UNIVERSIDAD REY JUAN CARLOS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Curso Académico 2012/2013

Proyecto de Fin de Carrera

Air Drum con Kinect. Simulación de batería instrumental con dispositivo de infrarrojos.

Autor: Aaron Sújar Garrido

Tutor: Francisco Domínguez Mateos

Me gustaría agradecer a todos mis compañeros y profesores que me han acompañado a lo largo de mi carrera.
También se lo quiero dedicar a mi madre por apoyarme en mis estudios y a Lorena por su gran ayuda.

RESUMEN

El propósito de este proyecto es la realización de una simulación de una batería instrumental mediante una aplicación orientada a gráficos 3D, inspirada en las aplicaciones más representativas de este campo como son los videojuegos y los simuladores. Tomando estos como referencia, en este proyecto se han realizado investigaciones y estudios para el desarrollo de la aplicación, tales como buscar formas de conseguir una simulación lo más parecida posible, y tomado como ejemplos aplicaciones previas que sirvan como guía y modelo para la realización del proyecto.

Para ello, se estructuró los pasos a seguir con el objetivo de crear una aplicación de simulación de un entorno educativo y musical, además de utilizarlo como medio de aprendizaje hacia la orientación profesional.

El primer paso para la realización del proyecto, fue elegir OpenInventor, una librería que ayudó a comprender los aspectos de los gráficos 3D. Además se ha acompañado de tecnologías conocidas y presentes en la industria de los videojuegos, como el dispositivo de infrarrojos Kinect, en cuanto al sonido se ha utilizado OpenAL, y OpenCV, como librería para operaciones de visión computacional.

El segundo paso era seguir una metodología aproximada al modelo de cascada y así ser capaz de realizar un desarrollo de software de calidad, con una gestión de configuración, realizando la documentación del software con Doxygen, definiendo unas pruebas, e incorporando un log profesional con Log4cxx.

Además de estos aspectos, se ha incluido una evaluación de la aplicación en un grupo de usuarios, para valorar factores claves del desarrollo, como el realismo, la diversión y la facilidad de uso, y así obtener un resultado por un procedimiento científico semejante a los utilizados en las publicaciones y artículos científicos.

Se ha conseguido una simulación dónde el usuario será capaz de interactuar con la batería musical, con una respuesta realista y en tiempo real, con una curva de aprendizaje mínima, siendo además, independiente a la posición, dimensión y vestimenta del usuario.

ÍNDICE

GUÍA DE LECTURA

En este libro se tratará toda la información relativa al proyecto fin de carrera. Como tal, está estructurado según las directrices que marca la Escuela Técnica Superior de Ingeniería Informática.

Primero se encuentra la Introducción, donde se detallará los aspectos más importantes del proyecto A continuación, se comentará los Objetivos en el cual se ha basado el proyecto. Se enumerarán: las aplicaciones que han servido de inspiración, los conceptos previos que se han utilizado para desarrollar la aplicación, los trackers que existen y en particular, la descripción del Kinect®, los algoritmos que usa este tracker, y por último aplicaciones previas realizadas con este dispositivo.

Seguidamente, se detallará la Descripción informática en donde se encuentran las tecnologías usadas en el desarrollo de la aplicación. Además, se detallará el diagrama de clases con una descripción de las clases que se han implementado. A continuación, se explicará todas las funcionalidades que se han implementado en la aplicación, comentando las peculiaridades y decisiones que se han optado en el desarrollo de la aplicación.

También en este apartado se encuentra la definición de pruebas que se han realizado en la verificación y validación de la aplicación, y se comentarán los resultados obtenidos.

El siguiente aparado es Conclusiones , donde se muestra el resultado de las opiniones de los usuarios que han probado la aplicación, y las líneas futuras que podría seguir esta aplicación

En último lugar se encuentran la Bibliografía, el Apéndice y el Manual de Usuario, entre otros apartados auxiliares que ayudarán a completar esta memoria.

Comentar que la bibliografía se representará de la siguiente manera: [Número] para la bibliografía de libros y artículos, [Clave] para los enlaces de internet, e Ilustraciones Número para las imágenes usadas en toda la memoria.

1. INTRODUCCIÓN

1.1. Historia de los Gráficos

La aparición de los ordenadores ha sido un gran hito en la historia de la humanidad y, su incorporación en todos los ámbitos de la sociedad ha hecho posible un gran avance en el corto espacio de tiempo, de menos de un siglo, desde que están presentes. No sólo la capacidad de cómputo, en particular la posibilidad de crear contenido gráfico económico y rápido, ha sido capaz de impulsar tecnologías ya existentes, o la de impulsar también nuevas tecnologías. [1] [10]

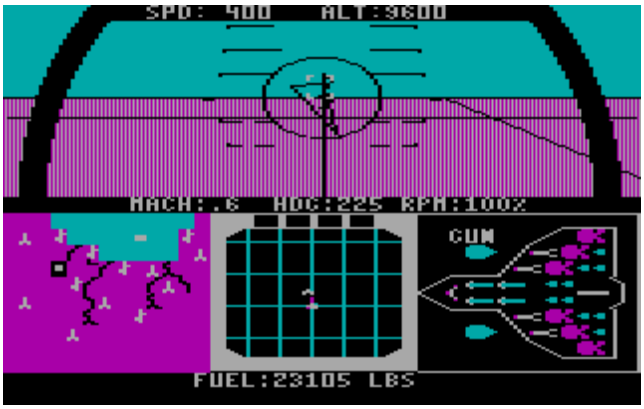


Ilustración 1: F-15 Eagle. Primeros juegos de simulación que aparecen a mediados de los 80. Se usaban para entrenar a soldados [1WE].

Las necesidades crecientes de gráficos más reales, han desencadenado en hardware y software cada vez más potentes, siendo posible la representación de objetos mucho más complejos. Los ordenadores son una fuente muy rica y poderosa con un coste relativamente pequeño, lo que ha resultado que esté presente en muchas áreas de la ingeniería y la ciencia entre otros. Son la ingeniería y la ciencia, ramas en las cuales existe multitud de necesidades, donde los gráficos simplifican mucho la representación de datos o, por ejemplo, mejoran la enseñanza.

Existen dos grandes grupos de software para gráficos por ordenador: paquetes de propósito específico y paquetes de programación general.

Los paquetes de propósito específico se diseñan para quienes no saben programar y quieran generar imágenes, gráficos o diagramas en algún área de aplicación sin

preocuparse por los procedimientos gráficos necesarios para producir las imágenes. La interfaz de un paquete de propósito específico es habitualmente un conjunto de menús, que permite a los usuarios comunicarse con los programas. Como ejemplos de aplicaciones se pueden citar: los programas de pintura, sistemas CAD, programas para crear presentaciones, etc. En particular, en este proyecto se ha utilizado 3ds Max y Sculptris.

En cambio, un paquete de programación general, proporciona una biblioteca de funciones gráficas que se pueden utilizar en lenguajes de programación tales como C++, Java, Ruby, u otros como por ejemplo, Python. Entre las funciones básicas de una biblioteca gráfica típica, se incluyen aquellas para especificar componentes de la imagen (figuras y formas), establecer colores e iluminaciones, seleccionar vistas de la escena, o aplicar rotaciones u otras transformaciones. Algunos de los ejemplos de paquetes de programación gráfica son: OpenGL, Java 3D, etc. En el caso de este proyecto, se ha hecho uso de paquetes como VRML, Open Inventor.

Ilustración 2: fotograma de la película de animación "Gru, mi villano favorito" (2010 Universal Studios). Película creada expresamente para aprovechar la tecnología 3D. [GRU]

1.2. Representación de datos 3D

Los datos 3D se pueden representar de múltiples formas. En el mundo real, existen multitud de tipos de objetos tales como rocas, árboles, edificios, animales, y un sinfín de cosas, que el ser humano lleva intentando representarlos desde las cavernas y las pinturas rupestres, hasta los gráficos por ordenador. Como en la pintura, en los gráficos 3D hay muchos estilos y técnicas para mostrar los objetos y que el receptor sea capaz de interpretarlos.

No existe una técnica universal, ya que cada forma de representar los objetos es más adecuada para unos casos en particular. A continuación se comentarán algunas de las distintas técnicas para representar datos y modelos en 3D. [3]

1.2.1. Superficies paramétricas

Una superficie paramétrica es una superficie en el espacio euclidiano R^3, la cual es definida por una ecuación paramétrica con dos parámetros. Mediante la utilización de parches de curvas de Bezier, B-Splines o NURBS se pueden representar superficies perfectamente curvas. Estos se pueden situar uno detrás de otro, dando continuidad a la superficie. Esta técnica tiene un coste alto a la hora de representarse, pues es difícil garantizar la localidad de cada parche. Una modificación en alguno de ellos, afecta a los demás.

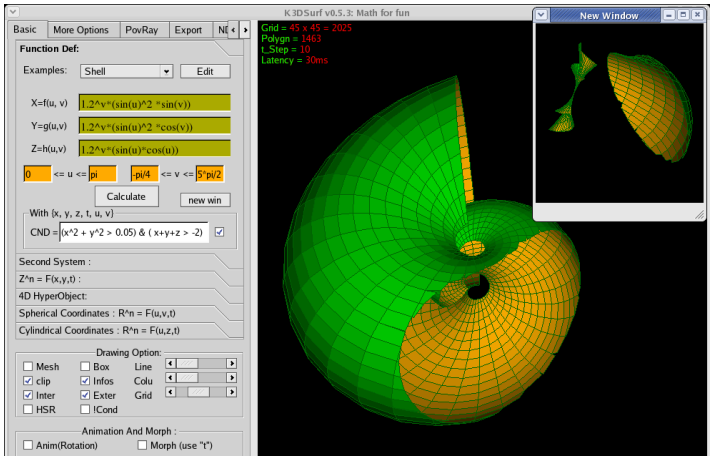


Ilustración 3: superficie paramétrica representada en el software K3DSurf. [K3D]

1.2.2. Superficies implícitas

Las superficies implícitas se definen mediante un conjunto de componentes básicos, como puntos o esferas, y su combinación mediante alguna función. Generalmente la función es de tipo potencial, de tal forma, que definen campos de influencia sobre cualquier punto del espacio. La idea es conceptualmente sencilla si se piensa en el comportamiento de gotas de agua que, cuando se juntan lo suficiente se unen formando una gota más grande.

Esta técnica se lleva empleando durante años en el modelado de formas orgánicas. Se hicieron muy famosas a principios de esta década debido a la aparición de varios plugins para paquetes de diseño 3D comerciales que explotaban sus posibilidades.

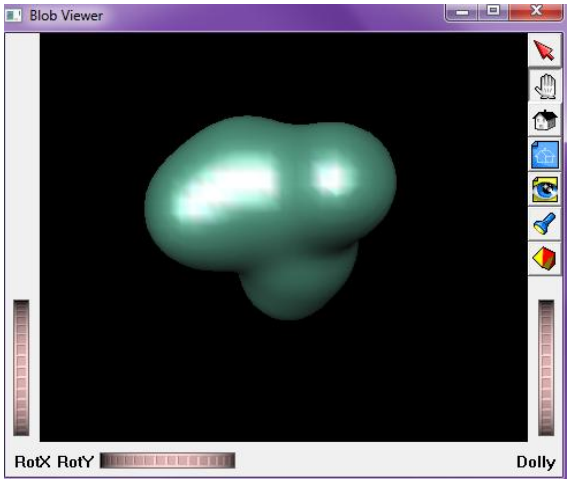


Ilustración 4 Ejemplo de la librería de Coin. Se puede observar una metabola.

1.2.3. Superficies poligonales

Las mallas poligonales se basan en describir la superficie del objeto a partir de un conjunto de polígonos conectados, a través de vértices que definen sus aristas. Éstos así componen polígonos, que en la mayoría son triángulos al ser la unidad mínima de una superficie definida sólo por 3 vértices. Otros polígonos tiene la desventaja de falta de coplanariedad, llamado efecto pajarita, que afectan a la complejidad de su representación. Las ventajas de esta técnica son la simplicidad y la disposición del hardware.

Esta técnica es sencilla en cuanto a representación pues requiere menos cálculo pesado y la mayoría del hardware está orientado a procesar la información en calcular vértices y triángulos.

La naturaleza no es poligonal y no corresponden las representaciones de los objetos con las superficies poligonales. Así también, las superficies demasiado complejas no son sencillas de representar.

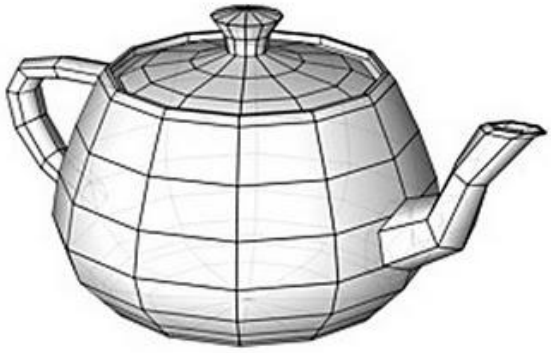


Ilustración 5: Tetera de Utah modelada con polígonos. [DES10]

Varias de estas técnicas han sido utilizadas para la representación tridimensional de la aplicación. Por ejemplo, la batería de música que se visualizará en la aplicación, tiene una estructura poligonal.

Se realizará un repaso sobre las distintas aplicaciones gráficas, que tienen relación con este proyecto.[1] [3]

1.3. **Herramientas CAD**

Una aplicación de los gráficos que se usan en gran parte de la ingeniería y la arquitectura, son las herramientas CAD (*computer-aided design).* El diseño asistido por ordenador, es un uso bastante generalizado en la fabricación de productos, visualización de objetos, diseño de edificios, etc.

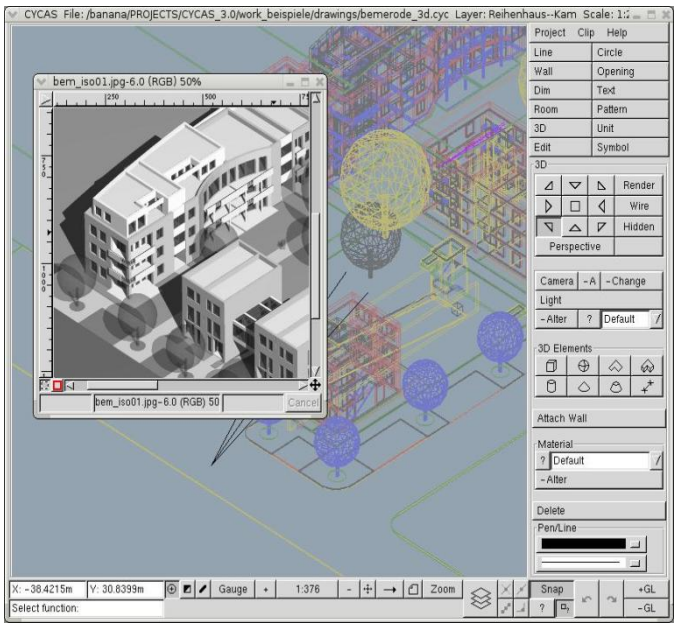


Ilustración 6: CYCAS Aplicación CAD. En esta imagen se VE como una herramienta CAD muestra un modelo con mallas para facilitar posibles modificaciones. En la imagen también se puede observar cómo se renderiza en una imagen con texturas [CYCAS].

En estas aplicaciones de diseño, los objetos se representan normalmente en forma de mallas o, a veces, son recubiertas por una serie de texturas, mostrando la forma general del objeto.

Normalmente, las aplicaciones CAD permiten ordenar y procesar la información relativa a las características de un objeto o modelo. En el caso particular de la arquitectura, la herramienta CAD, sirve para construir un modelo análogo del edificio o instalación. En el espacio imaginario es posible construir con elementos también imaginarios, la mayor parte de los componentes del objeto o modelo; colocar cada elemento en la posición que le corresponde en relación a los demás, caracterizar cada elemento en función de sus propiedades intrínsecas (forma, tamaño, material, etc.) y también caracterizarlo en sus propiedades extrínsecas (función, precio, etc.). Por tanto,

permite a la vez, ver en la pantalla las plantas, cortes o vistas necesarias del modelo que se está construyendo, y así tener la posibilidad de modificar, en cualquier momento las características del mismo.

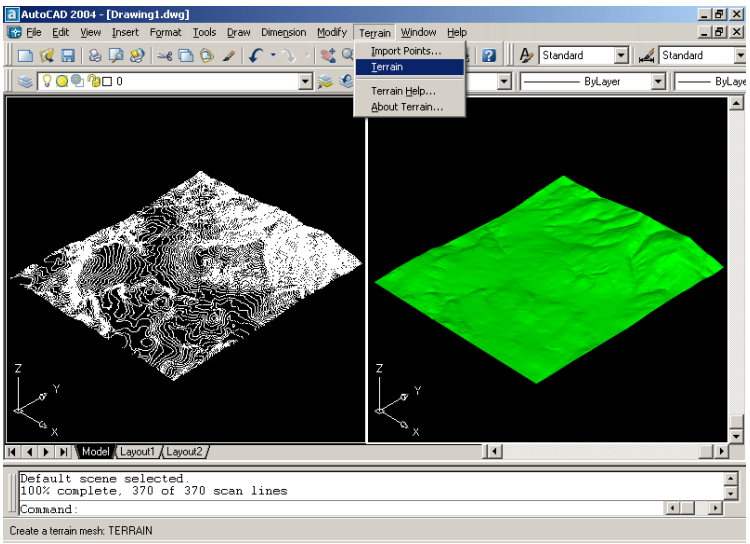


Ilustración 7: AutoCAD, herramienta más representativa de las CAD [ACAD].

Los cambios al modelo son reflejados instantáneamente en las distintas formas de representación, por lo que estas herramientas hacen posible la verificación constante de las decisiones del arquitecto, creador o diseñador, sin necesidad de rehacer una y otra vez, los dibujos.

Para la realización de este proyecto, se han utilizado dos programas CAD para realizar ciertos modelos según la dificultad y el objetivo que se quiera realizar. Son 3ds Max Studio y Sculptris.

1.4. Simuladores y Realidad Virtual

Una aplicación más reciente de los gráficos, es la creación de entornos de realidad virtual (RV en adelante). La RV, se trata de una nueva ciencia que a través de dispositivos y ordenadores, se intenta crear un mundo que el usuario recibirá por el mayor número de sentidos, llegando a conseguir que se sienta inmerso. En este proyecto se integrarán los 3 sentidos más importantes del ser humano. Respecto a la vista, la aplicación muestra una ventana del mundo virtual; en cuanto al oído, la aplicación reproducirá sonidos con ciertas propiedades, y para la interacción con los brazos, la aplicación recibirá el

movimiento de los brazos como entrada.

Por tanto, se intenta engañar a los sentidos haciendo que la inmersión en ese mundo sea interactiva usando nuestra imaginación. Tales creaciones no tienen por qué limitarse a la reproducción más o menos fidedigna de entornos reales, pueden construirse entornos sintéticos mediante los que se manipulan determinados propiedades, de manera que se obtienen resultados improbables o imposibles en la realidad.

Se puede marcar el objetivo de la RV en tres hitos llamados las tres íes: Inmersión, Interacción e Imaginación.

Cuanto mayor sea la inmersión en el mundo virtual y se pierda el contacto, más realista será la aplicación. Para ello, también es necesaria una interacción con el sistema, de tal modo que ayuda a la inmersión del mundo si se es capaz de actuar y percibir la RV. La imaginación será la encargada de dar un abanico de posibilidades en el uso de la RV.

Con estos tres hitos, se mejora la presencia en el usuario. Se denomina presencia a la sensación de "estar dentro" el entorno virtual. [16] Los sujetos que pasan por entornos de realidad virtual no tienen la sensación de observar éstos desde fuera, si no de formar parte de ellos. Por ejemplo, investigadores han visto la oportunidad de crear multitud de aplicaciones no sólo para ocio sino utilidades para la vida cotidiana como por ejemplo, tratamiento de fobias, entornos educativos, etc.

Hay diferentes factores que contribuyen a incrementar la sensación de presencia en un entorno virtual. Algunos de ellos son de carácter perceptivo y otros motores. Aquellos equipos que limitan la entrada de estímulos del ambiente real y potencian los correspondientes al entorno virtual, por mecanismos perceptivos, disminuye la sensación de presencia en el mundo real e incrementan la presencia en el entorno virtual. Los cascos de RV es la solución más conocida de este tipo, con el cual el campo visual del usuario queda prácticamente cubierto por la información que proviene del entorno virtual. La presencia también depende de variables motoras. Si el sujeto tiene posibilidades de interacción con el entorno virtual (desplazarse, tocar objetos, moverlos, etc.) su sensación de presencia será mayor que si debe limitarse a observar lo que ocurre. Una variable central en este punto, es la velocidad de respuesta en tiempo real, es determinante que la interacción lleve a un aumento de la presencia.

Los componentes de un sistema de realidad virtual pueden ser muy diversos. Todavía no se ha llegado a un sistema estándar, de manera que muchos de ellos tengan aspecto de prototipos. Se suelen construir con un objetivo particular, y eso hace que los productos de diferentes grupos de investigación sean difícilmente exportables a otros. La

falta de estándares en este campo complica también notablemente los intentos de replicación de resultados por parte de diferentes grupos.

Uno de los dispositivos de salida visual más inmersivos, y que elimina algunos inconvenientes de los cascos, es el CAVE, cuevas virtuales que rodean al usuario.

En este proyecto sólo se contempla la utilización de un monitor estándar, pues cualquier otro dispositivo de salida sería más complejo y retrasaría más el desarrollo de este proyecto.

Por otra parte, el dispositivo de salida auditivo, son los altavoces, al desarrollar la aplicación para que reproduzca sonidos en estéreo, el usuario recibirá los sonidos con cierta información de posición, el cual ayudará a aumentar el sentimiento de presencia del usuario.

Por último, en el aspecto motor, el usuario moverá los brazos como si realmente estuviera tocando la batería y por tanto aumentar la presencia haciendo cómplice al usuario de cómo se toca una batería.

Ilustración 8: vista del simulador ESG Elektroniksystem- und Logistik-GmbH (ESG) [IBE].

Crear unas imágenes cada vez más reales, es el camino que ha seguido este campo y ha hecho posible la realización de aplicaciones como películas de animación, simuladores militares, médicos o civiles, o por ejemplo el gran mundo de los videojuegos cada más diversos.

1.5. **Videojuegos**

A comienzos de la década de los ochenta, los juegos de ordenador se habían consolidado firmemente como una nueva e importante forma cultural de masas. Empezando en el siglo XXI, los juegos han llegado a situarse a la altura del cine en términos de tamaño de mercado e importancia cultural. Mantienen entretenidos a los jugadores con diversas modalidades de control y de respuesta en tiempo real ante imágenes, acciones y sonidos. La tendencia que se sigue en cuanto a las imágenes, es la obtención de gráficos más realistas y detallados, y prácticamente la tecnología tridimensional. [3]

Desde los primeros juegos como el Pong, hasta los videojuegos actuales ha habido un salto vertiginoso. No sólo la calidad gráfica es importante en un videojuego, sino también la capacidad de entretener e introducirte en un mundo virtual y vivir una experiencia gratificante. La inmersión en un videojuego no sólo viene a través de las imágenes, sino que también un control retroactivo (vibración, oposición,..) y una narrativa que haga volar la imaginación. Todas ellas son características que hacen al videojuego un peculiar y fascinante mundo.

Además del utilizar los juegos como instrumento de ocio, investigadores y empresas están desarrollando los videojuegos con otro tipo de aplicaciones, como las educativas, formativas o médicas. [8]

En este contexto se definieron los “Serious Games”, juegos diseñados para un propósito principal, no sólo por pura diversión. Normalmente, el adjetivo "serio" pretende referirse a productos utilizados por industrias como la de defensa, educación, exploración científica, sanitaria, urgencias, planificación cívica, ingeniería, religión y política. [16]

Un juego serio puede ser una simulación con la apariencia de un juego, pero está relacionado con acontecimientos o procesos que nada tienen que ver con los juegos, como pueden ser las operaciones militares, empresariales, medicinales, o de tratamientos. Los juegos están hechos para proporcionar un contexto de entretenimiento y autofortalecimiento con el que motivar, educar y entrenar a los jugadores.

En este proyecto se intenta crear una especie de “Serious Games”. Que el usuario a través de la simulación de una batería musical, sea capaz de aprender o de intuir como se toca un instrumento de estas características, pues no es fácil ni barato practicar con una

batería de música, incluso haciendo referencia a los juegos musicales con sus periféricos específicos.

Ilustración 9: captura del videojuego Mario 64 de Nintendo® . Mario recorre un mundo en 3 dimensiones. [M3C]

1.6. Descripción del problema

El lenguaje no verbal que los humanos utilizamos como una forma de comunicación es bastante compleja. Usando gestos para interactuar con un ordenador puede ser un gran avance, sobre todo, en escenarios dónde los dispositivos de entrada son inservibles. Reconocer y trackear el cuerpo entero de una persona en tiempo real, es posible gracias a cámaras basadas en sistemas de captura. Previamente a Kinect®, estos sistemas utilizaban un traje especial o marcadores especiales, o en otros casos eran varias cámaras utilizadas que eran muy sensibles a la iluminación u oclusiones parciales.

Los avances recientes han impulsado las cámaras de profundidad, las cuales pueden escanear las 3 dimensiones en tiempo real, sin necesidad de usar más de una cámara. Las imágenes de profundidad proporcionadas son independientes de las condiciones lumínicas y variaciones en la apariencia, como por ejemplo, la ropa. Las

cámaras basadas en el tiempo de vuelo (*ToF*), requieren un hardware muy complejo y muy caro. [11] [15] [18] Con la aparición de Kinect® por parte de Microsoft®, empezó un movimiento por todo el mundo adaptando este dispositivo para la realización de todo tipo de proyectos diferentes, al original planteado por Microsoft® y Xbox360. La versatilidad de este dispositivo da pie a crear innumerables soluciones a problemas como navegación en robots, telepresencia, realidad aumentada, etc. [12]

La propuesta inicial que se planteó, fue reconocer el cuerpo del usuario para realizar alguna acción en el ordenador. Para darle más valor al reconocimiento de cuerpo a través del dispositivo Kinect®, se incorporó la idea de utilizar la realidad virtual para aumentar la inmersión e interacción del usuario en el sistema. Teniendo en cuenta que Kinect® aumenta los grados de libertad (*DoF*) del usuario para realizar cantidad de movimientos, se decidió desarrollar un prototipo de juego musical, donde el usuario final pueda tocar una batería de música virtual, simulando los sonidos al reconocer movimientos en los brazos.

Este desarrollo tiene una serie de hitos que hay que resolver y explicar: cómo funciona el dispositivo y los algoritmos utilizados para el reconocimiento del cuerpo, así como la creación de la escena virtual, y la solución para incluir música en la solución. Se especificaron los siguientes requisitos que definen la aplicación

1.6.1. Requisitos

- Reconocimiento de cuerpo
- Reconocimiento de movimientos de tocar la batería
- Permanecer con una persona siempre
- Reproducir música con las percusiones
- Música en estéreo

1.6.2. Requisitos no funcionales

- Funcionar con todo tipo de iluminación
- Funcionar con OpenNI y Windows
- Tiempos de respuesta adecuados

2. OBJETIVOS

La funcionalidad del sistema es la de ser capaz de reconocer una acción que hace un usuario delante del dispositivo, y crear una respuesta interactiva, de tal manera que se consiga una interacción entre el usuario y el sistema. Esto dará la posibilidad de investigar y desarrollar una aplicación con la librería OpenNI, responsable del control del dispositivo, y del manejo de la información captada por éste.

Por otra parte, el sistema tiene que ser capaz de capturar imágenes del dispositivo, y así poder conseguir información adicional, para mejorar la interacción del sistema.

Se experimentará con la librería OpenCV, donde se desarrollará algunos métodos de análisis y tratamiento de imágenes, para complementar la información aportada por el módulo anterior.

El sistema tiene que reproducir sonidos. Por tanto, determinar qué pasos son necesarios para escribir una aplicación que utilice OpenAL, para llevar a cabo operaciones básicas de procesado de audio digital, y conseguir una inmersión más realista.

Una vez determinado el reconocimiento del cuerpo y la reproducción de música, se incorporará una interfaz gráfica en un motor 3D para realizar una aplicación más inmersiva. Para ello se utilizará OpenInventor, en el cual se realizará un entorno 3D en el que se desarrollará la aplicación, de tal modo, se consigue un efecto realista.

Antes de mostrar la descripción y el resultado del proyecto, se explicarán a continuación una serie de conceptos como antecedentes de la aplicación que se deberán tener en cuenta para comprender completamente de lo que se está tratando en el proyecto.

2.1. **Batería de Música**

En esta aplicación se tiene como objetivo simular una batería musical. El usuario podrá simular el movimiento de los brazos para reproducir sonidos como si estuviera tocando el instrumento.

Una batería se trata de un conjunto de instrumentos musicales, todos ellos de percusión, usado para muchas agrupaciones musicales. En otras ocasiones se refiere al músico que toca estos instrumentos.

Los instrumentos de percusión están considerados como los más antiguos de los instrumentos musicales junto a los de viento. El origen de la batería radica en 1890 en la unión de unos cuantos instrumentos: los tambores y los timbales, que surgen de África y China, los platillos, que derivan de Turquía y también de China, y el bombo, de Europa. [19]

En el siglo XIX los músicos románticos comenzaron a utilizar grupos («baterías») cada vez más grandes, que fueron utilizados a principios del siglo XX. Antes de que todos los instrumentos fueran unidos eran tocadas por varias personas (entre 2 y 4), cada una de las cuales se encargaba de alguno de los instrumentos de percusión. Durante la Primera Guerra Mundial afectó a la alta burguesía, que solía contar con pequeñas orquestas privadas, y se vieron obligados a reducir el número de músicos, y en muchos casos éstos, sobre todo los percusionistas, aprendieron a tocar varios instrumentos a la vez. Fue con la invención del pedal de bombo (primero, de madera; después, de acero), en 1910, por parte de Wilhelm F. Ludwig, cuando se permitió que casi toda la percusión pudiera ser tocada por un solo músico.

Una batería está compuesta por un conjunto de tambores, comúnmente de madera, cubiertos por dos parches, uno de golpeo (en la parte superior) y otro resonante (en la parte inferior). Estos tambores pueden variar su diámetro, afectando al tono, y la profundidad, variando la sensibilidad sobre el parche de resonancia. Además la batería también es acompañada por los imprescindibles platos, y otros accesorios (tales como cencerro, panderetas, bloques de madera, entre otros). La batería se puede afinar con una llave de afinación y un sistema de capachos o lugs y pernos de afinación que tensan el parche, así mientras más tenso esté el parche, más agudo será el sonido, y viceversa.

De tal forma, la batería que se incluye en la escena, será una visión simplificada de una batería de verdad y sólo se reproducirán 5 sonidos:

- Dos cajas:

A diferencia de los otros tambores, la caja posee una bordonera o conjunto de alambres que, colocada en contacto con el parche de resonancia, produce su vibración por simpatía y el característico sonido a zumbido de la caja. Su función es marcar los compases, lo que no impide que se use libremente logrando cambios en la marcha y/o contratiempos.

- Un Tom tom:

Generalmente suelen ir montados sobre el bombo, pero si se usan más de dos, llevan soportes adicionales muchas veces los toms van sobre los pedestales de los platillos.

- Un Ritmo:

Se trata de un platillo grande, muchas veces se usan para llevar el ritmo en sustitución del hi-hat.

- Un Hi-hat:

Es un sistema que consta de 2 platillos instalados en un soporte con pedal que permite que uno caiga sobre el otro haciéndolos sonar. Los Hit hats se puede tocar cerrado y abierto, usando el pedal.

Para reconocer los movimientos del usuario se utilizará el Kinect® que es un tracker. A continuación se explicará que es y cómo funciona un tracker, y en especial Kinect®.

2.2. **Trackers**

Para saber la posición de nuestra cabeza, se usa un dispositivo de seguimiento, en concreto Kinect®.

Los dispositivos de seguimiento/rastreo (en adelante trackers) están presentes en todos los sistemas de realidad virtual. [DAC] Dan información al ordenador sobre la posición y orientación del usuario (o alguna parte de su cuerpo: mano, cabeza...). Estos trackers se caracterizan por tener 6 DOF (6 *Degrees Of Freedom*). Son capaces de detectar 6 grados de libertad, es decir que detectan la posición mediante las coordenadas (x, y, z) y un grado de giro (orientación) en cada uno de los ejes: yaw, pitch,roll.

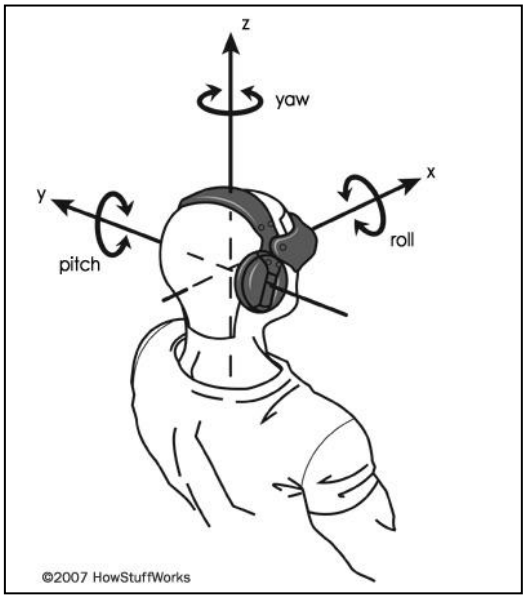


Ilustración 10: Los 6 grados de libertad [DAC]

Se pueden destacar varias características comunes a todos ellos:

- Latencia:

Tiempo transcurrido entre que se produce un cambio en la posición y/o orientación del objeto y el momento en que se informa al motor de realidad virtual de este cambio. Si este tiempo es lo suficientemente pequeño, no será perceptible para el usuario.

- Precisión:

Diferencia entre la posición real del objeto y la que proporciona el sistema de seguimiento.

- Repetibilidad:

Número de medidas que proporciona el tracker cuando el objeto se encuentra en reposo.

- Resolución:

Magnitud del mínimo cambio detectable por el sistema de seguimiento.

- Tasa de medida:

Número de medidas por unidad de tiempo que el sistema de seguimiento proporciona al ordenador.

- Ruido:

Variaciones en las medidas de la posición del objeto cuando este se encuentra quieto.

En general los trackers están compuestos por un dispositivo que genera la señal, un sensor que la recoge y una unidad de control que la procesa y la envía al ordenador. En algunos sistemas de seguimiento, llamados inside-out, los emisores se encuentran repartidos en puntos fijos del entorno virtual, mientras que los sensores se sitúan en el usuario. También existen sistemas outside-in, en los que son los emisores los que lleva el usuario y los receptores se encuentran en el entorno.

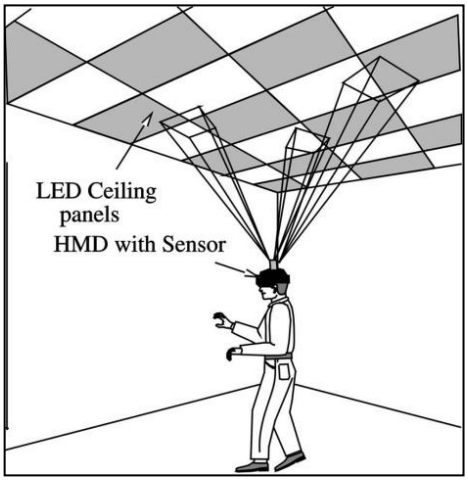


Ilustración 11: Sistema inside-out [DAC]

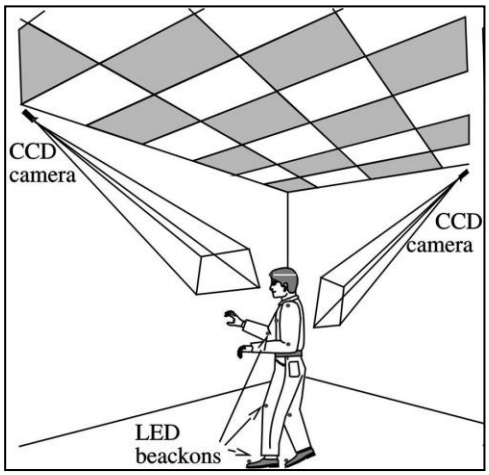


Ilustración 12: Sistema outside-in [DAC]

2.3. Tipos de Trackers

Se pueden encontrar distintos tipos de sistemas de rastreo en el mundo de la realidad virtual: [DAC]

- Trackers Mecánicos:

Están basados en una conexión física entre el objetivo del seguimiento y un punto fijo. Normalmente el objeto sobre el cual se quiere medir su posición y orientación se encuentra en el extremo de un brazo articulado. Estos sistemas proporcionan una latencia muy pequeña, pero tienen el inconveniente de la limitada movilidad y el peso, que los hace menos manejables.

Ilustración 13: Phantom Omni [DAC]

- Trackers Electromagnéticos:

Estos sistemas miden los campos magnéticos generados por un transmisor fijo para averiguar la posición de un objeto receptor. Para ello hacen uso de la triangulación, empleando 3 emisores y un número variable de receptores. Normalmente proporcionan tiempos de latencia muy bajos, pero como contrapartida, son sensibles a verse interferidos por cualquier objeto que pueda crear un campo magnético.

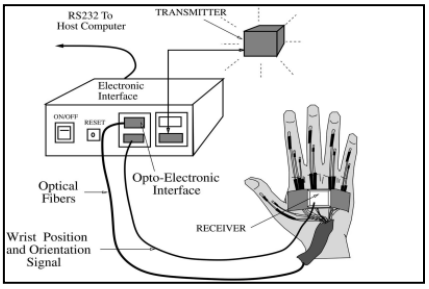


Ilustración 14: Tracker magnético sobre un guante [DAC]

- Trackers Ultrasónicos:

Utilizan ultrasonido producido por un transmisor fijo para determinar la posición y orientación del elemento receptor. Están basados en triangulación: existen tres emisores de sonido fijos y en el receptor, que es triangular, se encuentran 3 micrófonos. Al igual que los magnéticos, se pueden ver afectados por la interferencia de otros sistemas que utilicen ultrasonido. Además, suelen proporcionar un ratio de actualización bastante baja.

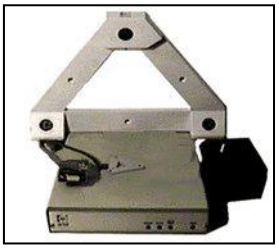


Ilustración 15: Tracker ultrasónico de Logitech [DAC]

- Trackers Inerciales:

Usan las propiedades físicas asociadas al movimiento para detectar la aceleración y la rotación de los objetos y así conocer su posición y orientación. La principal ventaja de este tipo de sistemas de seguimiento es que son independientes: no necesitan ningún tipo de fuente o referencia externa para funcionar.

Ilustración 16: Wii Remote y Wii MotionPlus [DAC]

- Trackers Ópticos:

Se sirven de la luz para conocer la orientación y posición del objeto. Frente a los anteriores sistemas, este proporciona un mayor ratio de actualización y latencia menor.

El emisor suele consistir en una serie de LED's infrarrojos y los sensores son cámaras repartidas por el entorno que detectan infrarrojos. Estos sistemas pueden verse afectados por la luz del ambiente u otra radiación infrarroja, y requiere que haya suficiente luz y cámaras alrededor del escenario donde se encuentra el objeto a seguir.

También se usan cámaras RGB para detectar caras, personas, objetos, y ser capaces de seguirlo por los distintos frames en los que está presente. Los ejemplos más claros son seguimientos de vehículos, seguimientos de personas en aeropuertos, o control de calidad en cadenas de producción.

Otra manera son los sistemas laser, capaces de generar un mapa de profundidad sobre el cual aplicar los distintos métodos de tracking.

En el mundo del entretenimiento, Nintendo® fue el primero en incluir un tracker en una consola. El WiiMote de Nintendo®, se basa en una cámara de infrarrojos en el mando (activo) la cual busca los leds de la barra horizontal (pasivo) que incluye la consola Wii. Para saber su posición triangula respecto a estos leds, que además de contar con la información de unos acelerómetros, puede saber la orientación del mando respecto a la pantalla y registrar ciertos movimientos del usuario. Así pues, Nintendo® revolucionó el mercado con los juegos casuales, realizando acciones como cortar, empujar, girar, etc.

Ilustración 17: Wii Remote Infrared Tracking [DAC]

Por su parte, Sony sacó a la venta PlayStation™ MOVE, una versión actualizada de Wii utilizando su PlayStation™ Eye, una cámara RGB con la que se podía interactuar en la consola. En este caso, la cámara es pasiva, y es el usuario mueve mando con una bola de color (llamativo) la cual es capturada por la cámara, para traducirlo a movimientos en el juego. Sony mejora respecto a Nintendo® la precisión respecto a la profundidad.

Ilustración 18: PlayStation™ Move [DAC]

El uso de las cámaras RGB eran más común debido a su bajo coste en diferencia a los sistemas de infrarrojos o láseres. Pero los problemas de iluminación, colores y texturas que sufren las imágenes de RGB eran el gran inconveniente de estos dispositivos. Cuando Microsoft® presentó Kinect®, supuso un salto cualitativo en los trackers ópticos, ya que su tecnología infrarroja era barata y muy robusta. Aunque la principal actividad que realiza el Kinect®, es la detección, reconocimiento y seguimiento de personas, también incluye una batería de micrófonos para realizar grabación de voz.

Ilustración 19: Kinect® Sensor [DAC]

2.4. Dispositivos Kinect y Xtion

El dispositivo más conocido es Kinect® para Xbox 360™ pero existe un dispositivo igual de Asus® llamado Xtion. En adelante se referirá a los dos por Kinect®.

Este es un periférico para videojuegos que prescinde de mandos gracias a un sensor de detección de movimientos. La tecnología hardware de estos dos dispositivos esta creada por la empresa israelita PrimeSense, la cual firmó un contrato de exclusividad con Microsoft®. El software de cada dispositivo es diferente, optando Primesense por aliarse con Asus® para hacer un dispositivo propio.

Ilustración 20: Asus® Xtion y Kinect®

El sensor de Kinect® es una barra horizontal conectado a un pivote, diseñado para estar en una posición longitudinal. El dispositivo tiene una cámara RGB y un sensor de profundidad, que conjuntamente capturan el movimiento de los cuerpos en 3D. [12]

Según las características definidas para los trackers, se especificarán los valores de estas características para Kinect®:

Data Streams (Flujo de datos)

- 640 × 480 a 16 bits de profundidad @ 30fps Cámara Infrarrojos
- 640 × 480 32-bit de color @30fps Cámara RGB

Campo de visión

- Campo de visión horizontal: 57 grados
- Campo de visión vertical: 43 grados
- Rango de inclinación física: ± 27 grados
- Rango de profundidad del sensor: 0,5 – 8 metros
- Rango de precisión: 1mm en el eje Z – 3 mm en el eje X,Y

La disposición electrónica es prácticamente la misma en los dos dispositivos, excepto en dos diferencias. El Asus® no dispone del motor que permite a Kinect® "buscar" el suelo inclinándose hacia delante y atrás. Y la otra diferencia, es que Kinect®

necesita un transformador de corriente para conseguir alimentación adicional, mientras que Asus® solo necesita la conexión USB.

El límite del rango visual del sensor de Kinect® está entre 0.5 y 8 metros de distancia, aunque a partir de 3.5 metros, el error que introduce es importante.

La configuración óptica Kinect® es lo que le permite seguir tus movimientos en tiempo real. Está constituido de dos partes principales: un proyector y una cámara de infrarrojos VGA. El rebote de un haz de láser en todo el campo de juego, es lo que permite que la cámara capte lo que se llama un "campo de profundidad."

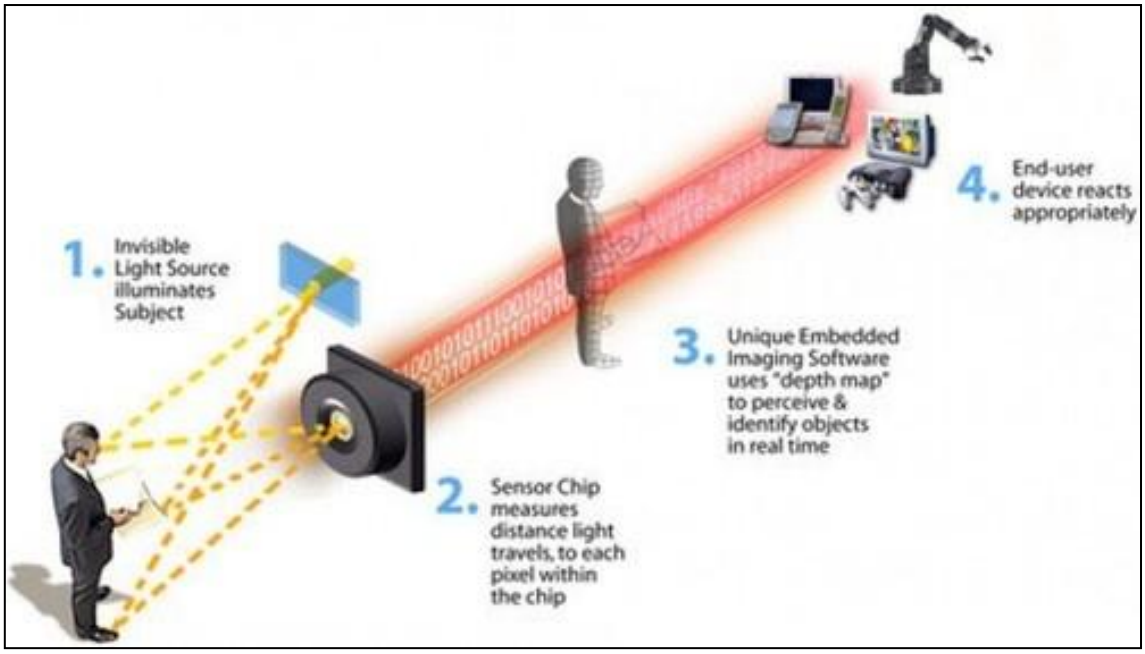


Ilustración 21: Funcionamiento Kinect®[11]

El dispositivo lanza un haz de luz infrarroja con un patrón, como si de una malla se tratase, y con la cámara infrarroja analiza la deformación de ese patrón. Básicamente se trata de que Kinect® recibe una malla con una deformación del patrón, de tal manera, analizando resuelve la distancia a la que están los objetos. Esta operación, según algunas fuentes, se realiza hasta 200 veces por segundo [12], y otras lo cifran en 30 veces por segundo.

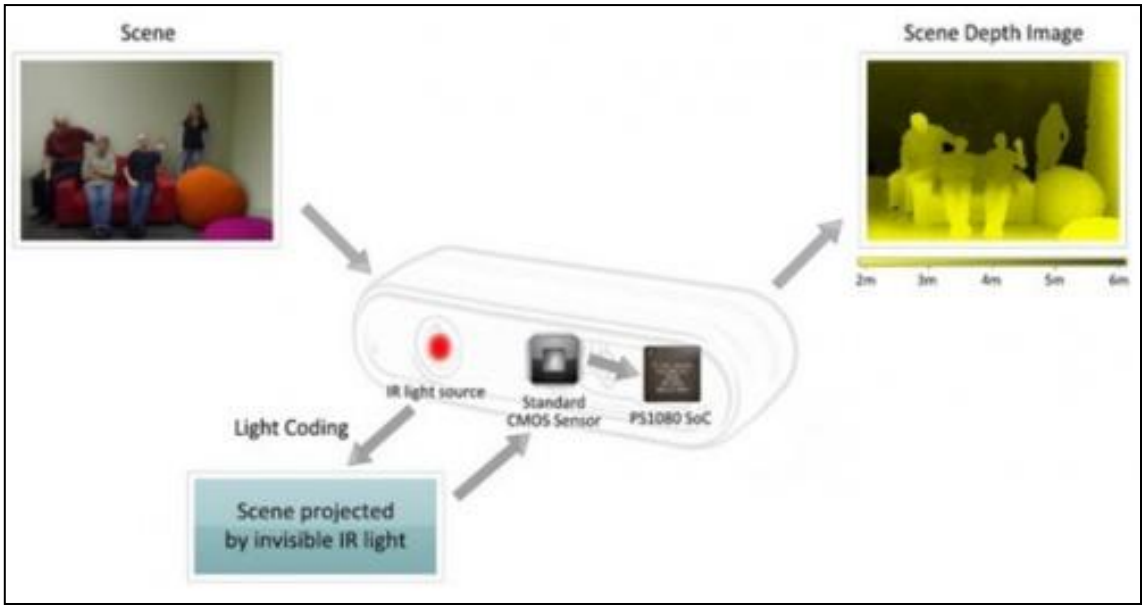


Ilustración 22: Distancia objetos con Kinect®[11]

Una vez la cámara Kinect® sabe a qué distancia están los objetos, es capaz de distinguir el movimiento en tiempo real. Esto se simplifica gracias a la gran recurrida tarea de substraer el fondo almacenado a la imagen para aislar todo lo que no es un humano u objeto en movimiento.

Kinect® puede llegar a distinguir la profundidad de cada objeto con diferencias de 1 centímetro y su altura y anchura con diferencias de 3 milímetros. Este mapa de profundidad es pasado a la GPU de la XBOX360 que lo procesará para hacer el reconocimiento del cuerpo humano, o a la CPU del ordenador en el caso del dispositivo Asus®.

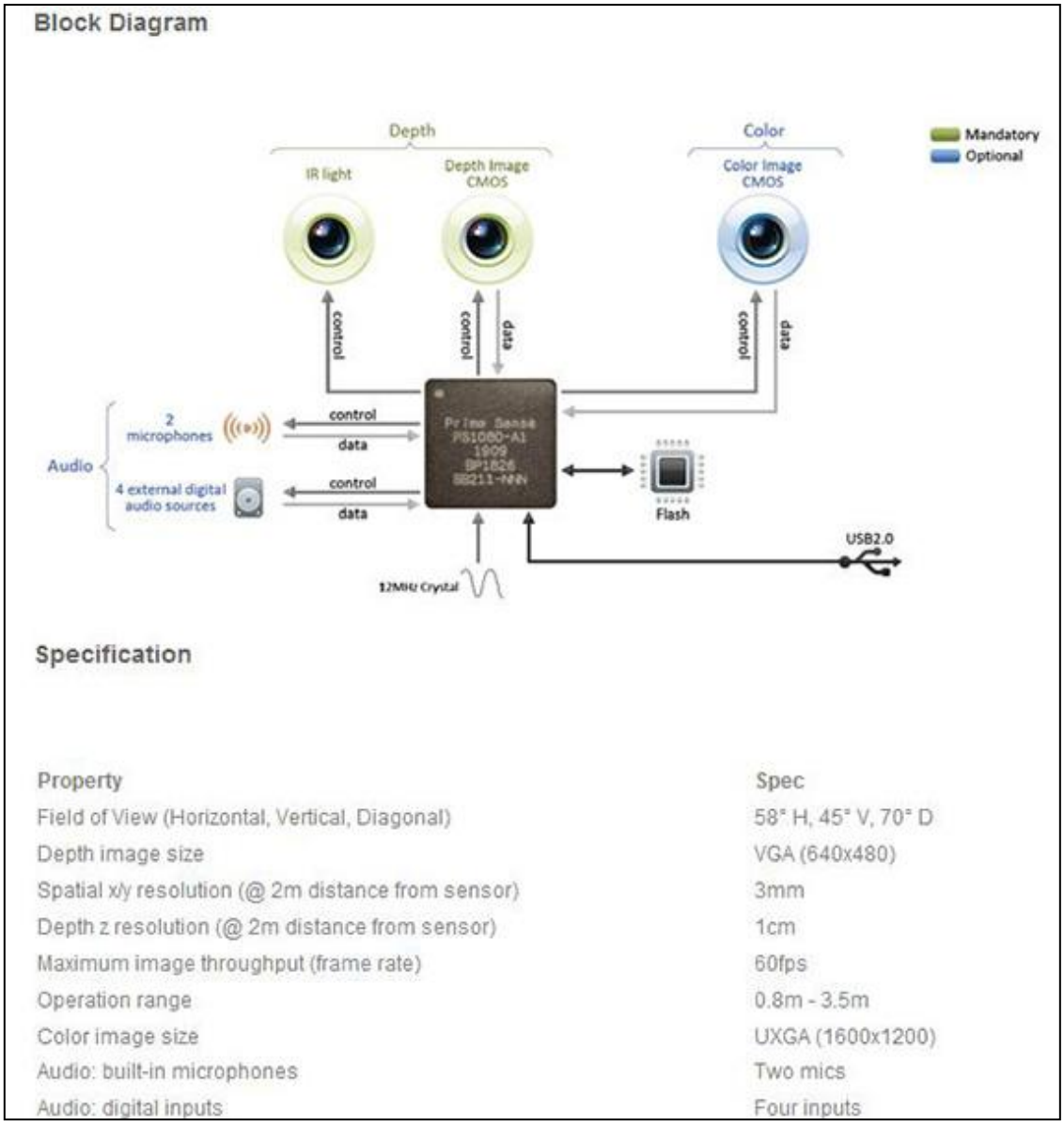


Ilustración 23: Arquitectura y Especificaciones Kinect® [11]

Como comentario adicional, al tratarse de una cámara como las demás, tiene muchos problemas al haber rayos de luz directa sobre la cámara. Por tanto, estos dispositivos están pensados para utilizarse en interiores.

A continuación se mostrará cómo a partir de la imagen de profundidad, se realiza la detección de un cuerpo humano.

2.5. **Algoritmo de detección de cuerpo**

La literatura sobre la detección de cuerpos humanos es extensa, tanto en imágenes RGB como en imágenes de profundidad.

Los problemas de iluminación, colores y texturas que sufren las imágenes de RGB han dejado claro el predominio de las imágenes de profundidad en cuanto a reconocer cuerpos se refiere.

La introducción y detección de un humano en la escena puede ser de dos formas. Ambos algoritmos, detectan un cuerpo que se introduce a la escena gracias a la substracción del fondo anteriormente comentado, una vez detectado el cuerpo, el sistema tiene que definir las partes de un cuerpo humano: cabeza, brazos, piernas…

Kinect® usa un algoritmo que puede reconocer al humano "al vuelo" y PrimeSense usa un algoritmo que necesita una pose concreta para reconocer al humano, llamada T-Pose.

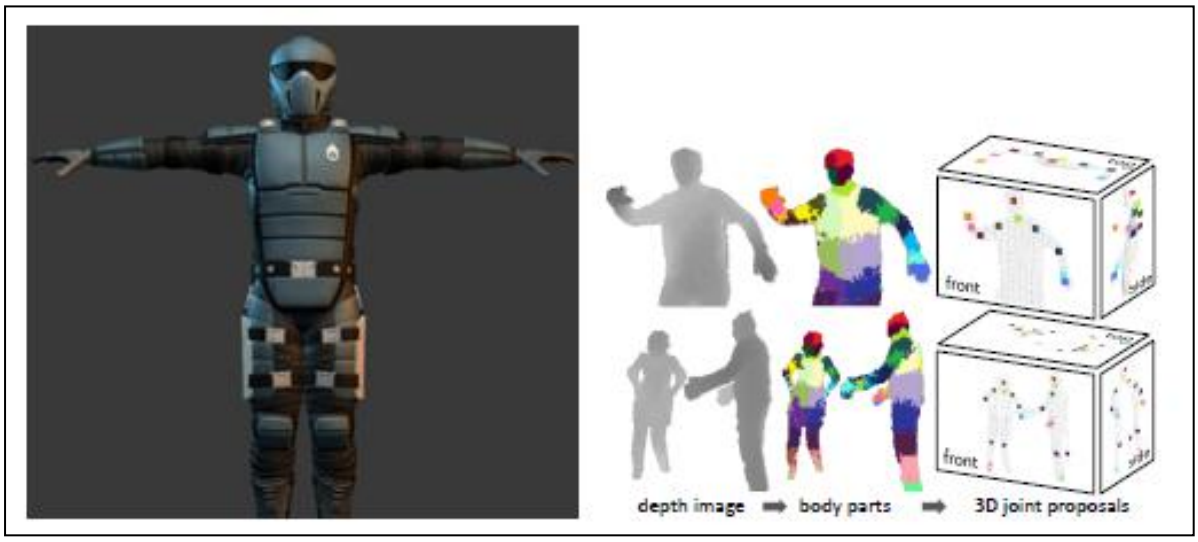


Ilustración 24: T-Pose (izquierda) y Reconocimiento al vuelo (derecha)

A continuación se pasa a explicar más detalladamente cada uno de los algoritmos.

2.5.1. Algoritmo de Kinect®

El Kinect® manda la imagen de profundidad y la Xbox lo pasa a través de una serie de filtros para que se puede calcular lo que es una persona y lo que no. El sistema sigue un sistema de directrices, como “una persona tiene dos brazos y dos piernas”. [12]

Kinect® está precargado con 500 mil poses comunes de personas. Estas poses forman una base de datos, dónde se almacenan personas bailando, conduciendo, etc. No se almacenan todos los frames de estas poses, sino una muestra de ellas. Se pueden asumir los espacios entre ellas y poder rellenar en los casos que la cámara sólo vea parcialmente a la persona. El único inconveniente es que los dedos no se asignan de forma individual sobre el esqueleto.

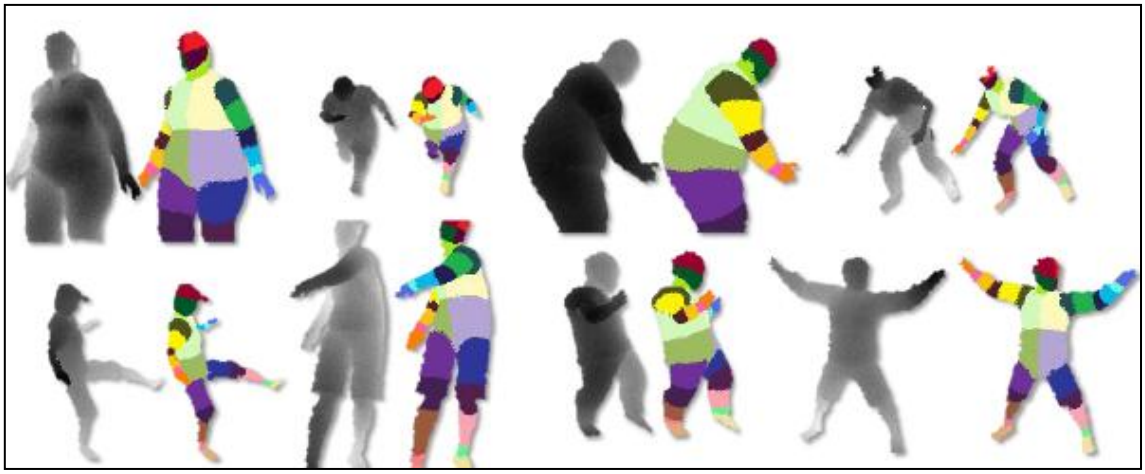


Ilustración 25: Algunas poses de la base de datos [12]

Se compara el cuerpo que se ha recortado de la escena (escalado o no) y la base de datos, y se hace una aproximación etiquetando unas 31 partes del cuerpo. Esta aproximación con las distintas etiquetas se pasa por un árbol de decisión previamente entrenado, para determinar la situación exacta de cada etiqueta.

Este árbol de decisión fue entrenado con una base de datos de imágenes de profundidad, previamente etiquetado a mano. Este árbol tiene una profundidad de 20 niveles. Este entrenamiento con un millón de imágenes, tarda un día en un cluster de 1000 cores.

Una vez determinada la situación exacta de cada parte del cuerpo, el algoritmo tiene que decidir los puntos exactos de las articulaciones, que se llaman puntos de unión. Estos puntos de unión se detectan con una variación de la media ponderada. Así, ya se

tienen los puntos interesantes de la persona que está enfrente de la cámara. Esos puntos son los que se comunican a las aplicaciones de la Xbox como puntos de control.

Ilustración 26: Cálculo de los puntos de unión [12]

La precisión de este método es muy alta, generan un algoritmo bastante robusto y rápido para utilizarlo en aplicaciones tan exigentes como los videojuegos. Decisiones como entrenar los 3 filtros offline, y tener a disposición una cantidad enorme de dinero y recursos como es Microsoft®, ha dado como resultado un porcentaje de acierto en el reconocimiento humano (de pie) sea del 90%.

Hay decisiones curiosas como: identificar el suelo en el reconocimiento de una persona aumenta el éxito del reconocimiento, aceptar escalas de las figuras de la persona, aumentar el tamaño del árbol de decisión (con su consecuencia de coste en cuanto a memoria en el momento de entrenarlo), por ejemplo.[12]

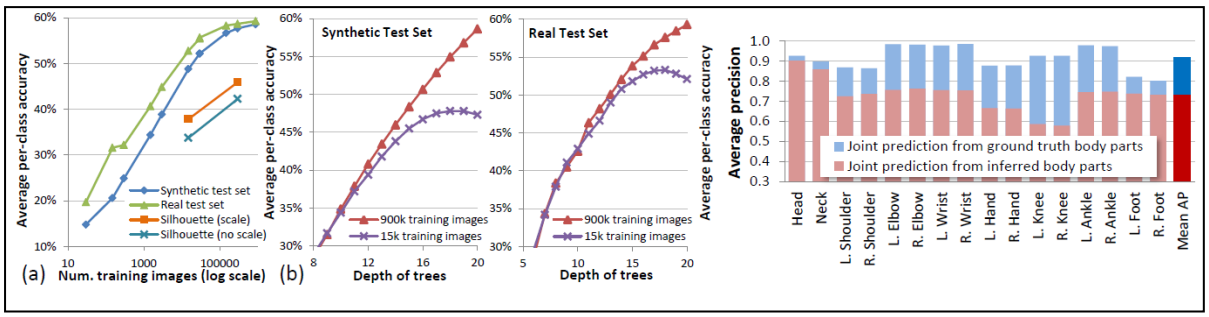


Ilustración 27: Gráficas de memoria [12]

2.5.2. Algoritmo de PrimeSense

PrimeSense tiene una calibración previa al reconocimiento del cuerpo. Para hacer la calibración, la persona se tendrá que mantener en una posición determinada.

El primer paso es el mismo que con Kinect®, una vez que se tiene substraído la persona del fondo, el sistema está comprobando si la persona se pone en la posición de calibración. [18]

Esto lo hace al aproximar la mancha que hay en la imagen de profundidad que corresponde con la persona, intenta encajar un círculo para la cabeza y un rectángulo para el torso, de esta manera puede aproximar el tracking de una persona.

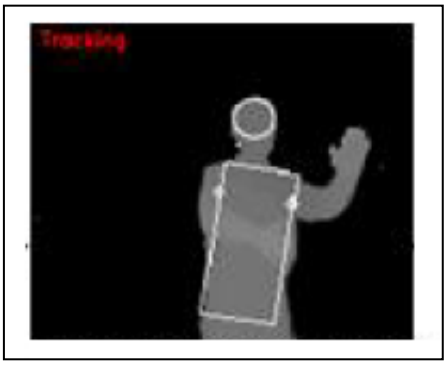


Ilustración 28: Tracking de cabeza y torso[18]

Con esta aproximación, está comprobando si la persona se posiciona en la pose predeterminada. Una vez se detecta la posición de pose predeterminada, el sistema sólo comprueba la figura con un patrón ya definido. Etiqueta todas las partes del cuerpo como el algoritmo anterior, y calcula los puntos de unión de la figura.

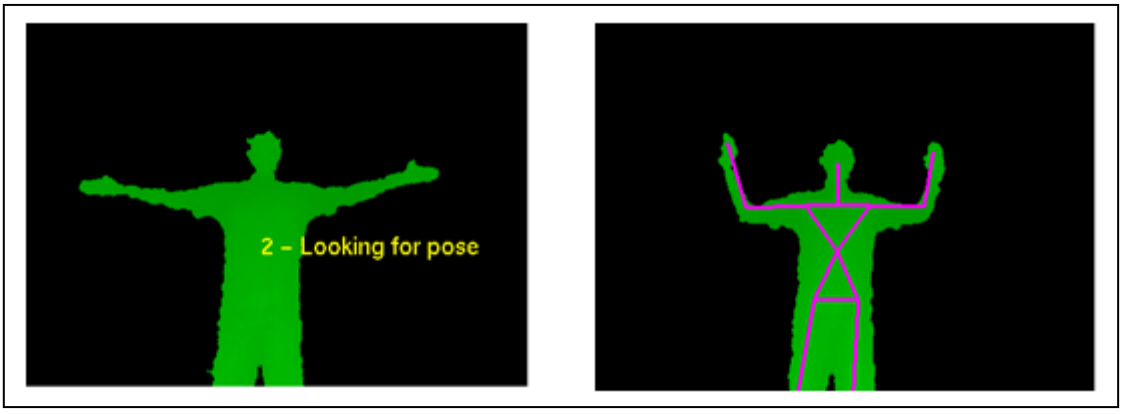


Ilustración 29: Reconocimiento de la pose[11]

La precisión es igual de alta que el algoritmo anterior, ya que fuerza a la persona a ajustarse a un modelo inicial predeterminado. Este algoritmo tiene una ventaja sobre el anterior, ya que no hace falta que se vea el cuerpo entero, y por tanto no tiene los problemas de distancia que tiene el Kinect®. [12]

2.5.3. Algoritmo actualizado

Durante la realización del presente proyecto, PrimeSense, desarrollador de OpenNI y del procesador del Kinect®, actualizaron el algoritmo del reconocimiento del esqueleto de un usuario. Con la nueva actualización ya no es necesario realizar una postura determinada delante del Kinect®. Así los controladores del dispositivo realizan una calibración "al vuelo" para ser capaz de reconocer el esqueleto del usuario. Aunque PrimeSense no ha publicado ninguna publicación respecto al nuevo algoritmo, es una modificación de un algoritmo defino de en [15].

En esta publicación se basan en la medida geodésica que se puede realizar en superficies con volumen 3D. Kinect® ofrece una imagen 3D de la superficie de la persona que permite usar la distancias geodésicas entre diferentes puntos del cuerpo. Mientras que la distancia euclidia de dos puntos en un espacio 3D puede variar significativamente con el movimiento del cuerpo, la distancia geodésica está definida a lo largo de la superficie del cuerpo. Así, la distancia geodésica de dos puntos del cuerpo, por ejemplo el centroide y un punto extremo, puede asumirse constante independientemente de la postura del cuerpo como se cita en: [15]

Se puede, por tanto, extraer puntos claves del cuerpo que mantengan la distancia geodésica, para buscar puntos con distancias que corresponden a las actuales medidas de una persona. Usando sólo la información sobre la imagen de profundidad pueden determinar las distancias entre puntos, pues se conocen prácticamente todos los puntos en coordenadas 3D.

2.6. Aplicaciones Previas con Kinect®

Desde que se publicó Kinect®, como se ha comentado en anteriores apartados, surgió una inmensa comunidad de desarrolladores que desarrollaron multitud de aplicaciones fuera del framework de Xbox. En internet se les denomina KinectHack y tienen una comunidad con una página que agrupa todos estos desarrollos. Previamente a este proyectó, se desarrolló varios prototipos antes de proponer la aplicación que se ha planteado para este proyecto. El primero fue utilizar la mano como mouse del ratón con el Kinect®. El segundo, se intentaba crear un efecto 3D creando una aplicación que simulara una ventana en la pantalla del ordenador.

2.6.1. Mouse controlado con la mano

Aunque OpenNI, que se detallará más adelante, proporciona los controladores y ciertas operaciones con el dispositivo, no proporciona un control directo sobre las manos. Se creó una aplicación que aprovechaba las virtudes de los dispositivos infrarrojos para hacer un seguimiento de la mano y observar si la mano se cerraba o no, y así simular el comportamiento del ratón.

El procedimiento era el siguiente: se averiguaba dónde se situaba la mano en el mapa de profundidad. Una vez se sabe su posición, se segmentaba la imagen de profundidad, quitando aquellos puntos 10 centímetros por detrás de la mano, y 10 por delante. Además, se rodeaba la mano con un círculo con centro en el centro de la mano, y se eliminaban otros puntos que estuvieran a la misma distancia que la mano. Así, se conseguía tener una imagen segmentada o ROI (*region of interest*) que se viera únicamente la mano. Con esta información ya se podía mover el ratón del ordenador.

Ilustración 30: Captura de pantalla ejecutando la aplicación

Para conocer si la mano estaba cerrada o abierta, se utilizó una operación matemática llamada cierre convexo y la relación con el área. Se llama cierre convexo de un conjunto de puntos del plano al menor conjunto convexo que lo contiene. Es decir, si la mano está cerrada, el cierre convexo se ajusta muy bien al contorno de la mano. Pero si la mano está abierta, el cierre convexo no se ajusta, y contiene también las áreas que hay alrededor de los dedos. Por tanto su área respecto al área de la mano difiere bastante. Así pues, se puede saber cuando el usuario ha cerrado la mano o abierto delante del dispositivo. [4]

Para hacer estas operaciones matemáticas, OpenCV dispone de métodos para calcular estas operaciones en imágenes.

2.6.2. Simulación Ventana

Uno de los temas interesantes es la estereoscopía o tecnología 3D, así se ideó una aplicación que demostrara el efecto 3D sin necesidad de tener unas gafas. Además, se diseñó una aplicación que simulara una ventana, es decir, que a través del monitor del ordenador, se viera un mundo virtual, en este caso una cocina, que variara la perspectiva según en la que se encuentre el usuario respecto al monitor. [3]

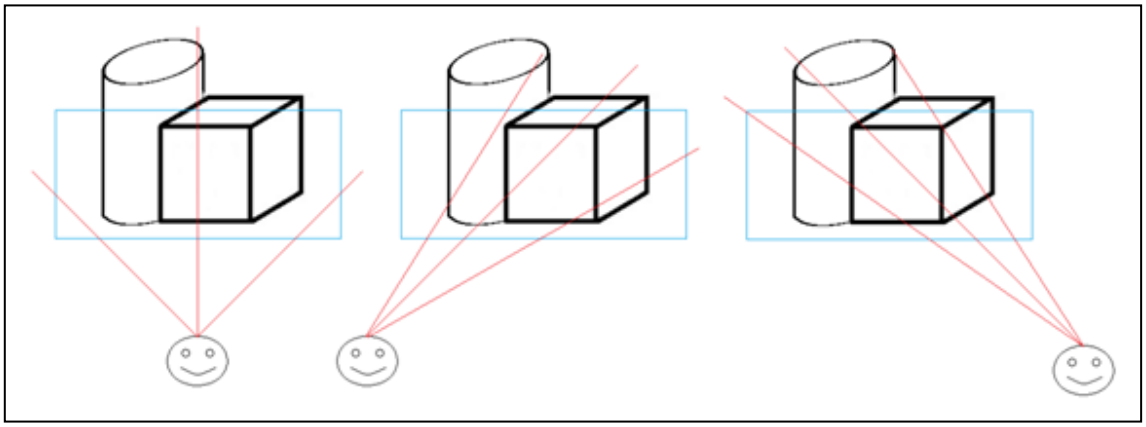


Ilustración 31: Gráfico que representa la correspondencia entre la escena y la posición del observador

Gracias al tracking del Kinect®, la aplicación era capaz de reconocer la posición del usuario respecto de la cámara y por tanto modificar el plano de proyección que tiene la cámara para crear un efecto de 3D. Esta modificación del plano de proyección es posible gracias a la definición de una cámara Frustum. [1] A continuación se mostrarán

dos imágenes: la primera con la persona a la izquierda del monitor y visible la parte derecha de la cocina, y la segunda, con la persona a la derecha del PC y es visible la parte izquierda de la cocina.

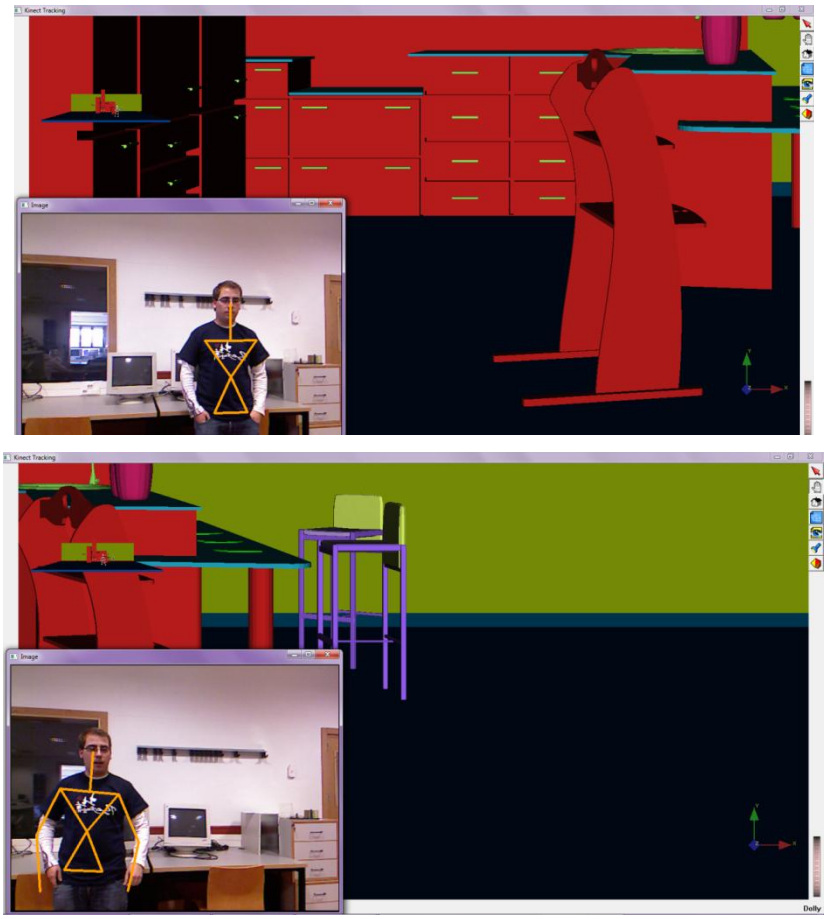


Ilustración 32: Capturas de pantalla de la aplicación en ejecución

2.7. **Metodología**

El objetivo de este proyecto es la investigación y utilización de técnicas tridimensionales y de visualización para crear una aplicación que cumpla los objetivos prefijados. Para ello, se ha seguido una metodología aproximada al Modelo de Cascada:

- Análisis de requisitos:

Una vez definido los requisitos, se hace un análisis de la arquitectura necesaria para crear el sistema, y definir los módulos necesarios para la realización de la aplicación.

Se marcan unos objetivos para ir desarrollando la aplicación de forma ordenada y clara.

- Diseño e implementación:

Una vez obtenido los objetivos, se ha definido la estructura del proyecto a seguir, determinado los pasos sucesivos y acordando una serie de pruebas y ejemplos del cual se elegirá aquel que mejor cumpla las expectativas predefinidas.

Se ha hecho un seguimiento habitual para retocar y refinar la aplicación.

- Pruebas:

Con una versión casi final de la aplicación, se ha seguido una serie de reuniones y pruebas, para intentar mejorar y finalizar el desarrollo de la aplicación.

2.8. Gestión de configuración

La gestión de configuración es el conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo de un Sistema de Información, a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable.

Se ha realizado un control de versión a través de una herramienta de subversión. Subversion es un sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Es un software libre bajo una licencia de tipo Apache/BSD y se le conoce también como *svn*, por ser el nombre de la herramienta utilizada en la línea de comando.

Una característica importante de Subversion es que, a diferencia de CVS, los demás archivos con versionamiento no tienen cada uno un número de revisión independiente, en cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado en el que se está trabajando.

Ilustración 33 : Logotipo de Tortoise SVN

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintas computadoras. A cierto nivel, la posibilidad de que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones, fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada —si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

En este caso sólo hay un usuario, pero para asegurar la calidad de este proyecto, se ha realizado un control de versiones sobre el código y además de la documentación relacionada con el proyecto.

Se ha utilizado el interfaz Tortoise SVN, la cual se incorpora al explorador de Windows. Una vez integrado, se puede realizar las operaciones de subversion tales como: Commit, Update, Checkout... desde el menú desplegable del explorador.

Una vez se que se quiere hacer un commit de los cambios realizados en el desarrollo, la interfaz avisará de los archivos afectados por un cambio, y dará la posibilidad de incluir un comentario para la revisión que se va crear en este momento.

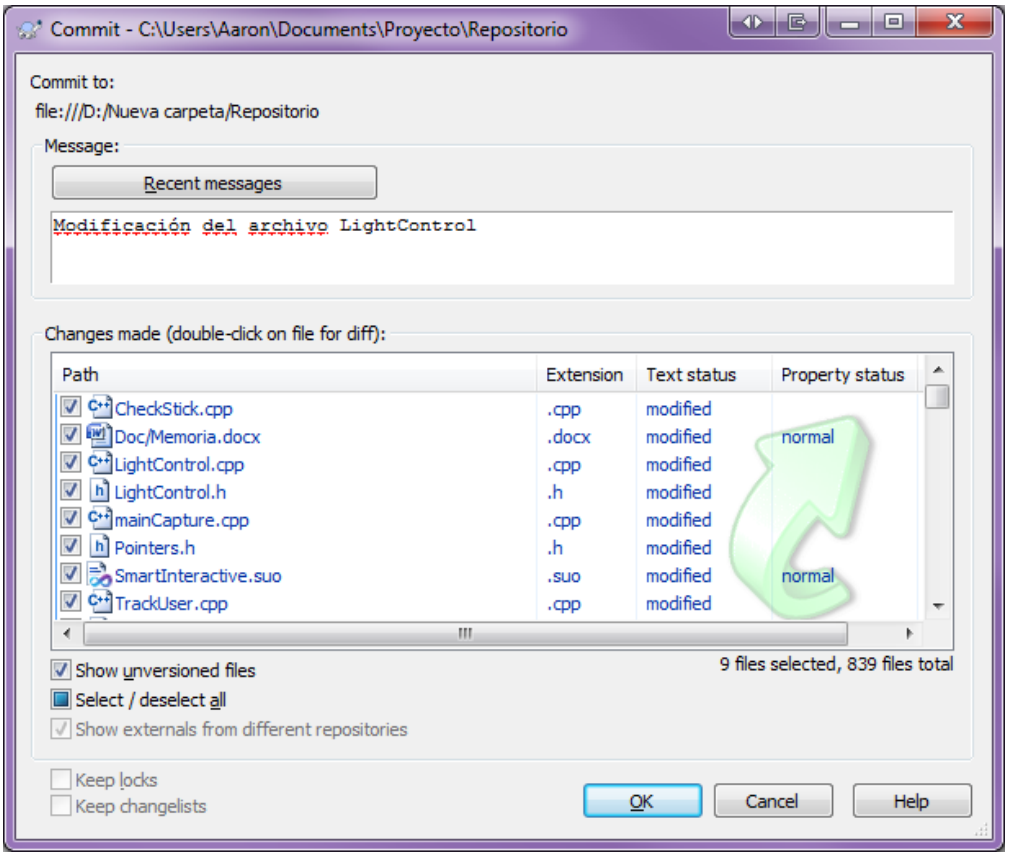


Ilustración 34: Interfaz de Tortoise, cliente de Subversion

Una vez guardado los cambios, se podría volver a un estado anterior, incluso varios atrás. Así mismo, se podría sobrescribir los cambios, o crear una rama nueva, por la cual se desarrolla otra línea base del desarrollo.

2.9. **Documentación**

Al igual que la gestión de configuración es importante para controlar los cambios y garantizar la calidad del software, la documentación que acompaña al sistema de información, en este caso el proyecto, es importante tanto para la calidad y la mantenibilidad del código. Si el código está documentado y todo el sistema está documentado, será más fácil realizar cambios o corregir errores. La herramienta que se ha elegido para realizar esta documentación es Doxygen.

Doxygen es un generador de documentación para C++, y otros lenguajes. Dado que es fácilmente adaptable, funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X. Doxygen es un acrónimo de dox(document) gen(generator), generador de documentación para código fuente. Doxygen recupera toda la información del código del proyecto, recopilando los nombres de las clases, sus módulos, la relación entre ellos y además, información extra que el programador puede incluir. A través de una sintaxis no muy distinta a los comentarios en código tradicionales, se pueden clasificar las clases, se puede explicar ciertas variables o métodos en especial, e incluso omitir alguna parte de una clase. Todo esto sirve para crear una documentación en la cual no hace falta mirar el código directamente y servir de referencia para personas ajenas al desarrollo.

Doxygen al recopilar los datos, es capaz de generar un documento Latex con hipervínculos y también una página web dónde se almacena toda la información. Además es capaz de hacer diagramas de clases a partir del código, incluyéndolo posteriormente en la página web.

Ilustración 35: Página de documentación resultado de esta aplicación

2.10. **Log**

En un sistema informático es imprescindible monitorear la ejecución de la aplicación para resolver o informar de ciertos errores. Ya que si el IDE no está disponible cuando una aplicación tiene un fallo crítico, se necesitan herramientas para subsanar problemas una vez la aplicación ha sido lanzada.

En aplicaciones de gran tamaño incluyen su propio logging o API de trazas, insertando trazas en el código como un método rudimentario de debug. Pero hay situaciones en las que esta opción no es viable debido a casos de aplicaciones multihilo o aplicaciones distribuidas.

La experiencia indica que el logging es un componente importante en el ciclo de desarrollo. Ofrece numerosas ventajas, tales como, proveer un contexto preciso sobre la ejecución de una aplicación, o generar una salida de logging sin la intervención de una persona. Además, esta salida puede ser guardada en medios persistentes para poder analizarlas más tarde.

El Log registra eventos significativos, definidos por el programador, con un nivel asociado que muestra si es crítico o no. De esta manera, los administradores pueden

observar dónde y porque ha fallado una aplicación, filtrando por el nivel que interese en ese momento. Esto hace que Log4cxx sea una herramienta rápida, útil y extensible.

Para esta aplicación se ha usado Apache log4cxx, un framework de logging en C++ derivado de Apache log4j. Log4cxx usa el Apache Portable Runtime para la mayoría del código de plataformas específicas y debería ser usable en cualquier plataforma soportada por APR. Es software libre bajo una licencia de tipo Apache License.

```
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Velocidad
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir altura anterior
TRACE 17/09/2012 19:02:48  MusicVirtual - X=0,355560,Y=-0,802829,Z=-0,625019
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir anterior estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Velocidad
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir altura anterior
INFO  17/09/2012 19:02:48  MusicVirtual - Detectado percusion en
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir anterior estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Manos Comprobadas
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Renderizamos
```

Ilustración 36: Traza del LOG en el bucle principal en modo Verbose

3. DESCRIPCIÓN INFORMÁTICA

Una vez explicado los antecedentes, se procederá a explicar el desarrollo de la aplicación.

3.1. Open Inventor y Coin3D

Open inventor es una librería de objetos y métodos usada para crear aplicaciones gráficas en 3D e interactivas. Aunque está escrita en C++, incluye funciones en C. Open inventor es una serie de bloques que te permiten escribir programas que aprovechen la potencia gráfica del hardware con un esfuerzo menor. Basado en *OpenGL*, la herramienta dispone de una librería de objetos que se puede usar, modificar y extender a gusto [5].

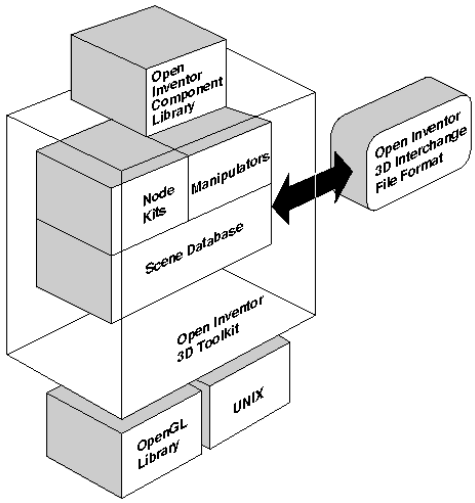


Ilustración 37: Arquitectura Inventor [5]

Se estudiaron diversos motores gráficos para la representación usual de los datos. Entre los más destacados se encuentran Coin3D y OGRE. Se ha descartado OGRE debido a su complejidad y que el objetivo del proyecto es investigar la interacción usuario-ordenador e introducción a la RV.

Coin3D ofrece la economía y la eficiencia de un sistema orientado a objetos. El motor de *Coin* usa la estructura de datos de una escena gráfica. En esta se puede encontrar unos nodos básicos como son figuras, motores, manipuladores u otros nodos como cámaras, luces, que se colgarán del árbol. Además ofrece la facilidad de mover datos

entre aplicaciones con el formato propio de los ficheros 3D. Los usuarios finales pueden guardar y pegar la escena 3D y compartirla por otros programas. Como es sólo un motor gráfico, *Coin* necesita una librería aparte para comunicarse con el usuario. *SoWin* y *SoQt* son unos ejemplos característicos de sus gestores de ventana. *SoWin* está orientado a los sistemas Windows por lo que ha sido elegido para crear la aplicación, aunque es posible su portabilidad de manera sencilla.

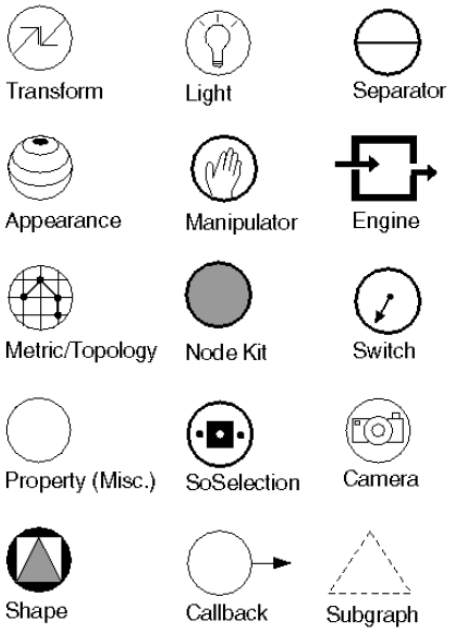


Ilustración 38: Representación de los nodos-objeto que sirven para definir una escena [5]

En el árbol se han colgado los nodos mencionado anteriormente, definiendo toda la escena, que se ha representado en el visualizador y se almacenará en la base de datos de *Coin*. Hay que tener en cuenta el orden en el que se cuelgan los nodos, pues el orden determina las transformaciones a las que se ven sometidas cada uno de los nodos. Estas transformaciones se ven afectadas de izquierda a derecha y de arriba abajo. Por ejemplo, es importante colocar la cámara a la izquierda del árbol, pues verá todo lo que hay a su derecha. Todos los nodos que haya a la izquierda de la cámara no serán representados en la escena.

3.2. **Escena**

Para poder realizar lo que se ha comentado en la introducción, se ha creado una escena (véase Ilustración 39).

En la escena se tiene un nodo SoUnits que dará las unidades con las que se va a trabajar, posteriormente se tiene un nodo SoSeparator que es el ojo derecho y el cuál lleva el peso de la aplicación y finalmente otro SoSeparator que es el ojo izquierdo.

Para los dos ojos se tiene una parte común del árbol que es la escena, en este caso una cocina, y un esqueleto que será el que se actualizará con los datos que proporcione el Kinect®.

Este esqueleto, estará compuesto por los puntos de unión de un esqueleto: cabeza, manos y el torso. Todos estos puntos, son modelos (un modelo en el caso de la cabeza y un modelo en el caso de las manos), afectadas por un nodo transformación que les dará la posición en el mundo. Todos los nodos excepto el de la cabeza, dependen de este, así todos los nodos dependen de la cabeza (el nodo más seguro) y cuelgan de este ante posible pérdida de información. La transformación de la cabeza afecta a todos los nodos, y cada nodo después tiene una transformación propia para definir la distancia que tienen hacia la cabeza.

Para el ojo izquierdo, lo que se quiere conseguir es la visión general desde fuera de la escena (cocina y esqueleto) y para ello la cámara se ha colocado de tal manera que observe toda la escena. Esta cámara actuara como ventana principal y se tenga una vista global del escenario.

Para el ojo derecho, se ha conectado la cámara Frustum para que tenga la misma posición que la cabeza y así conseguir un movimiento más real.

Hay que destacar, que se ha añadido un nodo rotación para que el modelo de la cabeza, rote como si fuera nuestra cabeza, esto es visible en el modo inmersivo. La cámara Frustum en este caso está configurada por defecto.

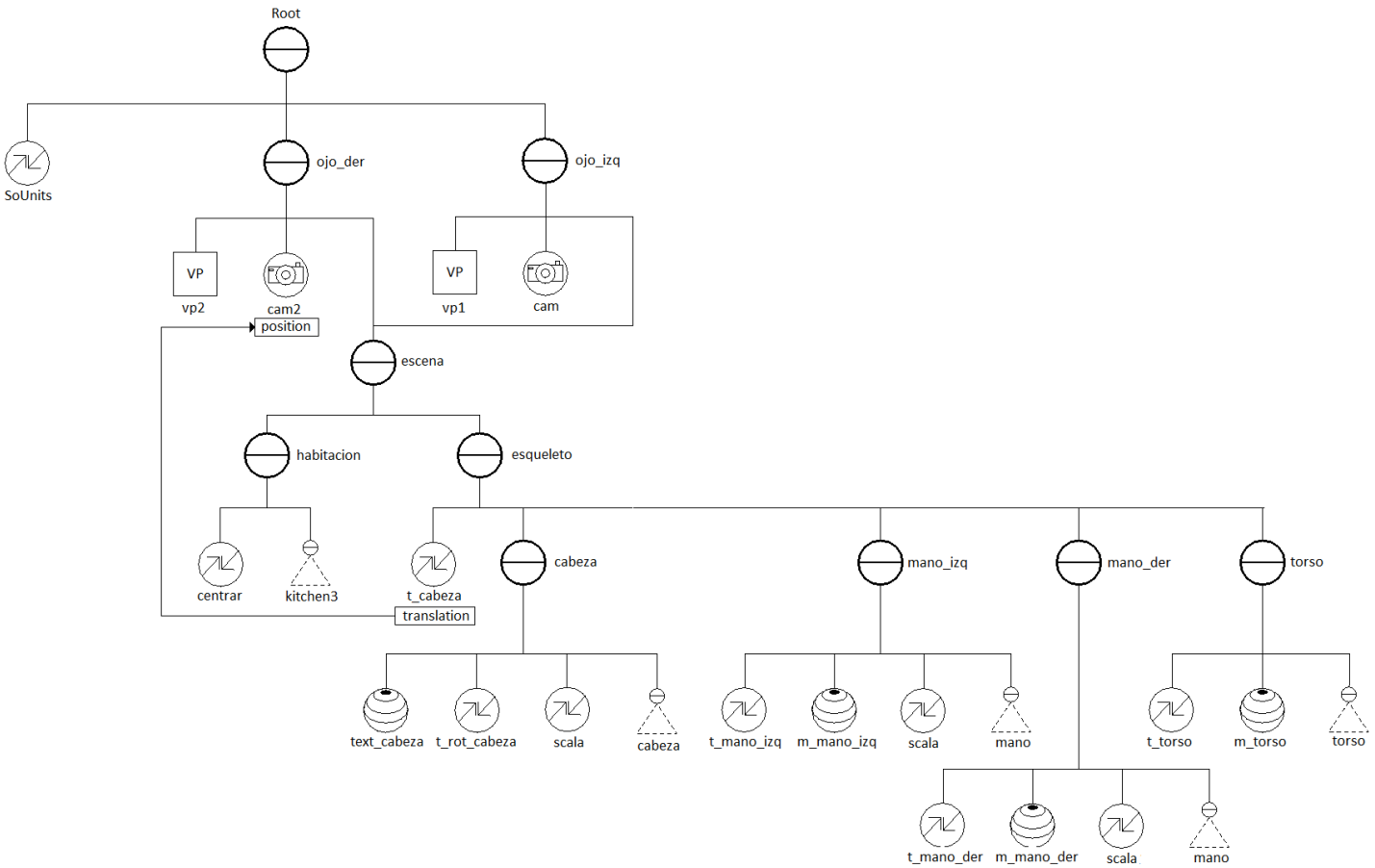


Ilustración 39: Esquema de la escena de la aplicación

3.3. **OpenAL**

En este desarrollo se ha utilizado OpenAL como librería de audio.

Ilustración 40 Logotipo de OpenAL tomados del sitio web.

OpenAL es una interfaz de programación multiplataforma1 para audio multicanal 3D apropiada para el uso en aplicaciones de videojuegos y otras relativas al tratamiento de audio.

En la mayoría de distribuciones de su sistema operativo de elección, se puede encontrar disponible (por ejemplo, en GNU/Linux busca los paquetes que empiecen con 'libalut' y con 'libopenal'), Actualmente en Mac OS X, iPhone, GNU/Linux (tanto para OSS como para ALSA), BSD, Solaris, IRIX, Microsoft® Windows, Xbox, Xbox 360™ y MorphOS.

La librería permite modelizar una colección de fuentes de audio moviéndose en un espacio tridimensional que son oídos por un único oyente en algún lugar de ese espacio. Los objetos básicos en OpenAL son un oyente (*Listener*), una fuente de audio (*Source*) y una zona de memoria (*Buffer*) que contiene la información de audio. Cada *buffer* puede ser asignado a una o más fuentes que representan posiciones (definidas por coordenadas en un espacio tridimensional) de donde "brota" el audio. Siempre hay un oyente, que representa el punto donde se "escuchan" los sonidos que generan las fuentes (*rendering*).

La funcionalidad de OpenAL se estructura en base a estos objetos:

- Una fuente (*source*) contiene un puntero a una zona de memoria (*buffer*), la velocidad, posición, dirección e intensidad del sonido.

- El oyente (*listener*) representa la velocidad, posición y dirección del mismo, así como la ganancia asociada a todos los sonidos. Aunque se pueden definir varios oyentes sólo uno puede estar activo.
- Los buffers contienen audio en formato PCM, en muestras de ocho o dieciséis bits, tanto en monofónico como en formato estéreo.

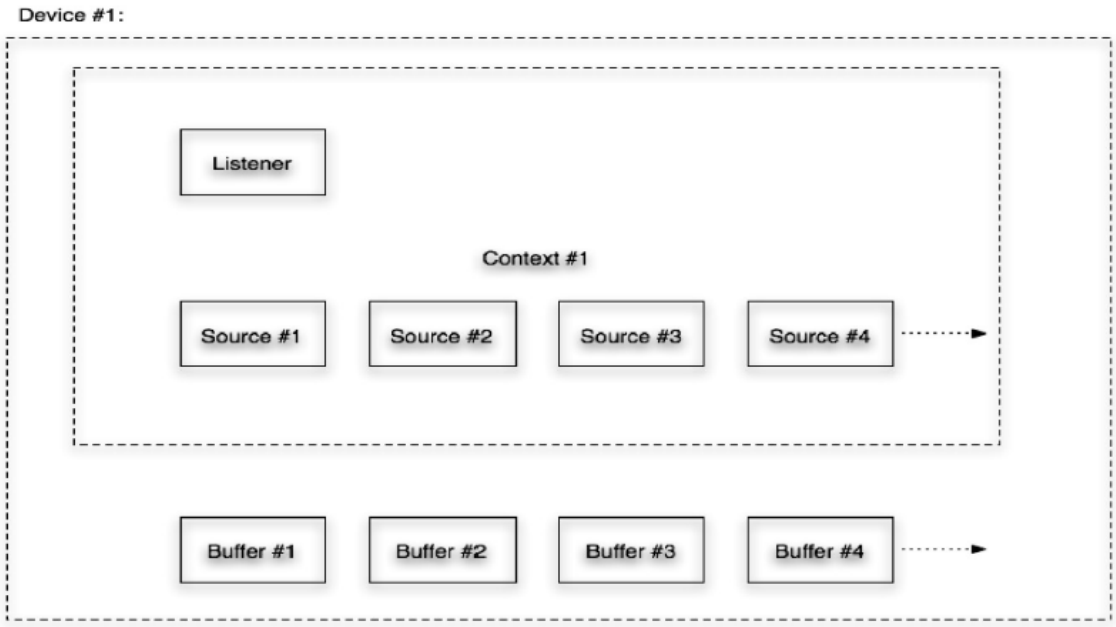


Ilustración 41 Un ejemplo de jerarquía entre objetos de OpenAL

Desde el punto de vista de la utilización de OpenAL en una aplicación, puede abstraerse como un conjunto de instrucciones que permite especificar las fuentes de sonido y el oyente bajo un sistema de coordenadas espaciales en tres dimensiones, junto a órdenes que permiten controlar cómo serán *renderizados* esos sonidos.

En el desarrollo de la aplicación, se ha incorporado la librería para reproducir ciertos sonidos. Para ello, se ha creado una clase que controla todos los aspectos relacionados con la reproducción de música. Esta clase realiza las instrucciones necesarias para inicializar el componente de OpenAL y su comunicación con la tarjeta de sonido. Inicializará las diferentes fuentes de sonido, en este caso, los 5 componentes de la batería musical que tendrán que sonar. Esta librería al permitir renderizar los sonidos según a la distancia que este, se inicializaran los distintos componentes con su posición tridimensional. Se detallará más adelante la clase responsable.

3.4. **OpenCV**

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa, en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi núcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

OpenCV da la posibilidad de realizar ciertas operaciones complejas para el análisis de imágenes. Además, permite enseñar la imagen capturada del dispositivo Kinect® a través de la cámara RGB. En los primeros prototipos, se analizaba la región de interés alrededor de la mano, para poder realizar el cálculo del área del contorno y de la envolvente convexa, y así conocer si la mano está cerrada o abierta. En las siguientes iteraciones del desarrollo, se desechó la idea de centrar la atención en la mano, para centrar la atención en todo el cuerpo. Así, OpenCV sigue en la aplicación para el tratamiento de imágenes y facilitar la visualización de la cámara.

3.5. OpenNI

OpenNI es una interfaz estándar con algoritmos de proceso de datos para sensores 3D. Es una interfaz abierta y estructura genérica pensada para más de un dispositivo. El propósito es definir tipos de datos (mapa de profundidad, mapa de color, pose del usuario, etc.) y una interfaz para modular lo que puedes generar (sensor, algoritmo de reconocimiento de cuerpo), para desarrolladores terceros. [11]

Los desarrolladores pueden escribir aplicaciones que utilicen el soporte de la librería para crear productos con tecnología 3D con ayuda del reconocimiento del esqueleto, puntos de las manos, etc., aunque también da posibilidad a crear un middleware para escribir algoritmos a partir de los datos en crudo, no obstante ya viene distribuido por el productor de los dispositivos. Incluso los desarrolladores de dispositivos pueden implementar la interfaz de su sensor de tal manera que se comuniquen con OpenNI y se pueda utilizar con las aplicaciones existentes.

El propósito principal de OpenNI es proveer una interfaz para aplicaciones que usen una interfaz natural (gestos, poses) de entrada. Las aplicaciones que se desarrollan, se podrán incluir en los packs de venta de estas.

En la siguiente imagen se muestra la arquitectura de OpenNI en la que cada capa representa un elemento:

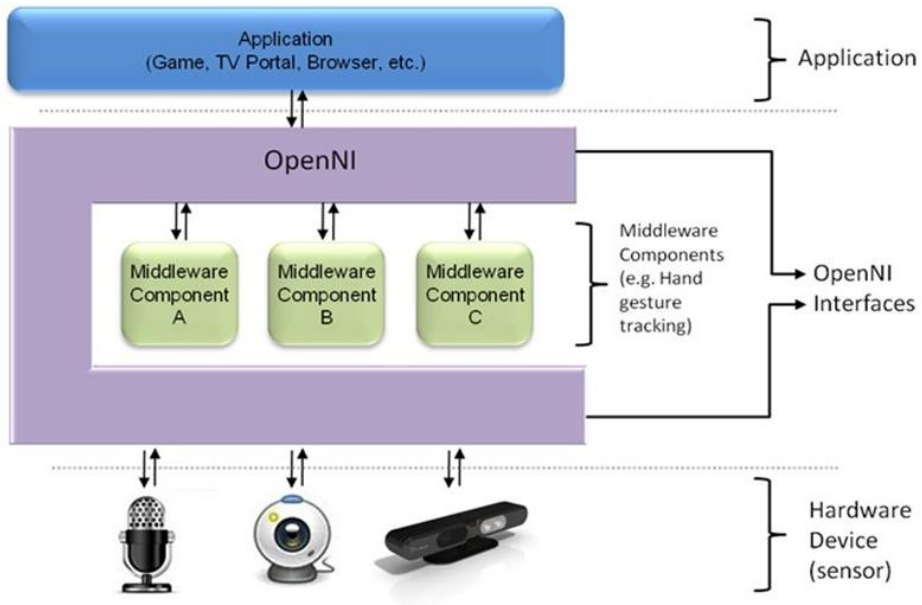


Ilustración 42: Diagrama de la posición de OpenNI[11]

- Top:

Representa el software que implementa aplicaciones con interacción natural de más alto nivel.

- Middle:

Representa OpenNI, facilita una interfaz de comunicación, que interactúan entre los sensores y los componentes middleware, que analizan los datos de los sensores.

- Bottom:

Controla los dispositivos hardware que capturan los elementos audiovisuales de la escena.

OpenNI define "unidades de producción" dónde cada unidad recibe datos de otras unidades, y, opcionalmente, produce otros datos que pueden ser usadas por otras unidades o por la aplicación. Para ordenar estas unidades, se conectan a un grafo, llamado grafo de producción. En la mayoría de los casos, la aplicación sólo está interesada en los nodos más altos del grafo. OpenNI permite a la aplicación usar un único nodo, sin saber los nodos conectados por detrás. Aunque hay posibilidad de acceder al grafo y configurarlos.

3.6. **Aplicaciones utilizadas**

A continuación se comentarán unas aplicaciones relacionadas con el tema que se está tratando. Son aplicaciones que están vigentes en el ámbito de la tecnología 3D. No se pueden comentar todos los programas que se han tomado como referencia en el desarrollo de este proyecto y se comentará AutoDesk 3ds Max.

3.6.1. AutoDesk 3ds Max

Hay que hacer una especial mención a Autodesk 3ds Max. Es un programa de creación de gráficos y animación 3D desarrollado por Autodesk. 3ds Max es utilizado en mayor medida por los desarrolladores de videojuegos, aunque también en el desarrollo de proyectos de animación como películas o anuncios de televisión, efectos especiales y en arquitectura.

3ds Max tiene muchas herramientas para la creación de escenas y modelizar objetos. Se puede empezar con primitivas como cajas, esferas, etc., y modificando sus facetas, y con operaciones como "Extrude", es decir, elevar esta faceta hacia su normal, se puede crear diversas formas. 3ds Max incluye multitud de modificadores, como suavizados, rotaciones, deformaciones, etc., para completar de una primitiva modificada a un modelo de buena calidad.

No sólo eso, 3ds Max permite crear cámaras, luces, animaciones, y muchas otras opciones desarrollado para cada profesión que usa 3ds Max.

En este proyecto, se ha diseñado la escena dentro de 3ds Max. Para no perder tiempo en modelar una batería completa, ya que no es objetivo de este proyecto, se ha buscado en una galería de modelos de Internet. La batería tenía su propio sistema de referencias y escala, la cual se tuvo que ajustar para introducirlo en nuestra escena. Así después de introducir la batería, se procedió a completar la habitación con paredes y el suelo.

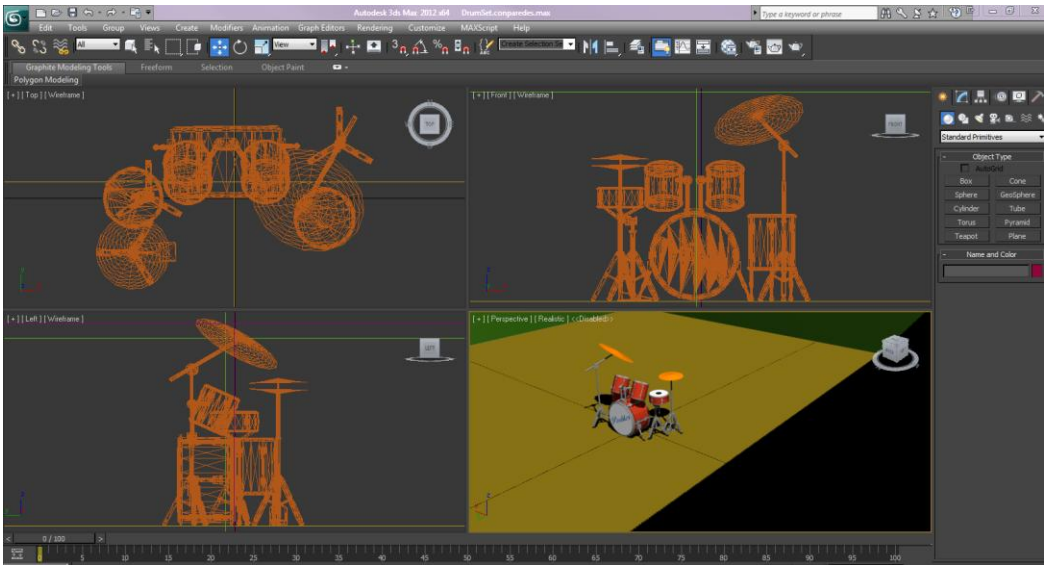


Ilustración 43: Pantalla de 3ds Max, creando la escena de la batería para la aplicación

3.6.2. Sculptris

Sculptris es un fácil, elegante y potente software de Escultura 3D, que permite al artista centrarse sólo en la creación de ilustraciones 3D sorprendentes. Atrás han quedado las limitaciones técnicas y a menudo aburridas típicamente asociados con el arte digital. Sculptris es el terreno ideal sobre el que empezar, si el usuario no tiene experiencia en modelado. Si en cambio, tiene experiencia encontrará en Sculptris una manera increíblemente rápida de realizar los conceptos. Es divertida, intuitiva y fácil de usar, lo que podrá centrarse como artista en la creatividad pura.

Los modelos que se crean con Sculptris pueden ser enviados a ZBrush con el click de un botón en el programa.

Desde el lanzamiento del ZBrush hace más de una década, Pixologic creadora de Scupltris ha sido reconocida por crear innovaciones pioneras en el mundo del arte digital.

ZBrush es la aplicación más utilizada en escultura digital en el mercado actual y es el estándar de la industria.

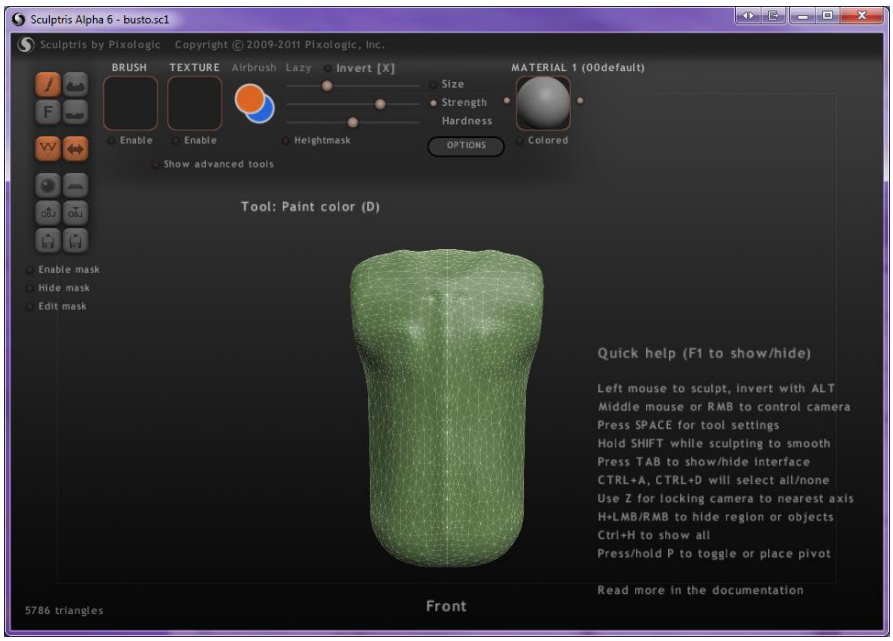


Ilustración 44: Pantalla de Sculptris, modelando el torso de la aplicación

3.7. Diagrama de clases

Se va a mostrar el diagrama de clases de la aplicación, y los patrones de diseño utilizados. [6]

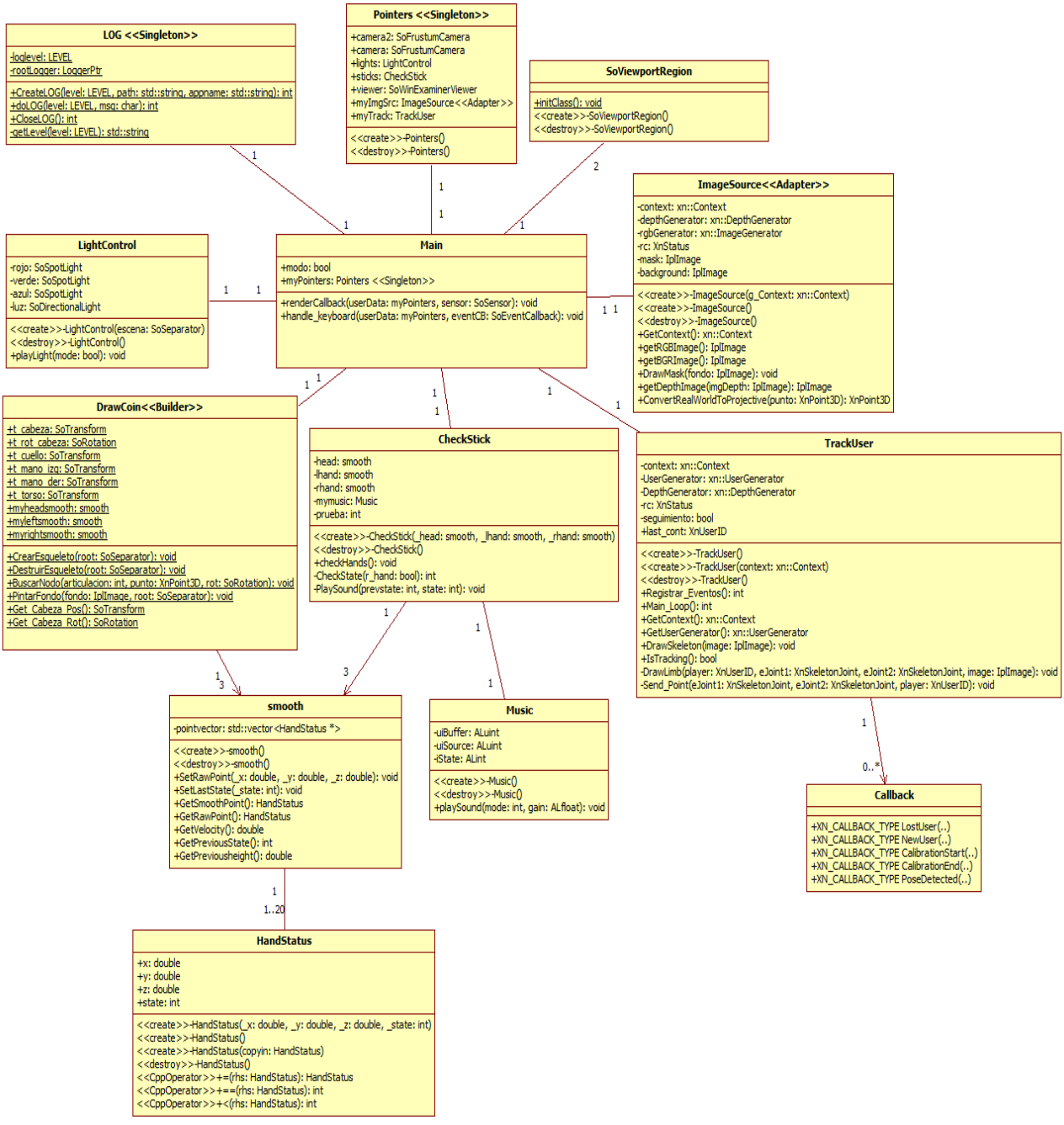


Ilustración 45: Diagrama de clases de la aplicación

3.7.1. Clase Main

Realmente no hay una clase Main, pero se quiere representar el inicio y el núcleo de la aplicación donde se encuentra el bucle principal llamado renderCallback. Este método se ejecuta cada 25 *fps*, y se comentará en el apartado Incializando el Kinect®, como se inicializa esta clase, y el bucle principal. Por tanto, hay que inicializar los módulos que se van a utilizar, OpenInventor, NITE, OpenAL, el LOG, con las distintas clases especializadas para ello.

En la inicialización también se tiene que crear la escena que se va a controlar con el Kinect® (véase la Ilustración 39). Una vez creada la escena e inicializada y configurada la aplicación, la clase Main dejará la responsabilidad al procedimiento renderCallback de ejecutar el bucle principal de la aplicación, donde la Clase Pointers entra en juego, dando la funcionalidad que se describirá en su apartado.

3.7.2. Clase LOG

La primera clase que se va a explicar, es la clase LOG. Esta clase es la encargada de registrar todas las trazas necesarias en tiempo de ejecución de la aplicación, por tanto es la primera clase que tendrá que ser inicializada. Si en alguna de las otras clases hubiera problemas al inicializarlas, o más adelante errores en ejecución, la clase LOG tiene que ser conocidas por todas para registrar el evento correspondiente y plasmarlo en el log del sistema.

Por ello, esta clase implementa el patrón de diseño Singleton. Este patrón se basa en que se garantiza la existencia de una única instancia para una clase, y la creación de un mecanismo de acceso global a dicha instancia. [6]

En caso de C++, hacer todos los miembros de la clase LOG *static*, proporciona el mecanismo necesario para que el acceso se haga a una única instancia.

3.7.3. Clase Pointers

Esta clase es la segunda en orden de importancia. La clase Pointers implementa el patrón de diseño Singleton. En este caso, esta clase agrupa varios punteros relacionados, como clases, singletons, y métodos, que se utilizan globalmente, en una entidad única. [6]

En el caso de este proyecto, agrupa los punteros necesarios para la ejecución del bucle principal, en una clase donde estos punteros son públicos. Por ello, es la segunda clase que se crea, ya que según se inicialicen las siguientes clases, se deberán ir guardando los punteros en esta clase. Además, esta clase da la posibilidad de compartir los punteros entre ellos, ya que la clase Main conoce todas las clases agrupadas en Pointers ya que las inicializó al principio de la ejecución. La clase Pointers no tiene ningún método ya que no era necesario ningún desarrollo en particular.

3.7.4. Clase SoViewportRegion

Esta clase es la siguiente en ser inicializada, ya que la aplicación tiene que comunicar al motor gráfico OpenNI las opciones de visualización que tendrá en la aplicación.

Esta clase es proporcionada por el profesor de la URJC de la asignatura RVA, es una extensión de OpenNI, que facilita la creación de regiones de visualización en el visualizador. En la aplicación, esta clase permite tener un segundo viewport donde se mostrará información adicional al primer viewport. Esto esta especificado en el apartado Viewport.

3.7.5. Clase Image Source

Esta clase implementa el patrón Adapter, el cual determina una clase que adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla. En este caso la clase es la encargada de inicializar el dispositivo, explicado en el apartado Incializando el Kinect®, quitando la responsabilidad a la Clase Main, y dar

la manera recuperar las imágenes que recoge el dispositivo como es la Imagen RGB de la cámara.

Esta clase también tiene una utilidad secundaria que es convertir las coordenadas del mundo a coordenadas proyectivas, que se utilizan para dibujar el esqueleto en el Ayudante visual.

3.7.6. Clase TrackUser:

Una vez inicializado el dispositivo a través de la Clase Image Source, se le pasa la información del contexto del dispositivo (ver Incializando el Kinect®) para poder gestionar unos módulos característicos de OpenNI. Estos módulos están relacionados con el tracking de una persona. Esta clase almacena el puntero de la clase UserGenerator, que proporciona los puntos de unión y la orientación de la cabeza.

TrackUser es la encargada de que sólo haya un usuario controlador dentro de que podrá haber más de un usuario en la escena. Este comportamiento está detallado en el apartado Usuario Controlador.

Esta clase también registra los callbacks necesarios que se lanzan en las detecciones de personas, donde se podría implementar otras acciones especiales, aunque en la aplicación no ha sido necesario.

3.7.7. Clase Callback

La clase callback, como en el apartado anterior se comentó, es la clase dónde están definidos los callbacks que se lanzarán en el procedimiento de reconocimiento y detección de personas como, por ejemplo, cuando un usuario entra en la escena, si empieza la calibración, si la calibración ha sido exitosa y cuando un usuario abandona la escena.

En esta clase se podrían definir comportamientos adicionales como, por ejemplo, reproducir un sonido de desaprobación cuando un usuario se marcha de la escena.

3.7.8. Clase LightControl

Esta clase es la encargada de controlar las luces de la escena virtual. Quitando complejidad a la Clase DrawCoin, realiza la inicialización de las luces, y mantiene los punteros para poder manejar su comportamiento en cada frame de la ejecución. Esta clase permite a la clase Main ordenarle el comportamiento de las luces en la aplicación a través del método PlayLight.

El comportamiento de esta clase está explicado detalladamente en el apartado Luces.

3.7.9. Clase DrawCoin

La clase DrawCoin tiene que objetivo abstraer a la clase Main de la creación de la escena, como se ha comentado en el apartado Escena, la escena virtual está creada en OpenInventor y tiene sus particularidades en cuanto al árbol de escena y su método de representarlo.

Por tanto, la clase DrawCoin implementa el patrón Builder, el cual se define como aquella clase que abstrae el proceso de creación de un objeto complejo, en este caso el árbol de escena, centralizando dicho proceso en un único proceso. El árbol de escena además de los nodos clave, tiene que insertar las transformaciones, escalados, separadores, etc, propios de OpenInventor que la clase Main no tiene porque conocer.

Además de la creación de la escena, es la encargada de modificar el estado del esqueleto en la escena virtual, ya que es la clase encargada de intercambiar información con el motor gráfico. A través de la clase TrackUser y del método BuscarNodo (en el diagrama de clases no se ha podido especificar esta relación 1-1), TrackUser comunica una posición en la escena virtual y el nodo correspondiente, donde DrawCoin maneja esta información. DrawCoin se encarga de guardar las coordenadas que le llegan con la Clase Smooth, donde se almacenarán una secuencia de coordenadas por cada nodo para practicar un Suavizado. DrawCoin comparte con la Clase CheckStick la información sobre la posición de los nodos.

Por último, DrawCoin es la encargada también de generar los fondos que se imprimen en las paredes de la escena virtual.

3.7.10. Clase Smooth

La clase Smooth se encarga del trabajo de almacenar y suavizar los puntos de coordenadas de un nodo en concreto. A través de la librería STL [7], se implementará un vector de la Clase HandStatus. La clase Smooth es la encargada de guardar los puntos en este vector, y de realizar las operaciones matemáticas necesarias para proporcionar a la clase que lo necesite las operaciones como: suavizado, velocidad, conseguir el último estado, e información relacionada con las posiciones. Esto se detallará en el apartado Suavizado.

3.7.11. Clase HandStatus

La clase HandStatus es la encargada de mantener la estructura que relaciona una posición 3D más el estado de la mano para poder recuperar estados y posiciones anteriores al momento del frame. Esta clase proporciona los constructores necesarios para que esta estructura se integre con el vector de STL.

3.7.12. Clase CheckStick

Esta clase recoge la complejidad necesaria para encargarse de la creación del módulo de sonido y la capacidad de reconocer cuando se ha producido un golpe, comprobando las posiciones a través de la Clase Smooth. Este procedimiento será detallado en el apartado Comprobación de golpe. Cuando reconoce una percusión, comunica a la Clase Music, que sonido tendrá que reproducir.

3.7.13. Clase Music

La clase Music es la encargada de inicializar el componente OpenAL. Creando los buffers y fuentes necesarias para la reproducción de los sonidos. Estos sonidos tienen sus propiedades, descritas en el apartado Reproducción de Música. Esta clase proporciona un método donde se le indica cual de los sonidos tiene que reproducir.

3.8. Incializando el Kinect®

Para inicializar el Kinect®, se crea un hilo de ejecución que será el encargado de gestionarlo. En este hilo, lo primero que se debe hacer es poner en marcha el Kinect®, para eso se llama a la clase ImageSource que es la encargada de dar información del Kinect®. Esta clase, a través de una clase llamada Context, inicializa y toma el mando del dispositivo. [11]

Una vez se tiene el dispositivo encendido, con la clase TrackUser se inicializará el módulo de Nite para trackear personas. Esta clase necesita la clase Context para buscar el nodo UserGenerator en el árbol interno que tiene Context.

Una vez inicializada la clase, se necesita también registrar los Callbacks a los que se llamarán en determinados momentos. Esto es cuando: se detecta a una persona en la escena, cuando se detecta una posición de pose o cuando termina la calibración de la persona. Estos Callbacks, son muy útiles para controlar el estado de las cosas.

Una vez que el Kinect® está encendido, y se han registrado todos los eventos, el hilo de ejecución entra en un bucle infinito. Este bucle llama infinitamente al método de TrackUser MainLoop, este método cada vez que da una vuelta, tiene que indicar al driver y de éste al dispositivo, que está disponible para recibir nuevos datos, en nuestro caso, la imagen de la cámara RGB y la posición (en el caso de que hubiera) de la persona trackeada. Para indicárselo, el contexto tiene un método bloqueante llamado WaitAndUpdate, cuyo nombre es bastante explicativo.

Una vez llamado a este método, se puede recuperar la imagen actual y la posición actual de la persona. A partir de ahí, la clase TrackUser es capaz de comunicarse con OpenInventor.

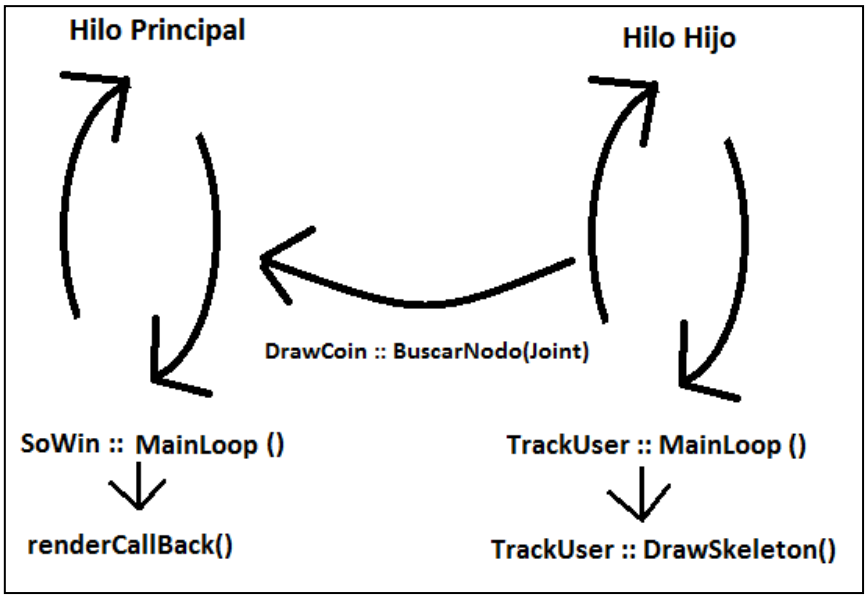


Ilustración 46: Esquema de funcionamiento

3.9. Conexión del Kinect® con la escena

Cuando la clase TrackUser toma el control en el hilo y ejecuta el método de DrawSkeleteon. Se comprueba que haya alguna persona trackeada, si no lo hay, este método no hará nada. Este método será el encargado de recorrer todos los puntos de unión, éstos se tratan igual excepto el punto que corresponde a la cabeza.

Como se ha comentado anteriormente, el resto del cuerpo está conectado a la posición de la cabeza. De esta manera, el primer nodo en actualizarse, es la cabeza, de la cual se pedirá la información de la posición respecto del Kinect® y también la información sobre la orientación de la cabeza.

Para los demás puntos de unión, se pide la información de ese punto y el de la cabeza. De esta manera, se restarán las posiciones y calcular la distancia que los separa. Esto da una ventaja, ya que si para algún punto, no se dispone de la posición, no se actualizará y se quedará con la última transformación que tenia, y así al estar conectada con la cabeza, seguirá al cuerpo y no se quedará parada en medio del vacío dando una sensación de continuidad.

Una vez se tiene esta información se le pasa a OpenInventor a través de la clase DrawCoin. La clase DrawCoin, almacena todos los punteros de las transformaciones, así que al método se le pasa el punto de unión y la información correspondiente, y este actualiza el puntero correspondiente.

3.10. **Renderizado**

El otro hilo de ejecución, se trata del propio de OpenInventor que controla el renderizado de la escena. Así pues, se desactiva el renderizado automático con la opción del visualizador AutoRedraw, y se pasa el control del renderizado a un Callback llamado RenderCallBack(). En este método, se calculan frame a frame, la configuración de la cámara Frustum. Se recupera la posición de la persona a través de la posición de la cámara, y se calcula el cono de proyección.

Se usan las fórmulas comentadas en el apartado de la visión Frustum para configurar la cámara. De esta manera, en todos los frames, se configura.

Para cambiar de modo, se ha añadido un callback que se lanza al crear un evento de teclado. Este callback, mira si la tecla pulsada era la letra M, y configura la cámara según en el modo que estuviera.

3.11. **Usuario Controlador**

El algoritmo de OpenNI es capaz de detectar y hacer tracking a más de una persona, pero nuestra aplicación sólo se centra en la calibración de un único usuario. A este usuario se le llamará usuario controlador, pues es el usuario que controlará las interacciones con la aplicación.

Para conseguir un único usuario en la aplicación se ha incluido en la clase TrackUser, un pequeño algoritmo dónde se especifica que usuario focalizar la atención del tracking. Este pequeño algoritmo se trata de escoger al usuario que este más al centro de la pantalla y a su vez, el que más cerca este del dispositivo. Así se evita centrar la atención en usuarios que sus brazos podrían salir de la escena, o usuarios que podrían estar ocluidos por usuarios más cercanos.

Ilustración 47: Distintos usuarios captados por una aplicación de OpenNI de ejemplo

3.12. Suavizado

Aunque en el tracking de Kinect® es bastante bueno en cuanto al reconocimiento del cuerpo, se ha añadido un pequeño suavizado a las posiciones de las manos, para evitar ruido y conseguir una funcionalidad mejor. Para ello se utiliza la media aritmética como suavizada, pues añadir un coeficiente sería demasiado complejo y la media permite un suavizado eficaz. Aunque tiene una desventaja, ya que limita cierta precisión en los movimientos, al ser sólo en el intervalo de 100ms, se puede ignorar esta limitación.

Dados los *n* números $\{a_1, a_2, \ldots, a_n\}$, la **media aritmética** se define simplemente como:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{a_1 + a_2 + \cdots + a_n}{n}$$

3.13. Comprobación de golpe

Para la comprobación del golpe, la clase CheckStick está comprobando si se cumplen las condiciones para reproducir un sonido. Esta clase almacena los punteros de los suavizados, tanto de la mano derecha como de la mano izquierda. Cada frame comprueba la posición de la mano para saber si está en contacto con alguno de los tambores, definiendo un espacio 3D para cada uno. Si la mano está dentro de ese espacio, CheckStick, pedirá la velocidad con la que viene la mano hacia ese punto

La velocidad es tomada sólo en el eje Y, y es calculada como el diferencial de la distancia entre diferencial del tiempo: $v = \frac{\Delta s}{\Delta t}$. Esta velocidad da la ventaja de que es direccional, pues con un movimiento hacia abajo de la mano, esta velocidad es positiva. De otra forma, si el movimiento es hacia arriba, la velocidad sería negativa y por tanto, no representaría un auténtico golpe, ya que la desventaja que tiene este proyecto es la ausencia del tambor o platillo que devolvería el golpe a consecuencia de la tercera ley de Newton.

3.14. **Reproducción de Música**

Una vez que el algoritmo ha determinado que hay que reproducir un golpe, CheckStick manda a la clase Music reproducir el sonido en concreto. La diferencia entre una reproducción normal y elegir la librería OpenAL radica en la posibilidad de renderizar los sonidos. Como se ha comentado antes, OpenAL es una librería de sonido muy usada en los videojuegos y otras aplicaciones, al permitir definir propiedades a las fuentes de sonido como la posición, velocidad, orientación,… de esta manera, la librería será capaz de modificar En este caso, sólo se necesitará saber la posición de las fuentes de sonido y la posición del oyente. Para simplificar, que el oyente se encuentra en el {0, 0, 0}.

Se han elegido unas posiciones aproximadas y simplificadas que servirán como ejemplo.

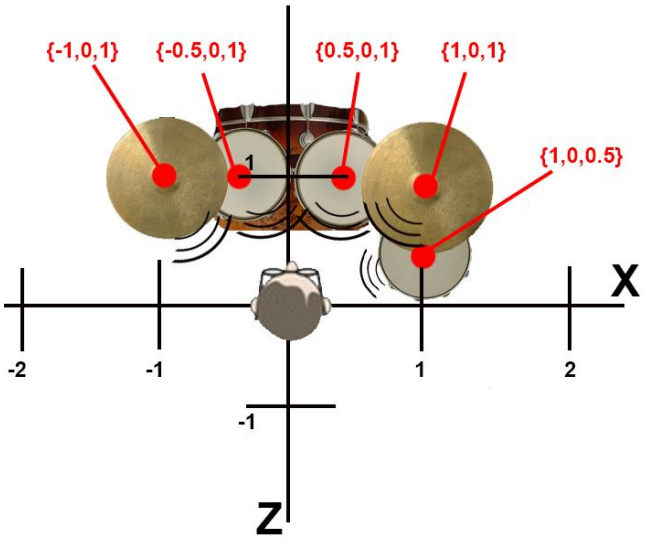


Ilustración 48: Aproximación de la situación de todos los elementos

De esta manera, el platillo izquierdo que sitúa en el eje X en la posición -1, por tanto a la izquierda del oyente si este está mirando sobre eje Z hacia el lado positivo. Cuando el platillo sea golpeado, la clase Music comunicará a OpenAl que reproduzca el sonido asociado, que después del renderizado, sonará de tal manera que parezca que proviene de la izquierda. En el caso de un sistema estéreo, sonará sólo por el altavoz izquierdo. Por el contrario, en un sistema multicanal como unos altavoces 5.1, el sonido parecerá que proviene de la zona delantera e izquierda.

3.15. **Conexión Cámara-Cabeza**

Para conseguir una sensación de inmersión en esta aplicación, es importante que la cámara, gire con una cierta similitud a la orientación de la cabeza. OpenInventor usa los cuaterniones en las rotaciones, también llamados cuaternios, son una extensión de los números reales, similar a la de los números complejos. Esta rotación se limita sólo al eje Y, de tal forma que no introduzca demasiado movimiento en la escena y el usuario no tenga sensación de mareo.

3.16. **Luces**

Para crear una inmersión más realista, se han incluido unas luces que cambiarán según qué momento este el usuario en la aplicación. Estas luces son del tipo SpotLigth, es decir, luz focal [5]. Esta luz proyecta un cono de luz con una dirección, con un color y una intensidad. Esta luz simula los focos que se usan en los conciertos de música, que se han introducido en la escena, con variedad de colores y posiciones.

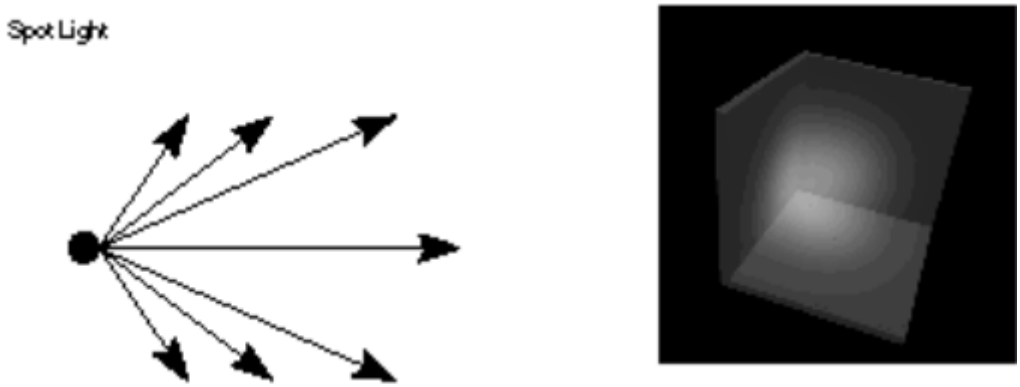


Ilustración 49: Definición de luz focal [5]

Estas luces no se encienden siempre, sino que tienen un algoritmo de iluminación. En cada frame, se genera un número aleatorio introduciendo en la semilla de aleatoriedad el parámetro time del ordenador. Una vez se tiene el número aleatorio se procede a utilizar la operación resto para introducir variedad en las luces. Cada luz tiene una operación de resto con números diferentes y se puede conseguir una secuencia sin tener un patrón definido de luces.

Estas luces se utilizan cuando el usuario está introducido en la escena, como si de un músico se tratara. Cuando no hay nadie en la escena, se apagan las luces focales, y se enciende una luz direccional con dirección -1 en el eje Y para crear una iluminación parecida a la del Sol o una lámpara encendida.

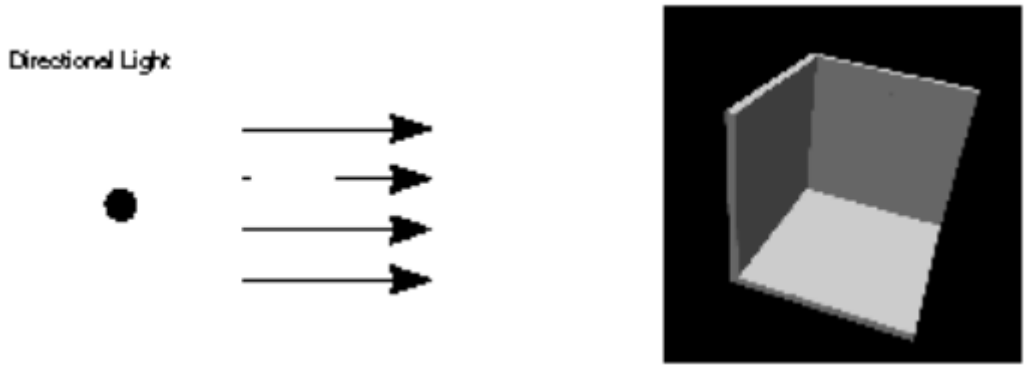


Ilustración 50: Definición de luz direccional[5]

3.17. **Ayudante visual**

En los primeros pasos del desarrollo, se detectó cierta dificultad en situarse correctamente delante de la cámara, pues no había ningún feedback para saber la posición aproximada en la que el usuario debería estar. Así pues, como el Kinect® proporciona también una cámara RGB, se puede utilizar para dar una respuesta al usuario.

Así pues, sin llegar a realidad aumentada, se añade información a la imagen recuperada del dispositivo. Se aplica una máscara encima de la imagen para poder superponer a la imagen, una posición aproximada de los tambores.

3.18. **Viewport**

Además del principal visualizador, dónde se representa en primera persona las acciones del usuario visualizando la batería musical, existe un problema, que al utilizar una pantalla de ordenador no se consigue una inmersión completa, se decidió sacrificar inmersión consiguiendo mejorar la usabilidad.

Para ello se planteó una forma de orientar al usuario para que viera lo que está haciendo desde otro punto de vista que no sea la propia visión en primera persona. Se ha añadido al visualizador un segundo viewport, dónde se vea la escena con una cámara en una posición diferente. Esta segunda cámara diferente a la principal, está mostrando su vista en un viewport situado en la parte superior izquierda, ocupando un octavo del principal. Esto se consigue con una clase externa a open inventor, creada por profesores de la universidad para la práctica de visión estéreo en la asignatura RVA.

Este viewport tiene dos opciones que se habilitan por la tecla M, una visión frontal, para ver al usuario de frente a la batería como si fuera un espectador y otra visión superior, dónde la cámara está apuntando hacia abajo situada encima de la batería, donde se podrá observar como el usuario está posicionado respecto a la batería.

3.19. **Diseño final**

Tomando todas estas decisiones, la interfaz de la aplicación se muestra en la imagen a continuación:

Ilustración 51: Muestra del diseño final de la aplicación

3.20. **Pruebas**

Las pruebas en el ciclo del software es parte del proceso de confirmación que suele realizarse durante el desarrollo y después de este. Las pruebas consisten en ejercitar el programa o la aplicación utilizando datos similares, iguales o distintos a los datos reales que harán de ser ejecutados por el programa, observar los resultados y deducir la existencia de errores o insuficiencias del programa a partir de las anomalías de este resultado.

En ocasiones, se piensa que las pruebas y la depuración de programas son una misma cosa. Aunque están muy relacionadas, en realidad son procesos distintos. Las pruebas es el proceso de establecer la existencia de errores en la aplicación. Depuración es el proceso de localizar dónde se produjeron esos errores y corregir el código incorrecto.

En el IDE que se ha utilizado, Visual Studio, la notificación de errores, las herramientas de depuración, acompañado de la implementación del LOG, facilitan la tarea de descubrir y subsanar los errores que se hayan detectado mediante las pruebas.

Es muy importante saber que las pruebas nunca demuestran que un programa es correcto. Siempre es posible que existan errores aún después de una batería de pruebas concienzuda. Los casos de pruebas sólo pueden demostrar la presencia de errores en un programa, no la ausencia de los mismos, pero se asegura de minimizarlos en la medida de lo posible.

La realización de pruebas se asimila a un proceso destructivo. Se diseña para hacer que el comportamiento de la aplicación sea distinto del que intentaba su diseñador. Como es una característica humana natural que un individuo tenga cierta afinidad con los objetos que ha construido, el programador responsable de la aplicación del sistema no es la persona más apropiada para probar un programa, pero en este caso, al no tratarse de un desarrollo comercial o grupal, no hay otra posibilidad. Psicológicamente, no se querrá destruir la propia creación, lo cual hará que, de manera consciente o inconsciente, se elijan pruebas no adecuadas y no será posible demostrar la presencia de errores en el sistema.

Por otra parte, el conocimiento detallado de la estructura de un programa o sistema de programación puede ser muy útil para identificar los casos de prueba apropiados, y el desarrollador tiene una parte importante de esto. La clave de unas

pruebas buenas es que tienen una probabilidad alta de encontrar errores en la aplicación. En el software los errores son inevitables dada la complejidad de los sistemas implicados y no deberían ser condenables. El proceso de prueba no se debe ver como una amenaza para el funcionamiento de una aplicación, sino una herramienta para conseguir un software de mejor calidad.

Para diseñar las pruebas para esta aplicación, se tienen que tener en cuenta ciertos criterios. Aunque las pruebas son importantes, es importante saber que es imposible hacer unas pruebas exhaustivas y poner unos límites tanto de esfuerzo como de tiempo. Cuando se crea una prueba, se tiene que evitar la redundancia, hay que probar todos los módulos sin necesidad de incidir demasiado en uno solo. Igualmente, las pruebas no pueden ser triviales, ni tampoco pueden ser muy complejas, pues el usuario no realizará acciones irreales. Y por último, es importante anotar todas las pruebas realizadas y documentadas, para el posterior análisis y seguimiento. De acuerdo con esto, a continuación se detallaran las pruebas realizadas.

Primero, se empezarán con las Pruebas de Caja Blanca, a veces denominada prueba de caja de cristal. Este método de diseño de casos de prueba, usa el conocimiento de la estructura de control del diseño para diseñar las pruebas de modo que ejerciten todas las partes del código buscando comportamientos no deseado.

Segundo, se continuará con las Pruebas de Caja Negra, las cuales se centran en el correcto funcionamiento sin conocer la estructura interna. La persona tiene que probar la aplicación ejercitándola completamente intentando buscar comportamientos extraños.

Las Pruebas de Caja Negra y Blanca son independientes y no se pueden intercambiar entre ellas, sino se tienen que realizar las dos completamente. A continuación se comentarán las pruebas diseñadas para esta aplicación.

3.21. **Pruebas Caja Blanca**

Estas pruebas garantizan que se ejercita por lo menos una vez todos los caminos para comprobar que el código no produce errores. Son independientes de cada módulo y tienen que ejercitar toda decisión lógica, todos los bucles y estructuras internas de datos. [20]

En este documento se desarrollará sólo las pruebas en las estructuras más importantes y relevantes. Estas pruebas son, la comprobación del pintado de todas las articulaciones que se comunican desde el Kinect®, la estructura del bucle principal y también las decisiones lógicas que corresponden a la correcta finalización de la aplicación si hubiera algún error en la ejecución. Para hacer pruebas de caja blanca, se ha situado instrucciones de LOG en los distintos cauces de ejecución para comprobar efectivamente que la aplicación ejercita todo el programa sin producir fallos. En este caso, el LOG se situará en nivel debug, donde el Log se comportará de manera *verbose*, es decir, muy detallado.

3.21.1. Bucle Principal

El bucle principal es la estructura más importante en la ejecución de la aplicación, pues es el código que se ejecuta en cada frame del renderizado de la aplicación. Al ejecutarse 25 frames al segundo, es importante que no haya ningún fallo para que no se produzcan errores.

En este bucle hay que probar las instrucciones y decisiones lógicas que se ejecutarán por orden: actualizado de la información de Kinect®, conseguir la imagen del Kinect®, actualizar los datos del esqueleto, pintar el ayudante, comprobación de golpes, encendido de luces y renderizado. Se incluirán instrucciones de LOG para comprobar todas estas situaciones.

Al actualizar la información del Kinect®, se tiene que asegurar que esté disponible cuando se le requiere. Eventualmente, por posibilidad de desconexión o problemas en el bus, el Kinect® no estaría disponible, y la aplicación se tendrá que terminar ordenadamente. En las pruebas, se ha probado las distintas situaciones, como forzar la desconexión del Kinect®, como otros procedimientos para simular que

Kinect® está ocupado, como terminar el proceso correspondiente, incluso, arrancar otra aplicación que necesite de Kinect®. En el caso de error, los mensajes de LOG indicarán el motivo del fallo.

Una vez comprobada la disponibilidad del Kinect®, se procede a recuperar la imagen del Kinect®. Para ello hay que comprobar que la estructura que contiene las imágenes, esta inicializada correctamente y la imagen es correcta.

Con la imagen correcta, se procede a pedir a Kinect® la posición del usuario en el caso de que esté haciendo tracking. Para saber si el Kinect® está haciendo tracking a un usuario, se puede fijar en el funcionamiento de las luces de la escena. Si no hay ningún usuario delante del Kinect®, la luz de la escena será de color y dirección ambiental. Si hay un usuario interactuando con el sistema, unas luces parpadeantes de colores iluminarán la escena creando una situación más inmersiva.

Una vez comprobado que el usuario está haciendo tracking, hay que comprobar la correcta relación entre las articulaciones reales respecto al Kinect®, y los correspondientes modelos en la escena. En cada iteración se puede observar en el LOG si han sido actualizados todos los nodos en la ejecución del bucle.

De igual manera se procederá a incluir LOGs para ver cuándo se detecta una colisión con los tambores virtuales. Cada golpe, incluirá un registro en el LOG con el tiempo y el tambor que ha sido golpeado.

Por último, el renderizado de la escena es evidente al ser mostrada. Para intentar generar errores, se procede a cerrar la ventana de renderizado, intentar modificarla, con el objetivo de estresarla y producir un error.

```
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Conseguido imagen de Kinect®
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un punto
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un punto
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un punto
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Pintado el esqueleto
TRACE 17/09/2012 19:02:48  MusicVirtual - Imagen no nula
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Liberando imagen
TRACE 17/09/2012 19:02:48  MusicVirtual - Luces: Pintando luces
DEBUG 17/09/2012 19:02:48  MusicVirtual - Luces: Modo discoteca
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Luces pintadas
```

```
DEBUG 17/09/2012 19:02:48  MusicVirtual - CheckSticks:comprobar golpes
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Velocidad
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir altura anterior
TRACE 17/09/2012 19:02:48  MusicVirtual - X=0,355560,Y=-0,802829,Z=-0,625019
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir anterior estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Punto Suavizado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir Velocidad
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir altura anterior
INFO  17/09/2012 19:02:48  MusicVirtual - Detectado percusion en
TRACE 17/09/2012 19:02:48  MusicVirtual - Suavizado:Introducir un estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Pedir anterior estado
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Manos Comprobadas
TRACE 17/09/2012 19:02:48  MusicVirtual - Main: Renderizamos
```

Ilustración 52: Traza del LOG en el bucle principal en modo Verbose

3.21.2. Comprobación de pintado

Para la comprobación de pintado de todos los nodos, se va a proceder a comprobar que los nodos son pintados en el ayudante. Para asegurarnos de probar todos los nodos uno a uno, se va a proceder a ocluir la cámara una vez esté haciendo el tracking del usuario. Al ocluir algún nodo, no sólo se ejercita la instrucción por defecto si no hay nodo, sino que al volver a mostrar el nodo, el programa ejecute las instrucciones pertinentes. Así se podrá asegurar de que la aplicación pinta todos los nodos y no omite ninguno.

3.22. Pruebas Caja Negra

Estas pruebas comprueban la correcta funcionalidad del programa, es decir si cumple con su objetivo de manera correcta. Las pruebas se centran en encontrar errores en funciones incorrectas o ausentes, en posibles fallos de interfaz, errores en estructuras de datos o en accesos a bases de datos. Incluso, las pruebas son capaces de comprobar el rendimiento de la aplicación, la inicialización y la terminación en el caso. [20]

En este documento, se desarrollarán las pruebas de Caja Negra más importantes. Para realizar estas pruebas, se tendrán que definir varias acciones para realizar con la aplicación, se escribe el resultado esperado, y la salida de estas pruebas es el comportamiento realizado por la aplicación. Si coinciden el comportamiento esperado y el real, es que esa prueba no ha detectado ningún fallo en ese módulo.

Las pruebas que se han realizado son referentes a los dos aspectos más importantes de la aplicación, las pruebas sobre el funcionamiento del Tracking, y las pruebas sobre la funcionalidad de la aplicación que tiene que cumplir con el Tracking como es la reproducción de sonido y la correspondencia del modelo en la escena virtual.

3.22.1. Pruebas de Tracking

En estas pruebas se tiene que probar el correcto funcionamiento del tracker y del dispositivo, haciendo una correspondencia entre el mundo real y el mundo virtual. Primero para asegurarnos un tracking adecuado, se situará el Kinect® a una altura de unos 80 cm, en una superficie plana, para conseguir unas posiciones de Y paralelas al eje X. En la componente de distancia, el Kinect® deberá estar colocado a 1,50 m de distancia del usuario, para que el cuerpo sea visible casi en su totalidad, y el rango de movimientos con los brazos sea amplio.

Cuando el Kinect® está bien situado, se situará una banqueta a la distancia anteriormente comentada. Se arranca la aplicación y se sitúa el usuario sentado en la banqueta. En este momento la aplicación no debería tardar más de 3 segundos en tener calibrado a la persona. A partir de aquí se harán las siguientes comprobaciones.

Una segunda persona, se introducirá en la escena para comprobar que la aplicación es unipersonal y que no genera interferencias al usuario controlador. Sólo en el único momento que la segunda persona esté más centrada que la primera, es decir, se han intercambiado las posiciones, el usuario segundo será el usuario controlador. Esto también se ha probado de manera que no tenga saltos indeseados.

También se procederá a probar el tracking del usuario en diferentes posiciones. Es importante que la aplicación reconozca al usuario tanto de pie, como sentado. La ventaja que tiene el algoritmo actualizado de OpenNI, permite cazar al vuelo al usuario, ya que el algoritmo principal tiene problemas con la gente sentada. Pues así, el usuario podrá entrar en la escena desde cualquier punto o posición, y podrá tardar en sentarse o

sentarse inmediatamente, la aplicación tiene que contestar con cierta rapidez, y como se ha comentado antes, no tardar más de 3 segundos en hacer tracking. La entrada en escena de usuarios y la calibración exitosa o errónea se podrán observar en el Log o la consola si estuviera activa.

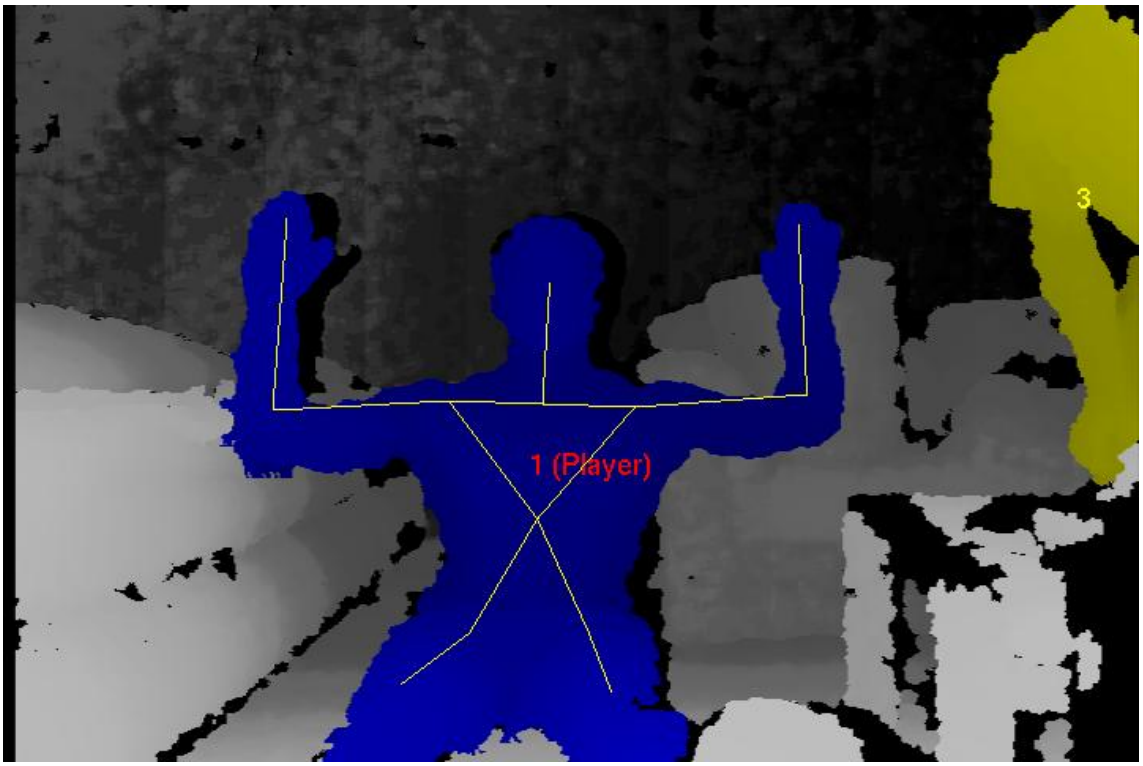


Ilustración 53: Una imagen de profundidad de la aplicación

En el caso de que las calibraciones fueran incorrectas o no detectara ningún usuario, al tratarse de un módulo externo, habría que encontrar las soluciones en el soporte oficial de OpenNI.

Por último, en las pruebas de tracking, se tendrá que observar la latencia introducida por el suavizado y el propio dispositivo. Como se comento en el apartado Suavizado se introduce un suavizado para quitar el ruido que tiene el dispositivo por defecto e intentar detectar mejor los movimientos extremadamente rápidos. Volviendo a las pruebas, el usuario deberá mover los brazos de manera usual y parecida al movimiento de un batería. En el ayudante, se podrá observar el seguimiento que tiene la aplicación sobre el usuario controlador, así pues observar, si hubiera mucha latencia, o si hubiera errores de tracking.

Una vez probado el sistema de tracking, se procederá a probar la funcionalidad asociada al tracking, es decir, al reconocimiento de movimientos, e interacciones en el mundo virtual.

3.22.2. Pruebas de Funcionalidad

En este caso, las pruebas se centrarán en la correcta interacción del usuario con la aplicación. Se tendrá que probar la correspondencia de los golpes con la posición virtual de los tambores, la correspondencia entre el movimiento de percusión, y la correcta posición de los sonidos con sus tambores.

Primero, se comprobará el correcto funcionamiento del tracking y la posición relativa que relaciona con la batería virtual. Si el usuario ya se ha sentado, al dirigir su mano hacia el frente, en la mano virtual deberá situarse al frente situado por encima de los tambores frontales. Si el usuario mueve los brazos hacia los lados, las manos virtuales deberían situarse por encima de los tambores laterales. Y si el usuario sube la mano derecha y la lleva hacia la derecha, la mano virtual debería estar por encima del platillo. Para realizar estas pruebas, se ha habilitado un modo donde la cámara del viewport en vez de tener una perspectiva frontal, se tiene una perspectiva de planta, en la que se podrá verse la posición de las manos desde arriba. Estas comprobaciones son importantes para saber si visualmente, el mundo está bien colocado, respecto al tracking del usuario.

Una vez colocado el usuario correctamente en el mundo virtual, se puede proceder a probar los movimientos de percusión. En la mismas posiciones que la prueba anterior, se sitúan las manos del usuario por encima de los tambores. A continuación se bajan los brazos para proceder a un movimiento de percusión. En este momento se tienen que comprobar dos cosas importantes en los movimientos que realiza el usuario. El primero será el nivel al que se producirá el sonido, todos los tambores tienen que tener una altura determinada y tiene que ser importante que generé el sonido a la altura donde la mano virtual llega al tambor virtual. Como segunda cosa importante que atender en la prueba, es que solo suene una vez, pues al bajar un brazo y no volverlo a subir, deberá sonar una única vez.

Además de esta prueba, se deberá comprobar que tanto la mano derecha, como la mano izquierda funcionan con cualquiera de los tambores, atendiendo también a los

límites físicos que tiene el propio usuario. Por otra parte, también se probará la resolución mínima del Kinect®, pues tendrá un límite donde el usuario moverá tan rápido los brazos que la aplicación no será capaz de seguir completamente el movimiento.

Por último, al reproducir las percusiones, se tendrá que comprobar la correcta colocación de los sonidos, tanto en su localidad, como en la correcta correspondencia. Cuando un usuario, por ejemplo hace el movimiento para tocar el platillo, la aplicación deberá reproducir el sonido del platillo, y no la de otro tambor. Como la aplicación ha utilizado una librería de sonido, como es OpenAL, con la opción de renderizar estos sonidos dándoles una serie de propiedades, se tendrá que comprobar que, por ejemplo, el platillo se reproduzca, y el sonido parezca que venga por la derecha. Esto se tiene que comprobar con un sistema de sonido estéreo que pueden ser dos altavoces o unos cascos.

3.23. **Evaluación en usuarios**

Para conocer una visión general de la aplicación, se ha presentado a un grupo de personas para saber la opinión sobre unos determinados factores. Interesará saber que aceptación tiene esta aplicación en un conjunto de personas, y además de poder analizar diferentes factores importantes en el desarrollo de esta aplicación. A este grupo se le pasará un formulario en el cual puntuaran estos factores (en una escala del 0 al 10, donde el 0 es muy negativo y 10 es muy positivo) y podrán añadir un comentario sobre la aplicación. Este formulario se analizar según el método de análisis de varianza, llamado ANOVA, de un factor por cada factor del formulario.

El ANOVA, es una técnica de análisis estadístico que se utiliza para comparar las medias de dos grupos o más, en el estudio de la variabilidad observada en los datos.[17]. Se llama factor o independiente a la variable que define los grupos que se desea comparar. A la variable cuantitativa se llama dependiente, es la variable que se va a medir en los dos grupos. En este caso se va a realizar 3 análisis, uno por cada variable dependiente, siendo dos grupos de estudio. Estas 3 variables son: realismo, diversión y facilidad. Los dos grupos de estudio son: un grupo compuesto por personas sin nociones de interpretación musical, que será el grupo a estudiar, y un grupo con conocimiento en manejar una batería instrumental.

La muestra que se ha utilizado está compuesta por 28 personas, de las cuales son: 7 personas con dominio en el manejo de la batería instrumental con distintos niveles de experiencia, todos ellos de género masculino. El resto de componentes de la muestra, son 21 personas con distintas cualidades, en edades comprendidas entre los 19 y 40, con una relación entre hombres y mujeres de 4 a 1.

Se ha de comentar que la muestra total elegida ha sido seleccionada dentro de mi entorno, pues no se corresponde formalmente a una amplia muestra elegida aleatoriamente que sea representativa. En cuanto a la cantidad de la muestra, para este caso, las técnicas recomiendan obtener un número más alto en los dos grupos. En este caso deberían ser 25 personas *no músicos* y 25 *músicos* para un análisis más correcto.

Los resultados de esta evaluación se comentarán en el siguiente apartado.

Una vez realizada la evaluación a los usuarios, se presentarán los resultados obtenidos y se comentará una conclusión final a continuación de estos.

3.23.1. Realismo

La aplicación se basa en crear una simulación de una batería instrumental, y como simulador de realidad virtual, es necesario saber el alcance de la aplicación en cuanto a la presencia y realismo. Por ello, se ha preguntado a la muestra cómo consideran el realismo de la aplicación, y a continuación, se muestra el resultado después de aplicar el ANOVA de un factor, comparación de medias. Los datos de los formularios son introducidos en el software SPSS de IBM, un paquete estadístico, que realiza los cálculos automáticamente, sin necesidad de hacer los cálculos en papel.[17]

Realismo

		N	Media	Desviación típica	Error típico	Intervalo de confianza para la media al 95%		Mínimo	Máximo	Varianza entre componentes
						Límite inferior	Límite superior			
Músico		7	6,2857	,75593	,28571	5,5866	6,9848	5,00	7,00	
No Músico		21	7,0952	1,26114	,27520	6,5212	7,6693	5,00	9,00	
Total		28	6,8929	1,19689	,22619	6,4288	7,3570	5,00	9,00	
Modelo	Efectos fijos			1,16418	,22001	6,4406	7,3451			
	Efectos aleatorios				,41536	1,6153	12,1705			,19859

Ilustración 54: Resultados de los datos de Realismo

A la vista de los resultados, se puede ver que los *músicos* consideran que la aplicación es menos realista que los *no músicos*. Se entiende que al ser expertos en este campo, les resulta deficitaria la aplicación, al no cumplir todas las expectativas que esperarían de una batería instrumental de verdad. Por el contrario, los *no músicos* les ha resultado más realista, ya que no tienen conocimiento veraz del funcionamiento de una batería instrumental.

3.23.2. Diversión

La segunda variable que tenían que valorar los usuarios es la diversión, pues la aplicación surgió con la idea de crear un juego musical. En este aspecto, los videojuegos tienen muchas cualidades, pero quizás la más representativa es la diversión que proporciona a los usuarios. A continuación, se muestra el resultado después de aplicar el ANOVA de un factor, comparación de medias.

Diversión

		N	Media	Desviación típica	Error típico	Intervalo de confianza para la media al 95%		Mínimo	Máximo	Varianza entre componentes
						Límite inferior	Límite superior			
Músico		7	7,0000	,81650	,30861	6,2449	7,7551	6,00	8,00	
No Músico		21	7,5714	,74642	,16288	7,2317	7,9112	6,00	9,00	
Total		28	7,4286	,79015	,14932	7,1222	7,7350	6,00	9,00	
Modelo	Efectos fijos			,76316	,14422	7,1321	7,7250			
	Efectos aleatorios				,29694	3,6556	11,2016			,10780

Ilustración 55: Resultados de los datos de Diversión

Según los resultados obtenidos, se puede observar que la media es más alta en los *no músicos* que en los músicos. Se interpreta, como el caso anterior, que los *músicos* al tener experiencia previa, se obtiene una respuesta más suavizada respecto a la simulación de una batería. La respuesta de los *no músicos* puede verse debida a la facilidad que se les presenta en tocar una batería sin necesidad de tener una batería instrumental físicamente.

3.23.3. Facilidad

Por último, una variable que se quería medir, es la facilidad de uso de la aplicación. Es relevante que una aplicación tenga una curva de aprendizaje lo más suave posible. Con la incorporación del Algoritmo actualizado, la facilidad de uso de la aplicación se consideró superior a la versión dónde el usuario necesitaba hacer una pose, pues implicaba que había que comunicárselo previamente. A continuación, se muestra el resultado después de aplicar el ANOVA de un factor, comparación de medias.

Facilidad

		N	Media	Desviación típica	Error típico	Intervalo de confianza para la media al 95%		Mínimo	Máximo	Varianza entre componentes
						Límite inferior	Límite superior			
Músico		7	7,7143	1,11270	,42056	6,6852	8,7434	6,00	9,00	
No Músico		21	7,6667	,79582	,17366	7,3044	8,0289	6,00	9,00	
Total		28	7,6786	,86297	,16309	7,3439	8,0132	6,00	9,00	
Modelo	Efectos fijos			,87914	,16614	7,3371	8,0201			
	Efectos aleatorios				,16614[a]	5,5675[a]	9,7896[a]			-,07248

Ilustración 56: Resultados de los datos de Facilidad

Como se observa, existe una la similitud entre las medias de los dos grupos. Esto indica que la aplicación resulta igual de sencilla de usar en ambos grupos.

4. CONCLUSIONES

El desarrollo de esta aplicación, con el diseño que se ha comentado anteriormente y las pruebas realizadas, dan como resultado una aplicación con las siguientes características:

- Interactiva:

Es importante para una aplicación de simulación y realidad virtual, que la aplicación sea interactiva. Como se comentó en el apartado *Simuladores y Realidad* Virtual, la interactividad ayuda mucho a aumentar el nivel de presencia del usuario. En este caso, la posibilidad de que el usuario mueva los brazos simulando tocar una batería resulta, por tanto, bastante intuitivo, similar a que lo que la gente realiza recurrentemente en la vida cotidiana con cualquier objeto al reproducir sonidos mediante percusiones.

- Respuesta en Tiempo Real:

La aplicación es capaz de hacer tracking al usuario y recrear la relación entre el mundo físico y el mundo virtual en tiempo real, en concreto, a 25 *fps*. La interacción sin latencia es imprescindible para crear una interacción realista. Cuanto más sienta el usuario que el mundo virtual reaccione con sus acciones, más inmersa será la sensación que tenga de ese mundo.

- Creible:

La aplicación tiene como objetivo ser realista y a la vez inmersiva. Para que sea más creíble la interacción con la aplicación, se ha procedido a mantener la escala entre el usuario y una batería de música de verdad. Además de esto, la reproducción de sonidos con información de posición. Así mismo, se incorporó distintos ayudas para que la aplicación fuera más creíble, como las luces parpadeantes, y otros métodos comentados en el apartado Descripción informática.

- Facil de Usar:

Si una aplicación tiene una curva de aprendizaje muy complicada, es fácil que el usuario desista rápidamente de seguir interactuando con la aplicación. En este caso, la

aplicación resulta muy sencilla, ya que el usuario sólo necesitará sentarse delante de la cámara, y mover los brazos de manera bastante sencilla. Además como la aplicación, le acompaña un ayudante visual comentado en el apartado Ayudante visual, que ayudará al usuario a saber la primera vez dónde se encuentran los distintos tambores. Así mismo, el usuario podrá observar su movimiento cerca de la batería virtual en el visualizador de la aplicación.

4.1. Dificultades

Aún después de los resultados obtenidos, resulta recurrente un problema en todos los usuarios que han probado la aplicación. Al no tener una respuesta física o visual (en donde debería estar la batería física), el usuario le cuesta recordar dónde se situaba el tambor y causaba errores de aproximación en los golpes de percusión.

Investigando este efecto para intentar resolverlo, se estimó que era un problema del sentido propioceptivo del ser humano.

La propiocepción, es la capacidad de sentir la posición relativa de partes corporales contiguas. La propiocepción regula la dirección y rango de movimiento, permite reacciones y respuestas automáticas, además interviene en el desarrollo del esquema corporal y en la relación de éste con el espacio, sustentando la acción motora planificada. Otras funciones en las que actúa con más autonomía son el control del equilibrio, la coordinación de ambos lados del cuerpo, el mantenimiento del nivel de alerta del sistema nervioso central y la influencia en el desarrollo emocional y del comportamiento.

Aún así, este sistema necesita saber de respuestas físicas o visuales para determinar la posición exacta de las extremidades, por lo tanto, si no las tuviera, no llega a ser fiable. Por ello, se incorporó un ayudante visual donde se enseña al usuario dónde debe mover los brazos, ya que utilización de un monitor limita la sensación de presencia. Para mejorar este inconveniente, se podría utilizar un HMD (head mounted display).

4.2. Líneas Futuras

Una vez finalizado el desarrollo de la aplicación, es momento para recopilar la información y tener así una visión general. El ciclo de software se define cómo el tiempo de uso del software, desde su creación hasta el abandono completamente del software. En este caso, este proyecto no tiene porque ser autocontenido y puede tener multitud de líneas por las que seguir adelante. A continuación, se explicarán posibles líneas que se pueden seguir en caso de querer actualizar la aplicación:

4.2.1. Rigging

Uno de los primeros objetivos fue conseguir un esqueleto que se moviera igual que el usuario. Conseguir un esqueleto realista o lo más cercano a la realidad es más complejo de lo que se pareció en un principio. Una vez que se decidió realizarlo, se pudo observar que era demasiado avanzado, en el que sólo el rigging sería casi un proyecto aparte. Rigging es el proceso técnico/artístico de configurar un personaje/modelo 3D/propiedad/objeto para que pueda ser posteriormente animado. Es un proceso que lleva mucha lógica pero también sentido artístico para poder hacer desde el modelo, algo que pueda expresar sentimientos, movimientos y expresiones dependiendo del caso. Para hacer una comparación, la persona encargada del riggin es como un "creador de marionetas", pero en este caso como son modelos digitales, se podría decir entonces "marionetas digitales". Primero, existe un modelo 3D que no se puede mover en ninguna forma, entonces un animador crea un sistema de huesos, expresiones, scripts para que dicho modelo se mueva en la forma en que debe hacerlo de acuerdo a lo que se requiere. A continuación se explicará los pasos a seguir en el procedimiento de rigging.[14]

Primero, se tendrá que hacer el modelo que se quiere animar, como por ejemplo un personaje. La propia modelización es muy compleja en sí misma, incluso hay especialistas que únicamente se centran en modelar objetos.

Después de modelizar, el personaje necesita una textura, que al igual que la modelización, tiene sus complicaciones y se pueden hacer grandes texturas incluso

ahorrando detalle en la etapa anterior, solo trabajando en la textura, mapa de normales, luces, etc.

Una vez creado el personaje a animar, sería muy complejo intentar mover el personaje vértice a vértice a las distintas posiciones clave o keyframes e intentar interpolar para crear la animación completa. En vez de eso, las técnicas actuales permiten hacer un trabajo más rápido a partir del rigging. Con el modelo creado, se ajusta un esqueleto previamente definido a las articulaciones interesantes de animar de nuestro modelo. Al igual el esqueleto humano sirve como base para poner encima los músculos y luego la piel, esta analogía es también compatible para las animaciones.

El ajuste del esqueleto al modelo, permite decidir que vértices tienen que seguir a que huesos, por cercanía. Para no producir cortes y movimientos raros, los vértices no sólo están influenciados por un único hueso, sino que están influenciados por otros huesos para conseguir un movimiento más suave y más realista.

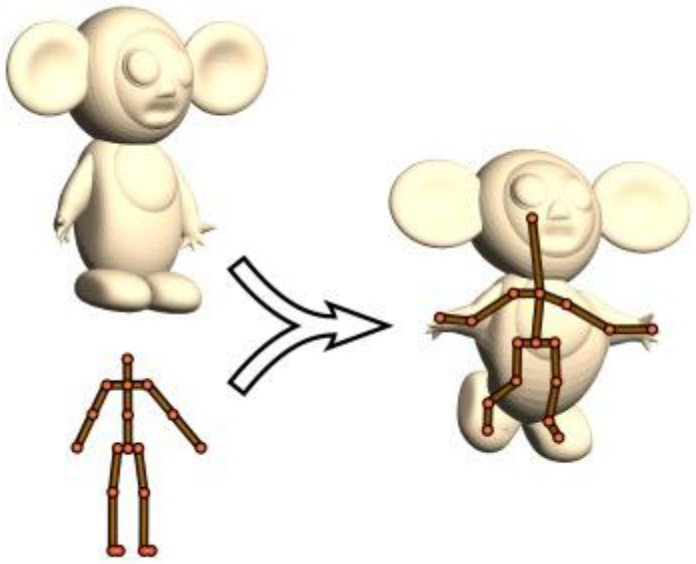


Ilustración 57: Ejemplo de rigging [14]

4.2.2. Inclusión de nuevos instrumentos

Esta aplicación se inspira en un juego de música pero sin contar con los periféricos específicos. Como todos estos juegos, se podría incluir más de un instrumento. Los instrumentos que más se ajustarían a la disposición de esta aplicación y más intuitivos serían mayoritariamente de percusión. Esta adaptación se podría hacer con pocos cambios en el código (ejemplo de mantenibilidad del código), ya que sólo creando una nueva clase CheckStick, e implementando el patrón de diseño Estado[6].

Otros instrumentos que se podrían adaptar a la aplicación sería:

- Xilófono:

Se puede detectar los movimientos respecto a una tabla con distintas láminas que sonarán con diferentes notas.

- Guitarra básica:

Se podría detectar la distancia entre mano y mano, de tal forma que sea capaz la aplicación de reproducir un acorde más grave o agudo, y además en el momento que la mano derecha haga el movimiento de sacudir.

- Arpa:

Se podría reconocer la disposición de las manos en las cuerdas de un arpa y reproducir sonidos cuando la mano alcance cierta profundidad.

- Trombón:

Se podría reconocer el ángulo formado por la cabeza y las manos, y reproducir sonidos

- Theremin:

Uno de los instrumentos que mejor se ajustarían, sin hacer muchos cambios en el código sería el instrumento Theremin. Este instrumento es uno de los primeros instrumentos musicales electrónicos. El diseño clásico consiste en una caja con dos antenas y se ejecuta acercando y alejando la mano de cada una de las antenas correspondientes, sin llegar a tocarlas. La antena derecha suele ser recta y en vertical, y

sirve para controlar la frecuencia o tono: cuanta más cerca esté la mano derecha de la misma, más agudo será el sonido producido. La antena izquierda es horizontal y con forma de bucle, y sirve para controlar el volumen: cuanto más cerca de la misma esté la mano izquierda, más baja el volumen, y viceversa. Por tanto, es muy fácil con Kinect® reconocer estos movimientos y reproducir sonidos sintéticos en el ordenador

Ilustración 58: Personaje de The Big Bang Theory tocando el Theremin [CBS]

4.2.3. Cambio de Motor

OGRE es un motor orientado a objetos libre para aplicaciones gráficas 3D interactivas. El nombre del motor OGRE es un acrónimo de Object-oriented Graphics Rendering Engine. Como su propio nombre indica, OGRE no es un motor para el desarrollo de videojuegos; se centra exclusivamente en la definición de un middleware para el renderizado de gráficos 3D en tiempo real. [3]

El proyecto de OGRE comenzó en el 2000 con el propósito de crear un motor gráfico bien diseñado. El líder del proyecto Steve Streeting define el desarrollo de OGRE como un proyecto basado en la calidad más que en la cantidad de características que soporta, porque la cantidad viene con el tiempo, y la calidad nunca puede añadirse a posteriori.

La popularidad de OGRE se basa en los principios de meritocracia de los proyectos de software libre. Así, el sitio web de OGRE 2 recibe más de 500.000 visitas diarias, con más de 40.000 descargas mensuales.

El núcleo principal de desarrolladores en OGRE se mantiene deliberadamente pequeño y está formado por profesionales con dilatada experiencia en proyectos de ingeniería reales.

Ilustración 59: Logotipo de la librería OGRE

OGRE tiene una licencia LGPL Lesser GNU Public License. Esta licencia se utiliza con frecuencia en bibliotecas que ofrecen funcionalidad que es similar a la de otras bibliotecas privativas. Por cuestión de estrategia, se publican bajo licencia LGPL (o GPL Reducida) para permitir que se enlacen tanto por programas libres como no libres. La única restricción que se aplica es que si el enlazado de las bibliotecas es estático, la aplicación resultado debe ser igualmente LGPL (porque el enlazado estático también enlaza la licencia).

La versión oficial de OGRE está desarrollada en C++ (el lenguaje estándar en el ámbito del desarrollo de videojuegos). La rama oficial de OGRE únicamente se centra en este lenguaje sobre los sistemas operativos GNU/Linux, Mac OS X y Microsoft® Windows. No obstante, existen wrappers de la API a otros lenguajes (como Java, Python o C#) que son mantenidos por la comunidad de usuarios (presentando diferentes niveles de estabilidad y completitud), que no forman parte del núcleo oficial de la biblioteca.

Algunas características destacables de OGRE son: [3]

- Motor Multiplataforma:

Es posible compilar la biblioteca en multitud de plataformas.

- Diseño de Alto Nivel:

OGRE encapsula la complejidad de acceder directamente a las APIs de bajo nivel (como OpenGL y Direct3D) proporcionando métodos intuitivos para la manipulación de objetos y sus propiedades relacionadas. OGRE hace un uso de varios patrones de diseño.

- Grafos de Escena:

Prácticamente cualquier biblioteca de despliegue de gráficos 3D utiliza un Grafo de Escena

- Aceleración Hardware.
- Materiales:

Otro aspecto realmente potente en OGRE es la gestión de materiales. Es posible crear materiales sin modificar ni una línea de código a compilar.

- Animación:

OGRE soporta tres tipos de animación ampliamente utilizados en la construcción de videojuegos: basada en esqueletos (skeletal), basada en vértices (morph y pose).

- Plugins, Gestión de Recursos y Otros.

Ilustración 60: Aplicación ejemplo de Ogre, integrando un esqueleto y OpenNI

4.2.4. Funcionalidades Interactivas

Como se ha comentado en el apartado 2.5, se trata de introducir al usuario en un mundo virtual dónde pueda interactuar para conseguir un fin. En la aplicación no se ha llegado tan lejos, principalmente por cuestión de objetivos. Desarrollar un juego es muy complejo y se puede alargar mucho en el tiempo, por no decir de los recursos necesarios que hacen falta, no sólo logísticos sino personales.

El desarrollo de este proyecto se centraba en la interacción principal entre el usuario, una batería de música e investigar la capacidad de inmersión que la aplicación es capaz de crear al usuario.

Además de la interacción, un videojuego como ya se ha comentado, necesita de algún fin para enganchar al usuario, y este se introduzca en el mundo virtual más fácilmente. Por ejemplo, en esta aplicación, ya que se trata de una batería musical, el videojuego podría tratar de tocar una canción siguiendo cierta partitura, y golpeando los tambores en los tiempos adecuados. RockBand, GuitarHero son videojuegos musicales que desarrollan esta idea, pero a diferencia de esta aplicación, necesitaban de un periférico especial para cada instrumento.

Lo que se conseguía conseguir con este desarrollo, era simular un instrumento sin necesidad de tener mucho espacio en la habitación y con un dispositivo genérico que sirve para otro tipo de videojuego o aplicación.

Para que fuera un videojuego completo, a esta aplicación habría que añadirle por ejemplo, una forma de seguir una partitura que coincidiera con la música y los tonos de las percusiones. Además esto implica que el juego sea capaz de mantener una puntuación y una correlación con las acciones del usuario. Por último un menú que englobara todas las acciones y modos de juego que contiene el videojuego.

Ilustración 61: Captura de pantalla de RockBand [ROC]

4.2.5. Reverberaciones y otros efectos

La reverberación es un fenómeno producido por la reflexión que consiste en una ligera permanencia del sonido una vez que la fuente original ha dejado de emitirlo.

Cuando se reciba un sonido, este llega desde su emisor a través de dos vías: el sonido directo y el sonido que se ha reflejado en algún obstáculo, como las paredes del recinto. Cuando el sonido reflejado es inteligible por el ser humano como un segundo sonido se denomina eco, pero cuando debido a la forma de la reflexión o al fenómeno de persistencia acústica es percibido como una adición que modifica el sonido original se denomina reverberación.

La reverberación, al modificar los sonidos originales, es un parámetro que cuantifica notablemente la acústica de un recinto. Para valorar su intervención en la acústica de una sala se utiliza el «tiempo de reverberación». El efecto de la reverberación es más notable en salas grandes y poco absorbentes y menos notable en salas pequeñas y muy absorbentes. [2]

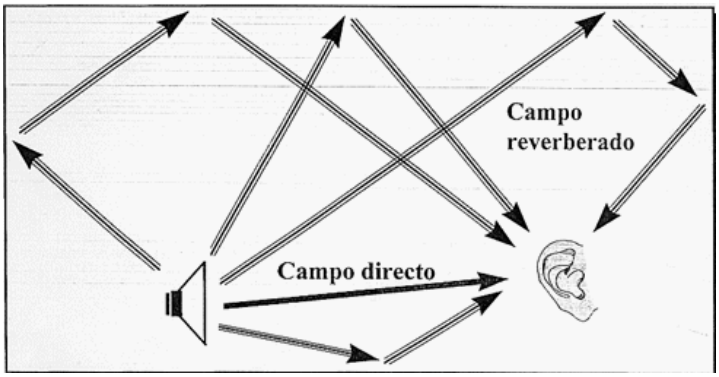


Ilustración 62: Explicación del efecto de reverberación

5. BIBLIOGRAFÍA

1. **Hearn, Donald y Baker, M. Pauline.** *Gráficos por computadora con OpenGL.* s.l. : Pearson Educación.

2. **Recuero López, Manue.** *Acústica arquitectónica aplicada.* s.l. : Ed. Paraninfo, 1999.

3. **Morcillo, Carlos Gónzalez.** *Programación Gráfica.* s.l. : Bubok.

4. **Oar, Manuel Abellanas.** *Envolvente convexa, triangulación de Delaunay y diagrama de Voronoi.*

5. **Wernecke, Josie.** *The Inventor Mentor:Programming ObjectÿOriented 3D Graphics.* s.l. : Addison-Wesley Publishing Company.

6. **Erich Gamma, Ralp Johnson, Richard Helm, John Vlissides.** *Design Patterns. Element of Reusable Object-Oriented Software.* s.l. : Addison Wesley.

7. **Stroustrup., B.** *El lenguaje de programación C++.* s.l. : Pearson Education, 2001.

8. **Revuelta, Francisco Ignacio.** *El poder educativo de los juegos online y de los videojuegos, un nuevo reto para la psicopedagogía en la sociedad de la información.*

9. **Prensky, Marc.** *Digital Game-Based Learning.*

10. **DARLEY, ANDREW.** *Visual Digital Culture.* s.l. : ANDREW DARLEY, 200.

11. **PrimeSense™.** *NITE 1.3 Algorithms notes.*

12. **Microsoft Research Cambridge & Xbox Incubation.** *Real-Time Human Pose Recognition in Parts from Single Depth Images.*

13. **Christian Plagemann, Varun Ganapathi, Daphne Koller,Sebastian Thrun.** *Real-time Identification and Localization.*

14. **Kwang-Jin Choi, Hyeong-Seok.** *Automatic Rigging and Animation of 3D Characters.* SIGGRAPH 2007.

15. **Loren Arthur Schwarz, Artashes Mkhitaryan, Diana Mateus, Nassir Navab.** *Human skeleton tracking from depth data using geodesic distances and optical flow.* Marzo 2012.

16. **Maldonado, José Gutiérrez.** *Aplicaciones de la realidad virtual en psicología clínica.*

17. **UCM, Universidad.** *Anova de un factor: Procedimiento.*

18. **Youding Zhu, Kikuo Fujimura.** *Constrained Optimization for Human Pose.*

19. **Mota, Rivera.** *La batería acústica.*

20. **Ibáñez, Clara Patricia Avella.** *Pruebas de Software.*

6. APÉNDICE

6.1. Enlaces

[1WE] Captura recogida de www.ellosnuncaloharian.com Visto en 08/10/2011

[K3D] Captura recogida de las screenshot http://k3dsurf.sourceforge.net/ 29/03/2012

[DES10] Captura recogida en el post de http://www.designaholic.mx/2010/02/la-tetera-de-utah.html Visto en 03/05/2012

[CYCAS] Captura de la aplicación http://www.cycas.de/ 02/10/2011

[ACAD] Captura de la aplicación http://www.sharewarebay.com/ 02/10/2010

[IBE] Captura del simulador www.ibersim.com Vista en 01/11/2012

[M3C] Captura recogida de HTTP://WWW.MONODE3CABEZAS.COM/ Vista en 10/11/2011

[DAC] Imagenes recogidas de la asignatura http://dac.etsii.urjc.es/docencia/RVA/ Vista en 04/06/2011

[ROC] Captura de www.rockband.com Vista en 10/10/2012

[CBS] Captura de un fotograma de la serie en www.cbs.com Vista en 09/08/2012

[GRU] Captura de un fotograma de la película "Gru, mi villano favorito".

6.2. Términos

FPS	Las imágenes por segundo (en inglés *frames per second*, FPS) es la medida de la frecuencia a la cual un reproductor de imágenes genera distintos fotogramas (*frames*). En informátia estos fotogramas están constituidos por un número determinado de imágenes, incidiendo en el rendimiento del ordenador que los reproduce.[1][3][10]
Keyframe	Keyframe es aquel frame clave en una secuencia de animación, donde se obtiene una posición clave para poder interpolar la animación sin especificar cada frame de la animación.[1][3][14]
Viewport	Es una región rectangular de la pantalla donde se pintará la imagen 2D que corresponde a la proyección 3D de una cámara. [3]
Static	Propiedad de un método o atributo en programación, que garantiza que existe únicamente una copia que compartirán todos los objetos de una misma clase.[7]
ToF	Tiempo de vuelo: Es el método de medir la distancia a través de la medición del tiempo de vuelo ida-retorno de una señal de sonido o luz[DAC] [12][13]
DoF	Grados de libertad. Números de parámetros que definen la configuración espacial de un cuerpo[DAC] [3]

6.3. **Formulario**

Este formulario ha sido entregado a los usuarios que se han reunido para probar la aplicación.

Edad: Género:

¿Sabe usted tocar una batería instrumental? Si No

En una escala del 0 al 10 responda a las siguientes preguntas, donde 0 corresponde a “Muy Pobre”, y 10 corresponde a “Muy conseguido”.

¿Cómo valoraría usted esta aplicación en cuanto al realismo?

0	1	2	3	4	5	6	7	8	9	10

¿Cómo valoraría usted esta aplicación en cuanto a la diversión?

0	1	2	3	4	5	6	7	8	9	10

¿Cómo valoraría usted esta aplicación en cuanto a la facilidad de uso?

0	1	2	3	4	5	6	7	8	9	10

Comentarios o sugerencias:

_____

_____

_____

_____

_____

7. TABLA DE ILUSTRACIONES

8. MANUAL DE USUARIO

8.1. Instalación

En el CD que se adjunta a la memoria en papel, se encuentra el directorio de la aplicación donde se incluyen todos los archivos necesarios para la ejecución excepto los drivers del dispositivo Kinect o Xtion. Estos drivers se necesitarán instalar en cada ordenador siendo imposible su portabilidad en el directorio de la aplicación.

El orden de instalación de estos paquetes será el siguiente:

1. OpenNI unstable Win32 *http://openni.org/*
2. PrimeSensor unstable Win32 *http://openni.org/*
3. NITE unstable Win32 *http://openni.org/*
4. Mod PrimeSensor for Kinect unstable Win32 *https://github.com/avin2/SensorKinect*

Tenga en cuenta, que estas instrucciones se han escrito en Octubre de 2012 y los enlaces, procedimientos o versiones pueden estar obsoletos en un futuro.

Si los archivos necesarios para la aplicación están correctamente en la misma carpeta que la aplicación, funcionará correctamente.

Se incluirá en el CD el código de la aplicación para modificaciones adicionales. En el siguiente apartado se especificará las librerías adicionales que se necesitan para compilar.

8.2. Compilación

8.2.1. Prerrequisitos

- Coin3d 3.1.0 www.coin3d.org
- SoWin www.coin3d.org
- SoImage www.coin3d.org
- OpenNI, NITE, PrimeSensor www.openni.org
- OpenAL 1.0 http://connect.creativelabs.com/openal/default.aspx
- OpenCV 2.1 http://opencv.willowgarage.com/wiki/
- OgreSDK http://www.ogre3d.org/
- Log4cxx http://logging.apache.org/log4cxx/

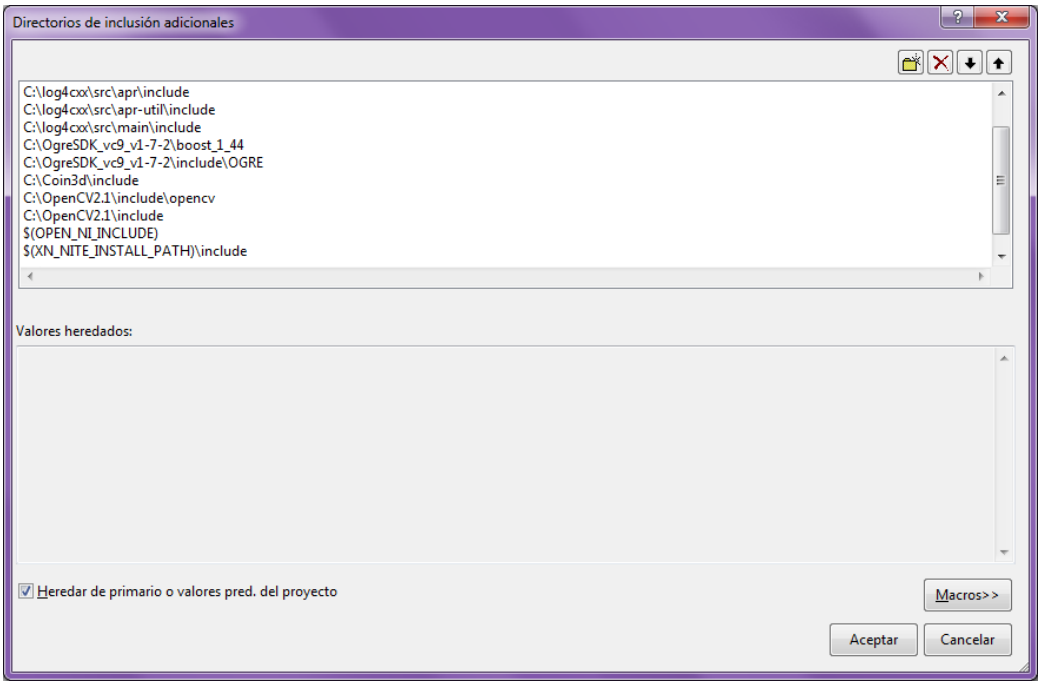
8.2.2. Configuración Visual Studio

A continuación se muestra los directorios que hay que configurar en el Visual Studio para poder compilar:

Carpetas Include:

Estas son las carpetas include que hay que incluir en el proyecto.

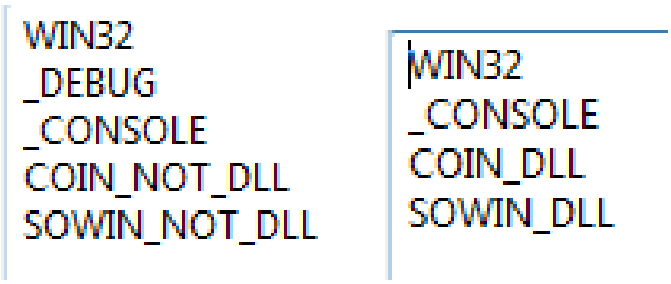
Propiedades->C/C++->General->Directorios de inclusión adicionales



Preprocesador:

La primera imagen para Debug, y la segunda para Release.

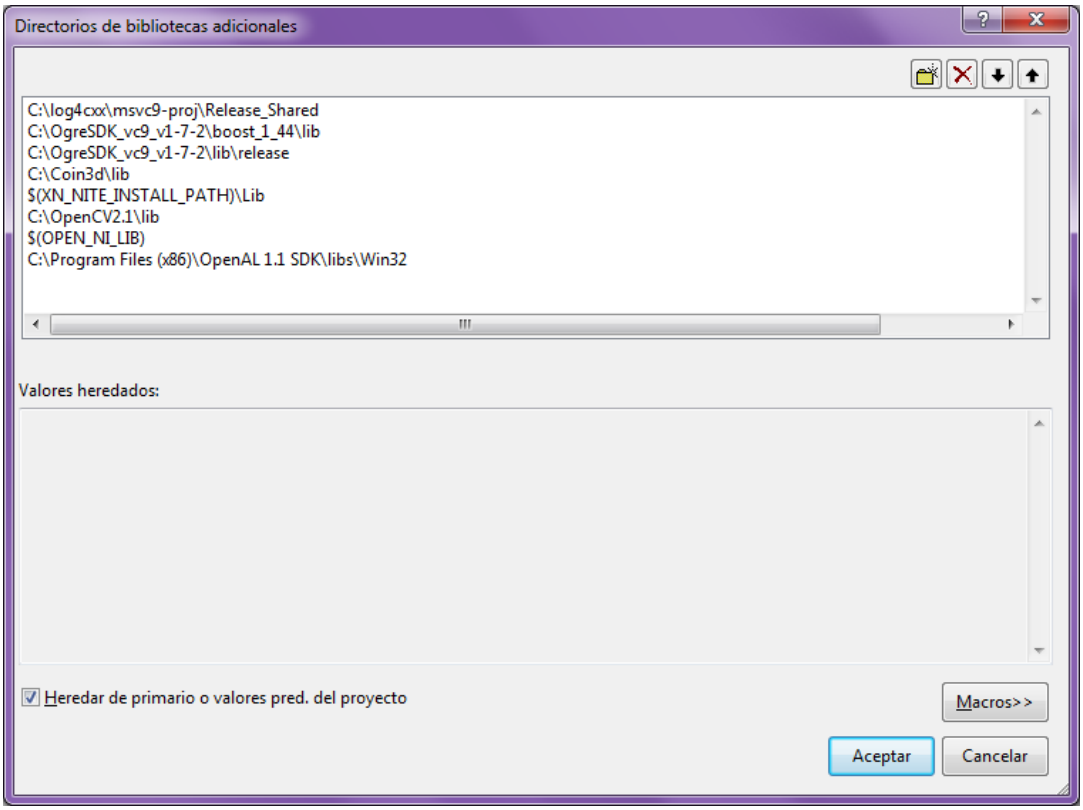
Propiedades->C/C++->Preprocesador->Definiciones



Bibliotecas Vinculador

Estos directorios corresponden al lugar donde se encuentran los .lib

Propiedades-> Vinculador->General->Directorios de bibliotecas adicionales.

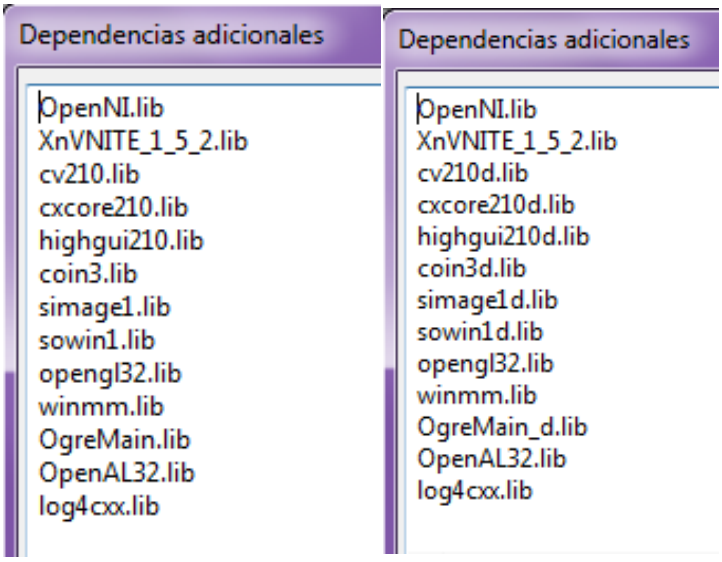


Dependencias Adicionales

Los ficheros .lib, corresponden a los dll.

Propiedades->Vinculador->Entrada->Dependencias adicionales

La primera imagen corresponde a Release y la segunda a Debug



8.3. Archivos Importantes

Imágenes:	**Sonidos:**	**DLL:**
Publico.jpg	Snare.wav	Sowin1d.dll
Mask.png	Midtom.wav	Simage1d.dll
Mask2.png	Lowtom.wav	OgreMain_d.dll
	Hitom.wav	Libsndfile-1.dll
Modelos:	Cymbal.wav	Highgui210d.dll
Busto.wrl		Cxcore210d.dll
Drumconparede.wrl	**Otros:**	Cv210d.dll
Mano.wrl	Configurate.cfg	Cvaux210d.dll
Cabeza.wrl	Si-Log.txt	Coin3d.dll
	SamplesConfig.xml	

Configurate.cfg tiene las configuraciones relativas al log4cxx.

Si-Log.txt es el fichero donde se vuelca el log de la aplicación.

SamplesConfig.xml es el archivo de configuración del contexto de OpenNI

En los dll, aquellos que tienen una d al final, son los respectivos a debug, los correspondientes a release, normalmente son sin d.

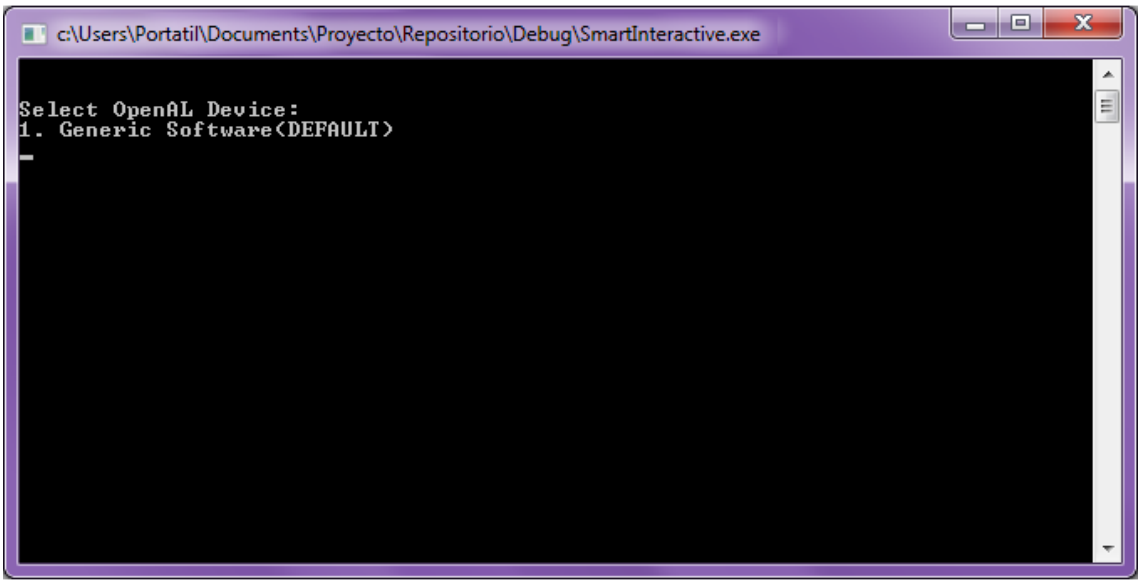
8.4. Juego

8.4.1. Preparación inicial

Es recomendable asegurarse que el Kinect funciona correctamente con los ejemplos que incluye la librería NITE o OpenNI. Antes de iniciar el juego, el usuario deberá poner una banqueta, preferiblemente sin reposabrazos a una distancia de 1,5m aproximado del Kinect. El Kinect deberá estar situado a unos 0,8m aproximado del suelo. Recuerde, que la aplicación toma una foto de la habitación para mostrarla en la pared trasera del mundo virtual.

8.4.2. Inicio del juego

Cuando iniciemos la aplicación, nos pedirá la confirmación de que dispositivo de sonido utilizaremos. En la mayoría de los casos solo existe un único dispositivo.



A continuación, el juego lanzará el visualizador, y la imagen de feedback. Se pueden colocar en cualquier posición, y ya estará lista la aplicación esperando la entrada de un usuario.

Cuando un usuario entra en la escena, la aplicación tardará unos 2 segundos en capturar el esqueleto del usuario. En este caso, podrá mover los brazos en la escena para hacer sonar la batería.

Para salir de la escena, levantese y salga de la visión del Kinect.

Para introducir un usuario nuevo, salga de la escena, y proceda a entrar el siguiente usuario.

8.4.3. Cambiar la cámara global

Selecciona en el Viewer, el botón  y pulsa M. La cámara cambiara del plano front, al plano top.

8.5. Posibles fallos

P: Sale una ventana de diálogo diciendo: *"Falta XXXX.dll"*

R: Revise todos los paquetes están bien instalados, los dll se encuentran en la carpeta de la aplicación o en el directorio esta en el Path.

P: Sale la ventana de MSDOS negra.

R: Revise el Si-Log.txt para detectar el error, es posible que el kinect no funcione correctamente.

P: No me captura o tarda mucho tiempo en capturar.

R: Haga las siguientes comprobaciones: No haya luces directas hacia el dispositivo (luces, el sol,…). No haya objetos en movimiento en la escena. Cuidado con posibles oclusiones

P: El mensaje que produce el error es similar a: *"Error, not production node found."*

R: Asegurese que el SamplesConfig tenga el siguiente contenido.

```
<OpenNI>
	<Licenses>
	  <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4=" />
	</Licenses>
	<Log writeToConsole="false" writeToFile="true">
		<!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
		<LogLevel value="2"/>
		<Masks>
			<Mask name="ALL" on="true"/>
		</Masks>
		<Dumps>
		</Dumps>
	</Log>
	<ProductionNodes>
		<Node type="Image" name="Image1"><!—Enciende la cámara RGB-- >
			<Configuration>
				<MapOutputMode xRes="640" yRes="480" FPS="30"/>
				<Mirror on="true"/>
			</Configuration>
		</Node>
		<Node type="Depth" name="Depth1"><!—Enciende la cámara Depth→
	<Configuration>
				<MapOutputMode xRes="640" yRes="480" FPS="30"/>
				<Mirror on="true"/>
			</Configuration>
		</Node>

		<Node type="User" /> <!—Nodo para reconocer el skeleto→
	</ProductionNodes>
</OpenNI>
```