DESARROLLO DE INTERFACES

DE USUARIO NATURALES

CON KINECT

Universidad Politécnica de Valencia

DSIC

Proyecto Final De Carrera

Ingeniería Informática

Enrique Trilles Andreo

Supervisado por:

M. Carmen Juan Lizandra

Diciembre 2012

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1

INTRODUCCIÓN

1.1.- Motivación

En los últimos años, la popularidad de las interfaces naturales ha experimentado un crecimiento abrumador. Las interfaces naturales permiten a un usuario interaccionar con un ordenador de una forma sencilla y sin la necesidad de necesitar un aprendizaje complejo y/o especializado [DAVE 12]. Además, la adaptación de una de estas interfaces a un robot lo puede dotar de una mayor humanidad. En los últimos años diversas empresas, de entre las que destacamos Microsoft, Sony y Nintendo, han desarrollado diversos dispositivos que aportan interfaces naturales [Figura 1.1] y de fácil acceso para las personas a sus máquinas de entretenimiento respectivas, a saber: Xbox360 y Microsoft Kinect, Playstation MOVE motion controller, Nintendo Wii, y desde Noviembre de 2012, Nintendo WiiU.

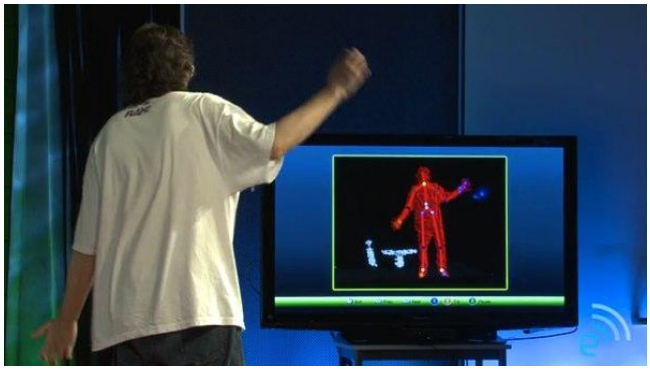

Figura 1.1 – Control de aplicación con Interfaz Natural

La lista de ejemplos expuesta hace referencia a los casos más populares a día de hoy, gracias en parte a su gran aporte al entretenimiento doméstico. Se tratan de dispositivos comerciales y cerrados que no permiten de forma legal el desarrollo de aplicaciones o herramientas que hagan uso de estos dispositivos para que puedan ser de algún tipo de utilidad para la comunidad científica. Solo una de estas empresas, Microsoft, ha ofrecido de forma gratuita un kit de herramientas para permitir a todos los desarrolladores interesados en esta materia el poder desarrollar y trabajar con su dispositivo de interfaz natural, Microsoft Kinect. Es así como empiezan a aparecer los primeros proyectos e ideas sobre el control y funcionamiento de este tipo de interfaces, aunque a día de hoy no existan muchos trabajos complejos al respecto, es cuestión de tiempo que el número de proyectos de carácter científico aumente, esperemos, de forma exponencial.

Así pues, con el objetivo de profundizar en el ámbito de las interfaces naturales, se desarrollará para Windows una aplicación que haga uso de dicha interfaz para poder jugar a un pequeño juego de ingenio, siempre utilizando herramientas y librerías de código abierto, ya que nos proporcionan una mayor libertad a la hora de trabajar con ellas, gracias a la posibilidad de poder acceder al código fuente, a modificarlo y adaptarlo acoplándolo a nuestras necesidades según se considere oportuno. En el caso particular de este proyecto, se

creará una librería extra para obtener únicamente la información necesaria (posición de las manos), separándola del resto de información obtenida.

1.2.- Objetivos del proyecto

Como ya se ha comentado, el objetivo del proyecto es realizar una aplicación que represente un pequeño juego de ingenio, un puzzle, que el usuario deberá resolver haciendo uso en todo momento de una interfaz de usuario natural. Para llevar a cabo este proceso, será necesario que el ordenador en el que se ejecute la aplicación cuente con un sensor Kinect instalado y perfectamente configurado. Además, debido a la naturaleza de la implementación realizada, también se ha añadido a modo de expansión, una ampliación al programa que nos permitirá, en caso de desearlo, manejarlo mediante el uso de teclado y ratón, en caso de no desearse por algún motivo específico, utilizar la interfaz natural.

Debido a que el desarrollo de la aplicación va a partir de cero en su totalidad, van a resultar necesarios diversos tipos de herramientas y librerías que abarquen los siguientes 3 aspectos:

- Modelado de Objetos en 3D.

- Drivers y controladores para sensores de movimiento y de Microsoft Kinect.

- Librería gráfica y de gestión de elementos 3D, así como manejadores para el control de la interacción de entrada/salida con el hardware del ordenador.

Dado que se ha buscado en todo momento utilizar herramientas libres para el desarrollo del proyecto, y debido a la naturaleza de los componentes y materiales necesarios, la elección de las herramientas se ha visto bastante restringida.

En el caso concreto del modelado de todos los objetos tridimensionales, y de la interfaz gráfica en general, se ha optado por la opción gratuita más popular, Blender. Mientras que para los drivers para ordenador del sensor Kinect, se ha optado por la opción más popular, OpenNI. Tanto el programa Blender como la interfaz de control de Kinect OpenNI se tratarán con más detalle en capítulos posteriores del documento.

Para cumplir los objetivos del proyecto se han seguido estos pasos:

1.- Modelado de Objetos en 3D.

Dada la complejidad de los diferentes elementos gráficos de nuestra aplicación, ha sido necesario utilizar una herramienta de diseño y modelado 3D para poder obtener dichos elementos con un acabado robusto y coherente. Además, se ha tenido que buscar una forma de poder exportarlos a la librería gráfica que utilizará nuestra aplicación.

2.- Interpretación de la información.

Los drivers y controladores de los sensores de Kinect que Microsoft nos ofrece por defecto ya nos facilitan la tarea de la interpretación de los datos obtenidos por los sensores hasta cierto punto. Pese a esto, va a ser necesario optar por una opción más completa que permita implementar una librería, o ampliar una existente, para recoger estos datos y procesarlos adaptándolos a nuestras necesidades. De esta forma, se facilitará lo máximo posible las fases de desarrollo posteriores de nuestra aplicación a nivel lógico. En el caso de la aplicación por ratón, no será necesario desarrollar este paso.

3.- Carga y funcionamiento

Con todos los elementos necesarios ya preparados, se implementará, por un lado, una librería que se encargue de gestionar todos los aspectos gráficos de la aplicación con la ayuda de una librería gráfica. Por otro lado, hay que desarrollar un manejador, que interprete la información obtenida por la librería del paso anterior (o por el ratón), para que la parte gráfica de la aplicación responda y actúe de la forma deseada.

1.3.- Estructura

El resto de capítulos presenta la siguiente estructura:

Capítulo 2: Introducción a los distintos aspectos tratados en el proyecto.

Capítulo 3: Estado del arte y situación de las interfaces naturales.

Capítulo 4: Presentación de las herramientas, aplicaciones y librerías utilizadas para el desarrollo del trabajo.

Capítulo 5: Explicación detallada paso por paso de todas las tareas realizadas para la realización de la aplicación.

Capítulo 6: Conclusiones y posibles ampliaciones y/o mejoras de la aplicación desarrollada.

Capítulo 2

CONOCIMIENTOS PREVIOS

2.1.- Introducción a las interfaces de usuario

Dentro de cualquier sistema de un ordenador, la interfaz de usuario se considera el aspecto más importante, debido a que, para muchos usuarios la propia interfaz es el sistema. Es la parte del sistema que podemos ver, que podemos escuchar, podemos interaccionar con ella, y en algunos casos, podemos incluso hasta tocarla. Cualquier otro aspecto del software queda oculto detrás de la interfaz, ya sea un monitor, un teclado, un ratón, o cualquier otro hardware que se utilice para interaccionar con el sistema [GALITZ 07]. Los objetivos que hay que buscar a la hora de modelar una buena interfaz de usuario son simples: Conseguir que el usuario pueda trabajar con el ordenador o sistema de forma sencilla, productiva, y en algunos casos, entretenida.

La búsqueda del diseño de una buena interfaz de usuario ha dado pie a un campo de estudio propio llamado Interacción Persona-Ordenador (Human-Computer Interaction, HCI). El HCI es el estudio, planificación y diseño de cómo los humanos interaccionan con los ordenadores para poder conseguir que las necesidades de esta persona sean satisfechas de la forma más efectiva posible. Con estas ideas en mente, una interfaz de usuario queda definida como las partes físicas de un ordenador, así como su software, que una persona/usuario podrá ver, escuchar, comprender e interactuar y dirigir. Una interfaz de usuario está formada por dos componentes: Componentes de entrada y componentes de salida. La entrada se define como la forma en la que una persona comunica sus necesidades y deseos a la máquina. Algunos componentes de entrada típicos de cualquier ordenador hoy en día son el teclado, el ratón, y en casos de dispositivos táctiles, la propia mano. La salida se define como la forma en la que el sistema ofrece al usuario los resultados de sus cálculos y operaciones realizadas, conforme a las peticiones del usuario. El mecanismo básico de salida de todo ordenador hoy en día es el monitor, seguido de los dispositivos que aprovechan las capacidades auditivas del usuario: voz y sonido. Así pues, una buena interfaz de usuario deberá proveer de buenos mecanismos de entrada y salida de forma que se puedan satisfacer todas las necesidades del usuario, tomando en cuenta sus capacidades y limitaciones, de la forma más eficaz posible. Una gran interfaz será aquello que haga que el usuario acabe olvidándose de ella, consiguiendo así que el usuario se pueda centrar en la tarea a realizar en vez de centrarse en los mecanismos que le presentan la información o en cómo realizar la tarea.

Es muy importante comprender la importancia de las interfaces de usuario. Primero, para muchos usuarios la propia interfaz es el sistema, ya que se trata del único aspecto visible y tangible de la aplicación con la que se trabaja. Segundo, porque es la única forma en la que una aplicación muestra los resultados obtenidos al usuario, resultados que, para muchas organizaciones, son de vital importancia para la productividad de dicha organización. Una mala interfaz podría llegar a causar, en casos extremos, comprometer la seguridad de sus usuarios o del público en general (Aplicación de un hospital, o de una planta de energía).

2.2.- Dispositivos de control de interfaces naturales

Entendemos por Interfaz de Usuario Natural a aquella Interfaz de Usuario convencional que, cumpliendo con todos los aspectos y funcionalidades típicas de una interfaz de usuario convencional, ofrece una interacción humano-computador basado en elementos y/o movimientos naturales del ser humano. La palabra natural es utilizada porque la gran mayoría de interfaces utilizan dispositivos de control artificiales, como pueden ser un teclado o un ratón; dispositivos que requieren por parte del usuario realizar un aprendizaje previo en el funcionamiento de los mismos. Por tanto, entendemos por natural al término u objetivo que se debe alcanzar con el desarrollo de la interfaz de usuario para que éste considere el control del dispositivo como algo natural, que se pueda aprender su funcionamiento de forma intuitiva.

Entre los diversos tipos de dispositivos y métodos que existen para la interacción de forma natural con el usuario, se van a destacar algunos de ellos:

- **Wii Remote o Wiimote:** Dispositivo de entrada principal de la consola Nintedo Wii y funciona como mando de control tanto para la propia consola como para sus juegos [DIAZ 11]. Se comunica con la consola con un dispositivo Bluetooth. El Wiimote dispone de un pad de direcciones o cruceta, así como de nueve botones.
  La principal característica de este dispositivo es su capacidad de captura de movimientos. Gracias a un sensor de tipo acelerómetro es capaz de detectar movimiento en cualquiera de los tres ejes de coordenadas. Además posee una pequeña cámara de infrarrojos con la que puede rastrear hasta cuatro objetos en movimiento. Con esta cámara y la ayuda de una barra de sensores que posee la propia consola puede obtener información sobre la posición apuntada por el mando. Existen librerías no oficiales para realizar aplicaciones utilizando este dispositivo sobre un ordenador.
- **Interfaz multi-táctil:** Una interfaz multi-táctil hace referencia a aquellas superficies capaces de detectar la presencia de dos o más puntos de contacto sobre la misma. Esta tecnología se utiliza hoy en día en la gran mayoría de dispositivos móviles o smartphones, así como las populares tablets, que han ganado un gran peso en el mercado electrónico en los últimos años. Como ejemplos de esta tecnología destacamos dos de los productos más importantes de la empresa Apple, el iPhone [APPLE 01] y el iPad [APPLE 02].
- **Playstation MOVE:** Dispositivo que desarrolló la empresa Sony para Playstation 3. De funcionamiento similar al del Wiimote. Se comunica con la videoconsola con un dispositivo Bluetooth y en su interior habitan diversos sensores. De estos sensores, se pueden destacar un sensor acelerómetro de tres direcciones, dos sensores de inercia y un magnetómetro interno que sirve para calibrar la orientación del aparato respecto al campo magnético de la tierra. De esta forma, se podrá realizar el seguimiento del sensor en todo momento. Incluidos los casos en los que la cámara sea incapaz de determinar la posición del PSMove, como por ejemplo cuando se encuentra detrás del usuario [PSMOVE 10].

Se desconoce si existe alguna librería no oficial en la actualidad que permita desarrollar algún tipo de aplicación con este dispositivo.

- **Microsoft Kinect:** El dispositivo de interfaz natural desarrollado para la consola Xbox360 por Microsoft [KINE 12]. Poco después de sacar este dispositivo a la venta, Microsoft decidió liberar una API de programación básica para permitir el desarrollo de aplicaciones para ordenadores utilizando Kinect. Todo lo relativo a las especificaciones técnicas así como a una de las librerías de desarrollo será tratado con detalle en un capítulo posterior.

2.3.- Gráficos por Computador: Grafos de Escena

Dentro del ámbito de los gráficos por computador, entendemos por grafos de escena a la herramienta conceptual utilizada para representar un entorno virtual en 3D en aplicaciones gráficas. Un grafo de escena es una estructura jerárquica que contiene diversos tipos de nodos conectados por enlaces. Cada nodo se encarga de gestionar los datos relativos a esa escena y los diferentes sub nodos conectados a éste describen la relación que existe entre ellos.

Podemos dividir los nodos de un grafo de escena en tres tipos, a saber: el nodo raíz, los nodos intermedios o nodos grupo, y los nodos hoja que se localizan en la parte más profunda del árbol jerárquico del grafo [WOOLFORD 03]. El nodo raíz representa el nodo inicial, al cual se conectan el resto de nodos de la estructura de forma directa o indirecta. A nivel interno, un nodo puede poseer todo tipo de atributos, siendo los más comunes los atributos relativos a las diversas transformaciones 3D que pueden afectar a un objeto de una escena 3D, como pueden ser: rotaciones, translaciones o escalados. Los nodos describen tanto la posición como la orientación del objeto, así como el estado del entorno 3D. Los nodos hoja normalmente contendrán información de carácter geométrico del objeto representado [Figura 2.3].

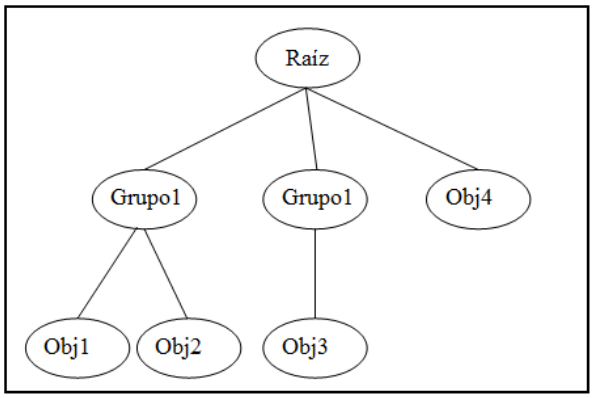

Figura 2.3 – Ejemplo de un Grafo de Escena

Los grafos de escena son pues una potente herramienta gráfica de alto nivel que evita los procedimientos a bajo nivel de otras librerías de gráficos, como pueden ser OpenGL y DirectX. Se tratan pues de estructuras de almacenamiento de datos que se crean y gestionan con un lenguaje de modelado de grafos de escena. Es este modelador el que se encargará de almacenar y actualizar la información necesaria y que interaccionará con la librería de bajo nivel deseada (Por ejemplo, OpenGL) para renderizar la escena. De esta forma todo el proceso

de renderizado queda oculto de forma abstracta al grafo de escena, liberando de esa tarea al diseñador de la escena 3D.

Una de las propiedades más interesantes de los grafos de escena es lo que se denomina estados heredados. Esta propiedad indica que cada nodo hijo heredará el estado del nodo padre. Cualquier transformación que apliquemos a un nodo padre, también se propagará a los nodos inferiores. Esta propiedad por tanto es una herramienta muy útil a la hora de transformar un grupo de elementos correctamente agrupados dentro del entorno 3D. En otras palabras, un grafo de escena es una estructura de almacenamiento de datos y propiedades que serán interpretados a nivel de renderizado por una librería de bajo nivel.

Otro aspecto a destacar de los grafos de escena son los volúmenes de inclusión, que se calculan y almacenan para cada nodo del grafo. Un volumen de inclusión es una estructura que encapsula en su interior una o varias geometrías de diversos objetos definidos en los nodos hoja. Estos volúmenes son utilizados para mejorar la eficacia en el renderizado de una escena 3D gracias a la operación de "frustrum-culling", esta operación consiste en realizar un descarte inteligente de los objetos quedan fuera del frustrum de visión, y que por tanto, no deben de renderizarse; de esta forma, si un nodo grupo no pasa el test de culling, todo el árbol puede descartarse de forma segura, ya que ninguno de sus hijos será visible en el volumen de visión actual. Otra gran ventaja de los volúmenes aparece a la hora de realizar las detecciones de colisiones. De forma análoga a la operación de culling, si el volumen de un nodo padre no colisiona con el objeto que deseamos analizar, se podrá descartar todo el árbol, ya que ninguno de sus ojos colisionará con dicho objeto. Los volúmenes de inclusión están formados típicamente por cajas, esferas y elipsoides.

Capítulo 3

ESTADO DEL ARTE

El estudio y aplicación de las interfaces naturales para el control de ordenadores u otros dispositivos tecnológicos ha dado pie a un gran número de artículos y estudios de carácter científico. Únicamente el sensor Kinect de Microsoft ha dado pie a un buen número de publicaciones. A continuación se citarán algunos de ellos, explicando brevemente en que ámbitos o estudios se centran. Como se podrá apreciar, el ámbito de las interfaces de usuario naturales es un tema que lleva relativamente poco tiempo tratándose y que dado su potencial, aun tiene mucho que ofrecer y le queda mucho camino por recorrer.

Dave et al. [DAVE 12] presentaron diversos análisis sobre la utilización de varios elementos para generar interfaces naturales con las que interaccionar con el ordenador. Entre estos elementos nos encontramos con el uso de unos guantes con marcadores de colores y de un sensor de profundidad. Se ha buscado realizar una simulación de todas las operaciones que se pueden realizar con un ratón convencional en un sistema operativo. El estudio presenta tanto las aproximaciones a los métodos de implementación del sistema, así como de diversas tablas con los resultados obtenidos en las diversas pruebas realizadas con usuarios en un entorno real. Un artículo publicado en febrero de 2012 presenta de una forma muy genérica el sensor de Microsoft Kinect, así como las múltiples oportunidades que puede brindar tanto al mundo del entretenimiento como a la comunidad científica [ZHANG 12]. El artículo se centra en las funcionalidades de Kinect, así como en el impacto que acabará generando más allá de la industria del videojuego.

Otro artículo publicado en julio de 2012 [CLARK 12] presenta un estudio en el que se utiliza Microsoft Kinect para analizar y realizar un seguimiento sobre las posturas y movimientos que realizan los usuarios para colocarse en diversas posiciones o realizar varias acciones motrices [Figura 3.1].

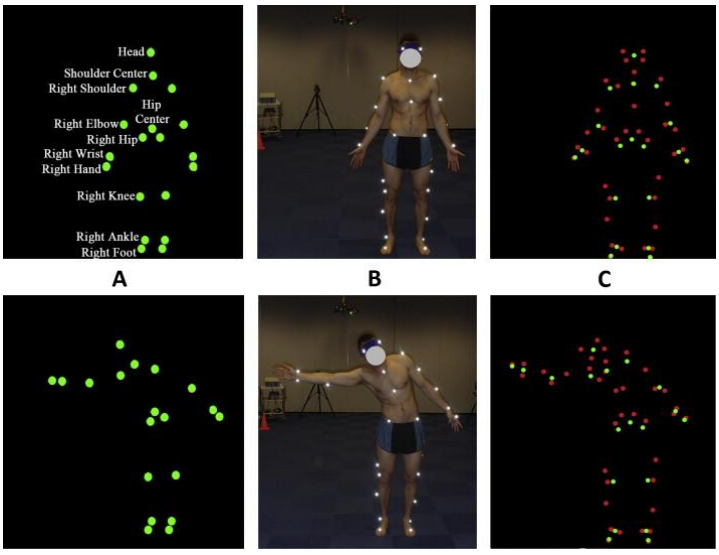

Figura 3.1 – Captura de Movimiento del estudio de control de postura

El estudio analiza los datos obtenidos con Kinect frente a una librería de conocimiento o benchmark para comprobar si dichos datos son suficientemente robustos, precisos y válidos.

A raíz de los resultados obtenidos, el estudio determina que, efectivamente, el dispositivo Kinect puede llegar a utilizarse para analizar de forma eficaz las posiciones y posturas que toman las personas analizadas. Destacar que este estudio motivar otros estudios que pueden servir para detectar anomalías de carácter motriz en el cuerpo humano, y generar así potentes herramientas de apoyo al diagnóstico para la medicina.

De carácter médico, se puede encontrar otro artículo publicado en noviembre de 2011 que trata sobre un estudio en el que se han realizado varias pruebas de carácter rehabilitador a varios sujetos con problemas motrices utilizando una interfaz natural basada en Kinect [CHANG 11]. Los autores presentan un estudio en el que demuestran que el uso de interfaces naturales para ayudar a la rehabilitación de dos personas con problemas de carácter motriz aumenta notablemente frente a otros métodos de rehabilitación convencionales [Figura 3.2]. Este tipo de aplicación demuestra ser muy interesante debido a las connotaciones que acarrea un aumento en dicha rehabilitación. Una mejora tan notable en dicha rehabilitación desemboca en un aumento del optimismo y la moral del paciente, tan necesario en casos extremos como éstos.

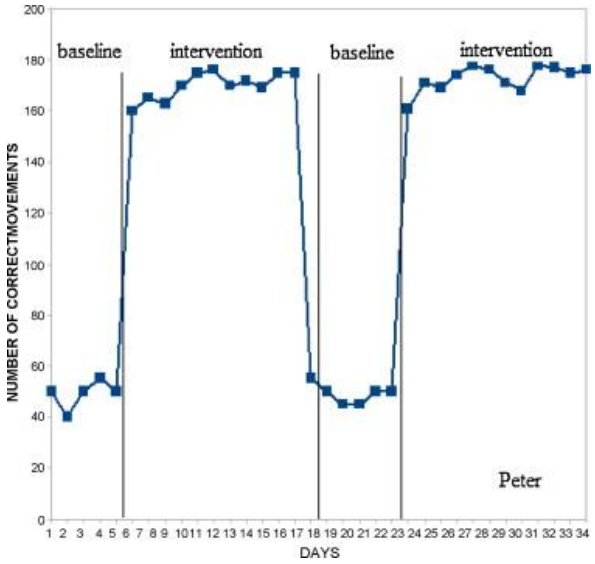

Figura 3.2 – Resultados obtenidos en el estudio de rehabilitación

De carácter más académico, se puede mencionar dos trabajos en concreto, realizados en la Universidad Politécnica de Valencia, presentados como proyectos de final de carrera.

Uno de ellos, publicado en abril de 2012, se centra en el desarrollo de una librería Wrapper para Kinect, con el fin de poder ser utilizada en un futuro para facilitar el acceso a los diferentes sensores internos de Kinect [PALERO 12]. Además, también se implementa una aplicación para realizar el calibrado de las cámaras de RGB e infrarrojos que posee Kinect. La aplicación está basada en un algoritmo de etiquetado semiautomático de las marcas de los patrones de calibración utilizados en el proyecto.

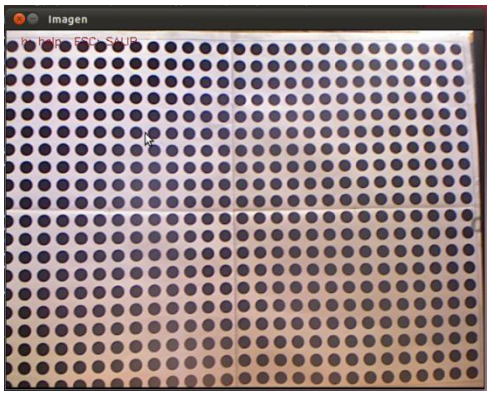

Figura 3.3 – Patrón de calibración para la aplicación de calibrado

El segundo trabajo, publicado en septiembre de 2011, se presenta como una guía de introducción al desarrollo de aplicaciones para Nintendo Wii [DIAZ 11]. La guía presenta una explicación detallada del uso de varias librerías y herramientas, pertenecientes en su mayoría a kits de desarrollo no oficiales, que permiten el desarrollo de aplicaciones que hagan uso de las funcionalidades básicas de la consola [Figura 3.4]. Estas funcionalidades pueden ser los gráficos, los sonidos y la interacción con los dispositivos de control que ofrece la consola (mando). También se incluyen en el proyecto diversas aplicaciones sencillas que sirven como ejemplo ilustrativo para las múltiples explicaciones que ofrece la guía, así como un listado de los componentes necesarios para poder ejecutarlos.

Figura 3.4 – Programa de ejemplo para la guía de desarrollo de Wii

Como se puede observar, ya se han realizado proyectos y estudios de diversas características que giran en torno a las interfaces naturales, así como de algunos de los diversos dispositivos que permiten aplicarlas. Además, en estos estudios queda demostrado el gran potencial que ofrecen estas interfaces, dejando abiertas las puertas a la investigación y desarrollo de todo tipo de aplicaciones. Aplicaciones que pueden llegar a poseer temáticas muy variadas, que van desde el desarrollo de programas de carácter doméstico para el entretenimiento, hasta aplicaciones de apoyo al diagnóstico, de carácter más formal y científico.

Capítulo 4

HERRAMIENTAS A UTILIZAR

Antes de pasar a la explicación del desarrollo de la aplicación, será necesario indicar y detallar qué herramientas, aplicaciones y librerías, así como el funcionamiento y utilización de las mismas, han sido utilizadas para llevar a cabo la aplicación a presentar.

4.1.- Blender

Blender es una de las aplicaciones Open Source de gráficos 3D más populares del mundo [BLEND 01]. Esta aplicación ofrece un amplio espectro de funcionalidades y herramientas de cara al modelado, texturizado, iluminación, animación y post-procesado de video en un único paquete. Por medio de su arquitectura abierta, ofrece una interoperabilidad entre plataformas, extensibilidad y un flujo de trabajo altamente integrado, estando pues destinada a nivel mundial a todo tipo de profesionales y artistas.

La primera aparición de Blender como producto utilizable fue en Agosto de 1994, integrando una serie de herramientas para la creación de una amplia gama de contenidos 2D y 3D. Fue desarrollado originalmente por la compañía 'Not A Number' (NaN). Años más tarde en el 2002, después de la desaparición de dicha empresa, Ton Roosendaal, fundador y desarrollador jefe de Blender así como de la desaparecida empresa NaN, creó la compañía no lucrativa Blender Foundation. Esta empresa nació con el objetivo de continuar el desarrollo y la promoción de Blender, convirtiéndola en una aplicación de código abierto en octubre de 2002. A día de hoy, el desarrollo y ampliación de Blender sigue llevándose a cabo, gracias a un equipo de voluntarios de procedentes de diversas partes del mundo y liderados por el propio creador de Blender.

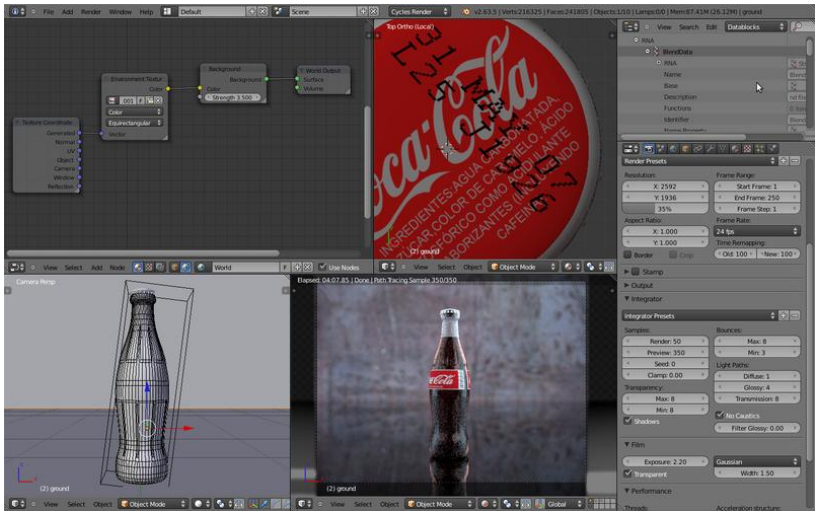

Figura 4.1.1 - Interfaz Gráfica de Blender

Entre sus diferentes utilidades, se puede destacar la creación de visualizaciones 3D, tanto de imágenes estáticas como videos de alta calidad [Figura 4.1.1], mientras que, al tener incorporado un motor 3D propio en tiempo real, permite además la creación de contenido interactivo que podrá ser exportado y/o reproducido de manera independiente[Figura 4.1.2].

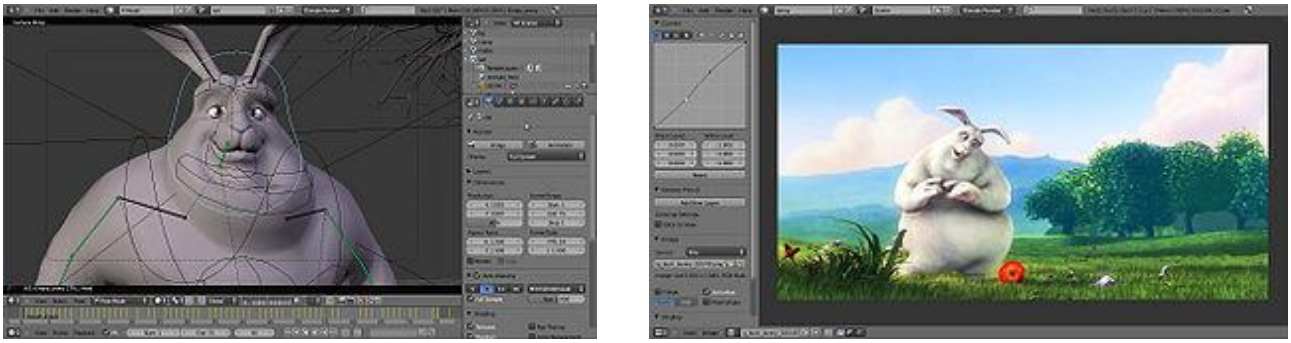

Figura 4.1.2 - Diversas funciones y ejemplos de Blender

Éstas son algunas de las características más destacadas de Blender [BLEND 02]:

- Multiplataforma, libre, gratuito y con un tamaño de origen realmente pequeño comparado con otras plataformas y paquetes 3D, estando disponible en varios sistemas operativos (Windows, MAC, Linux).
- Capacidad para una gran variedad de primitivas geométricas, incluyendo curvas, mallas poligonales, vaciados, distintos tipos de splines, fluidos…
- Junto a las herramientas de animación se pueden encontrar deformaciones por cuadrícula o armadura, vértices de carga y partículas, tanto estáticas como dinámicas.
- Edición de audio y sincronización de vídeo.
- Características interactivas, como pueden ser la detección de colisiones, recreaciones dinámicas y lógicas, enfocadas al diseño de videojuegos.
- Posibilidades de renderizado interno versátil y exportación con potentes trazadores de rayos o “raytracer” Open Source, como pueden ser kerkythea o YafaRay [KERKY 12] [YAFARAY 12].
- Automatización y control de tareas con la ayuda del lenguaje de programación Python.
- Aceptar diversos formatos gráficos como JPG, TGA o TIFF.
- Motor de juegos 3D integrado, con un sistema de lógica que utiliza el lenguaje de programación Python.
- Simulaciones dinámicas para cuerpos maleables, partículas y fluidos.
- Modificadores apilables, que permiten aplicar transformaciones no destructivas sobre mallas.
- Sistema de partículas estáticas para simular cabellos y pelajes, así como shaders para lograr texturas realistas.
- Capacidad para realizar Match Moving, o Motion Tracking, muy utilizado en películas de última generación.

4.2.- Sensor Microsoft Kinect

El sensor Kinect fue presentado por Microsoft como un "controlador de juego libre y entretenimiento" desarrollado originalmente para la videoconsola Xbox360 en el año 2010. Si bien a partir del año 2011 también estaba disponible para PC a través de Windows 7 y Windows 8. El objetivo principal y fundamental de Kinect es el de permitir a los usuarios controlar e interactuar con la consola sin necesidad de un controlador de videojuegos convencional o de cualquier contacto físico con el dispositivo. Eso se consigue mediante una interfaz natural de usuario que es capaz de reconocer una gran cantidad de gestos, comandos de voz, objetos e imágenes [Figura 4.2]. El objetivo principal de dicha interfaz natural era conseguir aumentar el uso de la propia videoconsola Xbox360. En un apartado posterior se hablará de su interacción y utilización en PC con más detalle.

Figura 4.2 - Sensor Microsoft Kinect

En cuanto al dispositivo en sí [KINEDOC 12], consiste en una barra horizontal de aproximadamente 23 centímetros conectada a una pequeña base circular con un eje de articulación de tipo rótula. El dispositivo final está diseñado para ser colocado longitudinalmente por encima o por debajo de la pantalla de visualización, ya sea un monitor o un televisor convencional.

El dispositivo cuenta, a nivel de hardware, con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador propio personalizado que se encarga de ejecutar el software patentado, que proporciona captura de movimiento de todo el cuerpo en 3D en tiempo real, reconocimiento facial y capacidades de reconocimiento de voz. El micrófono de matrices del sensor puede permitir llevar a cabo la localización de la fuente acústica y la supresión del ruido ambiente.

El sensor de profundidad es un proyector de infrarrojos con un sensor de tipo CMOS monocromo que permite a Kinect ver la habitación en la que se encuentre en 3D bajo cualquier condición de luz ambiental. El rango de detección de la profundidad del sensor es ajustable gracias al software interno de Kinect que es capaz de calibrar automáticamente el sensor, basado en la jugabilidad y en el ambiente físico del jugador, tal como puede ser la presencia de otros objetos, como por ejemplo sillas y sofás.

En cuanto a la utilización en PC del sensor, fue la propia Microsoft quien dejo de forma gratuita y a disposición de todo el mundo los drivers para el control de Kinect para Windows. Además, también ha ofrecido de forma gratuita una librería SDK oficial con la que poder interaccionar y desarrollar todo tipo de aplicaciones utilizando dicho dispositivo. Además de las APIs y drivers oficiales, también existen otras opciones basadas en código Open Source, entre las que se destaca PrimeSense y OpenNI, de los cuales se habla a continuación.

4.3.- Open Natural Interaction - OpenNI

La interfaz OpenNI consiste en una interfaz estándar y gratuita diseñada para controlar diversos tipos de algoritmos de procesamiento de datos recibidos por un sensor 3D [OPENNI 01]. La finalidad principal es la de definir una serie de tipos de datos (mapas de profundidad, mapas de colores RGB, poses del usuario…) y de una interfaz encargada de generar todos estos datos a través de diversas herramientas y métodos (el propio sensor, algoritmos de detección de esqueletos…) para que cualquier tercero pueda trabajar con ellos.

Con esta premisa en mente, los desarrolladores de juegos y/o aplicaciones podrán desarrollar sus aplicaciones independientemente del modelo y tipo de sensor 3D utilizado [OPENNI 02]. También es posible desarrollar todo tipo de algoritmo para el procesado de los datos proporcionados por el sensor directamente, sin tener que preocuparse del modelo y tipo de sensor. Así pues, una aplicación podrá tener acceso, por ejemplo, a la posición de una mano detectada por el sensor, sin necesidad de saber nada más de la obtención de dicha información a nivel interno. Además OpenNI también proporciona soporte total a la retro compatibilidad, asegurando que cualquier versión de drivers desarrollados para versiones anteriores seguirá funcionando sin ningún problema en las nuevas versiones.

Actualmente OpenNI permite la captura de movimiento en tiempo real, el reconocimiento de gestos con las manos, el uso de comandos de voz y la utilización de un analizador de escena que detecta y distingue las figuras situadas en un primer plano de las situadas en plano secundario o de fondo.

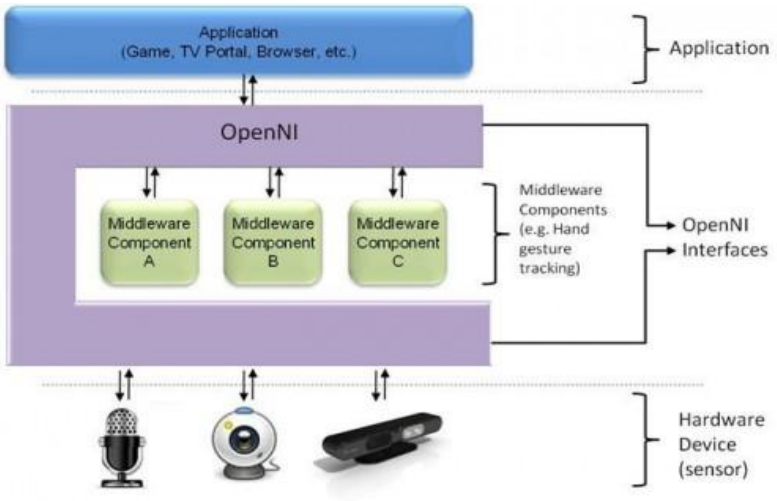

Figura 4.3 - Esquema detallado de la interfaz OpenNI

Aunque realmente formen parte del conjunto de OpenNI, es necesario destacar que tanto los algoritmos de procesamiento de datos, o Middleware, como los drivers del hardware de PrimeSense para el sensor 3D (Kinect en esta caso concreto) deben ser instalados de forma independiente a la interfaz de OpenNI para asegurar el correcto funcionamiento de toda la plataforma [Figura 4.3].

También es interesante destacar que OpenNI ofrece una retro compatibilidad total, asegurando que cualquier aplicación desarrollada con versiones antiguas de OpenNi funcionará sin ningún problema en las nuevas versiones.

Funcionamiento

En cuanto al modo de funcionamiento propiamente dicho de OpenNi, ésta se encuentra dividida en diversas capas de carácter abstracto que proveen soporte tanto a los componentes de tipo Hardware como los de tipo Software. Estos componentes son detectados por la propia interfaz como módulos, y son los encargados de producir y procesar los datos enviados por el sensor.

Los módulos de tipo Hardware que actualmente soporta OpenNI son los siguientes:

- *Sensor 3D*
- *Cámara RGB*
- *Cámara de Infrarrojos*
- *Dispositivo/s de audio*

Respecto a los módulos de tipo Software tenemos:

- *Algoritmos de detección y análisis de cuerpo completo:* Componente encargado de detectar y generar toda la información relativa a la posición y orientación del cuerpo (posición, dirección, centro de masas…).
- *Algoritmos de detección y análisis de manos:* Componente que procesa la información sensorial y proporciona la posición y otras características de las manos, en caso de detectarlas.
- *Algoritmos de detección de gestos:* Software capaz de detectar algunos gestos predefinidos, como por ejemplo saludar con la mano, y de producir alertas para la aplicación.
- *Algoritmos de análisis de escenas:* Software que se encarga de analizar una imagen de la escena con la finalidad de generar diferentes tipos de información. Entre los tipos de información generada, podemos encontrar información cosas tales como la separación de los diversos objetos detectados del fondo de la escena, las coordenadas del plano del suelo, o la identificación por separado de los diferentes componentes de la escena.

A la hora de generar los datos que solicita la aplicación, OpenNI utiliza lo que se denominan cadenas de producción. Una cadena de producción está compuesta por todos los nodos de producción que intervienen en el proceso de obtención de datos útiles: desde la obtención de los datos base del sensor correspondiente, hasta su gestión y devolución a la aplicación, pasando por la interpretación de los mismos. Algunas de estas tareas pueden llegar a ser muy complejas. Por ejemplo, a la hora de obtener información 3D del entorno, de algunos sensores obtendremos como información poco más que un mapa de profundidad, en

el que cada pixel representa la distancia de éste con el sensor. Recae pues en los nodos de producción la tarea de interpretar y transformar estos datos en información que sea de utilidad para la aplicación. Estos nodos se pueden dividir en dos tipos: Nodos a nivel de sensor y nodos a nivel de middleware o intermedios, dividiéndose en diferentes sub tipos cada uno de ellos.

Nodos a nivel de sensor

Estos nodos se encargar de recoger la información generada por el sensor directamente. Entre ellos tenemos los siguientes tipos:

- *Dispositivo:* Este nodo representa al dispositivo en sí. Su principal función es la de controlar la configuración del dispositivo
- *Generador de profundidad:* El siguiente nodo genera el mapa de profundidad. Este nodo debe poder ser generado por cualquier dispositivo, ya que es el nodo principal sobre el que trabaja OpenNI.
- *Generador de imagen:* Este nodo genera los mapas de imagen a color. Este nodo deberá ser generado por cualquier sensor de color para que pueda funcionar sobre OpenNI.
- *Generador de infrarrojos:* Nodo que se encarga de generar el mapeado de las imágenes basadas en infrarrojos. Como los demás generadores, este nodo deberá ser generado por cualquier sensor de infrarrojos de cara a que pueda funcionar sobre la librería.
- *Generador de audio:* Encargado de generar los streams de audio. Al igual que el resto de generadores, deberá ser generado por el dispositivo de audio del sensor para poder ser utilizado por los niveles superiores.

Nodos a nivel intermedio o Middleware

En este nivel se agrupan los nodos que interpretan, total o parcialmente, la información obtenida por los nodos a nivel de sensor. Estos son algunos de sus tipos:

- *Generador de alertas de gesto:* Crea llamadas al sistema o CallBacks para la aplicación con el identificador del gesto detectado, en caso de detectarse alguno.
- *Analizador de escena:* Analiza diversos aspectos de la escena. Algunos de estos aspectos incluyen elementos como: diferenciación del plano más cercano detectado del fondo, identificación de las figuras encontradas en la escena, o la detección del plano correspondiente al suelo.
  La salida que genera el analizador es un mapa de profundidad etiquetado, en el que cada pixel mantiene una etiqueta que indica que representa, si una figura concreta o el fondo.

- *Generador de puntos de mano:* Nodo que soporta tanto la detección como el seguimiento de una mano. Este nodo generará llamadas o CallBacks que proveen alertas cuando una mano ha sido detectada, o cuando una mano sobre la que se está realizando un seguimiento varía su posición y/o orientación.
- *Generador de usuario:* Crea una representación de un cuerpo (total o parcial) que se haya detectado en la escena 3D.

Lectura y escritura de datos

Otro aspecto importante a tener en cuenta gira en torno a la frecuencia a la que se leen y generan los datos por los nodos de producción. A la hora de leer los datos del sensor, aunque el sensor esté constantemente generando información, puede darse el caso en que a la hora de actualizar la información, un nodo de producción esté trabajando todavía con la información obtenida anteriormente. Para evitar este problema, OpenNI no actualizará nada hasta que se le pida explícitamente que actualice los datos con la última información actualizada. OpenNI ofrece una serie de llamadas que permiten controlar la actualización total o parcial o dichos datos. Para actualizar los datos, podemos encontrarnos con un problema similar: Es posible que la aplicación no necesite o no pueda procesar la información ofrecida al mismo ritmo que la genera OpenNI y/o el sensor, o las variables para almacenar dichos datos no estén disponibles. Para solucionar este problema, OpenNI no empezará a generar datos hasta que la aplicación se lo pida. De forma análoga, la aplicación puede solicitar a OpenNI que deje de generar dichos datos.

Gestión de errores

Todos los nodos de producción de OpenNI pueden generar un estado de error. Por defecto, el estado de error será siempre OK, y en el caso de que algún nodo falle, cambiará este estado por el del error correspondiente. Un nodo que no tenga implementada esta funcionalidad siempre mantendrá el estado de error como OK.

Una aplicación puede comprobar en todo momento el estado de error, aunque en muchos casos únicamente necesitará saber si se ha producido un error, restándole importancia al tipo de error. Para poder conseguir esta información, la aplicación deberá configurar una llamada o CallBack que la alerte de cualquier cambio producido en el estado de error. OpenNI unifica todos los errores en único estado, para poder facilitar a las aplicaciones la tarea de detección de errores. En caso de que varios nodos hayan fallado, se avisará con un estado concreto avisando de que varios errores se han producido a la vez.

4.4.- Open Scene Graph - OSG

La librería OpenSceneGraph (OSG) consiste en una agrupación de herramientas multiplataforma y de libre distribución para el desarrollo de gráficos de alto rendimiento para todo tipo de aplicaciones como pueden simuladores de vuelo, videojuegos, realidad virtual y aumentada o visualización avanzada científica [OSG 01]. La librería funciona en torno al concepto de los llamados grafos de escena, proveyendo una interfaz orientada a objetos creada sobre la librería gráfica OpenGL. De esta forma se consigue liberar al desarrollador de la implementación y optimización de gráficos a bajo nivel además de proveer muchas utilidades adicionales para conseguir desarrollar aplicaciones gráficas de forma rápida y sencilla.

El objetivo central de la librería es la de conseguir poner a disposición de todos los desarrolladores las ventajas y facilidades que ofrece la tecnología de los grafos de escena, tanto para la realización de aplicaciones comerciales como de libre distribución. Escrita totalmente en el lenguaje de programación C++ y OpenGL.

Los aspectos clave de OSG son su rendimiento, escalabilidad, portabilidad y el aumento de productividad asociado al trabajo con librerías de alto nivel [OSG 02]:

- *Rendimiento:* Es capaz de soportar a nivel nativo todas las estructuras y ventajas de OpenGL, entre las que podemos listar: view-frustum culling, occlusion culling, small feature culling, nodos de nivel de detalle (LOD), vertex arrays, vertex buffer objects, lenguaje de programación de Shaders LSGL y Array Lists. OSG también permite una fácil personalización de los procesos de trazado y dibujado de primitivas.
- *Productividad:* El núcleo de la librería encapsula la mayoría de las funcionalidades de OpenGL, incluyendo las últimas extensiones. También incluye diversas optimizaciones de renderizado como la ordenación de objetos o el culling, y una gran cantidad de librerías a modo de add-ons que hacen posible el desarrollo de aplicaciones de alto rendimiento de forma rápida y eficaz. De esta forma, el desarrollador de la aplicación puede centrarse en el contenido y en cómo gestionarlo, olvidándose así de la codificación de elementos a bajo nivel.
- *Cargador de ficheros y bases de datos:* Una de las librerías extra que se ofrece, osgDB, se encarga de gestionar y cargar todo tipo de ficheros y bases de datos externa para posteriormente poder ser utilizados por la propia OSG. Entre los diferentes ficheros y bases de datos destacamos 3D Studio Max (.3ds) y Alias Wavefront (.obj). Además también es capaz de cargar todo tipo de imágenes, como por ejemplo .jpg, .gif o .bmp, así como formatos y fuentes de texto.
- *Escalabilidad:* OSG es capaz de ejecutarse en todo tipo de Hardware, desde ordenadores portátiles hasta ordenadores avanzados con soporte multi-nucleo y/o multi-threading, así como sistemas de clustering o multi-gpu. Esto es posible gracias a que el núcleo de OSG soporta múltiples contextos gráficos tanto para las Display Lists y objetos texturados de OpenGL como las operaciones de trazado y culling. Como OSG ha sido diseñado para cargar dichos datos desde la memoria cache, convierte prácticamente todas las operaciones de grafos de escena en operaciones de solo lectura de memoria. Esto permite paralelizar muchas de las tareas en múltiples CPUs que a su vez están ligadas a múltiples subsistemas. El soporte para múltiples contextos gráficos lo proporciona la librería osgViewer.

- *Portabilidad:* El núcleo de la librería ha sido diseñado de forma que posea el mínimo número de dependencias posible, requiriendo poco más que C++ y OpenGL. Esto ha permitido a OSG poder ser exportado a un amplio abanico de plataformas. Fue diseñado principalmente para IRIX, siendo exportado más adelante a LINUX y poco después a Windows, para a continuación verse en FreeBSD, MAC OSX, Solaris…
  La librería no depende de ningún tipo de subsistema de visualización/ventanas, lo que permite a los usuarios añadir sus librerías de visualización/ventanas de forma rápida. La librería osgViewer ofrece soporte nativo para Windows (Win32), Unicles (X11) y OSX (Carbon). Además, esta librería puede ser integrada de forma sencilla en otros toolkits de visualización, como Qt, GLUT, SDL, Cocoa…
- *Soporte multilenguaje:* Existen como proyectos de comunidad módulos que permiten exportar OSG a otros lenguajes de programación diferentes a C++, como pueden ser Java, LUA o Python.
- *Librerías extra o Node-Kits:* Los node kits son las librerías externas que se pueden cargar de forma separada, que pueden ser cargadas en tiempo de compilación o incluso cargarse en tiempo de ejecución. De las diversas librerías podemos destacar las siguientes:
  - osgParticle – Sistemas de partículas.
  - osgText – Para texto de alta calidad y con anti-aliasing.
  - osgFX – Interfaz para efectos especiales.
  - osgShadow – Interfaz para efectos de sombreado.
  - osgManipulator – Para controles interactivos en 3D.
  - osgSim – Efectos de simulación visual.
  - osgTerrain – Renderizado de entornos y terrenos.
  - osgAnimation – Animación de personajes y objetos rígidos.
  - osgVolume – Renderizado de alta calidad de volúmenes.

A continuación se mostrarán, de forma específica para nuestra aplicación, el funcionamiento y utilidad concreto de las clases y librerías de OSG que se han utilizado durante el desarrollo:

Clase osg::Vec3D

Clase de apoyo que contiene todo lo necesario para cargar y operar con vectores de tipo 3D. Entre sus métodos se incluyen todo tipo de operaciones tanto escalares como vectoriales, así como operaciones lógicas y comprobación de rango y similares.

Clase osg:Matrix

Al igual que con vectores, OSG también posee una clase interna para la gestión de matrices. Contiene los atributos y métodos necesarios para trabajar con matrices de forma cómoda y rápida.

Clase osg::Node

Clase base para gestionar todos los diferentes nodos que puede poseer el grafo de escena. Un puntero de tipo Node podrá asignarse a cualquier subclase que herede de ésta.

Clase osg::Group

Sobre una variable de este tipo asignaremos el resto de elementos que cargaremos en nuestro programa. Así pues, ésta es la clase base para todos los nodos internos con los elementos del grafo de escena [Figura 4.4.1].

Figura 4.4.1 - Fragmento del diagrama de Herencias de OSG

Clase osg::PositionAttitudeTransform

Una de las clases que se encarga de gestionar las diferentes transformaciones que aplicaremos a los elementos de nuestra escena. La utilizaremos junto a la clase Node para controlar toda la escena.

Librería osgAnimation

Esta librería se encarga de gestionar todas las clases relativas al control de nodos animados.

Clase osgUtil

Esta librería posee clases de apoyo para poder realizar diversos tipos de acciones. Para nuestro caso concreto, destacamos las clases de LineSegmentIntersector e IntersectionVisitor, que utilizaremos para, a partir de una coordenada especifica, detectar qué elementos interseccionan con una línea imaginaria que pase por dicha coordenada.

Clase osgViewer::Viewer

Clase encargada de gestionar toda la escena, desde los elementos que la componen, como sus transformaciones, como los diversos manejadores y controladores que gestionan toda la aplicación. Esta clase es la encargada de inicializar y mantener la correcta visualización de la escena.

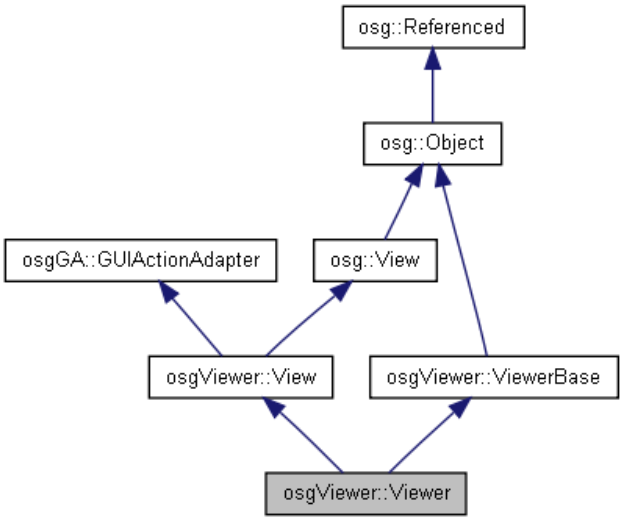

Figura 4.4.2 - Árbol de herencias de la clase Viewer

Sobre la clase Viewer cargaremos el resto de elementos y manejadores de toda la aplicación [Figura 4.4.2].

Capítulo 5

DESARROLLO

En este apartado se detallarán todos y cada uno de los pasos realizados para llevar a cabo el desarrollo que resultará en la aplicación objeto del trabajo.

5.1.- Modelado

Antes de poder empezar a programar el funcionamiento en sí de la aplicación, es necesario realizar el modelado de todos los componentes gráficos que formarán parte de ella. Así pues, el primer paso será modelar dichos componentes con una herramienta de diseño 3D. La herramienta elegida para tal propósito es Blender. Muy importante destacar que cada componente ha sido diseñada por separado del resto de componentes, para así facilitar su posterior exportación para poder utilizarla en nuestro programa.

Como imagen hemos utilizado una ilustración de la popular serie de dibujos animados "Hora de Aventuras" [Figura 5.1.1].

Figura 5.1.1 – Imagen de Hora de Aventuras utilizada

Con la ayuda de un editor de imágenes se han obtenido un par de máscaras con las que preparar las texturas para cada componente del puzle [Figura 5.1.2] [Figura 5.1.3].

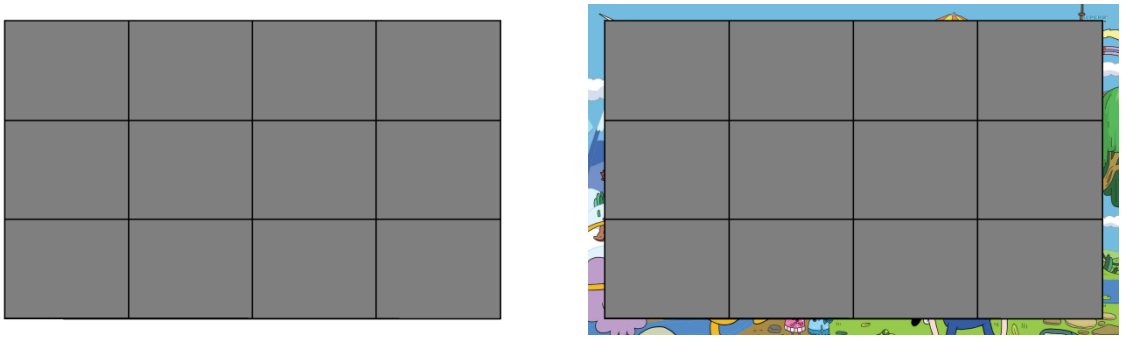

Figura 5.1.2 - Textura para el Tablero del puzzle

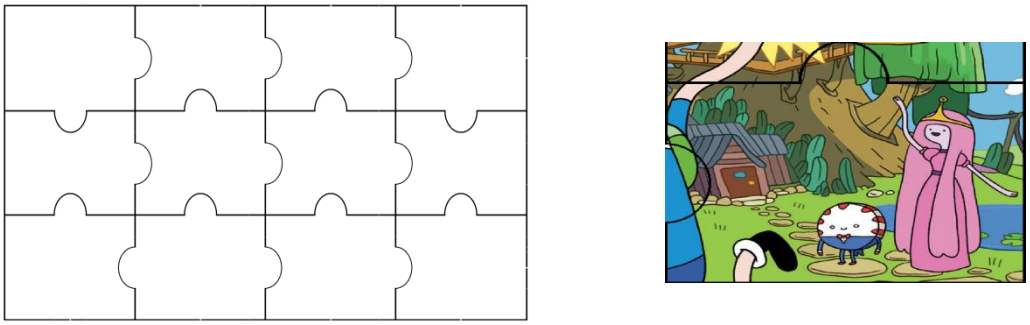

Figura 5.1.3 - Recortado y obtención de la textura para cada pieza

Como primer paso, ya dentro de la aplicación Blender, se generan las estructuras básicas para poder formar una de las piezas del puzzle.

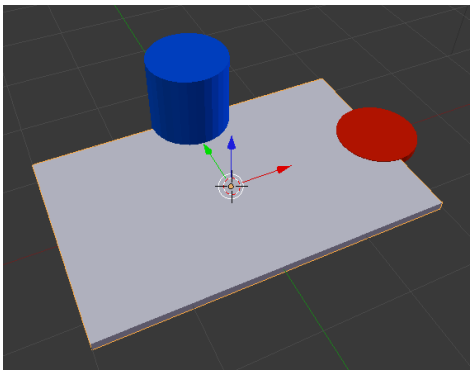

Figura 5.1.4 - Elementos utilizados para formar la pieza

Los componentes por separado que darán forma a la pieza final, en este caso particular, están formados por una primitiva de tipo Caja y dos primitivas de tipo Cilindro [Figura 5.1.4]. Como se puede observar uno de los cilindros tiene un tamaño mucho más grande que los otros dos componentes, ahora veremos a qué se debe esto.

Entre sus muchos módulos y funciones, Blender posee unas operaciones concretas llamadas Operadores Booleanos, que se encargan de realizar operaciones de tipo recortado, unión y colisión, entre muchas otras, de varias primitivas de la escena. En nuestro caso, vamos a realizar la operación unión sobre la caja y el primer cilindro para obtener la forma fusionada de las dos piezas, y la operación de substracción entre el segundo cilindro y el resultado de la operación anterior [Figura 5.1.5].

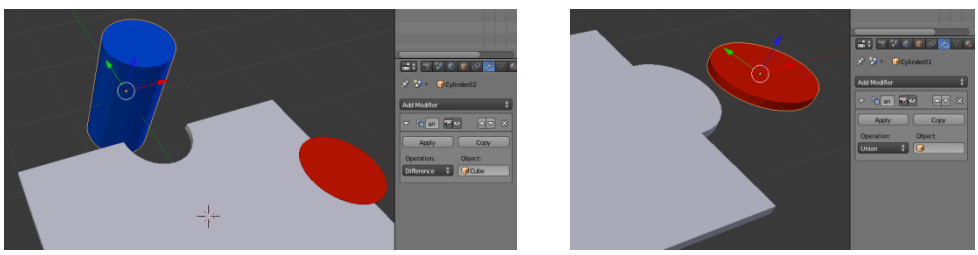

Figura 5.1.5 - Aplicación de las operaciones Booleanas

El resultado, como se puede observar, es una única pieza, que posee la forma deseada que se buscaba para formar la estructura de una de las piezas del puzzle [Figura 5.1.6].

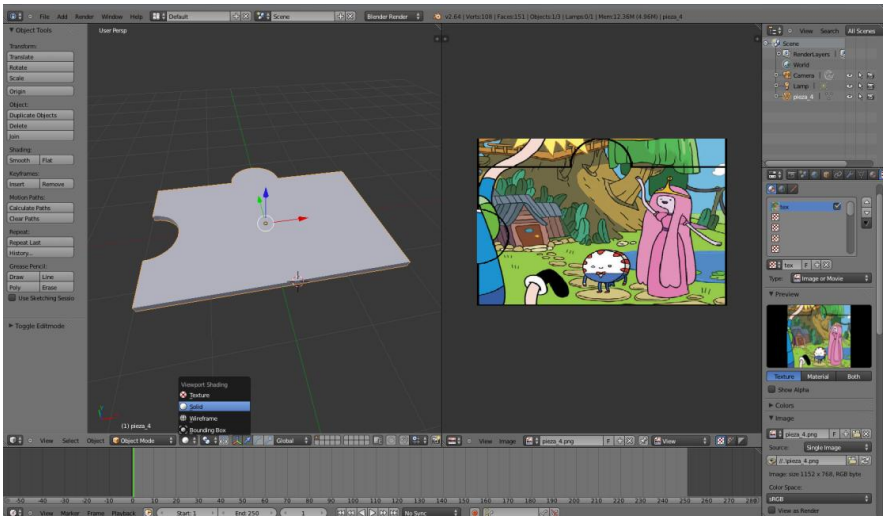

Figura 5.1.6 - Modelo sólido de la pieza nº4 con su futura textura

Antes de continuar con el próximo paso, indicar que, para que OSG pueda cargar tanto la figura exportada como su correspondiente textura, es necesario realizar unas cuantas operaciones extra en Blender. Esto se debe a que cada figura, aparte del modelo en sí, debe tener asociada en sus especificaciones tanto un tipo de material como una textura valida. En caso de no ser así, OSG no aplicará ni la iluminación ni la textura de forma correcta, generando unos resultados no deseados.

El siguiente paso consistirá en asociar la imagen cargada con la cara superior del sólido, para que tenga el aspecto final de pieza que andamos buscando. Entre las múltiples opciones que Blender nos ofrece, nos decantamos por utilizar el llamado UV Mapping. Esta operación consiste en realizar una proyección de los vértices seleccionados del sólido (toda la cara) sobre la imagen, para asignar a cada uno de estos vértices una coordenada de dicha imagen, para asociarla en la posición correspondiente en el sólido. Utilizamos para ello, dentro del menú de UV Mapping, la opción 'Project from View' obteniendo un mapeado bastante aproximado a lo que deseamos [Figura 5.1.7]. De forma manual, retocamos los puntos que creamos convenientes.

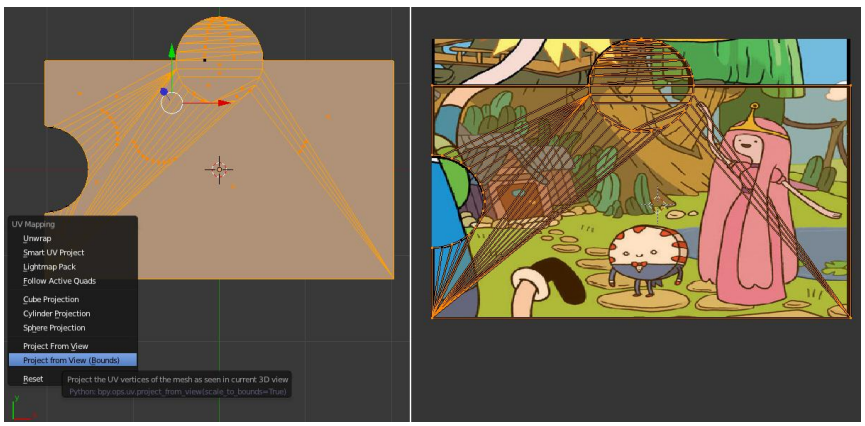

Figura 5.1.7 - Mapeado de los vértices de la cara

Ya para finalizar, exportaremos nuestra pieza a formato WaveFront (.obj) utilizando el exportador interno que nos proporciona Blender, obteniendo de esta forma el fichero que posteriormente cargaremos desde la librería OSG para trabajar con nuestros elementos creados con Blender [Figura 5.1.8]. Como resultado obtendremos un fichero que consiste en una lista de vértices, aristas y caras que conforma la figura exportada, así como toda la información correspondiente al tipo de material del objeto y todo el mapeado de la textura con sus correspondientes vértices.

Figura 5.1.8 - Obtención del fichero para exportar

El modelado del resto de piezas y del tablero del puzzle lo realizaremos de la misma forma, aplicando las operaciones y texturas correspondientes para cada elemento [Figura 5.1.9]. Tratando como caso concreto el tablero, hay que tener en cuenta que el tamaño de su superficie debe de ser lo suficientemente grande como para albergar dentro del mismo las doce piezas que conforman el puzzle completo, así como el borde del mismo.

Figura 5.1.9 - Resultado final de la cuarta pieza

Depurado y Etiquetado

Para el correcto funcionamiento de la aplicación, ha sido necesario realizar un giro de 90º grados sobre el eje X de coordenadas a todos los elementos que componen el puzzle, para dejarlos paralelos al eje Y de coordenadas. Con esto se consigue todos los elementos de nuestra escena queden correctamente situados en el espacio de coordenadas del mundo de nuestra aplicación. Aunque si bien se podía haber realizado este cambio en cualquier momento del desarrollo, nos hemos decantado en hacerlo antes de la exportación de los objetos en el propio Blender, para así aligerar el número de operaciones a realizar durante la ejecución del programa. De esta forma, los objetos pasan de estar definidos en el plano de coordenadas (x, y) al plano de coordenadas (x,z).

Otro cambio importante realizado se encuentra en el etiquetado de cada elemento que realiza Blender por defecto al realizar la operación de exportación.

Blender etiqueta los objetos usando una nomenclatura que no nos interesa en absoluto. Lo que se desea es poder identificar cada una de las piezas con un número entero y el tablero con un nombre identificativo, para poder manejar las etiquetas y realizar operaciones con ellas de forma sencilla desde C++. Así pues, se han modificado todos los ficheros exportados para reflejar las nuevas etiquetas, eliminando las que teníamos por defecto.

Como dato final, indicar que los ficheros para cada pieza han sido nombrados de la siguiente forma:

Pieza_NUM.obj

Donde NUM se corresponde con el número de la pieza. Esto se ha realizado así para poder cargar de forma rápida y cómoda más adelante estos elementos [Figura 5.1.10]. El nombre del fichero para el tablero no recibe ningún trato en concreto.

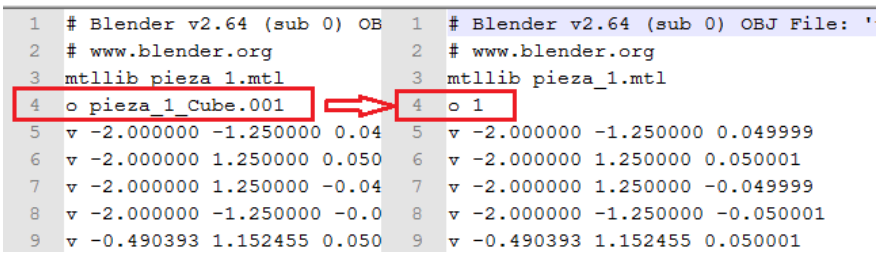

Figura 5.1.10 – Cambio manual al fichero objeto

5.2.- Librería Escena

Compuesta por los ficheros “Escena.h” y “Escena.cpp”, esta librería se encarga de gestionar todo lo relativo al apartado gráfico de la aplicación. Cosas como la carga de los modelos o el control de los desplazamientos y transformación de los mismos, es controlado aquí.

La librería construye una única clase llamada escena, que posee todos los atributos y métodos necesarios para gestionar y controlar los siguientes aspectos:

- Grupo principal de la escena: Almacenado en un atributo de tipo osg::Group, aquí se tendrá el nodo que pasaremos a una variable de tipo osgViewer para cargar la escena.
- Listado de piezas: Para gestionar las piezas necesitamos almacenar diversa información sobre ellas: Su correspondiente nodo, su PositionAttitudeTransform, su posición inicial fuera del tablero, su posición final dentro del tablero y si ya ha sido fijada en el tablero o no.

  En un principio, las piezas con toda su información asociada, se almacenan de forma ordenada, pero mediante la utilización de un método de apoyo extra, mezclamos de forma aleatoria todas ellas, para que a cada nueva ejecución de la aplicación éstas aparezcan en posiciones totalmente diferentes cada vez.
- Tablero: De forma similar a las piezas, almacenamos para el tablero tanto el nodo como el PositionAttitudeTransform del mismo.
- Cursor: Para la versión de nuestra aplicación que funciona sobre Kinect, también poseemos un nodo y PositionAttitudeTransform, así como un método y atributo extras de apoyo, que se encargan de gestionar el movimiento y animación de un cursor con forma de mano. Este cursor se moverá junto a nuestra mano derecha para saber en todo momento que estamos haciendo sobre la aplicación.

Como se ha podido apreciar, tanto las posiciones iniciales como finales de cada pieza han sido definidas de antemano y de forma interna [Figura 5.2.1]. De esta forma, se facilita enormemente la tarea de comprobar si una pieza se ha intentado colocar en una posición correcta o incorrecta. Para no ser muy restrictivos con la colocación de las piezas, se ha añadido un atributo interno al que llamamos offset, de forma que si la diferencia entre la posición correcta de la pieza y la posición en la que se ha detectado que se quiere soltar la pieza es menor que dicho offset, aunque no sea la posición exacta correcta, la daremos por válida.

También han sido implementados dos métodos que se ocuparan de realizar un desplazamiento o escalado de carácter global, que transformarán tanto los elementos de la escena como los vectores con las posiciones iniciales y finales de las piezas. Al realizarlo de esta forma, se asegura que toda la estructura de datos guarde siempre su correspondiente coherencia con las coordenadas del mundo y no se produzca ningún error a nivel de gráficos.

Existen también un par de funciones auxiliares, que dado un índice de una pieza, nos devolverán su posición y su estado (bloqueado o no); datos que nos serán útiles para el control de la aplicación.

A la hora de cargar los ficheros con los objetos de la escena, nos apoyamos en la librería string de C++ para generar cadenas de texto con los nombres de los ficheros de forma dinámica.

En todo momento, y en los lugares donde es necesario, se realizan las comprobaciones de errores correspondientes, de forma que nos aseguremos de que el programa finalizará siempre en caso de producirse algún error, independientemente del tipo que sea.

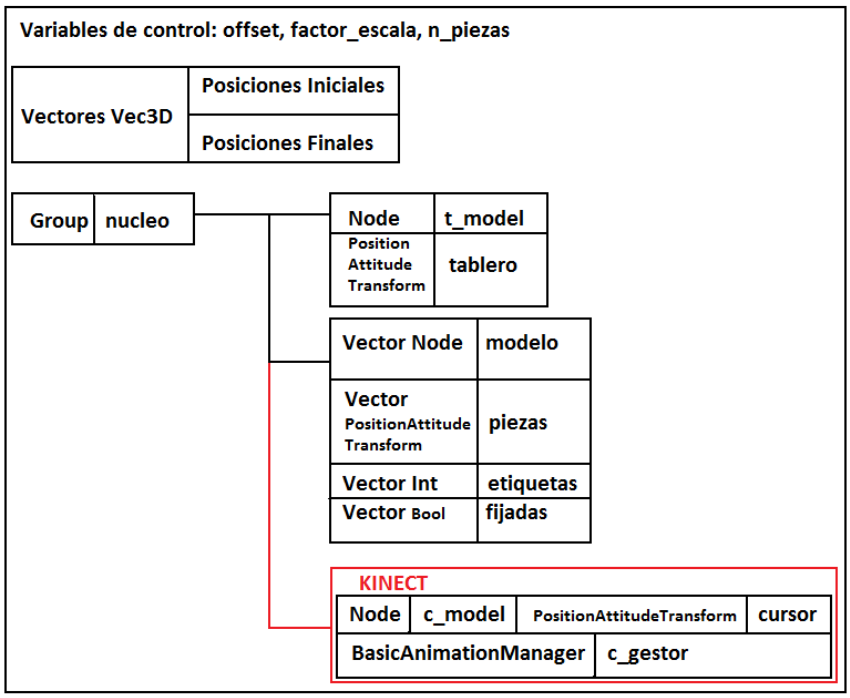

Figura 5.2.1 - Diagrama de Atributos para la clase escena

Generando la escena

A la hora de generar la escena y obtener el nodo grupo final, se realiza una única llamada a la función "genera_escena" la cual se encarga de esta tarea, realizando los siguientes pasos:

- Creamos e inicializamos todas las variables necesarias de forma dinámica.
- Establecemos el offset y el factor de escalado.
- Generamos las posiciones iniciales y finales.
- Generamos un vector de índices para las piezas y lo mezclamos.
- Cargamos el tablero, y en caso de estar en la versión de Kinect, cargamos el cursor animado. Ambos elementos los asociamos al nodo Grupo.

- En función al vector de índices, cargamos las piezas de forma desordenada en su correspondiente nodo y PositionAttitudeTransform, y las añadimos al nodo Grupo.
- Usando los PositionAttitudeTransform, desplazamos todas las piezas a su posición inicial.
- Si todo se ha creado correctamente, devolvemos un valor true. En caso de que alguno de los pasos haya fallado, devolveremos un valor false.

Para finalizar, recordar que en el paso de modelado, asignamos una etiqueta interna a cada pieza dentro de su correspondiente fichero .obj. Etiqueta que se ha guardado de forma interna y automática dentro de cada nodo al cargar una a una todas las piezas. Más adelante se verá cual es la finalidad de estas etiquetas.

Posicionamiento de los Elementos de la escena

De entre las diferentes opciones y formas que se podrían haber tomado para situar de forma correcta tanto las piezas como el tablero en el mundo, hemos optado por la que se ha considerado la más interesante y que mejor se adapta a las necesidades de la aplicación [Figura 5.2.2].

Figura 5.2.2 – Situación de las Coordenadas del mundo

Tomando como punto de origen de coordenadas (0, 0, 0) el punto central del tablero indicado de color verde, se han ido obteniendo los diferentes puntos de coordenadas necesarios para cada elemento. Se han guardado en dos vectores separados los puntos para las posiciones iniciales, marcados en rojo, y los puntos para las posiciones finales, marcados en azul. Muy importante tener en cuenta, que si se desea realizar algún tipo de escalado o transformación sobre el mundo, estos puntos deben de reflejar las mismas operaciones también, o la aplicación no funcionará de la forma esperada. Los métodos implementados para realizar transformaciones y desplazamientos globales tienen en cuenta estos detalles.

5.3.- Librería para el control con ratón

Formada por los ficheros “control_raton.h” y “control_raton.cpp”, esta librería se encargará de controlar todos los aspectos relacionados con la detección de las acciones y movimiento del ratón, su interpretación y actuación.

La librería contiene una única clase llamada PickHandler. Esta clase ha sido implementada como una especie de subclase que extiende una de las clases nativas proporcionadas por OpenNI, la clase GUIEventHandler. De esta forma, podremos pasarle esta nueva clase directamente al Viewer para que él se encargue de cargar como corresponda el manejador, pudiendo olvidarnos de ese aspecto por nuestra parte.

Los aspectos que controla esta librería son los siguientes [Figura 5.3]:

- Movimiento del cursor: Con este método, cada vez que se detecte un movimiento del ratón, buscaremos usando un LineSegmentIntersector si el ratón está situado sobre una pieza que pueda ser cogida. En caso positivo, la marcaremos como posible objetivo. Por supuesto, las coordenadas obtenidas por la posición del ratón deberán ser transformadas a coordenadas del mundo, para que la detección de intersecciones se realice de forma correcta.
- Movimiento de la pieza: Una vez tengamos una pieza marcada como seleccionada, este método se encargará de desplazar dicha pieza por la ventana junto al puntero del ratón cada vez que éste se mueva, normalizando y/o convirtiendo dichas coordenadas como sea necesario. Además si el ratón se sitúa cerca de la posición final de la pieza que estamos desplazando, la pieza será marcada como posible fijado.
- Manejador: Éste es el método principal, el cual será llamado de forma periódica por el Viewer para actualizar el estado de la aplicación.
  Con cada llamada, se comprueba el estado del ratón. En caso de estar en movimiento, se llamarán a los métodos descritos anteriormente. En caso de detectarse un clic, se comprobará el estado de la pieza (posible fijado, posible objetivo) y se actuará en consecuencia.

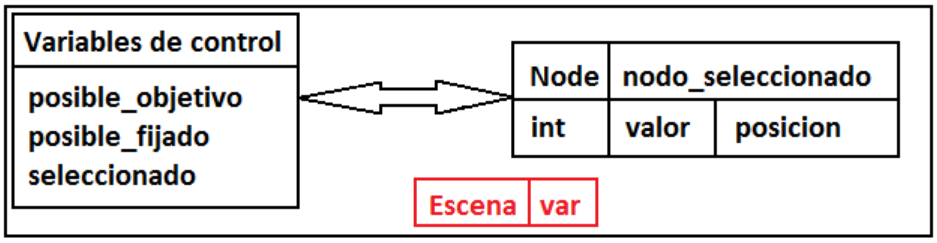

Figura 5.3 - Diagrama de atributos para la clase control_raton

Detectando las piezas

Como se ha detallado antes, se puede ver que el sistema de selección y liberación de las piezas está dividido en dos partes: Posible selección y Selección.

A la hora de detectar un posible objetivo, utilizamos la clase LineSegmentIntersector, pero esta clase detecta cualquier cosa que se encuentre, ya sean las piezas o el tablero. Para poder diferenciar tanto las piezas como el tablero, así como qué pieza exactamente hemos detectado, utilizaremos las etiquetas internas del nodo que ya hemos mencionado varias veces con anterioridad. De esta forma, si la etiquetada detectada es el tablero, no haremos nada, y en caso contrario, deberemos convertir la etiqueta a tipo entero (las etiquetas son de tipo string, independientemente de lo que guardemos en ellas) para obtener el identificador de la pieza. Una vez obtenido dicho identificador, marcaremos la pieza de la escena como posible objetivo.

Al detectar un clic de ratón, y en caso de que una pieza sea posible objetivo, y por tanto no tener ninguna pieza seleccionada, almacenaremos el identificador de dicha pieza y la marcaremos como seleccionada.

El funcionamiento a la hora de detectar la posición de fijado es similar al de la detección de un posible objetivo, solo que esta vez no es necesario usar una clase de tipo LineSegmentIntersector ya que en este caso trabajamos con posiciones pre establecidas en la clase escena. De esta forma, cuando el ratón pase por la posición de fijado de la pieza (recordar aquí el offset explicado en escena) marcaremos la pieza como posible fijado.

Debido a lo explicado anteriormente, es necesario añadir un nuevo parámetro de control a la hora de detectar el clic de ratón, de tal forma que en caso de que si tengamos una pieza seleccionada al hacer clic, se compruebe si la pieza está marcada como posible fijado. En caso afirmativo, la situaremos en su posición en el tablero, cambiando su estado a fijado para evitar volver a seleccionarla por error. En caso contrario, la devolveremos automáticamente a su posición inicial. En ambos casos, todas las variables de control serán reiniciadas (posible objetivo, seleccionado, posible fijado).

Con todos los métodos y atributos definidos y preparados, solo quedará cargar nuestro manejador en el Viewer usando el método correspondiente (addEventHandler).

5.4.- Librería para la gestión de Kinect

El objetivo de esta librería es el ofrecer de una manera simple y rápida los puntos de interés de un cuerpo detectado, ya sean las manos, los pies, la cabeza o cualquier parte significativa que deseemos saber. Formada por los ficheros “SkeletonSensor.h” y “SkeletonSensor.cpp”, contiene la clase SkeletonSensor, que se encargará de inicializar y preparar el sensor, así como de gestionar y generar la información relativa al cuerpo al que se le esté haciendo un seguimiento. La librería esta implementada de forma que, una vez inicializado el sensor, con solo un par de métodos se puedan realizar todas estas acciones de

forma automática sin necesidad de ningún otro tipo de implementación extra por parte del programador o de ningún tipo de inicialización por parte del usuario final.

Los aspectos a controlar por esta librería son los siguientes:

- Inicialización del Sensor: Se crean todos los atributos necesarios para realizar una inicialización correcta del sensor y se prepara para la recolección de información por parte del mismo.
- Llamadas del sistema o CallBacks: Se preparan las llamadas al sistema necesarias para realizar el seguimiento del usuario. Las llamadas que nos interesan son: la detección de un nuevo usuario, la pérdida de un usuario, la inicialización de la calibración, la finalización de la misma y el comienzo del seguimiento del usuario detectado.
- Seguimiento de usuarios detectados: La librería es capaz de mantener el control de varios usuarios detectados por el sensor, manteniendo una lista de usuarios detectados. Añadiendo o eliminando nuevos usuarios a la lista de forma dinámica.
- Obtención de puntos de interés: La librería permite solicitar en cualquier momento, con la ayuda de un método, los puntos de interés de cualquier cuerpo que estemos detectando, para después trabajar con estas coordenadas como se crea conveniente.
  Los puntos de interés que podemos detectar son los siguientes: Cabeza, cuello, hombros, codos, manos, caderas, rodillas, pies y torso.
  Para cada punto, obtendremos tanto su coordenada 3D (x,y,z) como su nivel de confianza.

Esta librería necesita la librería OpenNI para funcionar correctamente. Además, posee un fuerte control de los posibles errores relacionados con la comunicación y el funcionamiento del sensor.

5.5.- Librería para el control con Kinect

Una vez inicializado el sensor con todas las llamadas al sistema preparadas, y con un método de obtención para los puntos de interés de forma eficaz, será necesario interpretar y controlar la aplicación en concordancia con el movimiento de las manos. Para tal tarea nos apoyamos en la siguiente librería, compuesta por los ficheros “control_kinect.h” y “control_kinect.cpp”. Aunque el planteamiento pueda resultar similar al de los controles por ratón, algunas de las funcionalidades y peculiaridades de trabajar con reconocimiento de gestos naturales en tiempo real nos obliga a afrontar el problema del control de nuestra aplicación enfocándolo desde otro punto de vista.

De forma similar a los controles por ratón, la librería cubre los siguientes aspectos de la aplicación relacionados con el control de la misma, implementados todos ellos en una única clase llamada PickHandler:

- Movimiento del cursor: Lo primero que debe realizar este método es actualizar la posición de nuestro puntero animado, para que el usuario sea capaz de controlar en todo momento que está realizando. Además, cada vez que se detecte un movimiento con cualquiera de las dos manos, buscaremos usando un LineSegmentIntersector si la mano derecha está situada sobre una pieza que pueda ser cogida. En caso positivo, la marcaremos como posible objetivo. Será necesario normalizar las coordenadas de la mano obtenidas por Kinect para acto seguido transformarlas en la posición correcta dentro del mundo, para que la detección de elementos tenga el efecto deseado.
- Movimiento de la pieza: Una vez tengamos una pieza marcada como seleccionada, este método se encargará de desplazar dicha pieza por la ventana junto a nuestro puntero animado, para saber en todo momento hacia donde está llevando la pieza seleccionada. Por supuesto, será necesario normalizar y transformadas las coordenadas de la mano obtenidas por Kinect. Además si la pieza se sitúa cerca de la que debe ser su posición final, será necesario marcarla como posible fijado.
- Manejador: Este es el método principal, el cual será llamado de forma periódica, como en el caso del ratón, por el Viewer para actualizar el estado de la aplicación.

  Con cada llamada, se comprueba la posición de las dos manos. En caso de estar en movimiento, se llamarán a los métodos descritos anteriormente. En caso de ser detectado un gesto de tipo empujar o tirar con la mano izquierda, se comprobará el estado de la pieza (posible fijado, posible objetivo) y se actuará en consecuencia.

Al igual que con la librería para los controles de ratón, la clase de esta librería se ha implementado como una extensión de la clase GUIEventHandler para que sea el Viewer de OSG el que se encargue de controlar las llamadas al manejador de forma periódica [Figura 5.5].

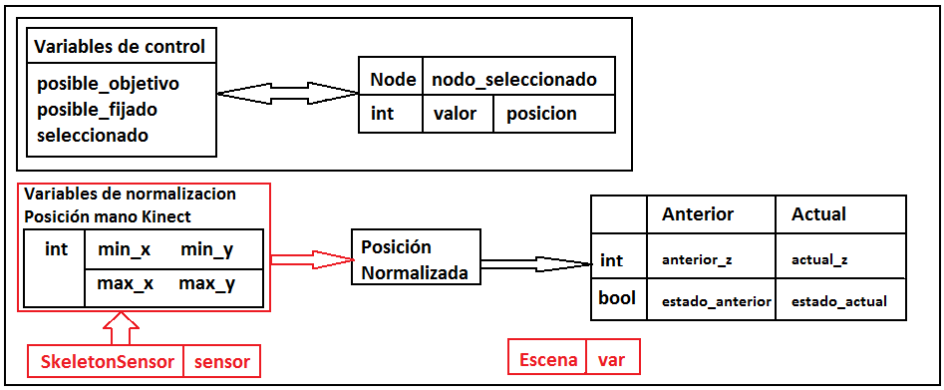

Figura 5.5 - Diagrama de atributos para la clase control_kinect

Primera versión

Aunque lo explicado anteriormente haga referencia al funcionamiento final de la librería, se hace interesante explicar con detalle el funcionamiento de la que fue la primera versión de dicha librería. De esta forma se puede mostrar y plantear los problemas que recaen en la captura de datos del sensor en tiempo real, así como la perdida de precisión y aumento de la complejidad del funcionamiento del programa, al intentar realizar todos los controles del mismo con una única mano.

Problema de precisión

El auténtico problema radica en que, a la hora de detectar la variación en profundidad de la posición de la mano, se hace inevitable que también se genere una variación tanto en desplazamiento como altura de la misma mano. A causa de esto, se estaba provocando de forma innecesaria que el seleccionado de una pieza se convirtiera en una tarea muy complicada, debido a los movimientos bruscos que era necesario realizar para capturar correctamente los gestos de empujar y/o estirar.

Después de intentar realizar diversas variaciones para facilitar la tarea de movimiento con una única mano, se terminó decidiendo separar el control de la aplicación entre las dos manos. Al separar los controles entre las dos manos, se consiguen solucionar los errores de precisión ocasionados por los movimientos bruscos de la mano comentados anteriormente. Con esta acción se consigue aislar el problema de detección de gestos a la mano izquierda, mientras que el problema de precisión quedará solucionado, al no tener que realizar ningún gesto brusco con la mano con la que se irán seleccionando las piezas del puzzle.

Problema de detección

El hecho de que se estén procesando los datos con la localización de la mano en tiempo real provoca que, a la hora de detectar un gesto concreto (empujar o estirar con la mano, éste tenga que realizarse de forma muy rápida para poder detectarlo. La solución que hemos utilizado consiste en añadir, a la hora de detectar la mano izquierda únicamente, un retardo entre actualización y actualización de la posición de dicha mano. De esta forma conseguimos, por una parte, reducir la complejidad de los controles (primer problema que queríamos resolver) y por otra parte capturar de una forma eficaz los gestos realizados con la mano izquierda.

5.6.- Programa Principal

Una vez explicadas todas y cada una de las librerías que componen la aplicación, solo queda detallar el funcionamiento del programa principal main.

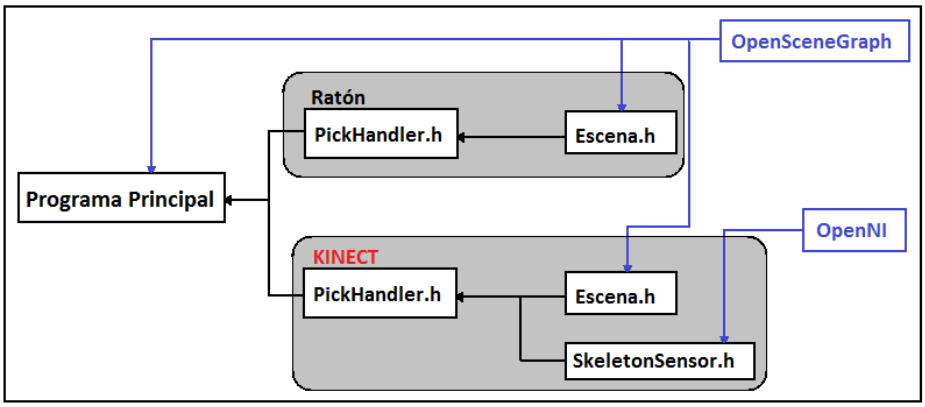

Figura 5.6.1 - Árbol de dependencias del programa principal

Gracias a la programación tan jerarquizada que hemos realizado en el resto de clases, la complejidad de la función principal del programa queda bastante simplificada [Figura 5.6.1]. En esta llamada nos centraremos en obtener, con ayuda de varios métodos de la librería OSG, las especificaciones del monitor o monitores, para acto seguido establecer una ventana de visualización sobre el monitor principal (en caso de haberse detectado varios monitores). A continuación, crearemos una variable de tipo PickHandler y la inicializaremos, junto a las variables internas de escena y de SkeletonSensor, para el caso de los controles con Kinect. Con las dos variables inicializadas, crearemos una variable de tipo Viewer sobre la que cargaremos tanto la escena, como los parámetros de la ventana de visualización, como el manejador de PickHandler. Una vez realizado esto, cargaremos los parámetros de la cámara que visualizará nuestra escena, siendo éstos el vector Posición, el vector Look y el vector UP [Figura 5.6.2].

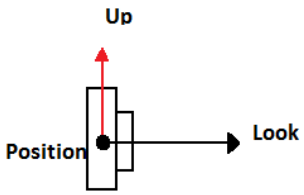

Figura 5.6.2 - Parámetros de una cámara en un entorno 3D

Para finalizar, solo quedará llamar a las funciones de inicialización del Viewer y generar el pequeño bucle principal que arrancará todo el programa. El resultado final se puede observar a continuación [Figura 5.6.3] [Figura 5.6.4]:

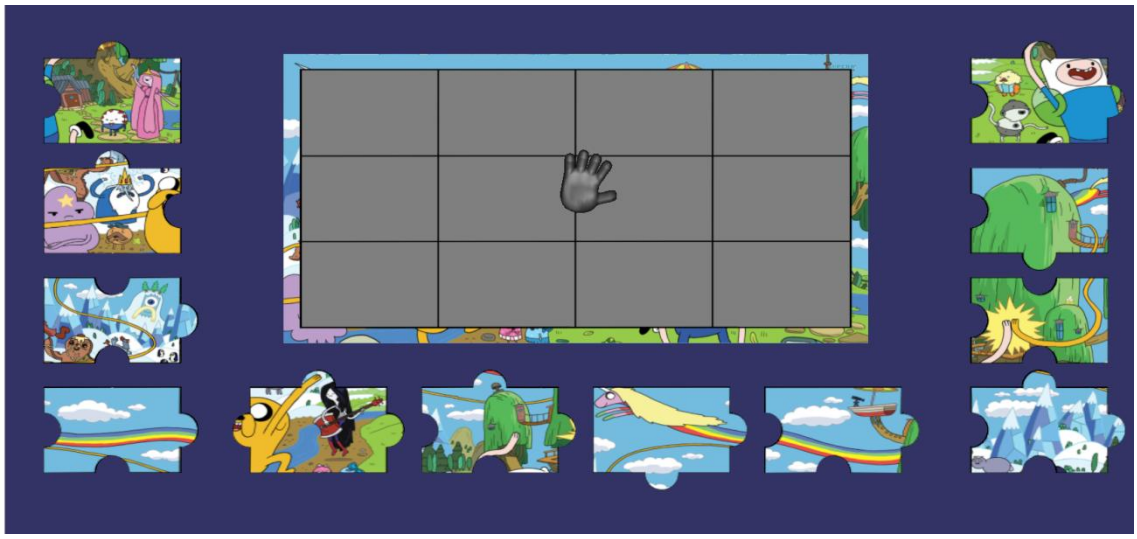

Figura 5.6.3 - Captura de pantalla 1

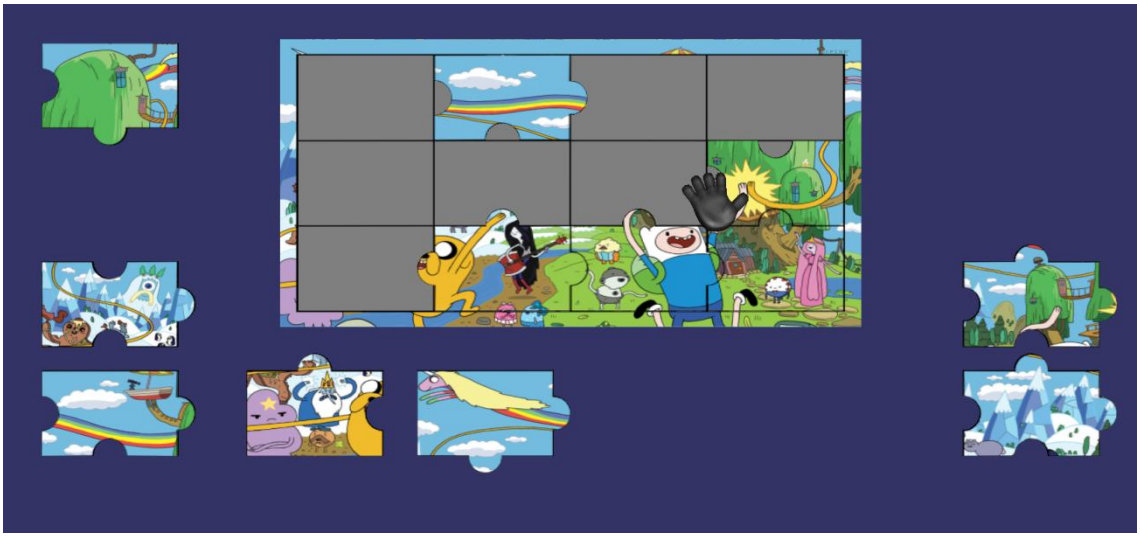

Figura 5.6.4 - Captura de pantalla 2

Estas dos capturas de pantalla han sido realizadas en dos ejecuciones diferentes de la aplicación con los controles para Kinect. En ellas se pueden apreciar todos los elementos de la clase escena, así como la forma que irá tomando el puzzle conforme se vayan situando en su sitio las diferentes piezas. También se puede observar cómo las posiciones iniciales de las piezas és diferente en cada caso, tal y como se explicó que pasaría en el apartado dedicado a la clase escena.

La aplicación con los controles por ratón es exactamente la misma, salvo por la ausencia del cursor animado, el cual se ha retirado ya que simplemente, no era necesario.

Capítulo 6

CONCLUSIÓN

Gracias a la realización de este proyecto se ha podido comprobar, en primera instancia, las ventajas y desventajas que puede conllevar el trabajar o interactuar con un ordenador o aparato tecnológico de forma dinámica mediante el uso de interfaces de usuario naturales. Además, también se ha investigado sobre las diferentes opciones, aplicaciones, herramientas y librerías que han sido creadas. Aplicaciones que se siguen creando y mejorando para tener un mejor acceso y más intuitivo a la programación de todo tipo de sensores de reconocimiento de entorno, tales como Kinect, para poder llevar a cabo el desarrollo de estas interfaces naturales.

Por otro lado, y ya metidos en el desarrollo de la aplicación del proyecto, se han hecho patentes la complejidad y problemática del modelado y diseño de una interfaz de este tipo. Cosas tales como la precisión a la hora de detectar un usuario e incluso la propia detección de gestos y movimientos del usuario, no son problemas con una resolución trivial. Ha sido necesario aprovechar toda la información que nos puede ofrecer Kinect como sensor de movimiento y profundidad, para poder aplicar algoritmos y técnicas que nos permitan conseguir una resolución al problema válida. Ofreciendo de esta forma una interfaz natural lo suficientemente estable para que no resulte ser un impedimento ni una molestia para el usuario el utilizarla. Ya que al fin y al cabo, el objetivo de dichas interfaces naturales es el de permitir al usuario interactuar con el ordenador de una forma más rápida y sencilla que los métodos convencionales que existen hoy en día.

Otro de los aspectos tratados en el proyecto ha sido el aprendizaje y posterior utilización de una potente librería para el procesado de gráficos y creación de entornos totalmente en 3D. El aprendizaje y utilización de los grafos de escena ha permitido realizar el desarrollo, modelado y transformación del entorno 3D de una forma ordenada y precisa.

La creación de los elementos necesarios para poder interactuar con el sensor de movimiento, así como el procesamiento de la información obtenida por el mismo a alto nivel, se puede considerar como otro de los aspectos tratados en el proyecto. La adaptación del entorno y de la aplicación para que trabaje en concordancia con la información procesada por el sensor ha sido sin duda la parte más compleja del trabajo realizado. Aunque la propia librería ofrece herramientas intermedias para la obtención de esta información, el correcto uso de dichas herramientas para poder obtener los resultados buscados ha resultado ser más complejo de lo esperado. Lo cual solo nos confirma que, aunque ya existen diversas opciones para la programación de aplicaciones con sensores, este campo y temática aun posee mucho espacio para la ampliación y mejora. Porque como ya ha quedado demostrado, el potencial de las interfaces naturales para la ampliación de la interacción entre personas y aparatos tecnológicos existe, y solo es cuestión de tiempo que este tipo de aplicaciones se expanda de forma exponencial, llegando a obtenerse resultados dignos de las mejores películas de ciencia ficción.

6.1.- Futuras Expansiones

Aunque las dos aplicaciones desarrolladas están completas y totalmente funcionales. En una aplicación de estas características siempre hay espacio a la mejora y ampliación. Se dejan planteadas una serie de posibles mejoras, que por falta de tiempo, o por no ser objetivo principal del desarrollo y de la problemática estudiada en este proyecto, se han tenido que posponer o dejar para un futuro:

- A nivel interno, aunque se ha intentado mantener el uso de memoria en lo menor posible, con un análisis detallado y conciso es más que seguro que se puedan aplicar mejoras relativas a la optimización a nivel de memoria. En cuanto al nivel de control de operaciones, es podrá investigar y aplicar mejoras para obtener un aumento de la precisión y robustez de los controles con las manos, con el objetivo de reducir al mínimo el número de acciones necesarias para hacer funcionar la aplicación correctamente, sin aumentar la complejidad de la misma para el usuario final de la aplicación implementada.
- A nivel de interfaz gráfica, cualquier aspecto típico de un videojuego puede ser aplicado y adaptado al programa. Cosas como puntuaciones en base al tiempo y al número de aciertos o fallos del usuario, mantenimiento de un ranking de dichas puntuaciones comparando con otros usuarios, mejoras del aspecto visual de la aplicación, aumento de la complejidad del puzzle, o la adicción de menús de entrada y salida a la aplicación… Son muchos y muy diversos los cambios y mejoras que se pueden realizar a este nivel con el objetivo de hacer más agradable e interesante la experiencia del usuario con el juego.
- Cuando se hayan realizado las mejoras necesarias para obtener una aplicación más robusta y de controles amigables para los usuarios, se ha planteado también realizar una serie de pruebas con usuarios para comparar la funcionalidad de las dos aplicaciones desarrolladas.

Capítulo 7

BIBLIOGRAFÍA

[APPLE 01] Web oficial de Apple iPhone - http://www.apple.com/es/iphone/, fecha de consulta: 15/12/2012.

[APPLE 02] Web oficial de Apple iPad - http://www.apple.com/es/ipad/, fecha de consulta: 15/12/2012.

[BLEND 01] Web oficial de Blender - http://www.blender.org/, fecha de consulta: 15/12/2012.

[BLEND 02] Blender Documentation - http://wiki.blender.org/, fecha de consulta: 15/12/2012.

[CHANG 11] Chang Y., Chen S., Huang J. (2011) "A Kinect-based system for physical rehabilitation: A pilot study for Young adults with motor disabilities", Research in Developmental Disabilities, Volume 32, Issue 6, pp. 2566-2570.

[CLARK 12] Clark R. A., Pua Y., Fortin K., Ritchie C., Webster K. E., Denehy L., Bryant A. L. (2012) "Validity of the Microsoft Kinect for assessment of postural control", Gait & Posture, Volume 36, Issue 3, pp. 372-377.

[DAVE 12] Dave A., Bhumkar Y., Abraham A., Sugandhi R. (2012) "Project MUDRA: Personalization of Computers using Natural Interface", International Journal of Computer Aplications, Volume 54, Number 17, pp. 42-46.

[DIAZ 11] Díaz V. (2011) "Guía de desarrollo de aplicaciones sobre Wii", Proyecto final de Carrera de la Escuela Técnica Superior de Ingeniería Informática, pp. 7-14.

[GALITZ 07] Galitz W. O. (2007) "The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques", Third Edition, Editorial Indianapolis, IN: Wiley Pub, pp. 3-12.

[KERKY 12] Web official de Kerkythea - http://www.kerkythea.net/joomla/, fecha de consulta: 16/12/2012.

[KINE 12] Web oficial de Microsoft Kinect - http://www.microsoft.com/en-us/kinectforwindows/, fecha de consulta: 15/12/2012.

[KINEDOC 12] Kinect Documentation - http://www.microsoft.com/en-us/kinectforwindows/develop/resources.aspx, fecha de consulta: 16/12/2012.

[OPENNI 01] Web oficial de Open Natural Interface OpenNI - http://openni.org/, fecha de consulta: 16/12/2012.

[OPENNI 02] OpenNI Documentation - http://openni.org/Documentation/home.html, fecha de consulta: 16/12/2012.

[OSG 01] Web oficial de OpenSceneGraph OSG - http://www.openscenegraph.org, fecha de consulta: 16/12/2012.

[OSG 02] OSG Documentation - http://www.openscenegraph.org/projects/osg/wiki/Support, fecha de consulta: 16/12/2012.

[PALERO 12] Palero F. (2012) "Librerías Wrapper para el acceso a cámaras Kinect y Herramientas para el calibrado de las cámaras RGB y profundidad", Proyecto final de Carrera de la Escuela Técnica Superior de Ingeniería Informática, pp. 5-7.

[PSMOVE 10] Web oficial de Playstation MOVE - http://es.playstation.com/psmove/, fecha de consulta: 16/12/2012.

[WOOLFORD 03] Woolford D. (2003) "Understanding and Using Scene Graphs", COMP4201 Lectures, pp. 3-7, 19-22.

[YAFARAY 12] Web oficial de Yafaray - http://www.yafaray.org/, fecha de consulta: 16/12/2012.

[ZHANG 12] Zhang Z. (2012) "Microsoft Kinect Sensor and Its Effect", Multimedia IEEE, Volume 19, Issue 2, pp. 4-10.