

Evaluación de periféricos para el control de mundos virtuales

Víctor Goñi Sanz

Universidad Pública de Navarra
Departamento de Ingeniería Eléctrica y Electrónica
Tutor: Jeser Zalba

VICOMTech
Tutor: David Oyarzun

Dedicado a mi familia.
Sin ellos no habría llegado a donde estoy, sin ellos no sería lo que soy.

Resumen del proyecto

En este proyecto se va a analizar, a raíz de los nuevos periféricos que salen al mercado, qué ventajas y desventajas tiene cada uno de ellos, para elegir cual es el que mejor para que cualquier usuario interactúe con un mundo virtual.

Para poder evaluarlo, se van a seguir varios pasos que se han considerado imprescindibles para llevar a cabo el análisis. En primer lugar, se ha creado un mundo virtual, en el que se puede controlar todos los elementos de la escena, el sonido, la imagen, y las señales enviadas y recibidas por los dispositivos. Además, se utiliza un módulo para monitorizar todas estas señales en tiempo real. De esta forma se puede calibrar y controlar de forma sencilla.

Otro paso importante, ha sido el integrar todas las librerías que permiten el manejo de cada periférico en el mundo virtual, de forma que sus datos sean tratados como capas independientes. Esto permite una utilización transparente de los datos que nos ofrecen los periféricos por parte del mundo virtual sin tener que, para su configuración y calibración, entrar en detalles de cómo funcionan las librerías concretas de cada dispositivo.

El último paso imprescindible, ha sido el crear una serie de pruebas y utilizar a 23 sujetos que las realicen y rellenen unas encuestas. Con estos datos, y una evaluación de los parámetros objetivos de cada periférico, ha permitido llegar a las conclusiones finales que pueden verse al final de esta memoria.

Índice general

I	Introducción, Objetivos y Motivaciones	3
1.	Introducción	5
1.1.	Introducción	5
1.2.	Motivación	6
1.3.	Objetivo principal	6
1.4.	Objetivos complementarios	6
1.5.	Realización	7
1.6.	Esquema del Proyecto Fin de Carrera	8
II	Estado del arte	9
2.	Periféricos	11
2.1.	Introducción	11
2.2.	Los distintos periféricos	11
2.2.1.	El teclado	11
2.2.2.	El ratón	12
2.2.3.	El Joystick	14
2.2.4.	El Mando o gamepad	15
2.2.5.	Dancepad	16
2.2.6.	El WiiMote	17
2.2.7.	Kinect	18
2.3.	Conclusiones:	19
3.	Librerías	21
3.1.	Introducción	21
3.2.	Librerías utilizadas	21
3.2.1.	OpenSceneGraph (OSG)	21
3.2.2.	Simple DirectMedia Layer (SDL)	22
3.2.3.	Wiiyourself	23
3.2.4.	OpenNI, NITE y openkinect	23
3.2.5.	Glovepie	23
3.2.6.	FAAST, Flexible Action and Articulated Skeleton Toolkit	24

3.3. Programación:	24
3.4. Conclusiones:	24
III Desarrollo, diseño e implementación	25
4. Desarrollo	27
4.1. Introducción	27
4.2. Interfaces	28
4.3. Sistema de módulos	29
4.4. OpenSceneGraph	31
4.5. Simple DirectMedia Layer	31
4.6. Open GL	31
4.7. Conclusiones	32
5. Diseño	35
5.1. Introducción	35
5.2. Sistema de módulos	35
5.2.1. ¿En qué consisten?	35
5.2.2. Estructura de un módulo:	36
5.2.3. Seguridad y estándares:	36
5.2.4. Composición:	37
5.2.5. Diseño lógico del sistema:	38
5.3. Diseño de la Escena:	43
5.3.1. Sistema de referencia tridimensional:	43
5.3.2. Cámara:	44
5.3.3. Objetos y texturas:	44
5.3.4. HUD:	44
5.3.5. Luz:	44
5.3.6. Periféricos:	44
5.4. Diseño del menú inicial:	46
5.5. Diseño del HUD:	46
5.5.1. Requisitos:	46
5.5.2. Clases auxiliares:	47
5.5.3. Elementos del HUD:	47
5.5.4. Textos del HUD:	47
5.6. Diseño de la clase cámara:	47
5.6.1. Especificaciones y tipos:	47
5.6.2. Atributos:	48
5.6.3. Controles:	49
5.6.4. Diseño de los mapas de configuración:	49
5.7. Diseño de las texturas 2D:	50

5.7.1.	Objetivos:	51
5.7.2.	Tipos de estructuras 2D:	51
5.7.3.	La estructura Billboard:	51
5.7.4.	La estructura Skybox:	51
5.8.	Diseño de los efectos para objetos 3D:	51
5.8.1.	Efectos:	52
5.8.2.	Desplazar:	52
5.8.3.	Rotar:	52
5.8.4.	Gravedad:	52
5.8.5.	Viento:	52
5.9.	Diseño de los objetos 3D:	53
5.9.1.	Objetos y derivados:	53
5.10.	Diseño del sistema de control:	53
5.10.1.	Especificaciones:	53
5.10.2.	Administración de mapas de caracteres:	54
5.10.3.	Administración de mapas de configuración:	55
5.11.	Diseño del un laberinto aleatorio:	55
5.12.	Diseño de grupos:	55
5.12.1.	Player:	56
5.13.	Conclusiones	56

IV Evaluación 57

6. Metodología de evaluación 59

6.1.	Introducción	59
6.2.	Planteamiento del problema	59
6.3.	Planteamiento del método de medida	59
6.4.	Diseño del método de medida	61
6.5.	Conclusiones	62

7. Evaluación 63

7.1.	Introducción	63
7.2.	Potencia	63
7.3.	Toma de datos de intuitivo y comodidad	64
7.4.	Conclusiones	65

8. Toma de datos y conclusiones 67

8.1.	Introducción	67
8.2.	Conclusiones de la potencia	68
8.3.	Conclusiones de lo intuitivos que son los periféricos	70
8.4.	Conclusiones de lo cómodos que son los periféricos	72
8.5.	Conclusiones	74

V Conclusiones	75
9. Conclusiones del proyecto	77
9.1. Tabla de puntuaciones	77
9.2. Cuestiones	78
I. Implementación	XI
I.a. Introducción	XI
I.b. Herramientas y configuración del entorno de trabajo:	XI
I.b.1. Herramientas:	XI
I.b.2. Configuración:	XII
I.c. Implementación de los módulos:	XII
I.c.1. Engine Core:	XII
I.c.2. Module registry:	XIII
I.c.3. Input module:	XIV
I.c.4. Display Module:	XVI
I.c.5. Object Factory Object:	XIX
I.c.6. Audio Module:	XX
I.c.7. Timer Module:	XXI
I.c.8. Game Module:	XXII
I.d. Implementación de las clases:	XXIV
I.d.1. Tipo Audio	XXIV
I.d.2. Tipo Auxiliar	XXV
I.d.3. Tipo Cámara	XXV
I.d.4. Diseño de los mapas de configuración:	XXVI
I.d.5. Tipo Efectos	XXVII
I.d.6. Tipo Escena	XXX
I.d.7. Tipo Grupos	XXXI
I.d.8. Tipo HUD	XXXII
I.d.9. Tipo input	XXXV
I.d.10. Tipo Menu	XL
I.d.11. Tipo Módulos	XLI
I.d.12. Tipo Objetos3D	XLI
I.d.13. Tipo Salida	XLV
I.d.14. Tipo Laberinto	XLVI
I.d.15. Tipo Texturas	XLVII
I.e. Implementación de código común o general:	XLIX
I.e.1. Sistema de control	L
I.e.2. Sistema	LI
I.f. Conclusiones	LI

II. Qué hacen las clases y como usarlas	LIII
II.a. Introducción	LIII
II.b. Qué hacen las clases	LIII
II.c. Como usar las clases: (HOW TO)	LIII
II.c.1. Tipo audio	LIII
II.c.2. Tipo auxiliar	LIII
II.c.3. Tipo cámara	LIII
II.c.4. Tipo efectos	LIII
II.c.5. Tipo escena	LIII
II.c.6. Tipo grupos	LIV
II.c.7. Tipo HUD	LIV
II.c.8. Tipo input	LIV
II.c.9. Tipo menu	LV
II.c.10. Tipo módulos	LV
II.c.11. Tipo objetos3D	LV
II.c.12. Tipo salidas	LVI
II.c.13. Tipo texturas	LVI
II.d. Conclusiones	LVI
II.e. Encuesta de intuitivo	LVII
II.f. Encuesta de cómodo	LVII

Índice de cuadros

Índice de figuras

1.1. Concepto general del proyecto	7
2.1. Imagen del teclado	12
2.2. Imagen del ratón	13
2.3. Imagen del gamepad	15
2.4. Imagen del Wiiremote	17
2.5. Imagen del Kinect	19
3.1. Aspecto de un ejemplo de escena usando OSG	22
3.2. Aspecto de un ejemplo del uso de SDL	22
4.1. Aspecto del interfaz del usuario	28
4.2. Aspecto de uno de los menús de la consola	29
4.3. Capa de niveles del proyecto	33
5.1. Diagrama de módulos	39
5.2. Diagrama de llamada usando instancias	42
5.3. Sistema de coordenadas	43
5.4. Envío de datos de los periféricos	45
5.5. Pasos para actualizar una clase	54
7.1. Potencia de los periféricos	63
7.2. Potencia de los periféricos	64
8.1. Resumen datos de los periféricos I, con las categorías usadas	68
8.2. Resumen datos de los periféricos II, con las valoraciones finales	69
9.1. Resumen datos de los periféricos	77

Parte I

Introducción, Objetivos y Motivaciones

Capítulo 1

Introducción

1.1. Introducción

Este proyecto fin de carrera consiste en evaluar diferentes periféricos para determinar sus ventajas y desventajas para la interacción de un usuario en un mundo virtual. Para ello, se va a diseñar un interfaz multimodal que sirva de soporte para gestionar todos los periféricos que se van a estudiar, se va a crear un sencillo mundo virtual, y se va a evaluar cómo se desenvuelven un grupo de usuarios en él.

En los últimos años, las nuevas tecnologías han invadido casi todos los aspectos de nuestra vida. Desde la revolución industrial, la sociedad ha ido evolucionando con nuevos medios de información y entretenimiento. La radio, la televisión, internet . . . Se avanza en una dirección en la que es importante estar informado, y se valora el entretenimiento. Cada día crece el nivel de sofisticación de las nuevas tecnologías. Se ha avanzado mucho en la creación de realidades virtuales, realidad aumentada y los avatares 3D. En un sector cada vez más competitivo, se investiga buscando la forma de conseguir un mayor realismo y comodidad en el uso de estas nuevas tecnologías.

Una parte muy importante a tener en cuenta, es el sistema de control de estas nuevas tecnologías. Se busca que sea cómodo para los usuarios, sencillo, intuitivo, pero potente. Es importante además, resaltar la importancia de buscar periféricos que permitan una interacción lo más natural posible, y ser, de esta forma, más competitivos y atractivos para los usuarios. Es aquí donde reside la dificultad, la búsqueda de métodos que tengan un equilibrio óptimo entre estas características. Así pues, es razonable pensar, que merece la pena investigar nuevos periféricos y configuraciones que permitan avanzar en la forma que tenemos de interactuar con las nuevas posibilidades que como usuarios se nos ofrecen. Por ello, en este proyecto se van a analizar varios periféricos que han salido al mercado en los últimos años, y se van a comparar con los convencionales. Para ver si ofrecen mejoras, y cómo se pueden aprovechar estas mejoras.

1.2. Motivación

Queda claro que, los mundos virtuales, es una tecnología que se ha expandido mucho en los últimos años, y está cada vez más presente en nuestras vidas. Por contra, no se ha profundizado tanto en formas de interactuar y sumergir a los usuarios de los mundos virtuales para sacar un mayor provecho a gráficos cada vez más realistas, sonidos envolventes, y experiencias virtuales más fuertes y emocionantes. Es un mercado en expansión y continuo crecimiento.

Por tanto, la principal motivación será la búsqueda de la mejor forma de interactuar con este sector creciente, para que se puedan desarrollar controles en este sentido.

Otra motivación importante de este proyecto va a ser el conseguir una herramienta, que no sólo permita el manejo de varios periféricos, unos tradicionales y otros más innovadores, sino que se pueda observar con detalle toda la información que ofrecen, y se maneje con un control absoluto que permita la coordinación de todos los procesos involucrados en un mundo virtual.

1.3. Objetivo principal

El objetivo principal de este proyecto es analizar diferentes periféricos, determinar sus pros y contras, y concluir cual puede ser más adecuado para utilizar, en caso de que sea necesario interactuar con un mundo virtual.

Para lograr alcanzar este objetivo, va a ser necesario cumplir varios objetivos complementarios. Cada uno de ellos es un paso que nos acerca al objetivo principal del proyecto.

1.4. Objetivos complementarios

- Diseñar un mundo virtual sencillo a nivel gráfico y lógico, que permita la coordinación de todos los procesos necesarios para garantizar su estabilidad, una navegación básica a modo de cámara libre con 6 grados de libertad, y una serie de opciones para el control de datos y procesos.
- Adaptar una serie de librerías y módulos que permita manejar los distintos periféricos independientemente de la arquitectura de dichas librerías.
- Diseñar una interfaz común para poder manejar varios periféricos distintos, que ofrezca toda la información que aporta cada dispositivo, y poder controlar una serie de órdenes, funciones, y macros, en el mundo virtual.

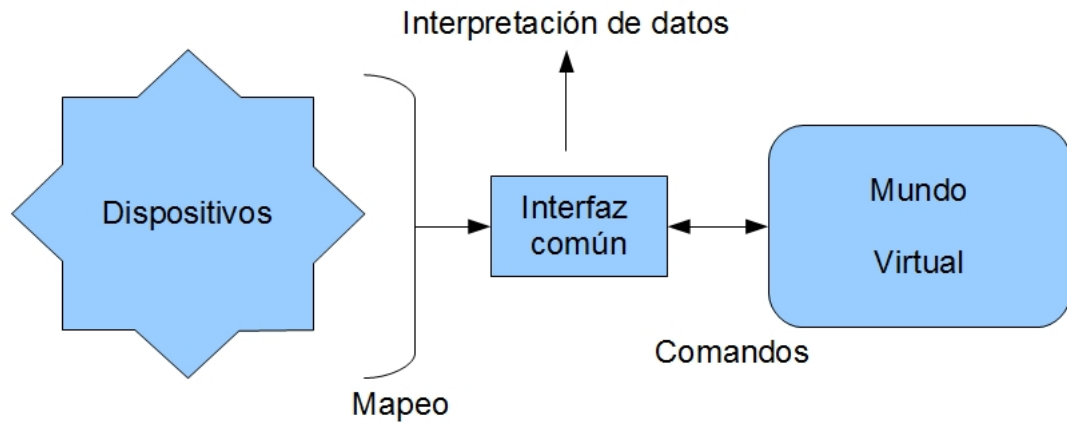


Figura 1.1: Concepto general del proyecto

- Evaluar la precisión de los distintos métodos diseñados para la interacción con el mundo virtual, analizando su calibración, potencia, pros y contras.
- Evaluar el uso de los periféricos por una serie de sujetos utilizando los métodos diseñados.
- Obtener conclusiones a raíz de los resultados.

1.5. Realización

Este proyecto se ha realizado para la empresa VICOMTech, en el área de animación 3D y entornos virtuales interactivos, en el marco de desarrollo I+D y con él se pretende orientar la toma de decisiones para el manejo de proyectos futuros.

1.6. Esquema del Proyecto Fin de Carrera

Este proyecto fin de carrera está compuesto por 5 partes, divididas en varios capítulos:

- La primera parte es la introducción al proyecto, declaración de objetivos y motivaciones. Contiene el primer capítulo.
- La segunda parte es el estado del arte. Incluye una investigación de los periféricos existentes en el mercado para poder utilizar en este proyecto, y de las herramientas disponibles para utilizarlos. Contiene el capítulo 2, con la investigación de los periféricos que se van a emplear, y el capítulo 3 con la investigación de las librerías disponibles.
- La tercera parte contempla todo el desarrollo y diseño del código. El capítulo 4, con el desarrollo, describe las directrices generales que debe cumplir el proyecto, para que pueda ser implementado. El capítulo 5 consiste en el diseño de todas las partes del desarrollo, es decir, qué se debe tener en cuenta para que se puedan cumplir los objetivos.
- La cuarta parte contiene la evaluación y conclusiones a las que se llega tras varias pruebas y estudios realizados a los distintos periféricos. El capítulo 6 es el planteamiento para la evaluación de todos los aspectos importantes de los periféricos. En el capítulo 7, se sacan las conclusiones a partir de los resultados.
- La quinta parte son las conclusiones a las que se llegan con este proyecto. Este último capítulo saca las conclusiones generales del proyecto.

Parte II

Estado del arte

Capítulo 2

Periféricos

2.1. Introducción

En este capítulo se van a describir los distintos periféricos que existen en el mercado, valorando a priori distintas cualidades de cada uno de ellos y se ha realizado una valoración de sus ventajas y desventajas.

2.2. Los distintos periféricos

2.2.1. El teclado

Descripción: Tradicional periférico indispensable hoy en día para interactuar de una manera cómoda y sencilla con un ordenador. Necesita usar superficie, preferiblemente plana, donde apoyarse para ser usado. Tiene un tamaño aproximado de 60 x 20 x 4 centímetros. Si se conecta por cable, no necesita alimentación externa, ya que se alimenta con él. Los inalámbricos necesitan pilas o baterías para funcionar.

Funcionamiento: Funciona mediante la detección de pulsación de teclas. Disponen normalmente de entre 80 y 120 teclas, lo que permite enviar información al ordenador e interpretarse mediante el código ASCII. Sirve para escribir cadenas de ellos en forma de palabras o comandos.

Características:

- Conexión: Mediante un cable con un puerto PS/2, USB o de forma inalámbrica mediante bluetooth.
- Compuesto por teclas únicamente.

PROS

- Fácil de usar.



Figura 2.1: Imagen del teclado

- Intuitivo.
- Barato.
- Potente para trabajar a bajo nivel, palabras o comandos.

CONTRAS

- Al sólo disponer de teclas, obliga al uso de comandos para aumentar sus posibilidades de uso.
- Aporta información absoluta. Una tecla está o no está pulsada.
- La información que ofrece sólo permite la ejecución de macros al pulsar una tecla. Puede interpretarse como escritura de un caracter para formar un comando, o desencadenar una serie de acciones pre-programadas.

2.2.2. El ratón

Descripción: Periférico de uso tradicional. Prácticamente todos los usuario de ordenador lo utilizan. Requiere una superficie plana donde usarlo, aunque se puede sustituir por touchpads; o lo que es lo mismo, una superficie táctil que emula al ratón. Es fácil de usar e intuitivo para manejar entornos gráficos. Su tamaño es aproximadamente 10 x 4 x 2 cm. Si se conecta por cable, no necesita alimentación externa, ya que se alimenta con él. Los inalámbricos necesitan pilas o baterías para funcionar.

Funcionamiento: Dispone de 2 o 3 teclas, para enviar información. Además da información del punto en el que está respecto a la pantalla o una ventana en un entorno gráfico y de su movimiento relativo desde la última muestra de posición. Además, muchos disponen de una rueda que permite enviar información si gira en un sentido u en otro y de cuánto gira.



Figura 2.2: Imagen del ratón

Características:

- Conexión: Mediante un cable con un puerto PS/2, USB o de forma inalámbrica mediante bluetooth.
- Compuesto por teclas y un dispositivo que le permite dar la posición de un plano en el que está.
- Opcionalmente dispone de una rueda para hacer scroll u otras acciones.

PROS

- Es fácil de usar.
- Intuitivo.
- Barato.

CONTRAS

- Puede ser algo impreciso.
- Manda menos información útil que el teclado.
- Requiere una superficie plana adecuada para utilizarse.
- Sustitutivos como los trackball no requieren superficie plana para usarse pero son más imprecisos e incómodos de usar.

2.2.3. El Joystick

Descripción: Este periférico ha sido tradicionalmente utilizado para manejar videojuegos. Aunque tienen muchos otros usos, por ejemplo para manejar robots (a corta y larga distancia), lanzamiento y guiado de misiles, telemedicina, manejo de herramientas por parte de personas discapacitadas, entre otros. Normalmente requiere una superficie de apoyo. Su tamaño varía mucho según el modelo. Si se conecta por cable, no necesita alimentación externa, ya que se alimenta con él. Los inalámbricos necesitan pilas o baterías para funcionar.

Funcionamiento: Es una palanca, que detecta cuánto la hemos girado sobre 2 planos. Suele disponer de botones adicionales y ruedas para incrementar la cantidad de acciones que se pueden realizar con él.

Características:

- Conexión: Mediante un cable USB o de forma inalámbrica mediante bluetooth. Antiguamente, aunque ya muy poco común, puertos anchos de pines como el IEEE 1284 o el PC Gameport.
- Compuesto por algunas teclas y una palanca que informa sobre el giro sobre 2 planos.
- Opcionalmente dispone de una rueda para hacer scroll u otras acciones.

PROS

- Es fácil de usar.
- Preciso para guiarte en espacios bidimensionales.
- Gama amplia de precios.
- Puede programarse para realizar movimientos progresivos.

CONTRAS

- Tiene unos usos muy limitados.
- Manda menos información útil que el teclado.
- Requiere una superficie adecuada para utilizarse, aunque no es algo tan crítico como el caso del ratón.
- En las versiones más baratas, no se detecta cantidad de giro sino que la palanca emula 4 botones.

2.2.4. El Mando o gamepad

Descripción: También este es un periférico muy utilizado en videojuegos, tradicionalmente en consolas. Dispone de un diseño más ergodinámico para adaptarse a la mano de un usuario. Es básicamente una pieza de plástico que dispone de varios botones y dispositivos para enviar información. Hay una amplia gama de tipos y modelos. No necesita superficie de apoyo, al ser tomado por el usuario. Su tamaño suele ser de aproximadamente 15 x 10 x 6 cm. Si se conecta por cable, no necesita alimentación externa, ya que se alimenta con él. Los inalámbricos necesitan pilas o baterías para funcionar.



Figura 2.3: Imagen del gamepad

Funcionamiento: Básicamente al pulsar los botones se realizan acciones. Suele disponer de otras fuentes de detección de eventos, como ruedas, que detectan cantidad de giro sobre 2 planos.

Características:

- Conexión: Mediante un cable USB o de forma inalámbrica mediante bluetooth. Antiguamente, aunque ya muy poco común, puertos anchos de pines como el IEEE 1284 o el PC Gameport.
- Compuesto por algunos botones, ruedas, y palancas que informa sobre el giro sobre 2 planos.

PROS

- Más cómodo que un teclado o ratón.
- No necesita una superficie en la que apoyarse.

- Gama amplia de precios.
- Si se dispone de Axis, se puede programar movimiento progresivo.

CONTRAS

- Tiene unos usos muy limitados.
- Manda menos información útil que el teclado.

2.2.5. Dancepad

Descripción: Periférico surgido a raíz de un videojuego que se popularizó a finales de los años 90. Requiere una superficie plana de apoyo que pueda soportar al usuario. Su tamaño aproximado es de 1,2 x 1,2 x 0,01 m.

Funcionamiento: Básicamente al pulsar los botones se realizan acciones, de forma similar a un joystick. No ofrece ejes, lo que supone una baja potencia por sí solo como dispositivo. Tiene aplicaciones muy concretas.

Características:

- Conexión: Mediante un cable USB.
- Compuesto botones que se pulsan con los pies.

PROS

- Interactúa con las piernas del usuario, que suele ser un aspecto poco aprovechado.
- Barato.
- Puede ser un buen complemento para ciertas acciones.

CONTRAS

- Tiene unos usos muy limitados.
- Manda menos información útil que el teclado.

2.2.6. El WiiMote

Descripción: Éste es un periférico relativamente nuevo. Lanzado al mercado por la compañía de videojuegos Nintendo en el año 2006, supuso una auténtica revolución en este campo. La razón, un sistema novedoso para interactuar más realista a todo lo visto anteriormente. Es un avance respecto al gamepad tradicional. Dispone de un detector óptico, que indica a dónde está mirando, y un acelerómetro, que mide la aceleración en los 3 ejes del espacio. Su salida al mercado como periférico fue incompleta, ya que no era demasiado preciso y no detectaba el tercer eje de rotación con su acelerómetro. Más tarde se sacaron una serie de periféricos para resolver los problemas que ofrece. No requiere ninguna superficie de apoyo, ya que igual que el gamepad, es tomado por el usuario. Su tamaño aproximado es de 12 x 3 x 3 cm. Dispone de varios conectores para conectar periféricos, además de un hueco para las 2 pilas AA que requiere como alimentación.



Figura 2.4: Imagen del Wiiremote

Funcionamiento: Dispone de varios botones, como un gamepad tradicional. El acelerómetro permite reconocer cuándo gira sobre los ejes perpendiculares al de la gravedad. Además, calcula cuánto está girado respecto al eje de la gravedad. Estos ángulos son llamados Roll y pitch. El tercer ángulo, llamado Yaw, no puede ser detectado sin utilizar el complemento wiimote plus. Este complemento además, confiere una mayor precisión, detectando con más exactitud la cantidad de giro en cada uno de los 3 ejes.

Características:

- Conexión: Inalámbrica con bluetooth.

- Compuesto por algunas botones, acelerómetros, y puntero que informa hacia que lugar está apuntando.

PROS

- Novedoso, con muchas nuevas posibilidades.
- No necesita una superficie en la que apoyarse.
- Intuitivo.
- La incorporación del acelerómetro supone un gran avance para el diseño de configuraciones que aprovechen esta característica.

CONTRAS

- Sin el complemento wiimote plus es impreciso en los movimientos que se realizan con él.
- Necesita complementos adicionales para el modo puntero, como es la barra de leds.
- Caro y frágil.

2.2.7. Kinect

Descripción: Nuevo periférico que salió al mercado en noviembre del 2010. Forma parte de los periféricos de nueva generación desarrollados para competir con el wiimote de Nintendo y buscar un concepto nuevo en la interacción con ordenadores y consolas. Se diferencia del resto significativamente porque ha sido diseñado para que el usuario no tenga que sostener ni manejar ningún objeto. Su tamaño aproximado es de 25 x 5 x 5 cm. Requiere estar apoyado en una superficie estable.

Funcionamiento: Una cámara graba imágenes de uno o varios usuarios. Utiliza un proyector de infrarrojos y una cámara especial para obtener información del mapa de profundidad. Sus drivers son capaces de distinguir varios usuarios, reconocer manos y movimientos, realizar un seguimiento del usuario etc. Todo esto permite interpretarse como entradas al sistema.

Características:

- Cámara RGB
- Cámara monocromo sensible en el rango de infrarrojos
- Proyector de infrarrojos
- 4 micrófonos
- Acelerómetro



Figura 2.5: Imagen del Kinect

PROS

- No necesita que el usuario sostenga ningún objeto.
- Experiencia más dinámica y sensación de inmersión.
- La incorporación de percepción de profundidad aporta múltiples posibilidades y usos.

CONTRAS

- Se tiene que realizar un procesamiento digital de la señal, por lo que conlleva un coste computacional mayor que cualquier otro periférico.
- Se deben cumplir unas condiciones muy concretas para que funcione correctamente. Entre ellas la calibración, campo de visión libre de obstáculos, etc.

2.3. Conclusiones:

La gama de periféricos es bastante amplia para el propósito que nos hemos propuesto. Si se combinan varios periféricos, puede aumentar exponencialmente las maneras de interactuar con un ordenador. Va a ser importante diseñar una gran variedad de configuraciones para hallar las mejores formas de interacción para sacar el máximo provecho a los periféricos y una experiencia lo más sencilla, intuitiva y con la mayor inmersión posible.

Capítulo 3

Librerías

3.1. Introducción

En este capítulo se van a describir las distintas librerías que se han empleado en el proyecto tanto para crear el mundo virtual, como para administrar los gráficos, sonido y periféricos. Estas librerías adicionales encapsulan funciones muy básicas para que no nos tengamos que centrar en detalles técnicos y podamos aprovechar al máximo los recursos que ofrecen. En función de las distintas librerías que se han encontrado disponibles y funcionales, se han seleccionado las más útiles a los propósitos del proyecto.

3.2. Librerías utilizadas

3.2.1. OpenSceneGraph (OSG)

Descripción: OpenScenegraph es un conjunto de herramientas open source, para diseñar sistemas 3D de alto rendimiento. Está escrito en lenguaje C++, es multiplataforma, y permite la manipulación a nivel medio de elementos en 3D.

Licencia: Utiliza la licencia OpenSceneGraph Public License. Permite su utilización en programas comerciales mencionando su uso. Las obras derivadas pueden ser de código abierto o propietario.

Utilidad: Tiene la capacidad de crear escenas y menús en 3D usando varios lenguajes de programación, varias estructuras de datos y librerías propias. Facilita el no usar las librerías de OpenGL a bajo nivel, siendo más sencillo usar el entorno tridimensional. Además, simplifica mucho la actualización de los modelos en 3D gracias a sus sistema de grafos. También gestiona texturas bidimensionales, lo que permite crear ilusiones y efectos usando imágenes.



Figura 3.1: Aspecto de un ejemplo de escena usando OSG

3.2.2. Simple DirectMedia Layer (SDL)

Descripción: Simple DirectMedia Layer es un conjunto de librerías open source, multiplataformas, que permite un acceso a bajo nivel al audio, periféricos y gráficos. Entre los lenguajes de programación soportados está C++, que se perfila como candidato ideal para el proyecto. Es un sistema muy popular utilizado para emuladores, juegos, administrar entradas y salidas a programas, sonido de aplicaciones, sistemas que empleen codificación MPEG o similares.

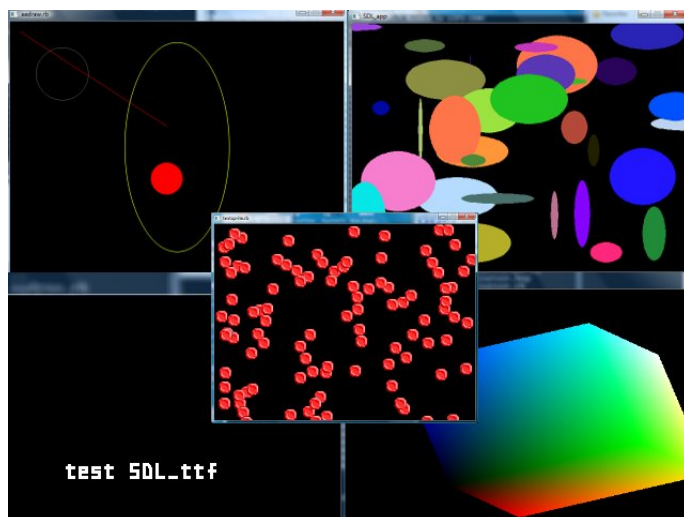


Figura 3.2: Aspecto de un ejemplo del uso de SDL

Licencia: Utiliza la licencia GNU LGPL versión 2. Permite su utilización en programas comerciales si se menciona su uso. Las obras derivadas pueden ser de código abierto o

propietario.

Utilidad: Permite el manejo a bajo nivel de todos los aspectos del mundo virtual. Gráficos, sonido, órdenes de entrada y salida.

3.2.3. Wiiyourself

Descripción: Es una librería implementada en C++ que permite manejar a alto nivel todas las funciones que ofrece el WiiMote.

Licencia: Utiliza la licencia opensource propia. Ver Anexo.

Utilidad: Servirá para el manejo de todas las características que ofrece el periférico WiiRemote.

3.2.4. OpenNI, NITE y openkinect

Descripción: Son un conjunto de librerías, disponibles en C++, que sirven como Framework para el manejo en PC del dispositivo kinect. Están en fase de desarrollo, pero existen versiones estables que permiten el manejo de la cámara RGB y la cámara de infrarrojos. Permite ver el vídeo que capta, grabarlo y dispone de varias funciones con reconocimiento de gestos y seguimiento de manos y esqueletos.

Licencia: OpenNI utiliza la licencia GNU GENERAL PUBLIC LICENSE. Por contra, las librerías NITE usan un Copyright (C) 2006 PrimeSense Ltd. All Rights Reserved. Aunque se facilita una clave de licencia gratuita para todo el que quiera trabajar con el código liberado para dicho fin en combinación con OpenNI.

Utilidad: Servirá para el manejo de todas las características que ofrece el periférico Kinect.

3.2.5. Glovepie

Descripción: Más que una librería es un programa externo que sirve para crear scripts. Dichos scripts son capaces de emular parámetros de entrada en un ordenador con privilegios de administrador, como salida de otros periféricos. Soporta múltiples teclados, joysticks, ratones, wiimotes, entre otros periféricos.

Licencia: Esta protegido por Copyright 2010 by Carl Kenner. Pero la atribución de los scripts es para el creador.

Utilidad: Sirve para emular distintas configuraciones de periféricos.

3.2.6. FFAST, Flexible Action and Articulated Skeleton Toolkit

Descripción: Otro programa basado en las librerías de OpenNi y NITE para reconocer posiciones y movimientos con kinect. Puede servir como complemento a Glovepie para conseguir datos del kinect como si fueran entradas de teclado, y luego procesarlas con Glovepie.

Licencia: Desconocida. Acaba de salir el programa para que pueda ser usado libremente por cualquiera y crear scripts. La intención de sus creadores, de la universidad del sudeste de California es liberar el código del programa más adelante.

Utilidad: Sirve para emular datos del esqueleto de kinect como entradas de teclado.

3.3. Programación:

El lenguaje elegido para realizar toda la codificación del proyecto ha sido C++. Aunque ya tiene 30 años, proporciona todas las herramientas necesarias para manejar las librerías. Desde punteros, hasta clases, encapsulamiento y herencia. Estas características no son necesarias pero sí recomendables para un proyecto de esta envergadura.

3.4. Conclusiones:

Gracias a SDL y OSG, vamos a poder crear desde 0 un mundo virtual, y a añadirle características y funcionalidades básicas. El lenguaje de programación óptimo a utilizar será C++.

El resto de librerías se utilizarán para la administración de los periféricos o partes concretas del código. En el anexo se facilitan las licencias de las librerías utilizadas, y referencias a tutoriales y ejemplos en los que se va a basar el proyecto.

El método que ofrece de emulación de los programas/librerías Glovepie y FFAST permite manejar un mundo virtual ya existente, simulando que se utiliza el teclado o el ratón. Es decir, permite enviar señales de un periférico, haciéndose pasar por otro, de esta forma, no hace falta tener un acceso al código fuente del mundo virtual, sino que se trabaja en una capa superior. Por ejemplo, podemos crear un script, que al levantar el brazo izquierdo, y este sea reconocido por kinect, se envíe la señal w del teclado, como está programado para responder a esta tecla, realizará la acción pertinente. A priori esto no es interesante ya que limita mucho la información que se puede enviar al mundo virtual, y que se parte de 0 a la hora de crearlo. Por tanto, es mejor diseñar un apartado que se encargue de administrar los datos de los periféricos, y manejar datos a un nivel más ofrecido, apoyándose en las librerías específicas de cada periférico. No obstante, no se puede descartar una utilidad una vez cumplidos los objetivos del proyecto, para comparar los 2 métodos y su funcionamiento.

Parte III

Desarrollo, diseño e implementación

Capítulo 4

Desarrollo

4.1. Introducción

En este capítulo se describe la idea general para la administración de todos los elementos del proyecto. Pretende dar una idea de la función de todas las partes del proyecto antes de comenzar el diseño concreto de cada una de ellas. Responde a la pregunta ¿Qué voy a necesitar desarrollar de forma global para concluir con éxito este proyecto? Otra forma de verlo es la siguiente, tengo un gran problema para evaluar los dispositivos, primero necesito un medio para administrarlos tanto a ellos, como todo lo que puedan hacer. Así que se coge ese gran problema y se organiza en diferentes niveles encargados de cada función específica necesaria.

Se busca que cada parte del proyecto sea independiente del resto. Es decir, aunque lógicamente las distintas partes se necesitan entre sí para un correcto funcionamiento de todos sus procesos, se trata de que cada una de ellas, ante ciertas condiciones de contorno devuelvan los correspondientes resultados, sin importar como están realizando estos cálculos. En otras palabras, ante unas entradas proporcionará una salida considerada correcta. En caso de no serlo, el sistema debe notificarlo.

La razón por la que usamos este método es simple. Este sistema ayuda a depurar errores de cada parte, sirve para poder añadir nuevas funcionalidades con el tiempo sin necesidad de preocuparnos de que puedan afectar a otras partes. Es un diseño eficaz al evitar tener puntos redundantes y da una visión más clara del funcionamiento a nivel global y local de cada parte del proyecto.

Aunque no está definido como uno de los objetivos, se ha considerado importante disponer de una estructura diferenciada en niveles que permita manejar todos los aspectos del proyecto según la necesidad de detalle o enfocado a distintos grados de manipulación. Estos niveles, vienen en parte marcados por las librerías externas en las que se apoya el proyecto, pero además en la estructura del código en las que se apoyan. Globalmente se puede considerar que este sistema está orientado a conseguir las funcionalidades más básicas de un motor gráfico comercial.

4.2. Interfaces

Esta parte es a la que tiene acceso el usuario final del proyecto. Mediante una serie de menús gráficos y opciones, se puede interactuar con el mundo virtual, a sus datos y a sus funciones. El acceso a los datos y funciones debe ser intuitivo.

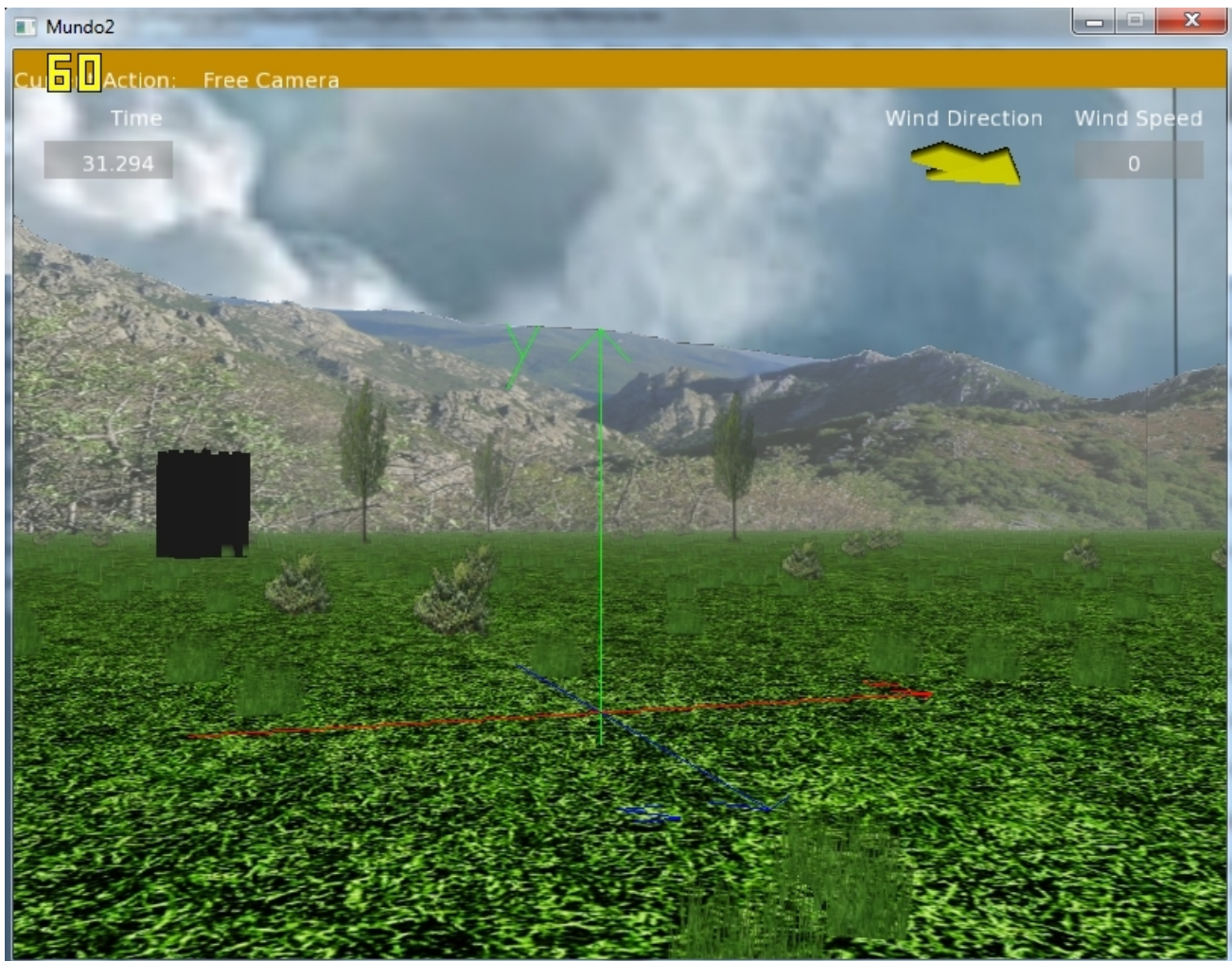


Figura 4.1: Aspecto del interfaz del usuario

Se puede considerar como el nivel "más alto" del proyecto. Este nivel debe poder manejarse intuitivamente y sin instrucciones. Cualquier usuario básico de un ordenador debe poder utilizarlo.

En resumidas cuentas, esta parte es el nivel Interfaz o de presentación, el más alto del sistema. Proporciona el interfaz final para:

- Manejar simple y llanamente todas las funciones del mundo virtual cargadas en los archivos de configuración.
- Modificar con menús todas las opciones que se permiten desde los niveles más bajos a éste.
- Acceder a toda la información básica que ofrece el sistema. Bien como un HUD cargado en el mundo virtual, o bien mediante menús o líneas de código por consola. Se puede prescindir de esta última parte, pero es recomendable tenerla.

Además, contará con un sistema de menús en la consola. Además de mostrar los avisos cuando ocurran, se podrá utilizar una serie de opciones como se puede ver en la figura.

```

Mundo 2 - Console
Menu 3.1. FreeCamera DATA
Position:      :-10.11 | 6.09 | 32.87 |
Velocity:      : 0.00 | 0.00 | 0.00 | Magnitud: || 0.00 |
Acceleration:  : 15.00 | 15.00 | 15.00 |
View Direction: | 0.31 | 0.01 | -0.95 |
Axi X:         : 0.95 | -0.00 | 0.31 |
Axi Y:         : -0.00 | 1.00 | 0.01 |
Axi Z:         : -0.31 | -0.01 | 0.95 |
Orientation:   : -0.99 | 0.01 | -0.16 | 0.00 || RSpeed: || 0.70 |
Console extra msg:
CAMERA MODE:
Spectator
GRAB MODE:
OFF
Current config:
9 - Back
  
```

Figura 4.2: Aspecto de uno de los menús de la consola

4.3. Sistema de módulos

Se pretende definir un sistema de módulos, para que, creando un sencillo motor lógico, cada aspecto técnico del proyecto funcione independientemente de los demás. Esto aporta flexibilidad para cambiar diversos aspectos sin afectar al resto, como cosas relativas al audio, a la visualización, o a la gestión de órdenes internas o externas.

Dicho sistema representa un nivel alto para el manejo de todos los sistemas, ya que proporciona funciones y técnicas para el manejo de todos los aspectos del proyecto. Además, se debe encargar de la coordinación de procesos. La coordinación es una parte necesaria, ya que de no poder coordinar eventos y funciones, el mundo virtual no funcionará correctamente, ni tampoco se podrá interactuar con él en tiempo real.

Por tanto, este sistema de módulos va a formar parte de la capa de gestión. Con esta capa se pretende

- Separar en módulos las distintas actividades del sistema.
- Independizar unos procesos de otros.
- Permitir un manejo más fácil y efectivo de cada una de las partes y del global de todas ellas.
- Coordinar el funcionamiento del motor.

La coordinación va a ser importante porque las entradas al sistema o inputs pueden llegar en cualquier momento, y además se tiene que estar continuamente realizando cálculos. Es decir, es un sistema asíncrono, no puedo esperar que un usuario solo dé ordenes en una franja temporal debe ser un proceso continuo, o al menos virtualmente continuo. Existen 2 formas de conseguir este efecto:

- Sobremuestreo de datos: Una parte de la estructura lógica del programa se encargará de leer los datos en un momento preciso de la ejecución. Si la frecuencia de muestreo es lo suficientemente alta, toda la información recogida podrá ser útil. En este caso se puede decir que el proceso es virtualmente continuo.
- Uso de distintos hilos de ejecución: Se puede programar varios hilos de ejecución, y usar uno concreto para captar todas las entradas al sistema. Si un ordenador dispone de sólo un núcleo o CPU, sólo es posible que los hilos sean virtualmente continuos. Si dispone de más, existen lenguajes de programación que permiten que funcionen de forma paralela real.

Para este proyecto se va a elegir la primera opción. Es mucho más sencilla de implementar, aunque hay que tener en cuenta en cuenta la coordinación de procesos, y además con los ordenadores actuales, y con el volumen de carga que se va a exigir al ordenador, va a funcionar correctamente. Por tanto, nos obliga al diseño de un módulo encargado de esta gestión. Una estructura estándar muy utilizada es la siguiente:

- Creación e inicialización: Parte en la que se preparan todos los módulos para su funcionamiento. Sin entrar en detalles de las necesidades de todos los procesos que tengan cada uno.
- Cuerpo: Esta parte se ocupa de unas acciones u otras en base a todo tipo de parámetros
- Update: Esta parte se encarga de actualizar los datos concretos de cada módulo. Pueden ser entradas, buffers de vídeo . . .
- Destrucción: Para cerrar adecuadamente una parte que ya no se necesita es necesario liberar la memoria que ha solicitado para funcionar.

Este diseño permite llevar a cabo todas las tareas necesarias para el correcto funcionamiento de todas las partes, de forma totalmente transparente para el usuario. En otras palabras, el impacto en la percepción del usuario es nulo para este sistema escalonado de procesos.

4.4. OpenSceneGraph

El uso de estas librerías permite cargar y manejar modelos y animaciones en 3D mediante el uso de grafos de datos. Dichos grafos están diseñados para abstraer al usuario, en este caso el programador de los módulos, del manejo a bajo nivel de estos.

Dicho sistema representa un nivel "medio". Con un lenguaje y un sistema de clases más o menos simple, podemos dibujar y manejar modelos en 2D y en 3D.

Va a pertenecer al nivel auxiliar para el manejo de librerías. Ya que es un conjunto de librerías que permiten abstraernos de operaciones de bajo nivel de las librerías de OpenGL que utiliza.

- Manejar cómodamente todos los modelos, imágenes, y menús, que se utilizan en el proyecto.
- Obtener y modificar datos sin preocuparnos de detalles de bajo nivel.

4.5. Simple DirectMedia Layer

Como ya se ha comentado anteriormente, SDL es un conjunto de librerías de bajo nivel que se encarga de manejar audio, gráficos, entradas y salidas del sistema.

Dicho sistema representa un nivel "bajo" para el manejo de todos estos aspectos. Si se profundiza en la estructura de las librerías, gracias a que son open source, se puede entrar con un nivel de detalle más bajo.

Va a pertenecer al nivel auxiliar para de manejo de librerías. Ya que es un conjunto de librerías que permiten abstraernos de operaciones de bajo nivel.

- Manejar cómodamente todos los modelos, imágenes, y menús, que se utilizan en el proyecto.
- Obtener y modificar datos sin preocuparnos de detalles de bajo nivel.

4.6. Open GL

Estas librerías son administradas directamente por OSG. Es decir, este proyecto se abstrae a un nivel de detalle medio en este aspecto.

No obstante, conviene tener en cuenta este aspecto, ya que si se precisa en algún momento un nivel de detalle más bajo, por ejemplo con el uso de cámaras y manejo de parámetros de matrices de visión, habrá que recurrir a este nivel.

Básicamente se encarga de:

- Manejar a bajo nivel de las opciones, administración de ventanas y representaciones gráficas.

4.7. Conclusiones

Esta jerarquía va a permitir una diferenciación de distintos usuarios y una abstracción de los niveles inferiores para que cada tipo de usuario no tenga que preocuparse del funcionamiento de las otras partes. Además, va a permitir el diseñar funcionalidades básicas orientadas a tener un pequeño motor gráfico que haga más sencillo la utilización y correcto funcionamiento de los periféricos y su análisis.

La única desventaja de este modelo, es que es algo complejo diseñarlo desde cero. Lo demás van a ser todo ventajas. A la larga es el mejor sistema que se puede adoptar ya que va a ahorrar muchísimo tiempo en el depurado y permite incorporar constantes cambios y mejoras. En la figura 4.7 podemos ver las capas de nivel que va a tener el proyecto.

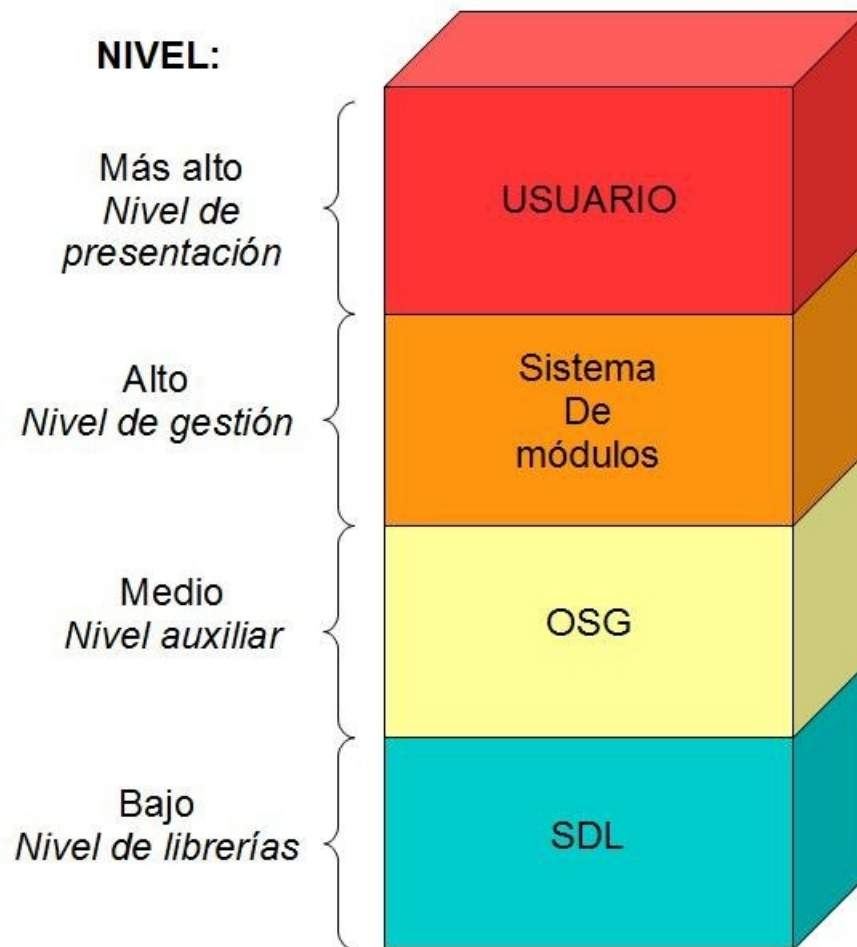


Figura 4.3: Capa de niveles del proyecto

Capítulo 5

Diseño

5.1. Introducción

En este capítulo se va a diseñar las distintas partes del programa que componen el proyecto para que cada una cumpla con su cometido. Expresado de otra manera, este capítulo responde a las preguntas ¿Qué voy a resolver en cada parte del código? ¿Cómo voy a resolverlo? ¿Quién (en referencia a las partes) va a resolverlo? y ¿Cuándo va a hacerlo?. Así pues, se plantean las especificaciones de cada problema y qué se necesita para resolverlo.

Concretamente, se buscan varios objetivos con el diseño:

- Minimizar problemas por parte del programador.
- Evitar código redundante.
- Asegurarse de que se reúnen los requisitos necesarios para un correcto funcionamiento de una sección y cumple con su cometido. Es más fácil y sencillo si las características relacionadas están en el mismo lugar.
- Evitar comportamientos del código inesperados al usar módulos independientes.

A continuación se van a diseñar cada una de las partes planteadas en el capítulo de desarrollo a un nivel más profundo, es decir, se divide cada gran problema en otros de más pequeña envergadura para llevar de una forma más sencilla a una solución satisfactoria.

5.2. Sistema de módulos

5.2.1. ¿En qué consisten?

Como ya se ha dicho en el capítulo de desarrollo, se precisan una serie de módulos que se coordinen entre sí y resuelvan distintos problemas de forma independiente. Por tanto, se va a diseñar una estructura base o plantilla aplicable a todos los módulos para que sean lo más

homogéneos posible en cuanto a estructura. Además, se establecen una serie de criterios o estándares a seguir para que sea más fácil manejarlos.

5.2.2. Estructura de un módulo:

Cada módulo, es una clase en el lenguaje de C++. Independientemente de su función, se compondrá de las siguientes partes:

- Constructor. Hay que definirlo obligatoriamente para un correcto funcionamiento de una clase en C++. Inicializa las variables y punteros públicos y privados de la clase.
- Destructor. Es importante liberar de forma correcta toda la memoria que ha sido solicitada por el módulo para funcionar. Éste es llamado al destruir la clase.
- Función init y/o derivados. Esta función, llamará a las funciones que el módulo necesite para poner en funcionamiento con los valores adecuados cuando se precise. Es importante no confundir con la parte del constructor, que simplemente asigna valores default a variables y punteros.
- Resto de funciones. Las demás funciones, serán llamadas por el propio módulo, o por otros, si son públicas, cuando la ejecución del programa así lo exija.
- Además, para tener a cada módulo localizado, usaremos un puntero estático instancia almacenado en un módulo de registro.

5.2.3. Seguridad y estándares:

Estándares de programación a seguir:

- Uso de elementos públicos y privados según corresponda.
- Uso adecuado de directrices `const`, `static` y `virtual`.
- Uso de técnicas como encapsulado y punteros.
- Cualquier creación de variables que reserve memoria de forma dinámica deberá ser correctamente liberada.
- Para las reservas de memoria se utilizarán excepciones, con los comandos típicos `try`, `catch` y `throw`.
- Por lo general, los nombres de variables y funciones, cada palabra comenzará por mayúscula, el resto en minúscula.
- Dichos nombres de variables deben ser lo más clarificadores posibles. Evitando nombres que den lugar a errores de concepto.
- Se usará comentarios para explicar cada función en el código.

Funciones y variables, públicas y privadas:

Para un mejor funcionamiento del código, se van a definir funciones públicas y privadas en los módulos. De esta forma, se evitan problemas si alguna vez un programador llama a funciones que no debería llamar o lo hace de forma indirecta. Siguiendo los estándares de programación:

- Variables públicas: Serán nombradas normalmente en minúsculas, con letra mayúscula al comenzar cada palabra.
- Variables privadas: Su nombre comenzará con un guión bajo.
- Ejemplos:

```
MiVariablePublica           NumeroDeCamara
_MiVariablePrivada         _RangoDeLaCamara
```

Directrices static y const: También se va a tratar de usar las directrices const y static al diseñar e implementar el código, para evitar errores. Aunque sea el programador el que diseña y emplea las funciones, ayuda mucho usar estas medidas de seguridad para minimizar los errores. Se detalla a continuación su correcto uso:

- Directriz static: Evita que la variable que usa un programa se destruya al salir de su ámbito, es decir, siempre está disponible para ser usada. Sirve por ejemplo para tener punteros a las distintas clases que vamos a utilizar y poder utilizarlos siempre, hasta que decidamos destruirlos.
- Directriz const: Se puede aplicar tanto a variables como funciones. En el caso de las variables, se consigue que ese dato NUNCA pueda ser modificado durante la ejecución del programa. Útil al definir parámetros por default. En el caso de funciones, evita que esa función, o cualquiera de las que llame, modifique ninguna variable, de este modo nos aseguramos de que las funciones de lectura no afecten a las variables.

5.2.4. Composición:

Los módulos que se van a implementar son los siguientes:

- Módulo de Audio: Será el encargado de todas las gestiones a bajo nivel del audio. Esto incluye efectos de sonido y música. Para ello usará las librerías SDL y SDL Mixer.
- Módulo de Gráficos: Será el módulo encargado de manejar el sistema de ventanas, las opciones gráficas de alto nivel, y las salidas de datos del sistema.
- Módulo de Entradas: Será el encargado de toda la gestión de datos de entrada. Es decir, administra y controla los periféricos con los que mandamos información al ordenador.
- Módulo de tiempo: Será el encargado de coordinar los ticks del reloj proporcionado por las librerías del OSG y SDL. Además brindará varias funciones para calcular diferencias de tiempos, además de guardar los tiempos necesarios de algunos procesos generales.

- Módulo game: Este módulo se encargará de toda la gestión lógica de nuestro mundo virtual. Será el módulo encargado de llevar el hilo del proceso de todos los menús y escenas.
- Módulo de creación de objetos: Éste módulo será el encargado de cargar y descargar todos los objetos que se carguen en una escena a nivel lógico. Según el tipo de objeto, solicitará a OSG la memoria de forma correcta para que funcione. De este modo, otros módulos podrán llamar a sus funciones para crear los objetos en el mundo virtual.
- Registro de módulos: Este será un módulo especial, que contenga un puntero al resto de módulos para tenerlos siempre localizados y tener una estructura organizada y ordenada. No es imprescindible, pero ayuda mucho en la gestión del proyecto. Crea e inicializa el resto de módulos.
- Engine Core: Este será, por así decirlo, el núcleo de todo el proyecto. Será el módulo que mande cargar al registro de módulos todos los módulos necesarios para el correcto funcionamiento del sistema. Llama crear al registro de módulos.

La idea es que todos los módulos sigan un orden concreto para cargarse y evitar problemas, ya que unos dependen de otros para realizar su trabajo. Cada módulo usará una sección de inicialización, además de la del constructor, para tener unos valores concretos de inicio. Esta función normalmente será llamada al cargar el módulo, por el registro de módulos. En definitiva, la carga de todo el sistema quedaría como se observa en la figura 5.2.4

Los módulos son cargados en el orden que muestra la figura 5.2.4. Hay que tener en cuenta que los módulos se comunicarán entre sí libremente mediante el uso de punteros, así que éste no es el orden en el que operan. Para más detalles consultar el anexo con la documentación de mundo2.

5.2.5. Diseño lógico del sistema:

Como se puede ver en la figura 5.2.4 , una vez cargados e iniciados los módulos, se pasa el control de la actividad al módulo game. Éste, al ser el encargado de gestionar a nivel lógico todo lo que vamos a mostrar, deberá seguir un orden muy concreto de ejecución para usar cada uno de los otros módulos en su contexto y sin causar errores.

5.2.5.1. Sincronización:

Ya se ha comentado en la parte de desarrollo lo importante que es el sincronizar los distintos puntos del programa. Sólo vamos a disponer de un hilo de ejecución. Así que se seguirá la estructura secuencial diseñada en el apartado 5. El ordenador mide el tiempo en ticks, es decir, en periodos de tiempo que dependen de la velocidad del procesador. Así que se deberá guardar este valor cuando sea necesario para calcular diferencia de tiempo.

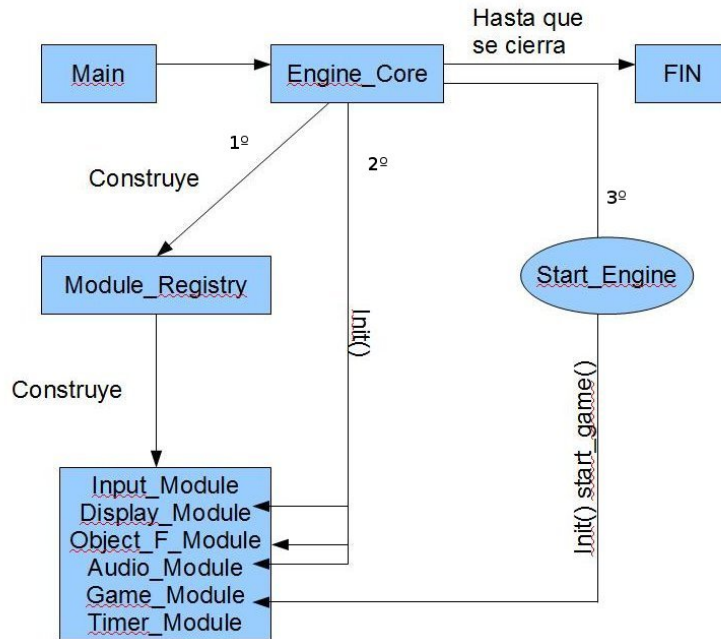


Figura 5.1: Diagrama de módulos

Este tema es importante dado que se está diseñando un mundo virtual. Al ser un sistema asíncrono por niveles, dependiendo de la cantidad de código que se ejecute en cada iteración, se tardará un tiempo variable. Si es poco código la diferencia será pequeña cada vez, pero conforme crece la cantidad de código a procesar este crece. ¿Por qué éste es un aspecto importante? Porque el mundo virtual simula física. Si cada iteración se ejecuta a distinto ritmo, ¿Cómo puedo actualizar correctamente los objetos del mundo? Se debe por tanto calcular, el tiempo entre actualizaciones, para calcular según la velocidad y aceleración de los objetos en qué lugar deben ser renderizados.

Cabe destacar además, que puede ser importante saber cuándo ocurrió un proceso como un click de ratón, el paso por un punto determinado del mundo . . . El módulo de tiempo será el encargado de ofrecer acceso a todos estos datos de una forma sencilla.

5.2.5.2. Menús:

Al dar el control al módulo game, éste crea el menú de inicio y espera a que el usuario elija que hacer. Mientras el usuario no decida salir del programa, esperará las elecciones. El primer menú será muy sencillo:

- Start world: Esta opción creará e iniciará la escena del mundo virtual. Además, creará e iniciará toda clase que sea necesaria.

- Exit: Mandará destruir todos los módulos y cerrará el programa.

5.2.5.3. Escena:

La creación de la escena es un poco más amplia. Necesitaremos definir qué elementos precisa la escena, cuándo van a ser creados y destruidos, y qué valores deben tener. La escena se va a componer de:

- Objetos 3D. Son los objetos que van a ser visibles. Algunos serán interactivos y otros no.
- HUD. Objetos visibles bidimensionales. Es decir, una serie de paneles con información importante.
- Luz. Se ha de definir las fuentes de luz para visualizar correctamente todos los elementos de la escena.
- Cámara. Ya sea fija o móvil, se necesita 1 o varias cámaras para observar la escena.
- Periféricos. Debe estar preparada para utilizar todos los periféricos del proyecto.

5.2.5.4. Consola:

Se creará una consola para tener una visualización básica, rápida, y sencilla de los parámetros que maneja el programa. Debe cumplir una serie de requisitos:

- Funcionamiento paralelo a la escena.
- Funcionamiento parando a la escena.
- Debe ser lo más sencilla y clara posible.
- Debe mostrar los datos de forma concisa y fácil de leer.
- Debe tener la capacidad de realizar múltiples acciones.
- Debe estar organizada por secciones de forma clara.
- Debe ser lo más intuitiva posible.
- En la implementación debe quedar clara la estructura para poder editarla.

Cada módulo es creado por el registro de módulos. Pero, ¿Cómo puede un módulo usar las funciones que ofrece otro distinto? Para resolver este problema, cada módulo contendrá una variable puntero, del tipo de la clase que sea, que apuntará a sí misma. De esta forma, si obtenemos este puntero, podemos operar con las funciones y variables del módulo públicas, como si hubiéramos declarado nosotros mismos el objeto.

El método deberá cumplir los requisitos que vienen a continuación:

- La instancia, o puntero a la propia clase deberá ser privado. De este modo se garantiza que nadie ajeno al módulo pueda borrarlo u ocupar memoria que sea requerida por él mismo.
- Será necesaria pues, una función que dé una copia del puntero, del tipo que sea la clase que lo contiene. Dicha función será:

```
get_instance()
```

- Para asegurarnos que NUNCA se devuelva un puntero con información basura, debido a que dicho objeto no existe y nos da una dirección con información aleatoria, si no existe tal puntero, lo construirá, y si existe lo devolverá.

Es importante tener en cuenta que:

- Antes de asignar un valor al puntero, como es static, se debe inicializar como 0 o NULL, sino no podremos compilar el programa.
- El método que evita devolver un puntero con información basura, la primera llamada reservará dinámicamente memoria con el comando `new`. Cuando dejemos de usarlo, habrá que destruirlo con el comando `delete`
- Para conseguir el puntero bastará con usar la función

```
get_instance()
```

con una llamada estándar

```
variable puntero tipo Clase = NombreDeLaClase::get_instance()
```

y se podrá usar entonces referenciado al puntero con la sintaxis

```
variable puntero tipo Clase ->NombreDeLaFuncion();
```

o podremos guardar la dirección de memoria y usarla como un objeto de la siguiente forma:

```
TipoDeLaClase& nombreX = *_PunteroALaClase->get_instance();
nombreX.NombreDeLaFuncion();
```

- Gráficamente el funcionamiento se puede ver en la figura:

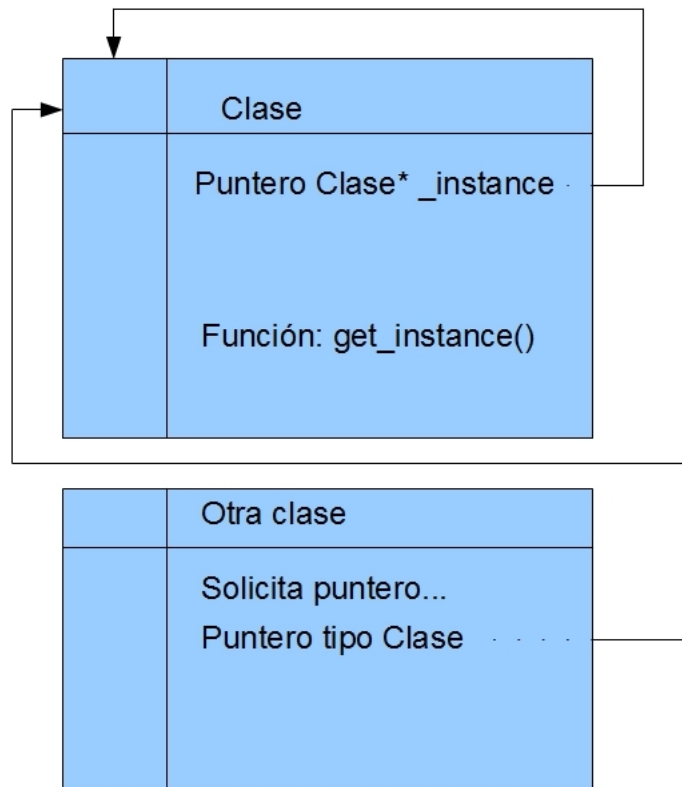


Figura 5.2: Diagrama de llamada usando instancias

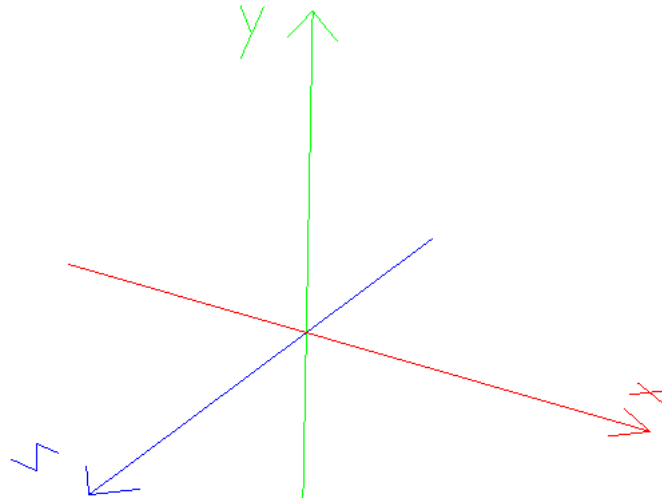


Figura 5.3: Sistema de coordenadas

5.3. Diseño de la Escena:

5.3.1. Sistema de referencia tridimensional:

Los ejes serán los habituales para los dibujos 3D.

- X: Asociado al color rojo. Formará el plano del suelo con el Eje Z.
- Y: Asociado al color verde. Definirá el concepto de altura.
- Z: Asociado al color azul. Formará el plano del suelo con el Eje X.

Para describir los ángulos, existen varias convenciones.

Nº giro	Avión	Telescopio	Símbolo	Vel.Angular
1º	Heading	Azimuth	θ (theta)	Yaw
2º	Attitude	Elevation	ϕ (phi)	Pitch
3º	Bank	Tilt	ψ (psi)	Roll

Utilizaremos la siguiente convención:

- Yaw: Giro en el plano formado por X-Z.
- Pitch: Giro en el plano formado por X-Y.
- Roll: Giro en el plano formado por Y-Z.

5.3.2. Cámara:

La cámara también es un aspecto importante de la escena. Define en que lugar está el espectador, la distancia de renderizado a la que puede ver, si tiene la posibilidad de moverse y cómo lo hace . . . Por esa razón, se va a diseñar una clase que contenga todas las acciones que pueda realizar la cámara.

5.3.3. Objetos y texturas:

Hay múltiples objetos 2D, 3D y texturas que estarán presentes en la escena, así que se van a emplear varias clases para utilizar a cada tipo encapsulado de una manera independiente.

- Hay que mencionar que los billboards, skybox y el suelo, son texturas 2D, que ayudan a la inmersión por parte del usuario en la realidad virtual. Así que serán clases distintas.
- Los objetos 3D tienen varias propiedades, pueden ser: móviles, inmóviles, visibles y no visibles. En función de ellos, se creará una clase general y derivados de ésta con propiedades distintas.

5.3.4. HUD:

El HUD es la forma que tiene el proyecto de mostrar por pantalla información que se considera importante. Se va a diseñar una clase que administre el HUD, que a su vez dividirá su trabajo en otras clases. Con el HUD se pretende que el usuario tenga una información breve y concisa sobre parámetros básicos. El HUD de pruebas dispondrá de un contador con el tiempo transcurrido, y la dirección del efecto del viento.

5.3.5. Luz:

Existen varios tipos de luz que puede manejar OSG. Poseen 4 parámetros al colocar cada tipo de luz, cantidad de rojo, azul, verde y transparencia. Todos valores entre 0 y 1 tipo float. Además, hay otras propiedades que pueden ser modificadas. No se ha creado una clase específica para esta tarea, pero podemos añadir fuentes de luz nuevas a la escena si: Ver HOW TO II. Se pueden emplear los siguientes tipos de luz:

- Luz ambiental, para emular una fuente lejana que ilumine todo el escenario. La luz viene de todas direcciones.
- Luz difusa, emula la luz emitida desde un punto concreto.
- Luz especular, emula la luz que refleja un objeto.

5.3.6. Periféricos:

Los periféricos no forman parte de la escena directamente, pero sirven para interactuar con ella. Por tanto, cada periférico va a necesitar:

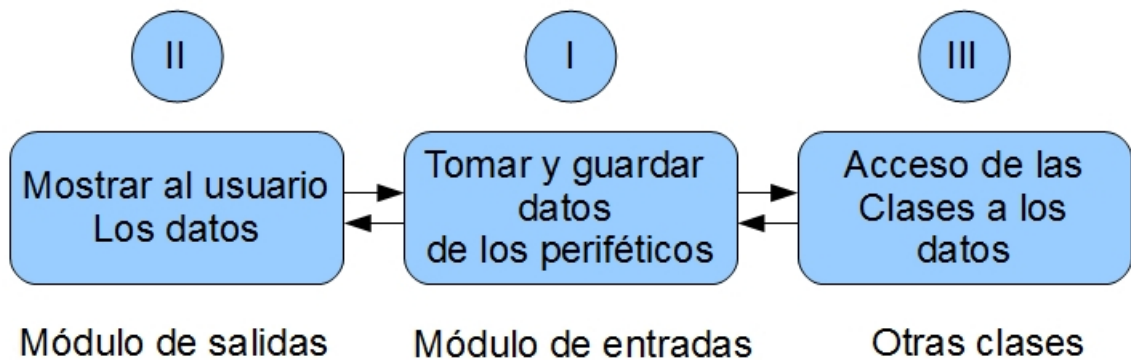


Figura 5.4: Envío de datos de los periféricos

- Ser cargado y reconocido por el sistema dentro de las escenas
- Disponer de varias configuraciones que puedan ser cambiadas cuando se desee en la misma escena
- Control de datos para calibrar y modificar los parámetros de los periféricos
- Abstracción total de uso para las capas de usuarios superiores

Como se van a poder manejar varios tipos de cámara, con múltiples formas de interacción, se recomienda diseñar un sistema que permita cambiar de forma de una manera sencilla. Este sistema se encargará de:

- Revisar todos los periféricos (Módulo input)
- Que detecte el teclado en todos los menús y escenas (Clase Control Map ∈ Módulo input)
- Que muestre de forma clara los datos (Clase Console ∈ Módulo de Vídeo)
- Que cualquier clase que necesite la información de los periféricos, disponga de un acceso lo más sencillo posible

A raíz de la figura, se concluye que el mejor método para cumplir los requisitos será:

- La parte I recolecta la información de cada periférico, siempre de la misma forma, y la almacena para que pueda ser interpretada.
- La parte II, se limita a mostrar esa información, sin alterarla.

- La parte III, cada periférico, hace que actúe de una forma y otra la clase que lo lea, usando archivos de configuración.
- Cada paso tiene que ser independiente del resto.

5.4. Diseño del menú inicial:

Se utilizará la clase HUD para que el usuario pueda verlos. Debe conseguir que el usuario pueda:

- Moverse por las opciones del menú
- Cargar el mundo virtual
- Salir del programa de forma correcta

5.4.0.1. Estructura:

Dispondrá de 3 estados distintos:

- Default: El usuario está en el menú inicial
- Start World: El usuario ha elegido la primera opción
- Exit: El usuario ha decidido cerrar el programa

5.4.0.2. Transiciones:

El estado default significa que el usuario está eligiendo. Si marca la opción empezar mundo, pasará a este estado, hasta que se deje de simular el mundo virtual, y volverá al estado default. En caso de marcar exit, se saldrá del programa.

5.5. Diseño del HUD:

Para que se puedan visualizar distintos HUDs, como el menú de inicio, o los paneles de información de la escena, se requiere una clase concreta que cumpla los siguientes requisitos.

5.5.1. Requisitos:

- Cree un HUD que esté fijo en la pantalla
- Debe tomar los datos automáticamente, de forma transparente en cuanto la proyección
- Se puedan agregar elementos al HUD que posean varias cualidades editables
- Se debe poder agregar texto a los elementos el HUD
- Se debe poder cargar, editar y descargar en todo momento los elementos del HUD

5.5.2. Clases auxiliares:

Se definirán unas clases auxiliares para que se encarguen de gestionar los elementos y los textos del HUD

- Clase elementos del HUD: Se ocupará de añadir, editar y borrar los elementos del HUD
- Clase textos del HUD: Se encargará de añadir, editar y borrar los textos del HUD

5.5.3. Elementos del HUD:

- Se podrán colocar en cualquier posición
- Se definirá un tamaño
- Podrá mostrarse u ocultarse

5.5.4. Textos del HUD:

- Se podrán colocar en cualquier posición, dentro de un elemento del HUD
- Se podrá editar el texto
- Podrá mostrarse u ocultarse
- Se guardará en un vector con textos, el elemento consultara la posición a escribir

5.6. Diseño de la clase cámara:

Este es un aspecto importante. Como el objetivo principal es encontrar el periférico más adecuado para interactuar con el mundo virtual, deberemos tener una cámara con muchos posibles movimientos y funciones.

5.6.1. Especificaciones y tipos:

- Debe tener 5 o 6 grados de libertad (6 DOF). Es decir, debe poder moverse y girar respecto a todos los ejes.
- Debe disponer de varios tipos de navegación, en los que el usuario pueda moverse por el escenario de forma acorde al sistema establecido.
 - Modo espectador:
 - Adelante y atrás: Respecto hacia donde mira.
 - De lado: Al lateral donde mira manteniendo Y constante.
 - Altura: Puede aumentar o decrementar su posición y como desee.
 - Giros: Yaw y Pitch.

- Modo primera persona:
 - Adelante y atrás: Respecto plano X-Z
 - De lado: Al lateral donde mira manteniendo Y constante.
 - Altura: Puede aumentar o decrementar su posición y como desee.
 - Giros: Yaw y Pitch.
- Modo avión:
 - Adelante y atrás: Respecto hacia donde mira.
 - De lado: No tiene.
 - Altura: Puede aumentar o decrementar su posición y como desee.
 - Giros: Pitch y Roll.
- Modo orbital:
 - Adelante y atrás: No tiene.
 - De lado: No tiene.
 - Altura: No tiene.
 - Giros: Yaw y Pitch, mirando a un punto.
- Estar basada en cuaterniones para realizar las operaciones con facilidad.

5.6.2. Atributos:

Para dar una sensación de realismo, estableceremos varios parámetros que se puedan adaptar a cada situación.

- Se debe poder colocar la cámara en un punto fijo, mirando en una dirección concreta.
- Se debe poder también mover por el espacio, respecto a los ejes o respecto a la rotación en un momento dado.
- Dispondrá de velocidad y aceleración variable. Su regulación permitirá alcanzar más velocidad máxima, y llegar a ella según el parámetro aceleración.
- Dispondrá de unos parámetros para limitarla en el espacio, mediante un sencillo sistema de colisiones.
- Para dar realismo, si se deja de acelerar, la velocidad se reducirá progresivamente hasta llegar a cero.
- Deberá ofrecer la posibilidad de variar la velocidad de giro.
- Por defecto, tendremos unos valores constantes. Éstos se cargarán en las variables correspondientes para poder simular en cada momento de la ejecución del mundo virtual un valor u otro según corresponda.
- Deberá exportar todos los datos en un formato que pueda ser interpretado por OSG, y será accesible por la consola y el módulo de display.

5.6.3. Controles:

Según el dispositivo, se utilizará un conjunto de configuraciones u otro. Como se van a emplear dispositivos bastante dispares, es imposible diseñar un método común a todos ellos, aunque deberemos tener en cuenta algunos aspectos.

- El teclado requerirá pulsaciones simples y continuas. Es importante que la cámara pueda distinguir los 2 casos para un comportamiento y control óptimo. Si la acción al pulsar una tecla es simple, solo se realizará una vez, independientemente de cuanto tiempo el usuario pulse la tecla. En caso de ser continua, mientras esté pulsada se realizará la acción solicitada.
- Otros dispositivos que no usen botones para ciertas acciones, deben poder adaptarse de una forma transparente a este funcionamiento de la forma más simple posible. Es algo más complejo ya que normalmente se dispondrá de datos numéricos de ciertos parámetros, con lo que se debe gestionar una serie de rangos de funcionamiento.

5.6.4. Diseño de los mapas de configuración:

Cada periférico posee varias configuraciones para el manejo de los objetos que lo permitan. Es interesante, antes de implementar, tener diseñado qué debe hacer cada configuración. El modo primera persona mueve adelante y atrás en el mismo plano independientemente de hacia dónde se mire. El modo de cámara flight cambia el eje yaw por el roll.

5.6.4.1. Mapas del Teclado:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- Movimiento habitual, con teclas AWSD; Rotaciones con RFDG; Altura con QE;
- p - Abre el menú.
- 1-4 - Cambia el tipo de cámara.
- 1-9(Num) - Permite elegir las opciones del menú.
- Barra espaciadora - Utiliza la función init de la cámara, posicionandola con su configuración por defecto.
- +/- (Num) - Sube y baja respectivamente la velocidad de rotación de la cámara.

5.6.4.2. Mapas del Ratón:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- Movimiento horizontal - Giro diferencial Yaw.
- Movimiento vertical - Giro diferencial Pitch.

5.6.4.3. Mapas del Joystick/Dancepad:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- Botones 6 y 8 - Aumentar y disminuir respectivamente el valor de la posición en el eje Y.
- Botones 2 y 3 - Movimiento adelante y atrás.
- Seta izquierda - Movimiento gradual.
- Seta derecha - Giro gradual.

5.6.4.4. Mapas del WiiMote:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- Giro Roll del wiimote - Girar la cámara en yaw.
- Giro Pitch del wiimote - Girar la cámara en pitch.
- Botones +/- - Cambiar la velocidad de rotación.
- Flechas de la cruceta - Movimiento.
- Seta del Nunchuk - Movimiento.

5.6.4.5. Mapas del Kinect:

Configuración por Default. Con esta configuración se pretende una navegación como si se operara con el plano de cara al usuario.

- Adelantar/Retrasar pierna derecha - Movimiento.
- Mano derecha sobre/bajo izquierda - Giro Yaw.
- Mano izquierda separada de derecha (que está en el centro) o viceversa - Giro Pitch.

Configuración 1. Con esta configuración se pretende una navegación utilizando el ángulo en el que están colocados los brazos.

- Brazo izquierdo - Movimiento.
- Brazo derecho - Giros.

5.7. Diseño de las texturas 2D:

La escena va a contener algunas texturas en 2D, para crear ciertos efectos con un rendimiento mejor que usando objetos 3D.

5.7.1. Objetivos:

- Menor gasto de memoria
- Sensación de un entorno amplio
- Efectos ópticos

5.7.2. Tipos de estructuras 2D:

Se definirán unas clases auxiliares para que se encarguen de gestionar los elementos y los textos del HUD

- Billboard: Textura 2D que siempre está orientada hacia la cámara, de forma que da la impresión de no ser una figura plana
- Skybox: Textura 2D lejana, que encajona la escena, para crear la sensación de estar en un entorno amplio, cuando en realidad sólo es una fotografía.

5.7.3. La estructura Billboard:

- Se podrán colocar en cualquier posición del mundo virtual
- Se podrá colocar cualquier textura.
- Se definirá un tamaño de textura y un color
- Podrá mostrarse u ocultarse

5.7.4. La estructura Skybox:

- Se deberá colocar en la posición de la cámara, para crear el efecto deseado
- Se podrá asignar cualquier imagen de fondo, incluso cambiarla
- Podrá mostrarse u ocultarse

5.8. Diseño de los efectos para objetos 3D:

Para simular a un nivel básico elementos de un mundo físico real, se van a diseñar sencillos efectos, que aplicados a la PAT del árbol OSG adecuadamente, van a simular la física del mundo virtual de forma sencilla.

Por tanto, debe existir una clase general que sea objeto, y unas clases derivadas que puedan contener uno o varios de los efectos que se van a crear.

5.8.1. Efectos:

- Desplazar
- Rotar
- Gravedad
- Viento

5.8.2. Desplazar:

Efecto que simula que un objeto pueda moverse por el espacio. Se debe cumplir que:

- Pueda desplazarse el objeto en cualquier dirección del espacio (x,y,z)
- Se debe poder realizar sobre cualquier objeto derivado de la clase objeto

5.8.3. Rotar:

Efecto que simula que un objeto rote sobre sus ejes en el espacio. Se debe cumplir que:

- Pueda rotar sobre cualquier eje espacio (x,y,z)
- Se debe poder realizar sobre cualquier objeto derivado de la clase objeto

5.8.4. Gravedad:

Efecto que simula el efecto de la gravedad sobre un objeto, es decir, decrece en el eje Y con el tiempo. Se debe cumplir que:

- Decremento en el eje Y
- Debe emular la ley de la gravedad sobre la superficie de la tierra
- Se debe poder modificar la constante de gravedad
- Se debe poder realizar sobre cualquier objeto derivado de la clase objeto

5.8.5. Viento:

Efecto que simula el efecto del viento sobre un objeto

- Movimiento en el plano de la tierra, ejes X-Z
- Carácter variable
- Se debe poder realizar sobre cualquier objeto derivado de la clase objeto

5.9. Diseño de los objetos 3D:

Cada objeto 3D que se va a utilizar en la escena, puede tener distintas propiedades, pero todos van a tener unas características comunes, que son:

- Identificador
- String con descripción
- Tipo de objeto

Por tanto, se puede crear una clase básica tipo objeto que contenga estos atributos. Para que pueda tener atributos extra, como pueden ser los efectos descritos en el apartado anterior, se creará una clase derivada que pueda contener uno o varios efectos. Para crear objetos concretos, se puede derivar una vez más para crear esos objetos, y poder cargar el tipo concreto tantas veces como sea necesario.

5.9.1. Objetos y derivados:

- Object: El más simple

Derivados de Object

- Movable: Dispone de vector con punteros a efectos

Derivados de Movable

- Sheep: Modelo oveja, al que afecta la gravedad.
- Castle: Modelo castillo, inmóvil.
- Catapult: Modelo catapulta, inmóvil.
- King: Modelo rey, inmóvil, con colisión.
- Arrow: Modelo flecha, con orientación.

5.10. Diseño del sistema de control:

5.10.1. Especificaciones:

Es necesario, para cumplir con la independencia entre lectura de los datos de los periféricos, su lectura, y su interpretación, disponer de una estructura que se pueda implementar en cada clase que vaya a utilizar los datos de los periféricos como entrada para realizar alguna acción. Además, es importante que la toma de datos sea lo suficientemente rápida para disponer de suficientes muestras que hagan la experiencia del usuario continua. También es importante actualizar lo suficientemente rápido para conseguir que funcione de forma fluida y sin retardo para el usuario.

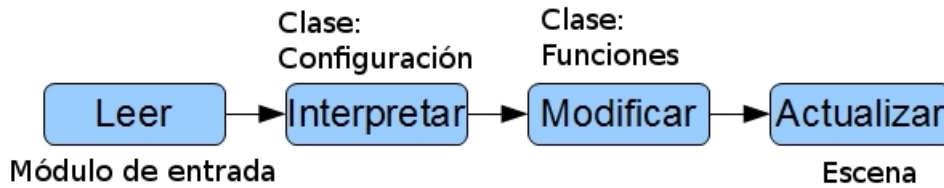


Figura 5.5: Pasos para actualizar una clase

Por ese motivo, de forma abstracta, toda clase que vaya a utilizar estos datos va a necesitar contener al menos las siguientes partes:

- I - Leer: Se necesita un bloque que lea los datos de los periféricos.
- II - Interpretar: los datos mediante la configuración activa.
- III - Modificar: Llamar a las funciones de la clase, con los parámetros adecuados o a las macros para actualizar el estado de los parámetros de la clase, y actualizar ese estado.
- IV - Actualizarlo en la escena, para que los cambios se vean reflejados en ella de cara al usuario.

Los pasos I y IV se encargan las clases de los módulos o clases correspondientes. Pero los pasos II y III deben ser implementados en cada clase que vaya a usar esos valores.

Se van a definir 2 sistemas de control con características distintas:

- Para el teclado, que se usará en cualquier entorno (Menú, escena ...)
- Para el resto de los periféricos (Solo válido en la escena)

5.10.2. Administración de mapas de caracteres:

Desde disponer de:

- Varios mapas de teclas para cada clase que los pueda usar.
- Sólo se debe usar un mapa cada vez, la mejor estructura para su administración es una pila de datos.
- Los datos de un mapa de una clase determinada deben guardarse en un vector.
- La pila de datos, maneja sólo el mapa que está situado en su parte superior. De esta forma, se puede pasar el control de unas clases a otras, devolviendo el control al que se lo otorgó a una determinada clase.

En esta parte se supone que las teclas del teclado se consultan cada cierto tiempo, y en función de cuales estén pulsadas, vamos a crear un mapa que indique qué hacer con cada posibilidad.

5.10.3. Administración de mapas de configuración:

Para el resto de periféricos, hay que detectar y guardar toda la información útil que ofrecen, para lo que se va a necesitar:

- Unas variables que indiquen qué estado o función de configuración se está utilizando.
- Las funciones que van a ejecutar para traducir los datos a acciones.

En esta parte vamos a suponer que ya se tienen los datos leídos y almacenados, y que falta cambiar el estado del objeto para poder actualizarlo en consecuencia.

5.11. Diseño del un laberinto aleatorio:

Para analizar las diferencias entre los diferentes periféricos, resulta interesante tener algún tipo de pruebas que el usuario deba completar con los diferentes periféricos. Por tanto, se ha implementado un algoritmo que genera automáticamente laberintos en el mundo virtual, utilizando texturas en 2D. Su fin, que el usuario lo recorra de principio a fin para evaluar la experiencia.

- Se debe utilizar un algoritmo básico que cree una estructura de laberinto
- Debe permitir flexibilidad, cambiar el tamaño y la posición donde es creado
- Debe tener una entrada y una salida
- Debe crear una estructura visible en el mundo virtual

5.12. Diseño de grupos:

Puede interesar agrupar varios objetos para administrarlos de forma más sencilla. Al crear una agrupación de objetos, se les puede añadir además, una serie de parámetros y propiedades que nos permitan interactuar de una forma compleja con el grupo, algo que no se podría hacer con los objetos por separado, así que se diseñan los siguientes grupos:

- Player

5.12.1. Player:

- Este grupo debe contener los objetos Catapulta, Castillo y Rey.
- Debe tener una identificación única
- Debe ser creado e inicializado por la escena
- Debe ser almacenado en una pila en escena para poder ser usado cuando se desee
- Debe poder interactuar con la cámara de forma especial

5.13. Conclusiones

Gracias a todos estos recursos, se puede esperar el mínimo de problemas al implementar cada faceta del programa necesario para crear y manejar nuestro mundo virtual. La estructura es ordenada y clara para su estudio, modificación, ampliación y mejora. Además, con este sistema se expresan al máximo las posibilidades del programa.

Parte IV

Evaluación

Capítulo 6

Metodología de evaluación

6.1. Introducción

En este capítulo se plantean las bases para la evaluación de los periféricos. Es importante especificar cuáles son los atributos que se pretenden evaluar. Para llegar al objetivo principal, es decir, para averiguar qué periférico, o combinación, es mejor para interactuar con un mundo virtual, es preciso plantear bajo qué directrices vamos a sacar las conclusiones.

6.2. Planteamiento del problema

Para decidir, qué periférico es el mejor, necesitaremos concluir:

- ¿Cuál es el más potente? En base a las teclas y los detectores de movimiento, cuál puede realizar una gran variedad de acciones.
- ¿Cuál es el más intuitivo de usar? Si el periférico va a ser usado por un usuario sin conocimientos avanzados de informática, uno de los puntos fuertes es que pueda cogerlo, y usarlo, de la forma más fácil posible. Si es posible, evitar la necesidad de que consulte manuales o tenga que realizar tutoriales.
- ¿Cuál es el más cómodo? Cada periférico puede ser muy distinto de otro. Es necesario valorar lo cómodo que puede ser utilizar cada periférico.
- Posibles aplicaciones. Las posibilidades de uso de cada dispositivo, y sus propósitos.

6.3. Planteamiento del método de medida

Para decidir los distintos aspectos de los periféricos, se estima que:

- Potencia: Esta forma de medida es completamente objetiva. Se va a estimar su potencia en relación a la información que es capaz de enviar el dispositivo en base a unos parámetros establecidos por el examinador. Dependiendo de la potencia se decidirá qué

periféricos o combinación de ellos merece la pena estudiar y cuáles no. Obviamente, el código diseñado para cada configuración influirá en la potencia, ya que según se programe de una forma u otra, se aprovecharán más o menos las características que ofrece cada dispositivo. En otras palabras, un mismo periférico podrá ser más o menos potente según sus rutinas de lectura e interpretación.

- Intuitivo: Esta forma de medida depende únicamente de los usuarios. Un usuario comenzará a utilizar el periférico sin ningún tipo de instrucción. Podrá solicitar señas si lo desea, o experimentar unos minutos con cada control.

- Comodidad: Esta forma de medida pretende valorar la comodidad de usar un dispositivo u otro. Se busca crear un perfil general de la percepción subjetiva que tiene un sujeto al uso de un dispositivo. Se necesita evaluar los siguientes parámetros:
 - Portabilidad del dispositivo
 - Desgaste físico del usuario
 - Interactuabilidad
 - Necesidad de espacio
 - Necesidades extra (baterías, alimentación)
 - Precio
 - Durabilidad

- Para medir los parámetros intuitivo y comodidad, se le hará superar una prueba al usuario con cada periférico, para todos será la misma. El objetivo es recorrer un laberinto 3D de principio a fin sin chocar con las paredes. No existe límite de tiempo ni penalizaciones, se pretende que tenga unos objetivos fijos el usuario para que, concentrándose en hacerlo lo mejor posible, pueda contestar a varias preguntas de unos cuestionarios.

Este método será aplicado a todos los dispositivos con todas sus configuraciones. Dependiendo de la configuración preparada para cada dispositivo, será más o menos potente, así que nos ayudará a descartar a priori las que se consideran que no van a ser suficientemente buenas. Las que sean lo suficientemente potentes serán utilizadas en las pruebas con los sujetos de prueba.

Para mejorar la potencia y la experiencia de los usuarios, se pueden utilizar combinados los distintos periféricos. En la tabla 6.3 se puede ver las posibles configuraciones:

Dispositivos						
Combinación	Teclado	Ratón	Joystick	Alfombra	Wiimote	Kinect
Teclado	-	OK	X	X	X	OK
Ratón	OK	-	X	X	X	X
Joystick	X	X	-	OK	X	OK
Alfombra	X	X	OK	-	OK	OK
Wiimote	X	X	X	OK	-	OK
Kinect	OK	X	OK	OK	OK	-

6.4. Diseño del método de medida

Para llegar a las conclusiones de cada planteamiento, se va a diseñar los siguientes métodos.

Potente:

Prueba para el diseñador. Valoración de los parámetros que ofrece el periférico, y cómo de aprovechable es. Por ejemplo, un giro puede ser gradual (mayor o menor en función de lo que quiera el usuario) o absoluto (con velocidad constante aunque adaptable).

Obviamente, es mucho más preciso e intuitivo un giro gradual, ya que permite al usuario moverse con más precisión y más o menos velocidad en función a su deseo. Este proceso conlleva una cierta habilidad para controlarlo. Existe un compromiso entre la precisión y la facilidad de uso que no puede ser ignorado.

Se propone calcular la potencia con la relación entre cuantos requisitos cumple un periférico con una configuración determinada y los requisitos totales.

$$P_{dispositivo} = \frac{\text{Requisitos cumplidos}}{\text{Requisitos totales}}$$

Donde Requisitos cumplidos son los que cumple cada periférico

Donde Requisitos totales = Necesidades + Disposición

La disposición es la forma que tiene el usuario de acceder a dichos elementos.

Se deben tener en cuenta varios aspectos:

- Detección absoluto o gradual. Se penalizará con un coeficiente si un periférico no aporta información gradual y ésta es requerida para mayor precisión.
- Ratio de acciones. Se establecerá para todos los periféricos que cumplan con un método útil para realizar acciones.
- Coeficientes de eficiencia. Se utilizará para penalizar acciones absolutas cuando debieran ser graduales. Será igual a 0,3 si es importante que sea gradual, y 0,7 si no es demasiado importante.

Los requisitos totales que se van a pedir son los siguientes:

Necesidades:

- Movimiento en los ejes X, Y, Z. Valorado con 6 puntos, uno por sentido. Se multiplicará por un coeficiente de eficiencia.
- Giro sobre un eje y su perpendicular. Valorado con 4 puntos, uno por sentido de giro, y multiplicado por coeficiente de eficiencia.
- Utilizar 4 macros. Valorado con 4 puntos, uno por macro.

Disposición y grados de libertad:

- ¿Existe una posición cómoda de equilibrio? Valorado con 1 punto.
- ¿Están los botones o acciones de macros bien situadas? Valorado hasta 3 puntos si es del todo conforme.
- ¿Necesita objetos? Valorado hasta 3 puntos. El máximo de puntos si el usuario no tiene que agarrar ningún objeto.

De esta forma la fórmula queda:

$$P_{dispositivo} = \frac{\frac{x_1}{6} * C_{ef} + \frac{x_2}{4} * C_{ef} + \frac{x_3}{4} + \frac{x_4}{1} + \frac{x_5}{3} + \frac{x_6}{3}}{21}$$

Intuitivo y comodidad:

Prueba para los usuarios. Durante unos minutos, el usuario puede utilizar el periférico para tratar de realizar los movimientos básicos. Se comprueba cuantas acciones es capaz de realizar. Se realizan las preguntas planteadas en la encuesta intuitivo y encuesta comodidad. Ver material adjunto cuestionarios.odt, contiene las preguntas que se formularon a todos los usuarios que participaron en las pruebas.

6.5. Conclusiones

En esta parte del planteamiento del análisis de los periféricos, se han fijado las bases para la evaluación de los periféricos, de forma que se puedan deducir varios atributos importantes de los periféricos, y a raíz de ello, estimar cual es el mejor periférico para cada aplicación elegida.

Capítulo 7

Evaluación

7.1. Introducción

En este capítulo se describen los procesos seguidos para la evaluación de los periféricos y se realizan todos los cálculos y mediciones que han sido planteadas en el capítulo del planteamiento 6. Todos los datos de este capítulo son utilizadas para sacar las conclusiones en el capítulo 8.

7.2. Potencia

La tabla de potencias se puede ver en la figura 7.2

	[Movilidad	Cm	Giro	Cm	Botones]	[Equilibrio	Distrib	Aparatos]	Potencia:
Teclado	6	0,7	4	0,3	4	1	2	0	0,59
Ratón	0	0,7	4	1	2	1	3	0	0,48
Joystick	6	1	4	1	2	1	2	0	0,71
Alfombra	6	0,7	4	0,3	2	1	2	1	0,54
Wiimote	6	1	4	1	4	0	3	2	0,90
Kinect I	4	1	4	1	0	0	3	3	0,67
Kinect II	6	1	4	1	0	0	3	3	0,76
Tec + Rat	6	0,7	4	1	4	1	2	0	0,72
Alf + Wiimote	6	1	4	1	4	0	3	0	0,81
Alf + Kinect II	6	1	4	1	4	0	2	3	0,90
Ideal:	6	1	4	1	4	1	3	3	1,00

Figura 7.1: Potencia de los periféricos

Se marcan en verde los periféricos o combinaciones de ellos que se han considerado lo suficientemente potentes como para manejar correctamente el mundo virtual. Los que aparecen en rojo no cumplen esta característica así que no serán dados a probar a los usuarios. El

margen necesario se ha definido como 0,71, que es la mediana de los valores de potencia. Se toma este valor para separar, del conjunto de periféricos del que se dispone, los menos potentes. Un valor de potencia menor supondría trabas para implementar un sistema adecuado o, en el mejor de los casos, una potencia significativamente menor que la ofrecida por otros dispositivos.

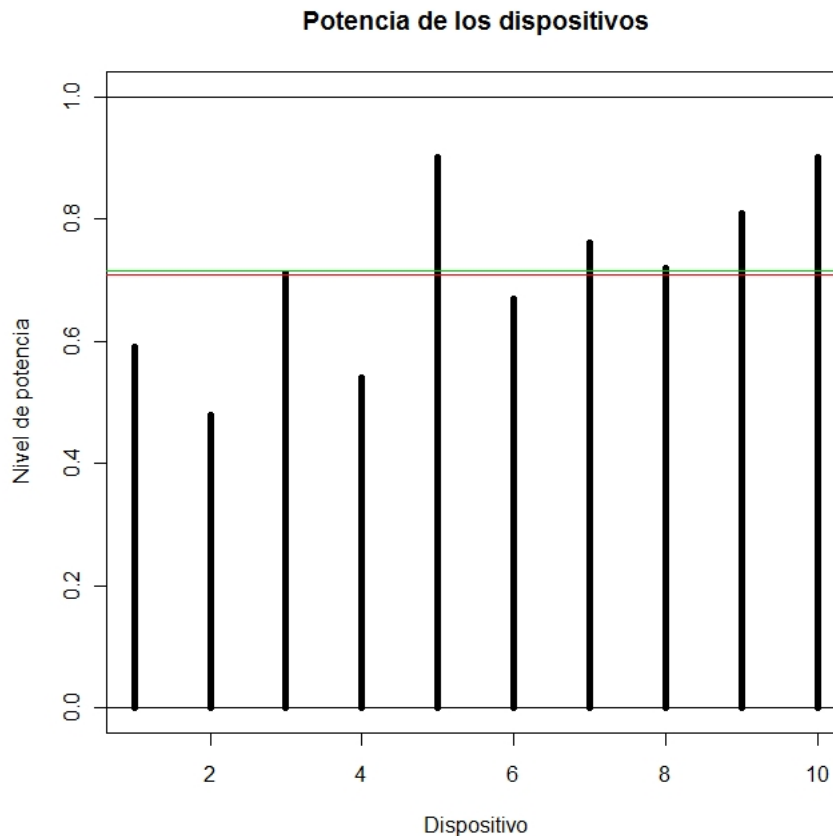


Figura 7.2: Potencia de los periféricos

En la figura 7.2 se puede apreciar los periféricos que quedan por encima de la media y la mediana, pintadas en rojo y verde respectivamente, y los que no.

7.3. Toma de datos de intuitivo y comodidad

Se ha realizado las encuestas para medir los otros 2 baremos, adjuntas en el anexo, a 22 usuarios. Se han dado distintos tratamientos a los datos obtenidos de las encuestas para puntuar los dispositivos, y sacar información útil para valorar aspectos de interés que influyen en la decisión del mejor periférico.

Intuitivo Los resultados de la toma de datos pueden verse en del material adjunto Encuestaintuitivo.ods, que es una hoja de cálculo con las respuestas de los 22 usuarios.

Cómodo Los resultados de la toma de datos pueden verse en del material adjunto Encuestacomodo.ods, que es una hoja de cálculo con las respuestas de los 22 usuarios.

Resultados globales Para tener una visión global de los resultados, y poder sacar unas conclusiones, se encuentra en el material adjunto el documento Resultadosglobalestadistica.ods, donde se han volcado todas las respuestas de los 22 usuarios, y existe una tabla resumen con las puntuaciones de cada dispositivo para cada categoría, y además el número de usuarios que hubiera retirado o votado como favorito cada dispositivo.

7.4. Conclusiones

En esta parte se han colocado todos los datos recabados en las encuestas a todos los usuarios en unas hojas de cálculo, para, de forma sencilla, sacar las conclusiones que se buscan para este proyecto.

En el siguiente capítulo se va a proceder al análisis de cada dispositivo.

Capítulo 8

Toma de datos y conclusiones

8.1. Introducción

Una vez hemos planteado y diseñado todos los experimentos necesarios para cumplir con el objetivo del proyecto, y la evaluación en los usuarios, se van a sacar varias conclusiones que se exponen a lo largo de este capítulo.

Para las pruebas, se entrevistó un total de 22 usuarios. Como se puede ver en 8.1, son los porcentajes de nota que sacaron para cada atributo de cada categoría. La categoría potencia, se basa sólo en los datos aportados en el archivo adjunto `calculodelapotenciadelosdispositivos.ods`. Para la categoría intuitivo, se ha empleado los datos normalizados de los atributos 1º predecible, 2º rápido, 3º fácil de usar. Y para la categoría cómodo, se han empleado los atributos 1º Preciso, 2º cómodo de utilizar, 3º asequible. Estos atributos se ven reflejados en las preguntas que se hicieron a los usuarios y que se pueden ver en el archivo adjunto `cuestionarios.odt`

Todos los datos, se encuentran disponibles en las tablas de datos adjuntos a la memoria. Conforme se utilicen, se irán citando. Para sacar las conclusiones, se recurre muchas veces a la tabla 8.1. Aunque para otras conclusiones, se citará en que dato se apoya.

Resumen datos

Dispositivo	Atributo	Media	Varianza
Teclado + Ratón	Predecible	36,36	33,77
	Rápido	33,07	60,67
	Fácil	18,64	7,58
	Preciso	28,86	164,12
	Cómodo	33,07	140,44
	Asequible	8,86	8,77

Dispositivo	Atributo	Media	Varianza
Wiimote	Predecible	31,36	145,67
	Rápido	26,25	205,8
	Fácil	16,14	33,17
	Preciso	27,95	130,14
	Cómodo	38,41	249,73
	Asequible	5,68	9,04

Dispositivo	Atributo	Media	Varianza
Wiimote+Dancepad	Predecible	33,18	108,44
	Rápido	33,98	77,18
	Fácil	18,18	15,58
	Preciso	33,41	167,59
	Cómodo	23,64	159,36
	Asequible	6,7	8,56

Dispositivo	Atributo	Media	Varianza
Gamepad	Predecible	37,27	49,35
	Rápido	21,93	159,48
	Fácil	13,64	38,53
	Preciso	26,14	149,84
	Cómodo	39,32	198,92
	Asequible	8,52	9,32

Dispositivo	Atributo	Media	Varianza
Kinect II	Predecible	8,64	136,15
	Rápido	10,45	118,24
	Fácil	6,82	53,68
	Preciso	19,09	122,94
	Cómodo	23,86	145,67
	Asequible	2,39	5,05

Figura 8.1: Resumen datos de los periféricos I, con las categorías usadas

8.2. Conclusiones de la potencia

Como ya se comentó en el capítulo 7, el sistema para medir la potencia que se ha definido

Resumen					SOBRE
Dispositivo	Potencia	Intuitivo	Cómodo	Promedio	
Teclado + Rat	71,00	88,07	70,80	76,62	100
Gamepad	90,00	72,84	73,98	78,94	100
Wiiote	76,00	73,75	72,05	73,93	100
Kinect II	81,00	25,91	45,34	50,75	100
Wiiote+Dan	90,00	85,34	63,75	79,7	100

	X > 0,7	X > 2/3	X > 2/3	X > 2/3
	-	1/3 < X < 2/3	1/3 < X < 2/3	1/3 < X < 2/3
	X < 0,7	X < 1/3	X < 1/3	X < 1/3

	Teclado y rató	Gamepad	Wiiote	Kinect	Wii + Dance
Votos favorito	4	1	2		15
Votos retirar:				15	1

Figura 8.2: Resumen datos de los periféricos II, con las valoraciones finales

descarta algunos de los periféricos. Resulta obvio que para los objetivos planteados, un ratón, por ejemplo, no es suficientemente potente para realizar todas las acciones planteadas. Pero había que valorar este punto de alguna forma. Algunos de sus puntos pueden parecer poco claros o difusos, según la percepción de la persona que puntúe los periféricos. Aunque cambien algo las cifras, en base a la valoración de la persona que los evalúe, unas décimas arriba o abajo, las conclusiones que se extraen no cambian si se siguen las directrices proporcionadas. Un ratón sólo no va a poder adaptarse al manejo de un mundo virtual por sí sólo. La combinación de un ratón y un teclado, (combinación muy habitual para muchos entornos) va a ser más adecuada, aunque con ciertas carencias. Y un Wiiremote va a permitir un grado mayor de funcionalidad.

Los datos numéricos de potencia se encuentran en los archivos del material adjunto `calculodelapotenciadelosdispositivos.ods`, y resumidos en el archivo `Resultadosglobalesestadistica.ods`. Se van a considerar, independientemente de los datos numéricos que:

- A menor potencia, más simple va a ser el manejo del mundo virtual, es decir, menos cosas debe tener en cuenta el usuario al manejar el mundo virtual.
- Mayor potencia implica normalmente más sensibilidad para el usuario, y por tanto, más concentración para realizar las acciones que desea hacer.
- Se debe tener en cuenta el compromiso que tiene la sensibilidad y la simplicidad.

De los datos, se puede deducir, que en términos de potencia:

- El teclado y el ratón pueden ser insuficientes por separado, juntos cumplen con las exigencias de potencia. Esto es porque su nota es inferior a la mediana del resto de potencias, por ello los descartamos. Al hacer una prueba usando los 2 juntos, algo habitual en usuarios de ordenador, superaría el nivel exigido. Es suficiente para un control básico del mundo virtual.
- El joystick cumple el objetivo de potencia algo justo. Tiene una nota de 0,71 sobre 10. Es suficiente para pasar las exigencias, pero va a ser difícil sacar más provecho del que se ha sacado de él hasta ahora, como muestra la nota obtenida. Permite, al igual que la configuración de ratón y teclado, un control aceptable del mundo virtual.
- El wiimote por sí solo destaca sobre los demás. Tiene una nota de 0,9 sobre 10, gracias en parte, a la información precisa que envía su acelerómetro. Este periférico permite un control básico del mundo virtual, desde el punto de vista de la potencia.
- El kinect es poco potente, a pesar de las posibilidades que ofrece. Tiene una nota de 0,76 sobre 10 para la configuración que supera el nivel mínimo exigido. Esto quiere decir que se debe investigar más de lo que se ha hecho en este proyecto con sus configuraciones para intentar mejorar este aspecto en términos de potencia.
- La alfombra, por sí sola es muy poco potente, pero su diseño permite aprovechar las piernas del usuario, que es un aspecto normalmente no aprovechado. Es un complemento que puede mejorar la potencia de otros, como es el caso del kinect. Si se observa la nota de la potencia de la alfombra combinada con otros periféricos, puede observarse que se alcanza una muy buena valoración.
- Curiosamente, la combinación de alfombra y wiimote baja la potencia. Esto es debido a que quita movilidad al usuario. Se pierde potencia, pero podría ganar en los aspectos de intuitivo o cómodo, o simplemente puede ser un buen complemento para buscar configuraciones más atractivas para el usuario. Por ello, merece que se preste atención a este aspecto en los análisis futuros.

8.3. Conclusiones de lo intuitivos que son los periféricos

De los resultados que se adjuntan en la hoja de cálculo `Encuestaintuitivo.ods` y `Resultadosglobalesestadistica.ods` podemos deducir las siguientes conclusiones:

Teclado y ratón

- El teclado puede considerarse intuitivo. Con una nota de 88,07 sobre 100.
- Es el mejor valorado en este aspecto. Esto es debido al gran uso que se hace de él. Todos los participantes del estudio lo usan a diario, por lo que pueden utilizarlo sin ningún tipo de indicación.

- Muchos de los usuarios valoran que se les explique detalladamente las funciones de las teclas, aunque consideran que lo habrían descubierto probando.

Gamepad

- El gamepad puede considerarse intuitivo. Tiene una nota de 72,84. A la vista de las preguntas realizadas, se les hace menos intuitivo al perfil medio de usuario encuestado su manejo.
- Es lo suficientemente predecible y rápido de aprender como para que la mayoría de los usuarios aprendan a usarlo en un periodo de tiempo pequeño.

Wiimote

- El wiimote puede considerarse intuitivo. Su nota es de 73,75 sobre 100. Es más novedoso que el teclado y el gamepad, eso hace que cueste un poco más de tiempo asociar la forma de uso a las acciones que se quieren realizar.
- Es menos intuitivo que el teclado, pero los usuarios, a su gran mayoría, lo han asimilado rápidamente.
- Es predecible tras unos minutos de uso, se controla con mucha facilidad independientemente de que el usuario no esté acostumbrado a utilizarlo.
- Otras formas de manejar la cámara hacen de la experiencia muy atractiva, como el modo avión.
- Añadir los periféricos que dispone añadiría precisión y funcionalidades, con lo que se conseguirían mejores resultados.

Kinect II

- Dispositivo muy poco intuitivo. Se ve claramente ya que la nota que obtiene es de 25,91 sobre 100. La configuración realizada es desechable.
- Requiere muchas explicaciones y detalles. Requeriría obligatoriamente un tutorial y unos ejercicios para practicar su manejo.
- Pocos usuarios terminaron controlándolo bien. Algunos de ellos desistieron de usarlos por su forma de manejo.
- No se maneja de forma realista, como se puede ver en las respuestas a la pregunta 5.
- Se debe orientar de otra forma para poder ser aprovechado.

Dancepad + wiimote

- Dispositivo muy intuitivo. Su nota sobre 100 es de 85,34
- Sorprendentemente es mucho más valorado que el wiimote solo. La razón es que el usuario se siente más inmerso en el mundo virtual, es más partícipe.
- Aunque necesita un poco más de movimiento, los usuarios lo asocian con diversión y entretenimiento.
- En muchos casos, el usuario necesitaba pensar menos en las acciones que quería realizar, o al menos, ésa era su percepción.
- Es interesante ver cómo combinar periféricos de forma dinámica, acciones muy sencillas, con otras menos sencillas pero más precisas, mejoran notablemente el partido que saca el usuario del dispositivo.

Comparando todos los sistemas, se puede inferir que resulta difícil separar al usuario de una forma muy extendida del manejo, como es el teclado y ratón. 4 usuarios prefieren la precisión que les da este a otras características. Serían más de no tener que elegir entre este y el wiimote combinado con el dancepad. Por otro lado, 15 consideran mucho más divertido y útil el manejo combinado del wiimote con el dancepad. Por tanto, no se debe prescindir de un buen manejo con teclado y ratón, dentro de lo posible. Es importante añadir funciones de wiimote y dancepad para, en caso de que sea posible, ofrecer al usuario esta forma de manejo.

8.4. Conclusiones de lo cómodos que son los periféricos

De lo resultados adjuntados en la hoja de cálculo `Encuestacomodo.ods` y `Resultadosglobalesestadistica.ods` podemos deducir las siguientes conclusiones:

Teclado y ratón

- El teclado y ratón es cómodo de usar. Se ve reflejado en la nota obtenida de 70,8 sobre 100.
- Se utiliza sentado, así que no supone cansancio de ningún tipo a largo plazo.
- La única pega que podría sacarse, es que si el usuario adopta malas posiciones eso le ocasiona problemas. Pero no suele ser el caso ya que no se utiliza para los objetivos propuestos por periodos muy amplios de tiempo sin descansar.

Gamepad

- Es muy cómodo de usar. Su nota es de 73,98 sobre 100.
- Se puede utilizar de pie y sentado.
- Es el mejor valorado, ya que se asocia a una mayor libertad. Se puede colocar en cualquier posición y de cualquier manera sin que afecte a la experiencia.
- La posición de equilibrio es pasiva, es decir, el usuario no tiene porque estar en una posición concreta, lo cual hace que sea más cómodo y menos cansado.

Wiimote

- El dispositivo es cómodo. Con una nota muy parecida al teclado y raton, o al gamepad, que es de 72,05 sobre 100.
- Para la mayoría de usuarios, el uso de pilas y bluetooth no supone una molestia importante.
- La mayoría prefiere usarlo de pie para un mejor control, y esto no supone una molestia. En muchos casos se valora positivamente.
- Añadir los periféricos de los que dispone no supondrían perjudiciales respecto a comodidad.

Kinect II

- El dispositivo no resulta cómodo, por lo que se espera, que aunque el usuario llegue a conseguir dominarlo con la suficiente destreza, no va a ser atraído para un uso continuado con la configuración actual. Su nota es de 45,34 sobre 100.
- Se puede calificar de incómodo e inútil para manejar la cámara de pruebas.
- Por contra, es muy útil para captar movimientos, por lo que sirve para captar posiciones en un avatar virtual.

Dancepad + wiimote

- En términos globales es algo menos cómodo. Este resultado es debido a que el usuario percibe, al ver lo que ocupa la alfombrilla, que ha de disponer de un espacio extra. Un espacio que es más o menos el mismo que el del wiimote y el kinect. Tiene una nota de 63,75 sobre 100.
- De todas formas el nivel de comodidad que percibe el usuario es muy bueno.

Además, los resultados de comodidad, entre todos los dispositivos salvo el Kinect II, son parecidos. Esto indica que los usuarios no prefieren estar sentados o de pie, así que no va a ser un factor determinante. Sí que puede resultar positivo, dispositivos que consigan que el usuario esté más activo, o más entretenido. De esta forma, se consigue una mejor experiencia y una predisposición por parte del usuario a utilizar el mundo virtual. Se necesita ofrecer un soporte básico, como es el teclado y ratón, al menos mínimo, para los casos en los que no se pueda utilizar ningún otro dispositivo por la razón que sea.

8.5. Conclusiones

Hemos visto en las distintas secciones del capítulo las valoraciones que se pueden concluir de cada baremo diseñado. No obstante falta de resolver las siguientes cuestiones. ¿Cuál es el mejor? ¿Cuál es el más útil? ¿Y el más versátil? ¿Cuáles merece la pena desarrollar? y ¿Cuáles son las posibilidades que ofrecen?

Parte V

Conclusiones

Capítulo 9

Conclusiones del proyecto

9.1. Tabla de puntuaciones

Como se puede ver en la figura 9.1, éstas son las puntuaciones de cada periférico o combinación de ellos según los baremos utilizados.

Resumen					SOBRE
Dispositivo	Potencia	Intuitivo	Cómodo	Promedio	
Teclado + Ratón	71,00	88,07	70,80	76,62	100
Gamepad	90,00	72,84	73,98	78,94	100
Wii mote	76,00	73,75	72,05	73,93	100
Kinect II	81,00	25,91	45,34	50,75	100
Wii mote + Dance	90,00	85,34	63,75	79,7	100

	$X > 0,7$	$X > 2/3$	$X > 2/3$	$X > 2/3$
	-	$1/3 < X < 2/3$	$1/3 < X < 2/3$	$1/3 < X < 2/3$
	$X < 0,7$	$X < 1/3$	$X < 1/3$	$X < 1/3$

	Teclado y ratón	Gamepad	Wii mote	Kinect	Wii + Dance
Votos favorito:	4	1	2		15
Votos retirar:				15	1

Figura 9.1: Resumen datos de los periféricos

9.2. Cuestiones

¿Cuál es el mejor?

No se puede hablar de un dispositivo mejor a otro. La gran cantidad de variables que entran en el proceso impiden una valoración absoluta. Se podría hablar de maximales, para cierto objetivo, un periférico puede considerarse mejor que otro. Si nos ceñimos a los baremos definidos, se puede considerar que el mejor es el wiimote, con el complemento del dancepad. Obtiene la mejor puntuación en términos globales gracias a su balance entre potencia, comodidad, precisión, diversión, facilidad y precio. Sólo el wiimote también resultaría interesante. Añadir los periféricos que mejoran su precisión y datos sería muy recomendable, ya que mejoraría las prestaciones.

Este es el caso, por ejemplo, del kinect. No hay un aparato que sostener, no tiene unos botones que pulsar, esto hace complicado crear una configuración para que haga una acción u otra. Se ve reflejado en los datos cómo es el peor valorado. El procesado de datos que realizan sus drivers permiten captar las posiciones del cuerpo del usuario y de sus orientaciones. Puede resultar muy útil para animar avatares en el mundo virtual de forma que no se puede conseguir en tiempo real con un teclado o un wiimote. Sería interesante buscar la forma de combinar la información que ofrece con la de otros dispositivos.

Por otro lado, el wiimote da los mejores resultados en cuanto a potencia, comodidad y además es intuitivo. Se puede plantear su uso para el manejo de mundos virtuales al reunir estos requisitos, ser barato, fácil de usar, y muy factible para que lo adquieran los usuarios para uso doméstico.

¿Cuál es el más útil?

A la vista de los resultados, en un modo más tradicional, el wiimote o el gamepad pueden cumplir esta premisa. El gamepad sería más sencillo y cómodo que usar, pero menos preciso que el wiimote. Dependerá de a cual de estas 2 posibilidades se quiera dar preferencia.

¿Cuál es el más versátil?

Esta pregunta es bastante difícil de responder, ya que no se ha conseguido explotar todas las posibilidades de los nuevos dispositivos. Con el wiimote se pueden crear muchas configuraciones dependiendo de qué objeto se quiera manejar o interactuar en el mundo virtual. No se ha implementado en este proyecto porque se alejaba un poco de los objetivos principales, pero con una serie de añadidos, se puede mejorar la precisión del wiimote y la cantidad de ejes de giro detectados, con lo que se podría desarrollar métodos más atractivos. Por otra parte, si se desarrollan librerías adecuadas, el kinect ofrece muchas nuevas posibilidades a explotar, con movimientos de menús mediante gestos, animación de avatares, captura de movimientos, información complementaria de las acciones que tome el usuario . . . Son puntos a favor de estos 2 dispositivos, que les dan una increíble ventaja frente a los demás periféricos.

¿Cuáles son las posibilidades que, en conjunto, ofrecen?

Puede resultar muy interesante combinar el reconocimiento de esqueletos del kinect, con la potencia del wiimote, lo que permitiría implementar un avatar que responde al 100 % con los los movimientos del usuario, con una movilidad muy precisa. Aunque los brazos estarían condicionados con el manejo del mando. Otra posibilidad puede ser el estimar la posición de la cabeza, detectar empujones o saltos usando el kinect, y responder a la navegación con el wiimote. También es posible detectar el tronco superior del usuario con el kinect, de forma que se refleje en el mundo virtual sus acciones, y controlar el movimiento con la alfombra. Existe un gran abanico de posibilidades, cada una con sus ventajas y desventajas.

Visión personal

A raíz de las conclusiones del proyecto, se puede ver que los nuevos periféricos que han salido estos últimos años son más sofisticados y están llenos de posibilidades. Existe todavía la frontera para adaptarse a ellos. Como usuario de ordenadores, acostumbrado al uso de teclado y ratón, éstos me dan una precisión y velocidad de acción muy superior a la conseguida con el gamepad, el wiiremote, o el kinect. ¿Significa esto que son mejores? La respuesta es no. Simplemente la costumbre y la experiencia son las bazas que inclinan la balanza a su favor. Si se profundiza más en configuraciones adaptables a cada usuario, con más posibilidades, más precisas, y sobre todo que permitan realizar acciones en el mundo virtual a la misma velocidad que se consigue con ratón y teclado, la cosa podría cambiar.

Viendo las nuevas posibilidades que ofrecen, los nuevos dispositivos aun están algo verdes. Cada día salen nuevas aplicaciones y hacks para explotar sus posibilidades. El tiempo dirá si van a superar a los periféricos tradicionales, o simplemente van a ser una opción más que elegir a la hora de manejar ordenadores y mundos virtuales.

Por tanto, si se dedica más tiempo en realizar más configuraciones adaptables, con puntos de vista diferentes, se podría dar con la manera de conseguir periféricos más realistas y más inmersivos para lograr una experiencia en un mundo virtual más profunda y precisa.

Otro aspecto importante que no me ha dado tiempo a investigar a fondo es la relación entre precisión y facilidad de uso. Aunque este aspecto queda más o menos cubierto con las encuestas realizadas a los usuarios y las conclusiones obtenidas de ellas, queda patente que unas configuraciones más pulidas pueden dar mejores resultados de las usadas en este proyecto.

Creo que, merece mucho la pena investigar cómo con acelerómetros se puede interactuar con objetos en un mundo virtual. Además, la captura de movimientos, relativamente eficaz y barata que ofrece el kinect, merece ser investigada más a fondo. Desarrollar software que procese las señales digitales que capta la cámara de infrarrojos y la cámara RGB, que reduzca el error y las exigencias del dispositivo puede resultar muy provechoso.

A parte de los periféricos evaluados, puede resultar interesante incorporar otros como las gafas inmersivas para ver y el kinect para manejar el avatar e interactuar con los objetos, etc.

Conclusión final

Con este proyecto, se puede concluir que aun está por explotar todas las posibilidades de los periféricos. Por separado, han supuesto novedades en sus respectivos campos. Al ser todos de distintos fabricantes, hay ciertas complicaciones para utilizarlos juntos, pero el desarrollo de librerías independientes, hechas por terceros, y hechas por los propios fabricantes, facilitan esta tarea. Con las pruebas realizadas, se ha dado un primer paso en la combinación de estos periféricos, y deja ver que las posibilidades que ofrecen juntos son inmensas. Merece la pena investigar combinaciones y calibraciones de varios de ellos, como se concluye en los apartados anteriores, para conseguir un manejo más preciso de los mundos virtuales, y una sensación de inmersión que no se ha podido conseguir hasta ahora.

Este proyecto ha sido enfocado como un primer paso en el uso de estos periféricos fuera del ámbito para el que fueron diseñados. A lo largo del proyecto, solo se ha manejado una cámara que puede moverse con hasta 6 grados de libertad en un espacio tridimensional. Queda pendiente un estudio más profundo para su manejo, y la posibilidad de utilizar varios dispositivos para el manejo de avatares en 3D, como primera persona.

Apéndice I

Configuración librerías SDL

¿Dónde conseguirlas?

Basta con acceder a la sección de descargas de la página oficial:

<http://www.libsdl.org/download-1.2.php>

Y las complementarias:

http://www.libsdl.org/projects/SDL_image/

http://www.libsdl.org/projects/SDL_ttf/

http://www.libsdl.org/projects/SDL_mixer/

¿Cuales se han usado en el proyecto?

Se han utilizado las librerías de desarrollo para VS2008.

- SDL-devel-1.2.14-VC8.zip
- SDL_mixer-devel-1.2.11-VC.zip
- SDL_ttf-devel-2.0.10-VC.zip
- SDL_image-devel-1.2.10-VC.zip

¿Cómo las utilizo con los archivos del proyecto?

Existen 2 formas, agregarlas a las librerías de OSG al compilar, o adjuntarlas a las librerías del proyecto directamente. Se han adjuntado directamente, como se ve en la parte Configuración de VS2008.

Para poder usarlas con el código fuente, falta un ultimo paso. Para que un programa las detecte, será necesario declararlas en la cabecera usando el código:

```
#include <SDL.h>
```

De este modo, los ficheros reconocerán todas las funciones, tipos, y enumerados declarados en las librerías.

Configuración librerías OSG

¿Dónde conseguirlas?

Basta con acceder a la sección de descargas de la página oficial:

<http://www.openscenegraph.org/projects/osg/wiki/Downloads>

Y las complementarias:

<http://www.openscenegraph.org/projects/osg/wiki/Downloads/Dependencies>

¿Cuales se han usado en el proyecto?

Se han utilizado las fuentes de la librería directamente. Como no se puede usar la fuente, antes se debe compilar

- OpenSceneGraph-2.8.3.zip
- 3rdParty_VC9sp1_x86_x64_V3.7z ; Utilizando el formato x86

¿Cómo las utilizo con los archivos del proyecto?

En este caso, debemos compilar las librerías primero. Osg permite el uso del programa Cmake, para compilar todo el proyecto. Los pasos a seguir son los siguientes:

- Abrir Cmake.
- Indicar la localización del código fuente
(Ruta de la carpeta descomprimida, Ej:C:/Programas/OpenSceneGraph-2.8.3)
- Indicar lugar donde se guardarán todas las librerías estáticas y/o dinámicas compiladas

(Ej: C:/Programas/OpenSceneGraph-2.8.3/BUILD)

- Apretar el botón `configure`. La primera vez detecta todas las cosas necesarias que se encuentran para la compilación. Si no aparecen las librerías de terceros bien detectadas (3rdParty) Indicar a mano, y volver a apretar `configure`
- Apretar `Generate`, elegir Visual Studio 9 2008, que es el que usa el proyecto.
- En la ruta elegida para la build, aparecerá un proyecto con todas las librerías.
- Compilar con Visual Studio todo el proyecto (ALL BUILD). Dependiendo de las opciones esto puede llevar mucho rato. Se recomienda ir a Build >Batch build, y elegir la opción build Debug y Release. Las otras a veces dan fallos. Guardar todo en la carpeta deseada, en mi caso: (C:/Programas/OpenSceneGraph-2.8.3/OpenSceneGraph-Data)
- El proceso puede dar fallos, pero si aparecen las librerías deberían funcionar.
- Para que Windows use esas librerías siempre, ir a propiedades de Mi pc >Propiedades avanzadas del sistema >Avanzado >Variables de entorno y añadir:

```
Variable_____Value
OSG_FILE_PATH____C:\Programas\OpenSceneGraph-2.8.3
                  \OpenSceneGraph-Data
OSG_PATH_____C:\Programas\OpenSceneGraph\bin
OSGDIR_____C:\Programas\OpenSceneGraph-2.8.3
Con esto el sistema debería localizarlas siempre.
```

Para poder usarlas con el código fuente, falta un último paso. Para que un programa las detecte, será necesario declararlas en la cabecera usando el código correspondiente:

```
#include <osg/Camera> (1)
#include <osg/MatrixTransform> (2)
```

Esto son solo 2 ejemplos de librerías, si se emplean las del módulo `osg`, la clase de la Camera, la primera línea las incluye, si deseamos usar matrices de transformación, lo hace la segunda. De este modo, los ficheros reconocerán todas las funciones, tipos, y enumerados declarados en las librerías.

Información adicional en estos tutoriales:

- <https://wiki.vicomtech.es/index.php/OpenSceneGraph>

Configuración Visual Studio 9 2008

¿Dónde conseguir el programa?

Basta con acceder a la sección de descargas de la página oficial:

<http://www.microsoft.com/spanish/msdn/latam/visualstudio2008/prueba.aspx>

Si se quiere utilizar más tiempo, hay que pagar la licencia. Este programa no es vital, se pueden utilizar otros compiladores de C++ gratuitos y/o libres.

¿Como detecta mi proyecto de VS2008 las librerías?

Para utilizar las librerías, el ordenador debe tener constancia de que se necesitan y se van a utilizar.

- Si son dinámicas (.dll) pueden colocarse en la ruta del ejecutable o el código fuente, y este las detectará automáticamente.
- Si se declaran adecuadamente las variables de entorno, el sistema operativo lo haría automáticamente.
- Existe una tercera opción, cambiar las propiedades del proyecto.

¿Cómo configuro el proyecto para usar el código fuente?

Si ya tienes las librerías listas para usarse, solo tienes que seguir los siguientes pasos:

Para la configuración Debug y Release, en las 2:
Configuration properties/C-C++/General/AID:
C:\Programas\OpenSceneGraph\include

```
C:\Programas\SDL-1.2.14\include
CConfiguration properties/Code Generator/ Multi-threaded DLL
L/G/ALD:
C:\Programas\SDL-1.2.14\lib
C:\Programas\OpenSceneGraph\lib
Linker/General/Additional Library directories:
SDL.lib
SDLmain.lib
SDL_mixer.lib
SDL_image.lib
SDL_ttf.lib
OpenThreads.lib
osg.lib
osgAnimation.lib
osgDB.lib
osgFX.lib
osgGA.lib
osgManipulator.lib
osgParticle.lib
osgShadow.lib
osgSim.lib
osgTerrain.lib
osgText.lib
osgUtil.lib
osgViewer.lib
osgVolume.lib
osgWidget.lib
```

Las de OSG en realidad no harían falta, ya que están definidas como variables de entorno, pero por prevenir.

Apéndice II

Uso de las librerías SDL

¿Cómo se usan?

Como son unas librerías de bajo nivel, hay muchas funciones que se pueden usar. Se pueden utilizar los tutoriales de la página oficial:

Tutoriales oficiales:

<http://wiki.libsdl.org/moin.cgi/Tutorials>

Cabe destacar:

http://lazyfoo.net/SDL_tutorials/index.php

Para el cometido del proyecto, suele bastar con consultar las funciones de la wiki del SDL. Aunque viene muy bien leer los tutoriales de Brian Malloy, que combinan además SDL y OSG.

Enlaces de interés:

Enumeración de las teclas:

<http://iie.fing.edu.uy/ense/assign/tap/obrar05>

[/santiagoackermann_comp/web/sdl/doc/html/html/sdlkey.html](http://santiagoackermann_comp/web/sdl/doc/html/html/sdlkey.html)

Ventanas SDL: <http://www.programaciongrafica.com/modules.php?name=News>

Uso de las librerías OSG

¿Cómo se usan?

Como son unas librerías de nivel medio, pero no es más difícil encontrar material adecuado.

Tutoriales oficiales:

<http://www.openscenegraph.org/projects/osg/wiki/Support>

Cabe destacar: (Se usan como base para la programación del proyecto)

<http://www.cs.clemson.edu/~malloy/courses/3dgames-2007/tutor/>

Además existen muchos libros sobre las librerías.

Libro sobre OpenGL: (No es necesario profundizar tanto)

Addison Wesley "OpenGL ES 2.0 Programming Guide Aug.2008"

Conviene saber:

Administración de memoria de OSG

Clase Referenced:

La clase referenced guarda un array de punteros, que sirve para administrar la memoria. Cuando el último es borrado, se quita de memoria el árbol. Debe activarse manualmente con la función `ref()`. Existe un método automático, `ref prt`.

Puntero `ref prt`:

Es una Template class, que garantiza la correcta administración de memoria, el manejo de los punteros de c++, y la actualización de la clase referenced para manejar más fácilmente los punteros de OSG.

En nuestro programa, el módulo que lo requiera, creará un puntero de la clase que necesite:

- Declaración:

```
osg::ref_ptr<ClasedeOSG> nombre
```


- Inicialización: nombre = new(ClasedeOSG())
- NOTA: Cada clase que vaya a usar un puntero, se debe declarar en la cabecera, pero no inicializar con new, sino llamar al puntero (variable privada normalmente) que la guarda.

Estructura del árbol de OSG

Elementos:

El árbol OSG se compone de varios elementos:

- Scenedata que hace de root. Tipo osg::Group. Contiene la serie de elementos que componen una escena.
- Objeto móvil: Tipo osg::PAT con hijo osgDB::RNF.
- Geometrías programadas: Tipo osg::Geode con hijo quad Tipo Osg::Geometry, con hijos State tipo osg::StateSet ; Image tipo OsgDB::RIF ; texture tipo Osg::Texture2D. Con diversas opciones para ser dibujada. inicializar con new, sino llamar al puntero (variable privada normalmente) que la guarda.
- Billboard: Textura en 2D con orientación del PAT hacia la cámara.

Tutoriales de programación en C++

¿Cuales usar?

Los tutoriales de C++ abundan en la web, pero recomiendo:

<http://c.conclase.net/curso/?cap=000#inicio>

Curso muy completo, muy bien explicado, con posibilidad de comentar, ejercicios, ejemplos ...

Demos en C++

¿Cuales han aportado algo?

Hay gente que libera ejemplos, demos, y código usando copyright, copyleft, GNU GPL ... para enseñar a programar, demostrar su capacidad para conseguir contratarlos ... Es importante respetar las peticiones de la licencia.

Cámara usando cuaterniones y C++:
<http://www.dhpoware.com/demos/glCamera3.html>

Curso muy completo, muy bien explicado, con posibilidad de comentar, ejercicios, ejemplos ...

Otros manuales de interés

Representación gráfica de funciones en 2D y 3D

Si se necesitan dibujar, en 2 dimensiones o en 3 dimensiones, una posibilidad es exportar los datos a un txt, darles el formato adecuado, y dibujarlo usando GNUPlot. Tiene copyright, pero está distribuido libremente, es muy usado, tiene gran variedad de tutoriales. Además, es compatible con latex.

GNUPlot página oficial:
<http://www.gnuplot.info/>

Manuales:

<http://www.scribd.com/doc/16593126/Manual-Gnuplot>

Apéndice I

Implementación

I.a. Introducción

En este capítulo se va a concretar como se ha implementado en el código cada parte diseñada en los capítulos anteriores. Además, se incluye una serie de pasos para poder realizar cambios con éxito en cada apartado. En este capítulo se responde a la pregunta ¿Cómo se ha resuelto a nivel técnico?

I.b. Herramientas y configuración del entorno de trabajo:

I.b.1. Herramientas:

Ya hemos explicado anteriormente que lenguaje se va a emplear y en que librerías nos apoyaremos. Es importante especificar las versiones de cada una de ellas.

- Microsoft Visual Studio 2008: Para la programación del proyecto en C++. Será este nuestro entorno de trabajo.
- SDL 1.2.14: Última versión estable al comienzo del proyecto.
- OSG 2.8.3: Última versión estable al comienzo del proyecto. Compilada con cmake por mi mismo.
- WiiYourSelf v1.15 by gl-tter
- Cmake 2.8.3: Para la compilación de las librerías de OSG

Además, se han utilizado los siguientes programas para la preparación y diseño del proyecto y la memoria:

- Openoffice 3.2.1: Redacción preliminar
- TexMaker: Entorno de trabajo para la redacción en latex

- MikTex: Compilador de Latex
- Gimp 2.4.1: Edición de imágenes

I.b.2. Configuración:

Para que nuestro sistema de módulos reconozca las librerías correctamente hay que realizar varios pasos.

- Obtener adecuadamente las librerías estáticas (.lib) o dinámicas (.dll), ya sea compilando la fuente nosotros mismos u obteniéndolas ya compiladas.
- Las librerías deben ser correctamente colocadas y declaradas para que puedan ser usadas.
- Se debe configurar las propiedades del proyecto.

Se adjunta más información de como se pueden realizar todas estas configuraciones en el Anexo X.

I.c. Implementación de los módulos:

En esta parte se especifican aspectos concretos necesarios para la implementación de cada módulo, así como una descripción de su función, su objetivo, sus funciones, momento de creación y llamada . . .

No se va a especificar todos los valores que toma, solo su descripción. Se omite la mención de toda la inicialización de los constructores, basta con consultar la zona correspondiente del código fuente adjunto en el Anexo ***.

I.c.1. Engine Core:

Localización del código:

Engine Core.

Qué función desempeña:

Se encarga de reservar la memoria para lanzar todo el programa. Además, cuando se termine de utilizar, se encarga de liberar la memoria.

Es creado por:

La función main, cuyo único propósito es seguir los estándares de programación de C++. Es decir, el programa principal debe llamarse "main.cpp".

Para: Iniciar el sistema.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

Al comienzo de la ejecución. Se le devuelve el control cuando se decide terminar.

Funcionamiento:

Reserva la memoria inicial, manda crear los módulos, cede el control al game módulo y no vuelve a hacer nada hasta el final de la ejecución.

Las clases que define son:

- Ninguna

Emplea las clases:

- Module registry.

Cabe destacar que:

Nada.

I.c.2. Module registry:

Localización del código:

Module Registry

Qué función desempeña:

Este módulo es auxiliar. Nos servirá para administrar y organizar el uso de los otros módulos principales del programa. Dispone de punteros al resto de módulos, por lo que incluyéndolo en cualquier función, podemos crear un puntero local al resto de módulos.

Es creado por:

El engine core.

Para:

Inicializar el resto de los módulos y tener punteros apuntando a cada uno de ellos. Esto servirá para poder referirnos a ellos y sus funciones desde cualquier clase si antes solicitamos un puntero a la instancia deseada.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

- El engine core ha reservado la memoria necesaria correctamente.
- Cada vez que alguna clase necesita un puntero.

Funcionamiento:

Su constructor llama crear automáticamente todos los módulos usando el método instance descrito en la parte *****

Las clases que define son:

- Ninguna

Emplea las clases:

- Ninguna

Cabe destacar que:

Es un módulo meramente organizativo, que brinda de forma sencilla una localización rápida y fiable de las zonas de memoria ocupadas por cada módulo.

I.c.3. Input module:**Localización del código:**

Input Module

Qué función desempeña:

Módulo encargado de administrar las entradas al programa. Define las clases intermedias para la gestión y visualización de los periféricos.

Es creado por:

La función Module Registry.

Para:

- Controlar las entradas en el menú.
- Si detecta una señal tipo QUIT, manda interrumpir la ejecución.
- Si es una señal tipo NORMAL, la interpreta y actualiza la estructura de datos. correspondiente.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

- Cada iteración, en la zona del código del update.
- Cada vez que se llama a la función handle input, para captar eventos.

Funcionamiento:

Hay zonas del programa en la que se ejecuta la función handle input. Al entrar en esta zona, se examinan todos los periféricos, se actualiza su estado y se guarda en las estructuras de datos creadas para este fin. Este proceso se repite muchas veces por segundo, así que, aunque es discreto, desde el punto de vista del usuario es continuo.

La ejecución del programa no se hace de forma constante, por tanto el muestreo de los datos de los periféricos no se hace con intervalos regulares. Este hecho no importa, ya que es un sobremuestreo. Aunque la información es guardada en las estructuras de datos, solo la utilizaremos cuando lo consideremos preciso, de forma controlada.

Se debe tener en cuenta, que cada periférico funciona de forma diferente al resto, por tanto, el tratamiento de los datos puede variar en cada caso.

Las clases que define son:

- Control Map
- Joystick
- Mouse
- WiiMote

Emplea las clases:

- Player
- HUD
- Scene
- Menu
- Module Registry
- wiimote (Librerías externas)

Cabe destacar que:

Al ser el módulo de inputs, se comunicará con toda clase con la que el usuario pueda interactuar. Además, albergará todas las clases de los periféricos para guardar los datos correspondientes a cada uno de ellos. Esto no significa que además, tenga que utilizar librerías adicionales para algunos periféricos.

I.c.4. Display Module:**Localización del código:**

Display Module

Qué función desempeña:

Módulo encargado del aspecto gráfico del motor y mostrar las salidas del sistema. Las salidas del sistema se podrán realizar a un nivel básico por consola, o mediante HUDs implementados sobre OSG.

Es creado por:

La función Module Registry.

Para:

Iniciar el aspecto gráfico del programa.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

- Se debe actualizar la pantalla.
- Al ser solicitada información para sacar por consola.

Funcionamiento:

La actualización de la pantalla se hace en cada iteración el programa en la zona de UPDATE. Se tiene en cuenta el tiempo que ha pasado entre cada iteración, para actualizar correctamente la física del motor gráfico. Al hacerlo, se guarda el tick preciso en el que se hizo, hasta la próxima llamada a actualizar.

Dispone de un sistema para ofrecer datos en tiempo real por la consola. Basta con entrar en el estado de menú auxiliar correspondiente.

Las clases que define son:

- Console

Emplea las clases:

- Ninguna

Cabe destacar que:

Utiliza excepciones para la reserva de la memoria de vídeo, ya que puede ser denegada por el sistema operativo. Puede cambiar todas las opciones de vídeo mediante las librerías SDL. La clase consola dispone de menus estáticos para ofrecer de la forma más simple información a bajo nivel.

Consola:

Menus consola:

Estructura 0:

Menú principal:

- 1- Mostrar estructuras de información de los input
- 2- Configurar los periféricos
- 3- Mostrar estructuras de información generales
- 4- Opciones generales
- 9- Volver

Estructura 1.0:

Menú elegir que estructura de periférico mostrar:

- 1- Mostrar Keyboard
- 2- Mostrar Mouse
- 3- Mostrar Joystick
- 4- Mostrar WiiMote
- 9- Volver

Estructura 2.0:

Menú elegir que estructura a configurar:

- 1- Configurar Keyboard
- 2- Configurar Mouse
- 3- Configurar Joystick
- 4- Configurar WiiMote
- 9- Volver

Estructura 3.0:

Menú elegir que estructura general mostrar:

- 1- Configurar Keyboard
- 2- Configurar Mouse
- 3- Configurar Joystick
- 4- Configurar WiiMote
- 9- Volver

Estructura 4.0:

Menú elegir que tipo de opción se quiere cambiar:

- 1- Configurar VIDEO
- 2-
- 3-
- 4-
- 9- Volver

I.c.5. Object Factory Object:

Localización del código:

Object Factory Module

Qué función desempeña:

Módulo encargado de crear todos los objetos de una escena. En él se declaran todas las clases de objetos que se van a usar. Se encarga de cargar los objetos al principio y eliminarlos cuando se usa el destructor. Usa la clase objeto, que es amiga para poder usar sus atributos privados. Los objetos son clases derivadas de movable, y todos se destruyen con destroy movable o sus derivados.

NOTA: Al destruir objetos en OSG, no los eliminamos directamente, simplemente cortamos el hijo de la escena, al no tener referencia posible, no se dibuja.

Es creado por:

Module Registry.

Para:

Encargarse de la gestión de todos los objetos de la escena.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

Es necesario iniciar, crear o destruir un objeto.

Funcionamiento:

es preciso

Las clases que define son:

- Ninguna

Emplea las clases:

- Object

Cabe destacar que:

Este módulo agrupa varias acciones importantes. No solo guarda una lista de texturas y objetos lógicos. Se encarga de cargar y descargar objetos del árbol del OSG, carga los objetos en memoria mediante excepciones, los inicializa, y crea estructuras extra como el suelo.

I.c.6. Audio Module:**Localización del código:**

Audio Module

Qué función desempeña:

Módulo encargado de cargar, descargar, y administrar los samples (Efectos sonoros) y la música del sistema.

Es creado por:

Module Registry.

Para:

Administrar audio.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

Necesitamos cargar algún sonido, conectar, desconectar, o administrar los parámetros de audio.

Funcionamiento:

Al tener encapsuladas todas las funciones relacionadas con el audio, solo es necesario tener acceso al objeto y emplearlas según convengan.

Las clases que define son:

- Ninguna

Emplea las clases:

- audio sample set

- music sample set

Cabe destacar que:

Simplemente encapsula todas las opciones disponibles de la librería SDL Mixer, especifica para esta función.

I.c.7. Timer Module:

Localización del código:

Timer Module

Qué función desempeña:

Módulo que controla el tiempo de cada ciclo y el tiempo cuando es llamado. Cualquier clase puede usarlo si tiene funciones que dependen de la variable independiente tiempo. Sirve para esperar cuando es necesario y sincronizar acciones si guarda variables de tiempo de eventos concretos.

Es creado por:

Module Registry.

Para:

- Obtener ticks del sistema mediante OSG.
- Guardar variables con tiempo.
- Limitar los FPS, si el ordenador es potente, deberá usar ciclos de memoria en otros procesos para que no funcione excesivamente deprisa el programa.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

- Es necesario un cálculo de tiempo.
- En la región update, para guardar tiempo de última actualización del motor.
- En la región update, para controlar los FPS.

Funcionamiento:

es preciso

Las clases que define son:

- Ninguna

Emplea las clases:

- Scene
- Menu

Cabe destacar que:

Es muy importante su empleo, ya que muchas clases que se emplean necesitan coordinación temporal para funcionar correctamente.

I.c.8. Game Module:**Localización del código:**

Game Module

Qué función desempeña:

Este módulo es el encargado de gestionar a nivel lógico el desarrollo de todo el programa. Es decir, sigue el hilo de la posición del programa en la que está el usuario, por ejemplo un menú concreto o una escena. Además, según los flags y estados activos, debe actuar consecuentemente.

Es creado por:

Module registry.

Para:

Administrar los estados y estructuras de datos del programa.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

El Engine Core::start Engine() le cede el control de la ejecución al llamar a sus funciones principales.

Funcionamiento:

Una vez iniciado y lanzado, se mueve por los diferentes estados de los menús y escenas. Los pasos que sigue son los siguientes:

- Comprobar las entradas, para ver las acciones del menú.
- Evaluar los estados del menú, para realizar una u otra acción.
- Actualizar los gráficos.

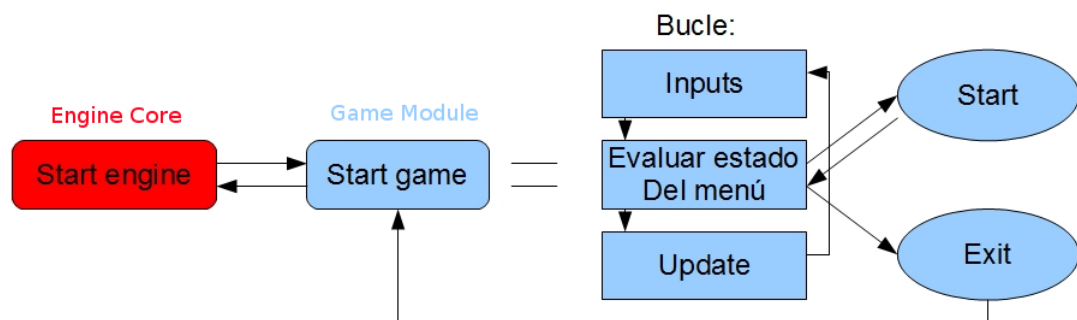


Figura I.c.8.

Los pasos que sigue cuando se entra en el estado start del menú son:

- Esconder el menú
- Crear e iniciar la escena
- Pasar el control a la función start level, que controla la escena
- Mostrar el menú, cuando se ha terminado

***** Zona que podría editarse, no dibujar aun:**

Primero, se suministran unas variables que guardan la dirección de los objetos que se van a necesitar usar. Además, se carga la escena, partiendo del estado de que está creada e inicializada la clase. El bucle que controla la escena realiza las siguientes acciones:

- Actualizar parámetros
- Comprobar entradas
- Actualizar SDL y OSG

Emplea las clases:

- Player
- Effects
- HUD
- Menu
- Scene

Cabe destacar que:

Es el encargado de que todo proceso del programa se ejecute en el momento correcto

I.d. Implementación de las clases:

En esta parte se especifican aspectos concretos necesarios para la implementación de las clases empleadas por los módulos para su correcto funcionamiento. Para una mejor localización y entendimiento de las clases, vamos a dividir las por tipos. Se han dividido las clases en varios tipos, para una mejor organización. ***** Adjuntar tabla datos.

I.d.1. Tipo Audio

Este grupo de clases son las empleadas para en la gestión del audio.

I.d.1.1. Audio sample set

Localización del código:

Audio Sample Set.cpp

Qué función desempeña:

Encapsula todas las funciones de sonido.

Es creado por:

Audio Module

Para:

Gestión del audio

I.d.1.2. Music sample set

Localización del código:

Music Sample Set.cpp

Qué función desempeña:

Encapsula todas las funciones de sonido relativas a música.

Es creado por:

Audio Module

Para:

Gestión de la música

I.d.1.3. Music sample

Localización del código:

Music Sample Set.cpp

Qué función desempeña:

Encapsula todas las funciones de sonido relativas a música.

Es creado por:

Audio Module

Para:

Da acceso al constructor y destructor.

I.d.2. Tipo Auxiliar

Este grupo de clases usadas por las librerías auxiliares. No van a ser aquí descritos.

I.d.3. Tipo Cámara

Este grupo de clases son las empleadas para el manejo de la cámara.

I.d.3.1. FreeCamera

Localización del código:

freecamera.cpp

Qué función desempeña:

Encapsula todas las funciones de la cámara 6DOF.

Es creado por:

Scene.cpp. Guarda un puntero a cada cámara creada en el vector privado freecameras.

Para:

- Manejar una cámara con sus propios parámetros.

Parámetros de entrada:

- ID, la identificación es única para cada cámara. Es un entero positivo. Se guardan en el vector posición por orden de IDs.

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se actualiza la cámara si esta en uso.

Funcionamiento:

Mediante los controles, podemos manejar esta cámara. Tiene varios modos de funcionamiento, para que se comporte de forma distinta ante los input. Además, puede cargar distintos tipos de movimientos para cada modo de funcionamiento. Dispone de unas funciones básicas para modificar sus parámetros (setters) y otros para leerlos (getters). Dispone de varias funciones básicas, y otras avanzadas, que funcionan como macros para que hagan funciones concretas con la cámara al ser activadas.

Controles:

Para controlar la cámara, se han programado los controles genéricos descritos en la sección de implementación común.

I.d.4. Diseño de los mapas de configuración:

Cada periférico posee varias configuraciones para el manejo de los objetos que lo permitan. Es interesante, antes de implementar, tener diseñado que debe hacer cada configuración.
***** ¿Vale la pena escribirlo todo?

*******Mapas del Teclado:**

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- A: Avanza de lado
- W: Avanza al frente

I.d.4.1. Mapas del Ratón:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- LMB
- RMB

I.d.4.2. Mapas del Joystick:

Configuración por Default. Con esta configuración se pretende una navegación sencilla:

- Axi0 arriba
- Axi0 abajo

Declara las clases:

- CrossEye. Una mira para el control de las iteraciones.

Cabe destacar que:

Los distintos modos de la cámara, permiten distintas perspectivas de manejo que pueden ser aprovechadas por los distintos periféricos. No todos los modos serán igual de intuitivos con unos periféricos que con otros.

I.d.5. Tipo Efectos

Este grupo de clases son las empleadas para añadir efectos externos a cualquier objeto 3D movable. Effect es una clase virtual, que permite la creación de todos los efectos que queramos, añadiéndoles funcionalidades.

Localización del código de la clase y derivados:

effects.cpp

Árbol de OSG:

Cada efecto, tiene acceso al puntero que apunta al nodo PAT del árbol OSG del objeto que quiere tratar, así que modifica su valor para aplicar el efecto.

I.d.5.1. Rotator

Qué función desempeña:

Esta clase derivada de efecto, permite modificar directamente el PAT de un objeto en el árbol de una escena de OSG.

Es aplicado por:

Objeto o movable. Esta clase permite añadir punteros a los efectos.

Para:

- Son utilizados cuando un objeto sufra el efecto rotación.

Parámetros de entrada:

- Dirección de memoria con el PAT de OSG a modificar
- Cantidad de rotación double en x
- Cantidad de rotación double en y
- Cantidad de rotación double en z

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se desee aplicar el efecto.

Funcionamiento:

Se le proporcionan los parámetros y lo hace automáticamente.

I.d.5.2. Gravity**Qué función desempeña:**

Esta clase derivada de efecto, permite modificar directamente el PAT de un objeto aplicándole una fuerza que haga que se decremente en el eje Y.

Es aplicado por:

Objeto o movable. Esta clase permite añadir punteros a los efectos.

Para:

- Son utilizados cuando un objeto sufra el efecto de la gravedad. No controla límites, se debe hacer de forma externa.

Parámetros de entrada:

- Dirección de memoria con el PAT de OSG a modificar
- Constante de gravedad.

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se desee aplicar el efecto.

Funcionamiento:

Se le proporcionan los parámetros y lo hace automáticamente.

I.d.5.3. Wind

Qué función desempeña:

Esta clase derivada de efecto, permite modificar directamente el PAT de un objeto aplicándole una fuerza que haga que se mueva en el plano X-Z (Horizontal) con una fuerza que dependa de la cantidad de viento.

Es aplicado por:

Objeto o movable. Esta clase permite añadir punteros a los efectos.

Para:

- Son utilizados cuando un objeto sufra el efecto del viento. No controla límites, se debe hacer de forma externa.

Parámetros de entrada:

- Dirección de memoria con el PAT de OSG a modificar
- Cantidad de movimiento double en x
- Cantidad de movimiento double en z
- Constante de la fuerza

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se desee aplicar el efecto.

Funcionamiento:

Se le proporcionan los parámetros y lo hace automáticamente.

I.d.5.4. Traslator

Qué función desempeña:

Esta clase derivada de efecto, permite modificar directamente el PAT de un objeto moviéndolo una cantidad determinada en X, Y y Z.

Es aplicado por:

Objeto o movable. Esta clase permite añadir punteros a los efectos.

Para:

- Son utilizados cuando un objeto sufra un movimiento. No controla límites, se debe hacer de forma externa.

Parámetros de entrada:

- Dirección de memoria con el PAT de OSG a modificar
- Cantidad de movimiento double en x
- Cantidad de movimiento double en y
- Cantidad de movimiento double en z

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se desee aplicar el efecto.

Funcionamiento:

Se le proporcionan los parámetros y lo hace automáticamente.

I.d.6. Tipo Escena

Este grupo de clases son las empleadas para gestionar la escena.

I.d.6.1. Scene

Localización del código:

scene.cpp

Qué función desempeña:

Administra los estados en los que puede estar la escena, algunos tipos de colisiones básicas y las variables que necesita la escena.

Es creado e inicializado por:

La función load level del game module.

Para:

- Que la escena llamada cree todos los elementos necesarios.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

Se necesita actualizar o editar cualquiera de los parámetros que maneja.

Funcionamiento:

El game module manda a la escena reservar memoria, construirse, y usar los parámetros definidos en su función init.

Árbol de OSG:

La clase scene maneja la escena desde el root. Usa la función de OSG `getscenedata`, para manipular los datos de la escena desde la raíz. Esto le permite la posibilidad de añadir cualquier componente de una escena.

Declara las clases:

- Ninguna.

Cabe destacar que:

Este clase contiene todos los elementos necesarios para crear la escena. Si requiere usar otras clases que encapsulen crear la escena, ésta lo hace simplemente llamándolos. Los parámetros más generales se declaran en ella al ser pocos.

I.d.7. Tipo Grupos

Este grupo de clases son las empleadas para agrupar varios tipos de clases distintos, para ser utilizadas como una sola.

I.d.7.1. Player

Localización del código:

player.cpp

Qué función desempeña:

En este grupo se consideran unidos los objetos catapulta, castillo y rey. Además, contiene un identificador y varias funciones para poder interactuar con la clase al seleccionarla.

Es creado por:

La función start game de escena. Se encarga también de asignarle los valores de inicio, las direcciones de los objetos, el billboard, y meterlo a la pila de jugadores de Scene.

Para:

- Tener una entidad jugador, y poder usar sus funciones.

Parámetros de entrada:

- Tamaño de la pila de jugadores
- Orientación en plano X-Z

Parámetros de salida:

- Ninguno

Se le llama cuando:

Es necesario utilizar alguna acción del jugador.

Funcionamiento:

Es una clase a la que se le puede transferir el control de la cámara, y puede rotar a los 4 lados a los que mira la cámara, sin avanzar.

Declara las clases:

- Ninguna.

I.d.8. Tipo HUD

Este grupo de clases son las empleadas para mostrar los HUDs por pantalla.

I.d.8.1. HUD**Localización del código:**

HUD.cpp

Qué función desempeña:

Es el núcleo que se encarga de gestionar las clases relacionadas para crear un HUD a partir de bloques individuales de HUDs. Cuando se destruye, elimina todos los elementos que le hayan sido añadidos.

Es creado por:

La escena, en su constructor.

Para:

- Tener un HUD completo.

Parámetros de entrada:

- X mínimo
- X máximo
- Y mínimo
- Y máximo

Parámetros de salida:

- Ninguno. Pero crea un HUD del tamaño que se le indique.

Se le llama cuando:

Se necesita añadir o quitar algún panel del HUD.

Funcionamiento:

Con las funciones add y remove hud element, se pueden añadir elementos con la clase hud element. Estos, son guardados automáticamente en una pila y borrados cuando se destruye la clase.

Árbol de OSG:

La clase HUD crea, y tiene un puntero al árbol de OSG de la matriz de proyección. Sus hijos, sufren el efecto de estar proyectados en la pantalla. Así que siempre que creamos un HUD element, se añadirá un hijo a esta rama del árbol.

No hace falta dibujarlos, ya que al estar en el árbol, OSG lo hará automáticamente. Si se quiere dejar de ver alguna parte, basta con usar la función remove hud element.

Declara las clases:

- Ninguna.

I.d.8.2. HUD element**Localización del código:**

HUD element.cpp

Qué función desempeña:

Añade al núcleo del HUD, un elemento la posición indicada, del tamaño indicado, y con el color indicado. Además, alberga la clase texto y dispone de funciones para editarlo.

Es creado por:

La escena

Para:

- Añadir un elemento al HUD.

Parámetros de entrada:

- El constructor ninguno, tiene por defecto. Pueden ser modificados después de crearlo

Parámetros de salida:

- Ninguno. Aunque añade un elemento al HUD

Se le llama cuando:

Se necesita añadir un elemento al HUD.

Funcionamiento:

Se crea el elemento, se cambian los atributos deseados y listo.

Árbol de OSG:

La clase HUD element crea todos los parámetros, geometrías y opciones, para que OSG reconozca correctamente la estructura.

Declara las clases:

- Ninguna.

Declara las clases:

- Ninguna.

I.d.8.3. Text box

Localización del código:

text box.cpp

Qué función desempeña:

Esta es una clase para añadir texto a los elementos del HUD. No puede dibujarse a sí misma, hace falta que sea añadida a otra estructura, como puede ser hud element. Simplemente es un array de datos con un número para identificarlos.

Es creado por:

La escena

Para:

- Tener texto disponible para ser mostrado en el HUD

Parámetros de entrada:

- Id, o posición en la estructura.

Parámetros de salida:

- Ninguno. Aunque guarda un elemento de texto en la estructura.

Se le llama cuando:

Se quiere añadir texto a un elemento de HUD.

Funcionamiento:

Se crea la clase, se le editan los atributos y se inserta en el elemento del HUD que lo va a contener.

Árbol de OSG:

No lo usa directamente, simplemente almacena los textos posibles asociados a un hud element concreto.

Declara las clases:

- Ninguna.

I.d.9. Tipo input

Este grupo de clases son las empleadas para la administración de los periféricos.

Localización del código y las clases empleadas:

input module.cpp

I.d.9.1. Input Module**Qué función desempeña:**

La clase Input Module, es el núcleo del módulo. Además de declarar las clases necesarias para la administración de los periféricos, administra el propio módulo.

Los tipos que define para saber el tipo de señal que envía un periférico son:

- NORMAL
- QUIT

I.d.9.2. Control Map**Qué función desempeña:**

La clase Control Map, permite al Input Module la administración completa del teclado, en cualquiera estado del programa. Utiliza una estructura de pilas para guardar el "significado" de cada región y que sea interpretada correctamente.

Es creado por:

Input Module

Para:

- Administrar las entradas por teclado.
- Usar las funciones asociadas al teclado.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno. Pero puede administrar la asignación de las teclas, y realizar las acciones asociadas a las teclas.

Se le llama cuando:

La región update del programa o el limitador de FPS lo requiere.

Funcionamiento:

Al llamar a handle input, se comprueba el estado del teclado. Ésta función, cierra el programa si se devuelve una señal tipo QUIT, y de ser tipo NORMAL realiza la acción correspondiente. Solamente se ocupa de la "lectura" del periférico.

Para su funcionamiento se necesita:

- Cargar un mapa con la interpretación de las teclas de la clase concreta que lo va a usar. La dirección de este mapa se guarda en la clase control map, en un vector de punteros a la clase propietaria.
- Para usar un mapa de control, es necesario crearlo con un puntero a la clase y guardar el puntero en el vector de la clase control map.
- Existen unas funciones de código para cargar o editar el mapa de caracteres.

Los tipos que define para saber el tipo de clase en el que está son:

- T NONE
- T SCENE
- T MENU
- T FREECAMERA

Los tipos que define para saber el tipo de pulsación:

- S SINGLE
- S REPEATING

Emplea las clases:

- Ninguna

Cabe destacar que:

Esta clase es para administrar el periférico, que no solo interactúa con el mundo virtual, sino también con los distintos tipos de menús. Por tanto es interesante ver como funciona la estructura del tipo pila y los punteros para el correcto funcionamiento al margen del estado lógico del programa.

I.d.9.3. Mouse**Qué función desempeña:**

La clase Mouse, permite al Input Module la administración completa del ratón.

Es creado por:

Input Module

Para:

- Administrar el estado del Mouse.
- Muestrear y guardar los valores del Mouse en tiempo real.
- Usar las funciones asociadas al Mouse.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno. Pero se puede leer públicamente las estructuras de datos internas mediante las funciones adecuadas.

Se le llama cuando:

Cuando se necesita muestrear el periférico, o se necesita conocer su estado o parámetros.

Funcionamiento:

Una vez iniciado, cada iteración del UPDATE actualizará los datos que ofrece el Mouse. Cada vez que se desee, se tiene acceso a los datos.

Las estructuras que define son:

- MyMouseMotion
- MyMouseWheel
- MyMouseButtonDown

Emplea las clases:

- Ninguna

Cabe destacar que:

Esta clase es transparente a las demás. Se ocupa de encapsular las funciones del Mouse y de administrar sus datos. Si cualquier otra clase necesita acceder a los datos, o realizar acciones con el Mouse, solo tiene que emplear las funciones públicas.

I.d.9.4. Joystick

Qué función desempeña:

La clase Joystick, permite al Input Module la administración completa del periférico Joystick o Game Pad.

Es creado por:

Input Module

Para:

- Administrar el estado del Joystick.
- Muestrear y guardar los valores del Joystick en tiempo real.
- Usar las funciones asociadas al Joystick.

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno. Pero se puede leer públicamente las estructuras de datos internas mediante las funciones adecuadas.

Se le llama cuando:

Cuando se necesita muestrear el periférico, o se necesita conocer su estado o parámetros.

Funcionamiento:

Una vez iniciado, cada iteración del UPDATE actualizará los datos que ofrece el Joystick. Cada vez que se desee, se tiene acceso a los datos.

Las estructuras que define son:

- MyJoystickButton
- MyJoystickHat
- MyJoystickAxis

Emplea las clases:

- Ninguna

Cabe destacar que:

Esta clase es transparente a las demás. Se ocupa de encapsular las funciones del joystick y de administrar sus datos. Si cualquier otra clase necesita acceder a los datos, o realizar acciones con el Joystick, solo tiene que emplear las funciones públicas.

I.d.9.5. WiiMote**Qué función desempeña:**

La clase WiiMote, hace de puente entre la librerías del wiimote externas, que encapsulan su funcionamiento, y los datos que se van a manejar en nuestro mundo virtual.

Es creado por:

Input Module

I.d.10. Tipo Menu

Este grupo de clases son las empleadas para la administración de los menús. Pero no incluye su muestra por pantalla, ya que esas clases forman parte del grupo HUDs.

I.d.10.1. Menu**Qué función desempeña:**

La clase menú, se emplea para la navegación en el menú inicial. Al iniciar el programa, espera hasta que el usuario elija una opción. Dispone de varios estados para poder distinguir las distintas opciones.

Es creado por:

Game Module

Para:

- Administrar el cambio de estados del menu

Parámetros de entrada:

- Imagen de fondo del menú

Parámetros de salida:

- Ninguno. Pero se crea un menú con las opciones elegidas.

Se le llama cuando:

Al comenzar el game module. Y cada vez que se vuelve al menú gráfico de inicio.

Funcionamiento:

Se resalta la opción en la que el usuario está situado. Puede cambiar la selección o confirmarla.

Las estructuras que define son:

- Ninguna

Emplea las clases:

- HUD

Árbol de OSG:

Solo emplea el árbol, para añadir o quitar el hijo del hud creado para el menú. Para ello usa la función `getscenedata` ya que el HUD se hijo directo de ella.

I.d.11. Tipo Módulos

Este grupo de clases son las empleadas como núcleo de los módulos, descrito en la sección de implementación de los módulos.

Localización del código:

xxxxxx module.cpp

I.d.12. Tipo Objetos3D

Este grupo de clases son las empleadas para crear los objetos 3D.

Localización del código y las clases derivadas:

objects.cpp

I.d.12.1. Object

Qué función desempeña:

La clase `object` enumera los tipos de objetos, y sirve para crear objetos derivados con más propiedades. Cada objeto dispone de una ID, nombre, y tipo.

Es creado por:

Object factory module

Para:

- Crear un objeto concreto diseñado como derivado de `object`.

Parámetros de entrada:

- ID
- String con un nombre
- Tipo enumerado en object.h

Parámetros de salida:

- Ninguno. Pero se crea un objeto nuevo.

Se le llama cuando:

Se le manda a Object Factory Module crear un objeto nuevo.

Funcionamiento:

Cada vez que se Object Factory Module llama a esta clase, crea un nuevo objeto del tipo especificado. Esta clase NO se encarga de cargar el modelo, ni prepararlo, simplemente crea o quita en el árbol de OSG objetos.

Las clases que define son:

- Movable
- Sheep
- Castle
- Catapult
- King
- Arrow

Emplea las clases:

- Effects

Cabe destacar que:

Esta clase únicamente se ocupa de derivar tipos de objetos concretos, e interactuar con el árbol de OSG, suponiendo que los objetos 3D ya están preparados para funcionar correctamente. Es decir, la clase Object Factory Module ya ha cargado el formato de objeto y lo ha inicializado. Esta clase numera y almacena todos los objetos que creamos, aunque sean iguales.

I.d.12.2. Movable

Qué función desempeña:

Permite añadir propiedades a la clase definida como object.

Es derivado de:

Object

Para:

- Tener un objeto al que se le puedan añadir los efectos declarados en effect.h

Parámetros de entrada:

- ID
- String con un nombre
- Tipo enumerado en object.h

Parámetros de salida:

- Ninguno. Pero se crea un objeto nuevo.

Se le llama cuando:

Se le manda a Object Factory Module crear un objeto nuevo.

Funcionamiento:

Similar al de objeto, al ser derivado de él.

Las clases que define son:

- Ninguna, aunque derivan de él varios modelos.

Emplea las clases:

- Effects

Cabe destacar que:

Esta clase es igual que object, con el añadido de que puede recibir varios punteros para añadir efectos al PAT de OSG. Además, deriva varios modelos distintos, de iguales propiedades pero que difieren en algún parámetro.

Árbol de OSG:

Al scenedata, es decir, el root de la escena, se le añade un subárbol compuesto de:

- Un osg::PositionAttitudeTransform, es decir, un nodo que indica la posición y orientación
- Un osgDB::ReadNodeFile, es decir, el model cargado de un archivo.obj

Para tener localizada la zona del árbol de cada objeto, se tiene un puntero al nodo PAT, que puede ser obtenido mediante la función get pat de la clase movable.

I.d.12.3. Derivados de movable**Qué función desempeñan:**

Son clases movable, que difieren en el TIPO de modelo asignado. Es decir, OSG los dibuja de forma diferente, aunque tengan las mismas propiedades.

Son derivados de:

Movable

Para:

- Tener distintos modelos con las propiedades de movable

Parámetros de entrada:

- ID
- String con un nombre
- Tipo enumerado en object.h

Parámetros de salida:

- Ninguno. Pero se crea un objeto nuevo

Se le llama cuando:

Se le manda a Object Factory Module crear un objeto nuevo.

Funcionamiento:

Similar al de movable, al ser derivado de él.

Los elementos derivados son:

- Sheep
- Castle
- Catapult
- King
- Arrow

I.d.13. Tipo Salida

Este grupo de clases son las empleadas para la salida de datos que no se produce por HUD, sino por consola.

I.d.13.1. Console

Localización del código:

display module.cpp

Qué función desempeña:

Gestiona la consola a un nivel mayor que el habitual. Además de disponer el modo típico, cada mensaje sacado por la consola aparece en una nueva línea, se puede usar en modos para mostrar pantallas con menús o parámetros fijas, sin deslizamiento vertical. Está pensado para que aunque funcione en modo menú, funcione en paralelo con las otras actividades del programa e incluso pueda mostrar mensajes extra debajo de los menús mostrados.

Es creado por:

Display Module

Para:

- Gestionar los parámetros de la consola, posición del cursor y color
- Mostrar menús y permitir navegación
- Mostrar listas de parámetros en tiempo real
- Utilizar macros

Parámetros de entrada:

- Ninguno

Parámetros de salida:

- Ninguno

Se le llama cuando:

- Se actualiza el estado de la consola, esté o no activa
- Se desea cambiar alguno de los parámetros de la consola

Funcionamiento:

En modo DEFAULT, la consola funcionará normalmente. Si se cambia al modo DATADISPLAY, se puede navegar en paralelo a la ejecución del programa por una serie de menús con macros y opciones. Si se quiere cambiar manualmente la posición del cursor, o su color, basta con usar las funciones que tiene implementadas para ello. Las funciones son públicas ya que el resto de clases probablemente necesiten en un momento dado utilizarlas.

Para acceder al menú de la consola, se debe pulsar la tecla P en la ventana de ejecución del programa. Utiliza el teclado para detectar. Las opciones se activan con el teclado numérico. Para evitar problemas visuales, se limitan a 9 las opciones de cada menú. La opción 9 siempre es volver o salir del menú. Si se necesitaran páginas, se avanza y retrocede con las teclas 7 y 8.

Las clases que define son:

- Ninguna

Emplea las clases:

- Ninguna

Cabe destacar que:

Aunque es una clase auxiliar, es fácil y cómoda de usar, para dar formato al texto que sale por la consola. Además, la implementación de menús permite añadir de forma más o menos sencilla nuevas macros y menús. *** Ver HOW TO

I.d.14. Tipo Laberinto**Qué función desempeña:**

Esta única clase es empleada para crear la estructura lógica del laberinto. Una vez creada, el módulo object factory se encargará de trasladarla al mundo virtual.

Es creado por:

Object Factory Module

Para:

Crear la estructura lógica del laberinto

Parámetros de entrada:

- Posición inicial (X,Z)
- Ancho y Largo del laberinto
- Ancho y Largo de la celda
- Ver solución

Parámetros de salida:

- Ninguno. Aunque guarda en el árbol OSG la estructura del laberinto

Se le llama cuando:

Se crea la escena

Funcionamiento:

Una vez creada, es integrada en la estructura del árbol de OSG.

Las estructuras que define son:

- Ninguna

Emplea las clases:

- Ninguna

Cabe destacar:

Nada

I.d.15. Tipo Texturas

Este grupo de clases son las empleadas para asignar todos los tipos de texturas que ha necesitado el proyecto.

I.d.15.1. Skybox

Localización del código:

skybox.cpp

Qué función desempeña:

Encajona la cámara entre 4 caras de texturas. Debe estar en la misma posición de la cámara para crear el efecto de tener un paisaje lejano. Sirve para dar una sensación de mundo inmenso, aunque es una simple fotografía.

Es creado por:

Escena

Para:

Crear la textura asignada al fondo del mundo.

Parámetros de entrada:

- Imagen que será el skybox.

Parámetros de salida:

- Ninguno. Aunque guarda en un vector la clase creada.

Se le llama cuando:

La cámara cambia de posición, para actualizar la posición del skybox y que funcione correctamente el efecto.

Funcionamiento:

Una vez creada la clase con la imagen, se le asigna la posición de la cámara y funciona.

Las estructuras que define son:

- Ninguna

Emplea las clases:

- Ninguna

Cabe destacar:

Está implementada de forma sencilla, usa la misma imagen para las 4 caras, y no está dibujado el cielo ni el suelo. Si se quiere suelo hay que crearlo aparte.

I.d.15.2. Billboards**Localización del código:**

billboard.cpp

Qué función desempeña:

Son texturas 2D, que siempre están de cara al observador. Se utilizan para consumir menos memoria, pues solo son imágenes, y tener texturas visibles de cara al espectador siempre.

Es creado por:

Object Factory Module

Para:

- Crear billboards

Parámetros de entrada:

- Imagen que usará el Billboard creado

Parámetros de salida:

- Ninguno.

Se le llama cuando:

Se crean o se destruyen

Funcionamiento:

Se carga el modelo de imagen que se desea usar en el Object Factory Module, se le asigna una posición y un tamaño, y se crea en OSG usando la clase Billboards.

Las estructuras que define son:

- Ninguna

Emplea las clases:

- Ninguna

Cabe destacar que:

Se debe guardar la imagen cargada a mano en un puntero. Se define en Object Factory Module para tenerlo más a mano y ahorrar memoria, al no tener que declararlo en otra clase y cargarlo cada vez que va a ser creado.

I.e. Implementación de código común o general:

Existen partes del código, que pueden ser reutilizadas para varias clases realizando cambios mínimos. Este código puede cumplir con las especificaciones de la fase del diseño con pequeñas adaptaciones. En esta sección se van a explicar cuáles son, porque se han implementado de esta forma, y qué clases las aprovechan.

I.e.1. Sistema de control

El sistema de control es común a varias clases del código. Esta parte común aparece en todas las clases que necesiten usar los datos de los periféricos y quieran disponer de varios tipos de configuraciones intercambiables. Se distinguen 2 clases de sistema de control descritos en la fase de diseño:

- Para el teclado
- Para el resto de los periféricos

¿Por qué hacer esta distinción?

La razón es sencilla, el teclado dispone únicamente de teclas, así que es fácil asignar unas macros a cada una de ellas, e incluso cambiarlas, gracias a la clase control map que se va a diseñar. Los demás periféricos, no solo disponen de teclas, también otros parámetros. Es más sencillo si en lugar de usar la clase control map, usamos un sistema de llamadas a funciones con las distintas interpretaciones de los valores.

I.e.1.1. Clase Control Map

La clase control map tiene un diseño ya implementado, descrito en el apartado *****. Aunque para funcionar, necesita un código común presente en las clases que requieran de su uso.

Si se desea que una clase tome o deje el control del teclado, se:

- I - Dispone de un mapa para esa clase: Se debe añadir una clase control map a la pila con el mapa de caracteres actualizado. Para ello, se debe crear una clase nueva, añadir las funciones, e insertar los datos en el vector del input module.
- II - Se le puede pasar el control, añadiendo el mapa a la pila maps del input module con la función push.
- III - Se le puede quitar el control, quitando el mapa a la pila maps del input module con la función pop.

I.e.1.2. Sistema de configuración de controles

Cada objeto que vaya a manejarse con los datos de los periféricos, necesita cumplir estos requisitos de implementación:

- Disponer de una enumeración de configuración general.
- Disponer de una enumeración de configuración para cada periférico.
- Cambiar de controles mediante la función correspondiente.
- Tener correctamente programada cada configuración.

I.e.2. Sistema ...

...

I.f. Conclusiones

Como se puede apreciar en este capítulo, siguiendo unos sencillos pasos es posible añadir y modificar cada elemento del diseño de una forma sencilla. Gracias a un diseño previo, que contempla las necesidades de cada apartado, se puede implementar de forma eficaz. A lo largo de toda la implementación, se encontraron muchos obstáculos y zonas con imprevistos. Se resolvieron, y se retroalimentaron en la fase del diseño para que todo cuadrara finalmente. Es difícil, por no decir imposible, el realizar un diseño teórico a priori que pueda implementarse correctamente. Gracias al sistema seguido a lo largo de estas partes, es posible conseguir como resultado el cumplimiento de los objetivos planteados en la primera parte de forma satisfactoria.

Apéndice II

Qué hacen las clases y como usarlas

II.a. Introducción

El objetivo de este capítulo, es describir a grandes rasgos, qué hacen y como añadir o cambiar cualquier parte del programa utilizando las clases diseñadas e implementadas. De esta manera, cualquier mejora que planteemos podrá ser fácilmente aplicada sin tener que cambiar partes básicas del diseño. En esta parte se responde a la pregunta: ¿ Cómo se puede editar a nivel alto el programa para obtener las funcionalidades implementadas?

II.b. Qué hacen las clases

Pequeño resumen para ver qué hace cada clase a nivel técnico.
/** Insertar foto de tabla

II.c. Como usar las clases: (HOW TO)

En esta parte se explica básicamente, como usar cada clase para que haga la función para la que fue diseñada. **** Añadir tabla con guías.

II.c.1. Tipo audio

II.c.2. Tipo auxiliar

II.c.3. Tipo cámara

II.c.4. Tipo efectos

II.c.5. Tipo escena

II.c.5.1. Como crear una fuente de luz

Para añadir satisfactoriamente una luz a la escena, debemos:

- Declararlo un tipo `osg::ref_ptr<osg::Light>light`
- Definir 1 tipos de luz
- Cambiar constantes de atenuación
- Crear nueva fuente de luz `osg::ref_ptr<osg::LightSource>ls`
- Añadir los tipos de luz creados `ls->setLight(light.get());`
- Se pueden asignar más tipos de luz a la misma fuente
- Asignar en el árbol de OSG la nueva fuente

Los tipos de luz se colocan con los comandos:

- `light->setAmbient(osg::Vec4d(r, g, b, a));`
- `light->setDiffuse(osg::Vec4d(r, g, b, a));`
- `light->setPosition(osg::Vec4d(r, g, b, a));`

Los tipos de constantes de atenuacion serán:

- `light->setConstantAttenuation(float);`
- `light->setLinearAttenuation(float);`
- `light->setQuadraticAttenuation(float);`

II.c.6. Tipo grupos

II.c.7. Tipo HUD

II.c.8. Tipo input

II.c.8.1. Como añadir un mapa de caracteres nuevo si una clase dispone de sistema de teclas:

Para crear un mapa de teclas nuevo, o editarlo, se necesita:

- Disponer de un puntero al control map en la clase que lo va a usar.
- Inicializarlo en el constructor con el tipo de la clase.
- Cargar los controles cuando se quieran usar:
 - Usar la función set para conseguir un puntero a la clase.
 - Editar el mapa de control con las funciones insert y remove.

- Usar la configuración colocando la estructura en la pila con la función push.
- Cuando se quiera devolver el control al anterior miembro de la pila, quitar la estructura de la pila con la función pop.

II.c.8.2. Como añadir una configuración:

Si lo que desea es añadir una configuración, hay que seguir los siguientes pasos:

- Añadir espacio en el enumerado de la cabecera de la clase para la configuración general y la de cada periférico que vaya a registrar.
- Si es la primera, inicializar en el constructor de la clase.
- Modificar las funciones para que cambiar la configuración también pueda tomar el valor nuevo.
- Modificar la zona de cases en el update del objeto, para que admita la nueva configuración.
-

II.c.9. Tipo menu

II.c.10. Tipo módulos

II.c.11. Tipo objetos3D

II.c.11.1. Como añadir un objeto nuevo

Para añadir satisfactoriamente un objeto nuevo a la escena, debemos:

- Declararlo en...

II.c.11.2. Como añadir sistema de teclas a una clase:

Si se desea que una clase se pueda controlar mediante la clase control maps, se debe seguir los siguientes pasos:

- Añadir el nombre del tipo en la cabecera del input module. Añadiendo además en el cuerpo del constructor el tipo de mapa para esa clase.
- Añadir función set(puntero a la clase) en la cabecera del input module. Añadiendo también su cuerpo, la variable privada para almacenar el puntero y su vector de mapa de teclas.
- Sobrecargar la función item con el puntero a la nueva clase.
- Añadir el cuerpo a las funciones Clear y execute.

II.c.12. Tipo salidas

II.c.12.1. Como añadir menús al modo DATADISPLAY de la consola...

La clase console está diseñada a un nivel bajo, para ahorrar complicaciones y que sea lo más simple posible. Por eso hay que seguir varios pasos básicos a la hora de añadir un nuevo menú. Estos pasos son:

- Crear una nueva función menú en la clase console
- Copiar una plantilla de menú. SIN reiniciar la variable `iaux = 0`. Esto solo se hace en el primer menú, que es `MenuDataDisplay`.
- Modificar al gusto la zona de texto.
- Añadir las macros u opciones correspondientes en el switch de los casos
 - Si es una macro o función, simplemente colocar la función
 - Si hay que subir de nivel, colocar la función `Console::Forward` y el menú siguiente a continuación. ATENCIÓN, es importante el orden.
 - Si hay que bajar un nivel, colocar la función `Console::Back`
- Si se han seguido los pasos correctamente, se debe poder acceder y usar el menú tal como se espera, sin importar qué hacen los demás.

II.c.13. Tipo texturas

II.d. Conclusiones

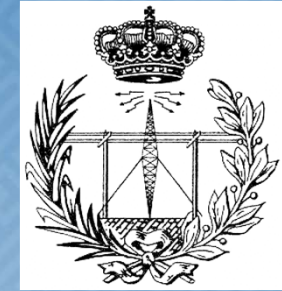
Como se ha podido ver, este capítulo es meramente descriptivo. Detalla unos pasos concretos para poder usar las clases sin tener necesidad de entender o volver a diseñar partes de la estructura. Esto ya se ha realizado en los capítulos anteriores, precisamente para poder usar todas las funcionalidades de la manera más sencilla, fácil, y rápida posible.

II.e. Encuesta de intuitivo

II.f. Encuesta de cómodo



Proyecto fin de carrera E.T.S.I.I.T.



Titulación:

Ingeniero de telecomunicaciones

Título del proyecto:

Evaluación de periféricos para el control de mundos virtuales



Víctor Goñi Sanz
Jeser Zalba
David Oyarzun

Pamplona, 9/9/2011

Tutores

UPNA

Jeser Zalba

Departamento de Ingeniería eléctrica y electrónica

VICOMTech

David Oyarzun

Área de animación 3D y entornos virtuales interactivos

Objeto del proyecto

- Recopilar información de los diversos periféricos.
- Análisis de sus ventajas y desventajas
- Investigar distintas formas de utilizarlos

Descripción del proyecto

Para poder llegar a este objetivo:

- Se ha diseñado un sencillo mundo virtual.
- Se han adaptado librerías y módulos
- Se ha diseñado un interfaz estándar común
- Se ha usado a 22 sujetos de prueba

Estado del arte I

Para comenzar a desarrollar el proyecto:

- Buscar que periféricos usar



- Usar librerías de scripts (Glovepie)

- Buscar una plataforma en la que implementarlos
Software propietario (No acceso al de la empresa)

Vs

- Software libre (Crearlos con tutoriales)

Estado del arte II

Para crear una plataforma adecuada (C++):

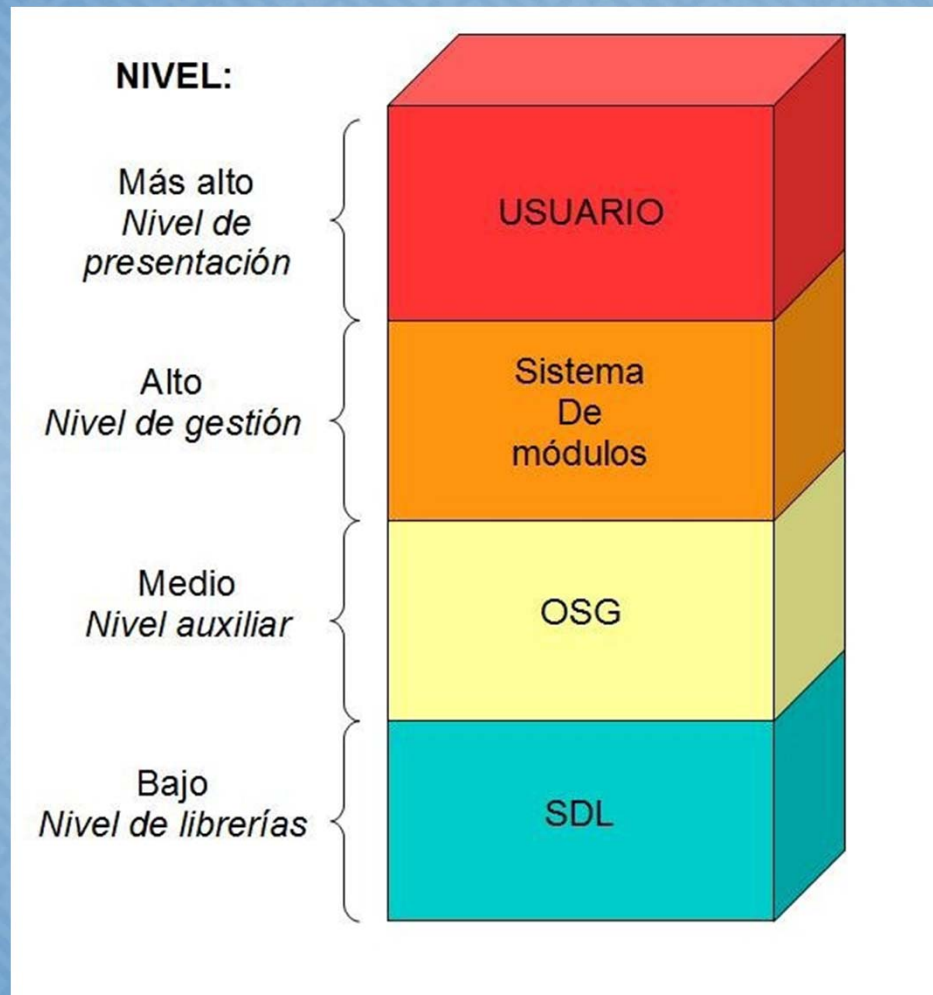
- OpenSceneGraph (OSG - OS)
- Simple DirectMedia Layer (SDL - OS)

Basándome en **Brian Malloy** de Clemson University, EEUU

Librerías de periféricos

- | | |
|--------------------|-----------------------------|
| - Wiiyourself (OS) | Muy usado |
| - OpenNI, NITE | Únicos libres en su momento |

Desarrollo del MV I



Separación en capas

- Manejo independiente
- Distintos niveles de profundidad para cambios
- Permitir desarrollo por módulos

Aspecto nivel presentación

El usuario final podrá iniciar el programa de forma sencilla

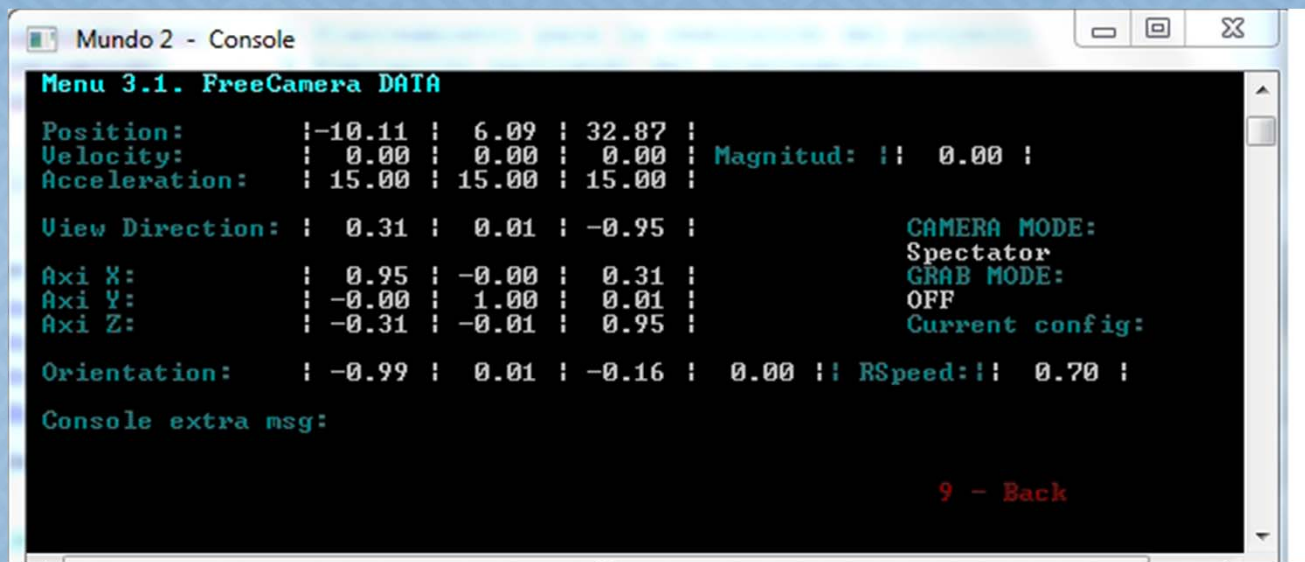


- Aspecto ameno
- No es simplemente un campo de pruebas
- Diseño de la escena

Aspecto final

Aspecto nivel presentación II

Se añade una sencilla consola



```
Mundo 2 - Console
Menu 3.1. FreeCamera DATA
Position:      |-10.11 | 6.09 | 32.87 |
Velocity:     | 0.00 | 0.00 | 0.00 | Magnitud: || 0.00 |
Acceleration: | 15.00 | 15.00 | 15.00 |
View Direction: | 0.31 | 0.01 | -0.95 |
Axi X:       | 0.95 | -0.00 | 0.31 |
Axi Y:       |-0.00 | 1.00 | 0.01 |
Axi Z:       |-0.31 | -0.01 | 0.95 |
Orientation:  |-0.99 | 0.01 | -0.16 | 0.00 || RSpeed:|| 0.70 |
Console extra msg:
9 - Back
```

- Permite visualización de datos
- Múltiples opciones útiles para los programadores que permiten Calibrar los periféricos, localizar errores, cambiar en caliente opciones

Diseño del MV I

Se va a diseñar todo el mundo virtual para:

- Minimizar problemas por conflictos entre partes
- Evitar código redundante
- Permitir cambios de forma aislada, asegurando que seguirán funcionando otras partes correctamente

Diseño del MV II (Módulos)

Se diseñan varios módulos

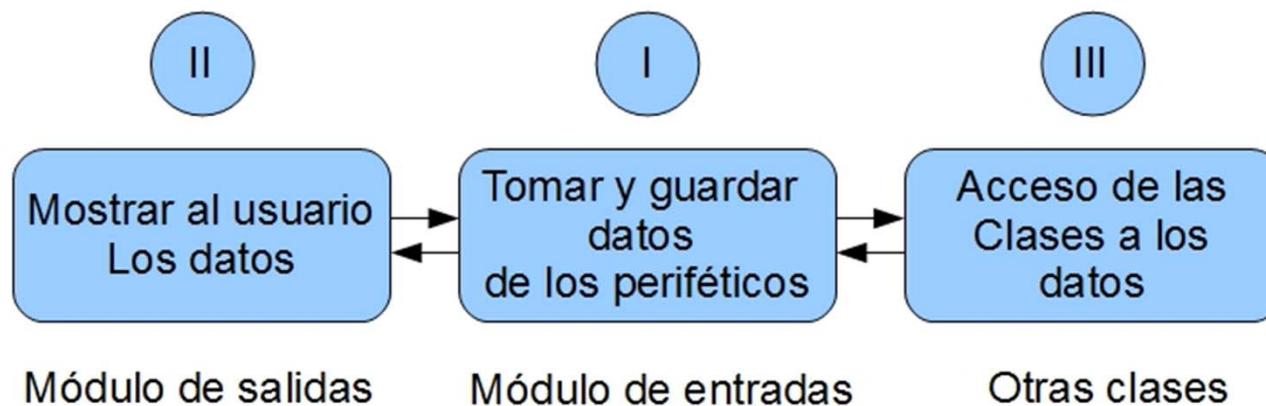
- Núcleo (Base)
- Registro de módulos
- Tiempo (Coordinar)
- Audio
- Video
- Entradas (Inputs)
- Consola (Gestiones)
- Game (Lógica de menús y escena)
- Crear objetos

Diseño del MV III (lógica)

Al iniciar la aplicación

- Se carga el núcleo, registro, y módulos.
- Se comienza por el 1er menú
- Se maneja hasta terminar la aplicación

Para control de tiempo y entradas



Fases (lógica)

- Inicialización
- Actualización de entradas
- Actualización de datos
- Actualización de video (máx FPS)

Diseño del MV IV (inputs)

La detección de entradas debe ser limpia y transparente.

- Lectura de todos los datos (Siempre)
- Actualización de las variables temporales (Siempre)
- Fase de lectura de las variables (Solo si necesario)

Permiso de escritura, solo administrador de dispositivos

Permiso de lectura, resto de recursos

Permite que muchos recursos accedan a la información

SIN modificarla

Evaluación I (Planteamiento)

Me voy a basar en 3 aspectos

- Potencia (Objetiva, según información TR)
- Intuitividad (Facilidad y rapidez de uso, Subjetivo)
- Comodidad (Percepción del usuario, Subjetivo)

Evaluación potencia

Se evaluará numéricamente con la fórmula:

$$= \frac{\text{---}}{\text{---}}$$

Donde

Rc = Los que cumple cada periférico

Rt = Necesidades exigidas por el proyecto + disposición

Se ha tenido en cuenta:

- Detección absoluta o gradual (Coeficientes de eficiencia aplicados)
- Ratio de acciones

Necesidades:

- Movimiento en ejes X,Y,Z (6 puntos)
- Giro 1 eje, y su perpendicular (4 puntos)
- Uso de 4 macros/botones (4 puntos)

Disposición:

- Existe posición de equilibrio (1 punto)
- Situación de los botones (3 puntos)
- Necesita mandos físicos (3 puntos)

Evaluación de los usuarios

- Se ha creado una escena virtual que resulta agradable.
- Se ha adaptado un algoritmo para crear laberintos 3D
- Se le hace al usuario atravesarlo
- Se emplea una cámara con 6 grados de libertad
- La cámara está limitada a manejo en primera persona.
- Tras realizar cada recorrido, se reinicia y se hacen las Preguntas correspondientes.

Evaluación Intuitividad y comodidad

Como ya se ha comentado, se usan encuestas

Intuitividad:

- Predecible
- Rapidez (Curva de aprendizaje)
- Facilidad de uso

Comodidad:

- Precisión
- Comodidad
- Asequible (Precio)

Resultados potencia

	[Movilidad	Cm	Giro	Cm	Botones]	[Equilibrio	Distrib	Aparatos]	Potencia:
Teclado	6	0,7	4	0,3	4	1	2	0	0,59
Ratón	0	0,7	4	1	2	1	3	0	0,48
Joystick	6	1	4	1	2	1	2	0	0,71
Alfombra	6	0,7	4	0,3	2	1	2	1	0,54
Wiimote	6	1	4	1	4	0	3	2	0,90
Kinect I	4	1	4	1	0	0	3	3	0,67
Kinect II	6	1	4	1	0	0	3	3	0,76
Tec + Rat	6	0,7	4	1	4	1	2	0	0,72
Alf + Wiimote	6	1	4	1	4	0	3	0	0,81
Alf + Kinect II	6	1	4	1	4	0	2	3	0,90
Ideal:	6	1	4	1	4	1	3	3	1,00

Usando la mediana de la potencia

- Se descartan los que quedan por debajo para su Estudio.

Resumen de los resultados

Finalmente, se examinaron 5 configuraciones distintas

Resumen					SOBRE
Dispositivo	Potencia	Intuitivo	Cómodo	Promedio	
Teclado + Ratón	71,00	88,07	70,80	76,62	100
Gamepad	90,00	72,84	73,98	78,94	100
Wii mote	76,00	73,75	72,05	73,93	100
Kinect II	81,00	25,91	45,34	50,75	100
Wii mote+Dance	90,00	85,34	63,75	79,7	100

	$X > 0,7$	$X > 2/3$	$X > 2/3$	$X > 2/3$
	-	$1/3 < X < 2/3$	$1/3 < X < 2/3$	$1/3 < X < 2/3$
	$X < 0,7$	$X < 1/3$	$X < 1/3$	$X < 1/3$

	Teclado y ratón	Gamepad	Wii mote	Kinect	Wii + Dance
Votos favorito	4	1	2		15
Votos retirar:				15	1

Conclusiones I

Teclado y ratón:

Potencia: 71

Intuitivo: 88

Cómodo: 70

Total: 76

- Clásico.
- Cómodo e intuitivo.
- No es tan preciso ni da tantas posibilidades como otros periféricos estudiados

Conclusiones I

Gamepad:

Potencia: 90

Intuitivo: 73

Cómodo: 73

Total: 79

- Cómodo de usar, sentado y de pie
- Es el mejor valorado en potencia y comodidad
- Posición de equilibrio pasiva

Conclusiones I

Wiimote:

Potencia: 76

Intuitivo: 74

Cómodo: 72

Total: 74

- Es cómodo y potente.
- Ofrece nuevas posibilidades para interactuar.
- Es preciso y divertido de usar.

Conclusiones I

Kinect II:

Potencia: 81

Intuitivo: 26

Cómodo: 45

Total: 51

- **NO** resulta cómodo, ni intuitivo.
- No sirve para el propósito del estudio, pero ofrece posibilidades nuevas que investigar.
- Útil para captar movimientos y manejar avatares v irtuales

Conclusiones I

Wiimote + Dancepad:

Potencia: 90

Intuitivo: 85

Cómodo: 64

Total: 80

- No es tan cómodo como otros, pero da sensación de inmersión y es divertido.
- Combinación potente, permite mucha precisión.

Conclusiones Final

- ¿Cuál es el mejor?

- No se considera uno mejor que otro, depende del uso.
- El que parece más adecuado es Wiimote + Dancepad para di versión e interacción.
- Como sistema tradicional, el teclado + ratón sirve.
- Se podría incluir el kinect, para añadir información y el manejo de avatares virtuales.

- ¿Cuál es más útil?

- Gamepad, mejor valorado
- Más preciso el uso de wiimote

- ¿Cuál es más versátil?

- No hay uno concreto, combinándolos se consiguen muchas Nuevas posibilidades.

Conclusión final

Licencias