase *FPSGLOVE* se llama *isClenched()*. Este método permite detectar si hemos cerrado los dedos del guante en forma de puño. El método es muy sencillo. Se declara una constante llamada *FINGERS_BENT* que almacenará cuantos dedos deben ser doblados para que se dispare la acción de *isClenched()*. *BEND_MIN* almacena el valor mínimo para considerar que un dedo esta doblado (se debe recordar que los intervalos varían de dedo en dedo). Después de revisar el valor absoluto en cada dedo y si éste es menor que la constante *BEND_MIN*, se incrementa el contador bentCount en uno.

```
private final static int FINGERS_BENT = 3;
private final static int BEND_MIN = 500;
public boolean isClenched()
 /* Devuelve true o false si al menos FINGERS_BENT dedos
 son doblados con un valor menor del que guarda BEND_MIN.*/
 {
  short[] fingerBends = gloveState.fingerAbsolute;
  boolean isClenched = false;
  int bentCount = 0;
  for(int i=0; i < fingerBends.length; i++) {
   if (fingerBends[i] <= BEND_MIN)
    bentCount++;
  }
  return (bentCount >= FINGERS_BENT);
 }
```

4.4. Java 3D

Java 3D [26] es un conjunto de clases diseñadas por Sun Microsystems para proveer a su lenguaje nativo Java de la representación e interacción de gráficos en tres dimensiones.

El programador no necesita estar consciente de todos los sucesos involucrados en la creación de gráficos en tres dimensiones. Java 3D proporciona una serie de clases ordenadas jerárquicamente que definen las primitivas básicas tales como figuras geométricas, apariencias, luz, texturas, rotaciones, traslaciones, etc., así como la forma en que pueden interactuar entre sí. Es a través de estas primitivas, que el programador puede definir objetos que residen en el universo virtual, el cuál es renderizado para su correcta representación.

En la Figura 4.13 se presentan los símbolos utilizados en un diagrama de escena para Java 3D.

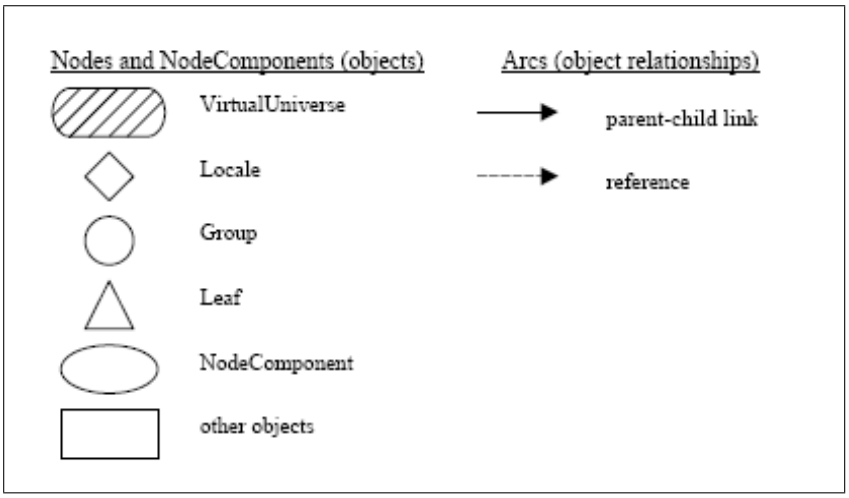

Figura 4.13: Símbolos utilizados en un diagrama Scene graph para Java 3D.

El diagrama de escena en Java 3D *scene graph*, es un conjunto de objetos en 3D estructurados de manera jerárquica que especifican y detallan el contenido de un *Universo Virtual* y su representación. Se utiliza como una ayuda al programador a la hora de diseñar sus clases, para poder desarrollar dichas clases de manera estandarizada y por lo tanto susceptible de ser modificada e interpretada por otros programadores.

Los nodos representan objetos derivados de las *clases primitivas* de Java, mientras que las flechas representan las relaciones entre dichos objetos.

Un nodo *group* puede tener muchos hijos, pero un solo padre. Un nodo *leaf* sólo puede tener un padre y ningún hijo. Si un nodo hijo posee más de un padre, el compilador pasara por alto el error, pero en tiempo de ejecución esté será detectado y no aparecerá imagen alguna. Esto es lo que se conoce como *scene graph ilegal*.

La relación *reference* asocia un objeto *NodeComponent* con un nodo del diagrama de la escena, por ejemplo asociar un nodo *Shape* (figura) con un nodo *Cube* (Cubo).

Se dice que la estructura de un diagrama de *Scene graph* es en forma de árbol, ya que a partir de un nodo raíz, este se relaciona con uniones padre-hijo con otros nodos creando las ramas del arbol. Estas relaciones no pueden llamarse a sí mismas. La primera rama del árbol siempre debe partir del nodo *Locale*.

Es importante mencionar, que sólo existe un camino entre la raíz del árbol y cada una de las hojas.

El diagrama de *scene graph* debe contener los objetos que se van a representar en el universo virtual, así como la localización, orientación, tamaño y apariencia de los objetos y el orden en que aparecen.

Los programas escritos en Java 3D pueden ser ejecutados como un paquete *JAR* (formato de archivo utilizado para reunir todos los componentes que requiere un programa de java en un único archivo), ó como *Applets* incrustados en una página Web.

Java 3D tiene predefinidos los objetos primitivos esfera, caja, cilindro y cono. Sin embargo, es posible construir cualquier figura geométrica que se desee a través de la especificación de las coordenadas de sus puntos así como de los vértices. En la Figura 4.14 se puede apreciar el diagrama de una escena sencilla.

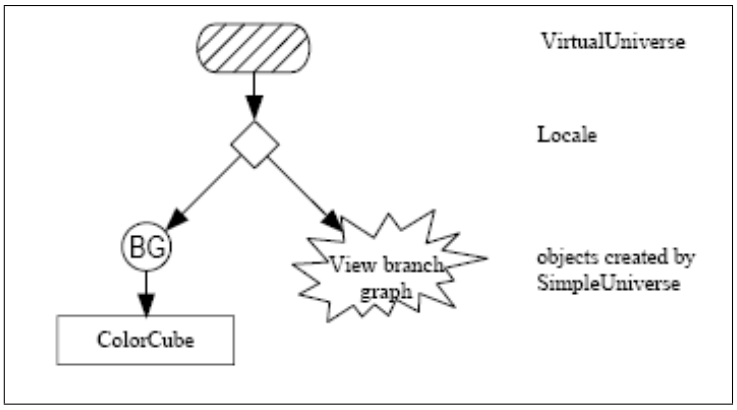

Figura 4.14: Ejemplo de un diagrama de escena sencillo.

Para modificar las figuras geométricas, es necesario utilizar un grupo de transformación *TransformGroup*. Un *TransformGroup* permite situar elementos en el espacio, así como aplicar rotaciones, traslaciones, expansiones, contracciones, etc. De manera más concreta, un *TransformGroup* es una matriz la cuál almacena las transformaciones que se le aplican a un elemento determinado; para poder inicializar un *TransformGroup* se le debe pasar como argumento una transformación 3D *Transform3D*. Un *Transform3D* es una matriz de doble precisión de tamaño 4x4 ordenados por filas. Esta matriz establece

el tipo de transformación que se aplica a un *TransformGroup* como puede ser rotación en *X rotX(ángulo)*; multiplicar dos matrices *mul(matriz 1, matriz 2)*, etc. Todo *TransformGroup* puede enlazarse con otro grupo para concatenar transformaciones, siempre y cuando el primero de ellos esté encadenado a un grupo de rama *BranchGroup*. Este a su vez se debe enlazar a un punto de referencia *Locale* que depende directamente del universo virtual.

4.4.1. Pasos para la creación de un programa en Java 3D

A continuación se presentan los pasos para escribir un programa en Java 3D de la forma más general posible.

1. Crear un objeto *Canvas3D*.

    a) Crear un objeto *SimpleUniverse* el cual referencia el objeto anterior *Canvas3D*

2. Personalizar el objeto *SimpleUniverse*.

3. Construir la rama de contenido (*Branch Group*).

4. Compilar la rama de contenido.

5. Insertar la rama de contenido dentro del punto de referencia *Locale* del objeto *SimpleUniverse*.

4.4.2. Programa sencillo en Java 3D

La Figura 4.15 presenta el código necesario para representar la cara de un cubo de colores en Java 3D, la Figura 4.16 presenta el diagrama de escena del programa y la Figura 4.17 una imagen del resultado obtenido.

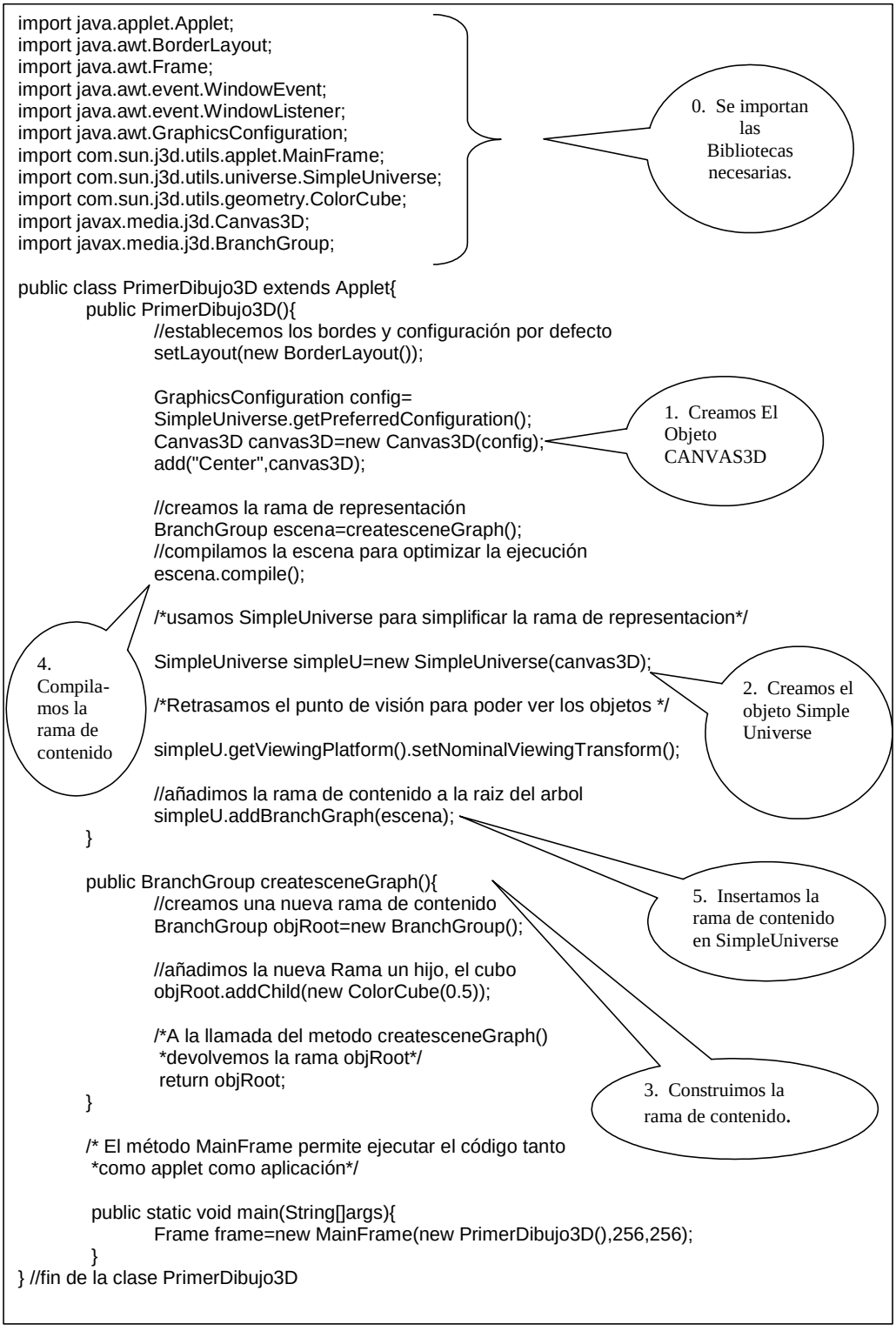

Figura 4.15: Código de un programa sencillo en Java 3D.

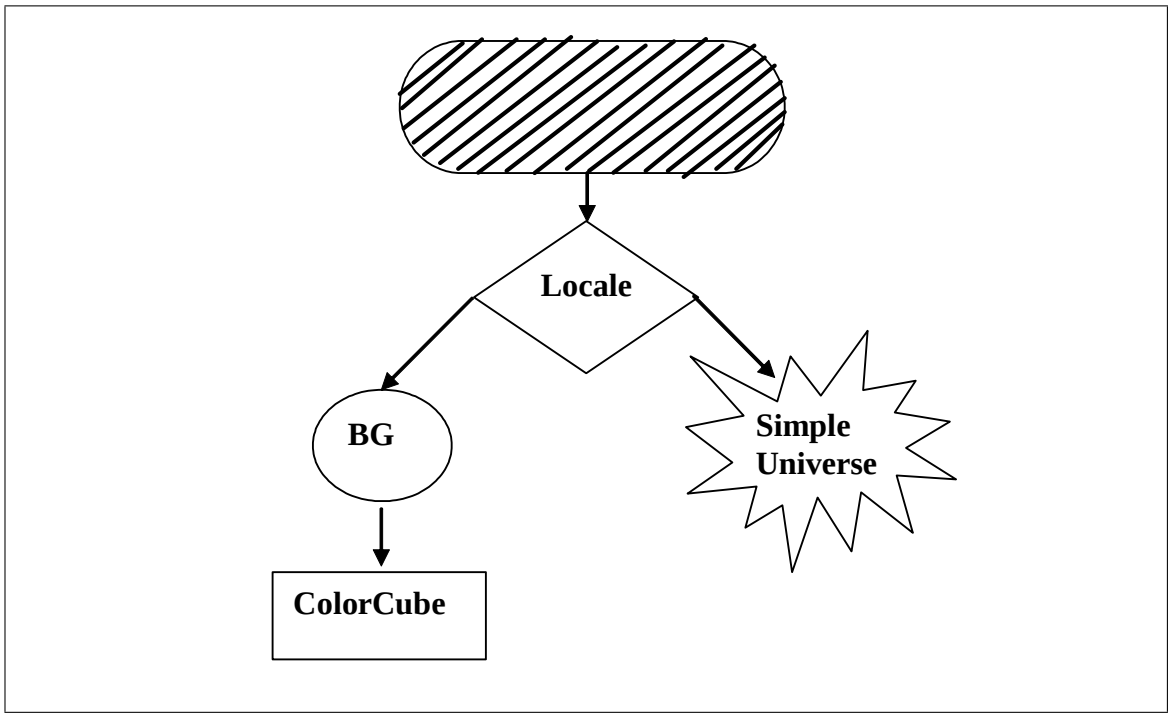

Figura 4.16: Representación del diagrama de escena de un programa sencillo.

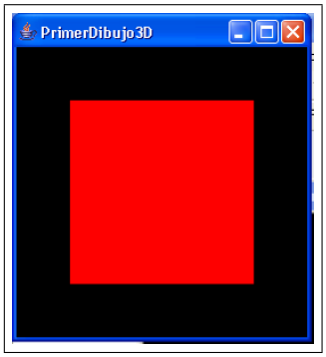

Figura 4.17: Captura de un programa sencillo en Java 3D.

Capítulo 5

Desarrollo

En este capítulo se presenta el desarrollo realizado en Java/Java3D para la implementación de la interfaz del guante de datos P5. Se comienza presentando las clases predefinidas para trabajar con el ratón y el teclado, la manera de desplegar texto, así como los pasos para programar clases de comportamiento de acuerdo a las necesidades del usuario. De ahí se establecen las características de la interfaz así como de los métodos que la componen. A continuación se presenta la programación de los módulos y en algunos casos alternativas de solución para problemas específicos como la concatenación de transformaciones en tiempo de ejecución. Una vez terminada la programación de la interfaz se proponen algunas mejoras a la clase de comportamiento y se presenta la aplicación final.

5.1. Java 3D y los dispositivos de entrada convencionales

Uno de los problemas de implementar un dispositivo externo a Java 3D es que no hay una clase genérica para manipular dispositivos diferentes del teclado y el ratón. Para dichos dispositivos, Java utiliza la clase *AWT(Abstract Windows Toolkit)* y a través de los métodos *KeyEvent* y *MouseEvent* es posible detectar si ha sido oprimida alguna tecla o botón respectivamente, y por consiguiente ejecutar alguna acción. Como ejemplo, se presenta el código para rotar un cubo de colores sobre el eje Y al presionar alguna tecla [27]:

```
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;
import java.awt.event.KeyEvent;
import java.util.Enumeration;
import javax.media.j3d.*;
import javax.media.j3d.WakeupOnAWTEvent;
public class Main {
 public Main() {
```

```
 SimpleUniverse universe = new SimpleUniverse();
 BranchGroup objRoot = new BranchGroup();
 TransformGroup objRotate = new TransformGroup();
 objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
 objRoot.addChild(objRotate);
 objRotate.addChild(new ColorCube(0.4));
 SimpleBehavior myRotationBehavior = new SimpleBehavior(objRotate);
 myRotationBehavior.setSchedulingBounds(new BoundingSphere());
 objRoot.addChild(myRotationBehavior);
 objRoot.compile();
 universe.getViewingPlatform().setNominalViewingTransform();
 universe.addBranchGraph(objRoot);
  }
  public class SimpleBehavior extends Behavior{
 private TransformGroup targetTG;
 private Transform3D rotation = new Transform3D();
 private double angle = 0.0;
// create SimpleBehavior - set TG object of change
 SimpleBehavior(TransformGroup targetTG){
 this.targetTG = targetTG;
 }
// Iniciamos el comportamiento
// Establecemos la condicion inicial para despertar al comportamiento
 public void initialize(){
 this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
 }
//Llamado por Java 3D cuando ocurre el estimulo apropiado.
 public void processStimulus(Enumeration criteria){
// Aqui va la accion que responde al estimulo.
angle += 0.1;
rotation.rotY(angle);
targetTG.setTransform(rotation);
this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
}
 }
 public static void main(String[] args) {
 new Main();
 }
```

}.

Como se puede observar en las líneas de código presentadas anteriormente, la clase *WakeupOnAWTEvent* perteneciente al paquete *javax.media.j3d* permite hacer uso de los dispositivos de entrada convencionales, como en este caso el teclado. A través del parámetro *KeyEvent.KEY_PRESSED*, Java 3D detecta si se oprime una tecla y si

esto ocurre, lleva a cabo las instrucciones que se encuentran localizadas en la función *processStimulus,* que de acuerdo a las líneas de código citadas, incrementa el ángulo de rotación del cubo de colores en 0.1 radianes sobre el eje Y.

Para empezar el desarrollo de la interfaz propuesta, se debe hacer notar a Java 3D de la existencia del guante, de la información que se recibe por ejemplo de la clase *FPSGLOVE* y de ahí ejecutar las acciones correspondientes en un modelo virtual de la mano que siga los movimientos del guante de manera natural.

El código anterior evidencia que no existen clases para trabajar con dispositivos de entrada no convencionales. Por lo tanto se debe empezar desde cero, creando un comportamiento específico para el Guante de datos P5 y descartando por lo tanto la clase *WakeupOnAWTEvent* de Java 3D. Tampoco existen ejemplos en el tutorial oficial de Sun de cómo implementar dispositivos de entrada no convencionales a Java 3D, por lo que se legitima el trabajo de la presente tesis, y se abren las puertas a trabajos futuros.

5.2. Modo ratón en Java/Java3D

Se ha mencionado en el Capítulo 3 del presente trabajo, que el guante P5 viene integrado con un modo Mouse en hardware del guante, y que dicho modo es relativo en su totalidad. Con base en esta premisa se deduce que cualquier programa en Java/Java 3D que esté desarrollado para trabajar con el ratón, puede ser también manipulado por el guante P5 haciendo uso de dicho modo. Como ejemplo, véase el siguiente código [27]:

```
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;
import javax.media.j3d.*;
public class main {
 public main() {
 SimpleUniverse universe = new SimpleUniverse();
 // Create the root of the branch graph
 BranchGroup objRoot = new BranchGroup();
 TransformGroup objRotate = new TransformGroup();
 objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
 objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
 objRoot.addChild(objRotate);
 objRotate.addChild(new ColorCube(0.4));
 MouseRotate myMouseRotate = new MouseRotate();
 myMouseRotate.setTransformGroup(objRotate);
 myMouseRotate.setSchedulingBounds(new BoundingSphere());
 objRoot.addChild(myMouseRotate);
 objRoot.compile();
 universe.getViewingPlatform().setNominalViewingTransform();
```

```
universe.addBranchGraph(objRoot);
}
public static void main(String[] args) {
new main();
}

  }
```

Este código permite rotar un cubo de colores con el ratón libremente, si se sostiene el botón izquierdo del ratón y se mueve sobre el eje X y/o Y. Si se usa el guante en lugar del ratón, se puede manipular el cubo del mismo modo, sin necesidad de programar una línea extra. Sin embargo no se pueden explotar las demás características propias del guante como el desplazamiento en el eje Z, sencillamente porque el ratón no esta diseñado para soportar tal operación; tampoco se puede programar los botones del guante, ya que dentro del modo ratón el botón izquierdo se representa doblando el dedo índice y el botón derecho se representa doblando el dedo anular. Mucho menos se pueden generar movimientos compuestos como cerrar los cinco dedos de la mano en forma de puño, etc. Esta es otra razón por la cuál se requiere programar una interfaz para explotar los beneficios del guante P5, ya que sólo de esta manera se puede pasar de un esquema de realidad virtual de escritorio a un esquema de realidad virtual inmersivo.

5.3. Texto en Java 3D

Para empezar a programar la interfaz de la presente tesis, es necesario asegurarse de que se reciben los datos del guante. Una forma sencilla de comprobar que esto sucede de una manera adecuada es desplegar en texto dicha información. Aunque desplegar texto en Java 2 es relativamente sencillo, se optará por desplegar el texto en Java 3D con el fin de empezar a trabajar directamente en el escenario virtual, así como con las clases y métodos que este lenguaje provee. A continuación se presenta el código para desplegar texto 2D en el escenario virtual de Java 3D [28] .

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.Frame;
import java.awt.event.*;
import java.awt.Font;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.Text2D;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
public class Text2D_I extends Applet {
 public Text2D_I() {
```

```
setLayout(new BorderLayout());
GraphicsConfiguration config =
SimpleUniverse.getPreferredConfiguration();
Canvas3D canvas3D = new Canvas3D(config);
canvas3D.setStereoEnable(false);
add(canvas3D);
BranchGroup scene = createSceneGraph();
SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
simpleU.getViewingPlatform().setNominalViewingTransform();
simpleU.addBranchGraph(scene);
}
public BranchGroup createSceneGraph() {
BranchGroup objRoot = new BranchGroup();
Text2D text2d = new Text2D("Text2D",new Color3f(0.0f, 1.0f, 1.0f),
"Helvetica", 32, Font.ITALIC);
objRoot.addChild(text2d);
return objRoot;
}
public static void main(String[] args) {
Frame frame = new MainFrame(new Text2D_I(), 256, 50);
}
```

} .

Ahora que se ha mostrado la forma de desplegar texto, se desplegará la información provista por la clase *FPSGLOVE* en términos de la posición del guante y su orientación con respecto a la torre receptora. Para ello se creará una nueva clase llamada *Text2D_H* haciendo uso de las siguientes funciones de la clase *FPSGLOVE*:

- *update()*, para actualizar los valores actuales del guante.
- *getZPosition()*, que devuelve un entero que va de 0 a 2, para indicar la cercanía del guante con respecto a la torre.
- *getroll()*, que devuelve un entero entre 3 y 5 para indicar la rotación del guante.

Además se usarán los nodos Text2D de Java 3D para desplegar dicha información en el universo virtual. La Figura 5.1 muestra el diagrama de flujo de dicha clase, mientras que la Figura 5.2 muestra la pantalla resultante. De esta manera se ha conseguido desplegar los valores que devuelve el guante P5 en el entorno de Java 3D.

5.4. Interacción con el usuario en Java 3D

Como se ha mencionado anteriormente, la interacción es una característica esencial de los ambientes virtuales. Java 3D provee de diferentes clases para manejar la interacción con el usuario; dichas clases se encuentran agrupadas dentro de la clase abstracta

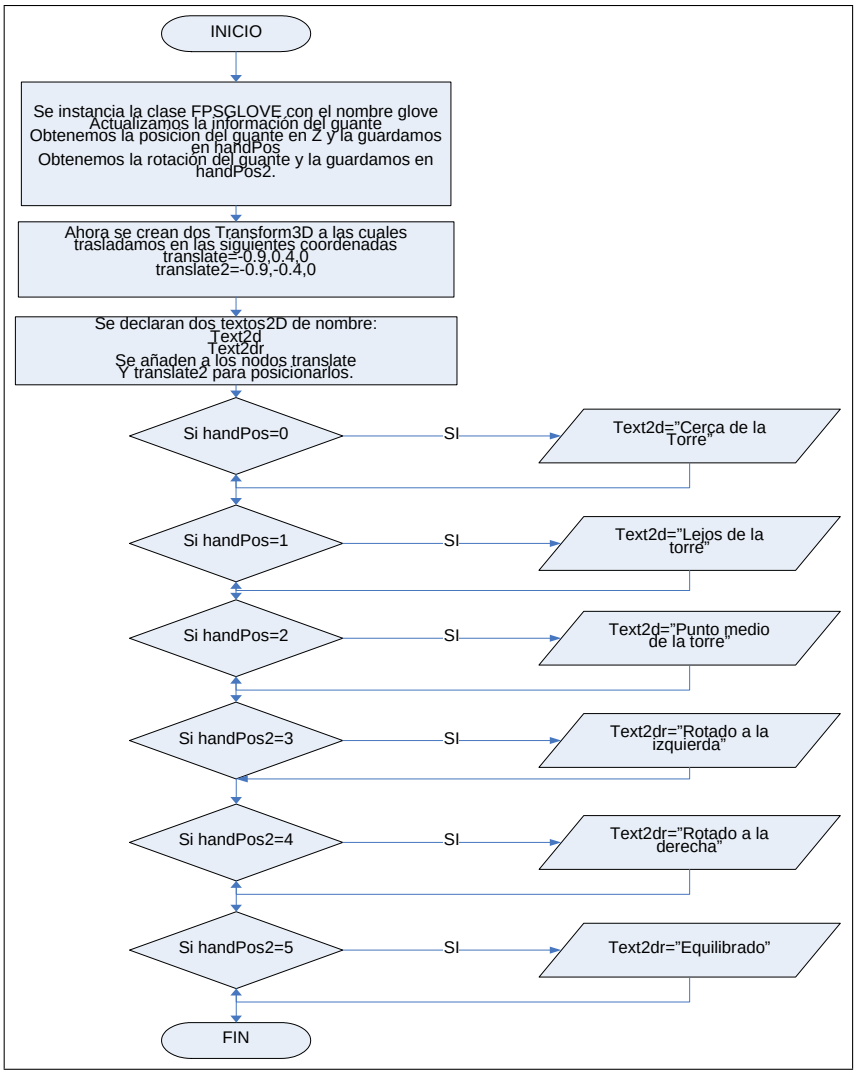

Figura 5.1: Diagrama de la clase Text2D_H.

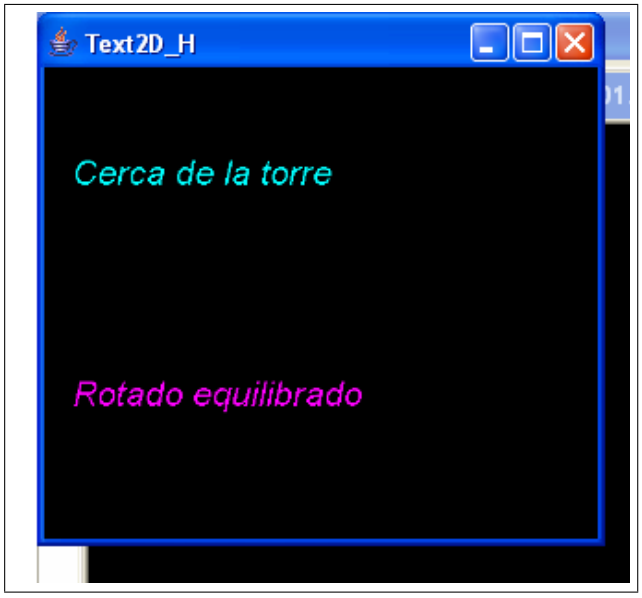

Figura 5.2: Corrida Text2D_H.

Behavior. La clase *Behavior* se divide en tres clases diferentes que agrupan las acciones de interacción más comunes y se llaman *Billboard*, *LOD* e *Interpolator*. En la Figura 5.3 se puede observar la estructura jerárquica de la clase *Behavior*. La clase *Billboard* provee de métodos y rutinas para controlar el cambio automático de orientación de los objetos en el universo virtual de acuerdo a la posición del observador. Por ejemplo cuando deseamos que un objeto éste siempre de frente al usuario, como el caso de un espectacular. A su vez, la clase *LOD* (Level Of Detail) controla el cambio automático de la representación de los objetos en cuanto a su distancia con el observador. Por ejemplo, no es necesario definir con gran precisión un objeto que se encuentra a 1 km de distancia del observador, pues se desperdician los recursos del equipo de cómputo. La clase *Interpolator*, controla la respuesta de los objetos de acuerdo al paso del tiempo. Esta clase es esencial para llevar acabo animaciones de cualquier tipo en un universo virtual. También existen las clases *MouseBehavior* y *KeyNavigationBehavior* las cuales controlan la interacción con el ratón y el teclado.

5.4.1. Clase Behavior convencional

A continuación se presenta el código para definir una clase de comportamiento convencional [28] , cuyo objetivo es desplazar un cubo en 0.02 metros al oprimir cualquier botón del teclado. La clase *NuevoBehavior* se encarga de definir el comportamiento al cual va a responder el escenario propuesto. La clase *PrimerBehavior* define el escenario donde se efectuará el comportamiento. Es importante mencionar, que la clase *NuevoBehavior* nunca es ejecutada, sino se instancia dentro de la clase *PrimerBehavior* con el nombre de *translationBehavior*. El objeto *translationBehavior* recibe a *objTranslate* que contiene al cubo y su respectiva matriz de transformación para aplicar los cambios

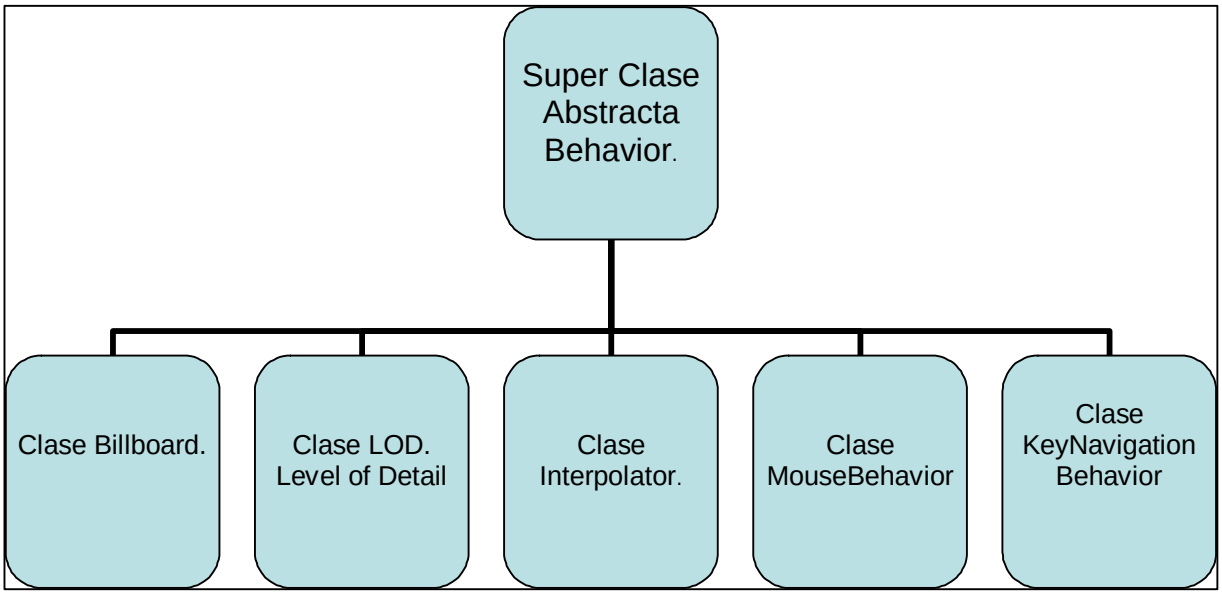

Figura 5.3: Clases de Interacción en Java 3D.

necesarios de acuerdo a la acción del usuario.

```
//NuevoBehavior.
public class NuevoBehavior extends Behavior {
 private TransformGroup modificarTG;
 private Transform3D translation=new Transform3D();
 private double mover=-1.0;
 NuevoBehavior (TransformGroup modificarTG){
 this.modificarTG=modificarTG;
 }
 public void initialize(){
  this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
 public void processStimulus(Enumeration criteria){
   mover+=0.02;
  translation.set(new Vector3f((float)mover,0.0f,0.0f));
  modificarTG.setTransform(translation);
  this.wakeupOn(new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED));
  }
}
//PrimerBehavior.
public class PrimerBehavior extends Applet {
 public PrimerBehavior() {
 setLayout(new BorderLayout());
```

```
 GraphicsConfiguration config =
 SimpleUniverse.getPreferredConfiguration();
 Canvas3D canvas3D = new Canvas3D(config);
 add("Center", canvas3D);
 BranchGroup scene = createSceneGraph();
 SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
 simpleU.getViewingPlatform().setNominalViewingTransform();
 simpleU.addBranchGraph(scene);
 }
 public BranchGroup createSceneGraph() {
  BranchGroup objRoot = new BranchGroup();
  Transform3D translation = new Transform3D();
  translation.set(new Vector3f(-1.0f,0.0f,0.0f));
  TransformGroup objTranslate = new TransformGroup(translation);
  objTranslate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
  objRoot.addChild(objTranslate);
  objTranslate.addChild(new ColorCube(0.3));
  NuevoBehavior translationBehavior = new NuevoBehavior(objTranslate);
  translationBehavior.setSchedulingBounds(new BoundingSphere());
  objRoot.addChild(translationBehavior);
  objRoot.compile();
  return objRoot;
 }
 public static void main(String[] args) {
 Frame frame = new MainFrame(new PrimerBehavior(), 200, 200);
 }
}
```

5.5. Diseño de una clase Behavior para interactuar con el P5 Glove

Para llevar a cabo una interfaz entre el guante P5 y un ambiente virtual se deben coordinar los datos que se reciben del guante con los objetos que representaremos en pantalla. Java 3D no proporciona métodos para establecer la comunicación con dispositivos no convencionales, por lo que se debe desarrollar una clase Behavior que controle las acciones del guante de datos con el universo virtual. Para ello se debe tomar en cuenta los requerimientos mínimos que solicita el entorno de Java3D para crear un clase de comportamiento (Behavior) a la medida. A continuación se enlistan dichos requerimientos:

1. Contener uno o más métodos constructores e incluir una condición.
2. Un método de inicialización llamado *initialize*.
3. Un método de respuesta al estímulo disparado por el usuario, *processStimulus*.
4. Una referencia al objeto sobre el cual se aplicarán las modificaciones (puede ser el mismo constructor de la clase).

La condición disparo para el método initialize, así como processStimulus se define por medio de la clase abstracta *WakeupCondition*. Una vez creada la clase *Behavior*, ésta se debe ligar a otra aplicación para proporcionar un campo de acción de acuerdo al comportamiento del usuario y la condición de disparo previamente definida. Por lo tanto, para trabajar con clases de comportamiento definidas por el programador se requiere un mínimo de dos clases: una para definir el comportamiento deseado y otra para definir la escena o mundo virtual con el objeto u objetos necesarios para llevar acabo la interacción. Es importante mencionar que el objeto que se establece para ser modificado por la clase *Behavior* debe coincidir con la el objeto propuesto en la clase de aplicación.

5.5.1. Definir un objeto WakeupCondition

Para poder escribir la clase *Behavior* se comenzará definiendo un objeto *WakeupCondition* que responda a un intervalo de tiempo particular; para ello se define una constante *Delay* con un valor de 75 (milisegundos). Este tiempo es suficiente para poder muestrear los movimientos del guante con una precisión aceptable [23], como oprimir un botón, doblar un dedo, girar la mano,etc.

Dentro del constructor de la clase *NuevoBehavior* se añade:

```
wakeUpCond=new WakeupOnElapsedTime(DELAY);
```

Ahora dentro del método processStimulus se añade:

```
if (glove.isAPressed()) //Si el boton A del guante es presionado
 mover+=0.02;
else
 mover+=0.0;
 //aplicamos la translacion
 translation.set(new Vector3f((float)mover,0.0f,0.0f));
 modificarTG.setTransform(translation);
 glove.update();//actualizamos el estado del guante
 this.wakeupOn(wakeUpCond);
```

Despues de añadir las lineas de código anterior al constructor de la clase *NuevoBehavior*, es posible desplazar la simulación de la mano, que por el momento está representada por un cubo de colores, al oprimir el botón A del guante P5.

Una vez que se ha logrado interactuar con un escenario en tres dimensiones, se comenzará a desarrollar la interfaz. Para ello se utilizará la clase *FPSGLOVE*, la cuál nos permite obtener datos simplificados del universo de información que provee el paquete *CP5DLL.jar* perteneciente al controlador dual de Kenner. Es importante mencionar que la clase *FPSGLOVE* fue diseñada para interactuar como un videojuego en primera persona (En inglés First-person shooter). Este tipo de videojuegos consiste en una simulación de un jugador que participa inmerso dentro de un escenario virtual, con el mismo punto de vista que se tendría si se estuviera presente en dicho escenario [29]. Por lo tanto, su uso para el presente trabajo de tesis es referencial, haciendo uso exclusivamente de las funciones que sean afines al objetivo de ésta tésis. A pesar de ésto se requiere programar más funciones y objetos para poder llevar acabo las metas planteadas en el Capítulo 1.

5.6. La Interfaz propuesta

Una vez que se han definido las clases para controlar el comportamiento de la mano virtual, se definen las acciones a las que responderá el guante. La interfaz propuesta, tiene como objetivo el aprendizaje de la realidad virtual; para ello se propone una simulación en tres dimensiones de una mano que siga los movimientos propios del guante y de los dedos de la mano de la manera más precisa posible. Dichos movimientos son:

- Traslación sobre los ejes X y Z.
- Rotación ajustable sobre los ejes Y ó Z.
- Doblar los cinco dedos a petición.
- Volver al origen, coordenadas (0,0,0) al presionar el botón A del guante.

Dentro de la clase *NuevoBehavior* que corresponde al comportamiento de un escenario virtual, se instancia la clase *FPSGLOVE* con un objeto de nombre *glove,* después de declarar el intervalo de sensado y la condición que invoca el sensado.

```
private static final int DELAY=75;//intervalo de sensar el guante en ms.
//condicion para invocar un nuevo sensado
private WakeupCondition wakeUpCond;
private FPSGlove glove;//Instanciar la clase FPSGLOVE
```

5.6.1. Creando un método para trasladar el cubo en el eje Z

El primer paso será la traslación de un cubo el cual simula la mano del usuario en primera instancia. Esa traslación se obtiene de la clase *FPSGLOVE* con la función *getZposition()*. Dicha función nos devuelve una constante entera entre 0 y 2 para determinar

la posición del guante respecto a la torre de recepción; dicha constante se obtiene de comparar el valor filtrado que devuelve la clase CP5DLL en el arreglo gloveState.filterPos en la posición 2 y que esta definido de la siguiente manera:

$Z < 500$ P5 Units indica que el guante esta *cerca* de la torre.
$Z > 900$ indica que el guante esta *lejos* de la torre.
$500 < Z < 900$ el guante se encuentra *en medio (MIDDLE)*. P5 units es una medida de referencia interna del P5 equivalente a 0.0488 centímetros [25].

En la Figura 5.4 se puede apreciar un diagrama de flujo de la traslación de un cubo en el eje *Z;* se puede observar que ahora la traslación no responde al oprimir un botón del guante, sino a una constante que se recibe del método *getZPosition()* de la clase *FPSGLOVE* sobre el eje Z.

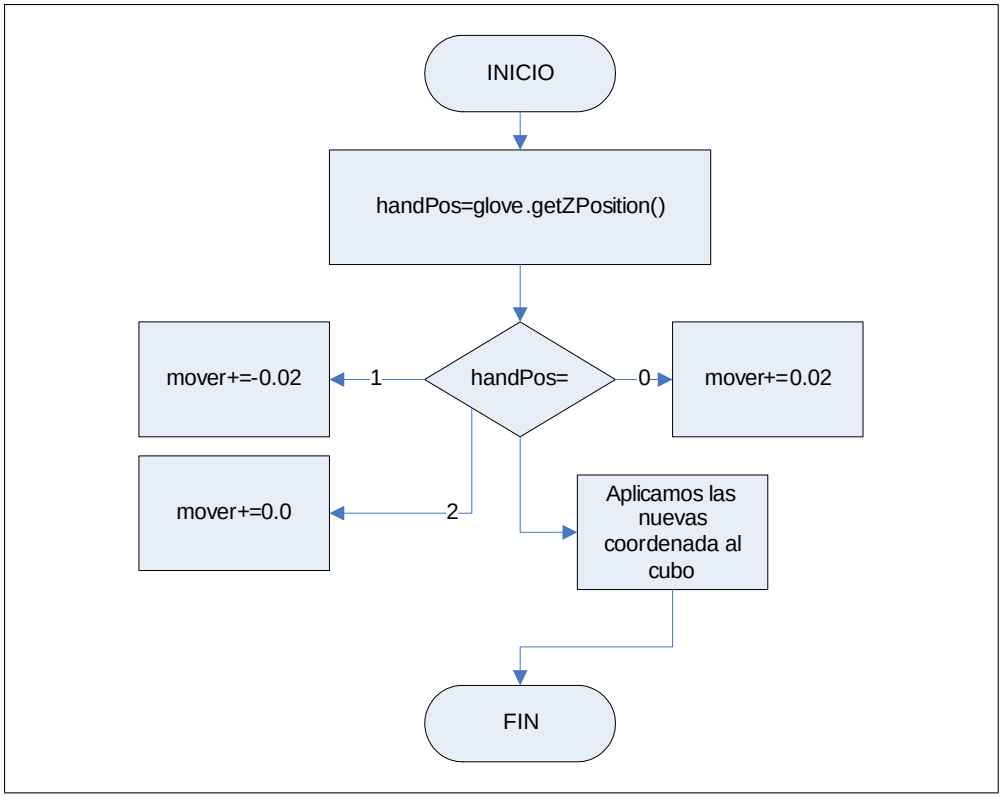

Figura 5.4: Diagrama de flujo de la traslación de un cubo sobre Z de acuerdo al movimiento del P5.

5.6.2. Creando un método para trasladar el cubo en el eje X

Como se ha mencionado anteriormente, la clase *FPSGLOVE* no posé ningún método para detectar el movimiento del guante P5 en el eje *X*. Sin embargo, en la interfaz

propuesta se ha decidido incluir dicha traslación, ya que se pretende que sea lo más natural posible. Por lo tanto, se tiene que agregar el código necesario para detectar la traslación en el eje X de los datos que se reciben de la clase *CP5DLL* y aplicar dichos cambios al escenario en 3D, específicamente al cubo que simula la mano.

Para implementar la traslación en X y la rotación más adelante se establece una notación para los archivos de prueba de la siguiente manera: Se tiene una clase para el escenario de nombre *PrimerBehavior.java* y una clase para el comportamiento *NuevoBehavior.java*. De ahora en adelante se cambiarán de *PrimerBehavior* a *t01escen.java*, y de *NuevoBehavior.java* a *t01imple.java*, con 01 como numero consecutivo conforme se vaya avanzando y con una longitud en el nombre de archivo de 8 caracteres.

La clase *CP5DLL* almacena la posición filtrada del eje X en el arreglo de tipo punto flotante llamado *gloveState.filterPos[0]*. Dicho valor está expresado al igual que el eje Y, y Z en "P5 units". Para cambiar dicho valor en forma de una constante siguiendo el ejemplo del método *getZPosition()*, primero se deben medir tres valores para señalar si el guante se encuentra hacia la izquierda, en el centro o a la derecha. Esos valores se pueden obtener usando la aplicación *Showglove* de Davison, la cual despliega en texto los valores que envía el P5 a la computadora en tiempo real.

Para el eje X se mide un valor de 80 para un movimiento a la derecha, y -80 para un movimiento a la izquierda. Por lo tanto se declaran las siguientes constantes:

LEFT, 6 para valores mayores o iguales a 80.

RIGHT, 7 para valores menores o iguales a -80.

CENTER, 8 para valores dentro del intervalo $-80<X<80$. Se usan 6, 7, 8 para dar continuidad a las constantes de rotación (5, 4, 3) y se implementan dentro de un nuevo *switch* en la clase *t01imple.java* para recibir dichas constantes. Siguiendo la pauta de *getZPosition()*, se logró que el cubo se traslade hacia la izquierda o hacia la derecha conforme se desplaza el guante. Se ha añadido el metodo *getXposition()* a la clase *FPSGLOVE*. Para poder recuperar el cubo en su posición de origen, se hace uso del método *glove.isAPressed()*, definiendo las coordenadas del cubo como 0,0,0.

5.6.3. Rotación en Y

El método *getRoll()* de la clase *FPSGLOVE*, devuelve una de tres constantes (*ROLL_LEFT*, *ROLL_RIGHT* y *LEVEL*) definidas por Davison para determinar si el guante ha sido rotado hacia la izquierda, derecha ó se encuentra equilibrado. Dichas constantes están basadas en los siguientes valores: para *ROLL_RIGHT* tenemos un valor de $Y > 80°$, para *ROLL_LEFT* $Y < -40°$ y en *LEVEL* un valor dentro del intervalo $-40° < Y < 80°$. El ángulo de rotación se obtiene de la clase *CP5DLL* , de la variable *gloveState.filterRoll*.

Para simplificar el código se usarán valores enteros de la rotación que son 3,4,5 para *ROLL_LEFT*, *ROLL_RIGHT* y *LEVEL* en la clase *t02imple*, respectivamente. Una

vez obtenidos los valores de clase *FPSGLOVE*, se deben aplicar los cambios al cubo que representa la mano. Para ello se aplica una rotación de 5° para un valor de 3, -5° para un valor de 4, y 0° para un valor de 5. Esto se hace declarando una variable de punto flotante de doble precisión (*double*) de la siguiente manera:

```
private static final double ROT_AMT=Math.PI/36
```

Se recuerda que la rotación se hace en radianes y π es igual a 180°, por lo tanto $\pi/36 = 5°$

Ahora se declara una función *turn()*:

```
private void turn()
 {
 int handOrient=glove.getRoll();
 switch (handOrient) {
 case FPSGLOVE.ROLL_LEFT: rotateY(ROT_AMT);break;//rotamos en 5 grados
 case FPSGLOVE.ROLL_RIGHT: rotateY(-ROT_AMT);break;//rotamos en -5 grados
 case FPSGLOVE.LEVEL: break;//no se rota
}
```

5.6.4. Aplicar los cambios a la matriz de transformación espacial

Una vez que se determinan los valores de translación y rotación en la clase *t02imple*, se deben aplicar al *TransformGroup* que almacena la transformación espacial de los objetos del escenario virtual definidos en *t02escen*. Esto se hace con las siguientes instrucciones:

```
translation.set(new Vector3f((float)moverx,0.0f,(float)mover));
modificarTG.setTransform(translation);
//aplica rotacion
t3d.mul(toRot);
modificarTG.setTransform(t3d);
glove.update();
```

En *translation* se almacenan las coordenadas del guante en X y Z. *modificarTG* es el *TransformGroup* que recibimos de *t02escen*. Con el método *setTransform* se aplica la transformación almacenada en *t3d*. Para aplicar la rotación se multiplica *t3d* por *toRot* y finalmente se aplica la transformación de la rotación a *modificarTG* y se actualiza el estado del guante.

Como se mencionó anteriormente, *t3d* es un *Transform3D*. Un *Transform3D* es un objeto de transformación generalizado, representado internamente como una matriz de 4x4, con un tipo de datos de punto flotante de doble precisión. Su representación matemática es de renglón mayor y se utiliza para realizar translaciones, rotaciones, y escalamientos.

En Java 3D es común que un objeto visual sea trasladado y rotado, rotado sobre dos ejes, trasladado sobre dos ejes, etc. [30]. En todos esos casos se aplican dos diferentes transformaciones a un objeto en particular. Dichas transformaciones pueden ser combinadas en una misma matriz y almacenadas en un sólo objeto *TransformGroup*.

El modelo de Java 3D para aplicar transformaciones, multiplica la matriz de transformaciónes ($m00..m33$) por las coordenadas de un objeto (x, y, z, w) y como resultado se obtienen las nuevas coordenadas del objeto (x', y', z') como se puede apreciar en la siguiente matriz:

$$\begin{bmatrix} m00 & m01 & m02 & m03 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ m30 & m31 & m32 & m33 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

De esta manera se puede trasladar el sistema de coordenadas mundiales a la posición del punto de vista, usando valores de coordenadas esféricas, para x, y, y z, con w como la distancia del sistema origen del sistema coordenadas de visión al origen del sistema de coordenadas mundiales.

5.6.5. Rotación continua

En la función *turn()* la rotacion en Y sólo se hace una vez hacia la derecha o izquierda; para hacer que la rotacion sea continua sin alterar la traslacion se realizan lo siguientes pasos:

1. Se declara una variable temporal antes del constructor, es decir, como variable global de *t04imple.java*.
2. Dentro del método *ProcessStimulus* en la estructura *switch* que compara la variable de tipo entero *handOrient* (donde se almacena el resultado del método *getRoll()* de la clase *FPSGLOVE*) se escribe el siguiente código:

```
switch (handOrient)
 {
 case 3:
 temprot+=-ROT_AMT;
 toRot.rotY(temprot);break;
 case 4:
 temprot+=ROT_AMT;
 toRot.rotY(temprot);break;
 case 5:
 toRot.rot.Y(0.0);break;
 }
```

De este modo, al detectar que el guante se inclina ya sea a la izquierda o derecha, la representación de la mano girará 10° de manera continua. Si se equilibra el guante, la representación de la mano es rotada en 0°.

5.6.6. Una clase para controlar el escenario virtual

La clase *t04escen.java* es la responsable de almacenar los objetos que poblarán el mundo virtual de la interfaz. En este primer acercamiento el único objeto que habita dicho mundo virtual es un cubo de colores, el cual representa la mano virtual que después se definirá con más detalle.

El diagrama jerárquico de la clase t04escen hasta este punto está representado en la Figura 5.5. En dicha figura se puede observar como la clase *t04imple* es instanciada con el nombre de *translationBehavior*. *objRotate* es hijo de *objTranslate* con el propósito de que la traslación y rotación se apliquen sobre el mismo objeto que es el *ColorCube*(cubo de colores de 0.3 metros), y a ambos *TransformGroups* se les asignan los valores iníciales de *translation* y *rotation* respectivamente. Tanto a *objTranslate* como a *objRotate* se les asigna el atributo *ALLOW_TRANSFORM_WRITE* para que puedan ser modificados en tiempo de ejecución de acuerdo al objeto *translationBehavior*.

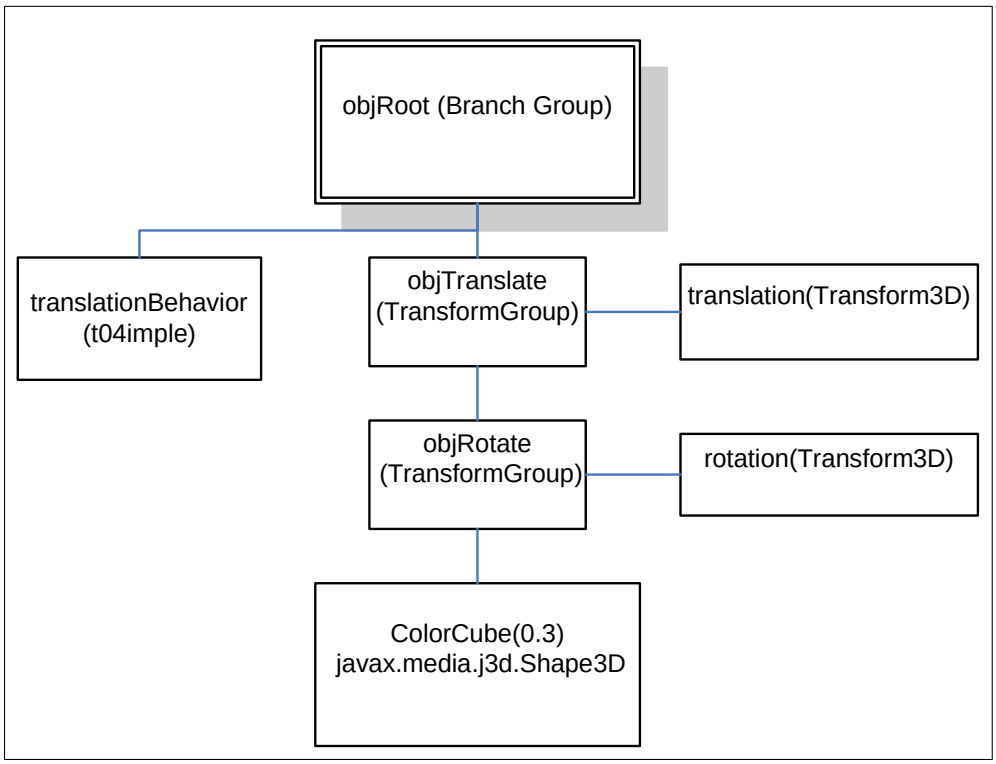

Figura 5.5: Diagrama jerárquico de la clase t04escen.java.

5.6.7. Diseño de la mano virtual

Una vez que se tiene el cubo con la capacidad de rotar y trasladarse de acuerdo a los movimientos del guante se plantea el diseño de la mano virtual. Para ello se dibujará una mano con medidas reales, para su posterior creación en Java 3D, siguiendo los pasos plasmados en el diagrama de flujo de la Figura 5.6.

Diseño de la mano en papel.

Se dibujó una mano en papel con base en las medidas físicas de mi propia mano, y que se pueden observar en la Tabla 5.1.

OBJETO	*X*	*Y*	*Z*
PALMA	7 cm.	2 cm.	10 cm.
DEDO INDICE	2 cm.	1.8 cm.	9 cm.
DEDO MEDIO	2 cm.	1.8 cm.	10 cm.
DEDO ANULAR	2 cm.	1.8 cm.	8.8 cm.
DEDO MEÑIQUE	2 cm.	1.8 cm.	6 cm.
DEDO PULGAR	2.4 cm.	1.8 cm.	6 cm.

Tabla 5.1 Medidas de la mano real.

Hacer una clase para la mano

La siguiente fase consiste en diseñar la mano de acuerdo a las medidas planteadas en la Tabla 5.1. En este primer acercamiento usaremos figuras geometricas basicas, en este caso cubos, para formar la palma y posteriormente los dedos de la mano.

Crear la palma de la mano. Al crear la palma se observa que el tamaño es demasiado pequeño en tiempo de ejecución, así que se escalará en 3 cm el tamaño de la palma y posteriormente el de los dedos. Como Java 3D no cuenta con un visor que permita manipular los objetos de manera gráfica (como el caso del *plug-in* de *VRML*) se tiene que aplicar una rotación para ver en perspectiva el diseño en tiempo de ejecución con el siguiente código:

```
//rotamos en 90 grados sobre el eje X, el cubo que representa la palma.
rotate.rotX(Math.PI/2.0d);
// Se define la palma.
Box Palma=new Box(0.27f,0.06f,0.3f,app);
TransformGroup ManoTGR1 = new TransformGroup(rotate);
//aplicamos la rotacion al TransformGroup de la mano
//ManoTGR1 incluye los dedos y la palma.
ManoBG.addChild(ManoTGR1);//Agregamos ManoTGR1 a la rama de contenido.
```

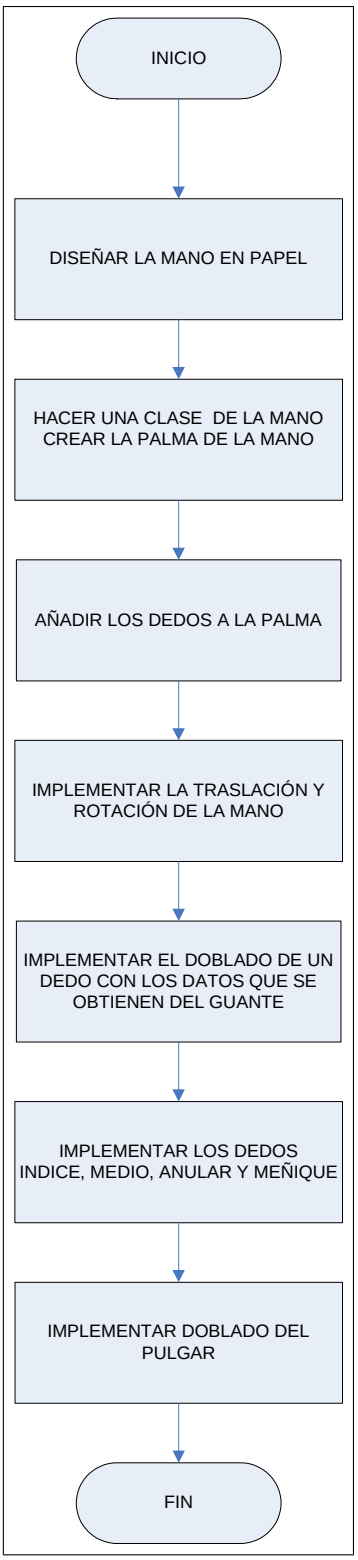

Figura 5.6: Diagrama de flujo del proceso para modelar la mano de la interfaz.

Añadir los dedos a la palma. Para ir agregando los dedos se utiliza la siguiente notación a las translaciones que se aplican a los dedos para posicionarlos de acuerdo a su orden natural:

TransformGroup dedoTGT1 para el índice
TGT2 para el medio.
TGT3 para el anular.
TGT4 para el meñique.
TGT5 para el pulgar.
translate—>tlate_ind(*Transform3D* para posicionar el dedo índice en la palma).
Box Dedo_ind (Cubo ó caja para representar al dedo índice).

Donde:
ind->índice
med->medio
anu->anular
men->meñique
pul->pulgar

Se debe agregar una rotación estática al dedo pulgar con el fin de darle una apariencia más natural, de la siguiente manera:

```
//Rotacion estatica del dedo pulgar
Transform3D rota_pul=new Transform3D();
rota_pul.rotY(Math.PI/8d);//Asignamos 22.5 grados.
TransformGroup PulTGR2=new TransformGroup(rota_pul);
```

En la Figura 5.7 se puede apreciar el diagrama jerárquico de la mano, con sus nombres y su tipo al que pertenecen de acuerdo a Java 3D y en la Figura 5.8 se puede observar la representación en pantalla. En ella, la mano presenta un aspecto blanco opaco, ya que no se ha agregado color, ni se ha definido el material, ni una fuente de iluminación para el mismo. Cabe destacar, que la parte de la iluminación para la clase de la mano (*Mano05*) se añade después de la creación de la escena, después de instrucción:

```
BranchGroup createSceneGraph()
```

En Java 3D existen diferentes tipos de iluminación como son ambiental, direccional, puntual (ó de bombilla), spot (ó reflector). También existen diversas características de reflexión de la luz, para definir el material de un objeto en particular. Para la mano virtual se define una luz blanca ambiental con el siguiente código:

```
AmbientLight ambientLight = new AmbientLight(new Color3f(1f, 1f, 1f));
ambientLight.setInfluencingBounds(bounds);
```

Se añade el objeto *AmbientLight* a la raíz del árbol de escena.

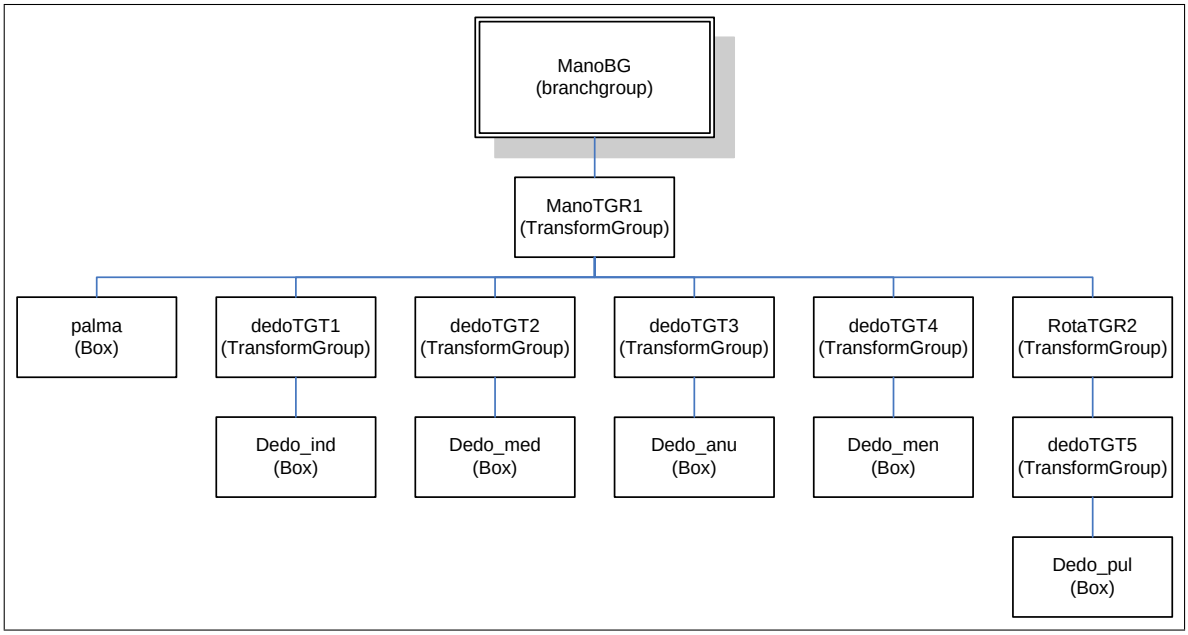

Figura 5.7: Diagrama jerárquico de la clase Mano.

Figura 5.8: Modelo de la mano en Java 3D.

```
objRoot.addChild(ambientLight);
```

Para modelar material existen los siguientes tipos de reflexión: emisivo, difuso, especular, y brillantez; estos se definen al momento de crear el objeto palma por medio del código siguiente:

```
material.setDiffuseColor(new Color3f(0.37f,0.37f,0.37f));
material.setSpecularColor(new Color3f(0.89f,0.89f,0.89f));
material.setShininess(17.0f);
material.setAmbientColor(new Color3f(0.8f,0.8f,0.8f));
```

Como resultado de los valores de iluminación y material se consigue modelar una mano con la apariencia del aluminio como se puede observar en la Figura 5.9.

Figura 5.9: Modelo de la mano virtual con apariencia de aluminio.

Implementar la traslación y rotación de la mano

Una vez que se han definido las clases para definir el modelo estático de la mano, para controlar el escenario y para manejar el comportamiento del guante, lo siguiente es implementar el modelado de la mano a las dos clases anteriores. Para ello existen dos alternativas:

1. Instanciar la clase *Mano06* y llamarlo dentro de *t05escen.java*
2. Copiar el código de *Mano06* desde *ManoTGR1* (Transform Group que abarca toda la mano) a la clase *t05escen.java*.

Con el fin de simplificar el desarrollo, la segunda opción es más atractiva ya que el trabajar con tres clases no es una tarea sencilla, puesto que se debe importar el *TransformGroup ManoTGR1* del *BranchGroup* de la clase *Mano06*, y este objeto no

se puede llamar de forma estática . Para el caso dos se tiene que importar el paquete *com.sun.j3d.utils.geometry* para poder hacer uso de figuras geométricas básicas como lo son las cajas. El código de la clase *Mano06* se toma a partir de la instrucción *Transform3D rotate = new Transform3D();* y se copia después de la instrucción *objTranslate.addChild(objRotate);* de la clase *t05escen.java*. Con ésto se obtiene la traslación y rotación del modelo de la mano y se continua con la implementación del doblez de los dedos.

Implementar el doblado de un dedo con los datos que se obtienen del guante

Lo primero que se debe hacer es medir el doblez que reporta el guante de cada uno de los dedos, ya que estos valores son diferentes para cada dedo. Dichos valores estan expresados en *P5 Finger Bent Units*, que es una medida interna utilizada para calibrar el guante Para ello se utilizó el programa *Showglove.class* y estos son los valores obtenidos:

1. INDICE 553 P5 Finger Bent Units.
2. MEDIO 648 P5 Finger Bent Units.
3. ANULAR 563 P5 Finger Bent Units.
4. PEQUEÑO 743 P5 Finger Bent Units.
5. PULGAR 696 P5 Finger Bent Units.

Para programar la lógica de la interfaz, se declara una variable booleana que devuelva 1 si el dedo en cuestión esta doblado ó 0 si no se cumple. Esto se programa en la clase *FPSGLOVE* y se recibe posteriormente en la clase *timple.java* con el nombre de *glove.indice_esta_Dblado()*.

Ahora corresponde modificar la clase que maneja el escenario virtual, por medio del siguiente código:

```
Transform3D rota_indi=new Transform3D();
rota_indi.rotX(0d);//iniciamos en 0 grados la doblez.
TransformGroup IndiTGR=new TransformGroup(rota_indi);
//Esta instruccion permite modificarse la figura en tiempo de ejecucion.
IndiTGR.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
```

Las siguientes lineas de código añaden el *TransformGroup* del dedo indice estático a el *TransformGroup* que controlará el doblado en tiempo de ejecución, *IndiTGR*. Éste *TransformGroup* es añadido a *ManoTGR1*, el *TransformGroup* de la mano que es rotado y trasladado de acuerdo a los movimientos del guante.

```
IndiTGR.addChild(dedoTGT1);
ManoTGR1.addChild(IndiTGR);
```

En *timple.java* se añaden las siguientes lineas:

```
private static final double ROT_DEDO=Math.PI/2d;//90 grados.
private TransformGroup TGRDEDO;
private Transform3D T_ROT_DEDO=new Transform3D();
private Transform3D T3D_ROT=new Transform3D();
private double tem_rot_dedo=0;
```

Para definir la condición del doblado del dedo:

```
if (glove.indice_esta_Dblado())
 tem_rot_dedo=ROT_DEDO;
else
 tem_rot_dedo=0;
 T_ROT_DEDO.rotX(tem_rot_dedo);
```

Acceder a IndiTGR

El siguiente paso consiste en multiplicar *tem_rot_dedo* por el *Transform3D* del dedo índice y luego aplicarlo. Se recuerda la forma en que se concatena la traslación y la rotación de la mano:

```
modificarTG.getTransform(t3d);
t3d.mul(toRot);
modificarTG.setTransform(t3d);
```

De la misma forma se debe concatenar la transformación estática del dedo (que incluye su posición y rotación) como parte de la mano en un momento determinado, con la rotación de 90° que se efectúa sobre el mismo dedo al momento en que se dobla el dedo del guante.

Antes de comenzar a trabajar la concatenación de las transformaciones del dedo índice, se renombra la clase *FPSGLOVE* a *t01inter*, para diferenciarla de la original con los cambios realizados al archivo tales como la creación de la función *indice_esta_Dblado*. Existen dos maneras diferentes de concatenar las transformaciones del dedo índice. La primera es haciendo uso del método de la clase *Group*(Grupo, nodo padre de la clase *TransformGroup* entre otras) llamado *getchild*, para obtener el *TransformGroup* del doblez del dedo *IndiTGR*.

Método getchild

getchild(int index) es un método público que devuelve un nodo hijo al especificarse un indice en la lista de nodos.

Parámetros:

- *index* - El nodo hijo a retornar.

Devuelve:

- El nodo hijo en la posición indexada indicada . El índice (*index*) debe ser un valor mayor o igual que 0 y menor que *numChildren()*.
- *CapabilityNotSetException* - Si el atributo *ALLOW_TRANSFORM_** no está establecido y el nodo es parte de un nodo *scenegraph* compilado.
- *java.lang.IndexOutOfBoundsException* - Si el índice es inválido.

Las siguientes líneas de código utilizan la función *getChild* para copiar el *TransformGroup objTranslate* que recibe de la clase *tescen*. Una vez que se obtiene el *TransformGroup*, se obtiene su *Transform3D* con la función *getTransform* y luego se multiplica por el *Transform3D* del dedo estático para finalmente aplicarse al *TransformGroup* de *TGRDEDO* (dedo doblado/no doblado) .

```
TGRDEDO=(TransformGroup) modificarTG.getChild(int_Indi);
TGRDEDO.getTransform(T3D_ROT_DEDO);
T3D_ROT_DEDO.mul(T_ROT_DEDO);
TGRDEDO.setTransform(T3D_ROT_DEDO);
```

Con este código la mano en su totalidad es rotada al doblar el dedo. El problema es obtener el *index* del dedo índice *IndiTGR*, ya que no hay manera de saber el número que le asigna Java 3D. Existe un método llamado *indexofchild* el cuál recupera el *index* de un nodo hijo enviado como parámetro. Su sintaxis es la siguiente:

public int *indexofChild*(Node child)

Parámetros:

- Hijo (child)- El nodo hijo deseado

Devuelve:

- El *index* del nodo *child* especificado
- -1 si el objeto no se encuentra en la lista del grupo.

La siguiente línea de código es para obtener el índice de *IndiTGR*. Sin embargo, al momento de compilar, recibimos un error de resolución de variables, indicando que no es posible acceder al nodo *IndiTGR* desde *modificarTG*.

```
int_Indi=modificarTG.indexOfChild(modificarTG.objRotate.ManoTGR1.IndiTGR);
```

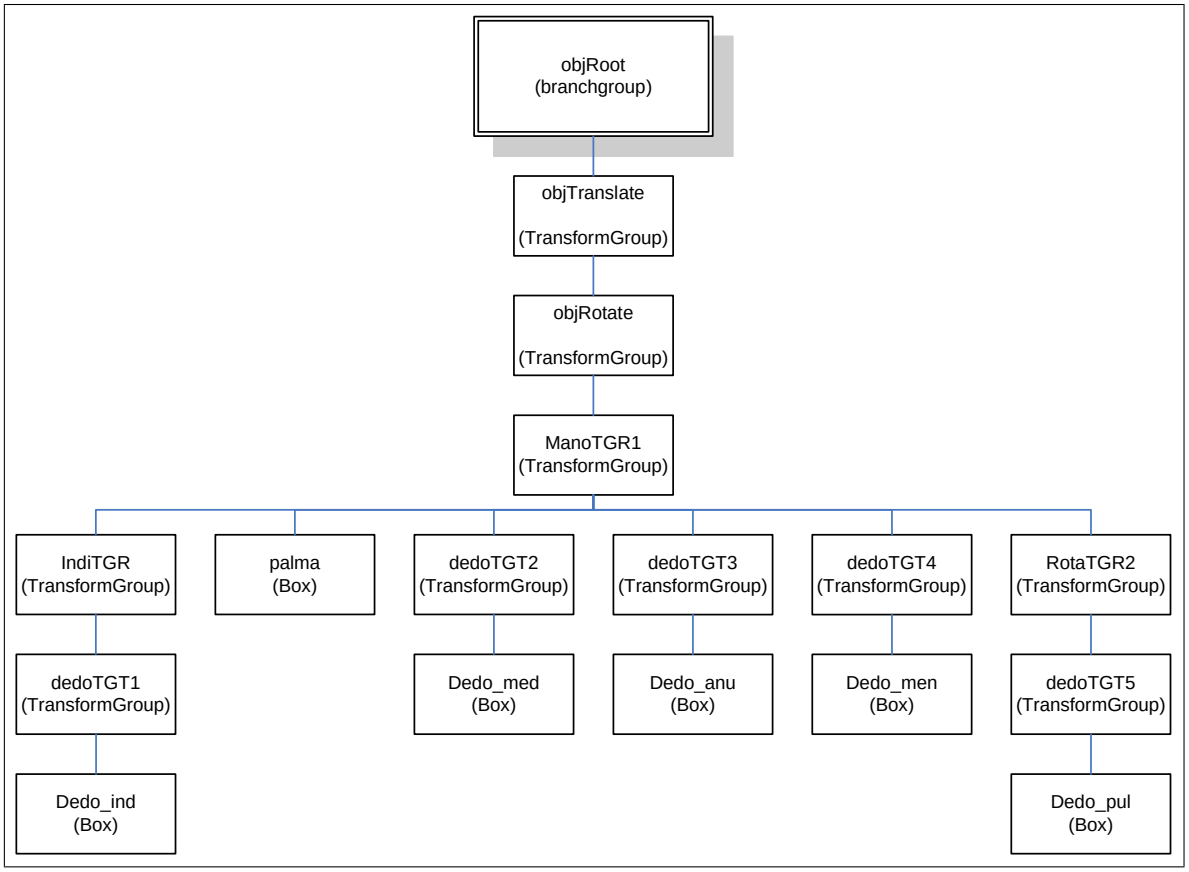

Figura 5.10: Diagrama jerárquico de la mano virtual con rotación y traslación dinámica.

La segunda alternativa consiste en enviar como parámetro el *TransformGroup* de cada uno de los dedos y trabajarlos en *timple.java*. Antes de empezar a explicar el proceso conviene repasar la estructura jerárquica de la clase *tescen*, con la traslación y rotación dinámica de la palma y de los cinco dedos. Ésta se puede observar en la Figura 5.10.

Para enviar *IndiTGR* a la clase *timple* se añade como parámetro de la siguiente forma:

```
t07bimple translationBehavior = new t07bimple(objTranslate,IndiTGR);
```

Con este cambio es posible doblar el dedo; sin embargo, se dobla más de lo que debe, a causa de un error de lógica en el siguiente fragmento de código de *timple*:

```
mIndiTGR.getTransform(T3D_ROT_DEDO);
T3D_ROT_DEDO.mul(T_ROT_DEDO);
mIndiTGR.setTransform(T3D_ROT_DEDO);
```

Este código hace que se conserve la rotación anterior, es decir, se van concatenando las transformaciones de rotación. La manera de corregirlo es quitar las dos líneas de *getTransform* y *mul,* dejando la siguiente línea:

```
mIndiTGR.setTransform(T_ROT_DEDO);
```

De éste modo se arregla problema y se observa que la rotación se esta efectuando en la parte positiva del eje. Hay que volverla negativa de acuerdo al movimiento natural de los dedos.

```
private static final double ROT_DEDO=-Math.PI/2d
```

También se debe trabajar aún más la rotación del dedo pues al doblarlo se sale de la mano. Al hacer la rotación de la mano sin perspectiva, se observa que la rotación se aplica sobre el eje Z, pero al doblar el dedo, éste se separa de la mano. Para solucionar este problema se propone agregar una mitad de dedo y colocarlo en la misma posición, para cuando se doble, de la apariencia de un dedo doblado a la mitad.

Implementación de una primera falange para los dedos

Para que el doblez de los dedos se lleve acabo sin que el dedo se salga de la mano, se debe establecer un punto de referencia, de modo que la parte que se dobla, se mantenga unida a la mano a pesar de que se mueva ésta o no. Por lo tanto, se va a implementar una mitad del dedo índice de diferente color para visualizar el efecto y perfeccionarlo, con otra apariencia y otro material. A ésta mitad estática de dedo se le denominara *PF*

(primera falange) aludiendo al hueso de la mano; para ello se crearán nuevos archivos: *t08escen* y *t08imple*. En *t08escen* se creara la nueva apariencia y material de color dorado con *app_pf* y *mat_dora*. La Tabla 5.2 presenta las medidas dela primera y segunda falange del dedo índice.

X	Y	Z
0.06 m.	0.064 m.	0.27/2=0.135 m.

Tabla 5.2. Medidas de la primera falange y el dedo índice

Como se puede observar, dividimos a la mitad el tamaño en Z de modo que no se altere el tamaño de los dedos.

Hay que agregar la transformación estática para posicionar la *PF* en el mismo lugar que el dedo índice,.por lo que se crea un *TransformGroup* que vaya a la misma altura que el *TransformGroup* de los dedos de nombre *Ind_pf_TGR* y su *Transform3D* de nombre *tlate_ind_pf*. A continuación se añade *Ind_pf_TGR* al *BranchGroup* del nodo *ManoTGR1* y se agrega el *Box Ind_PF* (la caja que representa a la primera falange) a *Ind_pf_TGR*. Se debe recorrer la primera falange hacia el eje Z negativo, pues su tamaño es menor.

Es importante mencionar, que se debe subir el dedo en el eje Y para posicionarlo en movimiento y después regresarlo a su posición original cuando se estire. Para lograr esto primero se crean nuevos archivos a partir de *t08*, que se llamarán *t08b*. En *t08imple* se cancela temporalmente el movimiento sobre el eje X para poder analizar mejor el doblez. Se imprime el código de ambos archivos y se empezara a revisar para aplicar la traslación del dedo doblado y como recuperar su posición normal. Se observa que se debe reducir el tamaño del dedo que se dobla. También se propone implementar el botón B del guante para cambiar el eje de la rotación en tiempo de ejecución, ya sea en el eje Y ó Z.

Reposicionamiento del dedo al doblarlo

Se va a implementar el reposicionamiento del dedo índice al doblarlo. Se empieza con *t08bimple*, que maneja el comportamiento. Para esto, se crea un nuevo *Transform3D* con las siguientes líneas:

```
private Transform3D trans_indice=new Transform3D
trans_indice.set(new Vector...)
```

Sin embargo, esta técnica no va a funcionar porque la traslación se debe hacer de acuerdo a la posición que tenga la mano en ese momento. Lo que se requiere es obtener del *TransformGroup* del dedo su posición usando el método *getTransform(TransformGroup)*. La sintaxis es la siguiente:

public void *getTransform*(*Transform3D* t1)
Parámetros:

- t1 – El objeto dónde se va a copiar la transformación.

Devuelve:

- *CapabilityNotSetException* – Si el permiso de acceso no has sido garantizado y este objeto es parte de un escenario (*scene graph)* activo ó compilado.

Se crean nuevos archivos para las pruebas *t09imple*, *t09escen*. La idea es reposicionar el dedo de acuerdo a la posición en Y de la primera falange, por lo tanto en *t09escen* se va a enviar el *TransformGroup* de *Ind_pf_TGR*. Con el método *getTransform* se obtiene el *TransformGroup* de *Ind_pf_TGR* con la siguiente línea:

```
Ind_pf_TGR.getTransform(trans_indice);
```

Una vez que se tiene el *TransformGroup* de la primera falange, se debe obtener la posición del objeto para los ejes X, Y, y Z. Para ello se hará uso del método *get(Vector3f)* de la clase *Transform3D*. La sintaxis se presenta a continuación:

public final void *get(Vector3f trans)*

Parámetros:

- trans – El vector que recibe la posición del objeto en X, Y, y Z.

```
trans_indice.get(Vec3f_indic_trans);
```

Una vez que se obtiene el vector de coordenadas del objeto, se prosigue a incrementar la posición de Z y disminuir la posición de Y cuando el dedo esté doblado.

```
Vec3f_indi_trans.z-=0.2;
Vec3f_indi_trans.y-=(float)fl_inc_t;
```

A continuación se aplica la traslación a *mIndiTGR* y se copia a *T3D_ROT_DEDO*.

```
mIndiTGR.setTransform(trans_indice);
mIndiTGR.getTransform(T3D_ROT_DEDO);
```

Finalmente se concatenan las transformaciones del dedo, estáticas y dinámicas y se aplican.

```
T3D_ROT_DEDO.mul(T_ROT_DEDO);
mIndiTGR.setTransform(T3D_ROT_DEDO);
```

Con esto se consigue doblar el dedo sin que se salga de la mano; lo que resta será reducir el tamaño de la primera falange y aumentar el tamaño de la parte que se dobla para darle un aspecto más real. La Figura 5.11 presenta la mano con el dedo doblado en tiempo de ejecución, y la Figura 5.12 el dedo sin doblar.

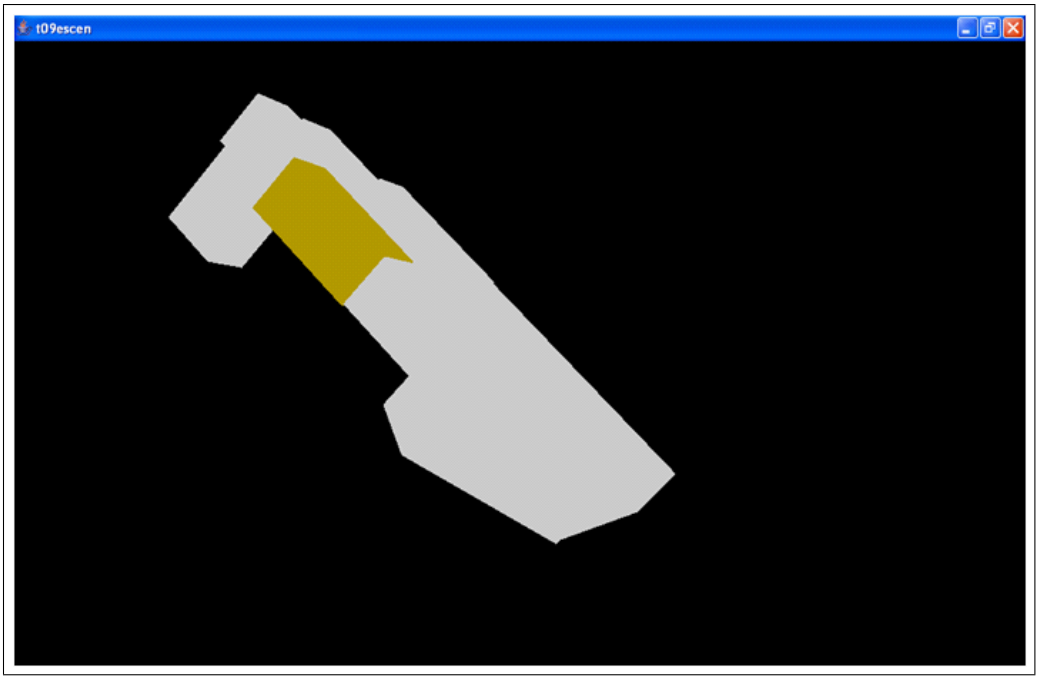

Figura 5.11: El dedo índice doblado.

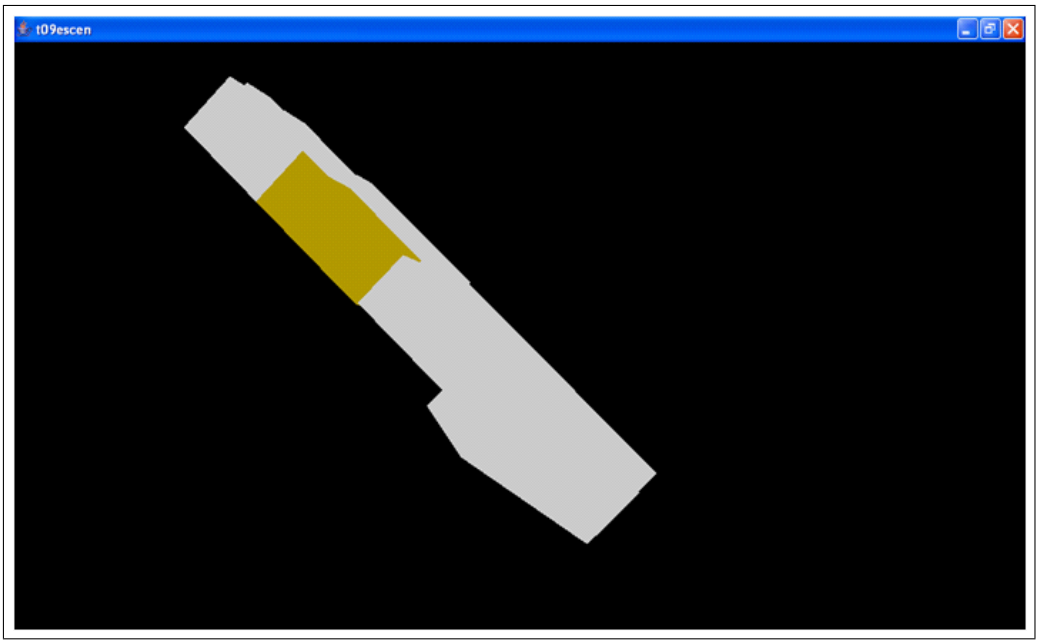

Figura 5.12: El dedo índice estirado.

Implementar los dedos medio, anular y meñique

Se va a trabajar con los demás dedos para implementar su doblez. Para ello se empezara con la clase *tescen* y luego se proseguirá a modificar *timple*. Los archivos actuales con los que se trabajaran son *t09escen y t09imple*.

El primer paso consiste en añadir el *Transform3D* del dedo medio así como inicializar el ángulo de rotación en 0. Posteriormente se aplica a su respectivo *TransformGroup* y se establece el atributo necesario para que pueda ser modificado en tiempo de ejecución.

```
Transform3D rota_Medi=new Transform3D();
rota_Medi.rotX(0d);
TransformGroup MediTGR=new TransformGroup(rota_Medi);
MediTGR.setCapability (TransformGroup.ALLOW_TRANSFORM_WRITE);
```

A continuación, se hace lo mismo con la primera falange del dedo medio y se añade su respectiva caja (*geometry Box*).

```
Transform3D tlate_med_pf=new Transform3D();
tlate_med_pf.set(new Vector3F(-0.072f,0f,*));
TransformGroup Med_pf_TGR=new TransformGroup(tlate_med_pf);
Med_pf_TGR.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
Box Med_pf=new Box(0.06f,0.064f,0.15f,app);
```

Del código de *t09escen* se deben eliminar las *Transform3D tlate_ind* y el *TransformGroup dedoTGT1*; en su lugar *Dedo_ind* (que es una caja) se agrega directamente a *IndiTGR*:

```
IndiTGR.addChild(Dedo_ind);
```

Para mantener un formato estándar en el código se cambia el nombre de las siguientes variables y se agregan las líneas faltantes.

```
private double fl_inc_t=0; por fl_inc_tIND
private Transform3D trans_medio=new Transform3D();
private Vector3f Vec3f_indi_trans=new Vector3f();
private double fl_inc_tMED=0;
private Transform3D T_ROT_DEDO por T_ROT_ind
private double tem_rot_dedoIND=0;
private Transform3D T3D_ROT_DEDOind;
ManoTGR1.addChild(Med_pf_TGR);
Med_pf_TGR.addChild(Med_pf);
```

Se envían los TransformGroup del dedo Medio y su primera falange.

```
t09imple...=new t09imple(objTranslate,.. MedTGR, Med_pf_TGR);
```

Ahora se presentan los cambios para la clase *timple.java*, y se añaden las siguientes líneas referentes al dedo medio.

```
private TransformGroup mMedTGR;
private TransformGroup Med_pf_TGR;
private Transform3D T_ROT_DEDOmed=new Transform3D();
private double tem_rot_dedoMED=0;
private Transform3D T3D_ROT_DEDOmed=new Transform3D();
```

Se añaden los TransformGroups que se reciben de *tescen* en el constructor de *timple*.

```
t09imple(TransformGroup mMediTGR, TransformGroup Med_pf_TGR)
 {
 this.mMediTGR=mMediTGR;
 this.Med_pf_TGR=Med_pf_TGR;

   ...
```

Se añade la condición que detecta si el dedo medio está doblado.

```
if (glove.medio_esta_Dblado())
 {
 tem_rot_dedoMED=ROT_DEDO;
 fl_inc_tMED=0;
}
else
 {
 tem_rot_dedoMED=0;
 fl_inc_tMED=0;
 }
T_ROT_DEDOmed.rotX(tem_rot_dedoMED);
```

Finalmente, se enlistan los pasos a seguir para doblar la simulación del dedo medio de la misma forma como con el dedo índice.

1. Se obtiene la matriz de transformación visual de la primera falange.
2. Se copia las posición a la variable Vec3f_medi_trans.
3. Se recorre hacia atrás el dedo sobre el eje Z.

4. Si el dedo esta doblado, se recorre hacia abajo de acuerdo al valor almacenado en la variable fl_inc_t.

5. Se aplican las nuevas coordenadas de posición al Transform3D llamado trans_medio.

6. Se aplica la translacion al TransformGroup mMediTGR usando como referencia a trans_medio.

7. Se recupera la matriz de transformación visual de mMediTGR y se almacena en T3D_ROT_DEDOmed.

8. Se multiplican la matriz de transformación anterior por la rotación del dedo.

9. Se aplica la matriz resultante a mMediTGR

   Ésta metodología es aplicable al dedo anular y el dedo meñique con la salvedad de que se tuvieron que ajustar los tamaños de los dedos y se verificó que la simulación resultante fuera lo más natural posible, por lo tanto se omite esa parte.

Implementar el doblado del dedo pulgar.

Se trabajan los archivos *t092escen* y *t092imple*. Dentro de *t092escen* se define *PulTGR2* para almacenar la rotación estática del pulgar y *PFPulTGR* para la rotación estática de la primera falange. Al *TransformGroup* del doblado se le nombra *PulgTGR*; una vez definidos dichos *TransformGroups*, se debe efectuar un cambio en el eje de rotación en el momento en que se dobla el dedo; en vez de doblarse sobre el eje X, el doblez se debe efectuar sobre el eje Y. Al igual que en los otros dedos, se debe hacer una ajuste de posición sobre los ejes Z y X. La diferencia principal entre el dedo pulgar y los demás dedos consiste en que el dedo pulgar, debido a su naturaleza, requiere una rotación estática adicional y que además se dobla sobre un eje diferente.

La Figura 5.13 presenta la mano virtual con los cinco dedos estirados, mientras que en las Figuras 5.14 y 5.15 aparece con los dedos doblados. El siguiente paso será afinar los detalles de la simulación tanto en su funcionalidad como en su estética. Esos cambios contemplan sustituir las cajas geométricas por cilindros, y modificar la traslación y la rotación de la mano de modo que se trabaje en modo *absoluto* total, es decir sin constantes de por medio.

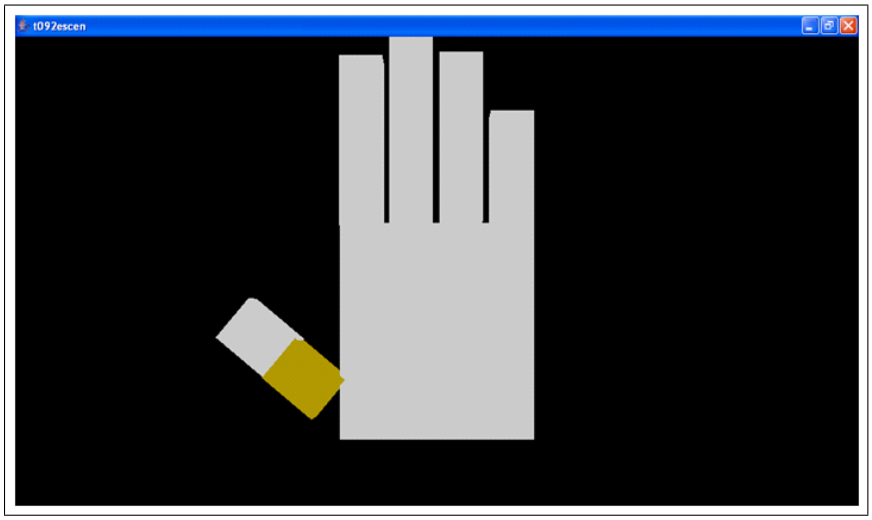

Figura 5.13: Mano virtual con los cinco dedos estirados.

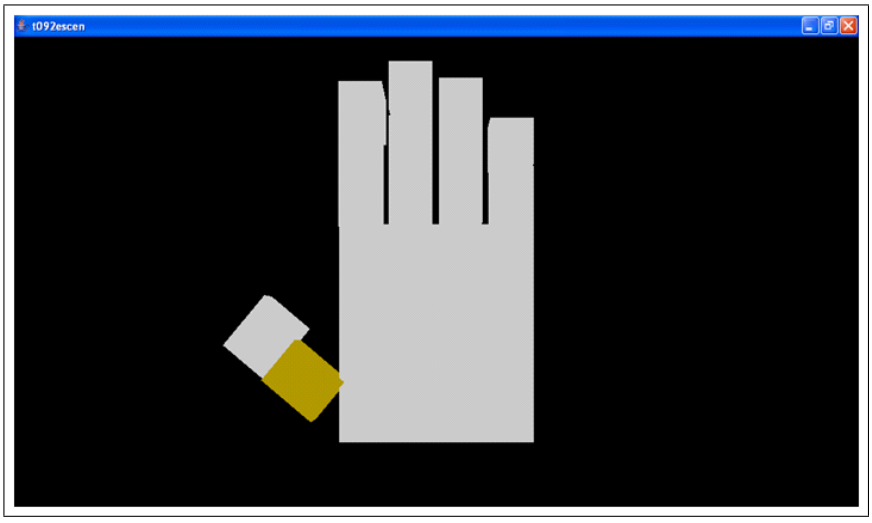

Figura 5.14: Mano virtual con los cinco dedos doblados.

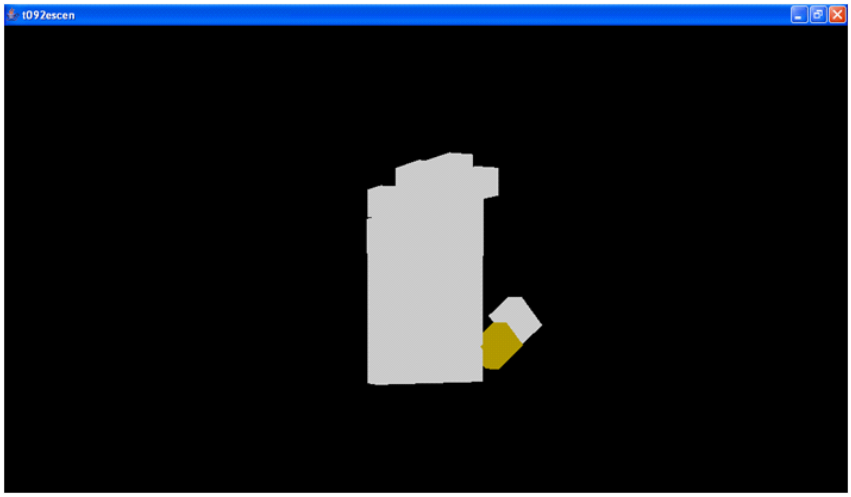

Figura 5.15: Mano virtual girando con los cinco dedos doblados.

5.6.8. Mejoras al diseño de la mano virtual

A) Cambiando la caja geométrica del dedo índice por un cilindro

Cambiar la caja geométrica por un cilindro es más complicado debido a que hay que rotar el cilindro 90° para acostarlo, lo que implica una rotación estática adicional. Además, se debe ajustar la posición del dedo, ya que por su misma naturaleza no coincide con las coordenadas de la caja geométrica que se trabajó con anterioridad. Se comienza por la primera falange y se continúa con la segunda falange que es la parte que se dobla del dedo. Las medidas de la caja geométrica son las siguientes: X =0.06 m., Y= 0.064 m., Z=0.135 m. Se realizan los cambios en las medidas para ajustar el cilindro a la mano y se obtienen las siguientes medidas: Radio r=0.064 m., Altura h=0.28 m.; las medidas de la primera falange se conservan. Además, se deben intercambiar los valores de la constante *glove.indice_esta_Dblado*, ya que el cilindro acostado significa que el dedo está estirado y en su posición natural que está doblado. También se debe ajustar su posición en Y para bajarlo cuando éste se dobla, por lo que se añade la siguiente variable:

```
private double fl_inc_tzIND=0;
```

Una vez que se comprueba que el cambio fue exitoso, se procede a trabajar con los demás dedos. Para ello se toman como base las medidas del cilindro del dedo índice y simplemente se efectúa una regla de tres para los demás dedos, sustituyendo la altura. Como el tamaño de los dedos es diferente, al igual que en la mano real, se hacen los ajustes necesarios en cuanto a posición. El siguiente paso será convertir el dedo pulgar a cilindro.

B) Cambiando la caja geométrica del dedo pulgar por un cilindro

Para sustituir la caja geométrica del dedo pulgar por un cilindro debemos tomar las siguientes consideraciones en cuanto a las transformaciones estáticas del mismo.

1. Existe una rotación de 90° sobre el eje X.
2. Una rotación de 60° sobre el eje Y.

Para el punto 1, se añade un nuevo *TransformGroup* de nombre *TGpulPF* con su respectivo *Transform3D (rot_ind_pf)*. Éste *TransformGroup* se inserta en la parte superior del *PF* del pulgar, como se puede observar en la Figura 5.16. El punto 2 se tiene contemplado dentro de *PfPulTGR*.

En cuanto al doblez del dedo pulgar (para ajustarlo a la *PF*), se debe tomar en cuenta que la traslación del mismo es directamente afectada por las rotaciones estáticas aplicadas con anterioridad. Por lo tanto, los ejes son alterados de acuerdo a la posición del cilindro, como se puede observar en la Figura 5.17. Una vez explicado el punto de los ejes, se hacen las modificaciones pertinentes para ajustar la PF a la mano.

Para la segunda mitad del dedo, el procedimiento es similar. Se añade un rotación estática en X de nombre *PulTGREX*, lal cual se agrega entre el *TransformGroup* de la rotación en estática en Y (*PulTGR2*), y la definición del cilindro (*Dedo_pul*). En la clase *t0943imple* se debe rotar el pulgar -80° sobre el eje Y en caso de que se doble el dedo y 0° cuando está estirado. La Figura 5.18 presenta la mano virtual con los cinco dedos estirados, haciendo uso de cilindros y la Figura 5.19 la misma mano pero con los dedos doblados.

C) Adaptando la traslación y rotación de modo absoluto parcial a modo absoluto total

Para que el movimiento en los ejes Z y X sea absoluto total y no absoluto parcial debemos plantear una regla de tres con la coordenada 0, 0,0 como la posición inicial en el mundo virtual, y compararla con la que registre el guante en un momento determinado. A continuación se envía el valor transformado como parámetro para que éste sea la nueva coordenada de la simulación de la mano. Se toma como base los valores iníciales propuestos por el Dr. Davison para su clase *FPSGLOVE* que son:

$zPos < 500$ cerca de la torre

$zPos > 900$ lejos de la torre

$500 \leq Enmedio \leq 900 \therefore 700$ es el punto medio de dicho intervalo. Para obtener la coordenada dentro del mundo virtual multiplicamos 100 por la posición reportada por el guante y la dividimos entre 700. Al efectuar los cambios en *t02inter* y *t0943escen* se puede observar que el resultado es satisfactorio, y que además se reduce el código significativamente para la función *t02inter.getZPosition ()*de:

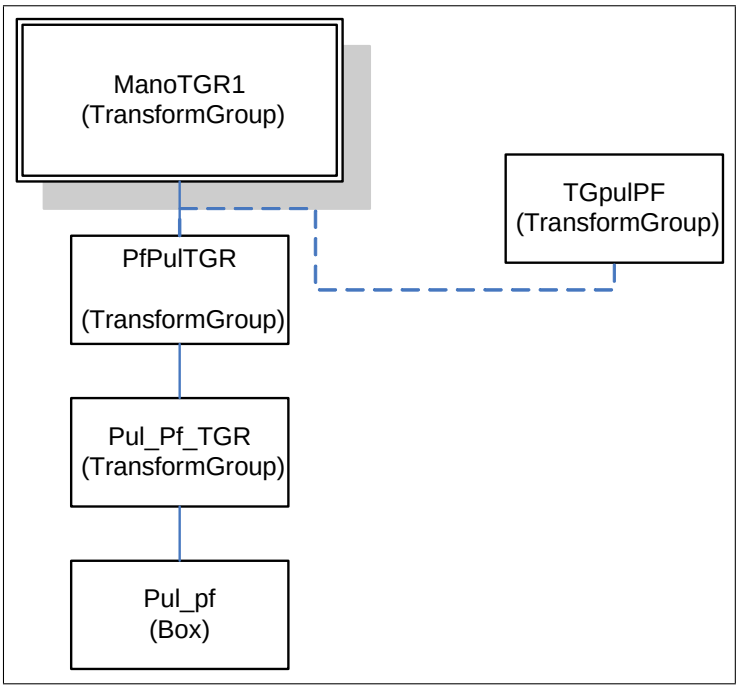

Figura 5.16: Diagrama Jerarquico de la primera falange del pulgar.

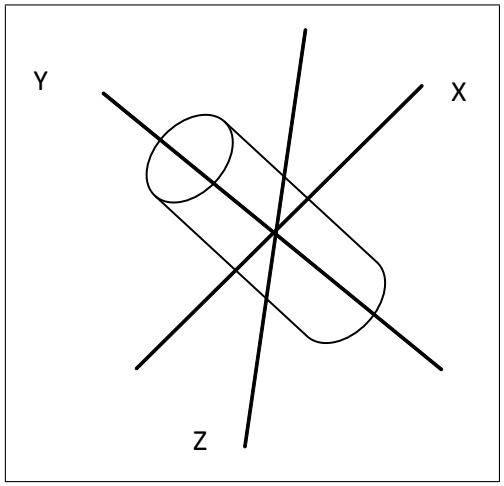

Figura 5.17: Ejes traslacionales en un cilindro rotado.

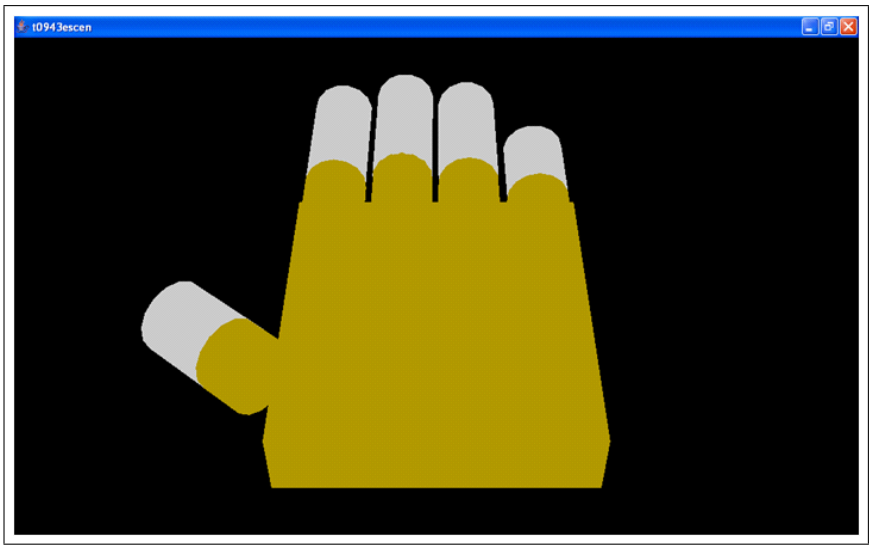

Figura 5.18: Mano virtual con los dedos estirados usando cilindros.

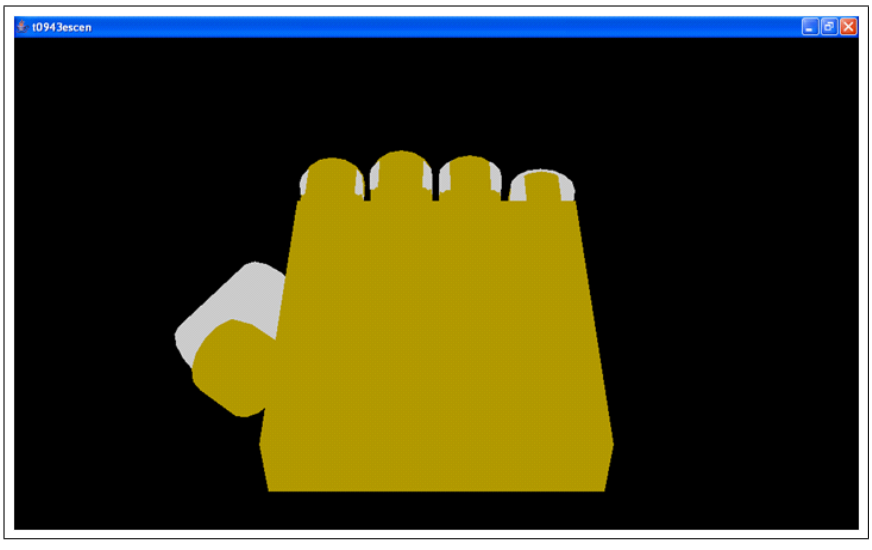

Figura 5.19: Mano virtual con los dedos doblados usando cilindros.

```
if (zPos < NEAR_MIN) { // near to the tower 0
 // System.out.println("front (" + zPos + ")");
 return NEAR;
 }
 else if (zPos > FAR_MIN) { // far from the tower 1
 // System.out.println("back (" + zPos + ")");
 return FAR;
 }
 else
 return MIDDLE;
```

A:

```
float mzpos;
 float zPos = gloveState.filterPos[2]; // posicion filtrada del eje Z
 mzpos=zPos/700;
 return mzpos;
```

Ahora toca el turno a la rotación. Como constantes se tienen los siguientes valores:

$roll < -40°$ girado a la izquierda.
$roll > 80°$ girado a la derecha.
$-40° <= Equilibrado \leq 80°$ por lo tanto equilibrado es igual a 20°. De igual forma que en la traslación se aplica una regla de tres multiplicando la posición por 1 y el resultado se divide entre 20. Lo anterior queda definido en la función *getRoll()* de la siguiente forma:

```
public float getRoll()
{
 float roll=gloveState.FilterRoll;
 float mroll=roll/20;
 return mroll;
 }
```

Para convertir en ángulos dicho valor se divide 180/*mroll* en la clase *t0943imple*.

```
amroll=180/handOrient;
```

Al hacer las pruebas pertinentes, la mano virtual se observa inestable, y al profundizar en los valores recibidos por el guante a través de la clase ShowGlove, se puede confirmar que los valores de rotación oscilan por lo que no es posible mantener la estabilidad. Éste problema es derivado del guante en si mismo y no hay manera de corregirlo. El camino a seguir es mantener el modo relativo en la rotación, agregando una constante en caso de que la mano sea volteada. Además se suprime la rotación continua de la mano virtual cuando se alcanza el intervalo explicado anteriormente y el código del método de la rotación queda de la siguiente manera:

```
public int getRoll()

 {
   float roll = gloveState.filterRoll;
   if (roll < LEFT_ROLL_MIN) {
    return ROLL_LEFT;
    }
   else if ((roll > RIGHT_ROLL_MIN)&&(roll<VOLTEADA_MIN)) {
    return ROLL_RIGHT;
    }
   else if (roll>VOLTEADA_MIN) {
    return VOLTEADA;
    }
   else
    return LEVEL;
 }
```

Y en *t0943imple*.

```
if (roteje)

 switch(handOrient){
  case 0:
   toRot.rotZ(Math.PI);
  break;//volteada
  case 3:
   //temprot+=ROT_AMT;
   temprot=ROT_AMT;
   toRot.rotZ(temprot);break;//izquierda
  case 4:
   //temprot+=-ROT_AMT;
   temprot=-ROT_AMT;
   toRot.rotZ(temprot);break;//derecha
  case 5:toRot.rotZ(0.0);break;//no hacemos nada
  }
```

Las líneas comentadas con // son las líneas que fueron sustituidas para eliminar el rotamiento continuo.

Para la traslación absoluta en el eje X se toma el valor filtrado directamente del guante con el siguiente código:

```
float xPos = gloveState.filterPos[0]; // posicion filtrada del eje X
return xPos;
```

Sin embargo, se detecta que el movimiento sobre dicho eje es demasiado rápido. Para solucionar este problema, se propone dividir el valor filtrado entre 10 , 100 y finalmente entre 1000; con eso se consigue que los movimientos de la mano virtual sobre el escenario no sean tan largos, por medio del siguiente código.

```
float xPos = gloveState.filterPos[0]; // posicion filtrada del eje X
float mxpos=xPos/1000;
return mxpos;
```

5.6.9. Características de la interfaz final

Como resultado de las mejoras al diseño de la mano virtual, surgen las clases *t0944escen*, *t0944imple* y *t032inter*. En la Figura 5.20 se presenta el diagrama de clases con los métodos de cada una de éstas y en la Figura 5.21 se presenta el diagrama jerárquico del diagrama de escena de la interfaz.

La clase principal es *t0944escen.class,* la cual contiene el método *main()*. Ésta clase instancia a la clase *t0944imple* y esta a su vez a la clase *t032inter*.

La estructura de la interfaz es la misma que en una clase *Behavior* genérica, pero con la adición de una clase extra para comunicarnos con el guante y que es una abstracción de la clase *CP5DLL* del controlador dual versión 3. Esta clase, llamada *t032inter* es la encargada de sensar el guante cada 75 milisegundos y devuelve el estado a la clase *t032inter*. Además, se optimizó el código de dicha clase al eliminar las constantes de conversión propuestas por Davison para ejecutar la traslación de la mano.

Gracias a éste modelo, es posible separar el diseño de la mano virtual del comportamiento y de la información que se recibe del guante. El resultado final es una interfaz funcional, elaborada en una plataforma no comercial como lo es Sun Java, Java 3D, y el controlador dual del guante P5. Dicha interfaz permite mover la mano virtual en los ejes X y Z de manera natural e intutiva, así como rotar en el eje Y ó Z. También es posible detectar el doblado de uno o más dedos y por medio de los botones que vienen integrados en el guante, es posible reposicionar a la mano virtual en el origen, y cambiar el eje de rotación.

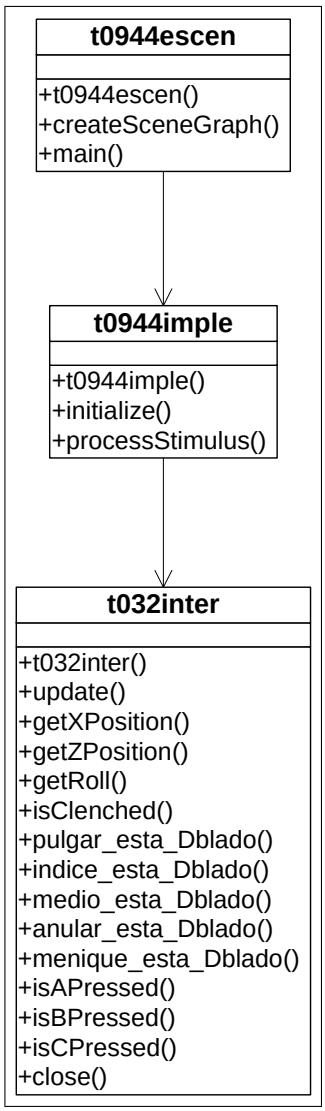

Figura 5.20: Diagrama de clases de la Interfaz para el aprendizaje de la realidad virtual haciendo uso del P5 glove.

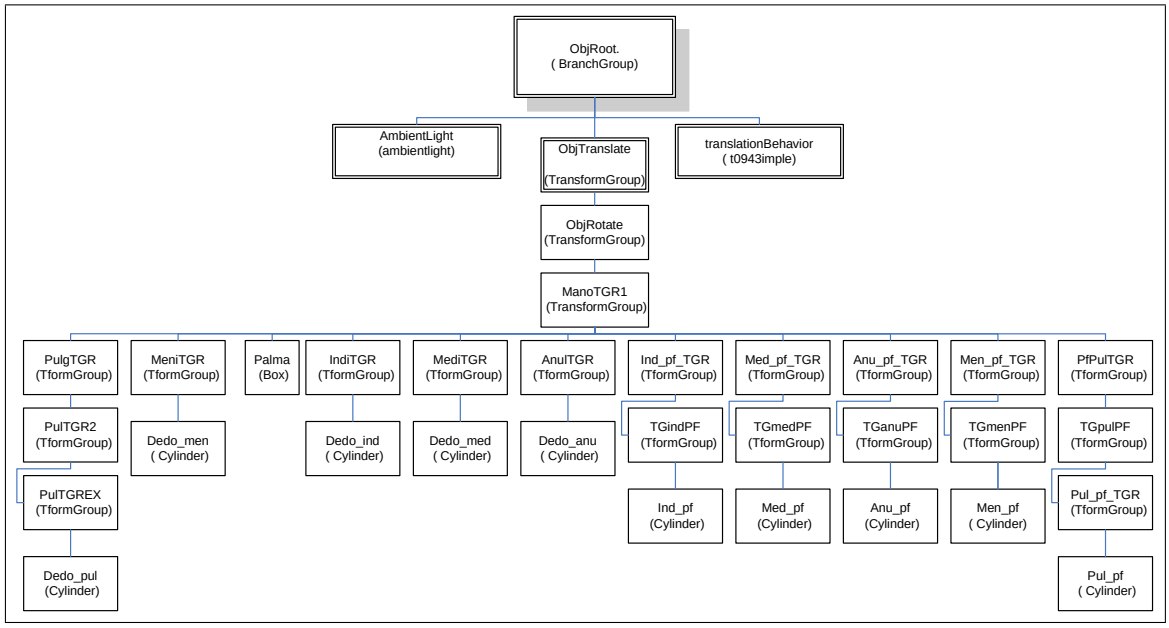

Figura 5.21: Diagrama de escena de la clase t0944escen.class.

Capítulo 6

Conclusiones y perspectivas

Conforme aumentan y mejoran las características de hardware de las computadoras, los usuarios desean programas más elaborados y capaces de explotar dichos recursos. Hasta hace poco la realidad virtual era considerada como un mercado para las empresas y centros de investigación con suficientes recursos para sufragar los altos costos de esta tecnología. Sin embargo, con la aparición de lenguajes como VRML o Java 3D y con ellos el surgimiento de ambientes virtuales de escritorio, el usuario puede experimentar la realidad virtual desde la comodidad de su hogar y aplicar ésta tecnología en diferentes áreas del conocimiento. De los dispositivos que componen los sistemas de realidad virtual, los guantes de datos son los dispositivos más utilizados en la simulación, en conjunto con los cascos, debido a que permiten al usuario interactuar de una manera más natural e intuitiva.

En este trabajo se presentó una propuesta de interfaz para el aprendizaje de realidad virtual, que une los ambientes virtuales de escritorio con un guante de datos, de modo que el usuario pueda sustituir el uso del ratón al navegar por un mundo virtual. Esto permite que los gestos y movimientos que realice con su mano sean reflejados por la simulación de una mano virtual como primer acercamiento a un sistema de realidad virtual inmersivo.

Se describieron los elementos que componen un sistema de realidad virtual, la diferencia entre realidad virtual inmersiva y no inmersiva, y como se puede agregar un guante de datos a un sistema de escritorio con el fin de enriquecer la experiencia del usuario. Así como su importancia en proyectos de investigación recientes como dispositivo de entrada preferido para interfaces que requieren mayor precisión en el control a distancia; su naturalidad para llevar a cabo tareas cotidianas como la organización de archivos y la manipulación de objetos en equipos de cómputo.

Se presentaron algunas características de los principales guantes de datos comerciales, como son: tipo de conexión, accesibilidad, herramientas de programación disponibles, alimentación eléctrica, y precio. Se profundizó en el estudió de las características del guante P5 porque es un dispositivo económico, fácil de instalar y accesible que dispone

de varias herramientas de programación en diferentes plataformas y que es ideal para trabajar en ambientes virtuales y como instrumento de mando en aplicaciones remotas.

Una vez seleccionado el guante, se presentaron las herramientas de programación disponibles, y se hicieron pruebas para determinar cuál de ellas era la más apropiada de acuerdo a las características de la interfaz producto de ésta tesis. Como resultado se obtuvieron conocimientos sobre Visual C++, VRML, y Java 3D para programar el guante P5, y algunas consideraciones sobre estas herramientas para desarrollos futuros.

En cuanto al desarrollo, se presentó una introducción a la programación en Java 3D y se trataron aspectos de modelado en tercera dimensión, diagramas de escena, y programación orientada a objetos. También se profundizó en el uso del controlador dual de Kenner [20] y las clases de Davison [23].

Para la programación de la interfaz, se aprovechó el modo absoluto explicado ampliamente en el capítulo 3, con el fin de reducir las líneas de código y algunos errores al momento de sensar el guante, que se producen con el modo relativo debido a que no se conoce con exactitud la posición física del guante, y las mediciones erróneas cuando éste es apartado de la zona marcada por la torre de recepción. Uno de los logros más significativos, fue eliminar las constantes de conversión para efectuar la traslación en los ejes X y Z únicamente, ya que se proporcionó mayor naturalidad a la mano.

El proceso de programación ha sido desglosado por medio de diagramas de flujo, jerárquicos, de clase y fragmentos de código, de manera cronológica. También se presentaron pantallas de los resultados obtenidos. Conforme se fue avanzando, la estructura del mundo virtual se fue haciendo más compleja y se tuvo que profundizar en aspectos técnicos del despliegue de objetos virtuales en Java 3D.

A través del uso de matrices de transformación visual y de aprovechar las estructuras de datos provistas por el lenguaje de programación como son: *TransformGroup*, *Transform3D* y los métodos heredados de las clases *Group* y *Node*, fue posible anidar translaciones, rotaciones y escalamientos para formar clases de comportamiento más complejas. De esta forma, se pudo conseguir la traslación-rotación de la mano y el doblez de los dedos con base en las primeras falanges de los dedos. Ésta forma de trabajar las transformaciones en tiempo de ejecución, sienta un precedente para desarrollar clases de control de comportamiento de objetos virtuales, más complejas.

La arquitectura de la interfaz se conservó en tres clases que abstraen el modelo de la mano, la implementación de los movimientos y gestos, y la comunicación con el guante. De este modo, es posible sustituir en lo futuro el guante de datos ó la representación de la mano con una serie de cambios menores.

Por todo lo anterior, se estima que mediante una interfaz que simula los gestos y movimientos de la mano, a través de un guante de datos en un ambiente virtual, es posible acelerar la curva de aprendizaje de los estudiantes de realidad virtual [8]. Por

lo que este desarrollo permite al estudiante del Instituto un mejor desempeño en la construcción del conocimiento con la ayuda de la tecnología de realidad virtual, siendo ésta una herramienta indispensable dentro del contexto educativo presente.

Los resultados obtenidos en esta tesis, presentan algunas limitantes. Por ejemplo, en el desarrollo de la interfaz se omitió la traslación sobre el eje Y debido a la dificultad que presenta para el usuario controlar la traslación en tres ejes de manera simultanea. En el caso de la rotación, se conservó el uso de constantes para efectuar los cambios al modelo en tres dimensiones, ya que el guante presenta discontinuidades en el envío de datos hacia la computadora. La estética de la representación no proporciona una apariencia realista de la mano virtual; esto se puede mejorar en gran medida al modelar el objeto final para proporcionar a los usuarios una sensación de mayor realismo por medio de estrategias de modelado tales como: la extrusión de sólidos, modelado con curvas, mapeo de texturas, digitalización de objetos reales, etc. La manipulación de los objetos está limitada al modelo de la mano; es posible extender la acción del modelo a objetos ajenos de modo que actué como un manipulador para mundos virtuales y consiguiendo con esto una mejora en el diseño, desarrollo e implementación de entornos simulados. La interfaz actual fue desarrollada como un sistema independiente pero con la posibilidad de trabajar en conjunto con desarrollos presentes y futuros de un sistema inmersivo completo de realidad virtual que incluyan sistemas de visualización estereoscópica, sistemas de rastreo corporal, sistemas de seguimiento de posición de los pies (caminadora), sistemas de múltiple visualización, etc. Finalmente, el desarrollo de mundos virtuales en Java 3D es un proceso complejo en comparación con VRML, que facilita más el desarrollo de entornos virtuales, por lo cual es recomendable generar una compatibilidad del guante con entornos desarrollados en VRML a través del uso de la *Interfaz de Autoría Externa* (EAI) propia de VRML ó a través de la importación de mundos VRML desde programas en escritos en Java 3D.

Algunas posibles extensiones a los trabajos realizados en esta tesis son:

- Aprovechar la interfaz para manipular instrumentos musicales virtuales.
- Integrar la interfaz con un sistema de comunicación inalámbrica para operar un robot a distancia.
- Utilizar la interfaz para desarrollar cursos de aprendizaje en tercera dimensión.
- Diseñar aplicaciones para educación en general que utilicen realidad virtual como herramienta para facilitar el aprendizaje, y que hagan uso de la interfaz diseñada en este trabajo.

Referencias

[1] Sloan Kevin, De Cesar Megan, Knowles Brendan, McGuire Steve, Moshtagh Megan, and Trump, Jonathan. "Integrated astronaut control system for EVA", *RASC-AL Forum The Pennsylvania State University, Mars Society, E.U.A., 2003.*

[2] Pinho, Marcio S. et al. "Robot programming and simulation using virtual reality techniques", *Virtual Reality Group, PUCRS, School of Informatics, Brasil, 1999.*

[3] Fujii Yuki, Furusho Junji, and Koyanagi Ken'ichi. "Development of VR-STEF system with force display glove system", *Dept. Of Enginnering, Osaka University. ICAT Christchurch, Nueva Zelanda, 2005.*

[4] Timo Ropinski, Frank Steinicke, and Klaus Hinrichs. "Tentative results in focus-based medical volume visualization", *In Conference Supplements of SmartGraphics 2005.*

[5] Frank Steinicke, Timo Ropinski, Klaus Hinrichs. "A generic virtual reality software system's architecture and application", *Institut für Informatik, ICAT Christchurch, Nueva Zelanda, 2005.*

[6] Ali Akgunduz and Hang Yu. "Two-step 3-dimensional sketching tool for new product development", *Dept of Mechanical and Industrial Enginnering, Concordia University, Canada, 2004.*

[7] Andreas Dengel, Stefan Agne, Bertin Klein, Achim Ebert and Matthias Deller. "Human-centered interaction with documents", *Knowledge Management Lab, Intelligent Visualization Lab, DFKI GmbH, HCM, Alemania, 2006.*

[8] José P. Molina, Arturo S. García, Diego Martinez, Francisco J. Manjavacas, Victor Blasco, Victor Lopez and Pascual Gonzalez. "The development of glove-based interfaces with the Tres-D methodology", *Lab. Of User Interfaces and Software Engineeting, Instituto de Investigación en informática de Albacete, University of Castilla-La Mancha, España, 2006.*

[9] Mandelis James. "Genophone: An evolutionary approach to sound síntesis and performance", *School of Cognitive and Computing Sciencies, University of Sussex, Brighton, Reino Unido, 2001.*

[10] Jamsa Kris, Lalani Suleiman, Weakley Steve, "Programación para el web", pp 241-244,*McGraw-Hill, México, 1998.*

[11] VRML97 and related specifications, Web3D Consortium. *http://www.web3d.org/x3d/specifications/vrml/*

[12] P5 Glove Wikia, *http://scratchpad.wikia.com/wiki/P5_Glove*

[13] "Data Gloves" Virtual Realities: Global distributor of quality virtual reality products, *http://www.vrealities.com/glove.html*

[14] Un mundo virtual, *Revista Muy interesante, Editorial Televisa. Año XX Nº 11, 1 nov 2003.*

[15] Grigore Burdea, Philippe Coiffet. "Virtual reality technology 2da edición", *Wiley-Interscience, 2003.*

[16] Realidad virtual, Network-Press.Org, *http://www.network-press.org/?realidad_virtual*

[17] Kahlesz Ferenc, Zachmann Gabriel, Klein Reinhard. "Visual-fidelity dataglove calibration", *Dept. of Computer Sciencie II, University of Bonn, Alemania, 2004 .*

[18] Zeltzer D and Sturman D J. "A survey of glove-based input" , *IEEE Computer Graphics and Applications, E.U.A., 1994.*

[19] P5 Developers, Videogamealliance, *http://www.videogamealliance.com/*

[20] Carl Kenner Dual mode Driver, *http://www.geocities.com/carl_a_kenner/p5glove.html*

[21] Parallelgraphics, *http://www.parallelgraphics.com/*

[22] Official FAQ (Frequently Asked Questions) for the VRML 2.0 External authoring interface. *http://www.frontiernet.net/~imaging/eaifaq.html*

[23] Davison Andrew. "Pro Java 6 3D game development", *Apress, E.U.A. , 2007.*

[24] Sun's Java 3D, *http://java.sun.com/products/java-media/3D/index.html.*

[25] P5 Glove Community, *http://groups.yahoo.com/group/p5glove.*

[26] Java 3D Parent Project, *https://java3d.dev.java.net/*

[27] Java en castellano, *http://www.programacion.net/java/tutorial/3d/5/*

[28] Pratdepadua, Joan J., "Programación en 3D con Java 3D", *Alfaomega-Rama, México, 2003.*

[29] Poole Steven, "Trigger happy: videogames and the entertainment revolution", *Fourth Estate, Reino Unido, 2001.*

[30] Dennis J Bouvier, "Getting started with the Java 3D^{TM} API Chapter 1", *Sun Microsystems, Inc, E:U.A.,1999.*

[31] Ortega Carrillo Hernando, Godoy Aguirre Victor Hugo, y Ramos Nava Carmen. "Prototipo de un sistema de captura de movimiento", *Universidad Nacional Autonoma de México, México, 2007.*

[32] Soriano Jaime, Rivero Alejandro. "Hi-Tech PFCS", *http://arivero.ciberpunk.net/hi-tech-pfcs.*

[33] Beier K.P. "Entrenador de football virtual", *Virtual reality lab, University of Michigan, E.U.A., 2003.*

Apéndice A

Manual de Usuario

En este apéndice se presenta el manual de usuario de la *Interfaz para el aprendizaje de la realidad virtual haciendo uso de un guante de datos* dividido en los siguientes puntos:

1. Requerimientos de Hardware.
2. Instalación de Java 2 SDK.
3. Instalación de Java 3D.
4. Instalación de las clases que conforman la Interfaz.
5. Iniciar la clase *t0944escen*.
6. Uso de la clase *t0944escen*.

Antes de comenzar la instalación se debe verificar que no exista alguna versión de J2SDK, Java Runtime Environment o Java 3D instalada en el sistema. Esto se puede verificar accediendo a *Panel de Control->Agregar o Quitar Programas*. En caso de contar con alguna versión de los programas mencionados, debe proceder a ejecutar la desinstalación de cada uno de ellos y reiniciar.

A.1. Requerimientos de Hardware

Las características de hardware para ejecutar la interfaz están basados en los requerimientos necesarios para instalar el guante P5, así como la plataforma Java 2 mismos que se presentan a continuación.

- Computadora personal con un procesador Pentium a 200 MHZ. ó superior.
- 32 MB de memoria RAM.
- 15 MB de espacio disponible en el disco duro para la instalación del P5 más 200 MB (aprox.) para la instalación de la plataforma Java.

- Un Puerto USB libre compatible con el estándar 1.1.
- CD-ROM de velocidad 4X o superior.
- Microsoft Windows 98 ó superior (Éste manual de usuario ésta basado en Windows XP).
- Monitor Super VGA, 256 colores a 1024x768 o superior.
- Tarjeta de video con Chip 3D, compatible con DirectX 8.1 y/o OpenGL 1.2 ó versiones más recientes.

A.2. Instalación de Java 2 SDK

Java 2 SDK es el entorno gratuito de desarrollo de Sun Microsystems , que permite compilar, depurar e interpretar código escrito en Java. Para descargar la última versión diríjase a la siguiente dirección web: http://java.sun.com/javase/downloads/index.jsp**.** En este mismo sitio también se puede descargar Java 3D.

Una vez que haya descargado la última versión de Java 2 SDK a su disco duro, proceda a realizar un doble clic sobre el icono. Esto abrirá el asistente de instalación de la plataforma de programación de Java. Siga las instrucciones que vayan apareciendo y procure **no cambiar los nombres de directorios que aparecen por defecto**, ya que algunas veces, el cambiar un nombre o dirección puede causar problemas de compilación. En la Figura A.1 se puede observar la pantalla del instalador de Java versión 1.4.0.

Al terminar la instalación se añadirá el directorio j2sdk1.x.x a la raíz del disco duro de su computadora. Para comprobar que la instalación ha sido correcta, desde el menú incio seleccione ejecutar; teclee cmd y después enter. A continuación escriba `java --version` y deberá observar una pantalla similar a la de la Figura A.2.

A.3. Instalación de Java 3D

Una vez que haya descargado la ultima versión de Java 3D, haga doble clic sobre el icono de instalación del archivo java3d-1-xxx. Al igual que en la instalación de Java SDK, se abrirá un asistente como el que se puede apreciar en la Figura A.3. Siga las instrucciones de instalación y acepte del mismo modo, las opciones que por defecto aparecen.

Es importante mencionar, que existen dos versiones de Java 3D, una para DirectX y otra para OpenGL. Si UD. Cuenta con Direct X 8.0 instalado en su computadora, instale el primero. En caso contrario deberá instalar la versión para OpenGL.

Para comprobar que la instalación de Java 3D ha sido exitosa, entre a la carpeta de c:\j2sdk1.xxx\jre\lib\ext y verifique que existan los archivos con extensión .jar,

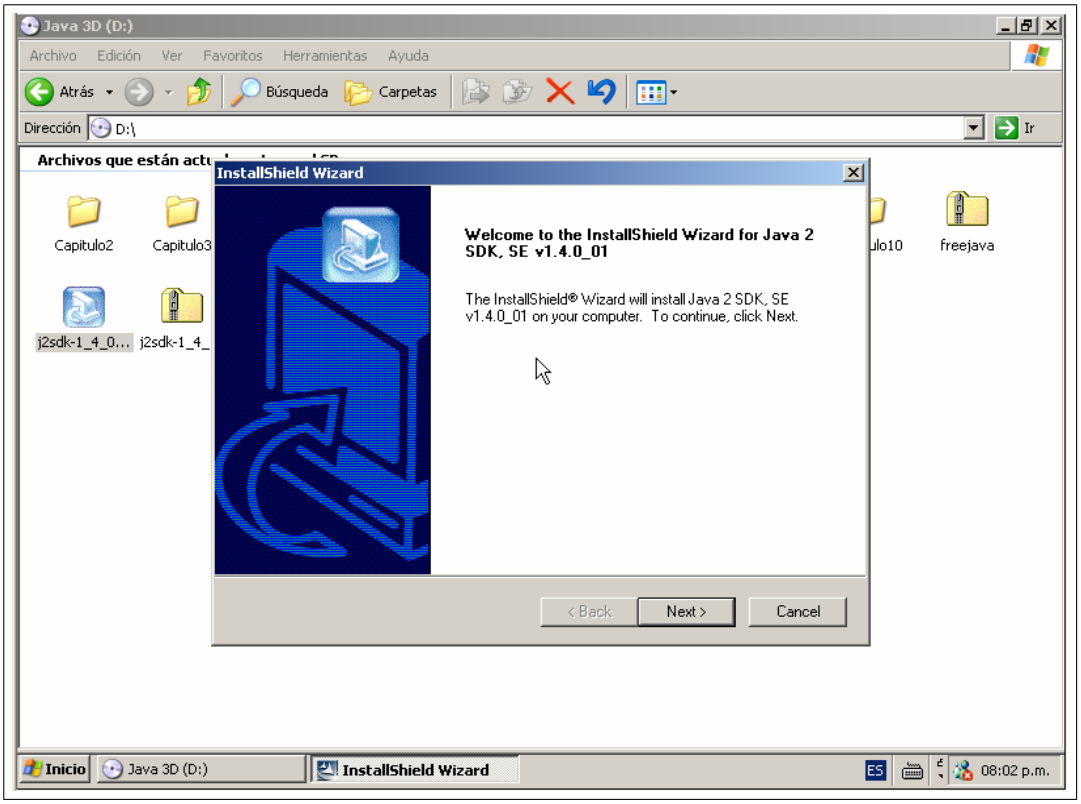

Figura A.1: Instalador de Sun Java versión 1.4.0.

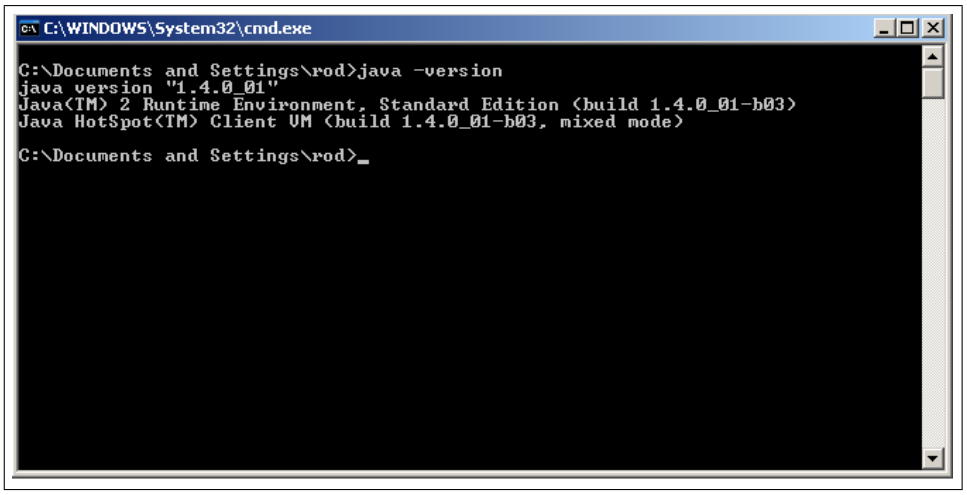

Figura A.2: Cómo verificar la instalación de Java 2 SDK.

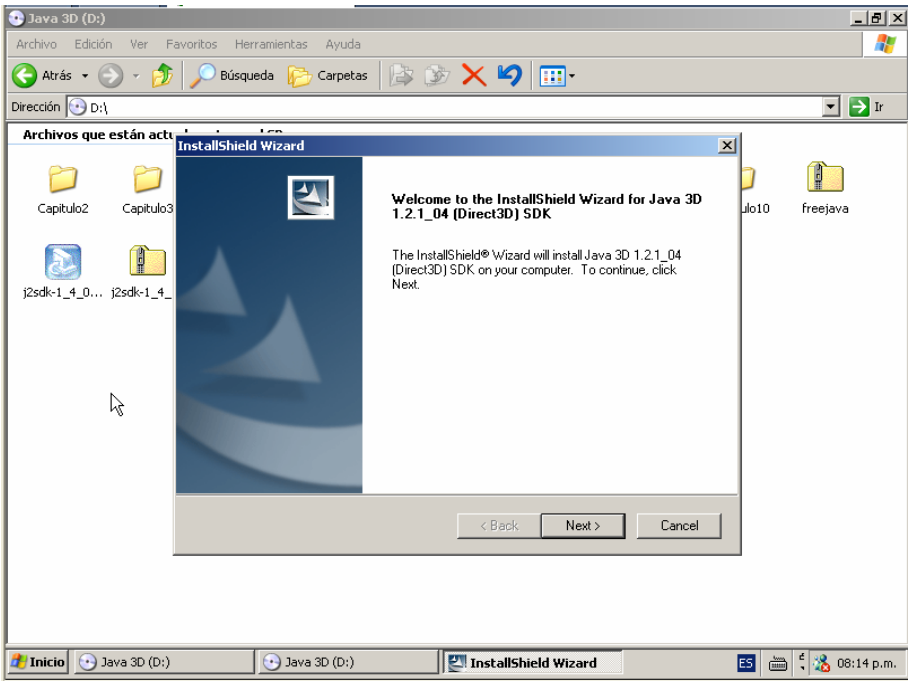

Figura A.3: Asistente de instalación de Java 3D.

j3daudio, *j3dcore* y *j3dutils*. Si existen dichos archivos dirigase a la siguiente dirección: c:\j2sdk1.xxx\demo\java3d\HelloUniverse\HelloUniverse.html. Al entrar se cargará un applet donde podrá observar un cubo de colores girando.

En las Figuras A.4 y A.5 se pueden observar las pantallas de verificación de la instalación respectivamente.

A.4. Instalación de las clases que conforman la Interfaz

Para instalar las clases que conforman la interfaz del presente trabajo de tesis, debe contar con el controlador dual beta versión 3 de Kenner. Éste puede ser descargado de la siguiente dirección: http://www.geocities.com/carl_a_kenner/p5glove.html ó en el Grupo del guante P5, http://tech.groups.yahoo.com/group/p5glove/. Una vez que se ha descargado el controlador dual (formato comprimido *.zip*), el siguiente paso es descomprimirlo en la raíz c:\. Para ello puede utilizar algún programa de descompresión, o la utilidad de carpetas comprimidas de Windows como se muestra en la Figura A.6. Una vez descomprimidos los archivos, abra la carpeta y copie el archivo P5DLL.DLL en el directorio de c:\windows\system32\. Si ya existe dicho archivo (por ej. al instalar el C.D. que viene con el guante), renombre el archivo existente para que pueda copiar el archivo en dicha carpeta. El siguiente pasó será copiar la carpeta com\essentialreality*.*, que

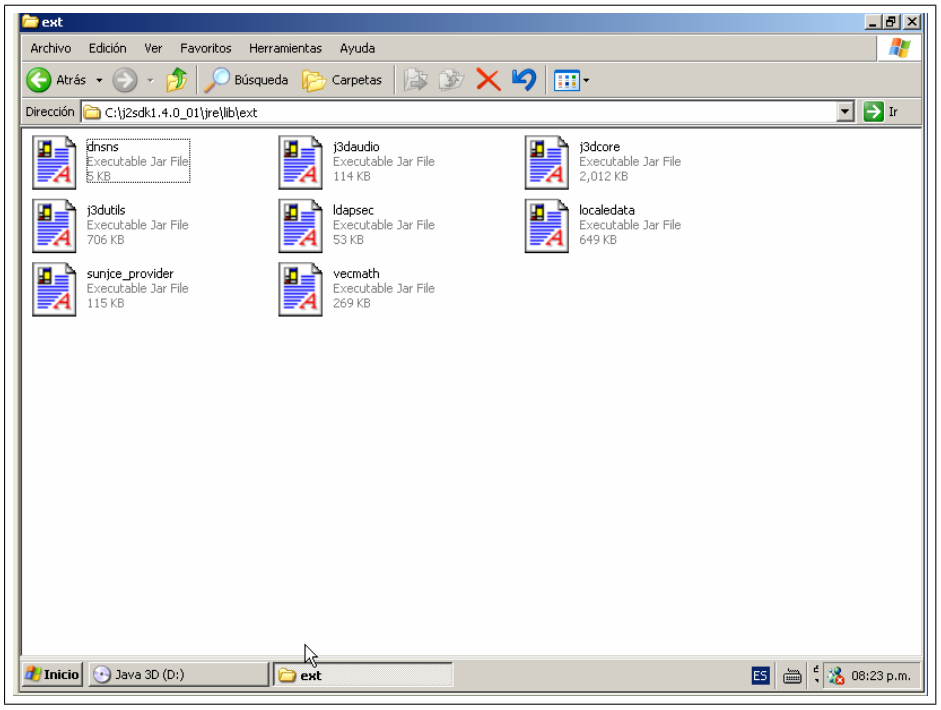

Figura A.4: Archivos de Java 3D.

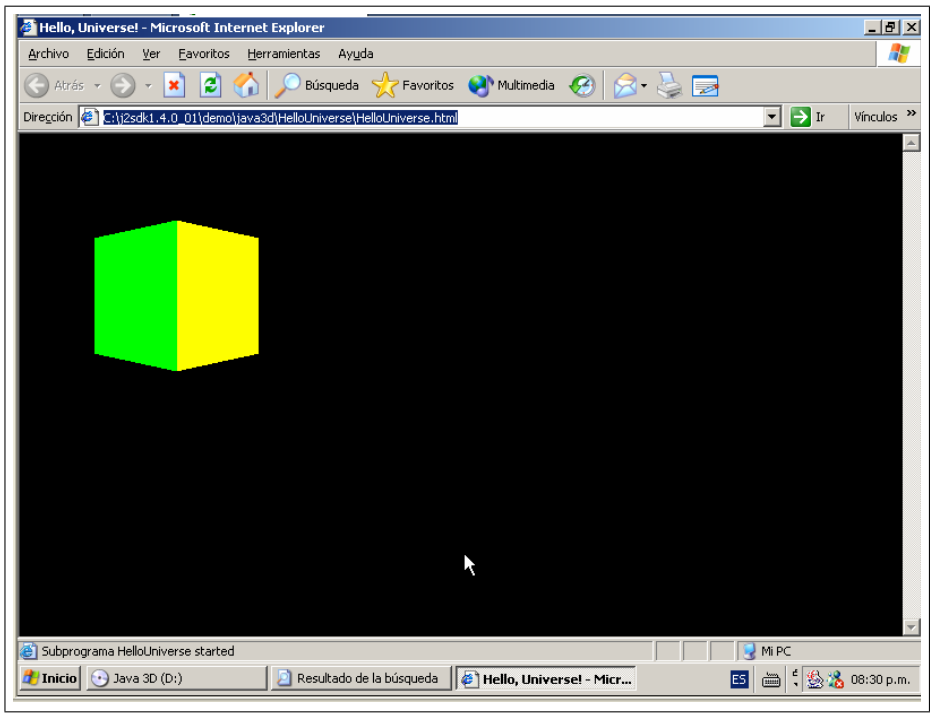

Figura A.5: HelloUniverse.html. Clase para comprobar que se instaló correctamente Java 3D.

se encuentra en el directorio include de los archivos que descomprimió, así como las clases *t0944*.class* y *t032inter.class* en c:\.

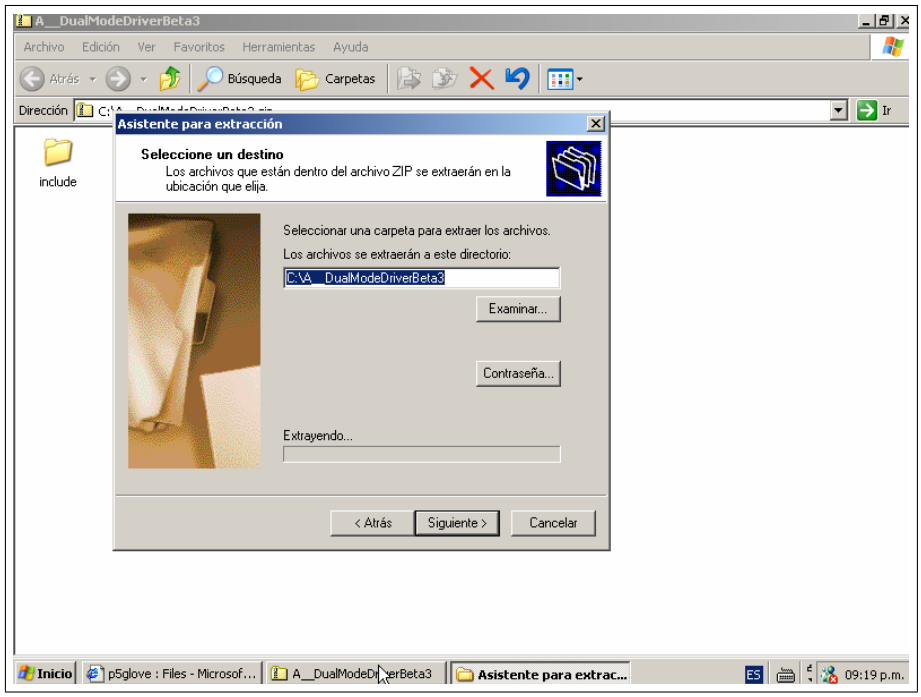

Figura A.6: Descomprimir el controlador dual por medio del Asistente de extracción de Windows.

A.5. Iniciar la clase t0944escen

Antes de iniciar la clase *t0944escen.class*, asegúrese de conectar el guante a la torre de recepción, y conecte esta a algún puerto USB de su computadora. Al conectar el guante, el led rojo de la torre se encenderá y el cursor del mouse se moverá conforme los movimientos del guante. Si el guante está fuera del alcance de la torre, éste parpadeara hasta que se acerque de nuevo al rango de alcance.

Para iniciar la clase *t0944escen.class*, dirigase al menú Inicio, Ejecutar y escriba `cmd` seguido de Enter. Teclee `cd\`. y escriba a continuación, `java t0944escen`. Se abrirá una ventana y aparecerá en el centro la pantalla de la interfaz. Se observará la mano virtual en pantalla y esta seguirá los movimientos del guante.

A.6. Uso de la clase t0944escen

Una vez que se ha iniciado la interfaz, aparecerá un formulario en el centro con la simulación de la mano virtual como la que se puede apreciar en la Figura A.7.

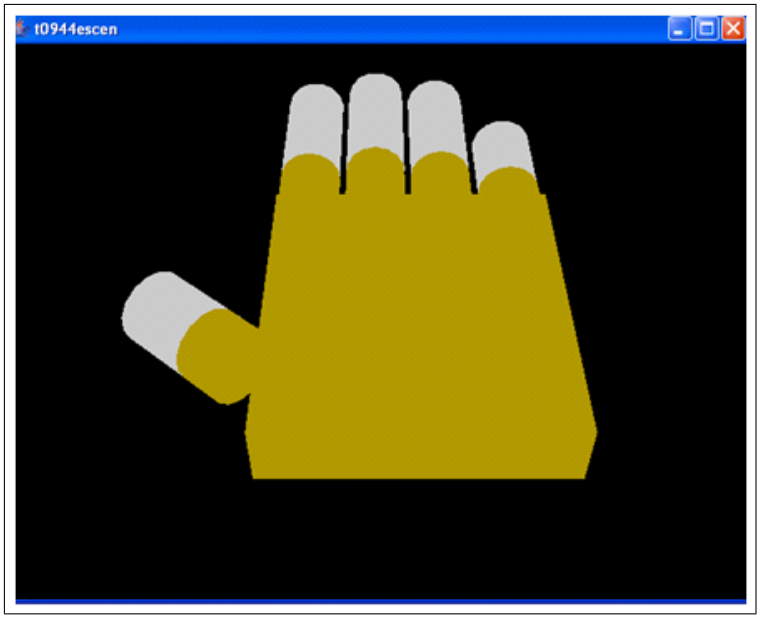

Figura A.7: Uso de la clase t0944escen.

A.6.1. Traslación y doblado de los dedos

Para mover la mano virtual, sólo mueva el guante hacia la derecha o la izquierda, hacia delante o atrás. Incluso puede combinar los movimientos, por ejemplo adelante y a la derecha, ó adelante y hacia la izquierda, etc. También puede doblar los dedos del guante y combinarlo con los movimientos anteriores. En la Figura A.8 se puede observar 4 dedos doblados y el índice estirado apuntado hacia adelante. En la Figura A.9 se pueden observar el guante trasladándose hacia la izquierda.

A.6.2. Rotación

Para rotar la mano virtual, incline el guante hacia la izquierda o derecha. También se puede voltear la mano 180°. Para cambiar el eje de Rotación de la mano oprima el botón B para rotar sobre el eje Y, y el botón C para rotar sobre el eje Z. Recuerde dirigir el guante siempre hacia la torre de recepción, ya que si se retira el guante del alcance de la misma, la mano virtual se comportará de la misma forma. Para reposicionar el guante en el centro de la ventana, oprima el botón A. En la Figura A.10 se puede observar la mano rotada a la derecha sobre el eje Z, y en la Figura A.11 se puede observar la mano rotada sobre el eje Y con los dedos doblados.

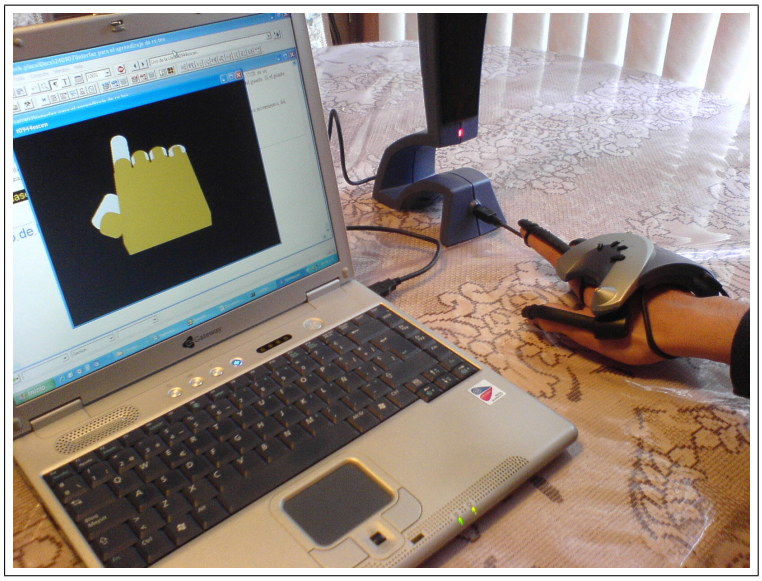

Figura A.8: Interfaz con 4 dedos doblados, y el índice estirado apuntado hacia adelante.

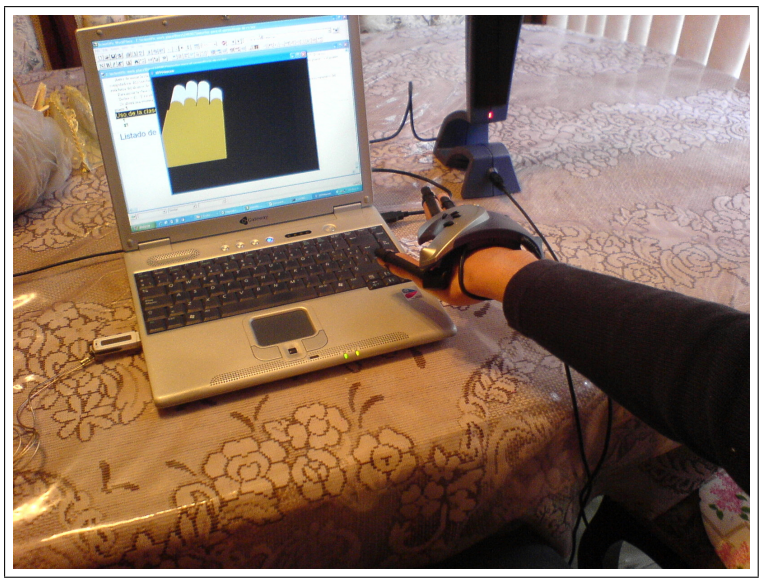

Figura A.9: Interfaz trasladándo la mano hacia la izquierda.

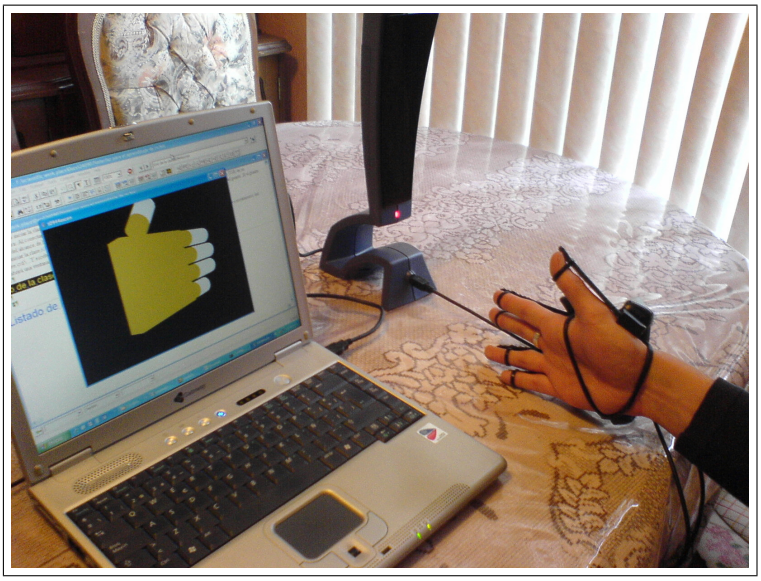

Figura A.10: Interfaz con la mano rotada a la derecha sobre el eje Z.

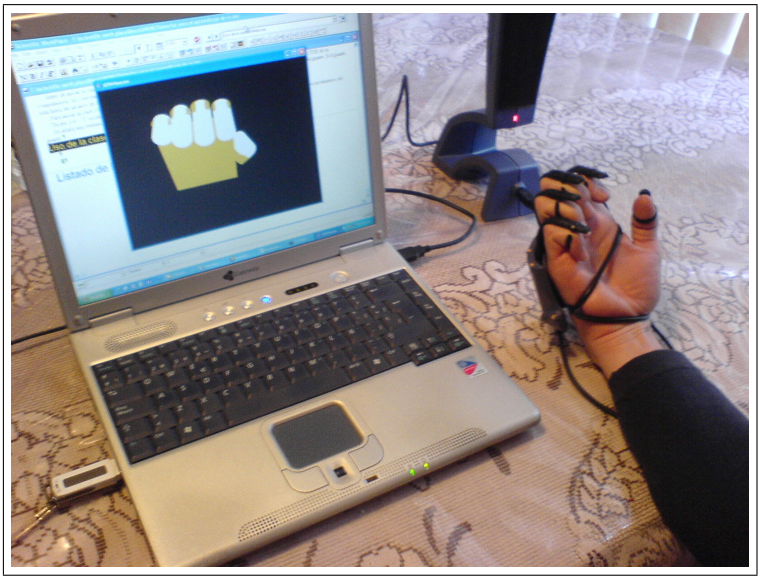

Figura A.11: Interfaz con la mano rotada sobre el eje Y con los dedos doblados.