UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Diseño y construcción de un sistema inmersivo y de un guante con retorno táctil para el modelado de objetos 3D con las manos

Jonatan Martínez Muñoz

Septiembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA POLITÉCNICA SUPERIOR

Departamento de Sistemas Informáticos

PROYECTO FIN DE CARRERA

Diseño y construcción de un sistema inmersivo y de un guante con retorno táctil para el modelado de objetos 3D con las manos

Autor: Jonatan Martínez Muñoz

Director: José Pascual Molina Massó

Septiembre, 2007

Resumen

Durante nuestra vida cotidiana extraemos gran cantidad de información del entorno que nos rodea mediante el sentido del tacto. Esta información, que se almacena en forma de modelos mentales, es aprovechada entre otras cosas para realizar cualquier actividad motora. Sin embargo, esta capacidad innata del ser humano para interaccionar con su medio a través del tacto no se aprovecha en una de las tareas más frecuentes de la actualidad, el uso del ordenador.

De las tareas que se realizan con un ordenador, hay algunas, como el modelado de objetos 3D, que además se ven afectadas por otro tipo de limitación. Esta limitación tiene que ver con la naturaleza 2D que tienen los dispositivos comunes, que hace que tengan que emplearse técnicas poco intuitivas y en general poco eficientes, como el uso de tres planos ortogonales.

En este marco, se ha desarrollado una aplicación de modelado de objetos 3D y un guante de datos con retorno táctil, y se han integrado en un sistema de realidad virtual formado por una pantalla de retroproyección, capaz de mostrar imágenes estereoscópicas, y de un sistema de localización espacial, gracias al cual se pueden seguir los movimientos de la mano del usuario.

El guante de datos se ha construido a partir de componentes que se pueden encontrar fácilmente a un precio asequible, y se presenta como una plataforma de experimentación en retorno táctil para aplicaciones de realidad virtual. Entre sus características destacan ser capaz de proporcionar una respuesta táctil independiente para cada dedo, y detectar el contacto entre dos o más dedos.

En cuanto a la aplicación, permite trabajar con objetos formados por una malla de polígonos, permitiendo su modelado a base de estirar de su superficie o de hacerle presión con los dedos. Además, proporciona varios tipos de retorno hacia el usuario, como el retorno táctil, sonoro, y visual, que contribuyen a una experiencia más rica e inmersiva.

Así, los usuarios del sistema construido han sido capaces de ver objetos virtuales "como si salieran de la pantalla", sentir su superficie a través del tacto y del oído, y manipularlo, modelándolo con su mano de una forma intuitiva como si se tratara de plastilina.

Agradecimientos

En primer lugar me gustaría dar las gracias a mi director de proyecto, José Pascual Molina, por toda su ayuda y apoyo recibido, así como por despertar en mí el interés por la Realidad Virtual.

También me gustaría dar las gracias a mis compañeros de clase, por hacer amenas tantas y tantas horas a lo largo de estos años, y del laboratorio, por su ayuda para resolver cada uno de los problemas técnicos encontrados.

Por último me gustaría agradecer el apoyo incondicional de mis amigos y amigas, que siempre estuvieron ahí en los buenos y malos momentos.

Muchas gracias a todos.

Dedicatorias

Este trabajo va dedicado con mucho cariño a mi familia, por apoyarme y darme ánimos en todo momento.

Índice

LISTA DE SÍMBOLOS

Símbolo	Significado	Apartado
mA	Miliamperios. Milésima parte del amperio, unidad de intensidad de corriente eléctrica en el Sistema Internacional de Unidades.	4.3.3 4.5.2 4.5.3
V	Voltio. Unidad de diferencia de potencial eléctrico en el Sistema Internacional de Unidades.	4.5.2 4.5.3
Ohm Ω	Ohmio. Unidad de resistencia eléctrica en el Sistema Internacional de Unidades.	4.5.3
N	Newton. Unidad de fuerza en el Sistema Internacional de Unidades.	2.3

ÍNDICE DE FIGURAS

ÍNDICE DE LISTADOS

CAPÍTULO 1. INTRODUCCIÓN

En este capítulo se realizará una breve presentación del proyecto, describiendo cuál ha sido la motivación que ha llevado a su desarrollo, los objetivos propuestos, y el plan de trabajo que se ha llevado a cabo para cumplirlos. Por último se hará un resumen del contenido y estructuración de la presente memoria.

1.1 MOTIVACIÓN

Durante nuestra vida cotidiana extraemos gran cantidad de información del entorno que nos rodea mediante el sentido del tacto. Esta información se almacena en forma de modelos mentales, que son usados a la hora de realizar cualquier actividad motora. Así, somos capaces de ajustar de antemano la fuerza que aplicamos al coger un objeto, pero también de reajustar esa presión si se percibe un peso no esperado, o una superficie resbaladiza.

No obstante, esta capacidad innata del ser humano para interaccionar con su medio a través del tacto no se aprovecha en una de las tareas más frecuentes de la actualidad, el uso del ordenador.

La forma que tenemos de interactuar con los ordenadores, estudiada por la disciplina IPO (Interfaz Persona-Ordenador), ha ido cambiando desde los orígenes de la informática. Sin embargo desde la aparición del PC y el paradigma WIMP (Windows, Icons, Menus and Pointers) a principios de los años 80, estos cambios se redujeron principalmente al software, estando los dispositivos basados en la misma pantalla, ratón y teclado que eran ya comunes por aquellos entonces.

Y es que la mayoría de los periféricos usados actualmente para interactuar con los ordenadores pueden ser programados para mostrar cualquier tipo de imágenes o de sonidos, pero no para transmitir diferentes sensaciones táctiles o de esfuerzo, lo que en general se llama retorno háptico. Esto hace que las acciones que se realizan estén basadas únicamente en un retorno visual, perdiendo así destreza y agilidad en su ejecución.

De las tareas que se realizan con un ordenador, hay algunas, como el modelado de objetos 3D, que además se ven afectadas por otro tipo de limitación. Ésta tiene que ver con el uso de las tradicionales interfaces 2D, que resultan poco idóneas para su utilización en entornos tridimensionales. Por ejemplo la pantalla muestra representaciones bidimensionales en las que no se aprovecha nuestra visión binocular, perdiendo la sensación de profundidad. De la misma manera el ratón sólo nos permite hacer

desplazamientos sobre un plano, por lo que en un espacio 3D es necesario usar tres planos ortogonales.

Estas dos limitaciones son las que sirven de motivación para la creación de una plataforma que permita modelar objetos 3D de una forma intuitiva, empleando dispositivos que aprovechen la capacidad del ser humano para percibir y trabajar en el espacio, y la vital información que aporta el sentido del tacto.

1.2 OBJETIVOS

El primer objetivo que se debe cumplir es la construcción de una aplicación inmersiva que permita el modelado de objetos virtuales de forma intuitiva. Este objetivo tendrá a su vez varios subobjetivos que serán:

- La aplicación debe basarse en un sistema de tipo proyectivo, esto es, que muestra las imágenes en una pantalla de proyección, y con gráficos estéreo, para lo que se contará con una pareja de proyectores con filtros polarizados y una pantalla de retroproyección.

- La aplicación debe ser capaz de seguir tanto el punto de vista del usuario, para mostrar los gráficos estéreo, como el movimiento de la mano con la que realice el modelado. Para ello será necesario utilizar un sistema de localización, acoplando un sensor a las gafas polarizadas, y otro más al dorso de la mano del usuario.

- Se debe proporcionar un soporte en la aplicación para la utilización de un dispositivo de retorno táctil, para que así el usuario pueda sentir cuándo ha tocado una superficie con su mano.

- Aunque no es un objetivo prioritario, al tratarse de una aplicación de modelado es recomendable permitir a los usuarios guardar los objetos virtuales modificados, así como cargarlos más tarde.

El segundo objetivo principal de este proyecto es la construcción de un guante de datos que sea capaz de proporcionar un retorno táctil. Como en el caso anterior, será necesario cubrir diversos subobjetivos:

- El retorno táctil debe ser independiente para cada dedo.

- El guante de datos se debe construir en forma de accesorios de quita y pon, de forma que se puedan adherir a cualquier otro guante para multiplicar sus posibilidades.

- Sería deseable contar también con algún mecanismo de entrada de información, como por ejemplo, el contacto entre dos o más dedos.

- Se debe construir además a partir de dispositivos electrónicos que se puedan encontrar fácilmente y a un precio asequible.

1.3 PLAN DE TRABAJO SEGUIDO

El desarrollo de este proyecto comenzó con la lectura de documentación relacionada con el mismo, principalmente en forma de artículos. También se busco información específica relacionada con las tecnologías disponibles para la construcción del guante de datos.

El siguiente paso consistió en elaborar un primer prototipo del guante de datos, así como de su testeo mediante la escritura directa de comandos en el puerto paralelo.

De forma paralela se hizo el diseño a alto nivel de la arquitectura de la aplicación, estudiando aplicaciones de ejemplo para aprender a usar tecnologías como VR Juggler y OpenGL. A partir de estos ejemplos la construcción de la aplicación fue siguiendo un modelo de prototipado rápido, creando versiones funcionales que cada vez integraban más funcionalidad.

En estas primeras versiones la aplicación era probada mediante el teclado y el ratón. Una vez implementado el driver del guante de datos, y estando la aplicación en una versión avanzada de desarrollo, se procedió a probarla junto al hardware específico de Realidad Virtual.

Mientras la aplicación era refinada, solucionando los problemas que aparecían y añadiendo nuevas características, se fue construyendo la versión definitiva del guante háptico.

Por último, una vez finalizado tanto el desarrollo tanto del guante, como de la aplicación, se procedió a hacer algunas pruebas con usuarios, para capturar así sus impresiones sobre el sistema implementado.

1.4 ESTRUCTURA DE LA MEMORIA

Los contenidos de la memoria se han estructurado en 7 capítulos y 3 anexos. A continuación se describe brevemente qué se puede encontrar en cada uno de ellos.

1. **Introducción.** Es el presente capítulo. Se trata de situar al lector en un contexto que describa la problemática que se pretende resolver y orientarlo en el resto de capítulos, para que sepa qué puede esperar de este proyecto.

2. **Realidad Virtual y el sentido del tacto.** Aquí se describirá de qué manera puede influir el sentido del tacto en aplicaciones de Realidad Virtual, y se hará un repaso por el estado del arte en cuanto a la tecnología que hace posible la transmisión de información a través de este sentido.

3. **Análisis del sistema informático.** En este capítulo se describirán las características con las que debe contar el sistema inmersivo que se desea construir, y finalmente se hará una elección de la tecnología a usar para el mismo.

4. **Diseño y construcción de un guante háptico.** Durante este capítulo se aborda el diseño y el desarrollo de un guante de datos capaz de proporcionar retorno táctil, así como del driver necesario para que pueda ser usado en una aplicación de Realidad Virtual.

5. **Diseño e implementación de la aplicación.** Se realizará en primer lugar un diseño tanto de la organización física del hardware como de la arquitectura de la aplicación de modelado. Finalmente se abordarán los detalles de implementación más importantes y se describirá el funcionamiento de los algoritmos usados.

6. **Puesta en marcha del sistema.** Aquí se describirán los pasos necesarios para comenzar a usar el sistema inmersivo creado, y se comentarán los problemas que han aparecido durante las pruebas y las impresiones de los usuarios que lo han probado.

7. **Conclusiones y trabajo futuro.** En este capítulo final se hará un resumen del trabajo realizado, repasando cada uno de los objetivos propuestos al inicio del proyecto, y extrayendo unas conclusiones al respecto. Para terminar se describirán posibles ampliaciones del trabajo desarrollado, que pueden servir como motivación para trabajos futuros.

8. **Anexos.** Se completará este trabajo con tres anexos que hacen hincapié en aspectos útiles que no se han desarrollado en el resto de capítulos. El primero de ellos está dedicado a la plataforma de implementación VR Juggler, el segundo trata de la configuración y calibración del sistema de retroproyección estereoscópico, y finalmente en el último se comentan aspectos de la configuración y uso del sistema de localización Flock of Birds.

CAPÍTULO 2. REALIDAD VIRTUAL Y EL SENTIDO DEL TACTO

En este capítulo se describirá en qué consiste la Realidad Virtual, y de qué manera puede el sentido del tacto ayudar en sus aplicaciones. Por último se realizará un repaso de las tecnologías que se pueden encontrar actualmente para transmitir información al usuario a través de este sentido.

2.1 DEFINICIÓN DE REALIDAD VIRTUAL

La Realidad Virtual es la simulación por ordenador de un entorno sintético, de tal forma que permita al usuario interactuar con él, y le cree una impresión de estar frente a un mundo físico mediante el engaño de sus sentidos.

Se distinguen dos tipos de Realidad Virtual: inmersiva y no inmersiva.

La Realidad Virtual **no inmersiva** hace uso de medios comunes, como internet, para permitir al usuario su interacción en tiempo real con diferentes personas en espacios virtuales. Además suelen emplearse periféricos como el teclado y el ratón, por lo que tiene un bajo coste y una rápida aceptación por parte de los usuarios. Un ejemplo muy popular es *Second Life*, un mundo virtual distribuido sobre una amplia red de servidores, al cual acceden por internet más de 30.000 personas simultáneamente [SecondLife].

Figura 2.1. Second Life. Ejemplo de mundo virtual no inmersivo.

La Realidad Virtual **inmersiva** se aprovecha de dispositivos creados específicamente para estimular los sentidos del usuario, aislándolo del mundo real (Figura 2.2). Estos dispositivos pueden estar especializados en el ambiente en el que se trabaja. Por ejemplo, en un simulador de conducción se puede usar un habitáculo de automóvil con un volante y pedales y una o más pantallas a su alrededor.

La inmersión, o presencia, es un efecto psicofisiológico según el cual los usuarios sienten que están "dentro" del ambiente virtual, y es mayor cuantos más canales sensoriales se ven implicados [Bar95]. Intervienen también otros factores como la calidad

de las imágenes percibidas, el campo de visión que se cubre del usuario, y la correspondencia o sincronización entre todos ellos.

Figura 2.2. Immersion Haptic Workstation. Interfaz para Realidad Virtual inmersiva.

Los ambientes virtuales inmersivos tienen la ventaja de poder recrear espacios que de otra manera serían inaccesibles, con riesgo, o muy caros, y poder interactuar con ellos. Un ejemplo de ello es la recreación de ambientes para entrenamiento, visitas virtuales a lugares de otras épocas, o diseño de prototipos que aún no se han construido.

2.2 EL SENTIDO DEL TACTO

El tacto es uno de los cinco sentidos básicos que posee el ser humano, y entre ellos es el más descentralizado, ya que está presente por todo nuestro cuerpo. Está compuesto por células nerviosas especializadas, estando la mayoría de ellas en la piel y más concentradas en zonas como la yema de los dedos.

Los receptores son capaces de transmitir frio, calor, presión y dolor, pero considerando la información individual de cada uno de ellos es posible apreciar sensaciones más complejas. Por ejemplo, la percepción de una presión variable a lo largo del tiempo permite sentir vibraciones, y la percepción de presiones irregulares a lo largo de la superficie de la piel, las texturas.

A partir de la información recibida por estos sensores, el cerebro es capaz de interpretar la forma de un objeto, su dureza, si es suave, si está templado, etc. Un mayor entrenamiento permite incluso la identificación de las letras del sistema Braille, formadas por puntos en relieve.

El tacto es también el responsable de la propiocepción, que es la capacidad para conocer el movimiento y posición de las articulaciones. La propiocepción es vital para la realización de cualquier actividad, y mediante ella somos capaces de conocer características de nuestro ambiente como el peso de un objeto, o la densidad de un líquido.

Toda esta información que proporciona el tacto se utiliza diariamente en la gran mayoría de las tareas conscientes e inconscientes, como la manipulación de objetos, sin embargo en los entornos virtuales actuales pocas veces es aprovechado.

En la vida real es fácil imaginarse la pérdida de la vista o del oído, simplemente cerrando los ojos o mediante unos tapones, respectivamente. Sin embargo, es difícil imaginar cómo cambiaría la percepción de la realidad sin el sentido del tacto, menospreciándolo en la mayoría de los casos. En un artículo de G. Robles-De-La-Torre [Rob06] se analiza la vida de dos personas que por diferentes motivos han perdido la mayor parte de este sentido. Entre otros muchos inconvenientes para realizar diferentes tareas que implican una coordinación motora, se observaron los siguientes:

- Pérdida de la capacidad para sentir el movimiento y posición de los miembros.

- Gran pérdida de precisión y velocidad de movimiento.

- Elevado nivel de concentración necesario para tareas en las que se requiere cierto nivel de precisión, como escribir.

- Gran dificultad para aprender nuevas tareas motoras, volver a aprender aquellas perdidas, o usar la experiencia para guiar estos procesos.

- Pérdida de la capacidad de realizar movimientos inconscientes sencillos, como el lenguaje corporal.

Así, incluso la aparentemente trivial tarea de coger y manejar un objeto común, se convierte en todo un reto en el que la mayoría de las veces o se ejerce mucha presión, o se resbala entre los dedos.

Algunos de estos problemas también se pueden presentar en entornos virtuales que no proporcionen una buena realimentación háptica, acentuándose sobre todo en aquellos en los que se requiere cierta destreza y también en aquellos con muchos grados de libertad. Este es el caso del modelado de objetos, donde una percepción con los dedos por parte del usuario es muy importante para hacerse una idea de su localización y propiedades físicas.

En resumen, incluir algún mecanismo de retorno háptico en un sistema de Realidad Virtual supone, además de lograr una mayor inmersión del usuario, habilidad para realizar tareas con mayor velocidad y precisión.

2.3 TECNOLOGÍA DEL TACTO

La tecnología háptica es aquella que se encarga de desarrollar dispositivos que estimulen el sentido del tacto. Este tipo de dispositivos abarcan desde servomotores que se encargan de trasladar a un piloto las fuerzas que actúan sobre el avión, hasta un guante que transmite diferentes texturas de un objeto virtual.

A continuación se hace una visión del estado del arte en cuanto a las distintas tecnologías hápticas usadas en Realidad Virtual.

2.3.1 Dispositivos hápticos de sobremesa

Son dispositivos capaces de proporcionar sensación de tacto mediante el uso de fuerzas que se oponen al movimiento de la mano del usuario. Para ello usan un sistema de motores y ejes que además miden hasta 6 grados de libertad.

La principal ventaja que ofrecen estos dispositivos es que son utilizados como si fueran un lápiz o puntero, por lo que no es necesario ajustar nada al cuerpo del usuario.

Las características que se suelen tener en cuenta a la hora de elegir entre ellos son:

- Resolución, desde 0.02 a 0.004 mm.

- Fuerza máxima a la cual son capaces de oponerse, desde 3.3 a 37.5 N.

- Rigidez, o fuerza continua a lo largo del espacio que son capaces de ejercer. De 1 a 15.0 N/mm.

- Grados de libertad que son capaces de medir y de responder, 3 ó 6 grados.

- Espacio de trabajo, 160x120x70 a 1080x900x600 mm.

- Precio, desde 1.780 a 85.000 €

Sensable Phantom

Tiene forma de brazo articulado, y actualmente existen varios modelos, cada uno de los cuales ofrece distintas prestaciones. Todos ellos miden 6 grados de libertad, y son capaces de responder a 3 ó 6 de ellos (traslación, o traslación y rotación). El modelo más básico es *Phantom Omni*, le sigue el modelo *Phantom Desktop* y por último el modelo *Phantom Premium* en sus versiones 1.5 y 3.0.

Figura 2.3. Modelos Omni, Desktop y Premium 3.0 del dispositivo háptico Phantom.

Force Dimension Delta/Omega

Está formado por tres articulaciones principales montadas sobre tres ejes y unidas en un punto central que es el que mueve el usuario. Está disponible en versión *3-DOF Omega*, *3-DOF Delta*, y *6-DOF Delta*. La versión Omega ofrece menor espacio de trabajo, menor fuerza y mayor resolución que la versión Delta. Ésta última puede medir y responder a 3 ó 6 grados de libertad (DOF, Degrees Of Freedom).

Figura 2.4. Dispositivos Omega, 3-DOF Delta y 6-DOF Delta de Force Dimension.

Haption Virtuose

Estos dispositivos tienen forma de brazo articulado, al igual que los de Sensable. Poseen una unidad de procesamiento propia, lo que permite reducir la carga de CPU del ordenador anfitrión y proporcionar más estabilidad en el movimiento. Además tienen una construcción modular.

Su producto, llamado *Virtuose* está disponible en 3 versiones, *6D35-45, 3D15-25 y 6D40-40*. Estos modelos indican por sí mismos el número de grados de libertad en el que trabajan, la fuerza máxima a la que son capaces de oponerse, y el espacio de trabajo que ofrecen (radio de la esfera, en centímetros).

Figura 2.5. Dispositivos 6D35-45, 3D15-25 y 6D40-40 de Haption.

2.3.2 Guantes con retorno háptico

Muy pocos guantes de datos cuentan con esta funcionalidad, es decir, que ofrezcan una respuesta táctil o de fuerzas. Por el contrario, no es difícil encontrar mandos y palancas de juegos a precios económicos que ofrezcan retorno de fuerzas, como por ejemplo los que comercializa Logitech. Curiosamente, los mecanismos integrados en los periféricos de esa casa comercial son de Immersion, el fabricante del conocido guante de datos Cyberglove, especializado también en dispositivos de realimentación táctil y de fuerzas. Sin embargo, el precio de un guante Immersion con tecnología háptica, incluso en la versión más económica Cybertouch, supera en varios órdenes de magnitud el precio de un joystick o un gamepad. Esto complica la decisión del desarrollador, quien debe estar muy seguro de sus posibilidades al adquirirlo, y más si lo que desea es contar con una pareja.

Immersion CyberTouch

CyberTouch consiste en 6 pequeños estimuladores vibrotáctiles que se le añaden al guante CyberGlove, uno en cada dedo y uno más en la palma. Cada uno de ellos se puede programar individualmente para cambiar la intensidad de la vibración. Mediante todo el conjunto es posible generar desde sensaciones simples hasta complejos patrones de realimentación táctil.

Figura 2.6. Immersion CyberTouch.

Immersion CyberGrasp

Este producto consiste en un exoesqueleto externo que se une al guante *CyberGlove* para proporcionar un retorno de fuerzas resistivo a cada dedo. Para ello aplica fuerzas individuales mediante una red de tendones guiados por el exoesqueleto y se puede programar para evitar que los dedos del usuario atraviesen o aplasten un objeto virtual. Gracias a esta estructura externa las fuerzas se aplican de forma perpendicular a los dedos a través de todo su rango de movimiento.

La fuerza máxima que se puede aplicar a cada dedo es de 12 N, y pesa 450 gramos, lo que lo convierte en su peor inconveniente.

Figura 2.7. Immersion CyberGrasp.

Immersion CyberForce

Este sistema consiste en una articulación móvil que proporciona realimentación de fuerzas a todo el brazo, y además mide 6 grados de libertad. Está diseñado para trabajar junto con CyberGrasp, que a su vez se coloca sobre un guante CyberGlove y proporciona realimentación de fuerzas a cada dedo.

Su espacio de trabajo es de 300x300x300 mm aproximadamente y puede ejercer una fuerza máxima de 8.8 N.

Figura 2.8. Immersion CyberForce.

2.3.3 Software para dispositivos hápticos

Los dispositivos hápticos necesitan el desarrollo de aplicaciones específicas para ser aprovechados plenamente. A continuación se hará un repaso de las librerías más populares.

Sensable Ghost SDK

Ghost es el acrónimo de General Haptic Open Software Toolkit, y es un conjunto de herramientas en C++ que facilitan el desarrollo de aplicaciones que aprovechen el retorno de fuerzas en los dispositivos de la familia **Phantom**.

Su misión es hacer de motor de físicas, es decir, encargarse de todos los cálculos de fuerzas necesarias para simular la interacción con un objeto virtual. Así, el programador no necesita ocuparse de estos detalles a tan bajo nivel, y simplemente se encargará de hacer uso de los objetos que se le proporcionan en la librería, definiendo sus características y comportamiento.

Ghost no está ligado a ninguna librería gráfica en concreto, siendo popular su uso junto con OpenGL y Open Inventor.

Sensable OpenHaptics

OpenHaptics es un conjunto de herramientas de desarrollo que permite añadir retorno háptico a aplicaciones que utilicen OpenGL, y al igual que Ghost, sólo soporta la familia de dispositivos Phantom.

Está diseñada siguiendo el mismo patrón de la API de OpenGL, por lo que su uso es muy familiar. De la misma forma que se define la geometría, se introducen comandos adicionales para definir propiedades como la fricción y rigidez del objeto.

Por último, OpenHaptics se puede integrar junto con otras librerías, como motores de físicas y de detección de colisiones.

Immersion VirtualHand

Este conjunto de librerías multiplataforma permite el desarrollo de aplicaciones que se beneficien de la interacción natural mediante los guantes de datos, así como de las capacidades hápticas de éstos. Soporta los dispositivos CyberGlove, CyberTouch, CyberGrasp y CyberForce, así como los sistemas de localización Polhemus Fastrak y Ascension Flock of Birds.

Posee además utilidades para configurar y manejar los dispositivos que soporta, capacidad para trabajar en red (capturando datos de un guante conectado a otro ordenador, por ejemplo), y diferentes algoritmos para proporcionar interacción con objetos virtuales.

Reachin API

Reachin API es otra plataforma que permite el desarrollo de aplicaciones con retorno háptico. Soporta varios lenguajes como C++, Python y VRML.

Es independiente del hardware, por lo que una aplicación desarrollada con esta API se puede ejecutar con dispositivos de Sensable, Force Dimension, Quanser o Haption.

Además se encarga de sincronizar las respuestas hápticas con los gráficos y audio de las aplicaciones 3D.

2.4 TECNOLOGÍA DEL TACTO EN DISPOSITIVOS DE CONSUMO

Los dispositivos comentados en el apartado anterior están desarrollados específicamente para ofrecer una respuesta háptica a nivel profesional, pero tienen en común una gran desventaja, su altísimo precio.

Sin embargo la tecnología del tacto también está presente en otros productos comerciales mucho más comunes.

Por ejemplo, es habitual encontrar joysticks y gamepads con uno o más vibradores, o volantes con retorno de esfuerzo. Y es que la industria de los videojuegos hace tiempo que usa esta tecnología para enriquecer la experiencia del usuario. De esta manera es posible transmitir la sensación de que se están recibiendo daños en un juego de primera persona, que el coche se ha salido del asfalto, produciendo vibración debido a los baches, o que se circula a gran velocidad, endureciendo la dirección como ocurre en la vida real.

En cuanto a los periféricos más comunes, también existen ratones que vibran, de tal forma que se puede sentir la superficie de un botón, o el borde de una ventana. Esta información extra ayuda al usuario a la hora de guiar el puntero.

Novint Falcon

Este dispositivo es una adaptación al mercado doméstico de los productos hápticos profesionales. Se compone de tres brazos articulados que recuerdan al producto Delta de Force Dimension, proporcionando localización y retorno de fuerzas. Está orientado al mercado de los videojuegos, proporcionando retorno de esfuerzo para sentir el retroceso de un arma al dispararla, el peso de una pelota de baloncesto, o la inercia de un palo de golf.

Su espacio de trabajo es de 104x104x104 mm. y su fuerza máxima es de 8.9 N. Está disponible por unos 190 dólares.

Figura 2.9. Novint Falcon.

TN Forcewear Vest

TN Games es la empresa desarrolladora de este chaleco que hace uso de tecnología neumática, integrando 8 pequeñas celdas independientes (4 en la parte delante y 4 en la trasera) que se hinchan de aire para así ejercer presión sobre el cuerpo.

Mediante estas celdas se puede simular un gran rango de impactos y explosiones, transmitiendo el lugar y fuerza del impacto de una bala en un juego de combate en primera persona. También es posible simular la fuerza G en juegos de conducción y simuladores de vuelo, o bien los golpes en juegos de combate. Por supuesto, estas sensaciones son de pequeña intensidad, por lo que no es posible causar daño alguno al jugador.

Estará disponible en el mercado a partir de Noviembre de 2007.

Figura 2.10. Chaleco TN Forcewear Vest.

Gamepad Logitech Rumblepad 2

Los mandos de juego o gamepads son muy comunes, y cada vez más éste es tan solo una muestra de la gran variedad presente en el mercado.

En este caso la empresa Logitech ofrece un mando con dos vibradores, uno a cada lado del mando, que pueden ser activados independientemente para proporcionar respuestas diferentes según el juego. Cuenta además con tecnología inalámbrica con un radio de alcance de unos 9 metros, y puede encontrarse en el mercado por unos 45€.

Es interesante destacar que la tecnología empleada por Logitech es de la compañía Immersion, creadora de algunos de los dispositivos comentados anteriormente como los guantes CyberGrasp o CyberTouch.

Figura 2.11. Gamepad Logitech Rumblepad 2.

Volante Logitech Formula Force EX

Otro tipo de tecnología muy aplicada en el caso de los volantes para juegos es la de retorno de esfuerzo. Mediante ella, la dirección se vuelve más dura o más blanda según la velocidad de marcha del vehículo, y es posible notar los baches e irregularidades de la carretera.

Este es un volante de Logitech en el que se hace uso de ella, pero hay muchos otros, como Microsoft Sidewinder ó Thrustmaster RGT Force Feedback.

Figura 2.12. Volante Logitech Formula Force EX.

CAPÍTULO 3. ANÁLISIS DEL SISTEMA INFORMÁTICO

El objetivo de este capítulo es el análisis de los requisitos funcionales y no funcionales del sistema inmersivo que se desea construir, así como la elección de las tecnologías hardware y software que se van a usar para el mismo.

3.1 REQUISITOS DE LA APLICACIÓN

3.1.1 Requisitos funcionales

A continuación se detallarán los requisitos funcionales de la aplicación, es decir, la funcionalidad mínima con la que deberá contar, y que servirán de base para realizar su diseño e implementación.

El objetivo principal que se debe cumplir es el desarrollo de un entorno virtual inmersivo que permita el modelado de un objeto virtual mediante las manos, y además que proporcione una respuesta háptica útil para el usuario.

El objeto que se va a modelar se mostrará en una pantalla de retroproyección estereoscópica de tal forma que cause en el usuario la sensación de que "sale de la pantalla" y que es posible tocarlo.

Para conseguir un mayor realismo, el punto de vista del usuario podrá cambiar para observar el objeto desde distintas perspectivas. Este cambio se debe reflejar en su representación para dar la sensación al usuario de que realmente se encuentra frente a él.

La sensación de profundidad producida por las imágenes estereoscópicas puede no ser suficiente para trabajar de forma intuitiva, por lo que se deberán implementar mecanismos adicionales que ayuden al usuario a determinar cuál es la posición de su mano en el entorno tridimensional. Estos mecanismos consistirán en lo siguiente:

- Resaltar de forma visual el vértice o cara del modelo que esté más cercano al dedo del usuario. Además también sería interesante mostrar la distancia a la que se encuentra mediante un cambio de color, por ejemplo.

- Representar una mano virtual que refleje el estado y la posición de la mano del usuario en la escena. Ésta es una técnica intuitiva y ampliamente utilizada en Realidad Virtual [Bow01].

- Reproducir un sonido al tocar la superficie del objeto o al seleccionar una cara/vértice. Esto además puede enriquecer la sensación de inmersión del usuario.

- Aprovechar el sentido del tacto, transmitiendo una respuesta háptica independiente para cada dedo que indique cuando se ha tocado el objeto.

El objeto virtual tendrá unas características predefinidas de color, textura, y factor de elasticidad. Su comportamiento podrá ser de dos tipos: elástico o plástico. Un objeto plástico mantiene su forma después de realizar una deformación sobre él. Por el contrario, un objeto elástico recupera una vez que desaparece la fuerza que causa su deformación.

El modelo virtual puede ser predefinido, o bien se puede permitir al usuario cargarlo desde un archivo.

Una vez finalizado el modelado del objeto virtual se le debe permitir al usuario guardarlo para poder recuperarlo de nuevo más tarde.

En cuanto a la interacción que tendrá el usuario con respecto a la aplicación, se ha diseñado un diagrama de casos de uso que da una idea sobre las distintas posibilidades (Figura 3.1).

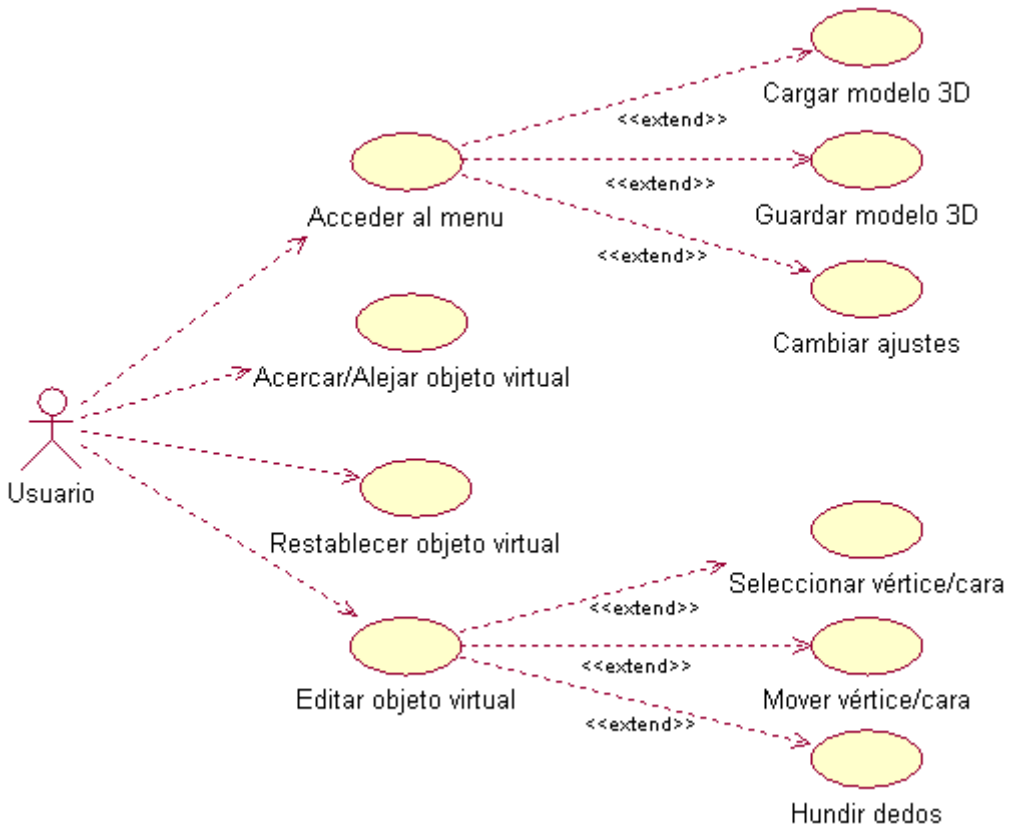


Figura 3.1. Diagrama de casos de uso.

Por una parte deberá ser posible acceder a un menú mediante el cual se pueda cargar o guardar el objeto virtual, y además activar o desactivar distintas opciones de configuración. Estas opciones serán:

- Retorno háptico en los dedos.
- Mostrar la mano virtual.
- Mostrar vértice/cara seleccionada.
- Mostrar vértices del objeto virtual.
- Mostrar aristas del objeto virtual.
- Elegir un comportamiento plástico o elástico del objeto virtual.
- Reproducir sonidos (realimentación sonora).
- Hundir dedos en los objetos.

Por otro lado, el usuario podrá interaccionar con el objeto virtual de las siguientes maneras.

En primer lugar, podrá seleccionar con su mano vértices o caras del modelo como si realmente las estuviera tocando. Una vez hecha la selección, podrá cambiar su posición en el espacio mediante un gesto de pinza con su dedo índice y pulgar, y arrastrando hacia un nuevo lugar.

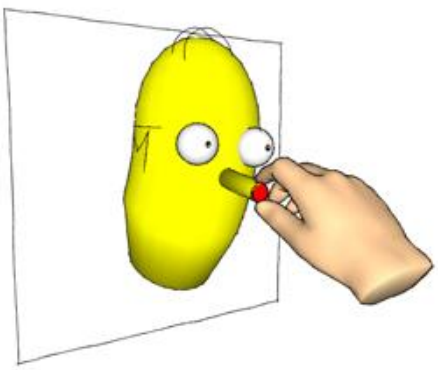


Figura 3.2. Manipulación del objeto 3D mediante una mano virtual.

Al arrastrar no sólo se cambiará de posición el vértice o cara seleccionada, sino que, tal y como ocurre en la realidad, se producirá una depresión o protuberancia local que afectará a su vecindad. La magnitud de ésta vendrá dada por su factor de elasticidad, y será permanente si el objeto tiene características plásticas. En caso contrario el objeto recuperará su forma una vez se libere.

En segundo lugar, el usuario tendrá la opción de tocar el objeto directamente con los dedos, y modelarlo como si fuera plastilina. La presión de los dedos sobre el objeto causará una deformación sobre su superficie que variará según su factor de elasticidad, y ésta será

permanente si el objeto tiene un comportamiento plástico. Como en el caso anterior, el objeto recuperará su forma al retirar la mano si tiene características elásticas.

Para la modificación del objeto virtual sólo se va a usar el gesto de la mano consistente en juntar los dedos índice y pulgar, quedando otros tres gestos libres (dedo pulgar junto con dedo corazón, anular o meñique). Éstos se pueden aprovechar para acercar el modelo hacia el usuario, alejarlo, y restablecerlo (cargarlo de nuevo desde el fichero), respectivamente.

3.1.2 Requisitos no funcionales

Los requisitos no funcionales detallan otros aspectos de la aplicación que se deben tener en cuenta en las siguientes fases.

Para dar una sensación de realismo e inmersión del usuario es necesario que el sistema sea fluido, por lo que el rendimiento del programa es crucial. Se deberán evitar los saltos que se producen cuando la velocidad de generación de imágenes es insuficiente.

En cuanto a la arquitectura, se deberá crear un diseño basado en módulos de software que puedan ser reutilizados por otras aplicaciones. Otro aspecto a tener en cuenta es que el diseño debe facilitar una posible ampliación futura de la aplicación con nuevas funcionalidades.

3.2 PLATAFORMA DE DESARROLLO SOFTWARE

A partir de los requisitos especificados en el apartado anterior se realizará un breve estudio sobre las diferentes tecnologías software disponibles para elegir aquella que mejor se ajuste a las necesidades del proyecto.

3.2.1 Herramienta de integración para Realidad Virtual

El desarrollo de la aplicación se puede realizar programándola directamente desde cero, apoyándose simplemente en una librería de generación de gráficos 3D, como OpenGL, y en los drivers de los distintos dispositivos que se vayan a usar. Esta solución tiene la ventaja de tener un control absoluto sobre los dispositivos, elementos de la escena, generación de gráficos 3D, cálculo de físicas, etc. En contra tiene la de producir una aplicación muy dependiente de la configuración hardware y software, además de ser muy

costosa en recursos, donde la mayoría de los esfuerzos se dedicarían al desarrollo de componentes que ya están hechos.

Otra opción es usar una de las distintas API de desarrollo orientadas a la Realidad Virtual. Estas herramientas se encargan de gestionar el sistema y de proporcionar una interfaz para los distintos dispositivos, descargando al programador de esta tarea. De esta manera el desarrollo se centra en la representación y funcionalidad de la aplicación.

Se tendrán en cuenta las dos herramientas más importantes de libre distribución orientadas a Realidad Virtual. Esto será garantía de una mayor portabilidad, facilidad de distribución y buen soporte, ya que existe una gran comunidad de desarrolladores implicados en su desarrollo.

Diverse

La primera API de las estudiadas es Diverse (Device Independent Virtual Environtment, Reconfigurable, Scalable, Extensible).

Es una API multiplataforma de código abierto compuesta de módulos de software, cuyo objetivo es permitir a los desarrolladores construir aplicaciones independientes de los dispositivos. Gracias a esto las aplicaciones desarrolladas se pueden ejecutar tanto en sistemas inmersivos como en configuraciones de escritorio. Actualmente sólo soporta Linux, estando el soporte para Windows XP y Mac OS X bajo desarrollo desde hace tiempo.

Diverse está orientada a la construcción de simuladores, y su diseño no depende de ningún paquete específico de gráficos, lo que es útil para simulaciones que no produzcan ninguna salida gráfica

Los dos paquetes que componen Diverse son:

- **The DIVERSE ToolKit (DTK)**. Es el paquete de utilidades que proporciona una plataforma de desarrollo para implementar aplicaciones distribuidas de simulación. Puede utilizarse de forma independiente o junto con otros paquetes.

- **DIVERSE graphics interface for Performer (DPF).** Es un paquete que añade gráficos 3D usando OpenGL Performer, que es una API gráfica de alto nivel basada en grafos de escena.

VR Juggler

VR Juggler proporciona una plataforma específica para el desarrollo de aplicaciones de Realidad Virtual, permitiendo su ejecución en casi cualquier sistema. Es una herramienta de código abierto, multiplaforma y escalable. Proporciona flexibilidad, ofreciendo un alto nivel de abstracción, pero también permitiendo a los desarrolladores trabajar a bajo nivel cuando es necesario.

Es compatible con gran cantidad de hardware de Realidad Virtual como por ejemplo: Cave, C2, HMD's, Ascension Flock of Birds, Logitech 3d mouse, BOOM, Immersion boxes, CyberGlove, Pinch Gloves, noDNA X-IST, Essential Reality P5 Glove, etc. Además es fácilmente extensible, ofreciendo una plataforma de desarrollo de drivers para nuevos dispositivos y gran cantidad de documentación asociada.

En cuanto al apartado gráfico, VR Juggler usa la API gráfica OpenGL, estando en desarrollo el soporte para Direct3D.

Opción escogida

Después de estudiar las distintas características de ambas plataformas parece más adecuado el uso de VR Juggler por los siguientes motivos:

- A pesar de que ambas plataformas se anuncian como multiplataforma, de momento Diverse sólo soporta Linux, mientras que VR Juggler soporta gran variedad de sistemas operativos.

- VR Juggler tiene mucha documentación asociada, tanto para el desarrollo de la aplicación como para el desarrollo de nuevos drivers para periféricos. Esto en principio lo hace idóneo para el posterior desarrollo del driver del guante háptico.

- Diverse está muy orientado a simulaciones en las que no necesariamente tiene que haber una salida gráfica, por lo que es necesario usar un paquete adicional para añadirle esta posibilidad. VR Juggler en cambio está totalmente orientado a sistemas de Realidad Virtual en las que hay una o más salidas gráficas, por lo que su integración con la API gráfica es total.

Además VR Juggler ya ha sido utilizada y probada en otros proyectos anteriores, por lo que tiene como ventaja añadida contar con la experiencia de otros compañeros que ya la han usado.

3.2.2 Grafos de escena

Las aplicaciones que trabajan con gráficos 3D necesitan tener almacenado en memoria información relativa a los objetos que componen una escena. Por ejemplo es común guardar sus vértices, aristas, caras, material que define su apariencia, relaciones entre los objetos, etc.

Las estructuras de datos que dan soporte a esta información son los llamados grafos de escena, y pueden ser desde sencillas tablas de vértices, hasta complejas jerarquías de objetos, dependiendo de las necesidades específicas de cada aplicación.

El uso de tablas es la forma más simple de implementar un grafo de escena. Su uso es adecuado cuando la aplicación sólo necesita acceder a los datos de forma lineal, por ejemplo para pintarlos, o cuando el número de objetos es muy pequeño.

Sin embargo la forma más común de implementar un grafo de escena es mediante un árbol compuesto por nodos grupo y nodos hoja. Los nodos grupo pueden tener asociados cualquier número de hijos y pueden incluir transformaciones geométricas. Los nodos hoja son los objetos que realmente forman parte de la escena.

En una aplicación de modelado los objetos que componen la escena cambiarán continuamente, y las relaciones entre ellos serán muy utilizadas. Por ejemplo en un algoritmo de deformación de mallas, un vértice deberá conocer cuáles son sus vecinos para calcular su posición.

Es por ello que considerarán algunas herramientas de libre distribución que implementan grafos de escena y que sean compatibles con VR Juggler.

OpenSceneGraph

OpenSceneGraph es una herramienta que implementa grafos de escena. Es multiplataforma, de código libre, y orientada a OpenGL.

Está diseñada para ser usada en aplicaciones de alto rendimiento, y actualmente tiene aplicación en campos como simulación visual, juegos, realidad virtual, visualización científica y modelado.

Sus características principales son:

- **Alto rendimiento**. Implementa un gran número de métodos para acelerar el proceso de generación de imágenes.

- **Librerías de nodos**. Estas librerías se pueden añadir al sistema para soportar sistemas de partículas, texto de alta calidad, efectos especiales, sombras, etc.

- **Cargador de escenas.** Esta característica es bastante común, y permite cargar escenas almacenadas en archivos. Algunos de los formatos soportados son: 3D Studio MAX (.3ds), Performer (.pfb), VRML (.wrl) y AC3D (.ac), entre otros. Una opción destacable es la posibilidad de leer datos a través de una red.

OpenSG

OpenSG es otra de las herramientas que se encargan de implementar un grafo de escena. Es de código libre, y está orientada a aplicaciones interactivas en tiempo real.

Su objetivo es proporcionar una capa de abstracción de los sistemas gráficos hardware y software, ya que API's gráficas como OpenGL están pensadas para trabajar a bajo nivel y su uso puede resultar complejo.

Las principales características diferenciadoras de OpenSG son:

- **Soporte de clustering**. Esto permite el uso de más de un PC trabajando conjuntamente para una misma aplicación, aportando más capacidad de proceso y salidas gráficas.

- **Procesamiento multihilo**. Cada vez es más común encontrar procesadores optimizados para trabajar con más de un hilo, procesadores con más de un núcleo o sistemas de multiprocesadores. El procesamiento multihilo aprovecha esta capacidad de cálculo.

- **Extensibilidad**. Las estructuras de datos están diseñadas para ser flexibles, de tal manera que pueden ser adaptadas para cada aplicación.

- **Cargador de escenas**. Al igual que OpenSceneGraph, OpenSG permite también cargar escenas desde archivos. Soporta formatos como 3D Studio (.3ds), VRML (.wrl) y Visualization Toolkit file (.vtk), entre muchos otros.

3D Toolbox

3D Toolbox es una implementación sencilla de un grafo de escena creada por *Pierre Alliez* e incluida en aplicaciones como "VRML viewer" y "OpenGL export" [Pierre].

Está construida específicamente para trabajar con OpenGL, y orientada al manejo de mallas de polígonos.

Sus principales características son:

- **Biblioteca de funciones geométricas**. Proporciona funciones geométricas y matemáticas muy útiles para la implementación de algoritmos como detección de colisiones o deformación de mallas. Algunas de estas funciones son: distancia de un punto a una cara, ángulo entre dos vectores o proyección de un punto sobre un plano.

- **Soporte de VRML.** Permite cargar mallas de polígonos a partir de archivos VRML, así como guardarlos posteriormente.

- **Extensibilidad.** Esta es tal vez una de las características más importantes de 3D Toolbox. Debido a que se trata de una implementación muy sencilla, comprender su estructura en profundidad requiere muy poco esfuerzo, y puede ser ampliada y adaptada fácilmente para adecuarla a los requisitos de cualquier aplicación.

Opción escogida

La opción escogida para implementar el grafo de la escena en la aplicación es 3D Toolbox. El principal motivo es que está orientada y optimizada específicamente para el tratamiento de mallas de polígonos, siendo el resto de alternativas de propósito general. Otro motivo importante es que se trata de la librería más sencilla, que es una ventaja tanto para su aprendizaje, como para su ampliación o adaptación.

3.3 DISPOSITIVOS HARDWARE

A continuación se realizará un repaso de los distintos dispositivos hardware presentes en el laboratorio de Realidad Virtual que pueden usarse para la aplicación. Finalmente se decidirá cuáles de ellos serán usados en base a sus características y a los requisitos estudiados anteriormente.

3.3.1 Guante de datos

Muchas veces los dispositivos comerciales no reúnen todas las características que uno busca. Es el caso de los guantes de datos. En el laboratorio hay un buen repertorio de ellos, cada uno con diferentes sensores, en número y tipo. Pero precisamente el hecho de que no todos tengan las mismas características, y que no siempre se cuente con la pareja de guantes izquierdo y derecho, lleva a una situación de compromiso que no siempre satisface.

Estos son los guantes de datos que se poseen actualmente y los sensores con los que cuentan:

- **5DT Dataglove 5** (sólo el derecho)
  5 sensores de flexión basados en fibra óptica.

- **EssentialReality P5** (sólo el derecho)
  5 sensores de flexión y sistema de localización óptico.

- **Immersion Cyberglove** (sólo el derecho)
  18 sensores de flexión y abducción.

- **Fakespace Pinch Gloves** (pareja de guantes)
  5 sensores de contacto, no detecta flexión ni abducción.

- **noDNA X-IST Dataglove SP1 system** (pareja de guantes)
  5 sensores de flexión y 5 de presión.

Como se puede ver, el tipo y número de sensores de cada uno de ellos es distinto y sólo uno de ellos cuenta con un sistema de localización. Además, ninguno de ellos proporciona realimentación hacia el usuario.

Es por ello que se plantea la construcción de un guante de datos con retorno háptico, o bien un accesorio que pueda ser acoplado a cualquiera de los guantes que se poseen. Este guante debe ser capaz de proporcionar una respuesta háptica variable e independiente para cada uno de los cinco dedos de la mano.

Dotar a este guante de capacidad para detectar la flexión de los dedos, o el contacto entre ellos es menos prioritario, ya que esto podría realizarse mediante cualquiera de los que ya se poseen.

En el caso de que por razones técnicas o de recursos este guante no dispusiera de ningún método de entrada de información, se usarían los guantes "Fakespace Pinch Gloves". Con estos guantes no se puede conocer el grado de flexión de cada dedo, pero haciendo uso de un sistema de localización sí se conocerá la posición y orientación del dorso de la mano. A partir de esta información, y suponiendo que la mano adopta un gesto de selección (señalando con el dedo índice), es posible calcular la posición aproximada de cada uno de los dedos. Estas posiciones servirán para conocer qué vértice o cara han sido seleccionados, así como para calcular el retorno háptico.

3.3.2 Sistema de localización

Para obtener la posición y orientación tanto de la mano como del usuario en la escena virtual, es necesario contar con un sistema de localización que proporcione 6 grados de libertad. En principio sólo son necesarias dos unidades receptoras (sensores): una de ellas colocada en la mano que se va a usar para modelar, y otra en la cabeza.

En el laboratorio contamos con el sistema Flock of Birds, que mide la posición y orientación de tres sensores mediante pulsos electromagnéticos (véase anexo C) y además es compatible con VR Juggler.

Figura 3.3. Sistema de localización Flock of Birds.

3.3.3 Dispositivo de visualización

Uno de los requisitos imprescindibles de la aplicación es la capacidad para mostrar un modelo tridimensional al usuario de tal forma que parezca que tiene profundidad. En Para ello es necesario usar un dispositivo de visualización que soporte estereoscopía.

Proview XL35

Se trata de un visiocasco que aísla al usuario de su entorno, mostrándole el mundo virtual como si estuviera frente a él. Es capaz de proporcionar una imagen diferente para cada ojo, por lo que se puede configurar para mostrar imágenes estereoscópicas.

La resolución máxima que puede mostrar es de 1024x768, y el campo de visión del usuario que cubre es de 21º en vertical, y 28º en horizontal. Su peso alcanza los 1020 gramos.

Al aislar al usuario de la realidad, no es capaz de ver su propia mano, por lo que sólo podría basarse en su "mano virtual" para interaccionar con la escena y la interacción sería menos natural.

Figura 3.4. Dispositivo de visualización Kaiser XL35.

Sistema de retroproyección estereoscópico

Está formado por dos cañones y una pantalla de retroproyección, y es capaz de mostrar imágenes estereoscópicas mediante el uso de luz polarizada (véase anexo B). El usuario necesitará llevar unas gafas con filtros polarizados muy ligeras.

Mediante el uso de este sistema el usuario tiene frente a él una ventana al mundo virtual, delante de la cual se podría hacer "flotar" los objetos de manera que diese la sensación al usuario de que los puede tocar con sus propias manos.

Además no se restringe a un único usuario, ya que la pantalla de retroproyección es grande y se pueden repartir gafas polarizadas para varios usuarios.

Esta es la opción escogida, debido a que permite una interacción más natural con el objeto virtual, es menos cansado para la vista, y a que las gafas que debe llevar el usuario apenas pesan unos gramos.

Figura 3.5. Sistema de retroproyección estereoscópico.

CAPÍTULO 4. DISEÑO Y CONSTRUCCIÓN DE UN GUANTE HÁPTICO

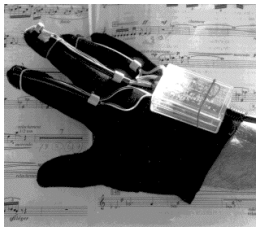
4.1 INTRODUCCIÓN

Como se comentó en capítulos anteriores, los dispositivos comerciales para Realidad Virtual no siempre satisfacen, ya que o bien no reúnen todas las características que nos interesan, o bien tienen un precio excesivo comparado con los periféricos de consumo habituales.

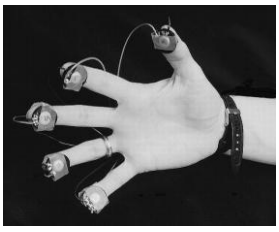
Con esta situación, no es difícil encontrar gente que se las ingenia para crear dispositivos novedosos, dispositivos que reúnen en uno las bondades de otros, o simplemente dispositivos más económicos. En el caso concreto de los guantes de datos, es posible encontrar en internet autores que publican sus creaciones. A continuación se muestran algunos de ellos.

Kevin Mellot es un joven apasionado por la Realidad Virtual que ha realizado numerosos proyectos relacionados con estos dispositivos. En uno de ellos ha adaptado un viejo PowerGlove de 1989 creado por Mattel para añadirle realimentación táctil. En otro de sus proyectos ha usado el mismo guante para hacer una versión inalámbrica [Mellot].

Cameron Browning, conocido internacionalmente por haber recibido numerosos premios de diseño por parte de instituciones como "Advertising & Design Club of Canada", muestra en su página web el proceso de la creación de un guante diseñado para navegar entre información interconectada en tres dimensiones [Browning].



En este caso Bertrand Merlier es el creador de un guante de datos casero al que ha llamado "La harpe magique" para producir música mediante gestos con la mano. El guante es capaz de detectar la flexión de los dedos y la inclinación de la mano, de tal manera que el usuario puede reproducir música en tiempo real [Merlier].



Pradeep Khosla junto con Roberta Klatzky han usado pequeños altavoces como estimuladores vibrotáctiles. Su trabajo se centra principalmente en averiguar cuál es la mejor forma de modular la vibración para transmitir información de forma efectiva al usuario [Kho95].

En cuanto al presente proyecto, ninguno de los guantes de datos del laboratorio de Realidad Virtual es capaz de ofrecer una respuesta háptica hacia el usuario, por lo que se ha decidido seguir los pasos de otros autores y crear uno propio. Este guante háptico se construirá a partir de componentes asequibles, con el propósito de utilizarlo como plataforma de experimentación en retorno táctil para aplicaciones de Realidad Virtual.

4.2 OBJETIVOS

El objetivo principal que se seguirá en la construcción de este guante de datos es que sea capaz de transmitir al usuario una respuesta háptica, es decir, que mediante algún tipo de actuador el usuario pueda sentir un objeto del mundo virtual a través del tacto.

Además sería deseable que el guante recoja información del usuario mediante algún tipo de sensor acoplado a cada uno de los dedos.

En términos de usabilidad se tendrá en cuenta la comodidad al vestirlo, así como también que el volumen y peso sean lo más ajustados posibles, ya que éstos afectan negativamente en la experiencia del usuario.

Por último, sería conveniente que el guante de datos esté formado por accesorios que puedan ser acoplados a cualquiera de los guantes ya existentes, multiplicando sus posibilidades.

Para todo ello se ha de tener en cuenta que hay que buscar soluciones sencillas que permitan abordar la construcción del guante en un tiempo limitado y con un coste que no sea excesivo.

4.3 ANÁLISIS DE TECNOLOGÍAS

4.3.1 Actuadores

Hay diferentes formas de transmitir al usuario una respuesta háptica mediante actuadores en un guante de datos. Según algunos autores como Burdea [Bur03] los interfaces hápticos se pueden dividir en tres categorías: retorno táctil, de fuerzas y propioceptivo.

Las interfaces de retorno táctil permiten transmitir datos como la rugosidad del objeto virtual, su temperatura, o su geometría, entre otros. Las que proporcionan un retorno de fuerzas aportan datos sobre el peso, inercia o elasticidad del objeto virtual. Por último, las interfaces con retorno propioceptivo informan al usuario sobre la posición de su cuerpo o postura.

La selección de una de ellas es clave, ya que cada una proporcionará información al usuario sobre un determinado campo. En este caso, se ha elegido el retorno táctil, debido a que es el que proporciona información más valiosa para la realización de trabajos minuciosos, como el modelado de un objeto virtual.

Las principales tecnologías de retorno táctil son:

- **Neumáticas.** Emplean pequeñas almohadillas inflables para simular presión sobre los objetos. Necesita un sistema de aire comprimido para funcionar, por lo que sería necesario contar con una bomba, con electroválvulas, así como de tuberías que conduzcan el aire.

- **Vibro-táctiles.** Utilizan bobinas de audio o pequeños motores eléctricos para provocar una vibración perceptible. Son actuadores baratos y fáciles de encontrar.

- **Tactores.** Son matrices de microvarillas que, al paso de una corriente, se doblan y presionan la punta del dedo del usuario. Se trata de actuadores muy específicos, y además de ser difíciles de encontrar, se necesitan muchas señales para cada tactor, por lo que su implementación se hace compleja.

- **Células de efecto Peltier.** Permiten transmitir sensación de calor o frio y suelen emplearse para transmitir señales de dolor, o de advertencia de peligro inminente. Se pueden encontrar fácilmente, pero su campo de aplicación es más reducido y es necesario tener cuidado para no lastimar al usuario, ya que pueden alcanzar una diferencia térmica de hasta 70 grados [Peltier].

Una vez contempladas las distintas posibilidades parece claro que la opción más factible es la del retorno vibro-táctil. Para ello se pueden emplear diferentes actuadores:

- **Mini-auriculares.** Una de las opciones a considerar es el acoplamiento de altavoces de pequeñas dimensiones (aproximadamente un centímetro de diámetro) en cada una de las yemas. Estos dispositivos están formados por una membrana acoplada a una bobina que es atraída por un imán dependiendo de la corriente que circule por ella. Se usan

normalmente para producir sonido, pero debido a su oscilación también se pueden emplear para producir una vibración perceptible. Para su funcionamiento necesitan circuitos osciladores específicos (no se pueden conectar directamente a una fuente de tensión).

- **Vibradores**. Otra forma más directa de producir una vibración perceptible es el empleo de vibradores. Consisten en un motor a cuyo eje se ha acoplado una masa descentrada, por lo que al girar produce una vibración proporcional a la velocidad de giro. Suelen tener unas dimensiones muy reducidas por lo que se pueden acoplar al guante fácilmente. Además tienen la ventaja de poder ser usados de forma directa conectándolos a una fuente de tensión.

De forma experimental se comprueba que la vibración producida por los mini-auriculares es mucho más baja en intensidad que la producida por los vibradores, por lo que acoplados a un guante apenas se perciben. Esto, unido a la mayor dificultad que entraña el uso de un circuito oscilador adicional, hace que se elija la opción de los motores vibradores como la más adecuada para este proyecto.

Los vibradores más fáciles de encontrar comercialmente son los que se utilizan habitualmente en los móviles. Los hay de dos tipos:

- **Planos**: Tienen forma de pila de botón con un diámetro aproximado de 1 cm. y están protegidos dentro de una carcasa metálica. Colocado plano sobre el dedo, su movimiento de vibración sería paralelo a la superficie de la piel.

- **Cilíndricos**: Tienen un diámetro de 3-4 mm. y una longitud de 15 mm. aproximadamente. No tienen carcasa exterior que los proteja, por lo que han de ser introducidos en un cilindro mayor para evitar que la masa roce al girar. La forma de colocarlo sería a lo largo de cada dedo, produciendo un movimiento de vibración perpendicular a la superficie de la piel.

Por su forma, el vibrador plano es adecuado para acoplarlo al dedo pulgar, mientras que los cilíndricos al ser más finos son más adecuados para el resto de los dedos.

En cuanto a su posición, se han considerado dos alternativas:

- Encima del dedo. Es la solución adoptada por la compañia Immersion en su producto CyberTouch (véase apartado 2.3.2), y la que se ha seguido para construir el primer prototipo del guante (Figura 4.1). Tiene la ventaja de no molestar en los

gestos, pero experimentalmente se comprueba que la sensación de vibración se siente más en el dedo en general y no en la yema.

- En la yema del dedo. Es la solución adoptada en la versión final (Figura 4.2). Se ha intentado reducir al máximo el grosor de los vibradores, limando la masa que tienen acoplada los vibradores cilíndricos para introducirlos en carcasas más pequeñas (Figura 4.3). La sensación de vibración percibida es mucho más localizada y, al haber reducido su tamaño, tampoco molestan en el uso del guante.

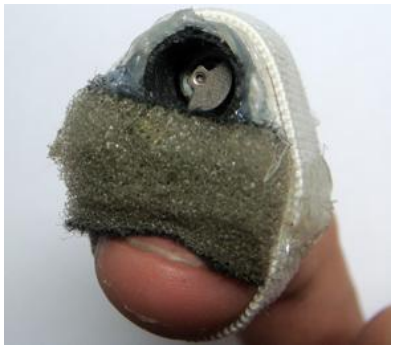


Figura 4.1. Colocación de los vibradores en el primer prototipo.

Figura 4.2. Colocación de los vibradores en la versión final.

Se han usado pequeñas bandas elásticas con Velcro en sus extremos para ajustar así los vibradores a los dedos de forma que se puedan quitar y poner de forma sencilla y rápida. Esto permitirá adaptarlos a cualquier otro guante de los disponibles en el laboratorio.

Debido a que los vibradores no cubren toda la superficie de la yema, su ajuste contra el dedo podría ser incómodo. Para evitar esto, se ha usado un delgado revestimiento de espuma (que se puede encontrar fácilmente en cualquier embalaje) con un hueco para el vibrador (Figura 4.4). De esta forma el grosor no aumenta y el vibrador se percibe de forma uniforme.

Figura 4.3. Reducción de la masa descentrada de los vibradores.

Figura 4.4. Vibrador encapsulado en una carcasa y revestido de espuma.

4.3.2 Captadores

El reconocimiento de gestos usando los dedos como contactos es un método muy útil para entrada de información, además de ser muy sencillo de implementar, ya que solamente requiere el uso de un material conductor en cada uno de los dedos. Como muchos de los guantes que miden la flexión y abducción de los dedos no cuentan con esta característica, se ha decidido incluirlo en el prototipo.

La detección se basará en el contacto o no del pulgar con el resto de los cuatro dedos. De esta forma el guante será capaz de transmitir hasta 2^4 posibles combinaciones.

El material más idóneo para realizar los contactos es tela conductiva, ya que es flexible y se adapta perfectamente a la forma de los dedos. Sin embargo, la dificultad para encontrarlo comercialmente hace que se escojan para el proyecto derivados más resistentes del papel de aluminio (Figura 4.2).

4.3.3 Interfaz con el ordenador

Una de las cuestiones más importantes a la hora de abordar el diseño de cualquier periférico, es la de decidir qué interfaz se va a utilizar para su comunicación con el ordenador.

La elección de una interfaz u otra repercutirá de forma determinante en la complejidad del sistema. Es por ello que se han barajado distintas alternativas que se comentan a continuación.

Puerto de juegos (game port)

Conexión tradicional para los dispositivos de control de videojuegos (Figura 4.5). De forma nativa sólo permite la entrada de datos (cuatro ejes analógicos y 4 botones digitales) por lo que descartamos esta opción antes de considerar otras características.

Figura 4.5. Conector del puerto de juegos en el PC.

Serie (RS-232)

Interfaz de comunicaciones que se caracteriza por transmitir la información bit a bit (Figura 4.6).

Ventajas:

- Cable de interconexión ligero. Al igual que el puerto USB, el cableado es pequeño ya que sólo se transmite un bit cada vez.
- Transmisión bidireccional.
- Driver sencillo de implementar. Existen numerosas librerías que simplifican el acceso al puerto serie para leer o escribir datos de él.

Desventajas:

- Baja velocidad. De todos los puertos estudiados es el que menor velocidad de transmisión de bits presenta.
- Lógica del controlador físico compleja. A pesar de que el circuito controlador es más sencillo y barato que el del puerto USB, sigue siendo necesario utilizar un chip microcontrolador que gestione la comunicación con el ordenador.
- No permite la conexión en caliente.
- No transmite energía eléctrica, por lo que sería necesario usar alimentación externa.

Figura 4.6. Conector del puerto RS-232 en el PC.

Paralelo

Interfaz cuya característica principal es la capacidad para transmitir uno o más bytes completos a la vez (Figura 4.7).

Ventajas:

- Proporciona mayor velocidad que el puerto serie, aunque menor que el puerto USB.
- Lógica de control sencilla. Al transmitir varios bits en paralelo, no se necesita un controlador especial para poder leer un byte.
- Driver sencillo de implementar. Al igual que para el caso del puerto serie, existen librerías predefinidas para la comunicación con cualquier dispositivo que se conecte al puerto paralelo.

Desventajas:

- Cable de interconexión pesado. La capacidad de transmitir varios bits en paralelo implica la utilización de más hilos que transporten la señal.
- No transmite energía eléctrica. Como en el caso del puerto serie, es necesario usar alimentación externa para el periférico.
- No permite la conexión en caliente.

Figura 4.7. Conector del puerto paralelo en el PC.

USB

Estándar muy común para la conexión de periféricos. Es el más reciente y gracias a sus características está desplazando al resto, imponiéndose en la mayoría de nuevos dispositivos comerciales.

Ventajas:

- Conexión en caliente, hace innecesario reiniciar el ordenador para conectar el dispositivo.
- Transmisión de energía eléctrica. Para dispositivos de hasta 500 mA de consumo, evita la utilización de fuentes de alimentación externas o uso de baterías.
- Alta velocidad. Es el que mayor caudal de bits proporciona de todos los estudiados.
- Cable de interconexión ligero. Al tratarse de una interfaz serie, se necesita poco cableado para transportar la señal.
- Transmisión bidireccional.
- Gran número de puertos. Actualmente todos los ordenadores cuentan con varios de ellos, siendo además sencillo ampliar su número mediante concentradores.

Desventajas:

- Lógica del controlador físico compleja. La circuitería electrónica necesaria es la más complicada, por lo que es más cara y su uso requiere un gran esfuerzo.
- Driver controlador complejo. Es necesario el desarrollo de un driver en modo kernel para que el sistema operativo sea capaz de reconocerlo.

Figura 4.8. Conectores USB en el PC.

Teniendo en cuenta las diferentes ventajas y desventajas de cada una de las interfaces, se ha considerado que la opción que mejor se ajusta a los objetivos del proyecto es el uso del **puerto paralelo**. La razón principal es que se trata de la interfaz que permite que la lógica de control sea más sencilla, permitiendo por ejemplo asignar un bit a cada uno de los 5 actuadores del guante sin necesidad de utilizar un circuito microcontrolador.

A pesar de que no se escoja el puerto USB como interfaz de datos, sí se utilizará para extraer la energía necesaria para el funcionamiento del guante, como se explicará posteriormente.

4.4 EL PUERTO PARALELO

Una vez elegido el puerto paralelo para la construcción del guante háptico, se describirán detalles útiles de su funcionamiento para el posterior diseño del circuito y del driver.

4.4.1 Características físicas

En una interfaz paralela cualquier comunicación se realiza mediante la transferencia simultánea de todos los bits que forman el dato.

Las líneas que componen el puerto paralelo son *latcheadas*, es decir, mantienen siempre el último valor establecido en ellas mientras no se cambien expresamente, y los niveles de tensión coinciden con los niveles de la lógica TTL. Estos valores son [PPort]:

- Tensión de referencia para el nivel alto: 5 voltios.
- Tensión de referencia para el nivel bajo: 0 voltios.
- Intensidad de salida máxima: 2.6 miliamperios.
- Intensidad de entrada máxima: 24 miliamperios.

El puerto paralelo está compuesto por 25 pines, de los cuales realmente solamente son necesarios 18 para transportar las señales de control y datos. Los hilos restantes son líneas de masa, que se enrollan alrededor de los cables de señal para proporcionarles apantallamiento y protección contra interferencias (Figura 4.9).

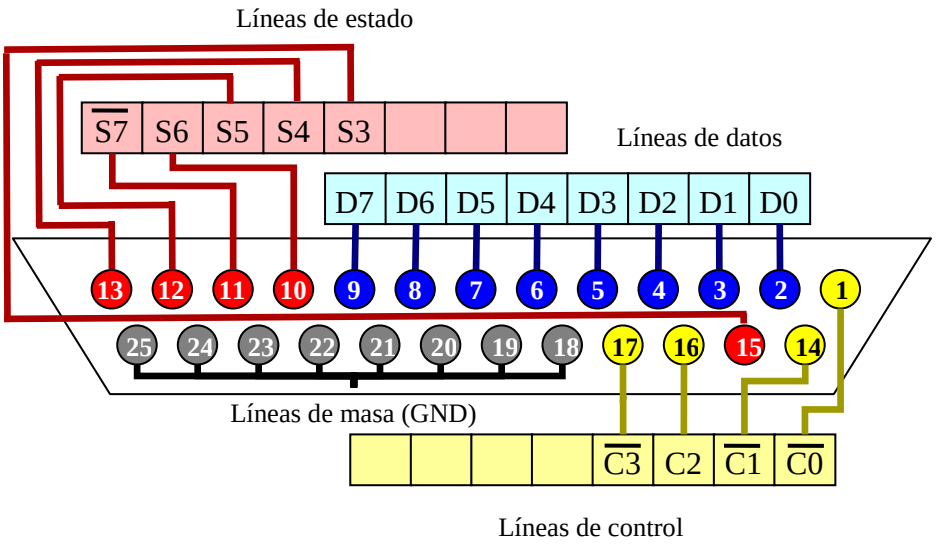


Figura 4.9. Esquema del conector DB-25 del puerto paralelo.

A continuación se describen todas las líneas que componen esta interfaz.

Los pines D0 a D7 se corresponden con las **líneas de datos**, que tradicionalmente son líneas de salida. Sin embargo en las implementaciones más recientes se permite que puedan funcionar de forma bidireccional, por lo que también podrían ser usadas como líneas de entrada.

Los pines S3 a S7 son **líneas de estado**, por lo que son usadas para entrada de datos (normalmente información de estado). Hay que tener en cuenta que el pin S7 está invertido (activo a nivel bajo de tensión).

Los pines C0 a C3 son **líneas de control**, que se usan para entrada y salida de datos, típicamente para mandar instrucciones de control. Los pines C0, C1, y C3 están invertidos.

Por último los pines 18 a 25 son líneas de masa.

El uso extendido del puerto paralelo para la conexión de impresoras hace que cada una de las líneas de control y estado tengan un significado especial (por ejemplo S5 indica que se ha quedado sin papel), pero para la interconexión del guante de datos esta información no es relevante.

4.4.2 Características lógicas

Los ordenadores suelen estar equipados con un puerto paralelo, pero físicamente son capaces de controlar hasta cuatro, que son llamados internamente como LPT1, LPT2, LPT3 y LPT4 respectivamente.

Cada una de las interfaces del puerto paralelo posee una dirección base. Tanto el número de puertos, como las direcciones base de cada uno de ellos suele mostrarse durante el inicio de la BIOS en el arranque del PC. Lo más común es que las direcciones base de los dos primeros puertos sean 0x378 y 0x278 respectivamente.

Para controlar cada una de las interfaces del puerto paralelo se usan tres registros:

- **Registro de datos:** Su dirección coincide con la dirección base del puerto paralelo (0x378 para LPT1). Puede ser escrito y leído por el procesador. Se corresponde con los pines físicos D0 a D7.

- **Registro de estado:** Es un registro de entrada de información. Su dirección se obtiene sumando 1 a la dirección base del puerto (0x379 en LPT1). Se corresponde con los pines físicos S3 a S7. Debido a que no existen los pines S0, S1 ni S2, los tres bits menos significativos de este registro no se usan.

- **Registro de control:** Es un registro de entrada/salida cuya dirección es el resultado de sumar dos a la dirección base del puerto. Físicamente los bits menos significativos se corresponden con los pines C0 a C3. El bit 4 permite controlar la generación de interrupciones desde el puerto paralelo. Los bits 5 a 7 no se usan.

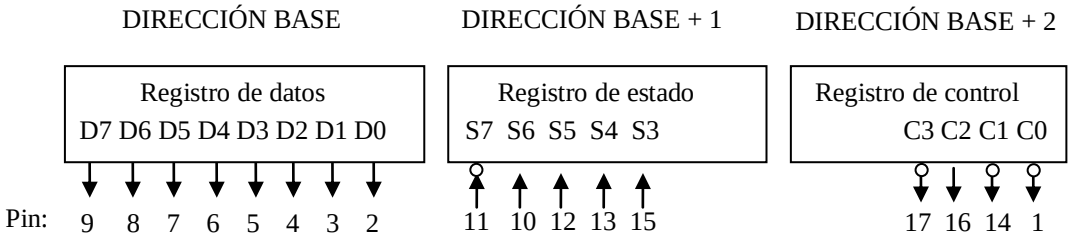


Figura 4.10. Registros del puerto paralelo y su correspondencia con las líneas físicas.

Los cables comerciales para la conexión de los periféricos con el puerto paralelo tienen una longitud entre 2 y 5 metros, que es suficiente para tener un rango de movilidad aceptable con el guante de datos. Una mayor distancia causaría más atenuación en la señal, y expondría al sistema a interferencias y errores en la comunicación.

4.5 DISEÑO Y CONSTRUCCIÓN DEL CIRCUITO

En este apartado se comentarán en primer lugar detalles generales que afectan al diseño del circuito controlador del guante háptico, como las líneas del puerto paralelo que se van a usar y qué tipo de alimentación es necesaria. Seguidamente se realizará un diseño del circuito justificando cada uno de sus detalles, para finalmente destacar algunos aspectos importantes de su construcción.

4.5.1 Distribución de las líneas de datos

Se usarán 5 de las 8 líneas de datos para establecer el estado de cada uno de los actuadores del guante. Esto en principio hace que sólo se pueda poner cada uno de ellos en estado encendido/apagado (sin regular la intensidad) pero como se verá más adelante esto se puede remediar mediante una *modulación por ancho de pulsos*.

Para la entrada de datos necesitamos 4 líneas, una para cada uno de los cuatro dedos que pueden hacer contacto con el pulgar. Se usarán las líneas de estado para este fin.

4.5.2 Alimentación

Como se comentó anteriormente, el puerto paralelo proporciona niveles de tensión TTL. Esto es: 5 voltios a nivel alto, y 0 voltios a nivel bajo. Los vibradores se alimentan a 3.6 voltios normalmente, por lo que en principio se podría pensar en conectarlos usando alguna resistencia de valor adecuado. Sin embargo hay que tener en cuenta que el puerto paralelo sólo es capaz de entregar 2.6 miliamperios de corriente frente a los 50-80 miliamperios de media que consume cada vibrador (medido experimentalmente).

Para poder entregar la corriente que necesita cada actuador es necesario usar una etapa amplificadora. Esto se consigue mediante una fuente de tensión independiente del puerto paralelo y el integrado *ULN2003* (Figura 4.11). Este chip se conecta directamente a las líneas de datos del puerto, y es capaz de proporcionar hasta 500 miliamperios para cada una de las salidas, a voltajes de hasta 50 voltios.

Figura 4.11. Chip integrado ULN2003.

Una vez decidido que se necesita usar una fuente de alimentación, hay que pensar en qué opciones hay. En principio parece que hay tres alternativas:

- Usar baterías en el guante. Esta solución presenta varios inconvenientes. En primer lugar las baterías son pesadas y ocupan un volumen considerable, por lo que podrían ocasionar fatiga e incomodidad a los usuarios. Por otra parte su vida es limitada, creando una necesidad de reemplazarlas o cargarlas cada cierto tiempo.

- Usar, además del cable paralelo de datos, otro cable de dos hilos conectado a una fuente de alimentación de 5 voltios.

- Reutilizar el cable de datos del puerto paralelo, aprovechando que tiene ocho líneas de masa redundantes. Esta es la solución adoptada, ya que evita la utilización de un nuevo cable. Sin embargo hay que tener en cuenta que hay que adaptar el conector que se enchufa al ordenador, desconectando la línea usada como alimentación positiva a +5 V.

El hilo que se va a usar como alimentación positiva será el correspondiente al pin 20 del conector hembra. Como masa se utilizará cualquiera de los 7 restantes.

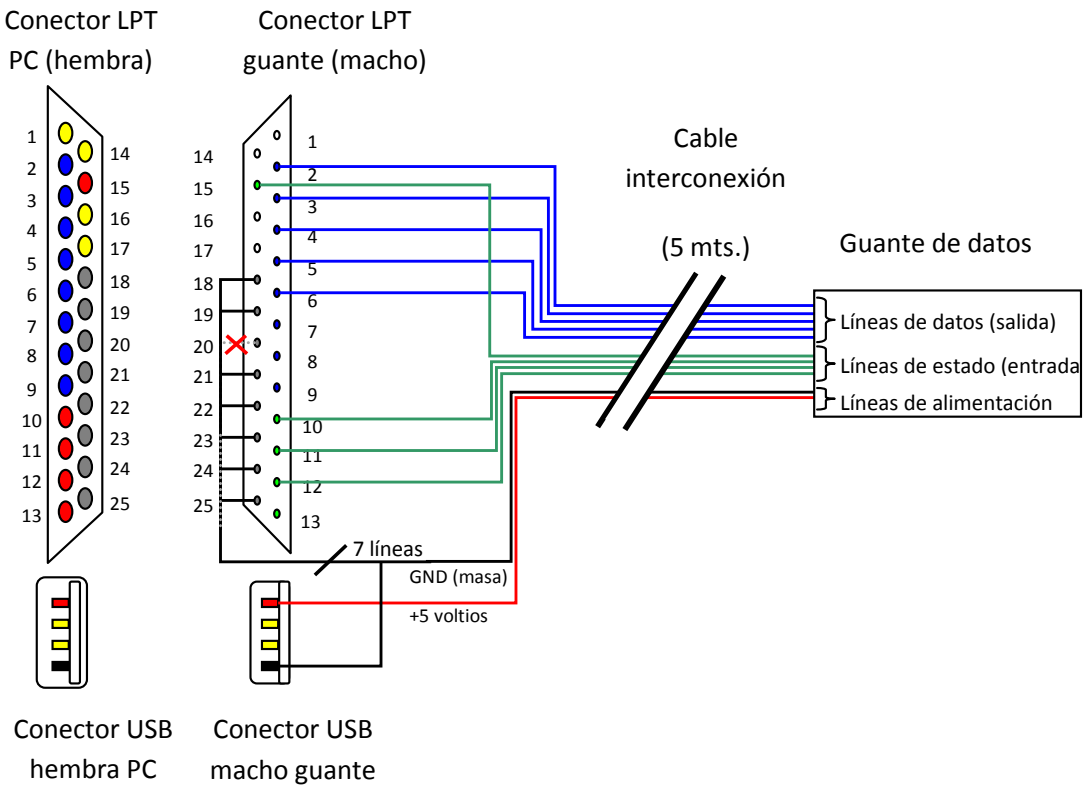


Figura 4.12. Esquema de las líneas de datos y alimentación del guante de datos.

Como fuente de alimentación se puede usar un transformador de corriente alterna (220 voltios AC a 5 voltios DC) o usar la alimentación del ordenador mediante el puerto USB.

El puerto USB proporciona 5 voltios y un máximo de 500 mA, más que suficiente para los 5 vibradores que como ya se comentó consumen unos 50-80 mA cada uno (en total 250-400 mA). Como además esta opción es más cómoda (ambos puertos suelen estar cerca), se va a aprovechar esta opción, por lo que se usará un conector USB macho al extremo del cable del puerto paralelo (Figura 4.12). De todas maneras, se deja abierta la opción de usar un transformador, simplemente usando como conector del mismo un USB hembra.

Con el fin de no modificar el conector del cable paralelo, y poder así utilizar cualquier prolongador, se ha construido un adaptador. Este adaptador desconecta el pin 20 y extrae las líneas de alimentación al conector USB (Figura 4.13).

Figura 4.13. Detalle del adaptador para alimentación.

4.5.3 Diseño del circuito

El circuito propuesto está basado en el chip ULN2003, que consiste básicamente en un array de 7 transistores Darlington acoplados a diodos de protección (Figura 4.14 y Figura 4.15). Su misión es la de entregar la corriente necesaria para activar cada uno de los actuadores (hasta 500 mA por canal) dependiendo del nivel lógico en el que se encuentre su entrada correspondiente. De esta manera se extrae la mínima cantidad de energía del puerto paralelo necesaria para activar un simple transistor.

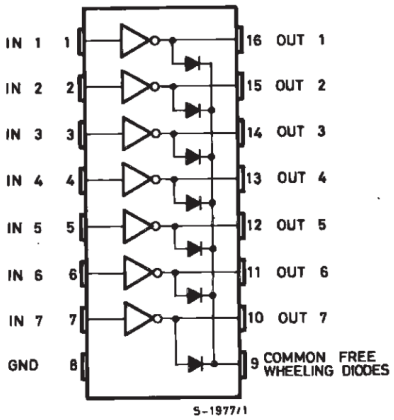


Figura 4.14. Esquema general del integrado ULN2003

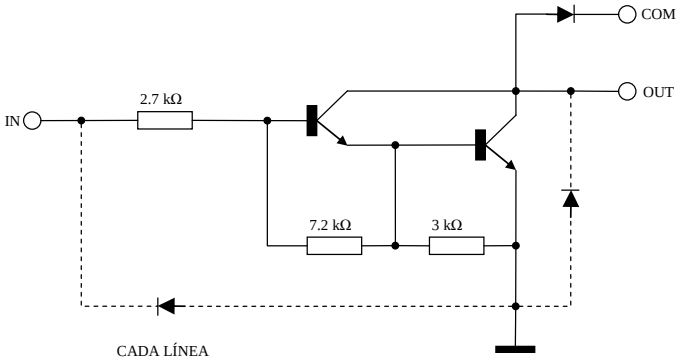


Figura 4.15. Esquema de cada una de las líneas del integrado ULN2003

Su uso está muy extendido para controlar las bobinas de los motores paso a paso, por lo que es fácil de encontrar en tiendas de electrónica a un precio muy asequible (en torno a 0.50 €).

Se conectarán las 5 líneas de datos utilizadas a las 5 primeras entradas del integrado. Las salidas amplificadas se deben conectar a cada uno de los actuadores intercalando resistencias de 20 ohmios para limitar su corriente. Además, se ha incluido una resistencia variable conectada en serie de 100 ohmios para poder realizar un ajuste individual de cada uno de los vibradores. De esta manera el ajuste de resistencia será entre 20 y 120 ohmios, y permitirá ajustar la intensidad de vibración tanto si es imperceptible como si es molesta.

Se han incluido unos diodos LED para monitorizar el estado de cada uno de los vibradores. Éstos se conectan de forma idéntica a los vibradores, por lo que por claridad no se muestran en el diagrama.

El diodo D1 se encarga de proteger el integrado contra corrientes inversas producidas por los vibradores. Debido a que éstos son de poco tamaño, su utilización es opcional.

Por último, se conectan cada uno de los cuatro contactos a las líneas de estado del puerto paralelo a través de una resistencia para limitar la corriente.

Para cerrar el circuito de cada contacto se usará el pulgar, que estará conectado directamente a masa.

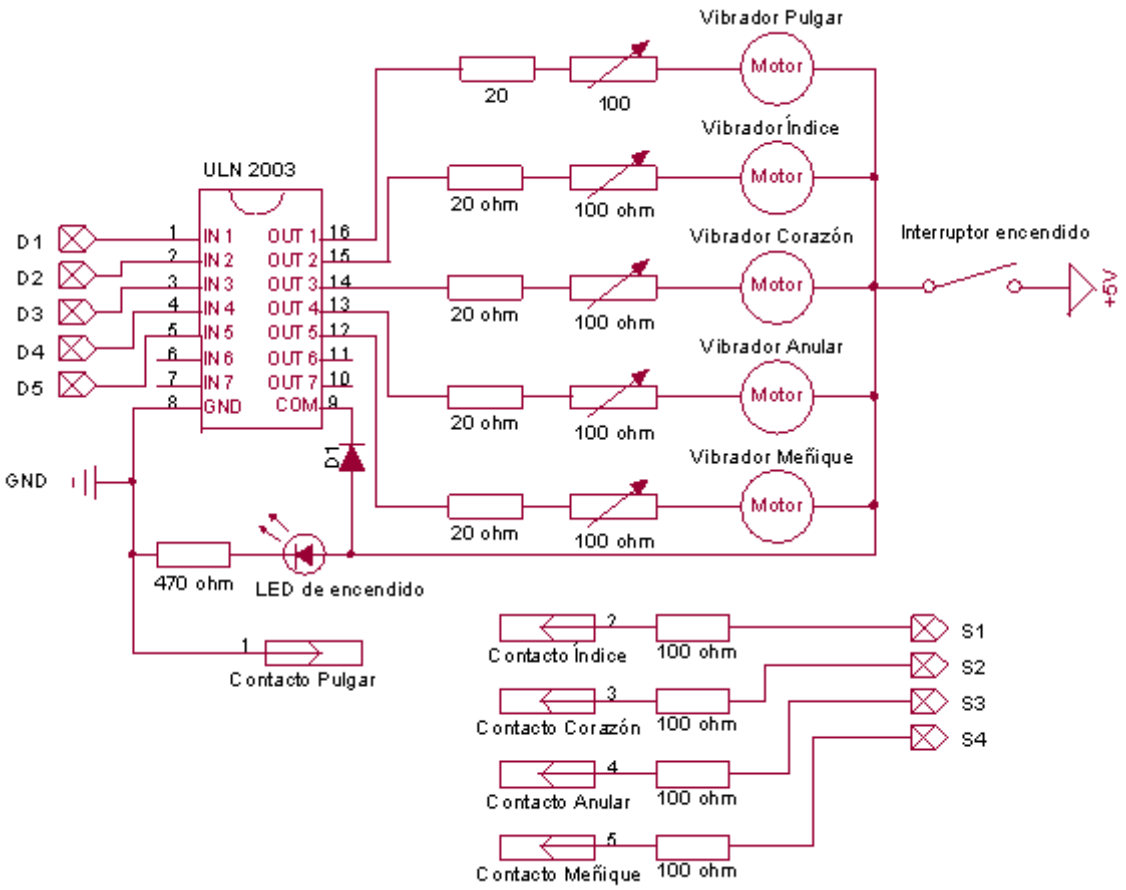


Figura 4.16. Circuito de control del guante de datos.

4.5.4 Construcción del circuito

En la construcción del circuito se han seguido dos fases, una primera fase de pruebas, en la que se han ajustado todos los componentes para que funcionara correctamente, y una fase final, en la que se ha dejado el circuito listo para su uso en la aplicación.

Para la fase de pruebas se ha utilizado una placa de prototipos (*protoboard*) para montar el circuito (Figura 4.17). En ella es fácil conectar los diferentes componentes para comprobar su correcto funcionamiento e introducir los cambios pertinentes.

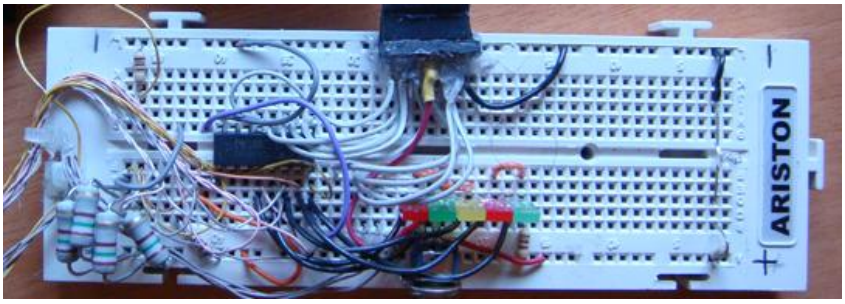


Figura 4.17. Circuito montado sobre una placa de prototipos.

Como se puede ver, en este montaje aún no estaban colocados los potenciómetros. Además, debido a la facilidad para experimentar que ofrece, se probaron soluciones como la introducción de un filtro paso-bajo mediante condensadores para "suavizar" la señal de entrada de cada uno de los vibradores y lograr un mejor resultado de la modulación por anchura de pulsos que se comentará posteriormente. Sin embargo, en las pruebas se podía apreciar un pequeño retardo de respuesta inherente a este tipo de filtros, así que finalmente se desestimó.

El guante usado en esta primera fase es un prototipo que tiene colocados los vibradores encima de los dedos sobre unas almohadillas.

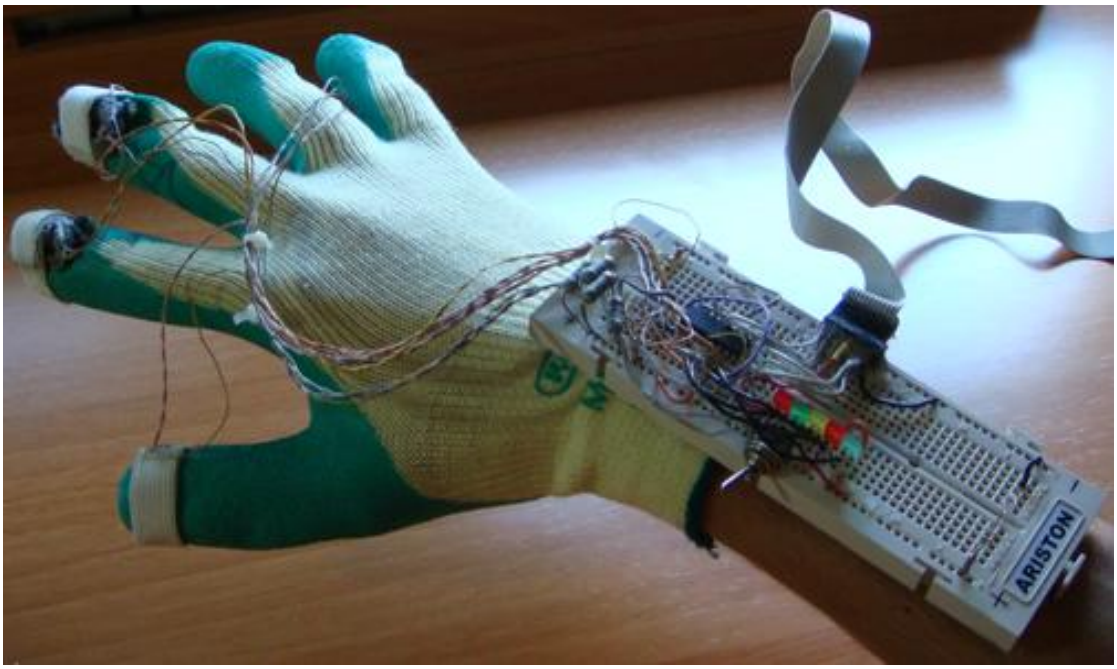


Figura 4.18. Prototipo del guante de retorno háptico.

Una vez refinado el circuito, se procedió a montarlo definitivamente sobre una placa PCB (*Printed Circuit Board*). Como es un circuito sencillo se ha usado una PCB preperforada con la que evitamos el proceso de insolación. De esta manera solamente será necesario soldar los componentes y crear las pistas mediante estaño o pequeños cables.

Finalmente se ha usado una pequeña caja de plástico que se ha recortado y adaptado al circuito. También se le ha añadido una tira de Velcro para poder ajustarlo rápidamente a la muñeca del usuario.

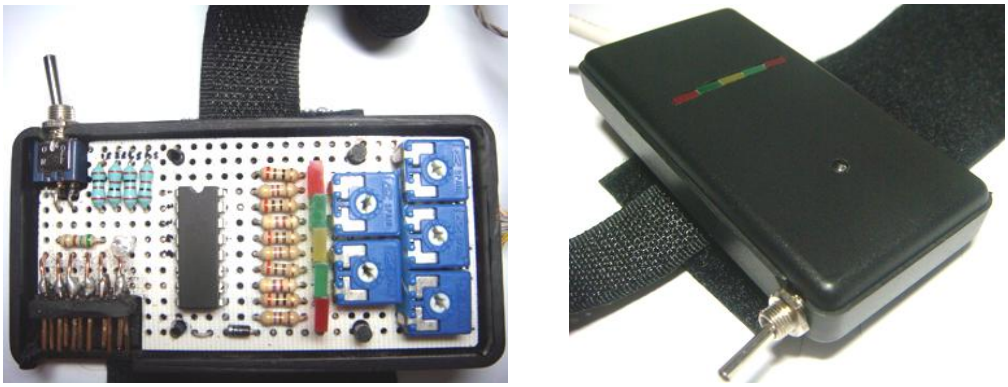


Figura 4.19. Versión final del circuito controlador del guante háptico.

Para esta nueva versión del circuito también se creó una nueva versión del guante, introduciendo las siguientes mejoras:

- Cambio en la posición de los vibradores, estando ahora en la yema de cada dedo.
- Las bandas elásticas que sujetan cada vibrador tienen ahora Velcro para poder realizar un mejor ajuste.
- Los cables están ahora mejor recogidos gracias a unas guías puestas en cada dedo y a unas fundas termo-retráctiles.
- Se ha ampliado la zona de contacto del dedo pulgar, ya que en el anterior prototipo resultaba insuficiente.
- Ahora se usa un tejido más fino y flexible, por lo que es más cómodo de llevar y da menos calor.

Figura 4.20. Versión final del guante háptico.

4.5.5 Lista de materiales usados

A continuación se hará un listado con los materiales usados para la construcción del guante de datos, así como un coste aproximado de los mismos.

Material	Unidad	Precio
1 Guante de lycra	1.60 €	1.60 €
5 Vibradores	1.00 €	5.00 €
Bandas elásticas	1.00 €	1.00 €
Tiras de Velcro	2.00 €	2.00 €
1 placa PCB perforada	5.00 €	5.00 €
1 placa de prototipos	10.00 €	10.00 €
1 chip ULN2003	0.50 €	0.50 €
5 LED baja luminosidad	0.30 €	1.50 €
1 LED alta luminosidad	0.60 €	0.60 €
6 Resistencias de 470 Ω	0.05 €	0.30 €
5 Resistencias de 20 Ω	0.05 €	0.25 €
4 Resistencias de 100 Ω	0.05 €	0.20 €
5 Potenciómetros	0.50 €	2.50 €
1 Interruptor	1.00 €	1.00 €
1 Caja de plástico	1.00 €	1.00 €
1 Diodo	0.10 €	0.10 €
1 Cable prolongador paralelo	3.00 €	3.00 €
2 Conectores puerto paralelo	0.50 €	1.00 €
Cables de interconexión	1.00 €	1.00 €
Fundas termo-retráctiles	0.50€	0.50 €
	Total:	38.05 €

Listado 4.1. Lista de materiales y coste aproximado.

Como se puede ver, el coste aproximado es de menos de 40 €, mientras que el precio de unos guantes comerciales Fakespace Pinch, con capacidad sólo para detectar el contacto entre dedos, es de más de 1.500 €.

4.6 DISEÑO E IMPLEMENTACIÓN DEL DRIVER

4.6.1 Introducción

Para poder hacer uso de cualquier dispositivo hardware en el ordenador, es necesario un módulo de software que realice la comunicación con el mismo mediante un puerto y un protocolo determinado.

El controlador del guante de datos construido es muy sencillo, por lo que el protocolo de comunicación consistirá únicamente en escribir datos en un registro para actualizar el estado de los vibradores, o bien leer datos de un registro para conocer el estado de los contactos.

En un principio solamente es posible establecer el estado de cada uno de los vibradores a encendido o apagado. Sin embargo, utilizando una modulación por anchura de pulsos (PWM, *Pulse Width Modulation*) se podría regular la potencia de forma sencilla.

PWM es una manera eficiente para los circuitos digitales de simular un rango de valores analógicos. Se basa en el principio de que mediante la conmutación rápida entre los valores 0 y Vcc se puede obtener como valor medio cualquiera comprendido entre ambos.

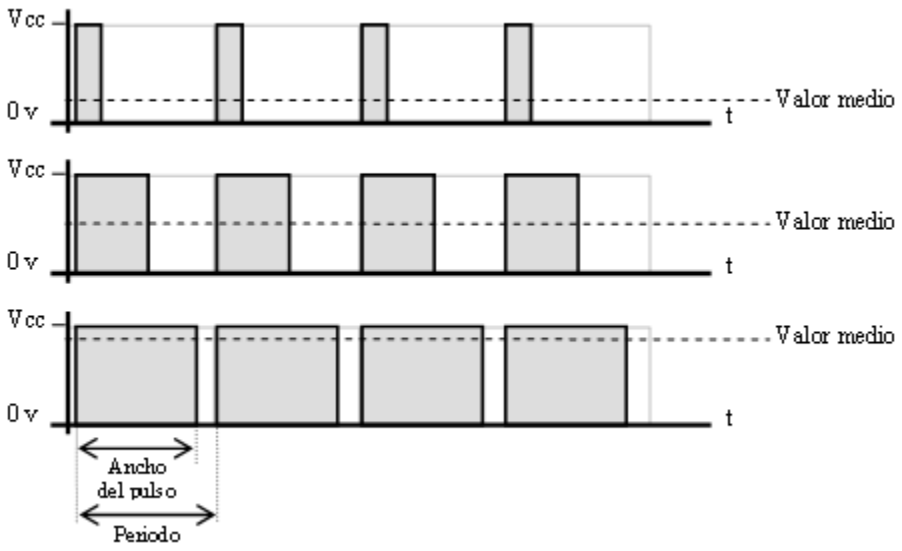


Figura 4.21. Utilización de PWM para ajustar el valor medio de una señal digital.

Como se puede ver en el esquema (Figura 4.21), según se aumenta el ancho del pulso positivo respecto al periodo, el valor medio de la señal crece.

En cuanto a la utilización de esta señal cuadrada por un motor, si tiene una frecuencia suficientemente alta no nota las variaciones (actúa como un filtro) y saca un giro constante proporcional al tiempo relativo que la señal está a nivel alto respecto al tiempo que está a nivel bajo.

Esta forma de ajustar la velocidad de un motor tiene la ventaja de mantener el par de giro (fuerza con la que se opone a quedarse parado), al contrario de lo que pasa cambiando la velocidad mediante ajuste de la tensión. De esta manera se evita en parte que el motor no arranque cuando está en reposo.

4.6.2 Plataforma

La plataforma ideal para la implementación del driver es *Gadgeteer,* que es el sistema gestor de dispositivos hardware multiplataforma usado por VR Juggler (véase anexo A). Esto permitiría integrar el guante de forma transparente usando una interfaz de alto nivel, y utilizar todas las ventajas de los ficheros de configuración para cambiar de dispositivos sin cambiar el programa.

El problema es que actualmente Gadgeteer no soporta dispositivos de salida (en particular, ninguno de los que ofrecen retorno táctil o de fuerzas), por lo que es necesario pensar en otra alternativa.

Gadgeteer usa para implementar sus drivers multiplataforma una librería llamada *VR Juggler Portable Runtime* (VPR) que proporciona abstracciones para hilos, sockets y primitivas de entrada/salida. Parece que una segunda opción podría ser hacer uso de ella sin implementar las interfaces impuestas por *Gadgeteer*. Sin embargo, consultando en profundidad la documentación asociada [VPR] se puede ver que no incluye soporte para el puerto paralelo, por lo que descartamos el uso de VPR.

La última alternativa consiste en programar el driver usando primitivas del sistema operativo. Bajo los sistemas operativos DOS, Windows 95/98 es posible escribir datos en el puerto paralelo de forma muy sencilla mediante las llamadas inportb/outportb (inb/outb bajo Linux). Sin embargo bajo Windows con núcleo NT (Windows XP, 2000 y NT4) esto no es posible, ya que éstas son instrucciones privilegiadas y no pueden ser lanzadas por un programa en modo de usuario (necesitan ser ejecutadas por un programa en modo kernel).

Crear un driver en modo kernel es una tarea compleja, y cualquier error asociado a él puede ocasionar graves problemas de estabilidad en el sistema operativo. Por ello es

mucho más recomendable hacer uso de una librería que proporcione las rutinas básicas necesarias para la comunicación. Una de las más conocidas es *inpout32* [Inpout32].

Inpout32 tiene las siguientes características:

- Funciona con todas las versiones de Windows.
- Usa un driver en modo kernel incluido en la dll.
- No se necesita instalación de software o del driver.
- El driver se instala y se configura automáticamente cuando se carga la dll.

4.6.3 Diseño e implementación

El diseño del driver estará orientado a conseguir una modulación independiente mediante PWM para el vibrador de cada dedo.

Esto supondrá la creación de cinco hilos de ejecución que deberán ser gestionados por una unidad de control que sirva de interfaz del driver.

Esta es la idea que se ha tenido en cuanta para realizar el diseño del diagrama de clases (Figura 4.22).

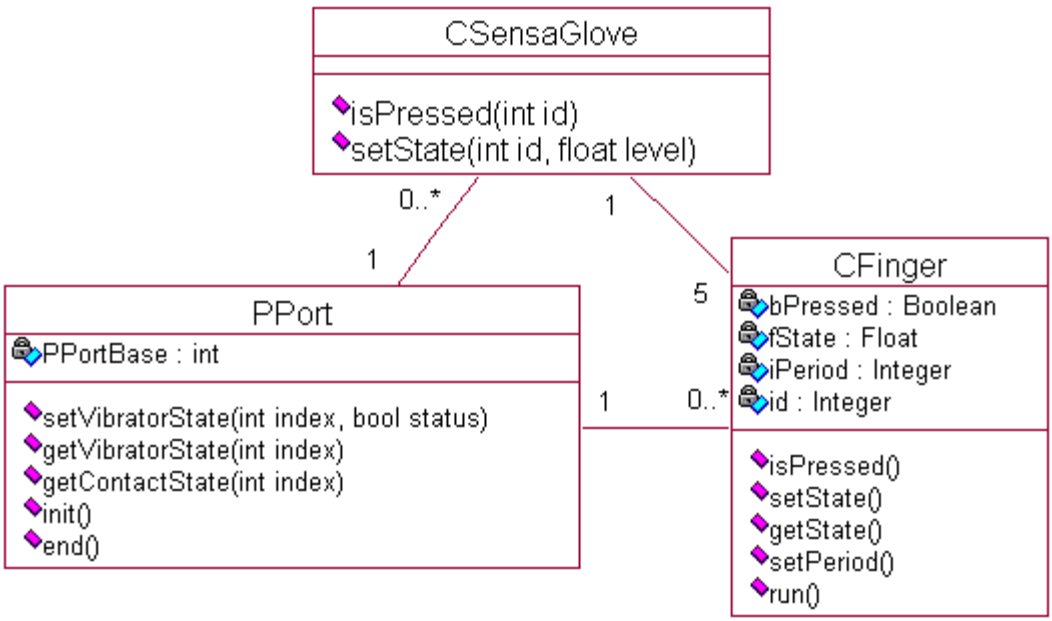


Figura 4.22. Diagrama de clases del driver.

A continuación se describe cual es la misión de cada una de ellas.

Clase PPort

PPort, se encarga de la comunicación con la librería en modo kernel para acceder al puerto paralelo.

Tiene como atributo de la clase la dirección base del puerto paralelo que se va a utilizar e implementa métodos para conocer o establecer el estado del los vibradores a nivel binario (*getVibratorState()* y *setvibratorState()* respectivamente), y para devolver el estado de los contactos (*getContactState()*).

Estas primitivas acceden al registro de datos o de estado del puerto paralelo a través de la librería Inpout32 y mediante operaciones a nivel de bit extraen la información necesaria.

Es el caso de la operación *getVibratorState* (Listado 4.2), que se sirve de una operación de desplazamiento y otra de *AND* para conocer el estado de un bit en el registro de datos.

```
getVibratorState(int id)
{
   //Capturar el valor del registro de datos
   Short sValue = Inp32(PPortBase);

   //Poner a cero todos los bits excepto el del vibrador
   sValue = (short)(sValue & (1 << index));

   //Devuelve uno si el bit del vibrador está a uno
   return sValue != 0;
}
```

Listado 4.2. Captura del estado del vibrador, método getVibratorState.

Clase Finger

La clase *Finger* se encarga de realizar la modulación PWM. Se creará una instancia para cada uno de los cinco dedos y proporciona los métodos necesarios para cambiar el estado de los vibradores (con un valor real entre 0.0 y 1.0) y para consultar el estado de su contacto asociado.

La modulación PWM se realiza mediante un bucle infinito que se ejecuta en un hilo propio. En él se envía la señal activado o desactivado a su vibrador, ajustando el tiempo que está la señal a nivel bajo y el tiempo en el que está a nivel alto (Figura 4.23).

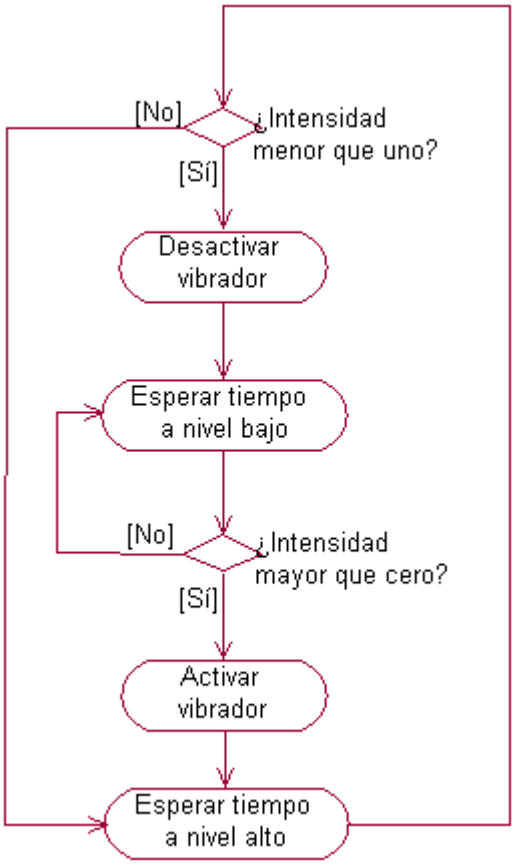


Figura 4.23. Diagrama de actividad del algoritmo PWM.

Como se puede ver en el diagrama de actividad, en el caso de que la intensidad de vibración sea máxima no desactivará el vibrador, y viceversa, cuando la intensidad sea cero no activará el vibrador.

El tiempo que debe estar el vibrador a nivel alto será ***fState * periodo***, y el tiempo que estará a nivel bajo será ***periodo – (fState * periodo)***, siendo *fState* la intensidad de vibración entre 0.0 y 1.0, y *periodo* el intervalo de tiempo necesario para completar un ciclo (Figura 4.24).

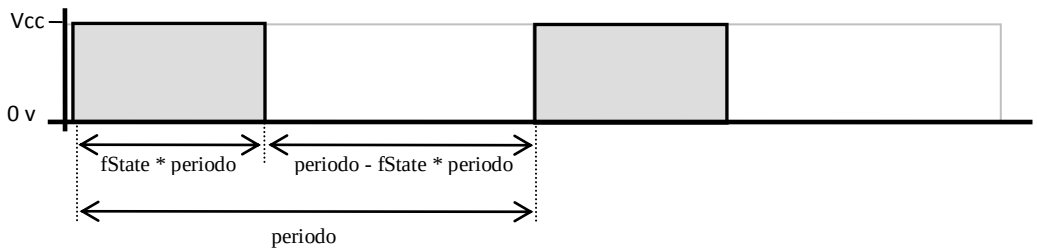


Figura 4.24. Variables que describen la onda cuadrada en PWM.

Clase SensaGlove

La clase SensaGlove se encarga de gestionar la creación de las 5 instancias de la clase Finger, así como de sus hilos de ejecución. Los hilos se han implementado haciendo uso de la librería *VPR (VR Juggler Portable Runtime)*, que proporciona métodos necesarios para su gestión independientes del sistema operativo.

Esta clase servirá como interfaz para la comunicación con cualquier programa que haga uso del guante de retorno háptico, por lo que proporcionará métodos para ajustar el nivel de vibración de cada dedo, y para consultar el estado de su contacto asociado.

Las funciones más importantes que implementa son: *isPressed(int id)* que indica si el contacto está activado o no, y *setState(int id, float level)*, que establece la intensidad de vibración de uno de sus actuadores a un valor que oscila entre 0 y 1.

CAPÍTULO 5. DISEÑO E IMPLEMENTACIÓN DE LA APLICACIÓN

En este capítulo se tratará el diseño y la implementación de la aplicación de modelado que hará uso del guante de datos con retorno háptico. En una primera parte se especificará el diseño del sistema inmersivo, tanto lo que respecta a la organización física del hardware, como a la arquitectura del software. Por último se hablará sobre la implementación de la aplicación, refinando y adaptando el modelo de la arquitectura a los requisitos de la API y describiendo los algoritmos utilizados.

5.1 DISEÑO

5.1.1 Organización física

Una vez analizados los requisitos de la aplicación, y decidido cuáles van a ser los dispositivos hardware que se van a usar, se realizará una descripción de cuál es la organización física de todos ellos.

En primer lugar, el uso de la pantalla de retroproyección condiciona a que el usuario tenga que estar de pie frente a ella, ya que su elevada altura no permite el uso de una silla.

Para poder ver correctamente las imágenes estéreo proyectadas sobre esta pantalla, el usuario necesitará llevar unas gafas con filtros polarizados que deben coincidir con los montados sobre los cañones de proyección (cada ojo con su proyector).

El sistema a usar para la localización es Flock of Birds, como ya se indicó en el capítulo 3, del que se usarán dos de los tres sensores disponibles. Uno de ellos estará colocado sobre el guante, junto encima de la palma de la mano, para así poder medir su posición e inclinación. El otro receptor irá colocado en un lado de la cabeza, en una de las patillas de las gafas polarizadas. De esta manera se puede seguir la cabeza del usuario y mostrar en la pantalla exactamente lo que vería desde su punto de vista si tuviera el objeto delante de él.

Además, el uso de Flock of Birds restringe la zona de trabajo a un radio aproximado de 1.2 metros alrededor del emisor. Para poder aprovecharla al máximo, es necesario colocar este emisor lo más cerca posible del usuario sin que pueda molestarle en su trabajo.

La zona de trabajo límite de un usuario modelando un objeto que "sale de la pantalla" comienza en esta misma pantalla, ya que no podrá atravesarla. También estará limitada por sus cuatro márgenes, ya que obviamente tampoco se podrá representar ningún objeto que se salga de los mismos.

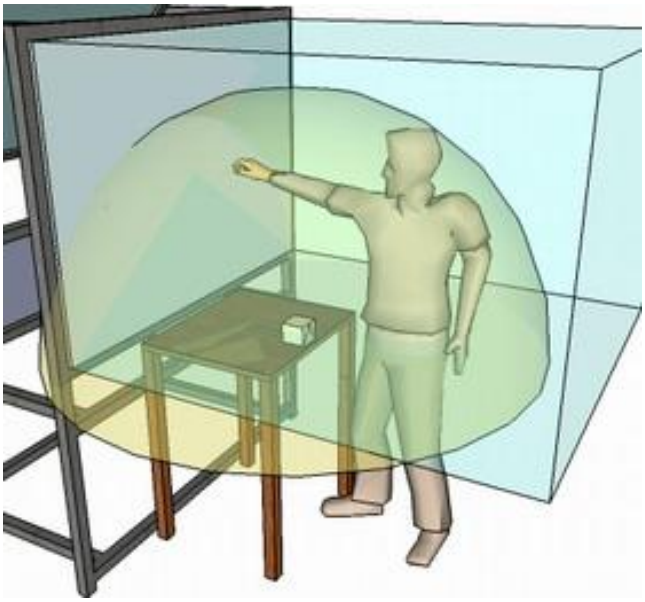


Figura 5.1. Zona de trabajo límite del usuario (azul) y de FOB (amarillo).

Con esta situación, el lugar más adecuado para la colocación del emisor es sobre una mesa que tenga una altura similar a la zona más baja de la pantalla (1 metro aproximadamente), y retirado de ésta unos 65 centímetros (Figura 5.1). Como el receptor tiene una zona de trabajo configurable con forma de semiesfera (véase anexo C), se configurará para que ésta sea la de la parte superior.

A partir de todas estas decisiones de diseño, es posible hacerse una idea de cuál será el aspecto final del sistema inmersivo (Figura 5.2). En él se puede ver a un usuario frente a un sistema de retroproyección estereoscópico llevando unas gafas polarizadas y un guante de datos, además de un sensor de localización sobre cada uno de ellos. La posición del emisor sobre la mesa es la óptima para maximizar el espacio de trabajo, y no molestará en la manipulación del objeto virtual.

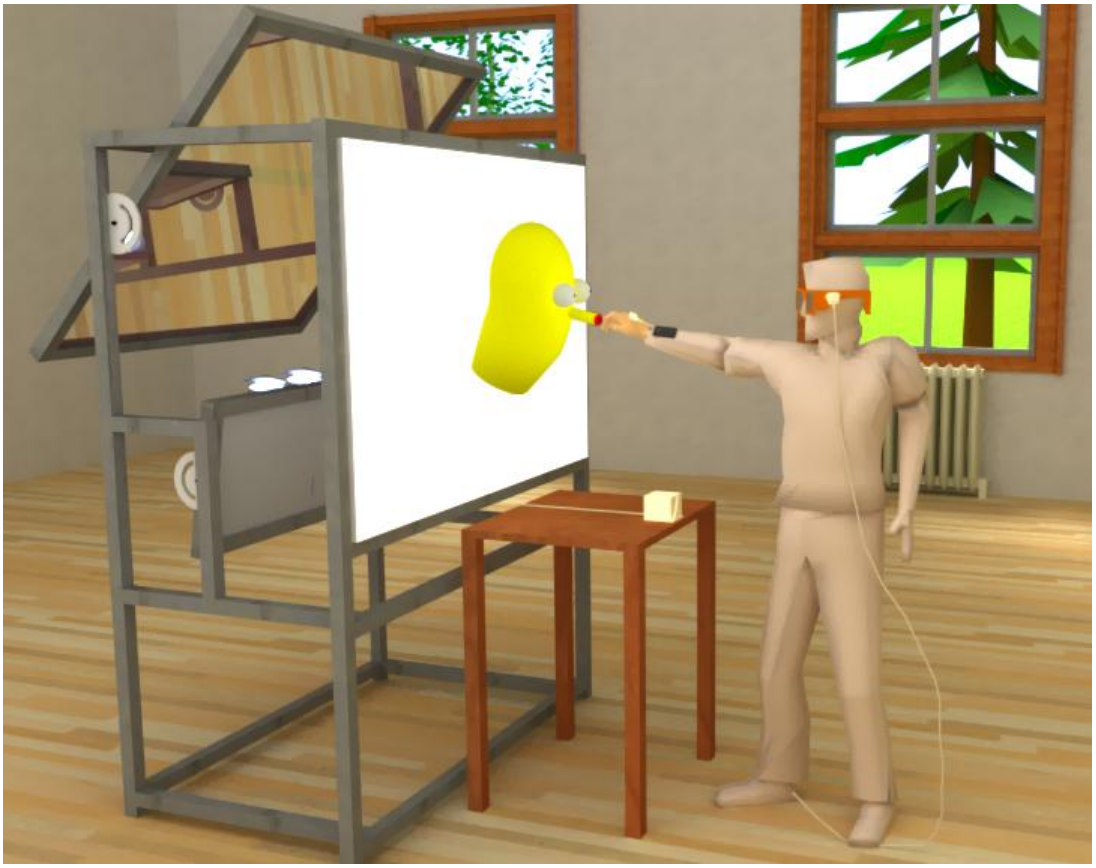


Figura 5.2. Impresión artística del aspecto final que tiene el sistema inmersivo.

5.1.2 Arquitectura Software

En esta primera fase del diseño se realizará una división de la aplicación en módulos. Para ello se buscará que cada uno tenga una funcionalidad bien definida, maximizando su cohesión, y además que sean necesarias las mínimas relaciones entre ellos, es decir, minimizando el acoplamiento. De esta manera será más fácil realizar un mantenimiento posterior de la aplicación, se mejorará la detección y corrección de errores, y será más fácil reutilizar estas unidades de software en el futuro.

A este nivel se realizará una especificación a alto nivel sin tener en cuenta detalles relativos a la API que se utilice, suponiendo simplemente que ésta proporciona una capa de abstracción del hardware y del sistema operativo. Tampoco se tendrá en cuenta aún el grafo de la escena usado, considerándolo una clase genérica a alto nivel que contiene al objeto 3D y sus componentes.

La especificación de las clases no será exhaustiva, ya que únicamente se describirán cuales son los métodos y atributos principales que deben tener para cumplir su cometido.

Es posible por tanto que este esquema sufra pequeñas variaciones más tarde cuando se tengan en cuenta las peculiaridades de implementación de la API en cuestión y del grafo de la escena.

Los módulos o clases principales de los que se compone la aplicación y su funcionalidad asociada son los siguientes (Figura 5.3).

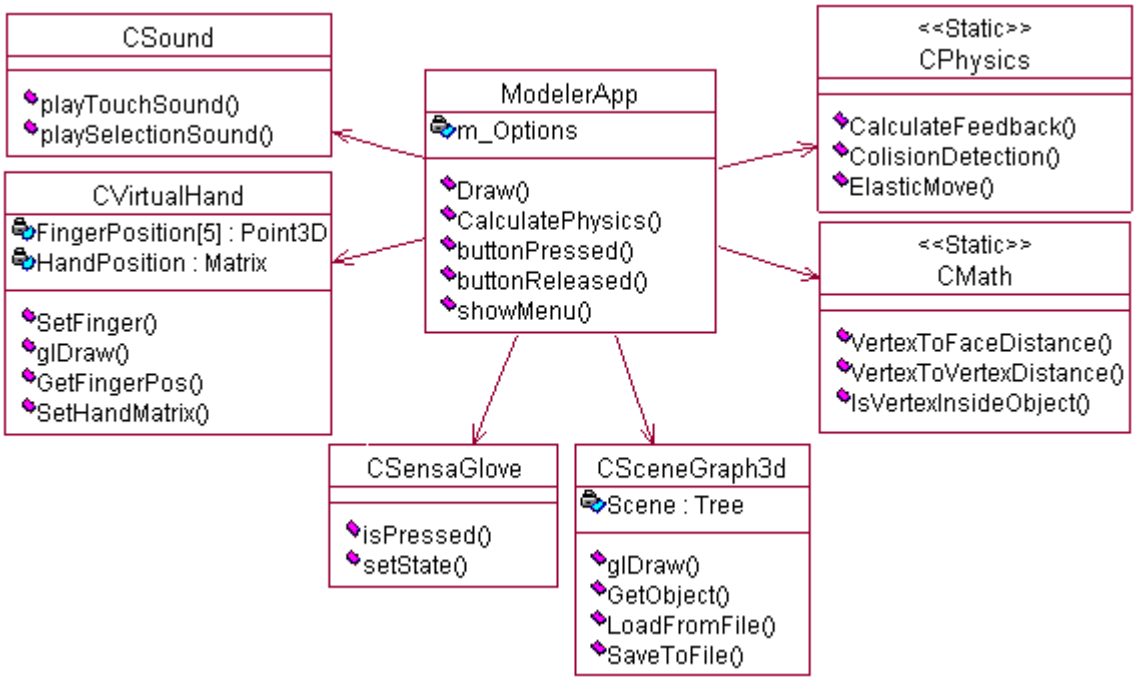


Figura 5.3. Diagrama de clases de la aplicación a alto nivel.

ModelerApp

Es la clase que implementa la aplicación y su función es la de decidir qué hacer frente a cada evento del sistema, interactuando para ello con el resto de clases. Además gestionará todas las opciones visuales y de retorno hacia el usuario, por lo que deberá almacenarlas como miembro de la clase. Para poder cambiarlas se mostrará al usuario un menú con el método *showMenu()*, que se activará mediante el teclado.

El método *Draw()* decidirá en cada momento qué es lo que hay que mostrar en pantalla, llamando a los métodos *glDraw()* del grafo de la escena, de la mano virtual, o del vértice seleccionado, por ejemplo.

El método *CalculatePhysics()* será el encargado de coordinar las operaciones de cálculo de físicas, como por ejemplo el cálculo del retorno háptico y deformación del objeto cuando haya una colisión.

Por último, los métodos *buttonPressed()* y *buttonReleased()* se llamarán cuando se activen o desactiven los contactos del guante de datos indicando una entrada del usuario. A partir de ellos se decidirá si hay que mover el vértice o cara seleccionada, si hay que acercar o alejar el objeto, o si hay que restablecerlo, según está especificado en los requisitos.

CSound

Se encarga de inicializar y reproducir sonidos que sirvan como realimentación para el usuario.

El método *playTouchSound()* se usará para reproducir un sonido cuando se toque una superficie con un dedo, y el método *playSelectionSound()* reproducirá un sonido para indicar que se ha seleccionado un vértice o una cara.

Las operaciones podrán llevar como argumento el volumen al cual se han de reproducir. Esto será útil para indicar la distancia a la que se encuentre un vértice cuando se seleccione, por ejemplo.

CVirtualHand

Implementa el modelo de mano virtual del usuario. Su cometido es en primer lugar calcular la posición absoluta de la yema de cada dedo respecto a la escena, y en segundo lugar representar gráficamente la mano del usuario con un modelo en 3D que le ayude a conocer su situación y su estado.

CVirtualHand ofrecerá métodos para establecer tanto la posición y orientación de la mano del usuario, mediante *setHandMatrix()*, como la de sus dedos, mediante *setFinger()*. También necesitará uno para dibujar la mano virtual, llamado *glDraw()*, y otro para devolver la posición absoluta de la yema de cada mano, llamado *getFingerPos()*.

CSensaGlove

Esta clase proporciona una interfaz para la comunicación con el guante háptico creado (véase apartado 4.6).

Las dos operaciones principales que se utilizarán son *isPressed()* y *setState()* que servirán para conocer si se está haciendo contacto entre los dedos y establecer el estado de cada uno de sus vibradores respectivamente.

CSceneGraph3d

Implementa el grafo de la escena, conteniendo en una estructura jerárquica la malla del objeto que se desea modelar, sus vértices, sus caras y sus aristas.

Debe proporcionar al menos una función para devolver la malla con la que se está trabajando, una función para cargar la escena a partir de un archivo, y otra para guardarla. A estas funciones se les ha llamado *GetObject()*, *LoadFile()* y *SaveFile()* respectivamente en el diagrama de clases.

CPhysics

Esta clase proporcionará una serie de funciones relacionadas con el cálculo de las propiedades físicas del objeto virtual. Será estática, ya que no será necesario almacenar ninguna información en forma de miembros de la clase.

Algunas de las funciones que debe implementar son: *colisionDetection()*,para hacer el cálculo de colisiones, *calculateFeedback()*, para calcular la intensidad del retorno háptico, y *elasticMove()* para calcular la deformación producida en la malla.

CMath

Por último en esta clase se incluirán todas las funciones matemáticas que sean necesarias para la implementación de los algoritmos. Será también una clase estática, y entre otros métodos pueden estar los del cálculo de distancias (*VertexToFaceDistance(), VertexToFaceDistance()*) o de comprobación de regiones interiores de un objeto (*isVertexInsideObject()*).

5.2 IMPLEMENTACIÓN

5.2.1 Refinamiento de la arquitectura

La arquitectura creada en la fase de diseño es útil para establecer una división en clases de la aplicación, y para dar una idea sobre la forma en la que interaccionarán los distintos módulos entre sí. No obstante esta arquitectura está diseñada sin tener en cuenta los detalles relativos a la API que se va a utilizar, en este caso VR Juggler.

Es por ello que se creará una nueva revisión en la que se incluirán estos detalles, además de los referentes a la implementación concreta del grafo de escena utilizado.

El diagrama de clases de la fase de diseño no ha sufrido muchas variaciones importantes, manteniendo su filosofía original (Figura 5.4).

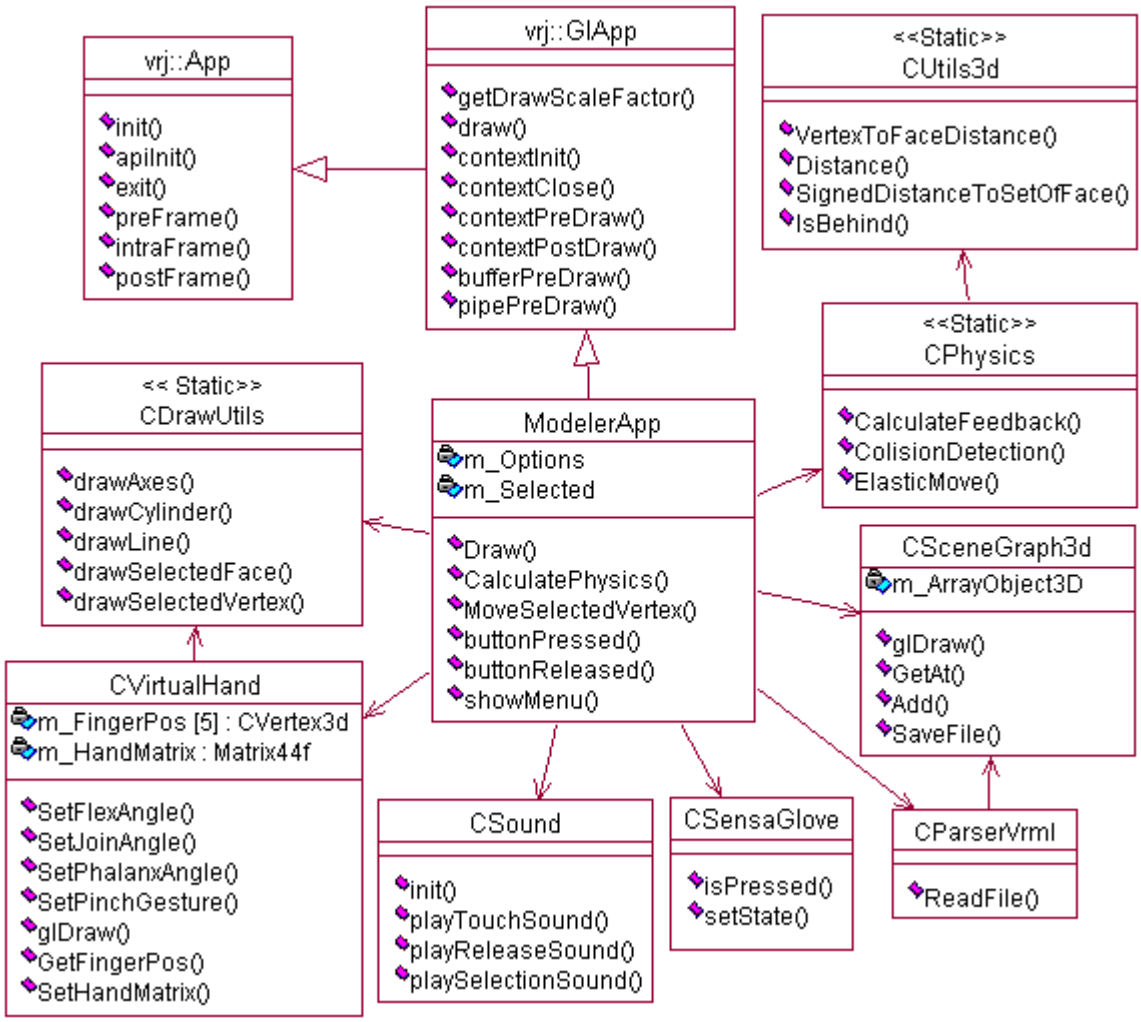


Figura 5.4. Diagrama de clases de la aplicación en detalle.

Como se puede ver, se ha añadido una clase estática *CDrawUtils* que contendrá todas las rutinas auxiliares necesarias para el dibujado de primitivas en la pantalla. Esta clase será usada por *CVirtualHand* en el proceso de dibujado de la mano virtual y por *ModelerApp* para mostrar si es necesario el vértice o cara seleccionado.

Se ha representado también en el diagrama una nueva clase *CParserVrml* que se encarga del proceso de interpretación de un archivo VRML para cargar la escena, y que forma parte de la librería 3D Toolbox.

La clase estática *CMath* ha sido sustituida por la clase *CUtils3d* que también forma parte de la librería 3D Toolbox y que proporciona numerosas funciones relacionadas con cálculos geométricos.

Por último se han representado en el diagrama las clases *App* y *GlApp* procedentes de la librería de VR Juggler, y de las cuales hereda la clase principal *modelerApp*. Estas clases definen los métodos que debe implementar la aplicación para que sean llamados por el núcleo de ejecución (véase anexo A).

5.2.2 Detalles de implementación

En este apartado se hará una descripción de los detalles más relevantes que se han tenido en cuenta al hacer la implementación de la aplicación. En primer lugar se comentará brevemente cómo está organizado el grafo de la escena proporcionado por 3D Toolbox. A continuación se comentarán detalles de implementación ligados al uso de VR Juggler como API, y a OpenGL para el dibujado de gráficos. Finalmente se darán unas nociones de cómo interactúan unos módulos con otros para lograr dar vida a la aplicación.

3D Toolbox

La librería 3D Toolbox utilizada está escrita en C++ y pensada para utilizar las MFC (Microsoft Foundation Classes) de Windows. Sin embargo, las referencias a esta librería son mínimas, ya que sólo se utiliza para la lectura y escritura de archivos. Como esta librería sólo está presente en los sistemas operativos de la familia Windows, y para evitar hacer la aplicación dependiente del sistema operativo, se ha realizado una sencilla adaptación sustituyendo cualquier referencia a MFC por un equivalente estándar presente en todos los sistemas operativos. Por ejemplo, la referencia a la clase *CFile* se ha sustituido por *FILE*, presente en *stdio.h*.

Respecto a su arquitectura, se ha representado en un diagrama las clases principales que se utilizan en la aplicación (Figura 5.5). No se han especificado todos los métodos ni todos los atributos, tan solo algunos de los necesarios para hacerse una idea de su estructura.

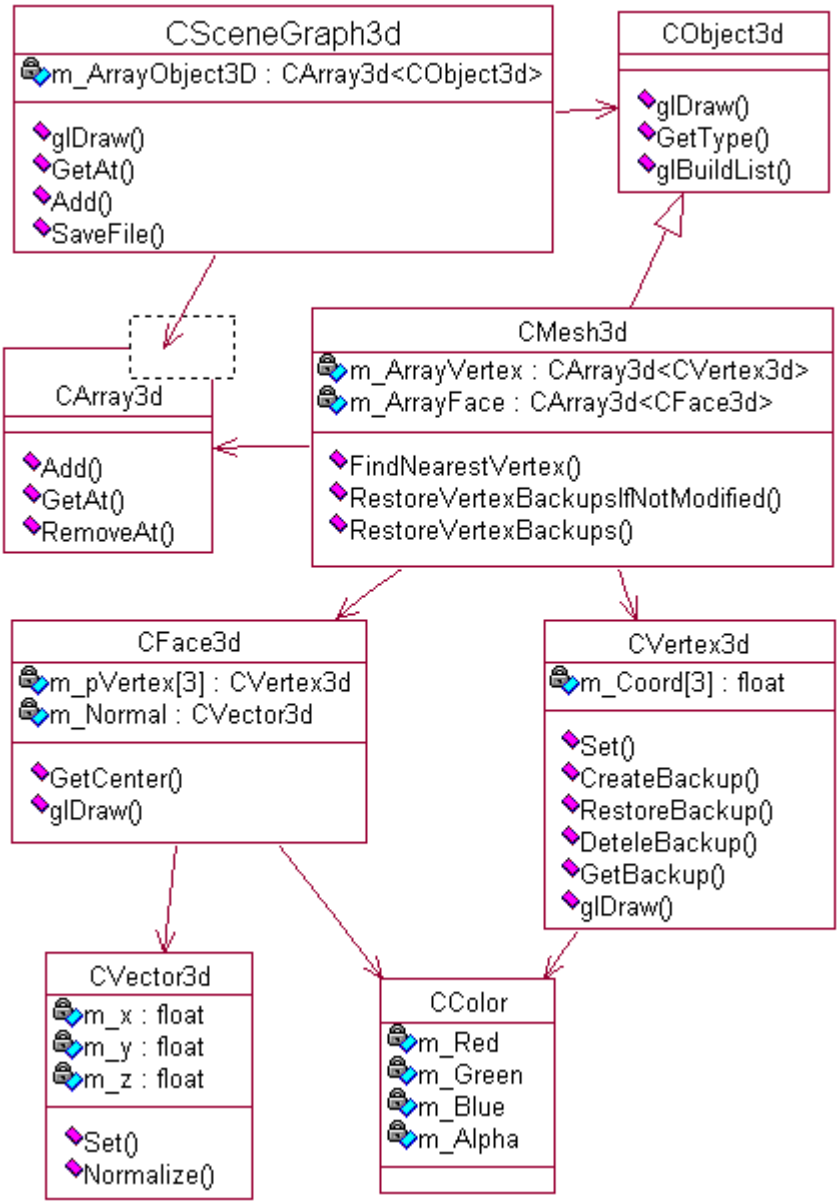


Figura 5.5. Diagrama de clases general del grafo de la escena.

La clase principal que implementa el grafo de la escena es *CSceneGraph3d* que tiene asociado un array de objetos 3d, representados por la clase *CObject3d*. Esta clase contiene una especificación de los métodos que tiene que implementar cualquier objeto para poder ser incluido en el grafo de la escena. Los métodos más importantes son el de dibujado (*glDraw)* y el de construcción de listas de OpenGL (*glBuildList*).

La clase *CMesh3d* hereda de CObject3d, y representa un objeto de tipo malla de polígonos, teniendo como atributos principales una lista de vértices y otra de caras.

Los vértices están representados por la clase *CVertex3d*, cuyos atributos serán tres valores reales correspondientes a su posición en el espacio. A esta clase se le han añadido métodos relativos a gestionar copias de los mismos. Estas copias serán útiles para el caso de los objetos elásticos, ya que para que puedan recuperar su forma será necesario saber cuál era su estado original.

Para poder tener la libertad de cambiar los vértices de la malla sin restricciones, los polígonos que forman la malla serán siempre triángulos. Esto es debido a que los tres vértices de un triángulo siempre forman un plano sin importar cómo estén situados, cosa que no pasa con el resto de polígonos. Por este motivo la clase *CFace3d*, que es la que implementa las caras, tiene como atributo los tres vértices de los que se compone.

Esta restricción también se ha tenido en cuenta en la clase *CParserVrml*, que se encarga de leer la malla de polígonos desde un archivo de tipo **X3DV**. Esta clase ha sido adaptada para permitir la entrada de mallas de cualquier tipo de polígonos, realizando una triangulación de los mismos cuando están formados por más de tres vértices (Figura 5.6).

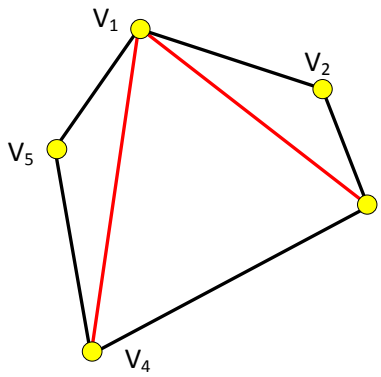


Figura 5.6. Ejemplo de triangulación de un polígono de 5 vértices.

VR Juggler

Debido a la arquitectura de VR Juggler, es su núcleo de ejecución el que se encarga de controlar todo el sistema, creando un bucle de control en el que realiza sucesivamente llamadas al programa para así ejecutarlo.

Las funciones que serán llamadas son las heredadas de las clases *glApp* y *App*, y que serán sobreescritas en la clase *ModelerApp*. Las más importantes son:

- **preFrame().** Esta función es llamada por VR Juggler después de actualizar los dispositivos y justo antes de dibujar.

- **draw().** Es la función de dibujado en la que se deben escribir primitivas de OpenGL para mostrar la escena.

- **intraFrame().** Una vez que se está dibujando, se lanza un hilo en paralelo en el que se llama a esta función. Es útil para aprovechar posibles tiempos muertos del procesador.

- **postFrame().** Esta función se ejecuta después de que haya finalizado el dibujado y el procesamiento de *intraFrame()*, justo antes de realizar la actualización de los dispositivos.

El funcionamiento de la aplicación se basará en especificar algunas de estas funciones según convenga, teniendo en cuenta cuando se ejecutará cada una de ellas. Además hay otras funciones relativas a OpenGL que son también interesantes, como *contextInit()* (véase anexo A).

Una vez que se ha introducido la forma que tiene un programa de VR Juggler, es necesario comentar cuál es la forma de acceder a los dispositivos.

Cada dispositivo en VR Juggler debe tener asociado un proxy de un tipo determinado en el fichero de configuración. Los tipos más comunes son: *analog*, para datos analógicos como el eje de un joystick, *digital*, para datos binarios como un botón de un gamepad y *positional* para datos con información de posición y/o orientación, como los de un sistema de localización.

A estos proxies se puede acceder desde la aplicación creando una variable de tipo *DigitalInterface, AnalogInterface* o *PositionalInterface* respectivamente y llamando en cualquier momento al método *getData()*.

En la aplicación de modelado se han definido dos variables de tipo *PositionalInterface* para la cámara y el guante, y 5 variables de tipo *DigitalInterface*, uno que será el que activará el menú, y el resto para poder emular los contactos del guante en el caso de que no estuviera presente.

Los datos proporcionados por los sensores son matrices de reales de dimensiones 4x4 que contienen una transformación directamente utilizable en OpenGL. Los datos

proporcionados por los botones digitales, sin embargo, no se utilizarán directamente. En lugar de ello, se consultarán para comprobar cuando ha cambiado su estado, y una vez que eso ocurra se lanzará un evento llamando a una función que lo maneje.

Las funciones definidas para el control de los eventos de los botones son: *buttonPressed()* que indica que se ha apretado un botón, y *buttonReleased()* que indica que se ha dejado de apretar un botón. Para no perder funcionalidad, se han definido también las funciones *buttonIsDown()* y *buttonIsUp()* que serán llamadas siempre que un botón esté apretado y siempre que esté liberado, respectivamente.

OpenGL

OpenGL es la API usada por VR Juggler para la creación de gráficos en 3D. Se trata de una especificación estándar y multiplataforma desarrollada por Silicon Graphics Inc. (SGI) en 1992 usada en multitud de aplicaciones y juegos [OpenGL].

OpenGL se comporta como una máquina de estados que cambia con cada llamada a la API que se realiza y perdura hasta la próxima. Estas llamadas permiten dibujar primitivas gráficas como puntos, líneas y polígonos, definir características de iluminación, mapear texturas, hacer transformaciones geométricas, etc.

Una forma de acelerar el dibujado de una escena es almacenar una serie de llamadas mediante un *display list*, de tal forma que puedan ser invocadas más tarde simplemente haciendo referencia a este identificador. Por ejemplo se podría guardar en un display list una definición completa del escenario virtual, y posteriormente en cada iteración del bucle de dibujado invocar al mismo, lo que hará que se ejecute mucho más eficientemente.

En VR Juggler cada ventana tiene asociada una máquina de estados de OpenGL diferente, lo que se conoce como **contexto de OpenGL**. Cada contexto guarda el estado de su instancia de OpenGL, que contiene información sobre el color, textura, sombreado, display lists, etc.

Hay que tener en cuenta que cuando se tiene una configuración con más de una ventana, habrá también más de un contexto de OpenGL, y la función de dibujado se ejecutará una vez por cada contexto. Es muy importante ser consciente de este hecho para evitar cambiar variables globales o miembros de la clase durante el dibujado, ya que podría producir resultados inesperados.

Otra cuestión a tener en cuenta es la creación de display lists, ya que es necesario crear un display list diferente para cada contexto, e invocarlo correctamente durante el dibujado. Esto se consigue mediante las variables dependientes del contexto proporcionadas por VR Juggler. Esta es una manera transparente de mantener una copia diferente de la variable para cada contexto de OpenGL, de tal forma que el programador no tenga que preocuparse de saber en qué contexto se encuentra.

Modelo de mano virtual

La clase *CVirtualHand* se ha diseñado e implementado tanto para crear un modelo de mano virtual que pueda ser representado gráficamente, como para calcular la posición de la yema de cada dedo respecto a las coordenadas de la escena.

Para ello se ha tenido en cuenta no sólo que pueda ser utilizada con unos guantes de datos que permitan conocer si los dedos hacen contacto entre sí, sino que además se permitan guantes de datos que devuelvan la posición "exacta" de cada dedo.

Así, esta mano virtual permite:

- Cambiar el ángulo de flexión de cada uno de los dedos (*SetFlexAngle*).
- Cambiar el ángulo de flexión de las falanges de cada uno de los dedos (*SetPhalanxAngle*).
- Cambiar el ángulo de abducción de cada uno de los dedos (*SetJoinAngle*).
- Establecer un gesto, indicando el dedo que está haciendo contacto con el pulgar (*SetPinchGesture*).

Figura 5.7. Modelo de mano virtual

Funcionamiento general de la aplicación

En primer lugar se procederá a describir el comportamiento de la aplicación en términos de estados. La aplicación tendrá dos estados principales: modo de calibración y modo de edición del objeto (Figura 5.8).

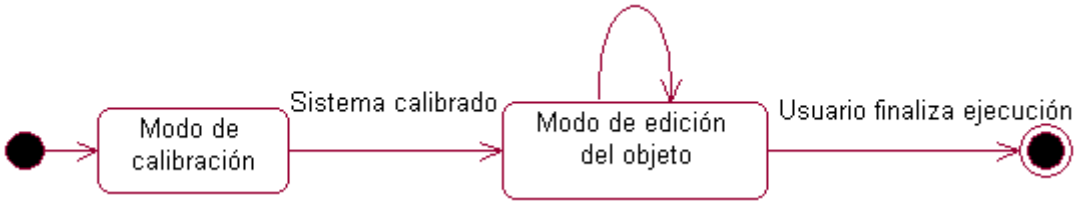


Figura 5.8. Diagrama de estados general de la aplicación.

Al primer estado se llegará al lanzar la aplicación y servirá para que el usuario calibre la posición de su mano respecto a la escena virtual. A pesar de que la configuración inicial puede ser suficiente en la mayoría de los casos, se permitirá hacer un ajuste fino para anular cualquier desviación. Así se asegurará una buena correspondencia entre la mano física del usuario y la mano virtual, que es vital en este tipo de aplicaciones [Swa06].

Esta calibración consistirá en iluminar el vértice del objeto más cercano al usuario para que el usuario intente cogerlo con sus dedos índice y pulgar. Una vez que el usuario haga el gesto de cogerlo, se supondrá que la mano virtual debería estar donde está el vértice, siendo la diferencia entre ambos el error de calibración. Este error se almacenará en un vector llamado *m_OffsetVector* en la clase principal, y se tendrá en cuenta al actualizar la información de posición de la mano virtual.

El segundo estado es el de edición del objeto virtual y se permanecerá en él hasta que el usuario decida finalizar la aplicación.

En este estado es necesario elegir el momento adecuado para realizar cada una de las actividades, teniendo en cuenta en qué momento ejecuta el núcleo de VR Juggler cada una de las funciones de la aplicación.

Las actividades que se han de realizar son:

- **Dibujado de la escena**. Esta actividad se realizará en la función *draw()* proporcionada por VR Juggler para tal efecto, y en ella se ordenará el dibujado de la mano virtual, de los elementos seleccionados, y del grafo de la escena.

- **Actualizar la mano virtual.** El mejor lugar para realizar la actualización en la posición de la mano virtual (*updateGlove()*) es justo después de que VR Juggler actualice los dispositivos, ya que los datos serán más recientes. Por tanto, se hará en *preFrame.*

- **Cálculo de la física.** El cálculo de la física (*calculatePhysics()*) comprende las actividades de deformación de la superficie del objeto, cálculo del retorno háptico, retorno sonoro y selección de vértices o caras. Estas actividades se deben realizar siempre que el usuario no esté estirando del objeto seleccionado, por lo que el dedo índice no debe de estar haciendo contacto con el pulgar. Su lugar por tanto es el del evento *buttonIsUp(INDEX).*

- **Mover el objeto seleccionado.** Esta tarea (*moveSelectedVertex()*) se debe realizar cuando el usuario esté haciendo el gesto de coger un objeto, estando el dedo índice en contacto con el pulgar. Es por ello que se debe poner en el evento *buttonIsDown(INDEX).*

- **Acercar el objeto.** Tal y como está especificado en el análisis, se debe acercar el objeto al hacer contacto los dedos corazón y pulgar. Por tanto se realizará en *buttonIsDown(MIDDLE).*

- **Alejar el objeto.** El dedo que se usará para alejar el objeto es el anular, por tanto se usará el evento *buttonIsDown(RING)*.

- **Restablecer objeto.** El objeto será restablecido al hacer contacto el dedo meñique con el pulgar, pero hay que tener en cuenta que no hay que realizarlo constantemente, sino sólo una vez. Es por eso que se puede elegir entre hacerlo justo al hacer contacto *buttonPressed(PINKY)* o al dejar de hacerlo *buttonReleased(PINKY).*

- **Mostrar menú.** El menú se debe mostrar con la tecla F1, lo que debe ser configurado mediante vrjConfig para asignarle un proxy digital específico (ver anexo A). En el programa se hará de la misma forma que se gestionan los contactos del guante, haciendo uso del evento *buttonReleased* para mostrarlo.

5.2.3 Descripción de algoritmos y actividades

Escalado automático

Debido a las características del sistema inmersivo implementado, el espacio de trabajo es limitado, por lo que los objetos que se modelen deben adaptarse a él.

La forma de adaptarlos será mediante un escalado proporcional en todas sus dimensiones, bien para agrandarlo o bien para encogerlo. También podrán ser trasladados si éstos se encuentran fuera del alcance del usuario.

Para ello se ha implementado una función en la clase *CMesh3d* llamada *AutoScale* cuyos argumentos son las dimensiones máximas y mínimas en cada una de las coordenadas. Los rangos de usabilidad son dependientes de la aplicación, por lo que se han definido como constantes en la clase *ModelerApp*. Estas constantes son:

- MAX_OBJECT_WIDTH = 1.0f
- MAX_OBJECT_HEIGHT = 1.0f
- MAX_OBJECT_DEPTH = 0.40f

Las medidas están expresadas en metros, y hacen referencia a la anchura máxima de un objeto (coordenada X), máxima altura (coordenada Y) y máxima profundidad (coordenada Z).

Respecto al algoritmo, se ha hecho una descripción del mismo en pseudocódigo para describirlo (Listado 5.1):

```
FUNCION AutoEscalado(minimaX, maximaX, minimaY, MaximaY, minimaZ, maximaZ)
COMENZAR
        //Calcular factor de escalado en cada coordenada
        FactorEscaladoX = (maximaX – minimaX) / (maximaXActual – minimaXActual)
        FactorEscaladoY = (maximaY – minimaY) / (maximaYActual – minimaYActual)
        FactorEscaladoZ = (maximaZ – minimaZ) / (maximaZActual – minimaZActual)

        //Calcular factor de escalado del objeto
        FactorEscaladoObjeto = minimo(FactorEscaladoX, FactorEscaladoY, FactorEscaladoZ)

        Centrar_objeto(0.0, 0.0, 0.0)

        PARA CADA Vertice
                Vertice.x = Vertice.x * FactorEscaladoObjeto
                Vertice.y = Vertice.y * FactorEscaladoObjeto
                Vertice.z = Vertice.z * FactorEscaladoObjeto
        FIN PARA
```

```
        Centrar_objeto((maximaX-minimaX)/2.0,
                       (maximaY-minimaY)/2.0,
                       (maximaZ-minimaZ)/2.0)
FIN
```

Listado 5.1. Descripción del algoritmo de escalado automático en pseudocódigo.

Función de retorno de fuerzas

La función de retorno de fuerzas es la encargada de calcular la intensidad con la que debe activarse cada estimulador vibrotáctil en función de la posición relativa del dedo respecto a la superficie más cercana.

Esta función se aplica suponiendo que el objeto es elástico de forma homogénea, y se basa en la ley de elasticidad de Hooke, según la cual la fuerza aplicada es proporcional a la deformación producida:

$$F = K \cdot x$$

Donde F es la fuerza aplicada y x es la deformación relativa.

Según la tercera ley de Newton, o principio de acción-reacción, cuando un cuerpo ejerce una fuerza sobre otro, éste ejerce sobre el primero una fuerza igual y de sentido opuesto. Por este motivo, la fuerza que sentirá un dedo tendrá el mismo valor absoluto que la que ejerza sobre un objeto virtual.

A partir de estos principios, la función devolverá una intensidad de cero mientras el dedo no toque la superficie, y crecerá progresivamente conforme se vaya atravesando. La intensidad será máxima a partir de cierta posición, llegando a la saturación del dispositivo háptico (Figura 5.9).

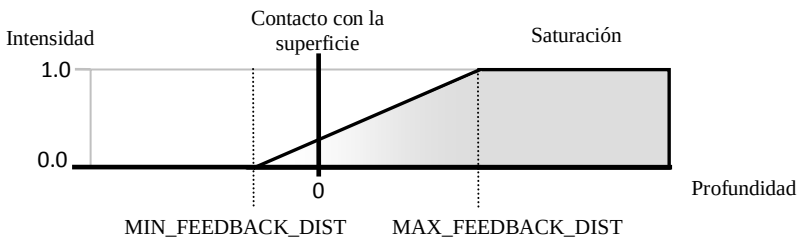


Figura 5.9. Función de retorno de fuerzas.

Los parámetros que definen esta respuesta son:

- **MIN_FEEDBACK_DIST**. Distancia a partir de la cual se empieza a proporcionar una respuesta con mínima intensidad. Suele estar en torno a la posición cero, justo cuando se realiza el contacto, pero puede comenzar un poco antes, ya que los valores más bajos de intensidad no son perceptibles.

- **MAX_FEEDBACK_DIST**. Distancia a partir de la cual se proporciona una respuesta de intensidad máxima. Depende de las características de elasticidad o rigidez que se le quieran proporcionar al objeto.

A partir de estas especificaciones, la función de retorno de fuerzas queda definida como sigue:

$$F = \min\left(1,\quad \max\left(0,\quad \frac{x - MIN\_FEEDBACK\_DIST}{MAX\_FEEDBACK\_DIST - MIN\_FEEDBACK\_DIST}\right)\right)$$

Algoritmo de deformación de mallas

Cuando se estira de un punto de un objeto elástico (por ejemplo, un globo) no solo se cambia de posición ese punto, sino que toda su vecindad le acompaña dependiendo de lo próximos que estén y de las características del material.

El planteamiento de este algoritmo, a pesar de estar basado en la ley de elasticidad de Hooke, no responde a una simulación estricta teniendo en cuenta todas las propiedades y leyes de la física. Su objetivo es mostrar al usuario una deformación creíble que pueda ser simulada en tiempo real de forma sencilla, ya que de lo contrario se necesitarían resolver costosas ecuaciones matemáticas.

Para modelar esta propiedad se puede considerar el objeto como una red de nodos con conexiones flexibles entre ellos, por ejemplo, muelles. Esta aproximación es usada en el modelado de objetos no rígidos como telas o bolsas de plástico [DHe95], pero también se puede usar para calcular las deformaciones de un objeto virtual.

Se puede considerar el problema desde un punto de vista bidimensional, asumiendo que todos los nodos están sobre un mismo plano, ya que a partir de ahí adaptarlo a las tres dimensiones es casi inmediato.

Si se tiene una malla de nodos en la que se desplaza un vértice $\boldsymbol{V_0}$ hacia una nueva posición $\boldsymbol{V_0'}$ (Figura 5.10), esta nueva posición refleja que las aristas que lo unen con sus

vecinos han cambiado también, siendo algunas de mayor longitud que al principio, y otras de menos. Considerando que estas aristas se comportan como si fueran muelles, aquellas que están estiradas ejercen una fuerza de atracción hacia el vértice, y las que están comprimidas ejercen una fuerza de repulsión.

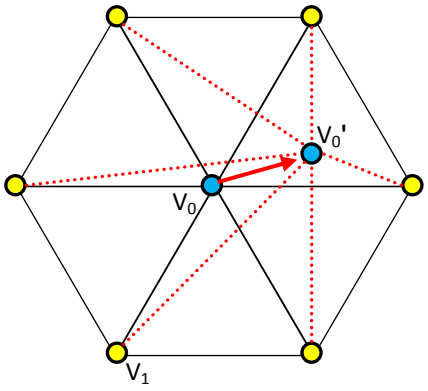


Figura 5.10. Desplazamiento de un vértice en una malla.

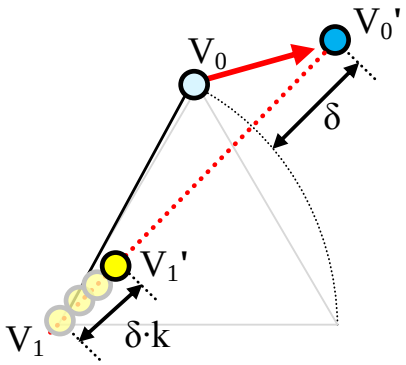


Figura 5.11. Desplazamiento de un nodo vecino (amarillo) al mover un vértice de una malla (azul).

Dependiendo de la elasticidad del material, el desplazamiento que se produce en el nodo vecino se atenúa en mayor o menor medida. Además la atenuación que se produce sobre una fuerza de compresión (que empuje el nodo vecino) es mucho mayor.

Siendo $\boldsymbol{\delta}$ la variación de longitud que se produce en la arista, y $\boldsymbol{k}$ el factor de elasticidad del material cuyo valor estará comprendido entre 0 y 1, el desplazamiento del nodo vecino $\boldsymbol{V_1}$ será $\boldsymbol{\delta \cdot k}$ (Figura 5.11). Si se quiere modelar enlaces muy rígidos, este factor estará próximo a 1, lo que provocará que se produzca un movimiento del nodo vecino casi de la misma magnitud. En cambio, un valor muy bajo de la constante hará que los nodos vecinos apenas se muevan.

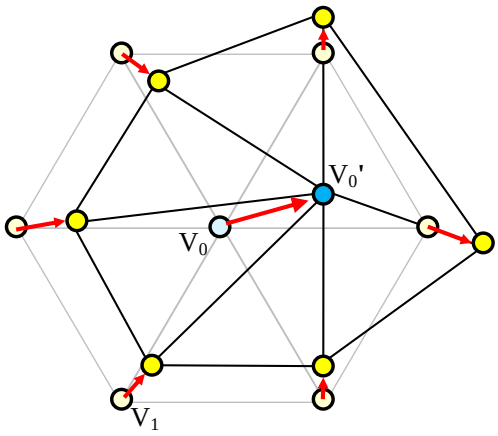


Figura 5.12. Deformación global de una malla al mover uno de los vértices.

Otra posibilidad que se puede implementar fácilmente, es el desplazamiento de los nodos vecinos teniendo en cuenta la distancia original a la que se encontraban, de tal

manera que aquellos más alejados se muevan menos que los más cercanos. Esto se puede conseguir multiplicando el desplazamiento conseguido por la inversa de la distancia original más uno.

$$desplazamiento = \delta \cdot k \cdot \frac{factor\_dist}{1 + dist\_original}$$

Una vez calculadas las posiciones de los nodos vecinos, se puede seguir calculando de forma recursiva las deformaciones producidas a su vez por éstos a nuevos vértices más alejados. El factor de elasticidad se puede mantener durante la propagación, o bien es posible atenuarlo a su vez, por ejemplo, elevándolo al cuadrado (Figura 5.13).

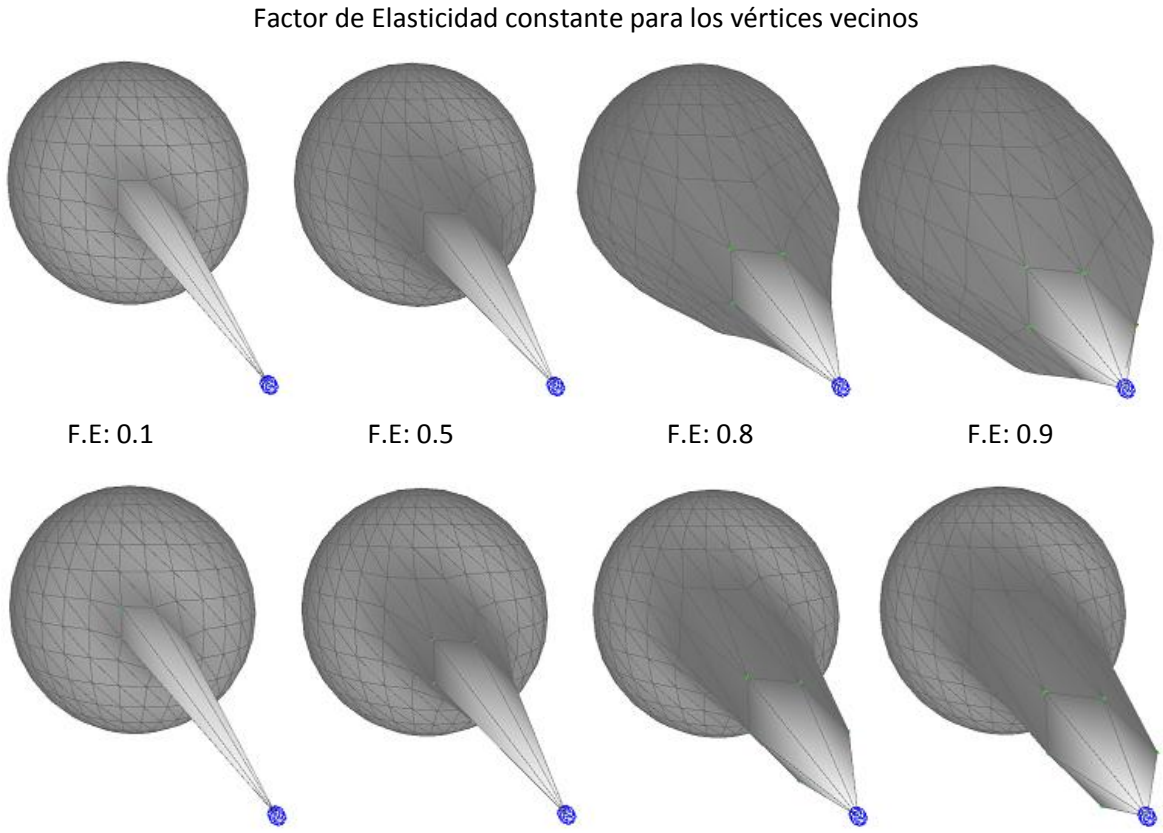


Figura 5.13. Deformación producida sobre una esfera con distintos factores de elasticidad.

Uno de los detalles que hay que tener en cuenta a la hora de propagar la deformación hacia nodos vecinos, es que estos nodos vecinos pueden ya haber sido movidos en un paso anterior. Esta situación se vuelve más delicada cuando se intenta mover uno de los vértices antecesores en la recursividad, ya que provocará sobre-compensaciones indeseables.

Para evitar esto, los vértices se marcan con el nivel en que han sido afectados. Por ejemplo, el vértice que se arrastra directamente por el usuario tiene nivel 1, los vecinos

directos de éste se afectarán con nivel 2, etc. De esta manera, durante la recursión se comprobará que no se pueda mover un vértice vecino marcado con un nivel superior al actual.

La condición de parada de la recursión se ha establecido teniendo en cuenta una recursión máxima de 10 niveles, y un desplazamiento mínimo de 1 milímetro.

```
FUNCION ElasticMove (vertice, vector_movimiento, factor_elasticidad, nivel)
COMENZAR
        //Condición de parada
        SI (Longitud(vector_movimiento) < 0.001) SALIR;
        SI(nivel > 10) SALIR;

        //Marcar vértice con el nivel actual
        MarcarNivel(vertice, nivel);

        PARA CADA vecino EN Vertices_Vecinos
                //Comprobar si se debe mover el nodo vecino
                SI (DevolverNivel(vecino) < nivel) CONTINUAR;

                //Calcular vector de movimiento del vecino
                vector_movimiento_vecino = Vector(vecino, vertice)
                distancia_original = Longitud(vector_movimiento_vecino)

                vector_movimiento_vecino+= vector_movimiento
                nueva_distancia = Longitud(vector_movimiento_vecino);

                //Calcular incremento de distancia
                delta = distancia_original – nueva_distancia;

                distancia = delta * factor_elasticidad
                EstablecerLongitud (vector_movimiento_vecino, distancia);

                //Recursión
                ElasticMove(vecino, vector_movimiento_vecino, factor_elasticidad, nivel + 1)
        FIN PARA

        MoverVertice(vertice)
FIN
```

Listado 5.2. Pseudocódigo de la función recursiva para la deformación de mallas

Como posible mejora futura se podría considerar la realización de una versión iterativa del algoritmo, por ejemplo mediante una lista dinámica a la que se fueran añadiendo los nuevos vecinos encontrados. De esta manera se podría hacer una gestión del detalle de la deformación para no demorar en exceso la velocidad de generación de imágenes.

Deformación elástica o plástica del material

Hasta ahora se ha supuesto que el material del que está compuesto el objeto virtual es de naturaleza plástica, de tal forma que una vez modificada su geometría ésta se hace permanente.

No obstante se ha considerado la posibilidad de que éste material pueda recuperarse, teniendo un comportamiento elástico. Esta característica no permite el moldeado del objeto, sin embargo es muy interesante desde el punto de vista de la simulación y del retorno háptico.

Para hacer esto posible, hay que tener en cuenta que el objeto debe tener "memoria" para saber cuál es su estado original. Esto se ha conseguido introduciendo métodos para hacer una copia de los vértices y para restaurarlos más tarde.

La forma de saber cuándo restaurar un vértice a su posición original, es marcando en cada iteración aquellos que son modificados por la acción del usuario. Los que queden sin modificar, se restaurarán a su posición original de forma progresiva si no lo están ya.

Como posible mejora futura, se podrían considerar materiales que tuvieran componentes elásticas y plásticas a la vez, tal y como suele ocurrir en la realidad. De esta manera un objeto tendría un comportamiento elástico dentro de unos límites, y fuera de ellos las modificaciones serían permanentes.

Detección de colisiones

El algoritmo de detección de colisiones es de vital importancia para el funcionamiento del retorno de fuerzas y para poder interaccionar con el objeto.

Existen algoritmos muy rápidos basados en aproximación de objetos por polígonos, como cajas o esferas [Bur03], pero la información que aportan es muy limitada y poco precisa.

Este algoritmo debe calcular cuál es la cara o vértice más cercano a cada uno de los dedos de la mano del usuario, la distancia a la que se encuentra y además si está dentro o fuera del objeto.

En primer lugar se aproximará el dedo del usuario por un punto en el espacio localizado en el centro de su yema, al que llamaremos *punto objetivo*. A continuación se recorren todos los vértices de la malla, calculando la distancia de cada uno de ellos al punto objetivo. La distancia de un punto a otro es muy rápida de calcular, pero además se puede acelerar si se tiene en cuenta la distancia cuadrática (sin realizar la raíz cuadrada).

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

La distancia de un punto a una cara es más costosa de calcular. Por ello no se recorrerán todas las caras, sino solamente aquellas caras vecinas al vértice más cercano. Esta es una aproximación que se ha adoptado por eficiencia, ya que podría haber una cara más cercana al punto objetivo que no sea vecina del vértice más cercano (Figura 5.14).

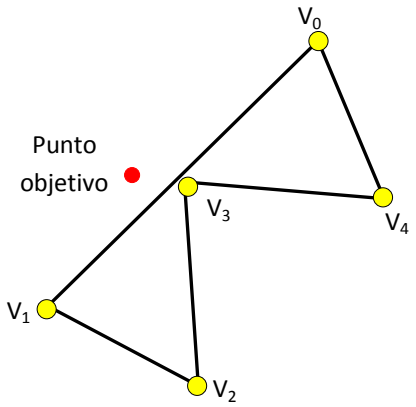


Figura 5.14. Situación de fallo para el cálculo de la cara más cercana.

Como se puede ver, el punto objetivo (marcado en rojo) está más cerca del vértice V_3, y de la cara situada entre V_0 y V_1, que no es vecina de V_3. Aquí por tanto el algoritmo fallaría diciendo que la cara más cercana es la comprendida entre V_3 y V_2 o entre V_3 y V_4.

De todas maneras esta situación no se dará en condiciones normales, y en el caso de que se produzca su efecto será muy leve.

Una vez calculada cuál es la cara más cercana, se comprobará si el punto objetivo está dentro o fuera del objeto, y si está fuera se devolverá la distancia cambiada de signo. Esto será útil más tarde para poder calcular el retorno háptico.

Regiones interiores de un objeto

Para saber si un punto objetivo está dentro o fuera de un objeto, se usa la cara más cercana al punto objetivo y su normal. En primer lugar se calcula el vector que va desde el centro de la cara hasta el punto objetivo, y el ángulo que forma este vector con respecto a la normal. Si este ángulo es mayor que 180º se puede considerar que el punto está dentro del objeto. Para simplificar los cálculos, en lugar de calcular el ángulo se hace un producto escalar entre ambos vectores, y si éste es negativo, el punto objetivo está dentro.

En la siguiente figura (Figura 5.15) se ha representado una sección de un objeto en la que se ven cinco caras, sus vectores normales y parte de las regiones interiores de cada una de ellas. Este objeto tiene tanto zonas cóncavas (ángulo formado por **C_3** y **C_4**) como convexas. Es necesario apuntar que una zona se considera cóncava cuando el ángulo interno (medido desde el interior del polígono) es mayor de 180 grados, en caso contrario se considera convexa.

Cuando dos caras forman un ángulo de menos de 90º, hay que tener cuidado con aquella que se selecciona como más cercana. Un punto objetivo situado en las regiones **A**, **B** o **C** tiene a la misma distancia las caras **C_1** y **C_5**, sin embargo en la región **A** elegir la cara **C_5** como la más cercana supondría para el algoritmo que es un punto interior del objeto, lo cual es erróneo. Lo mismo pasaría con un punto situado en la región **C**, siendo necesario para el algoritmo que se elija como más cercana la cara **C_5**.

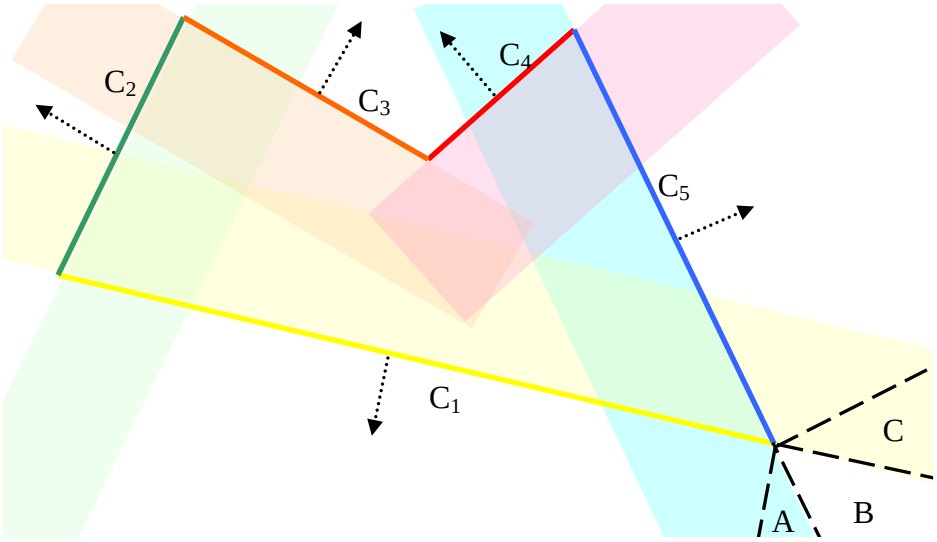


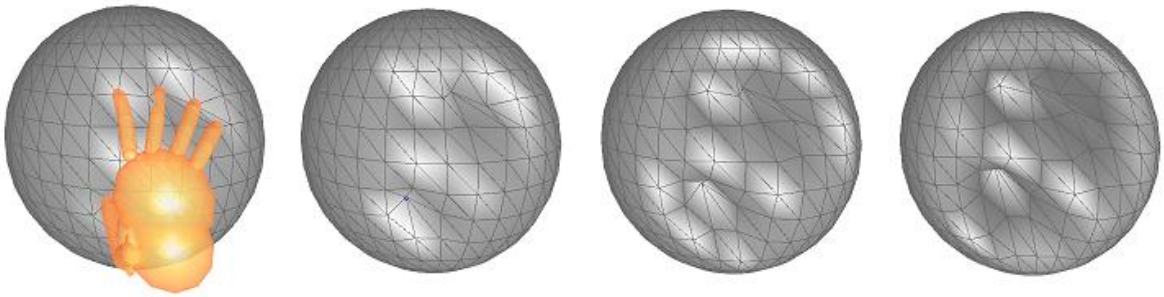
Figura 5.15. Representación gráfica de las regiones interiores de cada cara.

Para evitar esto, si hay dos o más caras a la misma distancia durante la búsqueda de la cara más cercana, se elegirá preferentemente la que suponga que es un punto exterior del objeto.

Deformación activa

Se ha llamado deformación activa a la capacidad de moldear una superficie simplemente haciendo presión con los dedos. Esta característica se ha implementado a partir de la detección de colisiones y del cálculo de la región interior de un objeto.

Lo único que hay que tener en cuenta al realizarla, es que la yema del dedo se encuentre dentro del objeto virtual. Si esto es así, se realizará una depresión en el material haciendo uso de la deformación de mallas.



5.16. Deformación activa de la superficie de un objeto.

Cálculo de la física

El cálculo de la física se gestiona en la clase principal de la aplicación *ModelerApp*. Su misión es la de coordinar las actividades de cálculo de colisiones, deformación activa de la malla, y retorno háptico, sonoro y visual.

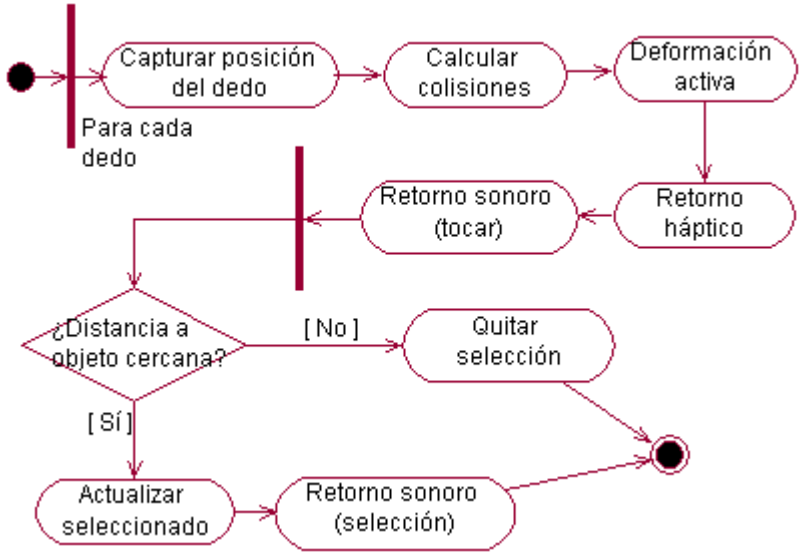


Figura 5.17. Diagrama de flujo para el cálculo de la física.

Como se puede ver en el diagrama de flujo (Figura 5.2) las colisiones se detectan independientemente para cada dedo, y en base a ellas se calcula la deformación de la superficie del objeto, se calcula el retorno háptico y sonoro, y se hace la selección de vértices o caras. Para esto último solamente se tiene en cuenta la posición del dedo pulgar, ya que durante el gesto de "coger" es el que se mantiene más inmóvil.

5.2.4 Requisitos de la aplicación

Software

Para el desarrollo de la aplicación se ha utilizado la versión 2.0.3 de VR Juggler, que se puede descargar libremente de su web y además incluye un instalador.

Es necesario comentar que una vez instalado deben establecerse las variables globales *SNX_BASE_DIR, VJ_BASE_DIR* y *VJ_DEPS_DIR* con la ruta del directorio en el que se encuentra VR Juggler.

La aplicación *vrjConfig*, incluida en la instalación de VR Juggler, necesita por su parte la instalación del JDK de Java versión 1.4 o posterior.

En cuanto al sistema operativo se ha usado Windows XP SP2, y como herramienta de desarrollo, Microsoft Visual Studio 2005, contando ambos con un soporte completo de VR Juggler.

Hardware

El equipo utilizado es una estación de trabajo HP con un procesador Intel Pentium 4 a 2.4 Ghz y 512 MB de RAM.

La tarjeta gráfica es una NVIDIA Quadro4 900 XGL con 128 MB de memoria. Esta tarjeta gráfica tiene capacidad para generar gráficos 3D en estéreo mediante dos salidas VGA, lo cual es un requisito indispensable. A estas salidas gráficas estará conectado el sistema de retroproyección estereoscópico que será el que represente la escena virtual.

Conectado al PC mediante el puerto serie estará el sistema de localización Ascension Flock of Birds, y por último el guante háptico creado se conectará al puerto paralelo y al puerto USB.

CAPÍTULO 6. PUESTA EN MARCHA DEL SISTEMA

En este capítulo se describirá el uso de la aplicación y del guante de datos háptico, y se recogerán algunas de las impresiones que ha tenido en los usuarios que lo han probado, así como diversos problemas relativos a la tecnología usada.

6.1 PREPARACIÓN INICIAL

Uno de los aspectos que tienen en común los sistemas de Realidad Virtual inmersivos, es la preparación inicial que se necesita para poner el sistema en marcha y poder usarlo. Es por ello que durante el desarrollo del mismo se ha aprovechado la capacidad de VR Juggler para trabajar con dispositivos simulados para usar la aplicación mediante el ratón y el teclado. Para ello ha sido necesario definir un fichero de configuración para el modo inmersivo y otro para el no inmersivo que se llaman "modeler.jconf" y "sim.modeler.jconf" respectivamente.

A continuación se describirá cuáles son los pasos a seguir para usar la aplicación en su modalidad inmersiva, ya que la otra tiene un propósito de depuración.

Se supondrá que ya está instalado VR Juggler, así como la aplicación de modelado, y que todos los dispositivos están conectados. Esto es, el sistema de localización FOB conectado al puerto serie, el guante de datos háptico conectado al puerto paralelo y USB, y los dos proyectores del sistema de retroproyección conectados a las dos salidas gráficas del ordenador.

El primer paso es encender los dos proyectores. Para comprobar si están correctamente calibrados, se puede mostrar una misma imagen en cada uno de ellos, y ver que coinciden en la pantalla de retroproyección, pareciendo que sólo hay una. Si se viera borrosa, o doble, sería necesario un ajuste de los mismos (véase anexo B).

El siguiente paso consiste en conectar el sistema de localización Flock of Birds y situar su emisor correctamente. Como se vio en el apartado 5.1.1, esta posición es encima de una mesa a un metro de altura sobre el suelo, coincidiendo con el margen inferior de la pantalla, y retirado de ésta 65 centímetros. La mesa puede ser también una caja de cartón o similar, pero es importante que no sea metálica, ya que alteraría los campos electromagnéticos de FOB.

Figura 6.1. Colocación del emisor de FOB.

Es el momento de vestirse el guante de datos. Este paso es uno de los que más tiempo requieren, y resulta útil la ayuda de una segunda persona, sobre todo cuando no se tiene práctica. El usuario se pondrá en primer lugar el guante de lycra, ajustando los accesorios vibradores a la yema de cada dedo. A continuación fijará la caja de control del guante sobre su muñeca, usando para ello la tira de Velcro acoplada al mismo. Por último se colocará el sensor de localización de FOB correspondiente haciendo uso también del Velcro que llevan ambos. Una vez que todo está colocado, se activará el interruptor de la unidad de control del guante y su LED de estado rojo brillará.

El último paso antes de poder empezar a usar la aplicación es colocarse las gafas con filtros polarizados, que deben llevar acoplado el sensor de localización de la cámara a la patilla derecha. Si hay más espectadores se podrán repartir gafas a todos ellos, de esta manera también podrán contemplar los objetos modelados con sensación de profundidad.


Figura 6.2. Sensor de localización en las gafas polarizadas.

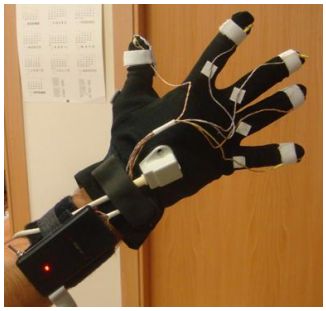


Figura 6.3. Sensor de localización en el guante con retorno háptico.

Antes de lanzar la aplicación resulta útil comprobar la correspondencia de los filtros de polarización de las gafas con los de los proyectores, sobre todo si se ha realizado algún cambio. Esto se realiza de forma sencilla mostrando una aplicación diferente en cada salida gráfica y tapando un ojo del usuario para asegurarse que ve lo que debería. Si no es así, será necesario girar 90º el filtro de polarización de uno o de ambos proyectores.

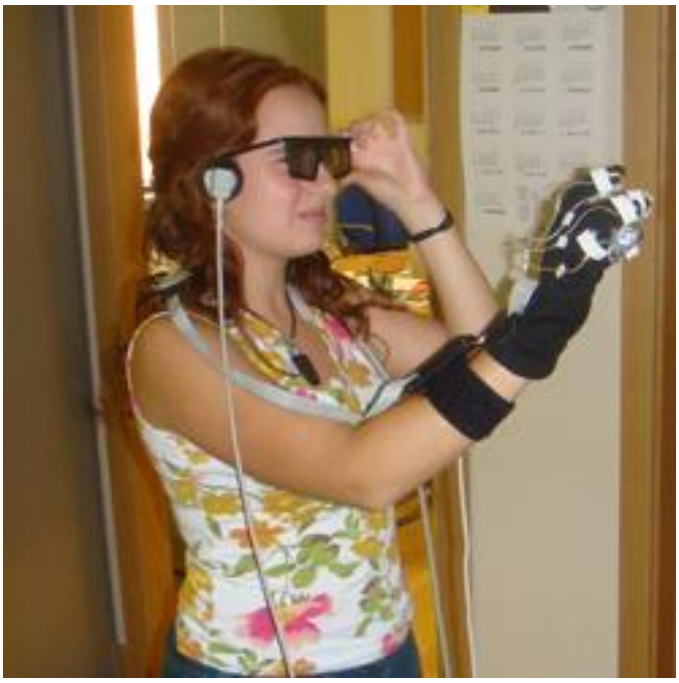


Figura 6.4. Usuario con el guante, gafas y sensores colocados.

6.2 USO DEL SISTEMA INMERSIVO

Ahora que ya está todo configurado y listo para usar, se lanzará la aplicación pasándole como argumento el nombre del fichero de configuración usado, en este caso, "modeler.jconf". La consola aparecerá mostrando información relativa a su inicialización, y comenzará la experiencia inmersiva.

Durante la inicialización, el driver del guante háptico hará un testeo activando sucesivamente cada uno de los vibradores. El usuario notará cómo esta vibración suave recorre cada uno de sus dedos, que se podrá ver visualmente gracias los diodos LED situados en la caja de control colocada en la muñeca.

Lo primero que verá el usuario en la pantalla de proyección es un objeto virtual con forma de cara (aunque se puede configurar para cargar cualquier otro). Gracias a la estereoscopía, se puede notar la sensación de que el objeto "flota" fuera de la pantalla y que realmente tiene profundidad, distinguiéndose sobretodo que el cuello y las orejas del modelo quedan mucho más atrás que su nariz.

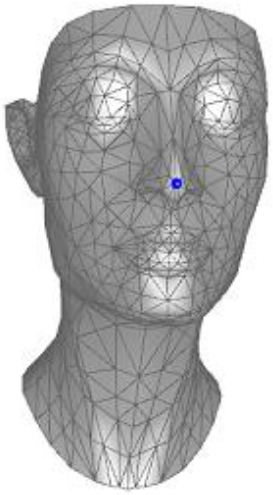


Figura 6.5. Objeto virtual mostrado al iniciar la aplicación.

Inicialmente el objeto virtual tendrá iluminado el vértice más cercano al usuario, en este caso, situado en la nariz (Figura 6.5). El usuario deberá intentar coger este vértice con sus dedos índice y pulgar, en el lugar que a él le parezca que está flotando. Esto se realiza para la calibración del sensor de localización del guante, ya que su posición en la escena virtual puede variar ligeramente dependiendo de la sensación de profundidad que produzca en el usuario, de pequeños errores a la hora de colocar los sensores, etc.

Una vez que se intenta coger el vértice, la aplicación pasa al modo de modelado. A partir de este momento el usuario, o un ayudante, pueden lanzar cuando deseen el menú mediante la tecla F1. Se activará en este momento la consola, que mostrará un listado de las distintas opciones de configuración (Figura 6.6).

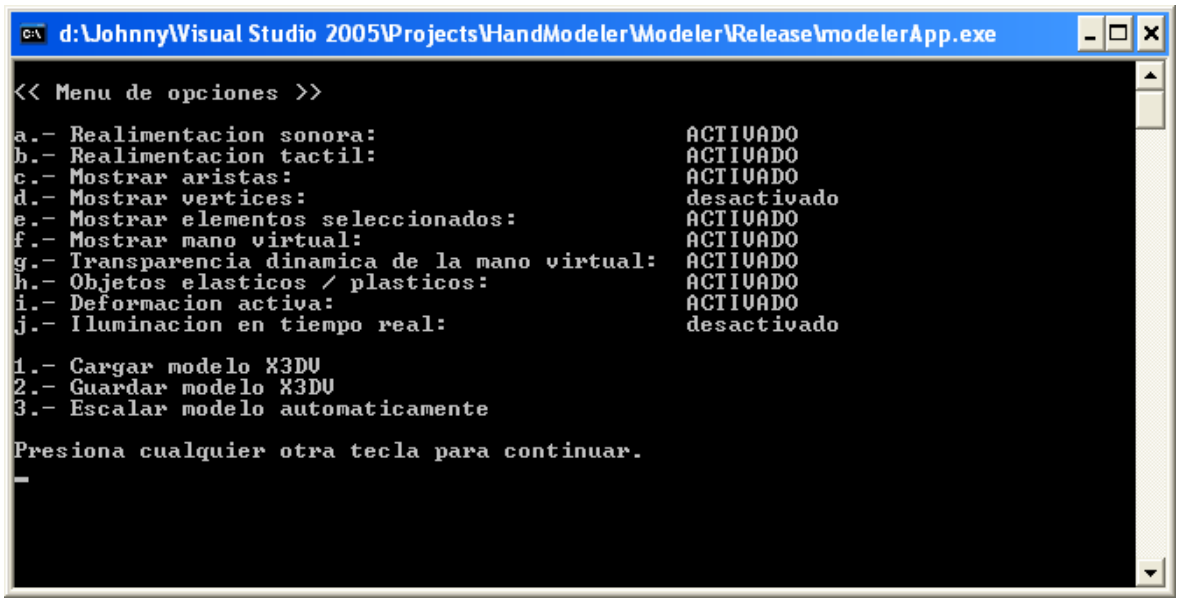


Figura 6.6. Menú de opciones de la aplicación de modelado.

Para activar o desactivar cada una de las opciones se pulsará la tecla correspondiente, lo que hará que se actualice el modelo y se vuelva a mostrar el menú. Para salir de él basta con pulsar cualquier otra tecla, por ejemplo, espacio.

Las opciones que pueden ser configuradas son:

- **Realimentación sonora**. Si está activa se reproducirán sonidos al tocar una superficie o seleccionar una cara o vértice.

- **Realimentación táctil**. Se usa para activar o desactivar la vibración en el guante háptico.

- **Mostrar aristas**. Si se activa se dibujarán las aristas que componen la malla del objeto virtual sobre el mismo.

- **Mostrar vértices.** Como la opción anterior, dibujará puntos de color rojo sobre cada uno de los vértices que componen la malla del objeto virtual.

- **Mostrar elementos seleccionados.** Si está activa resaltará el vértice o la cara seleccionada.

- **Mostrar mano virtual.** Permite desactivar la representación gráfica de la mano virtual.

- **Transparencia dinámica de la mano virtual.** Esta opción hace que la mano se haga transparente cuando se acerca a un vértice o cara, para que no lo tape y moleste en su manipulación.

- **Objetos elásticos/plásticos.** Define el comportamiento del objeto virtual mostrado. Si está activa, los objetos tendrán un comportamiento elástico, recuperando su forma después de ser deformados. En caso contrario los objetos tendrán un comportamiento plástico, y una vez deformados permanecerán así.

- **Deformación activa.** Si está activa el objeto se podrá deformar con todos los dedos al hacer presión sobre él. Si está desactivada, la única forma de poder deformar el objeto es seleccionando un vértice o cara y estirando de él.

- **Iluminación en tiempo real.** Si está activa se calcula la iluminación de cada cara mientras se deforma el objeto, lo que conlleva numerosos cálculos de los vectores

normales de cada cara y vértice. Si está desactivada estos cálculos se harán después de cada deformación.

- **Cargar modelo X3DV.** Permite especificar la ruta de un archivo X3DV que contenga una malla para cargarla en la aplicación.

- **Guardar modelo X3DV.** Al elegir esta opción se pedirá un nombre de archivo, que puede contener su ruta, para guardar el modelo actual en un archivo X3DV.

- **Escalar modelo automáticamente.** Este comando realizará un escalado proporcional del objeto virtual cargado, ajustándolo al espacio de trabajo del usuario.

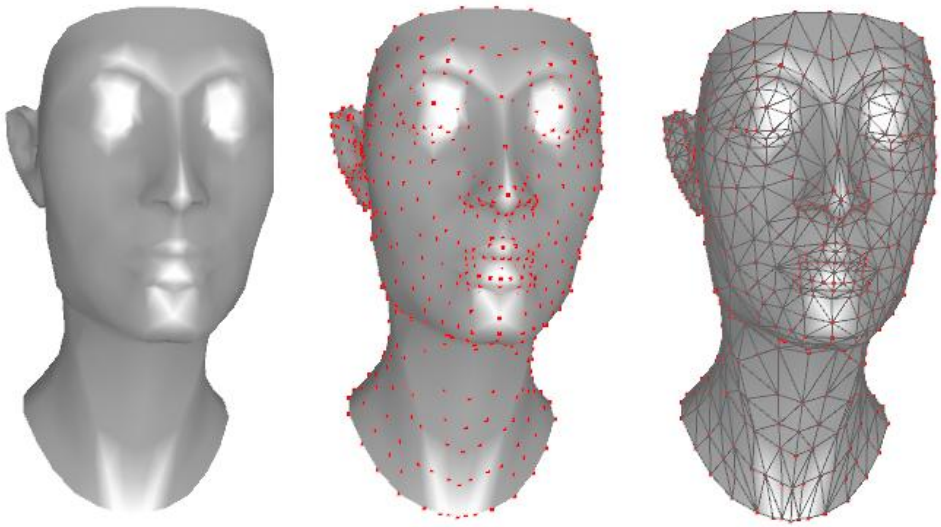


Figura 6.7. Distintos modos de visualización del objeto virtual.

Una vez que se han elegido las opciones deseadas, se ocultará la consola para que no moleste haciendo click con el ratón sobre la ventana principal de la aplicación, o cambiando con el teclado mediante *Alt+Tab*.

Ahora el usuario puede interaccionar con el modelo libremente, cambiando su geometría y deformándolo con las manos como si realmente lo estuviera tocando. Por ejemplo puede cogerlo con los dedos índice y pulgar y estirar, o cambiar su superficie haciendo presión con los dedos.

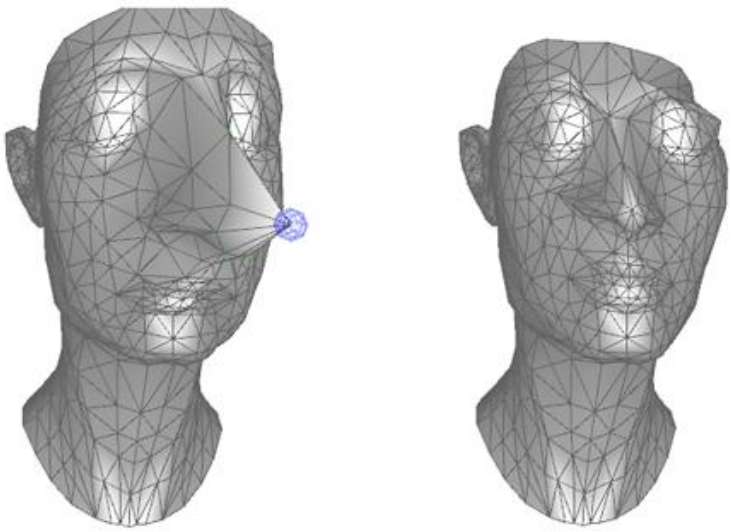


Figura 6.8. Deformaciones producidas al mover un vértice (izquierda) y presionar con los dedos (derecha).

Mientras el usuario está trabajando en el modelado, será capaz de sentir la superficie del objeto gracias a la vibración independiente de cada uno de sus dedos. Esto le ayudará a conocer cuando lo está tocando, y también la fuerza que está ejerciendo sobre el mismo.

Si la realimentación sonora está activada, también se escuchará un sonido cada vez que toque una superficie, o cambie la selección de un objeto. Esto ayudará a sentir que el objeto está realmente ahí.

Para jugar con las distintas opciones, se recomienda activar y desactivar la característica elástica del material. Así, si está activada se verá como al estirar de la superficie ésta recupera su forma progresivamente. Si el modelo cargado es la cara, parecerá que se estira su piel y que, tal y como ocurre en la realidad, recupera su forma progresivamente al soltarla. En cambio si se desactiva esta opción, la sensación que tendrá el usuario es la de estar manipulando un gran trozo de plastilina, en la que puede introducir los dedos y darle la forma que desee.

Durante el modelado del objeto, el usuario puede cambiar su posición para observarlo desde distintos ángulos, lo que ayuda a examinar los cambios producidos en él. También puede acercarse o alejarse de él, pero como el espacio de trabajo es reducido, existe la posibilidad de acercar o alejar el objeto virtual de sí mismo mediante los gestos del guante háptico. Así, el usuario puede traer al frente el objeto juntando los dedos pulgar y corazón, y enviarlo hacia atrás con los dedos pulgar y anular.

Por último, después de estar un rato probando con distintas deformaciones puede que el objeto quede tan cambiado que no se reconozca, y se desee empezar de nuevo. Para esta situación existe la posibilidad de restablecer el modelo cargado juntando los dedos pulgar y meñique, lo que hará que vuelva a su estado original.

6.3 PROBLEMAS ENCONTRADOS

Uno de los problemas encontrados en la ejecución de la aplicación es relativo a un bug en la gestión de dispositivos de VR Juggler que hace que no se apaguen correctamente las unidades electrónicas de FOB al finalizar la aplicación. Debido a esto, es necesario apagarlas y volverlas a encender cada vez que se quiera lanzar una nueva ejecución.

Este problema ha sido solucionado añadiendo código en el destructor del programa para hacer de forma manual que el dispositivo deje de muestrear datos. De esta manera se puede comprobar cómo se apagan las luces de estado al finalizar la aplicación, y a pesar de que VR Juggler hace saltar una excepción con la próxima ejecución (debido a que FOB se encuentra en un estado inesperado), ésta se puede ignorar y todo funcionará correctamente.

Otro de los problemas que han aparecido durante las pruebas, también referente a FOB, es la pérdida de sincronización del driver con el dispositivo. Esto sucede cuando en la aplicación se realiza demasiado procesamiento, por ejemplo, a la hora de calcular deformaciones o colisiones. A pesar de que la velocidad de generación de imágenes siga siendo lo suficientemente alta como para no notar ninguna bajada de rendimiento, el planificador de VR Juggler es incapaz de asignar el suficiente tiempo de ejecución al hilo que controla los dispositivos, causando errores de lectura.

Este problema se ha solucionado limitando los cálculos realizados en el proceso de cálculo de la física. Por ejemplo, se ha reducido el nivel de recursividad máximo a la hora de calcular las deformaciones en la malla, y se ha aumentado el periodo del algoritmo PWM que gobierna el guante háptico, aún a pesar de que se degrade la efectividad del mismo. También se ha añadido una opción en el menú para desactivar el cálculo de la iluminación en tiempo real mientras se producen las deformaciones (lo que implica el cálculo de los vectores normales de cada cara y arista).

Otra solución podría consistir en usar un ordenador con capacidades de procesamiento de múltiples hilos de ejecución, como un procesador multinúcleo, o portando el código a la nueva versión de VR Juggler aparecida recientemente.

Por último, se ha comprobado que VR Juggler realiza una conversión de unidades indebida al tratar la información de configuración creada con vrjConfig, pasando de metros a pies. Esto se realiza aún especificando en la aplicación que se usan metros mediante *getDrawScaleFactor()*, tal y como se especifica en el manual, por lo que la solución adoptada ha sido aplicar una conversión de pies a metros a las unidades de longitud especificadas en el archivo de configuración.

6.4 OPINIÓN DE LOS USUARIOS

Además del propio desarrollador, algunos usuarios han probado la aplicación creada, dando su opinión al respecto.

Figura 6.9. Usuario estirando de uno de los vértices del objeto virtual.

Durante la colocación de los distintos dispositivos, los usuarios suelen coincidir en que es una tarea laboriosa, y que en ese aspecto las aplicaciones de escritorio son mucho más usables, ya que se pueden comenzar a utilizar sin necesidad de preparación previa.

Una vez lanzada la aplicación, la sensación que produce es en un primer momento de asombro, al ver un objeto virtual que parece que sale de la pantalla. Seguidamente los usuarios intentan de forma intuitiva alcanzar el objeto y tocarlo.

Apenas han sido necesarias unas breves indicaciones para que los usuarios comenzaran a explotar todas las características que ofrece el sistema. La manera de deformar objetos les ha parecido muy intuitiva a la mayoría, ya que se asemeja a la forma en la que se hace en realidad, siendo los gestos para acercar, alejar o restablecer el objeto lo único que han tenido que aprender.

Uno de los problemas que han surgido durante las pruebas, y que ha sido comentado por más de un usuario, es que la representación de la mano virtual ocluye en ciertos casos una parte importante del objeto, haciendo más difícil la tarea de seleccionar un vértice o una cara. Este problema ha sido tenido en cuenta en una revisión posterior de la aplicación,

y se ha solucionado haciendo que la mano virtual se vuelva más transparente cuanto más cerca está del objeto que se está modelando. Esta característica se ha llamado "transparencia dinámica de la mano virtual" y puede ser activada o desactivada desde el menú.

Respecto al retorno háptico, los usuarios han comentado que resulta útil para saber cuándo se toca un objeto, además de crear una ilusión de que el objeto es tangible. Sin embargo, alguno de ellos ha sugerido que la intensidad de vibración debería ser menor, por lo que más tarde se han regulado las resistencias variables del circuito para reducirla.

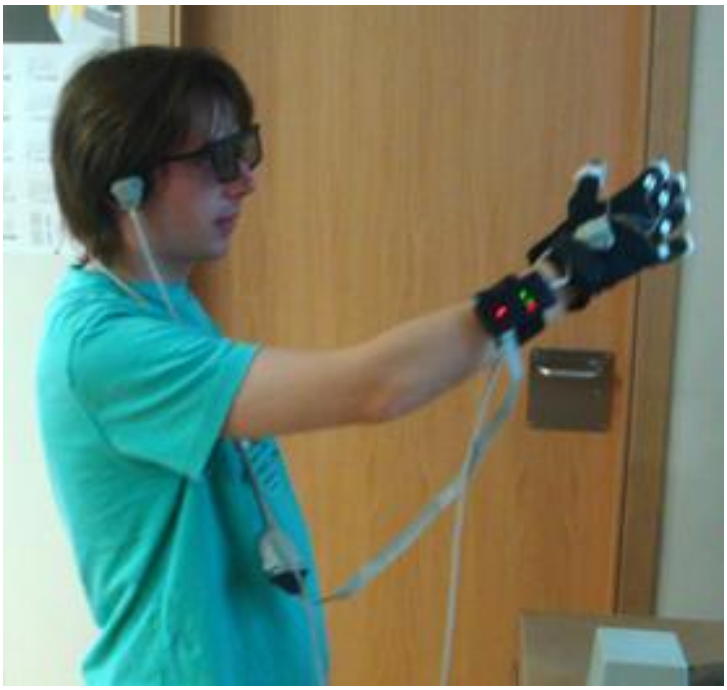


Figura 6.10. Usuario tocando el objeto virtual con el retorno háptico activado.

El problema más común, y que han padecido casi todos los usuarios, es el cansancio en el brazo al cabo de un rato de uso de la aplicación. Esto se debe en parte al cableado necesario tanto para el sensor de localización como para el guante, y en parte también por la altura a la que se encuentra la pantalla de retroproyección, obligando a levantar los brazos a la altura de los ojos.

CAPÍTULO 7. CONCLUSIONES Y TRABAJO FUTURO

A continuación se realizará un repaso del trabajo realizado, y se comprobará qué objetivos de los propuestos inicialmente han sido cumplidos. Finalmente se describirán posibles ampliaciones del trabajo realizado, de tal manera que puedan servir como motivación para trabajos futuros.

7.1 CONCLUSIONES

El primer objetivo propuesto en el inicio del proyecto, es la construcción de una aplicación inmersiva que permita el modelado de objetos virtuales de forma intuitiva.

El sistema desarrollado permite dos maneras diferentes de modelar un objeto virtual: seleccionando y estirando de vértices o caras, o simplemente haciendo presión con la yema de los dedos para darle forma. Además se ha implementado un algoritmo de deformación de mallas, de tal manera que las deformaciones producidas sean mucho más realistas. Para experimentar con los distintos comportamientos de los materiales, y probar así también las características del retorno háptico, se han definido dos comportamientos diferentes de los materiales: elásticos y plásticos. Los primeros recuperan su forma tras ser sometidos a una deformación, mientras que en éstos últimos las deformaciones son siempre permanentes.

Respecto al aspecto intuitivo, durante las pruebas se ha podido comprobar cómo los usuarios manipulaban el objeto presentado satisfactoriamente sin apenas requerir explicaciones ni entrenamiento previo.

La aplicación permite ser utilizada bajo diferentes configuraciones de hardware, gracias al uso de la abstracción de los dispositivos. En particular se ha creado un archivo de configuración para el uso del sistema de retroproyección estereoscópico, y del sistema de localización Flock of Birds.

En cuanto al soporte de la aplicación para el uso de un dispositivo de retorno táctil, se ha logrado gracias a la implementación de un algoritmo para la detección de colisiones, y a la definición de una función de retorno de esfuerzo.

El último subobjetivo relativo a la aplicación hacía referencia a la posibilidad de cargar objetos en la aplicación, y una vez modificados ofrecer también la opción de guardarlos. Esto se ha hecho siguiendo el estándar X3DV (compatible con el antiguo VRML).

El segundo objetivo principal propuesto era la construcción de un guante con capacidad para transmitir un retorno háptico hacia el usuario.

El guante construido está formado por vibradores que no sólo proporcionan un retorno táctil independiente para cada dedo, sino que además se pueden regular en intensidad, y se pueden calibrar independientemente gracias a los potenciómetros incluidos.

Estos vibradores se han montado sobre tiras elásticas con Velcro, a las que se les ha añadido una zona conductora. De esta manera se consigue que se puedan usar en cualquier otro guante fácilmente, y además reconocer cuando se hace contacto entre dos o más dedos.

Todos los componentes que forman parte del guante pueden ser encontrados fácilmente a un precio asequible, por ejemplo, los vibradores se pueden encontrar en móviles en desuso, y también en tiendas de repuestos por menos de 2 euros. El coste aproximado de todos los materiales no supera los 40 €.

7.2 TRABAJO FUTURO

Con todos los objetivos cumplidos satisfactoriamente, se podría decir que el sistema implementado es un éxito, y que promete ser una interfaz con mucho futuro. Sin embargo forma parte de una forma de interacción con los ordenadores relativamente joven comparada con otras interfaces tradicionales, por lo que queda aún mucho trabajo por hacer.

Uno de los esfuerzos que se podrían hacer el futuro para mejorarlo podría estar encaminado a resolver el mayor problema que han encontrado los usuarios durante su uso: el cansancio. Y es que son necesarios numerosos cables que, además de pesar, limitan la libertad de movimientos del usuario. Otro factor que influye es la posición erguida del usuario frente a la pantalla de retroproyección, que exige también que el usuario trabaje con las manos a la altura de sus ojos.

Esto se podría solucionar reduciendo el número y grosor de los cables necesarios, tal vez unificando los dispositivos, o mediante la utilización de interfaces inalámbricas. También sería ventajoso el uso de algún dispositivo de visualización que permita al usuario trabajar con las manos en una posición más natural. Un sistema adecuado para conseguirlo podría ser una CAVE, que no es más que un conjunto de pantallas de retroproyección que envuelven al usuario.

Siguiendo con el apartado del hardware, sería posible mejorar la experiencia del usuario aprovechando la facilidad de acoplar los estimuladores vibrotáctiles a un guante de datos que sea capaz de medir la flexión y abducción de los dedos. Esta posibilidad se ha tenido en cuenta durante el desarrollo de la arquitectura de la aplicación, ya que el modelo de mano virtual está preparado para interpretar tal información. Gracias a esta nueva característica el usuario podría hacer deformaciones más precisas de los objetos.

Por otro lado, en el sistema desarrollado el usuario hace uso de una única mano para modelar el objeto. Esto se podría complementar con la utilización de otro guante de datos en la otra mano para rotar y mover el objeto durante su modelado, por ejemplo.

En cuanto a la aplicación en sí, son también muchas las posibilidades que se abren para poder ampliar su funcionalidad.

Una de ellas podría ser la representación del menú de opciones en la propia escena virtual, de tal manera que fuera posible cambiarlas con el guante de datos como si se tratara de botones.

Siguiendo esta línea, también sería interesante la introducción de "herramientas virtuales" que permitan trabajar en el modelado de la misma forma que se usan herramientas para moldear la plastilina en el mundo real. Éstas podrían ser útiles para crear pequeños detalles, realizar incisiones, dar una forma plana, etc.

Respecto al sentido del tacto, sería útil la introducción de nuevos sistemas que permitan una realimentación hacia el usuario –como el uso de un dispositivo con un brazo articulado-, y hacer una evaluación de qué técnicas son más útiles para la realización de algunas tareas establecidas de antemano. También sería interesante introducir en esta comparativa otro tipo de realimentación más tradicional como la realimentación visual, o la sonora, que ya se encuentran implementadas en la aplicación.

BIBLIOGRAFÍA

Libros y artículos

[Bar95] Barfield, W. y Weghorst, S. (1993). "*The sense of presence within virtual environments: A conceptual framework*", Human-computer interaction: Software and hardware interfaces, pp. 699-704, Elsevier.

[Bow01] Bowman, D.A.; Kruijff, E.; LaViola Jr., J.J. y Poupyrev, I. (2001). "*An Introduction to 3-D User Interface Design*", Presence, Vol. 10, pp. 96-108.
http://people.cs.vt.edu/~bowman/papers/3dui_presence.pdf

[Bur03] Burdea, G.C. y Coiffet, P. (2003). "*Virtual Reality Technology*", IEEE Press, 2nd edition.

[Hay04] Hayward, V.; Astley, O.R.; Cruz-Hernandez, M.; Grant, D. y Robles-De-La-Torre, G. (2004). "*Haptic Interfaces and Devices*", Sensor Review 24, pp. 16-29

[Hea95] Hearn, D. y Baker, M.P. (1995). "*Gráficas por Computadora*", Prentice Hall.

[Kho95] Khosla, P.; Shimoga, K. y Murray, A. (1995). "*A Touch Reflection System for Interaction with Remote and Virtual Environments*", IEEE/RSJ International Conf. On Intelligent Robots and Systems.
http://www.ri.cmu.edu/pubs/pub_1790.html

[LaV04] LaViola, J.J.; Keefe, D.F.; Zeleznik, R.C. y Acevedo, D. (2004). "*Case Studies in Building Custom Input Devices for Virtual Environment Interaction*", VR 2004 Workshop: Beyond Glove and Wand Based Interaction.
http://cs.brown.edu/people/jjl/pubs/vr2004_workshop.pdf

[Lin06] Lingrand, D.; Renevier, P.; Pinna-Dery, A.; Cremaschi, X.; Lion, S.; Rouel, J.; Jeanne, D.; Cuisinaud, P. y Soula, J. (2006). "*Gestaction3D: a platform for studying displacements and deformations of 3D objects using hands*", CADUI, pp. 105-114.
http://www.i3s.unice.fr/~lingrand/Res/bib/CADUI2006.pdf

[Mol03] Molina, J.P. (2003). "*Multimedia, Hipermedia y Realidad Virtual*", Apuntes de Ingeniería Técnica en Informática de Gestión.

[Ram07] Ramos, R.; Larios, J.; Cervantes, D. y Leriche, R. (2007). "*Creación de ambientes inmersivos con software libre*", Revista Digital Universitaria. http://www.oei.es/noticias/spip.php?article823

[Rob06] Robles-De-La-Torre, G. (2006). "*The importance of the Sense of Touch in Virtual and Real Environments*", IEEE Multimedia, Vol. 13, pp. 24-30.

[Swa06] Swapp, D., Pawar, V., Loscos, C. (2006). "*Interaction with co-located haptic feedback in virtual reality*", Springer-Verlag.

Enlaces de Internet

[Browning] The Glove. Guante de datos creado por Cameron Browing. http://www.browningglove.com/

[Cyberglove] Guante de datos Immersion Cyberglove. http://www.immersion.com/3d/products/cyber_glove.php

[Dataglove] Guante de datos 5DT Dataglove 5. http://www.5dt.com/hardware.html#glove

[Diverse] Diverse – Device Independent Virtual Environments – Reconfigurable, Scalable, Extendable. http://diverse-vr.org/

[Essential] Guante de datos EssentialReality P5. http://www.alliancedistributors.com/Alliance_Brand/Products.php

[HapticR] Principles of Haptic Rendering for Virtual Environments. http://network.ku.edu.tr/~cbasdogan/Tutorials/haptic_tutorial.html

[Inpout32] Logix4f. Inpout32.dll. Librería para el acceso al puerto paralelo. http://www.logix4u.net/inpout32.htm

[Mellot] HomeBrew VR devices. Página de Kevin Mellot.
http://www.geocities.com/mellott124/

[Merlier] Guante de datos creado por Bertrand Merlier.
http://tc2.free.fr/Merlier/english/Cdv1.html

[OpenGL] OpenGL. API para la creación de gráficos en 3D.
http://www.opengl.org/

[OpenSG] OpenSG. Librería para la construcción de grafos de escena.
http://opensg.vrsource.org

[OpenSce] OpenSceneGraph. Librería para la construcción de grafos de escena.
http://www.openscenegraph.org

[Peltier] Módulos de efecto Peltier.
http://aasp.org.es/ccd/pelttier/ccd2.htm

[Pinch] Guante de datos Fakespace Pinch.
http://www.fakespace.com/pinch.htm

[Pierre] Proyectos de Pierre Alliez que incluyen 3D Toolbox 1.0
http://www.codeproject.com/script/articles/list_articles.asp?userid=181

[PPort] Interfacing the Standard Parallel Port
http://www.beyondlogic.org/spp/parallel.htm

[PPort2] Conexión y programación con el puerto paralelo
http://mimosa.pntic.mec.es/~flarrosa/puerto.pdf

[PWM] Modulación PWM aplicada a motores
http://www.micromouseinfo.com/introduction/dcmotors.html

[SecondLife] Second Life: Your World. Your Imagination.
http://secondlife.com

[ULN2003] Hoja de características del chip ULN2003.
http://www.chipcatalog.com/ST/ULN2003.htm

[VPR] VR Juggler Portable - Runtime Programmer's Guide.
http://developer.vrjuggler.org/docs/vapor/2.0/programmer.guide/programmer.guide/

[VRJuggler] The VR Juggler Suite.
http://www.vrjuggler.org

[VRSoft] VR Software List.
http://www.diverse.vt.edu/VRSoftwareList.html

[XIST] Guante de datos noDNA X-IST Dataglove SP1 system.
http://www.x-ist.de/shop.365.0.html?&L=1

ANEXO A. LA LIBRERÍA VR JUGGLER

A.1 INTRODUCCIÓN

VR Juggler es un framework especializado en el desarrollo de sistemas de Realidad Virtual. Esta plataforma proporciona una capa de abstracción que permite ejecutar aplicaciones bajo multitud de configuraciones hardware y software. Además es escalable, sirviendo tanto como para sistemas de sobremesa, hasta complejos sistemas inmersivos como CAVE's ejecutándose bajo un cluster de ordenadores.

Por último, VR Juggler es un sistema portable, soportando una gran variedad de sistemas operativos como: IRIX, Linux, Windows, FreeBSD, Solaris y Mac OS X.

A.2 ARQUITECTURA A ALTO NIVEL

VR Juggler actúa como elemento de unión entre los distintos componentes de los que se compone. Estos son:

- **Gadgeteer**. Es un sistema de gestión de dispositivos. Maneja su configuración, control, adquisición y representación de datos de los dispositivos de Realidad Virtual.

- **JCCL**. Es un sistema de configuración basado en XML que soporta tipos multivariable.

- **VR Juggler Portable Runtime (VPR).** Proporciona abstracciones independientes de la plataforma de hilos, sockets y primitivas de entrada/salida.

- **Sonix.** Es una abstracción a alto nivel del sistema de sonido hardware y software. Proporciona una interfaz sencilla para introducir efectos de sonido 3D en cualquier escena virtual.

- **Tweek.** Es un sistema que permite la comunicación entre distintas tecnologías como C++, Java, JavaBeans, y CORBA.

- **OpenGL.** Es una API multilenguaje y multiplataforma que permite la creación de gráficos 2D y 3D.

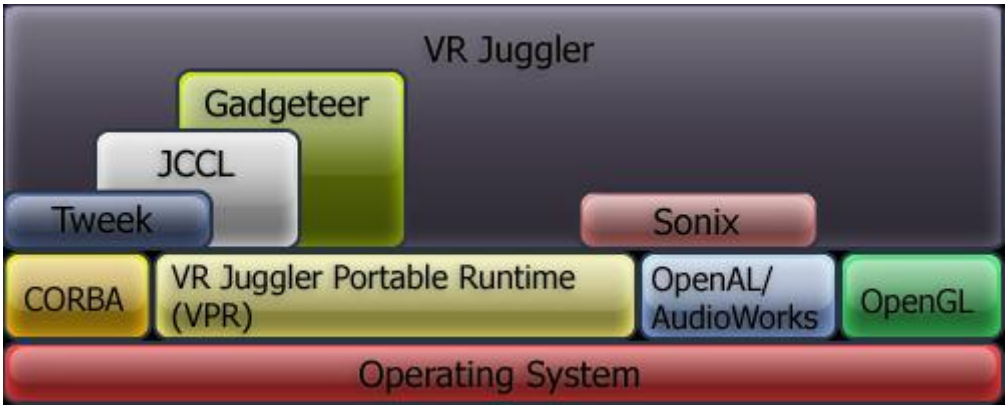


Figura A.1. Vista general de los componentes de VR Juggler.

Los componentes básicos de VR Juggler son Gadgeteer, JCCL y VPR. Además, hay otros módulos no nombrados anteriormente como "PyJuggler" y "VRJ.NET" que permiten la utilización de otros lenguajes como Python y cualquiera de la plataforma .NET respectivamente para crear las aplicaciones.

A.3 APLICACIONES PARA VR JUGGLER

A.3.1 Introducción

Las aplicaciones en VR Juggler son objetos manejados por el kernel, que es el que realmente los ejecuta. Es por eso que no tienen la tradicional entrada main(), sino que definen distintas funciones que son llamadas por el kernel, o núcleo.

El núcleo es el que se encarga de asignar tiempo de ejecución a cada uno de los hilos de cada una de las aplicaciones que esté ejecutando, haciendo de planificador.

Esta arquitectura de aplicaciones como objetos tiene varias ventajas. La más importante es que es el núcleo el responsable de coordinar la ejecución de los distintos componentes del sistema de Realidad Virtual. No ocurre así en el esquema tradicional en el que se define un punto de entrada mediante una función main() y se llama a las funciones de la librería cuando es necesario, ya que la librería sólo se ejecuta cuando es llamada.

Otra ventaja es la posibilidad de hacer cambios de configuración en tiempo real, ya que realmente VR Juggler es una plataforma virtual y un cambio en el hardware no tiene que ser notificado a la aplicación.

A.3.2 Detalles de implementación

Como se ha comentado anteriormente, las aplicaciones son objetos que deben implementar unas funciones determinadas, para que éstas sean llamadas por el kernel y realizar así su ejecución. VR Juggler define una relación de herencia entre varias clases abstractas que definen estos métodos (Figura A.2).

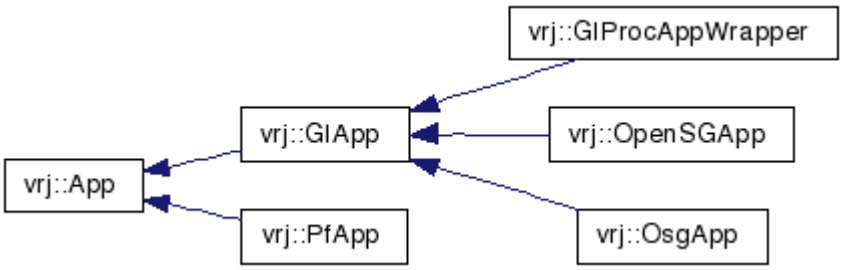


Figura A.2. Herencia entre las clases que definen una aplicación en VR Juggler.

La clase "App" encapsula todas las funciones comunes en cualquier aplicación para VR Juggler. A partir de ahí, otras clases abstractas más especializadas definen otros métodos dependiendo del tipo de aplicación que se vaya a realizar. Por ejemplo, si se va a usar OpenGL Performer se definirá una herencia sobre "PfApp" y si se va a usar OpenGL se heredará de "GlApp". También es posible heredar de otras clases más especializadas en el caso de utilizar OpenSG, OpenSceneGraph, etc.

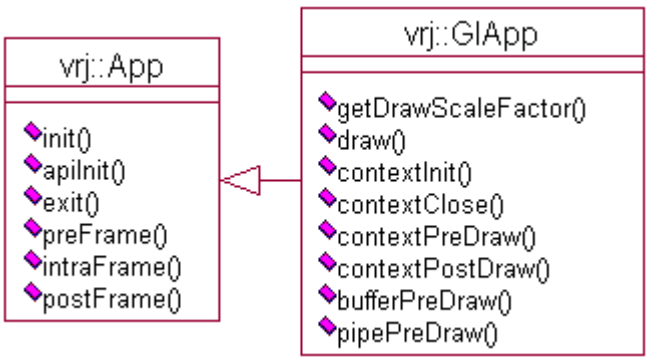


Figura A.3. Detalle de las funciones de las clases "App" y "GlApp" de VR Juggler.

El núcleo de ejecución de VR Juggler crea un bucle para la aplicación en el que, entre otras cosas, actualiza el estado de los dispositivos y dibuja en cada contexto de OpenGL los gráficos correspondientes generados mediante la función "draw". Entre estas dos operaciones se ejecutan las funciones "preFrame", "intraFrame" y "postFrame". Su orden de ejecución es, tal y como indica su nombre, antes del dibujado (y justo después de actualizar los dispositivos), durante el dibujado, y después del dibujado (Figura A.4).

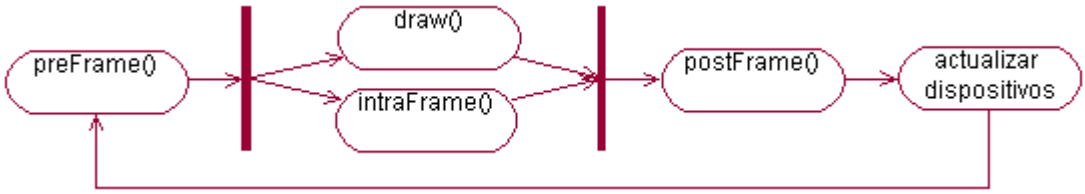


Figura A.4. Flujo de ejecución de las principales llamadas de VR Juggler.

Dependiendo de las necesidades de la aplicación, se implementará una o más de estas funciones para realizar las tareas convenientes. Hay que tener en cuenta que cualquier retardo introducido en "preFrame()" o "postFrame()" repercute directamente en la velocidad de ejecución de todo el bucle, y por tanto, en el número de cuadros por segundos que la aplicación será capaz de mostrar. Cualquier tarea realizada en "intraFrame()" en cambio, se realizará en paralelo mientras se realiza el dibujado, por lo que no influye tan directamente.

Otras funciones útiles son:

- **init().** El núcleo llama a esta función antes de iniciar el bucle de ejecución. Se usa para inicializar el estado de la aplicación, así como los dispositivos usados por la misma, asociando variables a los proxies definidos en la configuración.

- **getDrawScaleFactor.** Permite cambiar las unidades de la aplicación para escalar los gráficos generados en relación a ellas.

- **contextInit().** Se llama justo después de que se cree cada contexto de OpenGL. Aquí hay que inicializar el estado de OpenGL, así como la información dependiente del contexto.

- **contextPreDraw.** Se llama justo antes de realizar el dibujado de cada uno de los contextos de OpenGL. Es el lugar adecuado para actualizar variables dependientes del contexto, como por ejemplo los display lists.

A.4 CONFIGURACIÓN DE VR JUGGLER

Uno de los aspectos de VR Juggler más importantes es su alto grado de configuración. Esto es necesario para poder hacer frente a las grandes variaciones existentes entre distintos sistemas de Realidad Virtual. Para garantizar esto se usan los *elementos de configuración*, que permiten parametrizar los componentes de VR Juggler.

A.4.1 Elementos de configuración

Un elemento de configuración es una unidad elemental compuesta de propiedades y está identificada por un nombre. Cada propiedad tiene un tipo de datos, y algunas pueden tener una lista fija de valores permitidos, o incluso más de un valor a la vez.

Elemento: *display_window*	
Origin: int	X Coordinate
	Y Coordinate
Size: int	Width
	Height
Stereo: bool	In stereo?

Figura A.5. Ejemplo de elemento de configuración.

A.4.2 Proxies de dispositivos

Los proxies se utilizan en VR Juggler como una capa de abstracción e indirección para asegurar que las aplicaciones desarrolladas no dependan de dispositivos específicos. Así, un programa hace referencia por ejemplo a un proxy de localización para capturar datos de la posición del usuario, sin necesidad de saber con que dispositivo está tratando. La asociación de proxies a dispositivos se hace en el fichero de configuración, que normalmente se escribirá mediante la utilidad vrjConfig.

Existen distintos tipos de proxies: analog, command, digital, digital glove, gesture, keyboard/mouse, positional y string. Cada uno de ellos está orientado a un tipo de dispositivo, y proporciona un tipo de datos diferente.

A.4.3 Filtros de posición

En los dispositivos de localización, la información devuelta acerca de posición y orientación está expresada en unas unidades propias y en un sistema de coordenadas que no tiene por qué coincidir con el usado en la aplicación, y que cambian en cada configuración. Para hacer posible la transformación de esta información a un sistema de coordenadas común y con unas unidades estándar VR Juggler dispone de los llamados filtros de posición.

Los filtros de posición consisten en una serie de matrices de transformación a las que se le dan los valores de rotación y traslación necesarios para pasar de un sistema de coordenadas a otro. Estas matrices se multiplicarán por los datos proporcionados por el dispositivo en un orden concreto.

Las matrices que componen un filtro de posición, y su orden de aplicación son las siguientes:

*base = preTraslación * preRotación * escala * sensor * postTraslación * postRotación*

Donde *sensor* es la matriz con los datos de orientación y posición originales y *base* son los datos transformados.

Hay que comentar que los filtros de posición se pueden componer, es decir, se pueden aplicar varios a unos determinados datos. Esto se hace haciendo que "sensor_base" del primer filtro aplicado sea la matriz "sensor" del siguiente. Normalmente se tendrá un filtro de posición en el dispositivo en el que se configuran las matrices de pre-traslación, pre-rotación y escala, y otro más por cada uno de los sensores en los que se configuran las matrices post-traslación y post-rotación. En el anexo C se muestra un ejemplo de configuración del dispositivo Flock of Birds.

A.4.4 Ejemplo de configuración: el teclado

En la aplicación de modelado creada, el guante de datos háptico no es compatible directamente por VR Juggler (véase apartado 4.6), por lo que el único control de entrada que hay que configurar es el teclado. Mediante él se podrá acceder al menú mediante la tecla F1, y salir de la aplicación con la tecla Escape.

Para poder usar el teclado en VR Juggler es necesario emular un dispositivo de entrada digital abstracto que tenga tantos botones como teclas se quieran usar.

Para empezar se añade un dispositivo de tipo *keyboard_mouse_device* y un proxy que apunte a él. El siguiente paso es agregar un elemento de configuración de tipo *simulated_digital_device* en el que se especificará el proxy del teclado y qué teclas se desea que sean interpretadas como botones digitales.

En este paso ya está creado un dispositivo digital a partir del teclado, por lo que el resto de configuración sería igual si se deseara asignar las funciones de menú y salir a cualquier otro.

Para salir de la aplicación hay que crear un proxy de tipo *digital_proxy* que haga referencia al dispositivo y botón que se usará para tal acción. Para acabar, se crea un elemento de configuración especial de tipo *alias* al que se dará el nombre *VJSystemStopKernel* que apunte a este proxy. De esta manera la aplicación se terminará automáticamente al pulsar la tecla, sin necesidad de programarlo.

Por último, para la función de menú se creará otro proxy de tipo *digital_proxy* que apunte al botón del dispositivo digital simulado que fue creado anteriormente. De forma opcional también se pueden crear otros cuatro proxies para emular el contacto entre los dedos, aunque esto tiene más utilidad a la hora de llevar a cabo la depuración. A estos proxies se podrá acceder posteriormente desde la aplicación para conocer su estado.

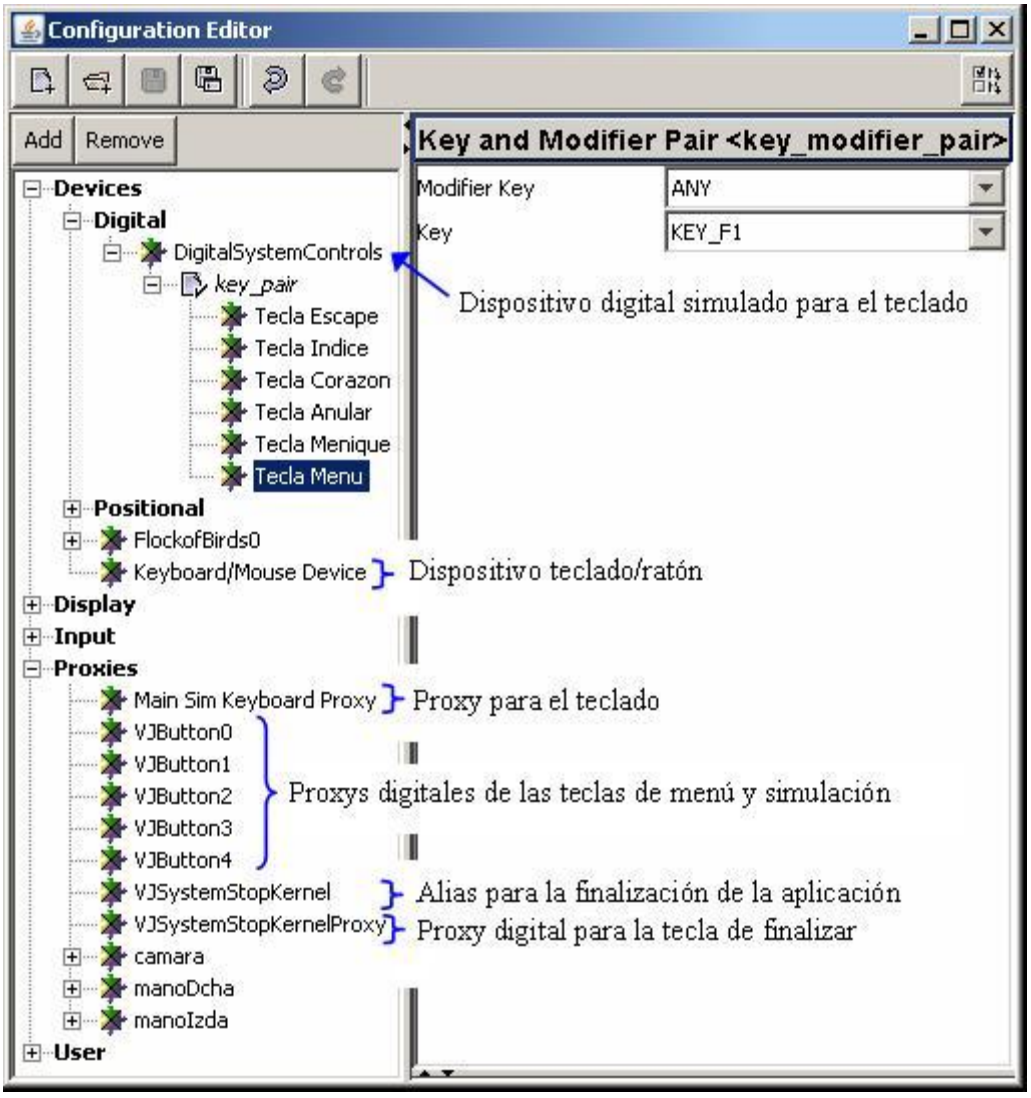


Figura A.6. Configuración del teclado mediante vrjConfig.

ANEXO B. SISTEMA DE RETROPROYECCIÓN ESTEREOSCÓPICO

B.1 INTRODUCCIÓN

El sistema de retroproyección estereoscópico está formado por una pantalla vertical translúcida y por dos proyectores montados en la parte trasera que mediante un espejo enfocan su imagen en la misma. Todo esto está montado sobre una estructura metálica ajustable (Figura B.1).

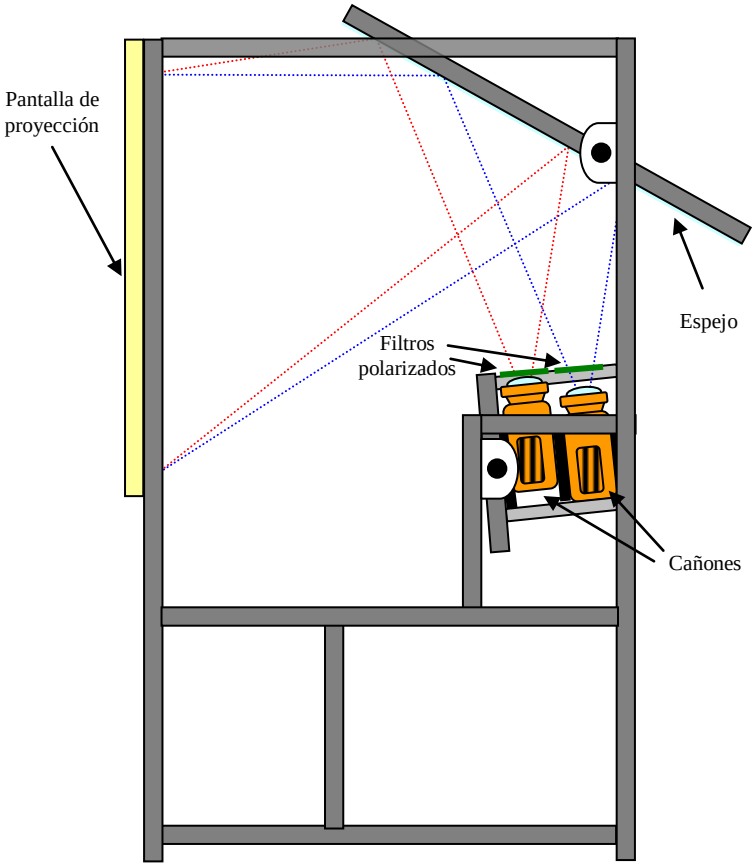


Figura B.1. Sistema de retroproyección estereoscópico.

A diferencia de otros sistemas estereoscópicos donde sólo se utiliza un dispositivo para mostrar ambas imágenes (intercalándolas en el tiempo, por ejemplo), en este caso se dispone de dos cañones, llegándole a uno de ellos la imagen para el ojo izquierdo, y al otro la imagen para el ojo derecho.

Para que cada ojo vea solamente su imagen correspondiente, se usan filtros de polarización vertical y horizontal. En los proyectores se colocan justo delante del objetivo

(Figura B.2), y en el usuario se usan unas gafas especiales muy ligeras. Cada ojo tendrá asociado un tipo de filtro que coincidirá con el del cañón que proyecte su imagen. De esta manera cada ojo recibirá únicamente la imagen que le está destinada.

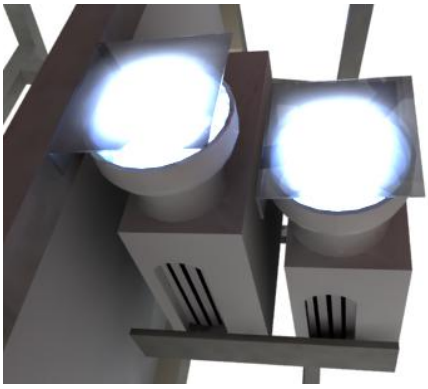


Figura B.2. Filtros de polarización colocados en la pareja de proyectores.

Es muy importante que ambas imágenes proyectadas coincidan geométricamente en la pantalla, ya que el cerebro debe fundirlas como si fuera una sola. En caso contrario la sensación de profundidad se degrada, e incluso puede llegar a provocar dolor de cabeza o mareo.

B.2 CALIBRACIÓN HARDWARE

Esta calibración es la más importante, ya que permite regular en un amplio margen la posición y enfoque de las imágenes proyectadas, y siempre sin provocar pérdidas en la calidad.

La estructura metálica que da soporte a ambos cañones permite una serie de ajustes con los que se intentará hacer coincidir las dos imágenes de la forma más fiel posible. Los grados de libertad que ofrece son los siguientes:

- Ángulo del espejo
- Regulación en altura independiente para cada proyector
- Ángulo del soporte de los proyectores
- Ajuste del ángulo relativo entre proyectores

Para realizar el ajuste hay que tener en cuenta que la geometría varía un poco cuando los cañones están calientes, por lo que es necesario encenderlos al menos 10 minutos antes.

Se ha utilizado una imagen que nos proporciona información tanto del ajuste punto a punto en toda la superficie (mediante la cuadrícula), como de la adecuada relación de aspecto ancho/alto (mediante las circunferencias).

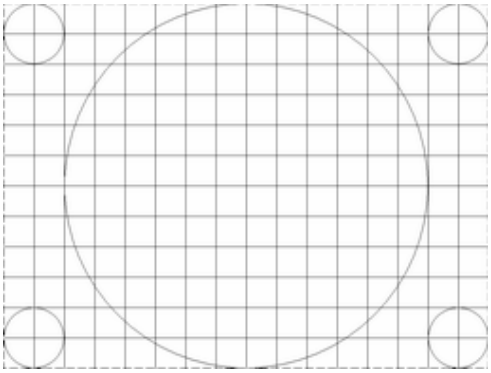


Figura B.3. Imagen usada para el ajuste de las dos imágenes.

Debido a que la posición de los cañones no es exactamente la misma para ambos, hacer coincidir la imagen en sus cuatro esquinas es muy difícil, ya que su geometría varía inevitablemente a pesar de los ajustes que podamos realizar. Este efecto recibe el nombre de *keystoning*.

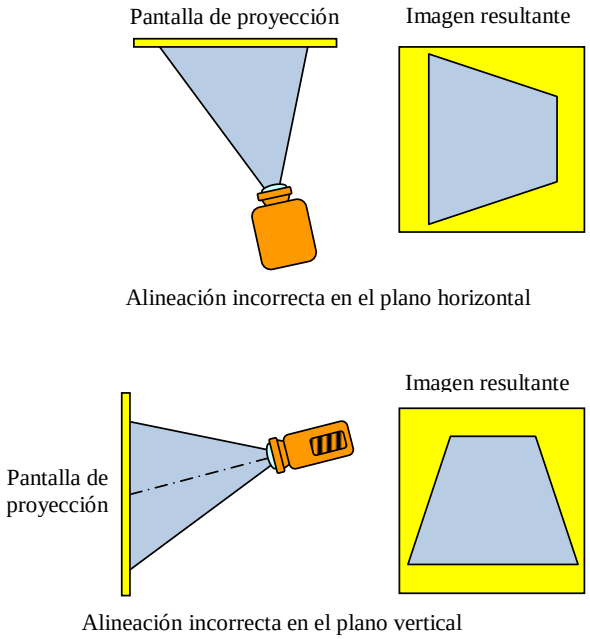


Figura B.4. Efecto keystone horizontal (arriba) y vertical (abajo).

El **keystone horizontal** se produce debido a una alineación incorrecta en el plano horizontal. Debido a que ambos proyectores se encuentran alineados en el plano horizontal, y centrados respecto a la pantalla, este efecto no se produce.

El **keystone vertical** se produce por una alineación incorrecta en el plano vertical. Como ambos cañones se encuentran uno encima del otro, al menos uno de los dos presentará este efecto. Algunos proyectores permiten cambiar el ángulo de su óptica para corregir este efecto, otros permiten hacerlo por software, como es el caso de los proyectores del laboratorio.

Aun así, se han realizado ajustes hasta lograr la mejor coincidencia posible entre las dos imágenes.

B.3 CALIBRACIÓN SOFTWARE

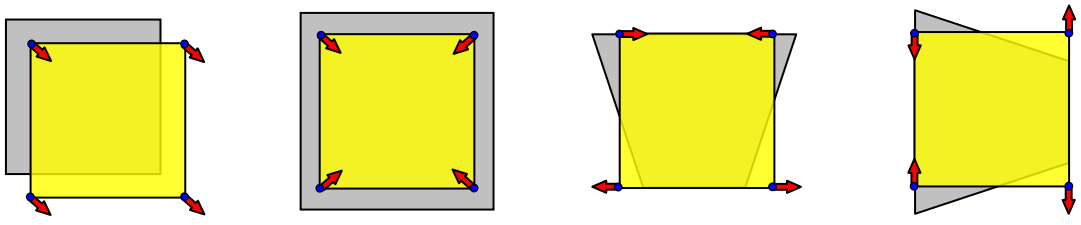
Una vez conseguida la mejor calibración por hardware posible, es posible realizar una calibración por software.

Esta calibración realiza una correspondencia entre los píxeles lógicos (bits con información de color) y los píxeles físicos (cada unidad mínima de color que puede ser cambiada en el dispositivo físico).

Para obtener los mejores resultados lo mejor es usar una correspondencia directa, es decir, que a cada píxel lógico se le asigne un píxel físico. En caso contrario se realiza una interpolación, por lo que la calidad de la imagen se degrada ligeramente.

Además, solamente es posible ajustar la imagen reduciéndola, ya que la localización de los píxeles físicos ya está definida por los ajustes del hardware.

Para realizar este ajuste, muchos proyectores disponen de opciones de corrección de la posición de la imagen, del tamaño e incluso de su geometría.



Ajuste posición (abajo-derecha) | Ajuste tamaño (reducir ancho y alto) | Ajuste geometría (keystone horizontal) | Ajuste geometría (keystone vertical)

Figura B.5. Ajustes de geometría más comunes.

En el caso de los proyectores usados en el laboratorio, las correcciones ofrecidas en el menú son (Figura B.6):

- Ajuste de posición (horizontal y vertical)
- Ajuste de keystone (horizontal y vertical)

Figura B.6. Ajuste de de keystone en el menú del proyector.

Durante la realización de las pruebas, estos ajustes han permitido una correspondencia muy buena entre las dos imágenes en regiones centrales, pero en las regiones periféricas se aprecia una ligera deformación. Por este motivo se ha recurrido a una herramienta de NVidia llamada NVKeystone incluida en el driver de la tarjeta gráfica.

Este software permite cambiar libremente la posición tanto de las cuatro esquinas que forman la imagen, como del punto central.

La potencia de esta herramienta junto con todos los ajustes realizados anteriormente ha permitido realizar un ajuste muy bueno, coincidiendo las imágenes de ambos proyectores de manera muy satisfactoria (Figura B.).

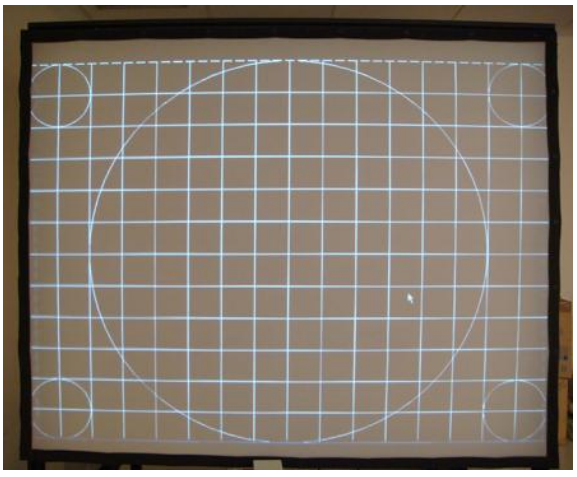


Figura B.7. Resultado final de la calibración de ambos proyectores.

B.4 CONFIGURACIÓN EN VR JUGGLER

Para poder hacer uso del sistema de retroproyección en VR Juggler, es necesario crear o modificar el archivo de configuración que más tarde se usará en la aplicación.

Para ello es necesario añadir un elemento de configuración de tipo *display_system*, en el que se especificarán las salidas gráficas disponibles en el sistema. Como la tarjeta gráfica que usada debe tener una salida para cada proyector, especificamos en el atributo *Number of Pipes* el valor 2.

A continuación hay que especificar cuál es la dirección de cada una de las salidas gráficas en el apartado *x11_pipes*. Se sigue la nomenclatura :N.M, siendo N el número de máquina empezando en 0 (es posible más de una máquina en una configuración de cluster), y M el número de salida gráfica de cada una de ellas, empezando a contar en 1. En esta configuración se usa un único ordenador, por lo que un display será **:0.1** y el otro **:0.2**

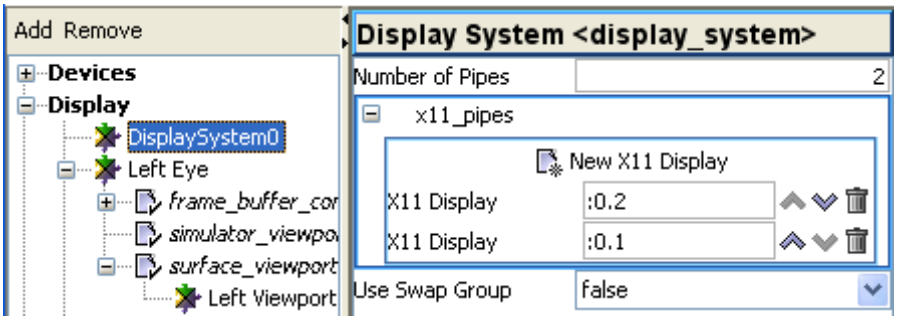


Figura B.8. Configuración de las salidas gráficas mediante vrjConfig.

Una vez están configuradas las salidas gráficas, se deberán crear dos ventanas –una para cada ojo- que estarán situadas en cada una de ellas. Las propiedades que hay que establecer son las siguientes:

- **Origin.** Indica la posición de la ventana respecto al escritorio. Como ambas salidas gráficas forman parte de un escritorio extendido, y se usa como resolución 1024x768, la ventana del ojo izquierdo tendrá un origen 0,0 y la ventana del ojo derecho 1024,0.

- **Size.** Tamaño de la ventana. Coincidirá con la resolución de la salida gráfica, esto es, 1024x768.

- **Pipe**. Indica la salida gráfica que se debe usar para mostrar la ventana. Se establecerá a 0 para la izquierda y 1 para la derecha.

- **GL_Frame_Buffer.** Permite especificar diversas opciones gráficas para OpenGL. Se dejarán las opciones por defecto.

- **In Stereo.** Indica si se deben mostrar dos imágenes para esa ventana usando alguna técnica como el entrelazado de líneas o los anaglifos. Como se va a usar una ventana para cada ojo, este valor hay que ponerlo a falso.

- **Use border**. Si se establece a verdadero se mostrará un borde para poder redimensionar la ventana. Esto es optativo, se puede dejar a true para poder identificar mejor cada ventana.

- **Hide Mouse Pointer**. Indica si se debe ocultar el puntero del ratón. Verdadero.

- **Use this window?**. Permite desactivar una ventana. En este caso, falso.

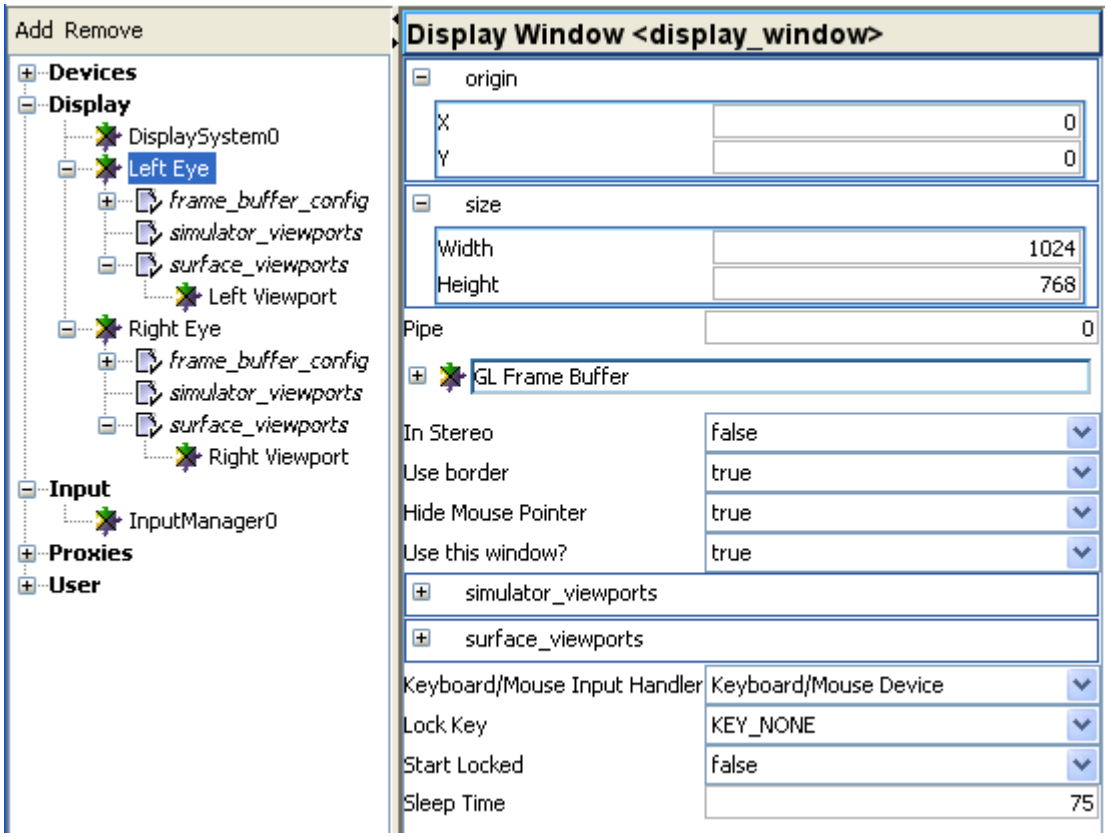


Figura B.9. Configuración de las características de la ventana mediante vrjConfig.

El siguiente paso es el más importante para la correcta configuración del sistema gráfico. Consiste en definir el plano de proyección que se usará para plasmar el mundo virtual tridimensional mediante dos imágenes en 2D. Mediante él se hará una correspondencia entre el mundo físico y el virtual, por lo que definirlo de manera ajustada es imprescindible para conseguir una buena calibración.

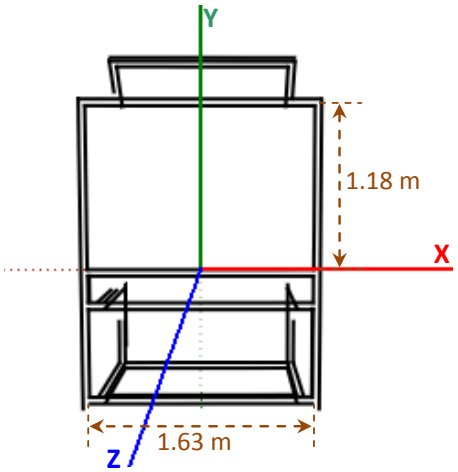


Figura B.10. Sistema de coordenadas y dimensiones de la pantalla de proyección.

El plano de proyección se establecerá definiendo la posición y dimensión de la pantalla de retroproyección, usando para ello el sistema de coordenadas de la escena (Figura B.10). Esto se hace mediante el apartado *surface_viewports*, en el que se definirán las siguientes propiedades:

- **Origin**. Indica en qué parte de la ventana creada se representará la imagen. Como se usa toda la ventana, las coordenadas deben ser 0, 0.

- **Size**. Indica el tanto por uno de la ventana que se usará. Suele ser siempre 1.0.

- **Lower_left_corner**. Coordenadas de la esquina inferior izquierda, teniendo en cuenta las dimensiones de la pantalla y el origen de coordenadas escogido. Se especifican en metros: -0.815, 0

- **Lower_right_corner**. Coordenadas de la esquina inferior derecha: 0.815, 0

- **Upper_right_corner.** Coordenadas de la esquina superior izquierda: 0.815, 1.18

- **Upper_left_corner.** Coordenadas de la esquina superior derecha: -0.815, 1.18

- **User.** Perfil de usuario. Esto se comentará posteriormente.

- **Use this viewport.** Establecer a verdadero para usar este viewport.

- **Is tracked.** Esta propiedad indica si el plano de proyección es móvil, y acompaña al usuario, o bien está fijo, cambiando únicamente el punto de vista del usuario al moverse. La primera opción se usa para el caso de los visiocascos, y la segunda para el caso de pantallas de proyección. Se establecerá por tanto a falso.

- **Tracker Proxy**. Aquí se elegirá el proxy que está unido a la cabeza del usuario. En nuestro caso se elegirá el proxy "camara".

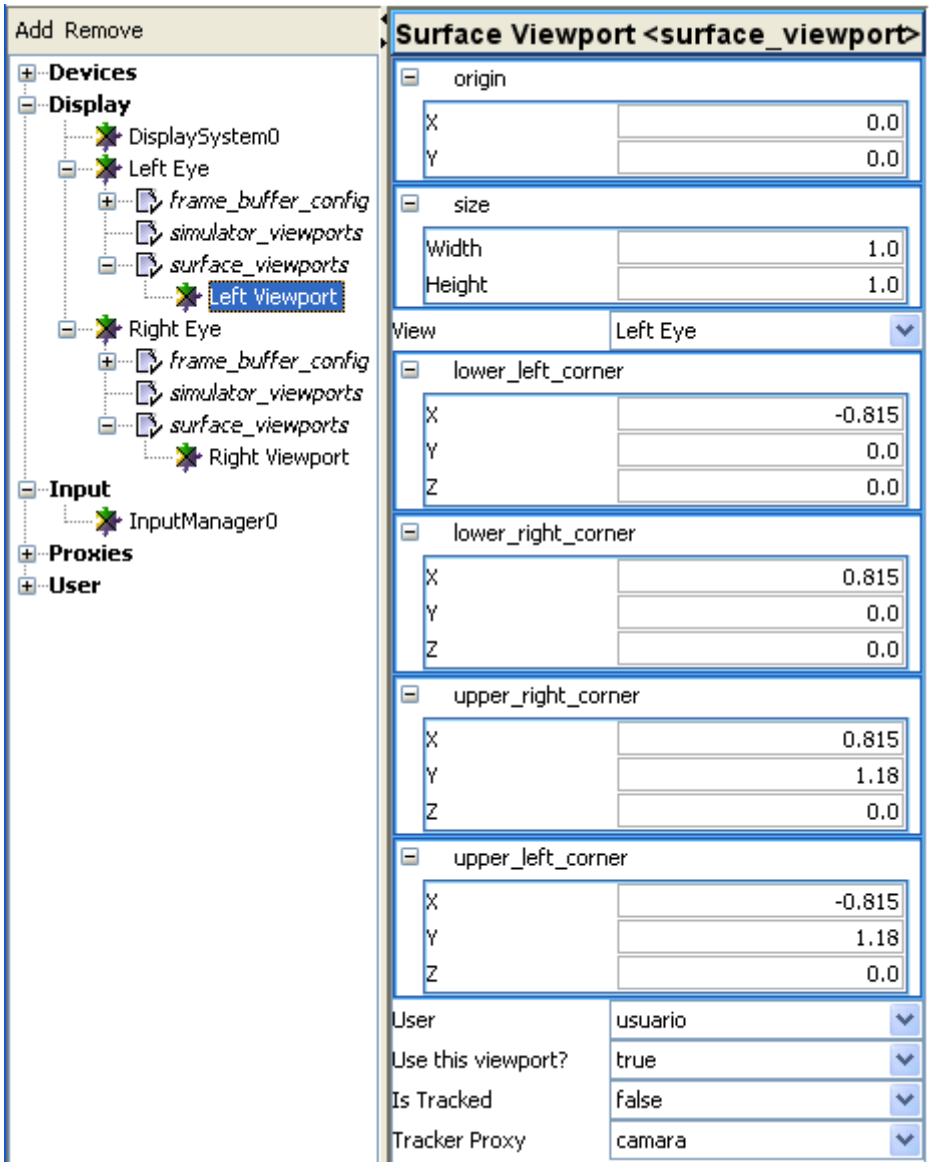


Figura B.11. Configuración del puerto de vista mediante vrjConfig.

Antes de dar por finalizada la configuración gráfica de VR Juggler, es necesario realizar el ajuste de distancia interpupilar del usuario en un elemento de configuración de tipo *user*. Esta distancia es usada para producir una imagen diferente para cada ojo, por lo que una separación excesiva podría causar malestar en el usuario, y una distancia demasiado pequeña degradaría la sensación de profundidad. El valor predefinido es de 6.9 centímetros, que experimentalmente se ve que es demasiado grande. El valor óptimo dependerá de cada usuario, sin embargo es conveniente establecer un valor que sea ligeramente menor.

ANEXO C. EL SISTEMA DE LOCALIZACIÓN FLOCK OF BIRDS

C.1 INTRODUCCIÓN

El sistema Flock of Birds, también nombrado por sus siglas, FOB, es un sistema de localización creado por la compañía Ascension que permite medir en tiempo real la orientación y posición de múltiples sensores a una velocidad de hasta 144 medidas por segundo.

Figura C.1. Sistema de localización Flock of Birds.

Su tecnología se basa en la emisión de pulsos electromagnéticos DC y proporciona una zona de cobertura en forma de semiesfera de aproximadamente 1.2 metros de radio en condiciones ideales, que se ven degradadas en presencia de objetos metálicos. Esto le permite total libertad de movimientos, ya que no se necesita una línea de visión directa para su correcto funcionamiento.

FOB está compuesto por un emisor electromagnético fijo de forma cúbica de unos 10 centímetros de lado, uno o más sensores móviles de unos 25 milímetros y de una unidad electrónica por cada uno de los sensores.

El emisor electromagnético está formado por tres antenas que emiten tres campos ortogonales entre sí con una frecuencia determinada.

Los receptores están formados por tres bobinas perpendiculares que miden simultáneamente la intensidad de cada campo. Esta información se transmite mediante un cable a su unidad electrónica correspondiente.

Las unidades electrónicas, llamadas Ascension Bird, calculan la posición y orientación de su sensor mediante un proceso de DSP y lo transmiten bien a otra unidad electrónica mediante un bus llamado FBB (Fast Bird Bus), o bien al ordenador mediante la interfaz RS-232 (puerto serie).

C.2 ORGANIZACIÓN FÍSICA

El sistema FOB puede ser configurado para la utilización de un solo sensor, o bien de forma más compleja con varias combinaciones de emisores y receptores. Aquí se describirá una configuración típica para trabajar con varios sensores y una sola conexión a un ordenador.

En todas las configuraciones se debe contar con una unidad Ascension Bird configurada como maestro y opcionalmente con otras unidades Ascension Bird configuradas como esclavo.

La unidad maestro realizará la comunicación con el ordenador, y será la encargada de coordinar las operaciones. Normalmente se conectará a esta unidad el emisor electromagnético.

El resto de unidades estarán conectadas mediante el bus FBB siguiendo el siguiente esquema:

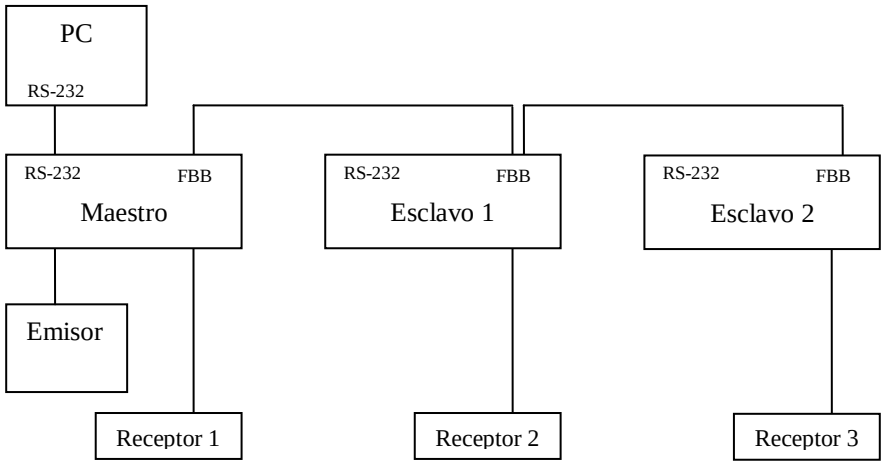


Figura C.2. Ajustes de geometría más comunes.

Para incrementar el número de sensores del sistema, simplemente es necesario conectar una nueva unidad Ascension Bird al FBB. Como cada unidad posee su propio

procesador independiente, FOB puede hacer el seguimiento de hasta 126 sensores sin que decaiga su rendimiento.

C.3 CONFIGURACIÓN DE LAS UNIDADES ASCENSION BIRD

Cada una de las unidades electrónicas cuenta con un conjunto de 8 interruptores llamado *dipswitch*. Mediante éste se seleccionará la velocidad de la interfaz en baudios y el identificador de la unidad, entre otras funciones.

La identificación de las unidades se puede hacer con direccionamiento normal (hasta 14 unidades), expandido (hasta 30 unidades), o super-expandido (hasta 126 unidades).

De forma predeterminada se usará el direccionamiento normal, y se asignará un identificador mediante los interruptores 4, 5, 6 y 7.

En este modo, los interruptores 1, 2 y 3 establecen la velocidad en baudios de la interfaz RS232. El interruptor 8 permite cambiar entre modo de funcionamiento normal (fly) y modo de prueba (test).

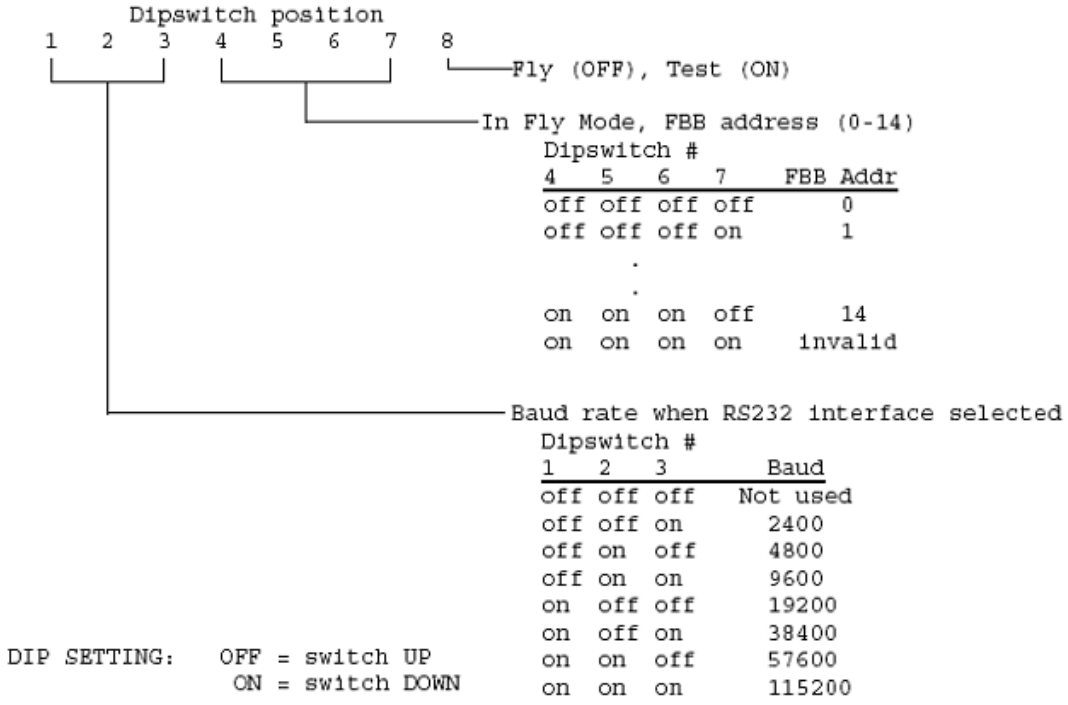


Figura C.3. Configuración del dipswitch de FOB en modo de direccionamiento normal.

Se configurarán tres unidades Ascension Bird con los identificadores 1, 2 y 3. La velocidad será de 115200 baudios. Siguiendo el anterior esquema, los switches deberán configurarse como en la siguiente figura.

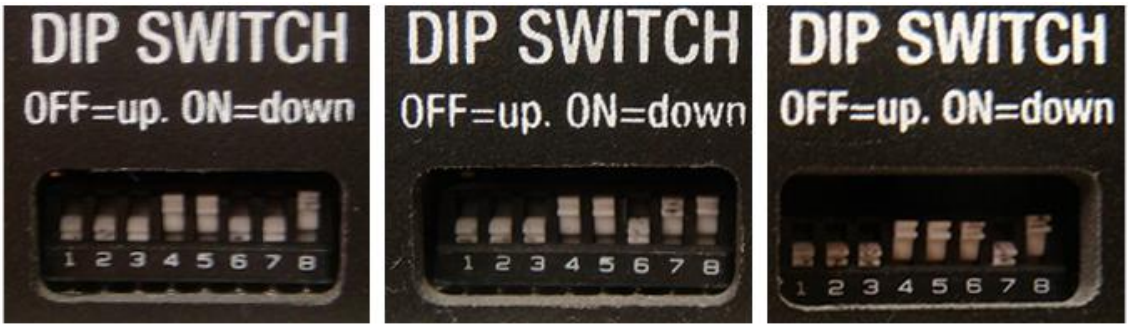


Figura C.4. Configuración de los switches de las unidades electrónicas de FOB.

C.4 CONFIGURACIÓN EN VR JUGGLER

Una de las tareas más importantes, y que más problemas trae a la hora de usar cualquier sistema de localización, es la configuración del software para que interprete correctamente los datos de localización. Es por ello que a continuación se explicará en detalle cuales han sido los pasos necesarios para la configuración de Flock of Birds.

C.4.1 Dispositivo de localización Flock of Birds

La primera tarea a realizar en la configuración de cualquier dispositivo es la especificación de los drivers necesarios que se deben cargar. Esto se hace añadiendo un elemento de configuración de tipo *input_manager* en el que se elegirá en "Driver module" la entrada "Flock_drv".

En segundo lugar se agregará un nuevo dispositivo de tipo "Flock of Birds". Los parámetros que se pueden configurar son:

- **Serial Port.** Se indicará el nombre del puerto serie en el que está conectado FOB. Por ejemplo, COM1.

- **Baud**. Indica la velocidad del puerto serie a la que se realizará la comunicación. Debe coincidir con la que se configuró mediante el dipswitch en cada unidad electrónica, por lo tanto se establecerá a 115200.

- **Hemisphere**. Dado que FOB proporciona una zona de cobertura en forma de semiesfera, se permite elegir cuál será la posición de la misma respecto del emisor. Para este proyecto se elegirá "Top", de esta manera se podrán mover los sensores por delante y detrás del emisor, pero siempre por la parte de arriba del mismo.

- **Master Address**. El identificador que se escogió para la unidad electrónica que actúa como maestro. En este caso, 1.

- **Addressing Mode**. El modo de direccionamiento que se ha elegido para el sistema FOB. Como se comentó anteriormente, puede ser Normal, Extendido o Super Extendido. Es este caso se elegirá *Normal*.

- **Filter**. Permite elegir un filtrado de los datos por software. Esto se hace para corregir pequeñas variaciones ocasionadas por el ruido. Normalmente se elegirá AC_NARROW para entornos con poco ruido.

- **Position_filters**. Se usa para especificar matrices de transformación de coordenadas que se van a aplicar a los datos recibidos del dispositivo de seguimiento. Esto nos será útil para cambiar el sistema de coordenadas, como se detallará más adelante.

- **Host Node**. Se utiliza para configuraciones de clusters. En este caso se elegirá *None*.

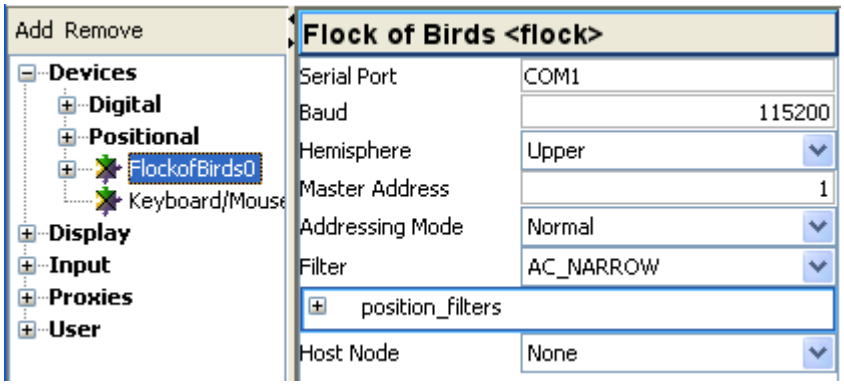


Figura C.5. Configuración del dispositivo Flock of Birds mediante vrjConfig.

A continuación se detallará la configuración de los **filtros de posición**. Con ellos es posible especificar la posición del emisor respecto del origen, cambiar el sistema de coordenadas, y especificar la escala de medida.

En primer lugar consultando el manual de FOB se puede ver que su sistema de coordenadas es de mano derecha, tal y como se ve en la figura C.6.

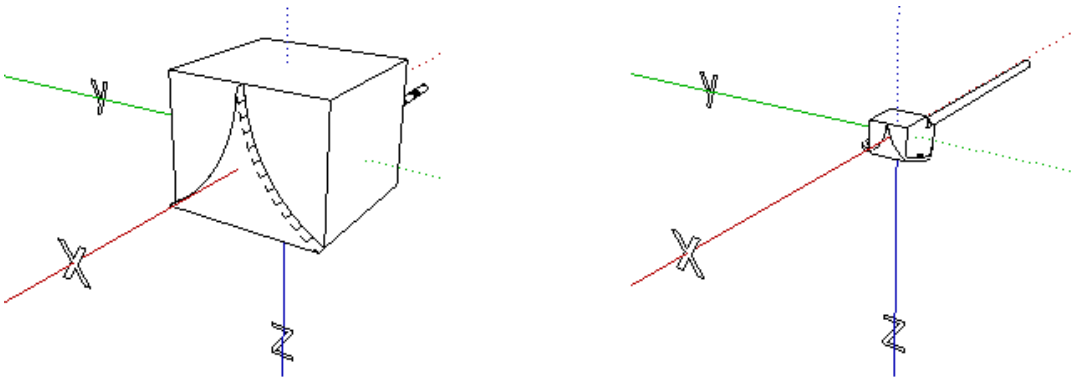


Figura C.6. Sistema de coordenadas del emisor de Flock of Birds (izquierda) y receptor (derecha).

El sistema de coordenadas de OpenGL, que es el usado también en la aplicación, es también de mano derecha, sin embargo la orientación de los ejes no coincide con la de FOB (Figura C.7).

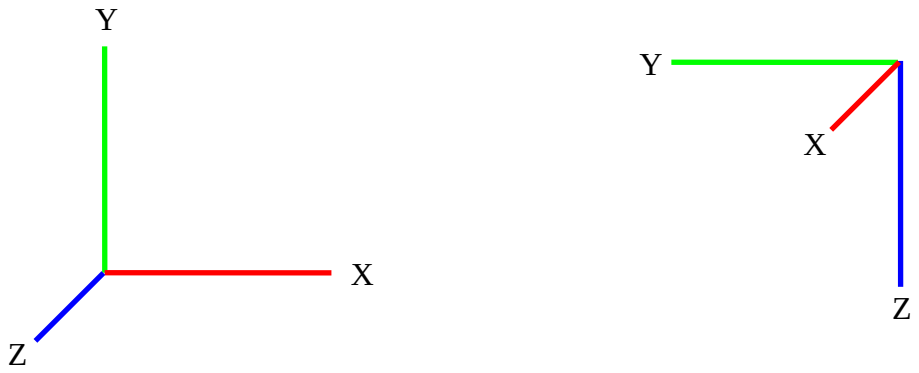


Figura C.7. Sistema de coordenadas de OpenGL (izquierda) y Flock of Birds (derecha).

Para cambiar de uno a otro, simplemente es necesario aplicar una rotación de 90º tanto en el eje X como en el eje Z. Estos valores se introducirán en el apartado *pre_rotation*.

El driver de FOB para VR Juggler devuelve las medidas en decímetros. Como se recomienda en el manual de vrjConfig, se aplicará un multiplicador de 0.1 para convertir las unidades de medida a metros. Para ello se seleccionará *Custom* en el apartado Device Units, y *0.1* en Custom Scale.

El emisor de FOB estará situado a 65 centímetros de la pantalla para aprovechar al máximo la zona de trabajo, mientras que el centro de coordenadas de la escena está definido en la parte más baja de la pantalla de proyección (Figura C.8). Esto hace que

coincidan en las coordenadas X e Y, pero no en la Z. Para especificar esta traslación se hará uso de los valores *pre_translation*, cuyo valor de Z será de 0.65 metros y el resto cero.

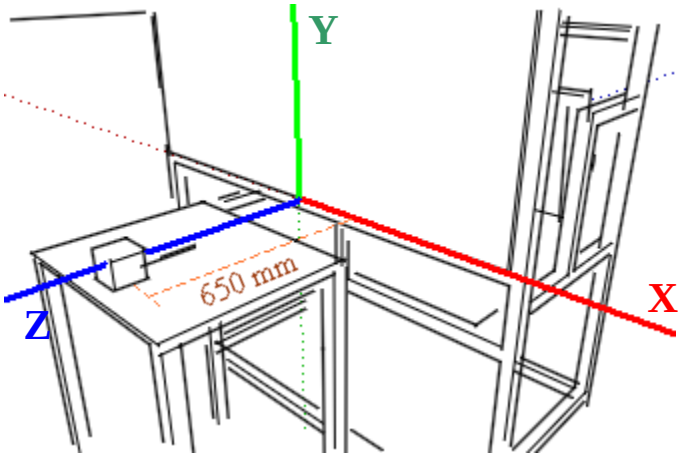


Figura C.8. Posición del emisor de FOB respecto al origen de coordenadas de la escena.

C.4.2 Sensores

El siguiente paso consiste en crear un proxy de tipo *position_proxy* para cada sensor del FOB (véase anexo A, apartado A.4.2). A pesar de que actualmente no se utiliza la mano izquierda, se definirá también para posibles ampliaciones.

Los nombres escogidos son *camara*, y *manoDcha* y *manoIzda.*

En la configuración de cada proxy, se debe elegir en la propiedad *Device* cuál es el dispositivo origen, por ejemplo "FlockofBirds0". En *Unit* se indicará qué unidad electrónica está conectada al sensor. Este número será 0 para la unidad maestro, 1 para la primera unidad esclavo, 2 para la segunda unidad esclavo, etc. Se ha escogido 0 para *manoIzda*, 1 para *camara* y 2 para *manoDcha*.

Cada proxy lleva asociado también uno o más filtros de posición. Estos filtros permitirán cambiar la posición y orientación de su origen de coordenadas respecto al punto que se desea medir.

El sensor de la cámara se colocará en una de las patillas de las gafas polarizadas. La posición que se desea medir es el punto medio entre ambos ojos, y teniendo en cuenta su sistema de coordenadas, se ve que el receptor está desplazado tanto en el eje Y como en el eje Z unos 10 centímetros (Figura C.9). Estos valores se introducirán en *post_translation*.

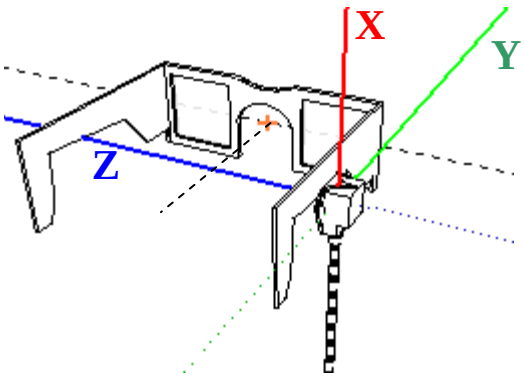


Figura C.9. Posición del receptor para la cámara.

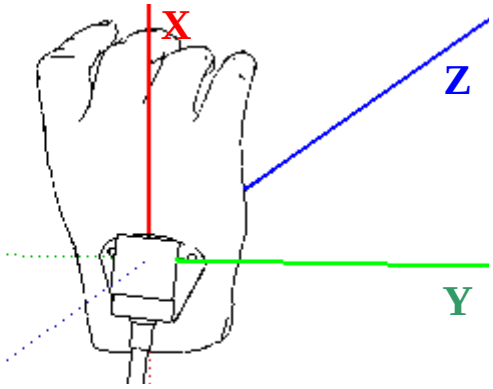


Figura C.10. Posición del receptor para el guante.

Como el sistema de coordenadas local del sensor no tiene la misma orientación que el sistema de coordenadas de la escena, hay que rotarlo 90º en el eje X y -90º en el eje Z para hacerlos coincidir. Estos valores son los correspondientes a *post_rotation*.

El sensor del guante estará colocado justo sobre el punto que se desea medir, por lo que no se necesitará realizar ninguna traslación. Sin embargo la orientación de su sistema de coordenadas no coincide con el de la escena, siendo necesario girarlo como en el caso anterior. Los giros serán de 180º en el eje X y de -90º en el eje Z.

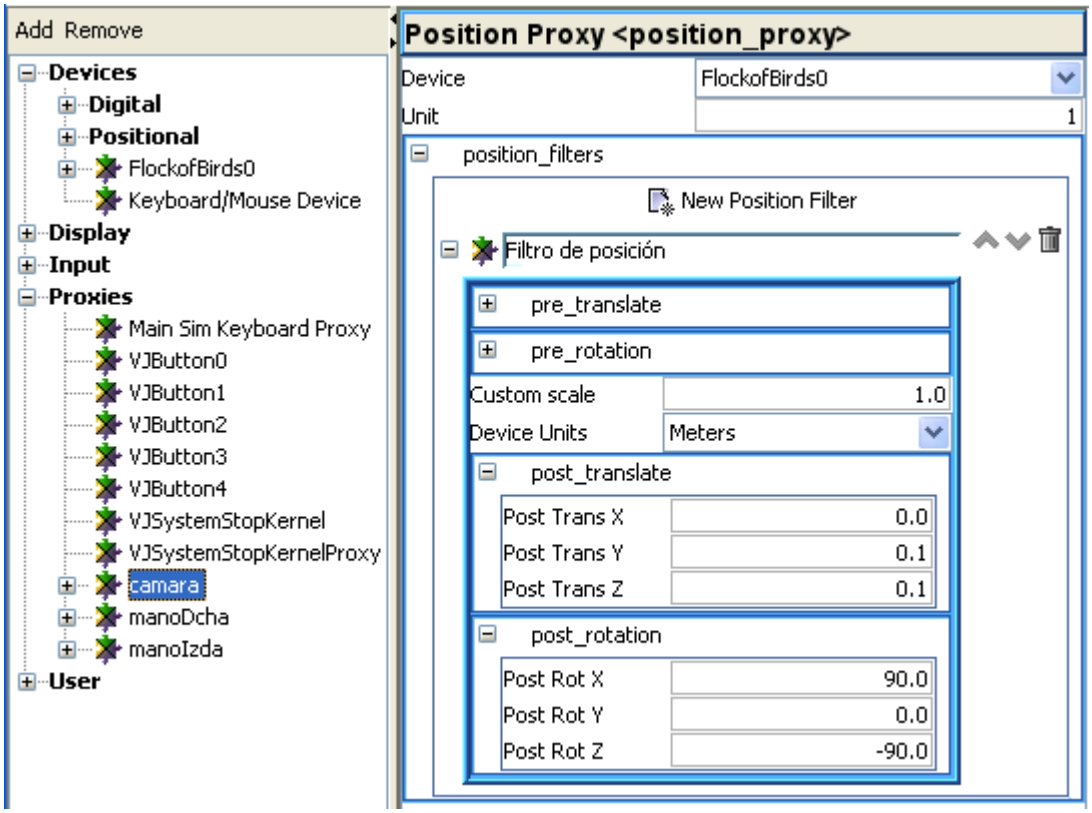


Figura C.11. Configuración del sensor de la cámara mediante vrjConfig.