PROYECTO TRABAJO FINAL DE CARRERA

EDITOR INTERACTIVO DE POLICUBOS

Alumno: Enrique Nucete Jiménez
Estudio: Ingeniera Técnica de Sistemas.
Escola Politècnica Superior, Universitat De Girona

Director: Ismael García
Tutor: Gustavo Patow
Departamento: Informàtica i Matemàtica Aplicada
Área: Informàtica i Matemàtica Aplicada
Escola Politècnica Superior, Universitat De Girona

Convocatoria (mes/año): Setiembre 2012

Contenido

Índice de figuras

1 Introducción

Hoy en día existen numerosas técnicas para aplicar texturas sobre objetos 3D genéricos, pero los mecanismos para su creación son, en general, o bien complejos y poco intuitivos para el artista, o bien poco eficientes en aspectos como obtener un texturado global sin costuras. Recientemente, la invención de los policubos ha abierto un nuevo espectro de posibilidades a la hora de realizar estas tareas, e incluso otras como animación y subdivisión, de crucial importancia para industrias como el cine o los videojuegos. Desafortunadamente, no existen herramientas automáticas y editables que permitan generar el modelo de policubos base.

Un policubo es una agregación de cubos idénticos de forma que cada cubo tiene como mínimo en común una cara con otro cubo. Con la agrupación de estos cubos se pueden generar diferentes figuras espaciales.

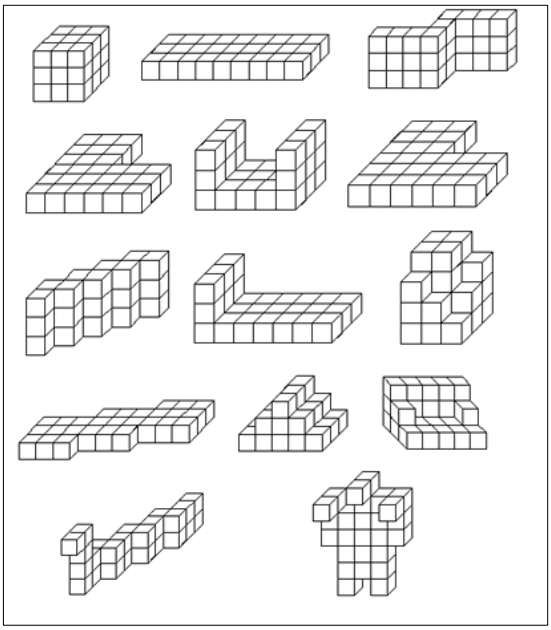


Figura 1 Ejemplos de policubos

Como puede observarse en la figura 1, gracias a los policubos podremos generar una representación esquemática y regular de los objetos 3D, para conseguir así una fácil manipulación y procesado por parte del usuario.

1.1 Motivación Personal

Mi motivación para realizar este proyecto ha sido el deseo de analizar, diseñar e implementar y, al mismo tiempo, hacer una herramienta que pueda ser útil para la gente. Pienso que un proyecto de final es un escenario perfecto que me permite trabajar al lado de un tutor experimentado y, gracias a su supervisión, podremos llegar al objetivo marcado.

Uno de mis objetivos una vez acabado el proyecto seria poder poner los resultados en la página de SourceForge[1], ya que me parece interesante que la gente lo pueda usar y además gracias a los consejos de los usuarios, poder ir mejorando la herramienta.

Creo que así podría empezar a abrirme paso en el mundo de la programación y empezar a hacer y/o colaborar en otros proyectos de mayor envergadura.

1.2 Propósitos y objetivos

El objetivo es desarrollar una herramienta para la creación y edición interactiva de un modelo de policubos a partir de un objeto tridimensional, la cual proporcionara una libertad y control al usuario no existente en las herramientas actualmente disponibles.

1.3 Estructuración del documento

Este documento se ha organizado en 15 capítulos, que son los siguientes:

1. Introducción, motivación, propósito y objetivos del proyecto. En este capítulo se explicará el porqué del desarrollo de este proyecto, cuáles son los objetivos propuestos y cómo se ha organizado el desarrollo.

2. Estudio de viabilidad. En este capítulo se justifican los parámetros que hacen posible el desarrollo del proyecto.

3. Metodología. Este capítulo contiene una explicación de la metodología utilizada.

4. Planificación. En esta etapa se define la estrategia seguida para llegar a los objetivos planteados.

5. Marco de trabajo y conceptos previos. En este capítulo se describen los diversos aspectos relacionados con el desarrollo general del proyecto, que ayudarán a entender mejor los siguientes capítulos. También se tratarán las principales acciones desarrolladas durante las primeras etapas de la realización de la herramienta. Se incluirán los pasos de estudio y aprendizaje de conceptos que se hayan utilizado para el desarrollo.

6. Requisitos del sistema. En este capítulo se definen los requerimientos del software, los cuales recogen, a grandes trazos, los objetivos de la herramienta juntamente con las funcionalidades que se quieren obtener. Este documento permite entender los elementos que rodean la herramienta de modelado que se intenta construir.

7. Estudios y decisiones. Esta sección contiene una descripción de las herramientas utilizadas, con sus características y el uso que se les ha dado, tanto de librerías y motores como de software.

8. Análisis y diseño del sistema. Este apartado proporciona una comprensión precisa de las necesidades del sistema. Es decir, se encarga del estudio y descripción analítica

[1] SourceForge es un sitio web de colaboración para proyectos de software.

del problema a resolver, todavía sin llegar a planear una solución definitiva. Usando las estrategias del software, en esta sección se traducen los requerimientos nombrados en capítulos anteriores a un lenguaje más formal. La parte de diseño permite aumentar el nivel de especificación, y realizar un esquema de implementación del sistema mediante diversas herramientas de programación orientada a objetos.

9. Implementación y pruebas. En este capítulo se dan a conocer cómo se ha construido la aplicación, las clases y los métodos implementados que resultan más significativos para la comprensión del funcionamiento de la herramienta.

10. Implantación y resultados. En este capítulo se muestran pruebas de ejecución de la herramienta, se muestran imágenes, incluyendo interfaces, y todo aquello que se pueda visualizar del conjunto que ha sido implementado.

11. Conclusiones. En este apartado se expondrán las conclusiones extraídas una vez finalizado el proyecto.

12. Trabajo futuro. En esta sección se expone todo aquello que se podría mejorar en la herramienta.

13. Bibliografía. Este capítulo contiene las referencias utilizadas para el desarrollo del proyecto.

14. Anexos. Esta sección contiene las definiciones de los tecnicismos más frecuentemente utilizados, así como explicaciones y experimentos no indispensables para la comprensión global del proyecto.

2 Estudio de viabilidad

Para desarrollar el proyecto no se requiere de gran infraestructura y los costes de estructura son limitados.

El material con que inicialmente se contaba es el siguiente:

- Ordenador con sistema operativo Windows (inicialmente XP, posteriormente se cambiaría a Windows 7).
- Entorno de compilación y desarrollo en lenguaje C++ (Visual Studio 2008)
- Tarjeta gráfica con soporte para las APIS[2] de programación grafica (Direct3D[3] y OpenGL[4]).

2.1 Recursos Humanos

En la creación de esta herramienta se necesita como mínimo dos miembros un Analista/Diseñador y un programador.

2.2 Evaluación previa de costes y medios

Para el desarrollo del sistema es necesario llevar a cabo un estudio de viabilidad, teniendo en consideración la viabilidad económica, la viabilidad técnica y la viabilidad legal. Debido a que la herramienta presenta riesgos bajos y pocos problemas legales, solo se consideran las siguientes áreas:

- Estudio de viabilidad tecnológica.
- Estudio de viabilidad económica.

2.2.1 Estudio de Viabilidad tecnológica

El estudio de la viabilidad tecnológica empieza el planteamiento de las siguientes preguntas: ¿Qué tecnologías se requieren para conseguir la funcionalidad y el nivel de rendimiento que es necesario en la herramienta? ¿Qué nuevos materiales, métodos o procesos se requieren? ¿Cómo afectarán al coste de desarrollo a estos elementos de tecnología?

[2] API es una interfaz de programación de aplicaciones.
[3] Direct3D es un conjunto de bibliotecas para multimedia, propiedad de Microsoft. Consiste en una API para la programación de gráficos 3D.
[4] OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Se dispone de los requisitos mínimos de hardware y software para realizar el proyecto, es decir un ordenador para poder prototipar y desarrollar la herramienta objeto del proyecto.

Como resultado del análisis técnico, se concluye que la Universidad de Girona cuenta con los recursos tecnológicos necesarios para implantar la herramienta. Este contexto es un paso adelante en el desarrollo del proyecto. Los recursos de hardware y software cubren las especificaciones que se requerirán para el buen funcionamiento de la herramienta.

2.2.2 Estudio de Viabilidad económica

La valoración del análisis económico será separada en dos partes: los costes de recursos humanos y los costes de la maquinaria.

2.2.2.1 Costes de Recursos humanos

Para calcular el coste de los recursos humanos se ha asociado un perfil o rol de trabajador para cada una de las tareas. De este modo tenemos dos perfiles diferentes, cada uno con los costes por hora, siguientes:

- Coste analista/diseñador: 35 € / hora.
- Coste programador: 30 € / hora.

Tarea	Perfil	Horas	Coste
Lectura e investigación	Analista / Diseñador	10	300
Estudio OGRE 3D	Analista / Diseñador	10	300
Diseño de algoritmos	Analista / Diseñador	30	900
Implementación de algoritmos	Programador	30	900
Pruebas	Programador	10	300
Optimizaciones	Programador	10	300
Memoria	Analista / Diseñador	40	1200
Total			4200

2.2.2.2 Costes de la maquinaria

En cuanto a los costes de la maquinaria, habría que tener en cuenta los costes de los ordenadores, y el material necesario para realizar todos los pasos a realizar. Se incluye el material que deberían utilizar los otros miembros del equipo para la realización correcta del proyecto.

Componente	Unidades	Precio
Ordenadores	2	1600
Libros de documentación	8	200
Total		3400

2.2.2.3 Costes de software

En cuanto a costes de software se ha optado por buscar versiones gratuitas y/o libres o bien versiones licenciadas para estudiantes.

Software	Licencia	Precio
Visual C++ 2008 Express	Gratuita	0
Gimp	Libre	0
Grantt	Libre	0
Office 2010 Hogar y estudiantes	Privada	99

De este modo, el coste total del proyecto sería de unos 7699€.

3 Metodología

Para la realización de este proyecto no se ha seguido una metodología de trabajo estándar, sino que se ha elegido y utilizado una metodología personalizada y que fue planteada con el tutor.

Esta metodología es la misma que se sigue en el proyecto **skyline**Engine, y los pasos que sigue son los siguientes:

1. Elegir el trabajo a desarrollar.
2. Decidir el lenguaje de programación y herramientas a utilizar.
3. Aprender el lenguaje de programación y las herramientas escogidas.
4. Estructurar el trabajo en partes según las funciones que se tengan que realizar.
5. Desarrollar la parte correspondiente siguiendo el orden de la estructura del trabajo.
6. Hacer comprobaciones para confirmar que el funcionamiento es correcto al finalizar cada parte.
    a. Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la ultima parte desarrollada o en las anteriores, si es necesario.
    b. Si al hacer las comprobaciones el resultado es el deseado, se desarrollará la siguiente parte volviendo al punto 5. Una vez se hayan finalizado todas las partes con sus respectivas comprobaciones, se iniciará el punto 7.
7. Unir todas las partes desarrolladas y comprobar que el funcionamiento es correcto.
    a. Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la última parte desarrollada o en las anteriores, si es necesario.
    b. Si al realizar las comprobaciones el resultado es el esperado, se iniciará el punto 8.
8. Generar diferentes modelos de ejemplo para comprobar que el funcionamiento es el correcto.
    a. Si al hacer las comprobaciones el resultado no es el deseado, se volverá al punto 5 para realizar los cambios oportunos en la última parte desarrollada o en las anteriores, si es conveniente.
    b. Si al realizar las comprobaciones el resultado es el esperado, se iniciará el punto 9.
9. Presentar documentación.

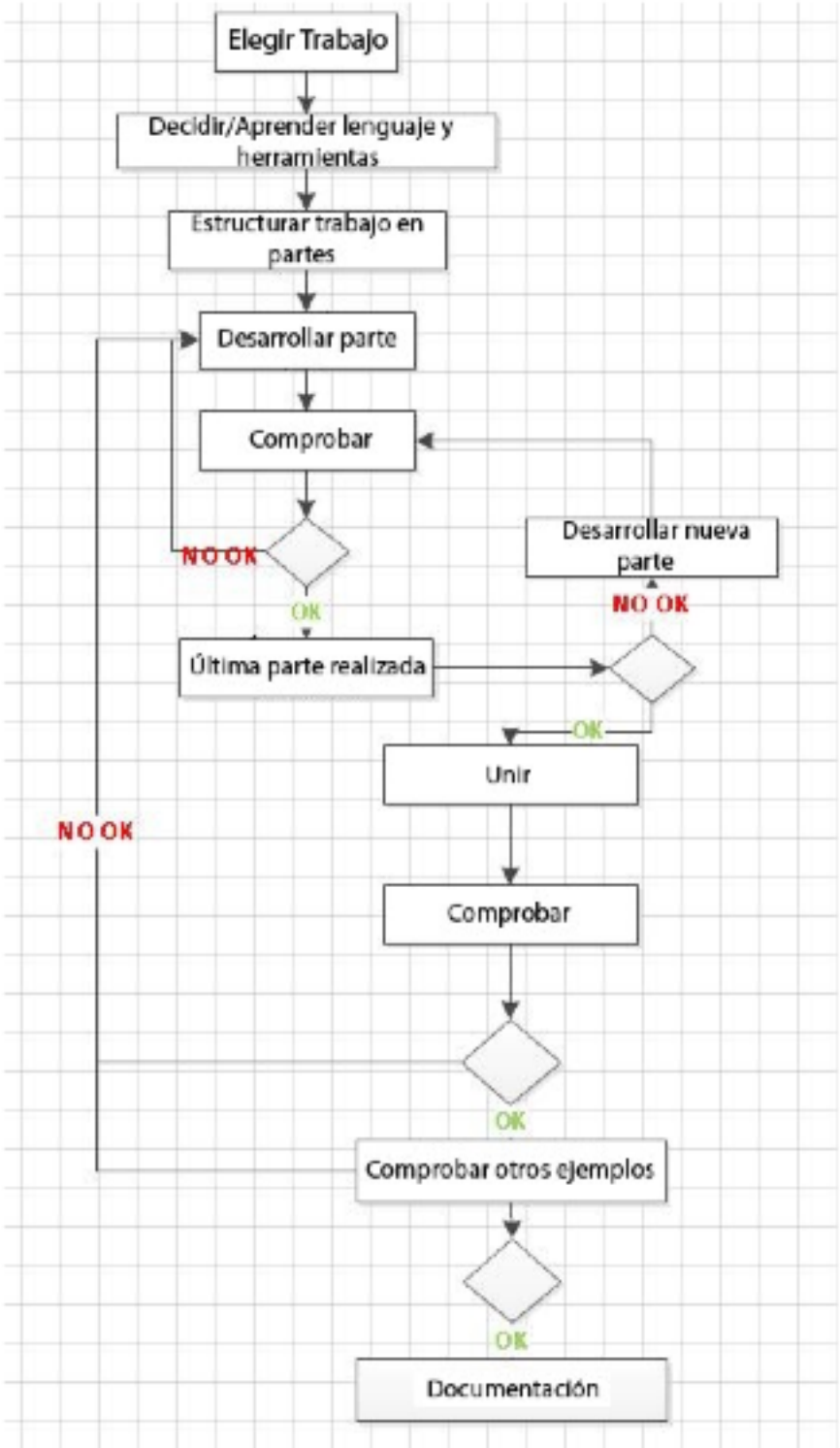


Figura 2 Metodología skyline Engine

4 Planificación

Para todo proyecto, siempre es importante hacer una valoración del tiempo empleado. Este capítulo describe la planificación que se ha seguido durante la elaboración del proyecto.

4.1 Plan de trabajo

Se planificaron las tareas que se debían realizar, y de esta forma se crearon 11 tareas.

4.2 Tareas planificadas

4.2.1 Introducción al Ogre 3D

Para poder desarrollar la herramienta, lo más importante es utilizar unas librerías que nos permitan trabajar fácil y eficazmente con la tarjeta gráfica. Para el desarrollo de este proyecto se realizó un análisis de los motores gráficos existentes. Ogre3D fue el motor elegido.

4.2.2 Mouse Picking, Render to Texture, Vertex Shader, Image Debugger, Intersección rayo triangulo

En esta fase del proyecto se aprendió a cargar objetos en el entorno Ogre y como poder obtener las coordenadas 3D de dichos objetos.

4.2.3 Control conjunto cubos

En esta tarea consiste en la implementación de la estructura de datos que gestionará la información de la representación del policubo.

4.2.4 Carga y grabación del conjunto de cubos

En esta fase se estudió cual era la mejor forma de guardar la estructura de datos en fichero para poder recuperar más adelante trabajo hecho anteriormente.

4.2.5 Interfaz de usuario

En esta fase se ha creado una interfaz amigable para el usuario en el que pueda modificar diferentes parámetros de la estructura de cubos, y proporciona un control extra sobre la herramienta.

4.2.6 Documentación

La documentación es una tarea constante donde se recopilan todos los apuntes, datos e ideas que surgen a lo largo del proyecto. Esta documentación se ha llevado a cabo durante todo el proyecto.

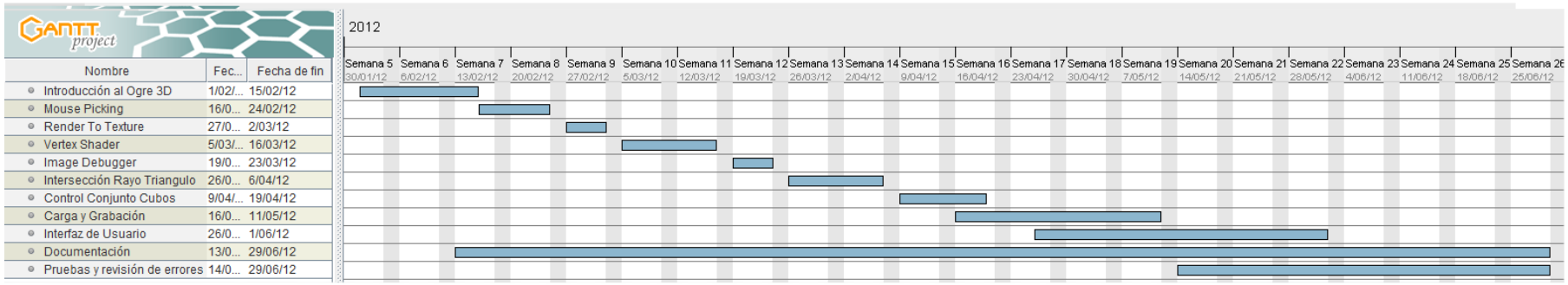


Figura 3 Diagrama de Gantt de la planificación

5 Marco de trabajo y conceptos previos

En este apartado se explicará el trabajo realizado durante las primeras etapas del proyecto:

- Elegir un motor gráfico 3D.
- Diseñar estructura de datos para el policubo.
- Elegir una interfaz de usuario.

5.1 Elegir un motor gráfico 3D

Motor grafico es un término que hace referencia a una serie de rutinas de programación que permiten el diseño, la creación y la representación de un entorno 3D.

Actualmente existen muchos motores gráficos, algunos de código libre y otros no, que permiten al usuario crear sus propias aplicaciones con una mayor facilidad.

La elección del motor grafico ha estado condicionada básicamente por los siguientes requisitos:

1. El motor gráfico debe ser libre.
2. El motor gráfico debe ser eficiente.
3. Tiene que tener una comunidad activa y/o foros para poder resolver dudas.

El motor gráfico elegido a sido Ogre 3D ya que cumple los requisitos anteriormente comentados.

5.1.1 Ejemplos de motores gráficos

5.1.1.1 OGRE 3D

OGRE 3D (acrónimo del inglés Object-Oriented Graphics Rendering Engine) es un motor de renderizado 3D orientado a escenas, escrito en el lenguaje de programación C++.

Sus bibliotecas evitan la dificultad de la utilización de capas inferiores de librerías gráficas como OpenGL y Direct3D, y además, proveen una interfaz basada en objetos del mundo y otras clases de alto nivel.

El motor es software libre, licenciado bajo MIT (*Massachusetts Institute of Technology*) y con una comunidad muy activa.

5.1.1.2 Crystal Space

Crystal Space es un framework para el desarrollo de aplicaciones 3D escrito en C++ por Jorrit Tyberghein. Fue fundado el 26 de agosto del 1997.

Crystal Space se usa típicamente como motor de juego pero el framework es más general y puede ser usado para cualquier tipo de visualización 3D. Es software de código abierto.

5.1.1.3 OpenGL

Open Graphics Library es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics Inc. (SGI) en 19922 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.

5.1.1.4 Direct3D

Direct3D es parte de DirectX (conjunto de bibliotecas para multimedia), propiedad de Microsoft. Consiste en una API para la programación de gráficos 3D. Está disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox y Xbox 360.

El objetivo de esta API es facilitar el manejo y trazado de entidades gráficas elementales, como líneas, polígonos y texturas, en cualquier aplicación en 3D, así como efectuar de forma transparente transformaciones geométricas sobre dichas entidades. Direct3D provee también una interfaz transparente con el hardware de aceleración gráfica.

Se usa principalmente en aplicaciones donde el rendimiento es fundamental, como los videojuegos, aprovechando el hardware de aceleración gráfica disponible en la tarjeta gráfica.

El principal competidor de Direct3D es OpenGL, desarrollado por Silicon Graphics Inc.

5.2 Elegir una interfaz de usuario

Una parte importante de la herramienta será la interfaz de usuario, en esta fase del proyecto se decidió que dicha interfaz como mínimo debería tener unos botones para guardar y poder recuperar el policubo y unos botones para el control de los cubos. La interfaz elegida ha sido el CEGUI.

5.3 Los shaders

Desde que comenzó la "revolución 3D" en el ámbito de los juegos por ordenador a mediados de la década de los 90, la tendencia de tecnología aplicada a estos temas a sido trasladar la faena de procesamiento de los gráficos tridimensionales, des de la CPU a la GPU. Lo que se quiere conseguir con este cambio es liberar a la CPU de faenas orientadas a gráficos, para poder dedicar más tiempo y recursos a otro tipo de cálculos.

Al principio esto se hacia a muy bajo nivel, en lenguaje ensamblador. Esto otorgaba mucho poder al programador, pero era difícil de programar.

Para conseguir traspasar a la GPU estas tareas de forma más fácil se crearon los shaders.

Un shader se define como un conjunto de instrucciones escritas en lenguaje de alto nivel, que se compila y se envía a la GPU para que esta procese y realice las tareas dadas.

Como a lenguaje de shader de alto nivel tenemos el Cg, que pertenece a la compañía Nvidia.

5.3.1 Proceso de visualización

En la actualidad, las tarjetas graficas constan de tres unidades que permiten incorporar elementos programables a las etapas de transformación de vértices y transformación del color de cada pixel. Estas unidades son el Vertex Shader, Geometry Shader, y el Pixel Shader.

El Vertex Shader se encarga de la transformación de vértices de una escena. Recibe solo un vértice cada vez y solo lo puede modificar. No puede poner de nuevos ni quitar.

El Pixel Shader se encarga de la transformación de la escena. Calcula el color de cada Pixel individualmente. Son usados básicamente para calcular iluminaciones.

El Geometry Shader se ejecuta después del Vertex Shader. Reciben una primitiva, como por ejemplo triángulos y, a poder ser, con información de adyacencia. Por ejemplo, si recibe un triangulo, la entrada del shader serán los tres vértices.

El hecho de separarlos proporciono a los programadores una mayor libertad a la hora de diseñar gráficos en 3D, ya que pueden tratarse cada pixel, cada triangulo y cada vértice por separado. De esta manera se pueden crear efectos especiales e iluminación más detallados.

6 Requisitos del sistema

Los requisitos del sistema se dividen en dos grupos los funcionales y los no funcionales:

- Requisitos funcionales define el comportamiento del software: cálculos, detalles técnicos, manipulación de datos y otras funcionalidades específicas que muestran cómo los casos de uso serán llevados a la práctica.
- Requisitos no funcionales es, en la ingeniería de sistemas y la ingeniería de software, un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales. Por tanto, se refieren a todos los requisitos que ni describen información a guardar, ni funciones a realizar. Algunos ejemplos de requisitos no funcionales típicos son los siguientes: rendimiento, disponibilidad, seguridad, accesibilidad, usabilidad, estabilidad, portabilidad, costo, operatividad, interoperabilidad, escalabilidad, concurrencia, mantenibilidad.

A continuación se describen en detalla los dos tipos de requisitos comentados.

6.1 Requisitos funcionales

Lo requisitos funcionales de la herramienta serán los siguientes:

- Poder construir una estructura de policubo manualmente a partir de una interacción con el teclado y el ratón.
- Poder guardar y cargar la estructura de policubo.
- Poder borrar cubos de la estructura de policubo.
- Poder elegir la orientación de la estructura de policubo.
- Poder elegir el tamaño de los cubos.

6.2 Requisitos no funcionales

Para la herramienta que se implementara se prestara atención a los siguientes requisitos no funcionales:

- Rendimiento, se optimizara la herramienta para que no utilice más memoria de la necesaria en la estructura de datos. Con ello se obtendrá un mejor rendimiento.
- Accesibilidad, se hará una herramienta que se pueda utilizar en cualquier ordenador actual y no sea necesario tener una estación de trabajo para poder modelar la estructura de policubo.
- Usabilidad, se diseñara una interfaz para el usuario que permita el manejo de la herramienta con más facilidad.

La implementación y las pruebas de la aplicación se han llevado a cabo sobre 2 ordenadores con las siguientes características.

	Ordenador 1	Ordenador 2
Procesador	Intel Core 2 Duo E6850 @ 3GHz	Intel Core 720QM @ 1.60 GHz
Tarjeta Grafica	Nvidia GeForce 8800 GTS	Nvidia GeForce 230M
Memoria	4 GB	4 GB
Sistema Operativo	Windows 7 Profesional	Windows 7 Home Premium

La aplicación está implementada en C++.

7 Estudio y decisiones

En este capítulo se describirán las librerías y el software utilizado para realizar el proyecto. También se mencionará el software utilizado para crear la documentación.

7.1 Sistema Operativo en el que se ha implementado la aplicación

El sistema operativo en el que se ha implementada la herramienta ha sido Windows 7. Se ha probado en las dos siguientes versiones Home Premium y Profesional, ambas en su versión de 64 bits. Se desestimando Windows XP ya que actualmente no se puede obtener una licencia y para el 2014 se dejara de dar oficialmente soporte, aunque la herramienta puede funcionar igualmente en este sistema operativo.

7.2 Software utilizado

En este apartado se explica el software utilizado para desarrollar el proyecto tanto a nivel de implementación de la herramienta como de documentación del proyecto.

7.2.1 Microsoft Visual Studio 2008 Express

Microsoft Visual Studio Express Edition es un programa de desarrollo en entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows, desarrollado y distribuido por Microsoft Corporación. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Es de carácter gratuito y es proporcionado por la compañía Microsoft Corporación orientándose a principiantes, estudiantes y aficionados de la programación web y de aplicaciones, ofreciéndose dicha aplicación a partir de la versión 2005 de Microsoft Visual Studio.

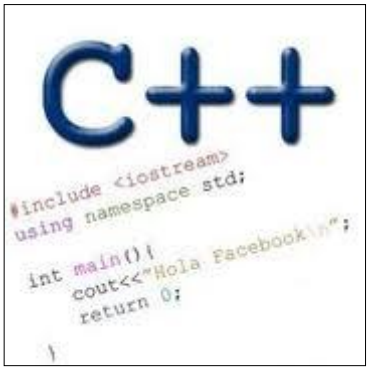
Visual Studio Express permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .net 2002, se incorpora la versión Framework 3.5 y Framework 4.0 para las ediciones 2005, 2008 y 2010). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles. Cabe destacar que estas ediciones son iguales al entorno de desarrollo comercial de Visual Studio Professional pero sin características avanzadas.

Las ediciones que hay dentro de cada suite son:

- Visual Basic Express Edition.
- Visual C# Express Edition.
- Visual C++ Express Edition.
- Visual J# Express Edition (Desaparecido en Visual Studio Express 2008).
- Visual Web Developer Express Edition

La edición que se ha utilizado para hacer el proyecto ha sido **Visual C++ Express**. Se ha utilizado este software ya que tiene una licencia para estudiantes y está respaldado por Microsoft, además de tener una comunidad muy activa.

7.2.2 C++



C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.

Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos.

Una particularidad del C++ es la posibilidad de redefinir los operadores, y de poder crear nuevos tipos que se comporten como tipos fundamentales.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

7.2.3 Cg

El Cg es un lenguaje de alto nivel creado por NVIDIA con la colaboración de Microsoft, para programar Vertex Shaders, Geometry Shaders, y Pixel Shaders.

El Cg esta basado en el lenguaje de programación C y, todo que comparten la misma sintaxis, algunas características de C se modificaron y se pusieron nuevos tipos de datos para que el Cg fuera mas adecuado para programar GPU's. El Cg solo sirve para programar tarjetas graficas, no substituye ningún otro lenguaje de alto nivel.

7.2.4 Microsoft Office 2007

Microsoft Office 2007 es una versión de la suite ofimática Microsoft Office de Microsoft y sucesora de Microsoft Office 2003. Originalmente conocido como Office 12 durante su ciclo beta, fue lanzado el 30 de noviembre de 2006 al mercado empresarial y el 30 de enero de 2007 al público en general, coincidiendo con el lanzamiento oficial de Windows Vista. Office 2007 incluye nuevas características, la más notable es la nueva interfaz gráfica llamada Office Fluent también conocido como "interfaz de cinta" (en inglés ribbon) que remplaza al menú y a la barra de herramientas que fueron características desde su inicio.

Office 2007 incluye nuevas aplicaciones y herramientas del lado servidor, de las cuales una sobresaliente es Groove, un sistema de colaboración y comunicación para pequeñas empresas que fue originalmente desarrollado por Groove Networks, hasta que Microsoft lo compró en 2005. También esta nueva versión incluye Microsoft Office Server 2007, un sistema de revisión en red de aplicaciones de Office, tales como Excel o Word.

Aunque Office 2007 incluye nuevas aplicaciones, pero sólo FrontPage fue eliminada por completo ya que sus sucesores fueron lanzados como suites independientes: SharePoint Designer y Expression Web.

La versión utilizada para documentar el proyecto es Microsoft Office 2007 Hogar y Estudiantes.

7.2.5 Gimp

GIMP es un programa de manipulación de imágenes que ha ido evolucionando a lo largo del tiempo, ha ido soportando nuevos formatos, sus herramientas son más potentes, además funciona con extensiones o plugins y scripts.

GIMP usa GTK+ como biblioteca de controles gráficos. En realidad, GTK+ era simplemente al principio una parte de GIMP, originada al reemplazar la biblioteca comercial Motif usada inicialmente en las primeras versiones de GIMP. GIMP y GTK+ fueron originalmente diseñados para el sistema gráfico X Window ejecutado sobre sistemas operativos tipo Unix. GTK+ ha sido portado posteriormente a Microsoft Windows, OS/2, Mac OS X y SkyOS.

GIMP permite el tratado de imágenes en capas, para poder modificar cada objeto de la imagen en forma totalmente independiente a las demás capas en la imagen, también pueden subirse o bajarse de nivel las capas para facilitar el trabajo en la imagen, la imagen final puede guardarse en el formato xcf de GIMP que soporta capas, o en un formato plano sin capas, que puede ser png, bmp, gif, jpg, etc.

Con GIMP es posible producir imágenes de manera totalmente no interactiva (por ejemplo, generar inmediatamente imágenes para una página web usando Scripts CGI) y realizar un procesamiento por lotes que cambien el color o conviertan imágenes.

7.2.6 PTRSCR

PTRSCR es un capturador de pantalla gratuito, las capturas también se realizan con la tecla Impr Pant, y pueden ser a pantalla completa, de un área que delimitemos mediante líneas con el ratón o marcando una zona con un rectángulo.

Una vez fotografiemos la parte del Escritorio de nuestra elección, PrtScr nos preguntará a dónde enviar el resultado con la captura flotando para que veamos cómo ha quedado.

Podremos enviar una copia al portapapeles o por correo electrónico, guardarla en un archivo de imagen, editarla, imprimir el resultado o descartarlo.

7.2.7 GANTT Project

Gantt project es una aplicación de software libre que permite realizar diagramas de Gantt. Un diagrama de Gantt es una herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

El diagrama de Gantt no indica las relaciones existentes entre actividades, pero la posición de cada tarea a lo largo del tiempo hace que se puedan identificar dichas relaciones e interdependencias.

7.3 Librerías utilizadas.

En este apartado una explicación de la librerías que se han utilizador para desarrollar el proyecto.

7.3.1 OGRE3D

Ogre3D (Object Oriented Graphic Rendering Engine) es una interfaz orientada a objetos para renderizar escenas 3D con independencia de la API gráfica utilizada Direct3D o OpenGL.

Las librerías de Ogre3D evitan la dificultad de la utilización de capas inferiores de librerías gráficas como OpenGL y Direct3D, y además, proveen una interfaz basada en objetos del mundo y otras clases de alto nivel.

Ogre3D está disponible para las siguientes plataformas: Windows, Mac OSX y Linux, y puede ser utilizado en gran variedad de aplicaciones tridimensionales aunque, su uso más común es el desarrollo de videojuegos.

Figura 4 Torchligth

Figura 5 oFusion

En la Figura 4 podemos ver Torchligth, que es un videojuego comercial desarrollado en Ogre3D, actualmente están desarrollando la segunda versión Torchligth 2.

En la Figura5 podemos ver Ofusion, que es una herramienta comercial diseñada para trabajar con 3ds Max también desarrollado con Ogre3d.

7.3.1.1 Componentes de Ogre3d

A continuación se hará una explicación detallada de los principales componentes de Ogre3d en la figura se puede ver un diagrama de las principales clases de Ogre3d y como están relacionadas entre ellas. Sólo aparecen las clases más básicas e importantes, pero no se explicaran todas ya que no son necesarias para entender la implementación del proyecto.

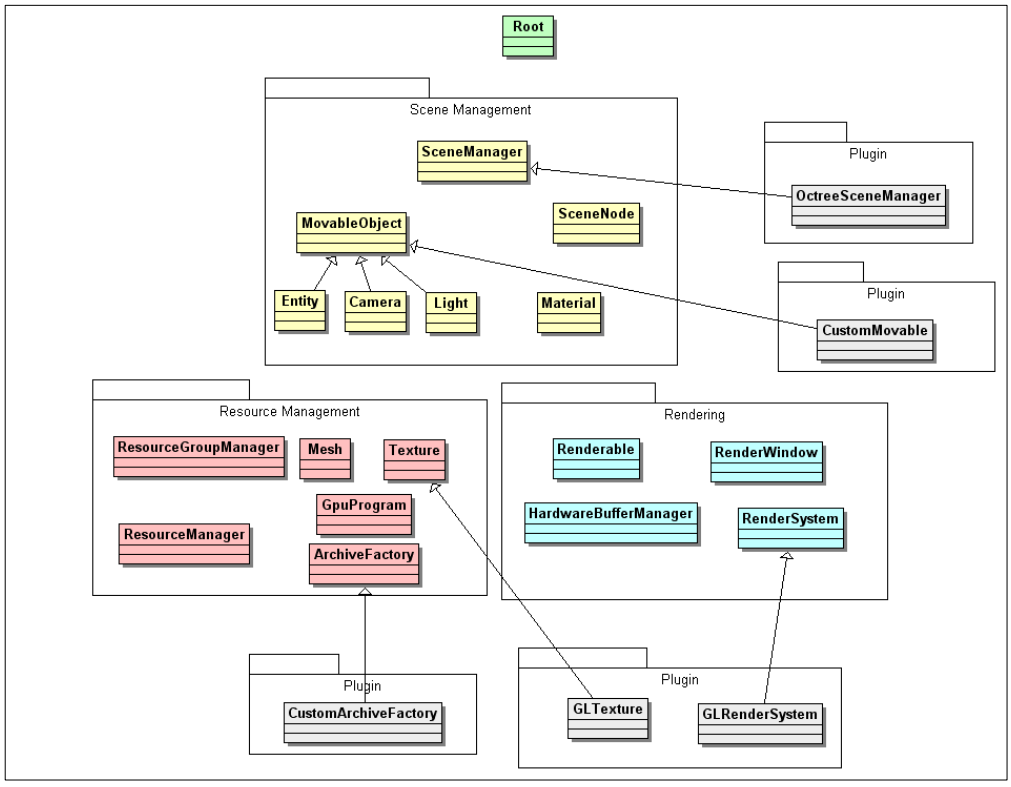
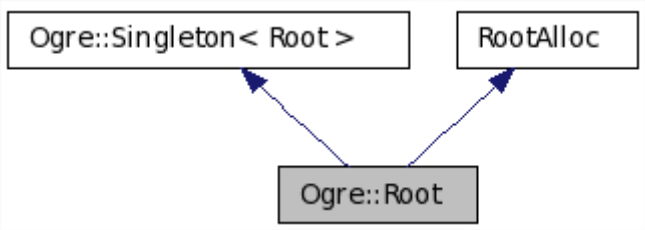


Figura 6 Diagrama de clase básicas de Ogre3d

Clase Root

La clase Root es el punto de entrada del sistema Ogre3d. Este objeto debe ser el primero en crearse, y el último en destruirse. Normalmente debe ser creado cuando arranca una aplicación basada en Ogre3d el primer paso es instanciar este objeto. Igualmente, en la finalización de la aplicación el último paso a realizar es la eliminación de este objeto.

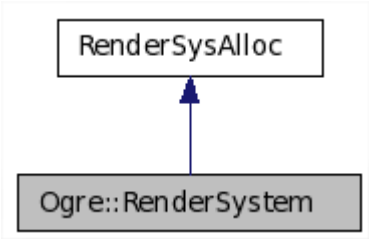
La clase Root permite la configuración del sistema, permite especificar la resolución de pantalla, la profundidad de color, opciones de pantalla completa, etc. Además, la clase Root es el método utilizado para obtener los punteros de otros objetos del sistema, como el del gestor de la escena SceneManager, el sistema de renderización RenderSystem y otros gestores de recursos.



Clase RenderSystem

La clase RenderSystem es la clase abstracta que define la interfaz de la API 3D que tenemos por debajo; es a decir Direct3D o OpenGl. Es la responsable de enviar las operaciones de renderizado a la API y configurar todas las opciones de renderización.

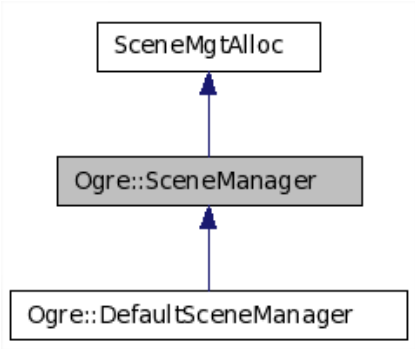
Una aplicación estándar no necesita manipular esta clase directamente, ya que desde las clases SceneManager y Material tendremos todo lo necesario para renderizar objetos y configurar sus propiedades.



Clase SceneManager

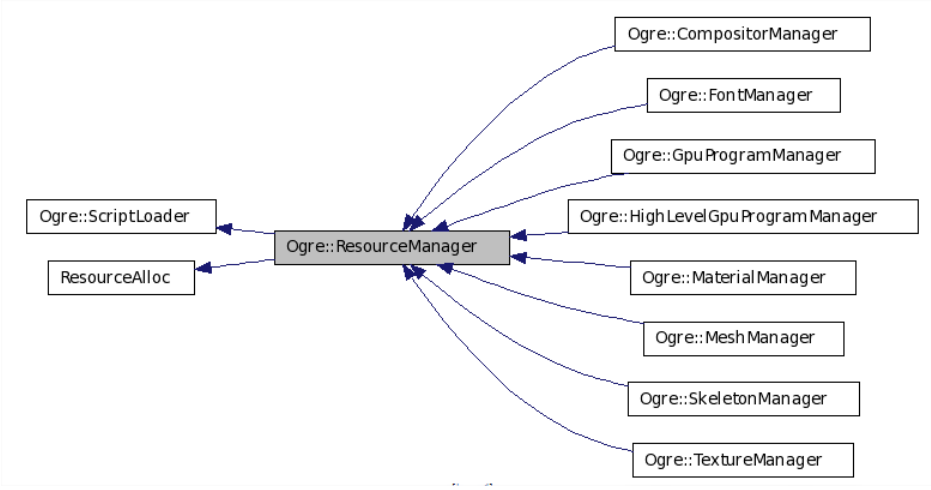
La Clase SceneManager es una clase importante dentro de Ogre3d. Es la encargada del contenido de la escena que será renderizada por el motor gráfico. Es la responsable de organizar el contenido de la escena, cámaras, objetos movibles (entidades), luces y materiales (propiedades de las superficies dela escena). También se encarga de gestionar la geometría del entorno.

El SceneManager es necesario cuando se quiere añadir una cámara para la escena. También cuando se quiere obtener o eliminar una luz de la escena. No es necesario mantener en la aplicación lista de objetos, ya que el SceneManager mantiene un conjunto de nombres para todos los objetos de la escena. De esta forma se puede obtener un objeto de la escena a través de su nombre, gracias a los métodos getCamera, getLigth, getEntity,…



Clase ResourceManager

La clase ResourceManager es únicamente una clase base para un gran número de otras clases que son utilizadas para gestionar los recursos. En este contexto, los recursos son conjuntos de datos que deben ser cargados desde algún lugar para proporcionar a Ogre3d los datos que necesita. Algunos ejemplos pueden ser texturas, mallas geométricas y mapas. Existe una subclase del Resource-Manager que gestiona cada tipo de recurso, p.e. el TextureManager carga las texturas o el Mesh-Manager que carga las mallas geométricas de los objetos.



Los ResourceManagers se encargan de controlar si los recursos están cargados y siendo compartidos a través del motor OGRE. Además también gestionan los requisitos de memoria de los recursos.

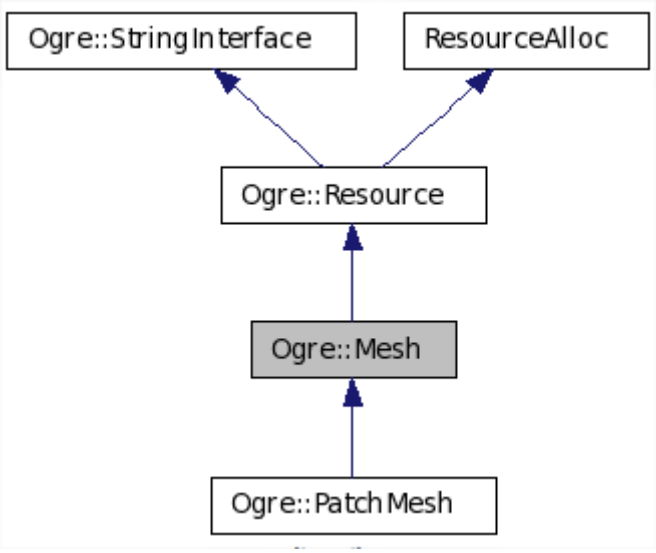
La mayor parte del tiempo no se interactuará directamente con los ResourceManagers. Los Resource-Managers serán llamados por otras partes del sistema OGRE cuando sea necesario, por ejemplo cuando se realiza una petición de una textura para añadirla a un Material. En este caso, el TextureManager será llamado automáticamente. Si es necesario, es posible llamar directamente al ResourceManager.

Clase Mesh

La clase Mesh representa un modelo discreto, un conjunto de geometría. La clase Mesh suelen utilizarse para representar objetos móviles y de tamaño medio o pequeño respecto la escala del entorno.

La clase Mesh es un tipo de recurso, y son gestionados por el MeshManager. Son cargados en el sistema utilizando el formato (.mesh). Los ficheros Mesh son típicamente creados exportándolos desde alguna herramienta de modelado. También es posible crear objetos Mesh manualmente. De esta forma es posible definir la geometría por uno mismo.

Los objetos Mesh son la base de los objetos móviles individuales del entorno, que son denominados entidades. Debe tenerse en cuenta que los objetos también pueden animarse a través de animación por esqueleto o morphing.



Clase Entity

La clase Entity es una instancia de la clase MovableObject. Esta entidad representa cualquier objeto que se pueda poner en la escena p.e. una persona, un coche, un árbol, etc.

Las entidades se basen en mallas discretas, p.e. colecciones de geometría representadas por el objeto Mesh, en la figura 7 podemos apreciar un objeto Mesh.

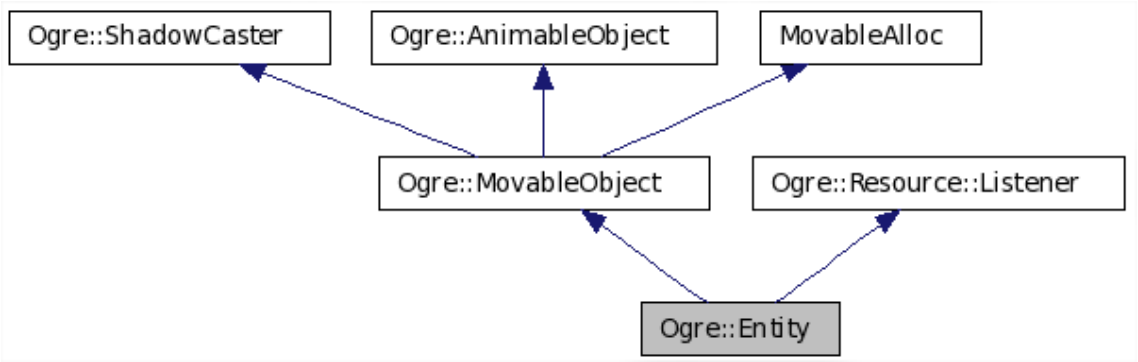
Figura 7 Ejemplo de un objeto Mesh

Para crear una entidad se llama al SceneManager, dándole un nombre y especificando el nombre del objeto Mesh. El SceneManager se encargará de comprobar que la Mesh ha sido cargada llamando al MeshManager. Únicamente se mantiene una copia de una Mesh en memoria.

Los objetos Entity no son considerados parte de la escena hasta que no son enlazados a un SceneNode. Enlenzado las entidades a los nodos (SceneNode), se puede crear complejas relaciones jerárquicas entre posiciones y orientaciones de las entidades.

Cuando una Mesh es cargada, automáticamente proporciona un número de materiales definidos. Es posible tener más de un material asignado a la Mesh. Cualquier entidad creada a partir de una Mesh utilizará los materiales que tenga asignados por defecto, pero se podrá cambiar a nivel de entidad y así generar diferentes entidades basadas en una misma malla, pero con diferente textura.

Los objetos Mesh están compuestos por objetos SubMesh, que representan partes de la Mesh utilizando un Material. Si una Mesh utiliza sólo un Material, entonces utilizara únicamente una SubMesh.

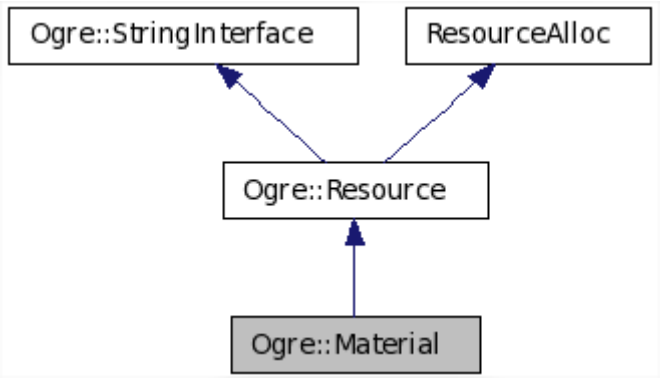


Clase Material

La clase Material controla cómo son renderizados los objetos de la escena. Especifican las propiedades básicas de las superficies de los objetos, textura, color, brillo, sombras, cuántas capas de textura utiliza, qué imágenes contienen y cómo son combinadas, que efectos especiales son aplicados, etc.

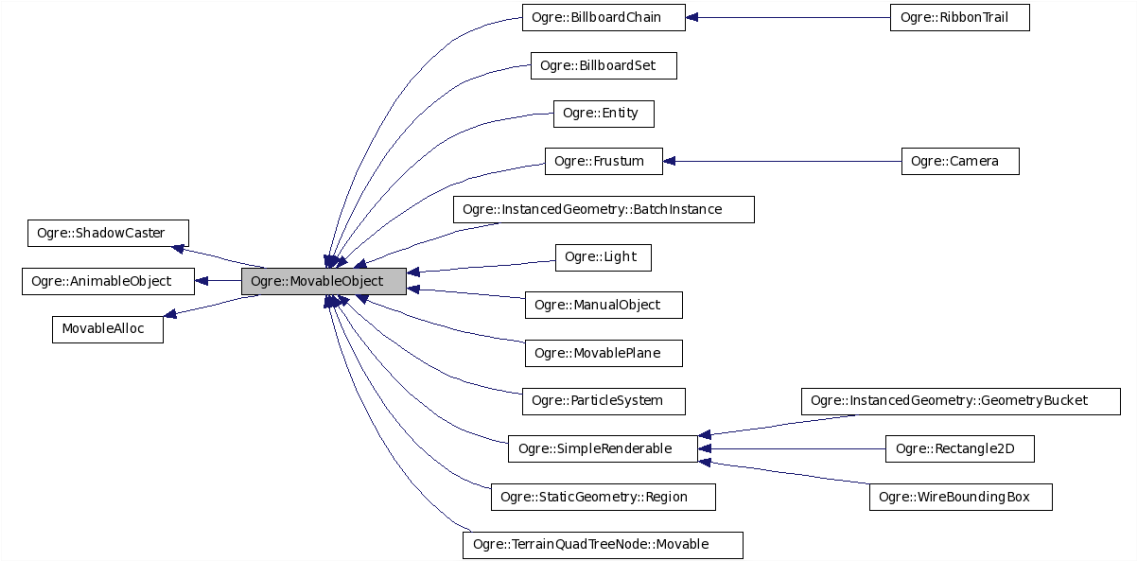
Los materiales pueden ser configurados manualmente mediante el código de la aplicación llamando a SceneManager::createMaterial y modificando sus propiedades, o especificándolas mediante un script.

Toda la apariencia del objeto es controlada por la clase Material sin contar la forma que se otorga con la clase Mesh.



Clase MovableObjet

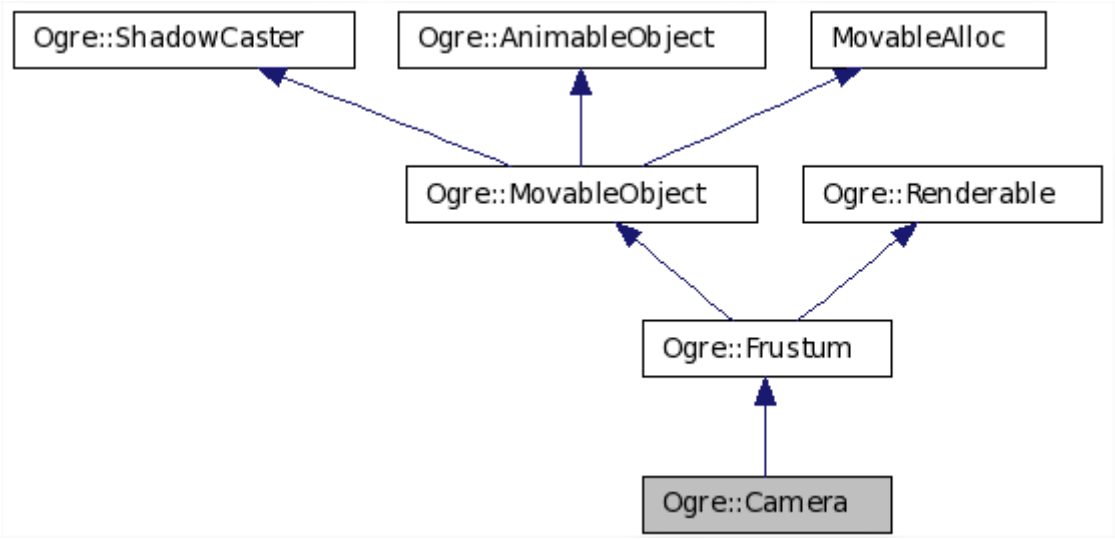
La Clase MovableObject es una clase abstracta para definir un objeto móvil dentro de una escena. Las instancias de esta clase se vincularan a la clase SceneNode para poder definir la posición y sus propiedades p.e. si el objeto es visible.



Clase Camera

La Clase camera representa los objetos que se visualizan en la escena. Ogre3d se encarga de renderizar la escena desde el punto de vista la clase Camera (que viene determinado por una posición y un vector que determina la dirección de la cámara) hacia un buffer[5] que suele ser una ventana o textura.

La clase Camera es una subclase de MovableObject, por lo tanto se puede asignar a un SceneNode, con esto podemos determinar la posición de la cámara a partir del nodo que tiene asignado, al ser un MovableObject dicha cámara se podrá mover con libertad por la escena.



[5] Buffer es una ubicación de la memoria reservada para el almacenamiento temporal de información.

7.3.2 CEGUI

CEGUI (*Crazy Eddie's Gui System*) es un librería libre que proporciona ventanas y widgets para las API graficas o motores gráficos que no tienen disponible esa funcionalidad de forma nativa. La librería esta orientada a objetos y esta programada en C++.

CEGUI se utiliza como interfaz grafica en Ogre3D, pero admite también Microsoft DirectX 8.1 y 9, OpenGl, y el motor Irrlicht[6] de forma nativa.

7.3.3 Image Debugger

El **Image Debugger** es una herramienta para los programadores que permite hacer una depuración de aplicaciones programadas en Windows que utilizan imágenes. Por ejemplo aplicaciones 3d como pueden ser videojuegos. La herramienta sólo está actualmente programada para Windows. Esta herramienta permite depurar una parte de la memoria convirtiéndola en una imagen de la que podremos obtener información.

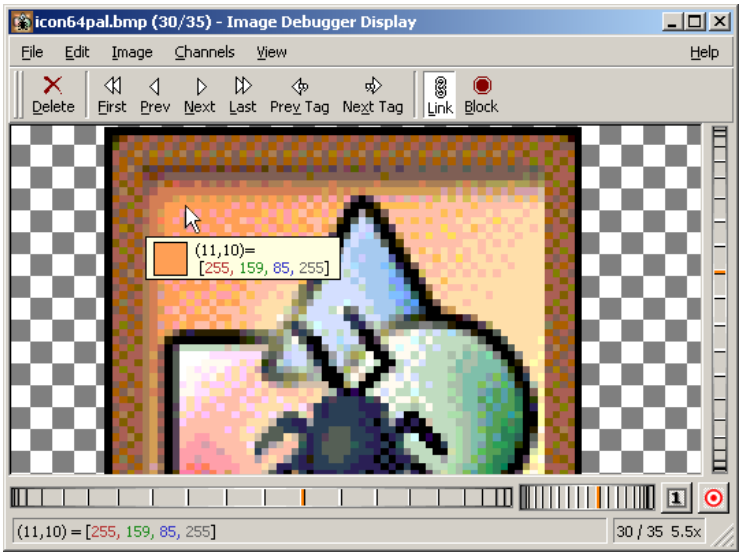


Figura 8 ImageDebugger

[6] Irrlicht es un motor 3D gratuito y de código abierto, escrito en C++, el cual puede ser usado tanto en C++ como con lenguajes .Net.

8 Análisis y diseño del sistema

En este capítulo se ocupara de estudiar de manera detallada las necesidades del sistema y diseño de la herramienta. Se utilizara UML[7] para explicar parte de los siguientes apartados, ya que es un estándar en el campo de las ingenierías informáticas.

8.1 Descripción general

El objetivo de este proyecto es crear una herramienta de código libre que permita crear un policubo, ya que no existe ninguna capaz de crear un policubo manualmente.

8.2 Diseño del funcionamiento

El editor de policubo nos permite crear un policubo a partir de un modelo base. Una vez cargado el modelo. Se pondrá el primer cubo simplemente apuntando con el ratón en una zona del modelo. La herramienta insertará el primer cubo en el lugar deseado. Ver Figura 9. Una vez creado el primer cubo podremos escoger el tamaño y la orientación del mismo. A partir de este momento se podrán ir colocando más cubos o borrando si fuera necesario con solo seleccionar una cara de un cubo.

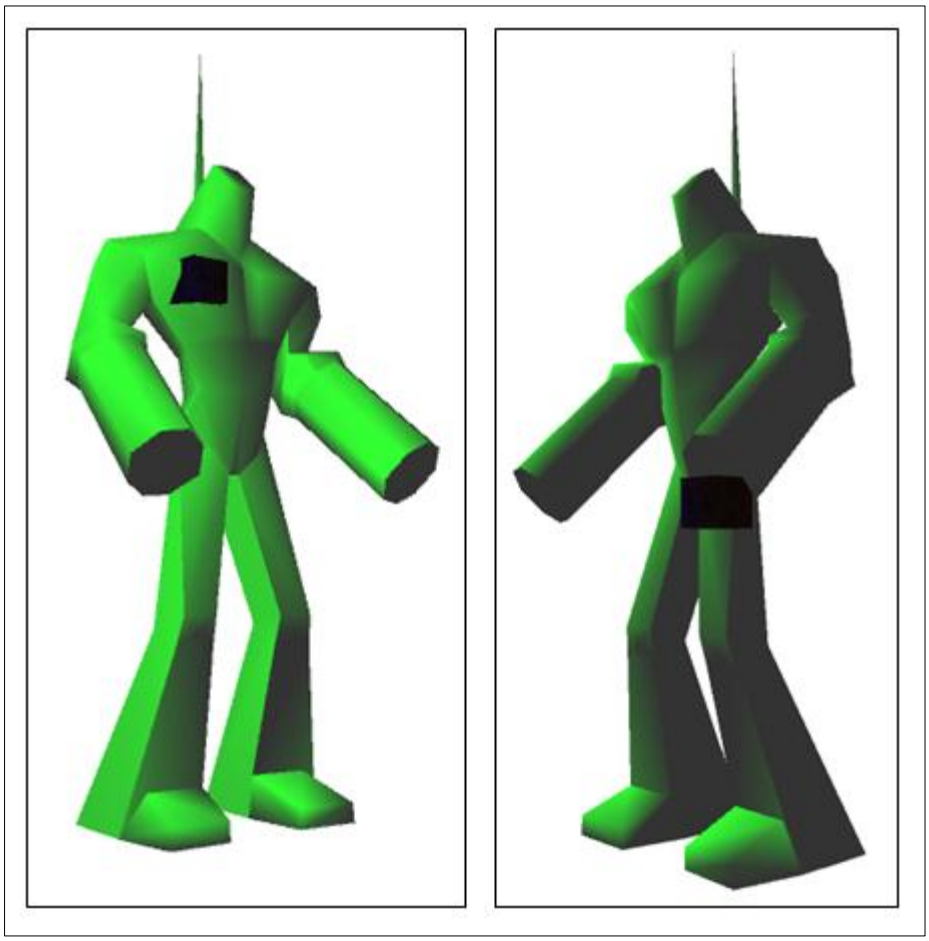


Figura 9 Ejemplo primer cubo

[7] UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

8.3 Identificación de actores

Un actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con esto se especifica que un actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

El actor será el usuario encargado de modelar el policubo. No habrá más de un actor en esta herramienta ya que sólo un usuario puede interactuar con ella.

8.4 Casos de Uso

Un caso de uso es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.

8.4.1 Diagrama de caso de uso general

Al iniciarse la herramienta el actor tiene las siguientes opciones: Cargar, guardar, salir y poner el primer cubo sobre el modelo base. Ver Figura 10.

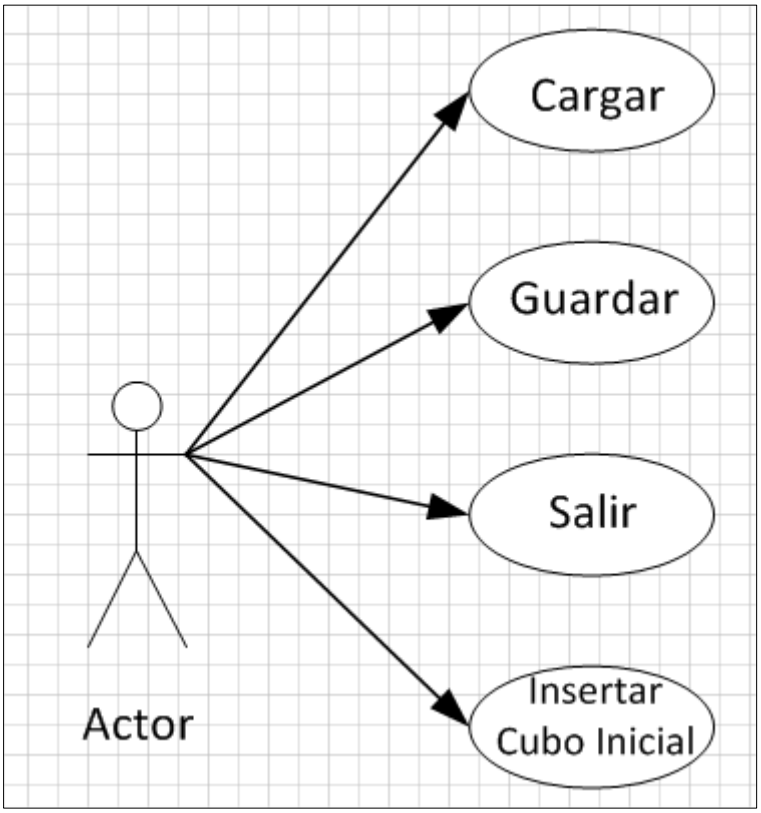


Figura 10 Diagrama de caso de uso general.

Ficha	Cargar Policubo
Funcionalidad	A partir de este caso de uso se podrá cargar un modelo de policubo previamente guardado en el ordenador.
Actor	Usuario.
Pre-Condición	Haber guardado anteriormente con la herramienta un policubo.
Flujo Principal	1. Insertar Cubo. 2. Borrar Cubo. 3. Cambiar Resolución. 4. Salir de la herramienta.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Se ha cargado el modelo de policubo a partir de un fichero guardado en el ordenador.

Ficha	Guardar Policubo
Funcionalidad	A partir de este caso de uso se podrá guardar el modelo de policubo al ordenador para posteriormente recuperarlo.
Actor	Usuario.
Pre-Condición	Ninguna.
Flujo Principal	1. Insertar Cubo 2. Borrar Cubo. 3. Cambiar Resolución. 4. Salir de la herramienta.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Se ha guardado el modelo de policubo en un fichero elegido por el usuario.

Ficha	Salir de la herramienta
Funcionalidad	A partir de este caso de uso se podrá salir de la herramienta.
Actor	Usuario.
Pre-Condición	Ninguna.
Flujo Principal	Ninguno.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Ninguna.

Ficha	Insertar Cubo Inicial
Funcionalidad	A partir de este caso de uso se podrá generar un modelo de policubo.
Actor	Usuario.
Pre-Condición	Al presionar el botón izquierdo del ratón, tiene que estar apuntando a una zona del modelo base. En caso contrarió no se crea el cubo.
Flujo Principal	1. Insertar Cubo 2. Borrar Cubo. 3. Cambiar Resolución. 4. Guardar Policubo. 5. Cargar Policubo. 6. Salir de la herramienta.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Se ha creado el cubo inicial donde apunta el puntero del ratón.

8.4.2 Diagrama de caso de uso parámetros del cubo inicial.

En el diagrama de la Figura 11 muestra las diferentes opciones que tendrá el usuario después de insertar el cubo inicial.

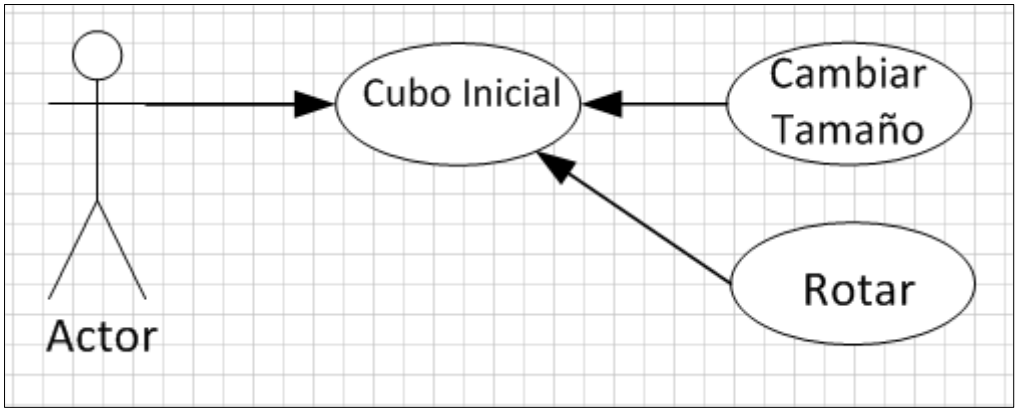
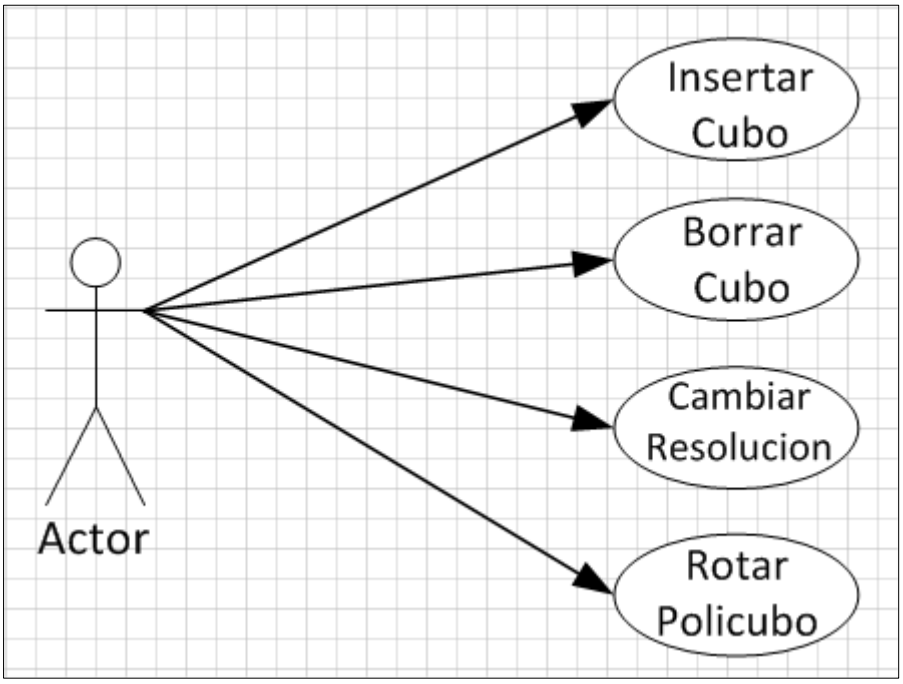


Figura 11 Parámetros del cubo inicial

Ficha	Parámetros cubo inicial.
Funcionalidad	Modificar los parámetros del cubo inicial.
Actor	Usuario.
Pre-Condición	Haber insertado el cubo inicial.
Flujo Principal	1. Cambiar tamaño. 2. Rotar cubo.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	El cubo queda modificado por el usuario. Una vez elegido el tamaño, no se podrá modificar. La rotación del cubo se podrá modificar en cualquier momento afectando a todo el policubo.

8.4.3 Diagrama de caso creación policubo

Ficha	Creación policubo.
Funcionalidad	Crear un policubo a partir de un modelo base.
Actor	Usuario.
Pre-Condición	Haber insertado el cubo inicial.
Flujo Principal	1. Insertar Cubo. 2. Borrar Cubo. 3. Cambiar Resolución. 4. Rotar policubo.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Generar un policubo.



8.4.4 Diagrama de secuencia de Inserción de un cubo.

En la siguiente Figura 12, se puede ver un ejemplo del diagrama secuencia de la inserción de un cubo en el que intersecta el rayo en la primera cara.

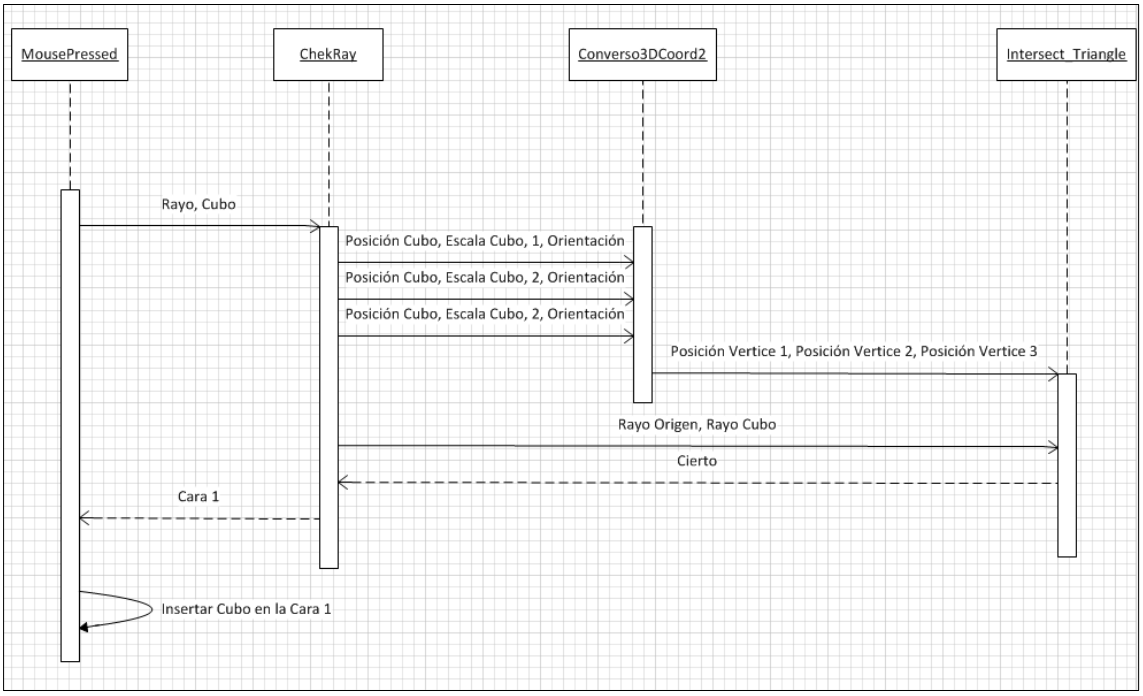


Figura 12 Diagrama de secuencia de insertar un cubo.

8.4.5 Diagrama de secuencia de cambiar la resolución del policubo.

En la siguiente Figura 13, se puede ver un ejemplo del diagrama secuencia del cambio resolución del policubo creado por el usuario.

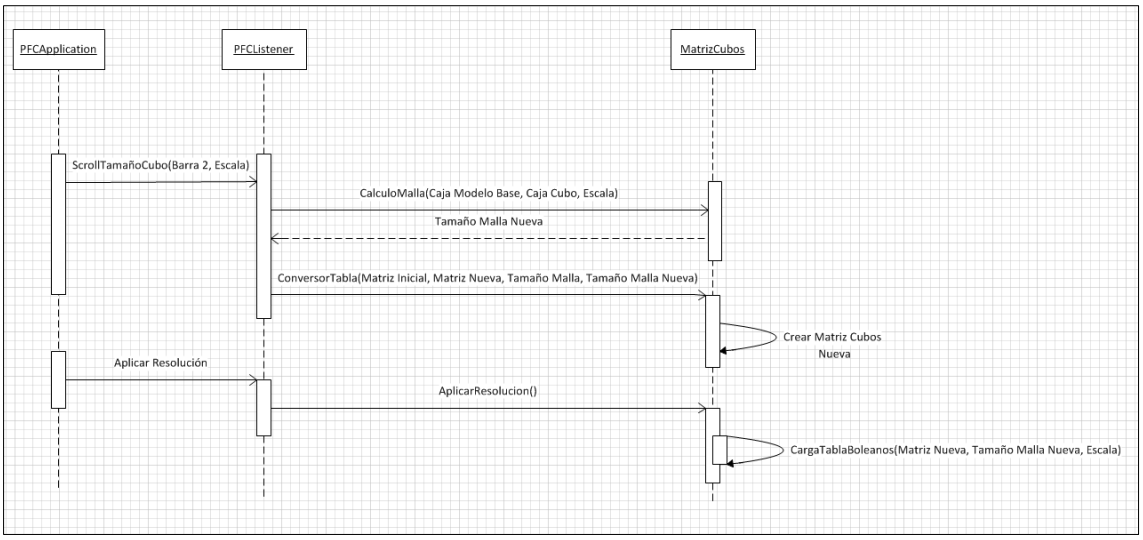


Figura 13 Diagrama de secuencia de cambiar la resolución del policubo.

8.5 Diseño del menú.

En la herramienta se ha creado un menú donde se puede interactuar con las distintas opciones:

1. Opciones Cubo.
    a. Tamaño.
    b. Transparencia.
    c. Resolución.
2. Modelo Base
    a. Resolución.
3. Cargar.
4. Guardar.
5. Salir.

En la Figura 14se puede observar como la apariencia del menú.

Las opciones tamaño, transparencia y resolución son barras deslizantes.

Figura 14 Menú del Editor

8.6 Diseño del movimiento de la cámara.

El usuario puede moverse libremente por el escenario 3D, moviendo la cámara. Esto se consigue con la combinación del teclado más el ratón.

- Con el teclado de puede controlar el desplazamiento de la cámara.

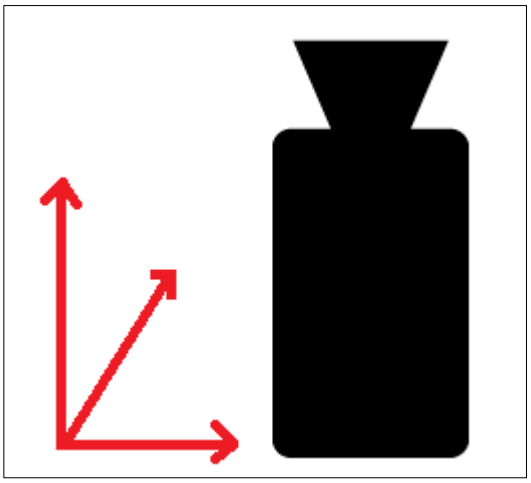


Figura 15 Desplazamiento de la cámara.

- Con el ratón se puede controlar la rotación de la cámara.

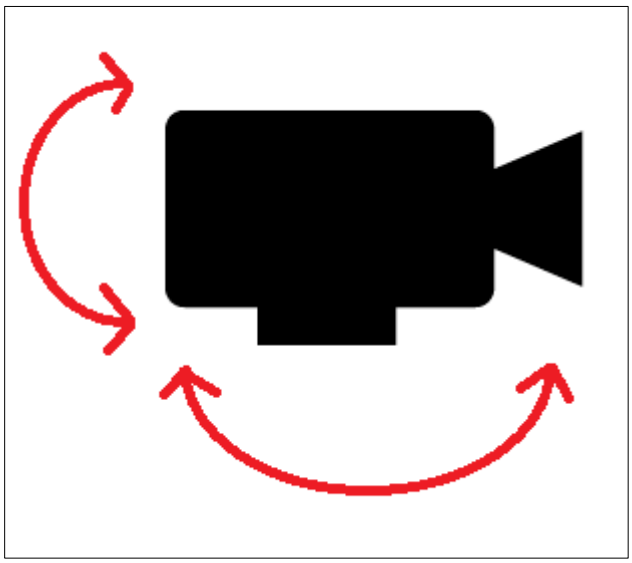


Figura 16 Rotación de la cámara

Ficha	Mover cámara.
Funcionalidad	Mover la cámara por el entorno 3D.
Actor	Usuario.
Pre-Condición	Ninguna.
Flujo Principal	Se mueve la cámara haciendo uso del teclado y el ratón.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Generar un policubo.

8.7 Diseño inserción del primer cubo.

El usuario para empezar a crear el policubo necesita insertar el primer cubo. La inserción del primer cubo se hace sobre el modelo base con ayuda del ratón.

En el siguiente ejemplo simplificado que se puede ver en la Figura 17 Inserción del primer cubo, se muestra el procedimiento para colocar el primer cubo: El usuario selecciona con el puntero, más el botón izquierdo del ratón, una parte del modelo base: sabiendo el tamaño del cubo se calculan cuántos cubos caben en el modelo base. Seguidamente el cubo se coloca en a la posición más cercana de donde ha señalado el puntero. El usuario al colocar el primer cubo, no percibe ningún movimiento del cubo entre su posición elegida, ya que el cambio a la posición real se crea al insertar el segundo cubo. En la Figura 18. Se muestra un ejemplo real.

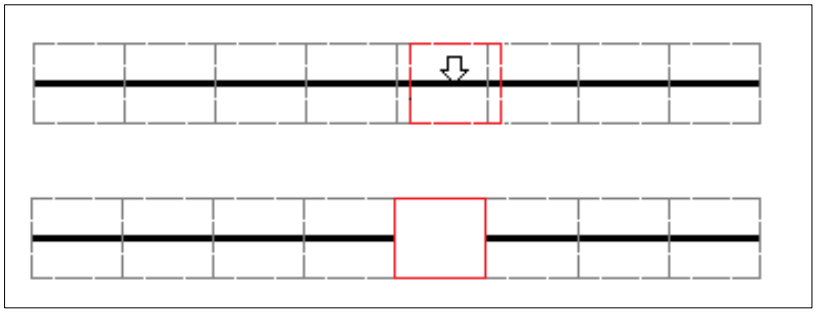


Figura 17 Inserción del primer cubo

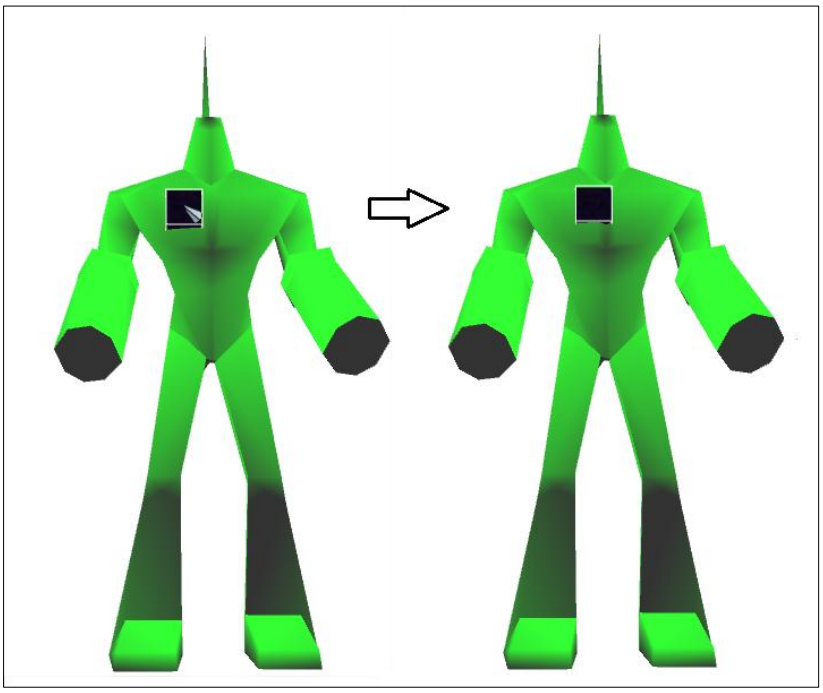


Figura 18 Ejemplo de el cambio de posición del primer cubo al insertar el segundo cubo.

Ficha	Inserción del primer cubo.
Funcionalidad	Insertar el primer cubo.
Actor	Usuario.
Pre-Condición	Seleccionar una zona del modelo base.
Flujo Principal	1. Cambiar tamaño. 2. Cambiar rotación.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno.
Post-Condición	Se ha insertado el cubo inicial.

8.8 Diseño de la inserción de un cubo.

Una vez puesto el primer cubo, el usuario puede comenzar a crear el policubo. Para poner los siguientes cubos se tiene que seleccionar una cara del cubo con el puntero más el botón izquierdo del ratón, la herramienta colocará el nuevo cubo en la cara seleccionada. En la Figura 19, se muestra un ejemplo simplificado.

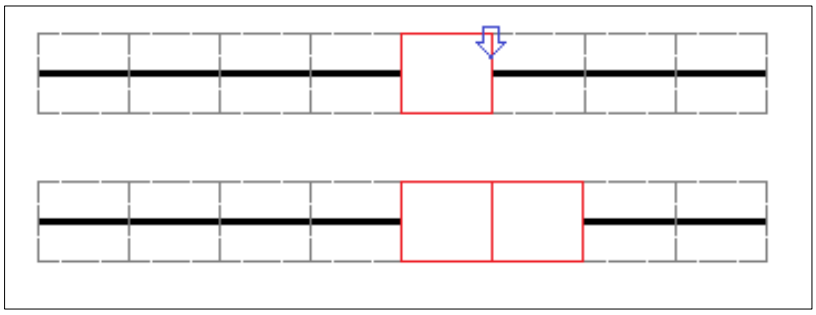


Figura 19 Inserción de un cubo simplificado.

En la siguiente Figura 20, se puede ver un ejemplo real de cómo se inserta un cubo.

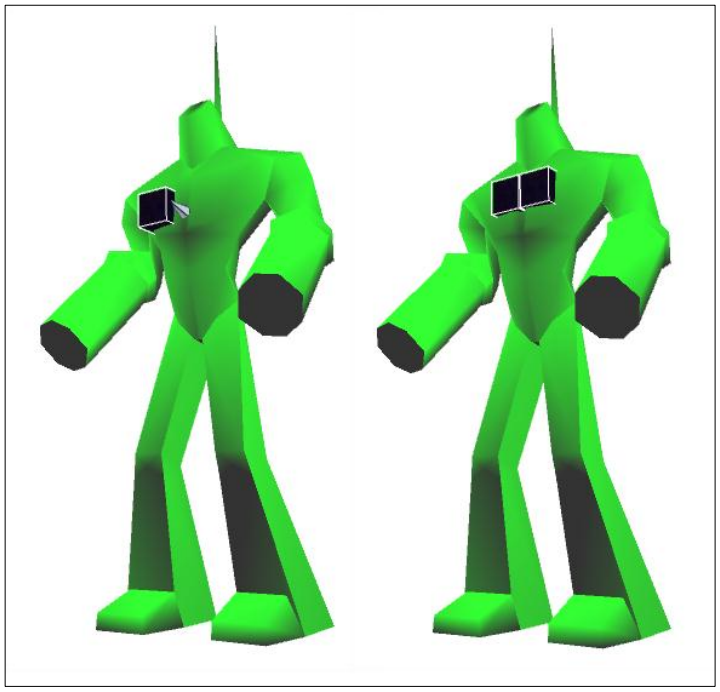


Figura 20 Ejemplo de una inserción de un cubo.

Ficha	Inserción cubo.
Funcionalidad	Insertar cubo.
Actor	Usuario.
Pre-Condición	Seleccionar una cara de un cubo.
Flujo Principal	Insertar nuevo cubo
Sub-flujo	Ninguno.
Flujo alternativo	Si se llega al límite del modelo base no se podrá poner más cubos.
Post-Condición	Se ha insertado el cubo nuevo en la cara del cubo seleccionado.

8.9 Diseño de borrar un cubo.

En este apartado se muestra como la herramienta puede borrar un cubo del policubo. La forma de hacerlo es seleccionando una cara del cubo con el puntero más el botón derecho del ratón y la tecla espacio del teclado al mismo tiempo. Con esta acción se borra el cubo.

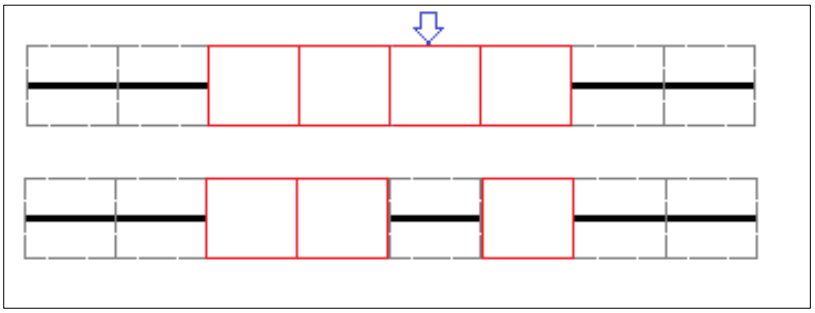


Figura 21 Borrar un cubo simplificado.

En la Figura 21, se puede ver un ejemplo real de cómo se borra un cubo.

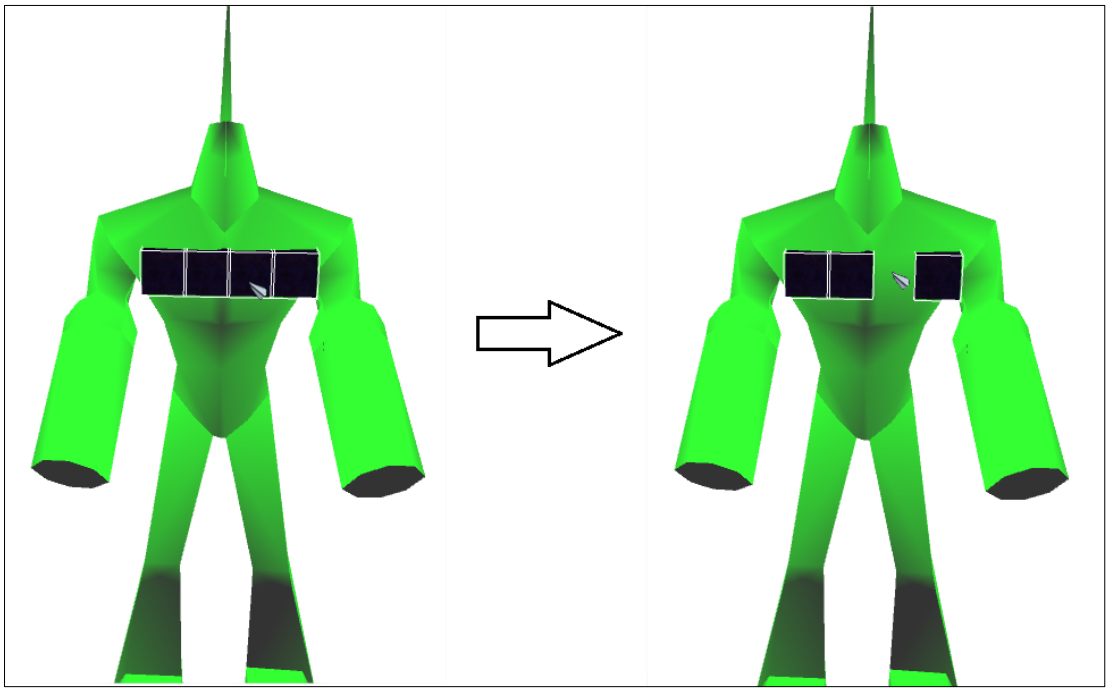


Figura 22 Ejemplo del borrado de un cubo.

Ficha	Borrar cubo.
Funcionalidad	Borrar cubo.
Actor	Usuario.
Pre-Condición	El usuario selecciona una cara de un cubo que se quiere borrar .
Flujo Principal	La herramienta borra el cubo seleccionado por el usuario.
Sub-flujo	Ninguno.
Flujo alternativo	Ninguno
Post-Condición	Se ha borrado el cubo seleccionado.

8.10 Diseño para cambiar la resolución de un policubo.

Una vez empezado a crear el policubo, se puede cambiar la resolución del cubo a mayor o menor tamaño según el deseo del usuario. La herramienta, mediante un método de programación, creará un nuevo policubo con la resolución deseada. Ver la Figura 23. Para hacer el cambio de resolución, el usuario lo tiene que hacer con la barra deslizante del menú.

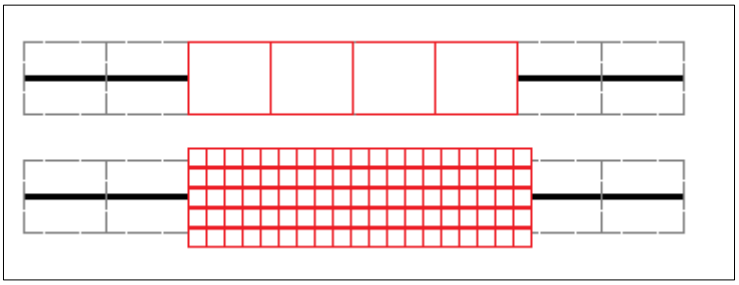


Figura 23 Cambio de resolución del policubo.

En la siguiente Figura 24, se muestra un ejemplo real del cambio de resolución del policubo.

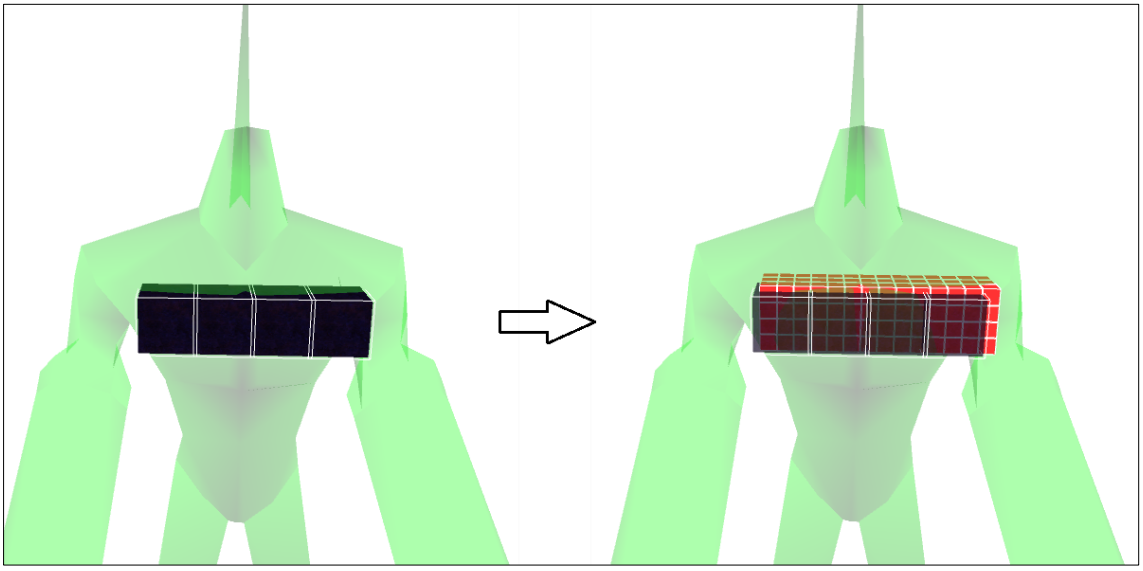


Figura 24 Ejemplo de cambio de resolución del policubo.

8.11 Clases y métodos

Se han implementado las siguientes clases: PFCListener y PFCApplication, MatrizCubos. También se han utilizado clases ya existentes de las que se destaran las más importantes para la compresión del proyecto, y en esta apartado se hará un breve referencia a los métodos utilizados: Ogre::Camera, Ogre::SceneManager, Ogre::Entity, Ogre::SceneNode. En la siguiente Figura 25 se puede ver el diagrama de clases de la herramienta realizada.

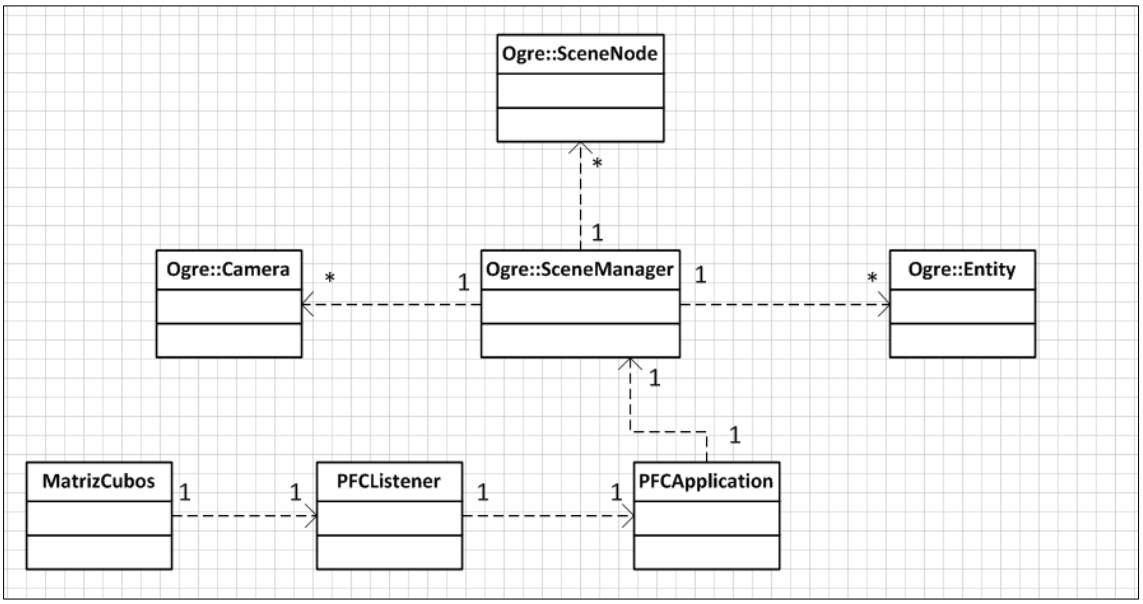


Figura 25 Diagrama de Clases

8.11.1 Clases utilizadas ya existentes

En este apartado se comentaran brevemente los métodos utilizados de las siguientes clases: Ogre::Camera, Ogre::SceneManager, Ogre::Entity, Ogre::SceneNode.

8.11.1.1 Clase Ogre::Camera

Esta clase de Ogre es la encargara de generar la cámara con la que se ve la escena resultante. Hereda la clase MovableObjet y es generada por la clase SceneManager. En el apartado 7.3.1.1 se habla del funcionamiento de la clase. En la siguiente Figura 26 podemos ver los métodos de la clase que han sido necesarios para el proyecto.

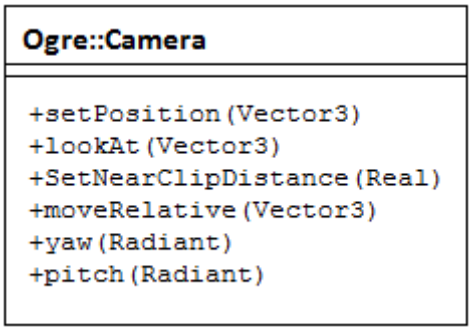


Figura 26 Clase Ogre::Camera

- setPosition: Este método se encarga de posicionar la cámara en la escena mediante una coordenada tridimensional.
- lookAt: Este método, a través de un vector de coordenadas tridimensionales, da a la cámara el punto de enfoque.
- setNearClipDistance: Este método sirve para determinar la distancia en la que la cámara es funcional mientras se utiliza.
- moveRelative: Este método, mueve la cámara por el escenario con un vector.
- Yaw: Este método, cambia el cabeceo de la cámara.
- Pitch: Este método, cambia el balanceo de la cámara.

8.11.1.2 Clase Ogre:Entity

Con esta clase se instancian objetos movibles basados en una mesh. Los objetos entity no se crean directamente, una entity se crea con la clase Ogre::SceneManager concretamente con el método createEntity (string entityname, string meshname). En la Figura 27 se puede ver el método más importante utilizado de la clase.

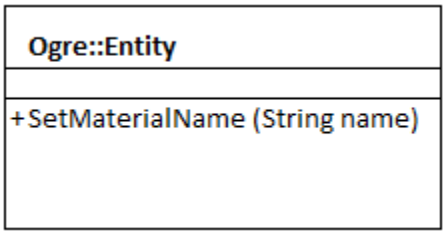


Figura 27 Clase Ogre::Entity

- SetMaterialName: Este método permite cargar un material a la entidad definido por el atributo name.

8.11.1.3 Clase Ogre::SceneNode

Es la clase utilizada para organizar los objetos dentro de la escena. Cada nodo puede almacenar objetos (luces, cámaras, "entity"…), la forma de almacenar los objetos es con una estructura de árbol, gracias a esta estructura podremos tener un padre y los demás hijos podrán heredar las propiedades del padre. En la Figura 28 se puede ver el método más importante utilizado de la clase.

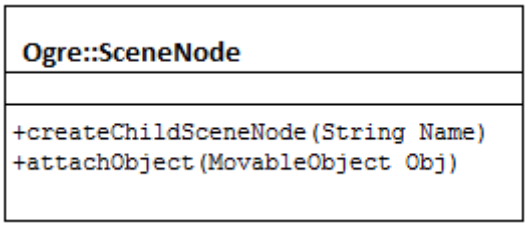


Figura 28 Clase Ogre::SceneNode

- creteChildSceneNode: Este método crea un hijo SceneNode definido por el atributo name
- attachObjet: Este método agrega una instancia del objeto al SceneNode.

8.11.1.4 Clase Ogre::SceneManager

La clase SceneManager nos permite crear luces, cámaras, entidades, nodos, etc. y también destruirlos. Como se explicó anteriormente en el apartado 7.3.1.1, es una clase importante en el entorno Ogre, así que se hace un uso intensivo de sus métodos para el desarrollo de la herramienta. En la Figura 29 se puede ver los métodos más importantes.

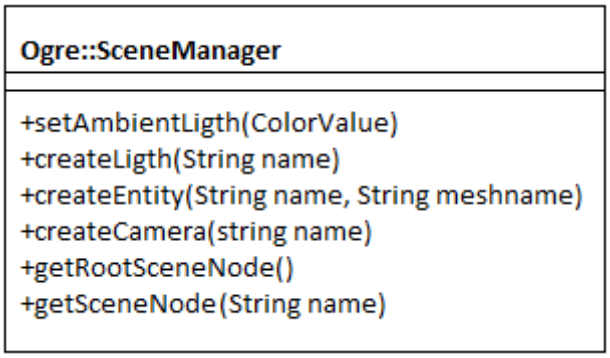


Figura 29 Clase Ogre::SceneManager

- setAmbientLigth: Este método establece el nivel de luz ambiente que se utiliza en la escena.
- createLigth: Este método crea una luz en la escena definido por el atributo name.
- createEntity: Este método crea una "entity" definido por un nombre y una mesh.

- createCamera: Este método crea una Camara definido por el atributo name.
- getRootSceneNode: Este método obtiene el SceneNode raíz de la escena.
- getSceneNode: Este método recupera un SceneNode definido por el atributo name.

8.11.2 Clases implementadas en el proyecto.

En este apartado se describe el funcionamiento de las clases implementadas para el proyecto.

8.11.2.1 Clase MatrizCubo

Esta clase contiene los métodos necesarios para crear el modelo de policubos y poder cargar/guardar su estructura. En la Figura 30 se pueden ver los atributos de la clase y los métodos.

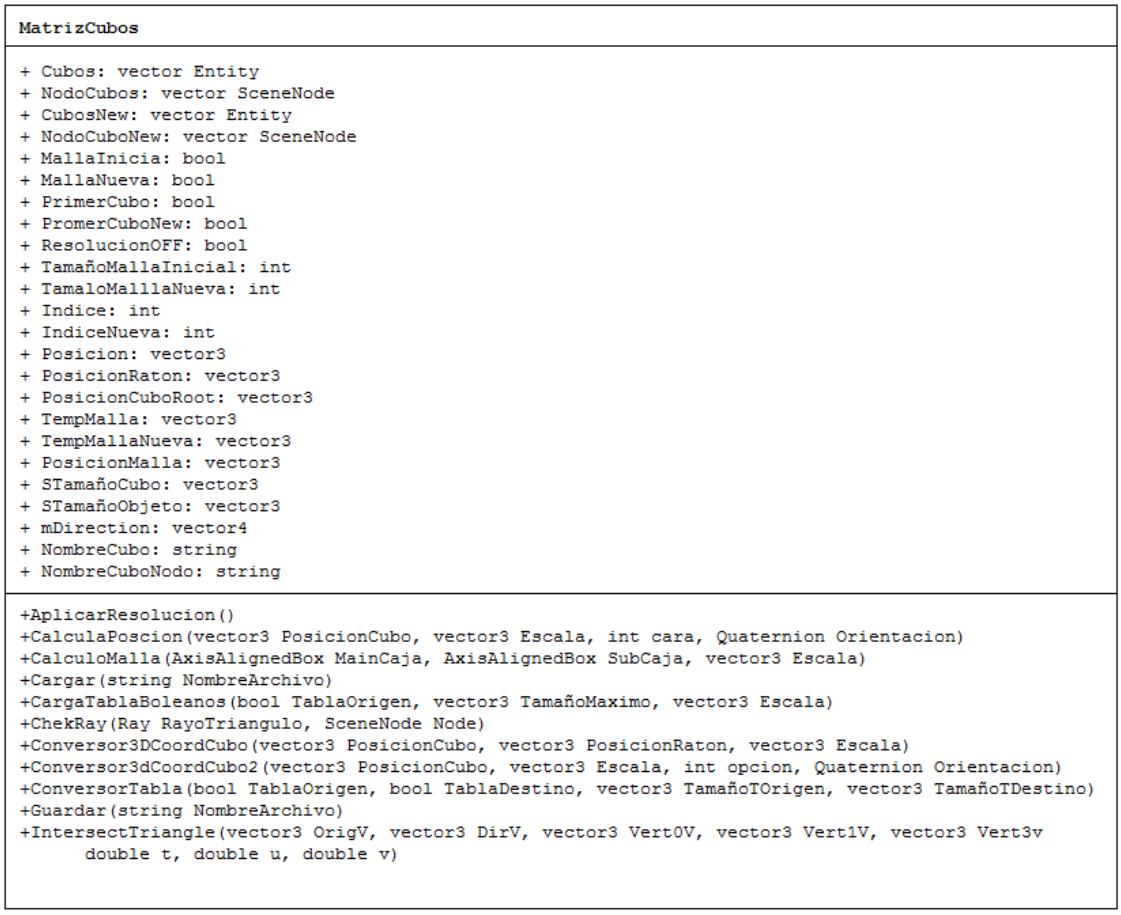


Figura 30 Clase MatrizCubo

Atributos

+ Cubos: Vector de Entity donde se guardan las entidades cubo del policubo.

+ NodoCubos: Vector de SceneNode donde se guardan las SceneNode del policubo.

+ CubosNew: Vector de Entity donde se guardan las entidades cubo de la nueva resolución del policubo.

+ NodoCubosNew: Vector de SceneNode donde se guardan las SceneNode de la nueva resolución del policubo.

+MallaInicial: Es un puntero a una matriz de booleanos donde se guarda la malla virtual de los cubos puestos en el policubo.

+MallaNueva: Es un puntero a una matriz de booleanos donde se guarda la malla virtual de cubos que hace referencia a la nueva resolución del policubo.

+PrimerCubo: Booleano que guarda si se ha puesto el primer cubo.

+PrimerCuboNew: Booleano que guarda si se ha puesto el primer cubo en la nueva resolución del policubo.

+ResolucionOFF: Booleano que controla si esta activado un cambio de resolución, en ese caso no se puede editar el policubo.

+TamañoMaximoInicial: Entero que guarda el tamaño máximo de cubos que puede tener el malla.

+TamañoMaximoNueva: Entero que guarda el tamaño máximo de cubos que puede tener la malla cuando se le aplica una nueva resolución al policubo.

+Indice: Entero que guarda el número de cubos puesto en el policubo.

+IndiceNuevo: Entero que guarda el numero de cubos puesto en el policubo cuando se quiere obtener una nueva resolución.

+Posición: Vector que guarda la posición del cubo.

+PosiciónRaton: Vector que guarda la posición del ratón.

+PosiciónCuboRoot: Vector que guarda la posición del cubo root.

+TempMalla: Vector que guarda el número máximo de cubos que se puede poner en la malla.

+TempMallaNueva: Vector que guarda el numero máximo de cubos que se puede poner en la malla cuando se quiere obtener una nueva resolución.

+PosiciónMalla: Vector que guarda la posición del cubo en la malla.

+STamañoCubo: Vector que guarda el tamaño del cubo de la barra del menú.

+mDirection: Vector que guarda la dirección en que se mueve la cámara.

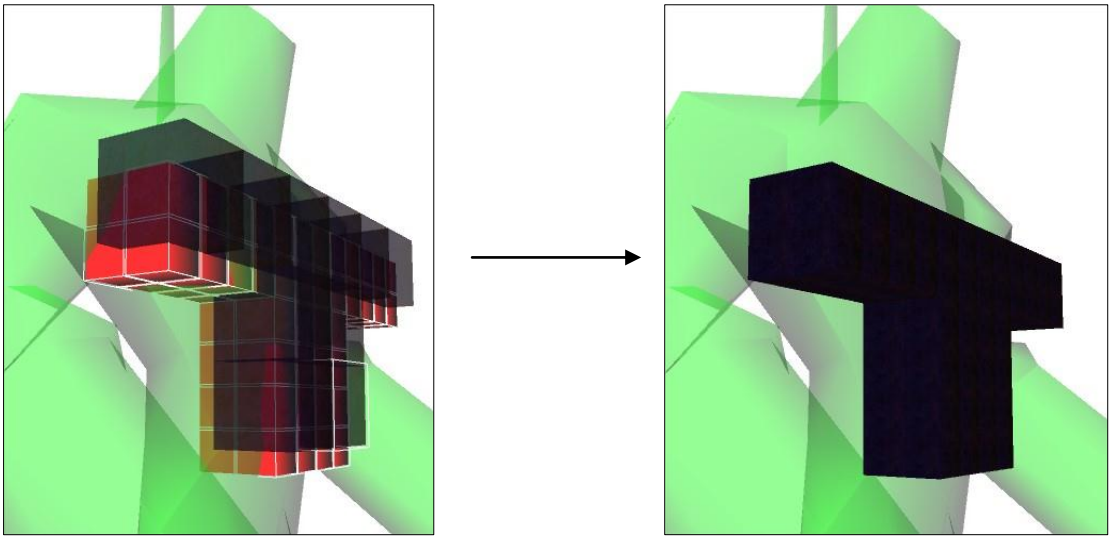
+NombreCubo: Es un puntero de caracteres que guarda el nombre del cubo.

+NombreCuboNodo: Es un puntero de caracteres que guarda el nombre del Nodo.

Métodos

+AplicarResolucion()

Este método es el encargado de aplicar la nueva resolución al policubo. En las figuras siguientes se muestra el resultado de este método de cómo se pasa a una resolución mayor obteniendo el doble de cubos en el nuevo policubo.



El pseudocódigo del método es:

```
ResolucionOff = True
Por (i = Índice + 1 hasta IndiceNueva -1) hacer
        Borrar SceneNode de i
        Borrar Entity de i
FinPor

IndiceNueva = 0

Por (i = 0 hasta Índice - 1) hacer
        Borrar SceneNode de i
        Borrar Entity de i
FinPor
Limpiar Cubos
Limpiar NodoCubos
Cubos.CambiarTamaño = CubosNew.Tamaño
```

```
Por (i = 0 hasta TamañoMallaInicial) hacer
        Si MallaNueva de i = cierto entonces
                MallaInicial de i = cierto
        Sino MallaInicial de i = falso
        FinSi
FInPor

STamañoObjeto = CuboRoot->setScale
TempMalla = TempMallaNueva
CargaTablaBooleanos (MallaInicial, TempMalla, STamañoObjeto)
Limpiar CubosNew
Limpiar NodoCubosNew
TamañoMallaNew = 0
```

Básicamente este método lo que hace internamente es borrar los dos policubos que hay en pantalla. En primer lugar borra el policubo con la nueva resolución y después borra el policubo editado por el usuario. El paso siguiente es copiar la Malla de booleanos y obtener el Tamaño del objeto para llamar al método CargaTablaBoolenas e insertar el nuevo policubo en pantalla.

+ CalculaPosición(vector3 PosicionCubo, Int Cara)

Este método se encarga de calcular la posición del cubo que se quiere insertar a partir de la cara seleccionada por el usuario.

+CalculoMalla(AxisAlignedBox MainCaja, AxisAlignedBox SubCaja, Escala)

Este método se encarga de calcular cuántos cubos caben en la BondingBox del modelo base.

El procedimiento es el siguiente, obtiene los valores máximos y mínimos de cada boundingbox y con la siguiente fórmula se obtiene el número de cubos.

$$Resultado = \left(\frac{Valor\ Máximo\ Main\ Caja - Valor\ Mínimo\ MainCaja}{(\text{Valor Máximo Subcaja} - \text{ Valor Mínimo Subcaja}) * \text{Escala}}\right)$$

+ Cargar (String NombreArchivo)

Este método se encarga de restaurar la aplicación a un estado anterior previamente guardado en un fichero .xml

El procedimiento es el siguiente:

Limpiar la escena e inicializar las variables.

Cargar la escena luces, cámara, etc.

Obtener la información guardada en el fichero .XML (Posición del CuboRoot, orientación del CuboRoot, Escala del CuboRoot, Posición del Modelo Base y la Malla).

Una vez obtenida la información del fichero se hace llamada al método CargaTablaBooleanos e inserta el policubo en pantalla.

+ CargarTablaBooleanos (bool TablaOrigen, vector3 TamañoMaximo, vector3 Escala)

Este método carga la malla de cubos a partir de una tabla de booleanos, el tamaño máximo de tabla y el factor escala del cubo. El pseudocódigo del método es:

```
PrimerCubo = Cierto
Por (xx = 1 hasta TamañoMaximo.x + 1) hacer
    Por (yy = 1 hasta TamañoMaximo.y + 1) hacer
        Por (zz = 1hasta TamañoMaximo.z + 1) hacer
        Si TablaOrigen de ((zz+(TamañoMaximo.z+1)*(yy-1)+
            ((TamañoMaximo.y+1)*(TamañoMaximo.z+1))*(xx-
            1))-1) hacer
            Si PrimerCubo es cierto hacer
                Insertar Cubo Root
            SiNo
                Insertar Cubo
            FinSi
            PrimerCubo = Falso
        FinSi
        FinPor
    FinPor
```

El procedimiento de este método se basa en un triple bucle, que hace un recorrido por la malla. En cara iteración calcula la posición real dentro de la tabla origen y

comprueba si el valor es cierto en ese caso quiere decir que existe un cubo. Si es el primer cubo encontrado se inserta el Cubo Root y después con las siguientes iteraciones se inserta los siguientes cubos.

ChekRay (Ray RayoTriangulo, SceneNode nodo)

Este método devuelve en la cara más próxima a la cámara que intersecta el rayo. En la siguiente Figura 31 podemos ver el funcionamiento gráficamente, donde la flecha azul es el rayo y los puntos rojos es donde intersecta el rayo con el cubo. En este ejemplo el rayo intersecta con el triangulo que tiene los vértices (2,6,8) y (5,7,8).

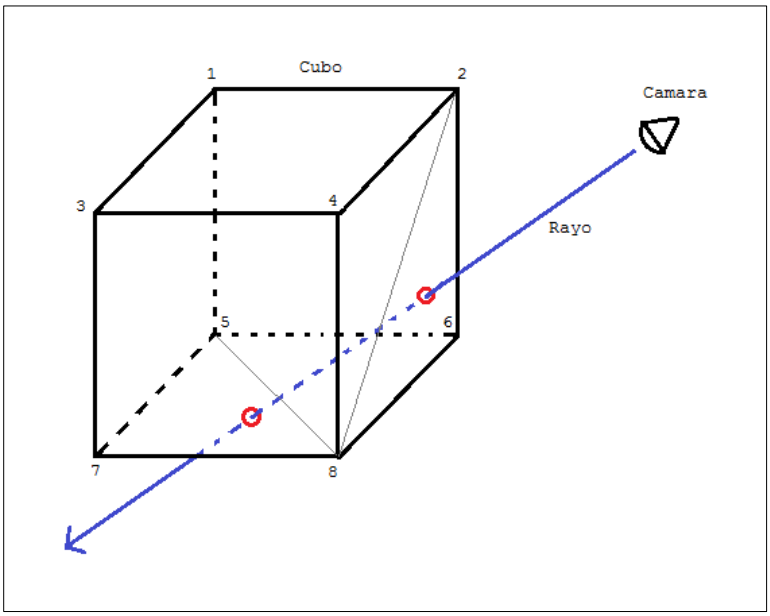


Figura 31 Ejemplo de la intersección del rayo en un cubo

El pseudocódigo del método es:

```
Double t, u, v, distancia
Int Cara
SumaTotal = Nodo->getPosition() * CuboRoot->Scale()
Si (intersect_triangle (RayoTriangulo.Origen, RayoTriangulo.Dirección,
Vertice1, Vertice2, Vertice3, t, u, v)) entonces
        Si Cara es igual a 0 entonces
                Cara = 1
                Distancia = t
        Sino Si (t < distancia) entonces
```

```
                Distancia = t
        FinSi
FinSi
Si (intersect_triangle (RayoTriangulo.Origen, RayoTriangulo.Dirección,
Vertice2, Vertice3, Vertice4, t, u, v)) entonces
        Si Cara es igual a 0 entonces
                Cara = 2
                Distancia = t
        Sino Si (t < distancia) entonces
                Cara =2
                Distancia = t
        FinSi
FinSi
```

Se hace la comprobación de todas las caras posibles del cubo, en el siguiente código se pude ver el último Si.

```
Si (intersect_triangle (RayoTriangulo.Origen, RayoTriangulo.Dirección,
Vertice2, Vertice5, Vertice6, t, u, v)) entonces
        Si Cara es igual a 0 entonces
                Cara = 12
                Distancia = t
        Sino Si (t < distancia) entonces
                Cara =12
                Distancia = t
        FinSi
FinSi

Devolver cara
```

El procedimiento de este método se base en obtener los vértices de cada cara del cubo y dividir esa cara en dos triángulos. Una vez hecho esto, gracias al método Intersect_Triangle, se conocerá si hay intersección.

+ Conversor3DCoordCubo2 (Vector3 PosicionCubo, Vector3 Escala, Int Opción, Quaternion Orientación)

Este método retorna la posición del cubo que se quiere insertar pasándole como referencia la posición del cubo y la cara que se ha seleccionado. También se le pasa la escala y la orientación del cubo root.

+ ConversorTabla (bool TablaOrigen, bool TablaDestino, Vector3 TamañoTOrigen, Vector TamañoTDestino)

Este método convierte las posiciones ocupadas de una malla mayor a una menor o viceversa. Una vez obtenida la TablaDestino crea el nuevo policubo con dicha tabla.

El pseudocódigo del método es:

```
Vector3 PosiciónOrigen
        Por XX=0 hasta XX <= TamañoTDestino.X hacer
                Por YY=0 hasta YY <= TamañoTDestino.y hacer
                        Por  ZZ=0 hasta ZZ <= TamañoTDestino.z hacer
                                PosicionOrigen.x = floor((((TamañoTOrigen.x) *
                                xx)/(TamañoTDestino.x)) + 0.5);
                                PosicionOrigen.y = floor((((TamañoTOrigen.y) *
                                yy)/(TamañoTDestino.y)) + 0.5);
                                PosicionOrigen.z = floor((((TamañoTOrigen.z) *
                                zz)/(TamañoTDestino.z)) + 0.5);
                                PosicionOrigen = PosicionOrigen+1;
                                Si TablaOrigen de (((PosicionOrigen.z +
                                (TamañoTOrigen.z + 1) * (PosicionOrigen.y - 1) +
                                ((TamañoTOrigen.y + 1) * (TamañoTOrigen.z + 1)) *
                                (PosicionOrigen.x - 1)) -1) entoces
                                        TablaDestino de ((zz + 1) + (TamañoTDestino.z
                                        + 1) * (yy)  + ((TamañoTDestino.y + 1) *
                                        (TamañoTDestino.z + 1)) * (xx)) -1) = Cierto
                                FinSin
                        FinPor
                FinPor
        FinPor

        Insertar Policubo TablaDestino
```

El procedimiento de este método se basa en un triple bucle, que hace un recorrido por la malla. Por cada iteración calcula la nueva posición origen, seguidamente mira si en TablaOrigen [PosiciónOrigen] = Cierto, en ese caso coloca a Cierto la posición de destino.

El la figura siguiente se puede ver la resolución del método matemático que se ha usado para calcular las nuevas posiciones, donde N es total de cubos del origen, M es el total de cubos del destino, i es la posición del cubo origen, i' es la posición que queremos saber y D es el tamaño. La formula resultante para calcular la posición es:

$$i' = N * \frac{i}{M}$$

En la Figura 32 se puede ver un ejemplo gráfico de cómo se obtiene la formula.

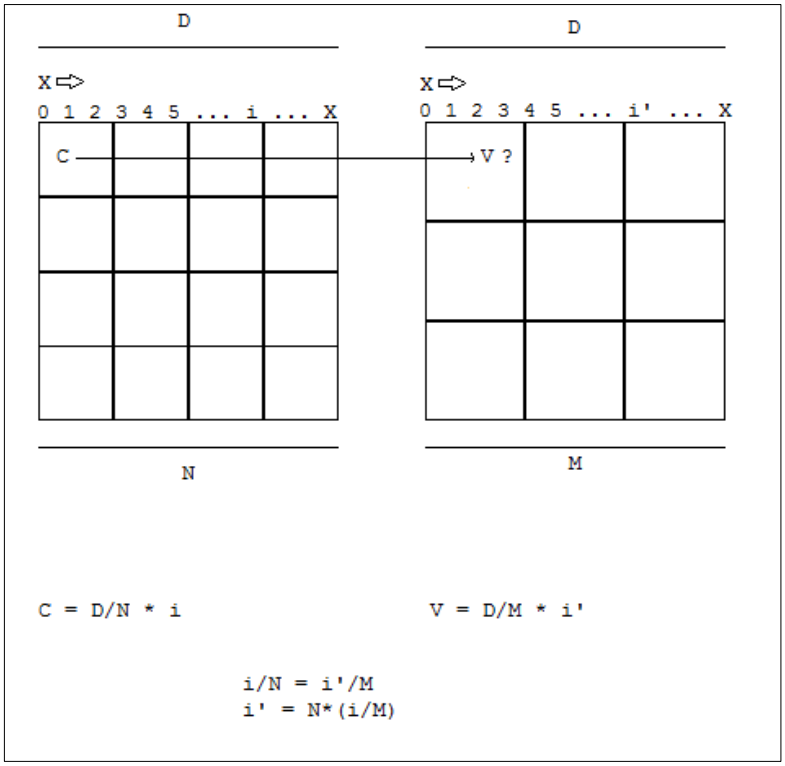


Figura 32 Conversión de matriz de mayor a menor tamaño

+ Guardar (String name)

Este método se encarga de guardar en un fichero XML el policubo y la información necesaria para más adelante recuperar el estado guardado. El fichero XML contiene la siguiente información.

- Modelo base: escala y posición
- Cubo Root: escala, posición y orientación.
- Tabla de booleanos que contiene los cubos puestos en pantalla

- Tamaño de la tabla de booleanos.
- La cantidad de cubos que caben en la malla virtual (x, y, z).

En la Figura 33 se puede ver un ejemplo de un fichero generado con la herramienta.

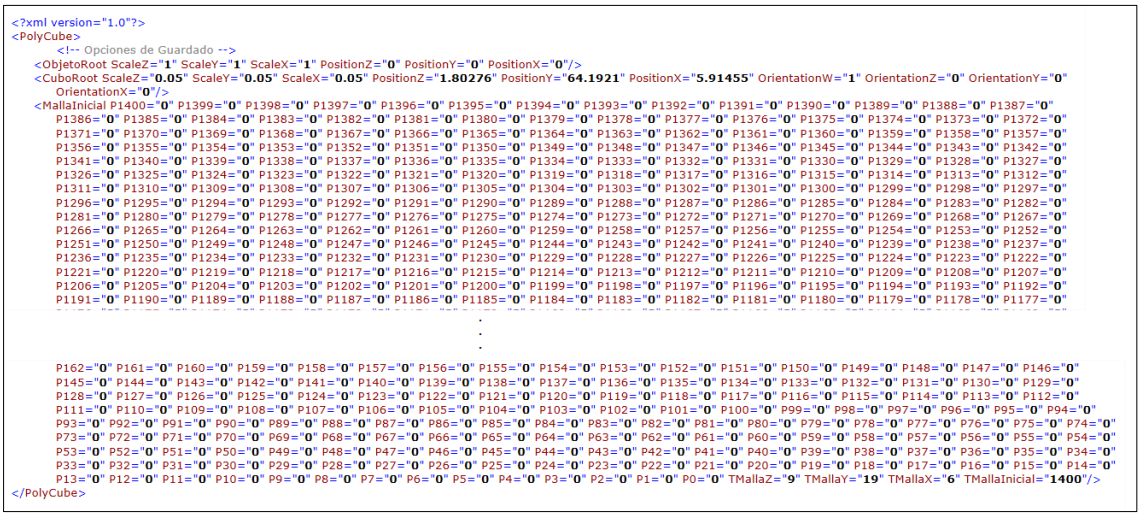


Figura 33 Ejemplo de fichero XML.

+ Intersect_Triangle (Vector3 OrigV, Vector3 DirV, Vector3 Vert0V, Vector Vert1V, Vector3 Vert3V, double t, double u, double v)

Este método retorna cierto si existe intersección entre un rayo y un triangulo.

8.11.2.2 Clase PFCListener

Esta clase se encarga de recoger los eventos que se generan en la herramienta (ratón, teclado, cegui,…). En la Figura 34 se pueden ver los atributos de la clase y los métodos.

PFCListener
bool mShutdownRequiested bool mContinue bool BotonDerechoRaton Real mRotate Real mMove
ExploradorWindowsCargar() ExploradorWindowsGuardar() frameEnded(Ogre::FraneEvent evt) keyPressed(OIS::KeyEvent e) keyReleased(OIS::KeyEvent e) mouseMoved(OIS::MouseEvent arg) mousePressed(OIS::MouseEvent arg, OIS::MouseButtomID id) mouseReleased(OIS::MouseEvent arg, OIS::MouseButtomID id) PFCListener(RenderWindow win, Camera cam, SceneManager sceneMgr, CEGUI::Renderer renderer) ScrollTamañoCubo(float Posicion, int Indice)

Figura 34 Clase PFCListener

Atributos

+ mShutdownRequested booleano que controla la salida del programa.

+ mContinue booleano que controla la salida del programa.

+ BotonDerechoRaton booleano que controla si esta apretado el botón derecho del ratón.

+ mRotate real que controla la velocidad de rotación de la cámara.

+ mMove real que controla la velocidad de movimiento de la cámara.

Métodos

+ ExploradorWindowsCargar()

Este método es llamado cuando se pulsa el botón del menú cargar, se encarga de abrir un explorador de Windows y obtener el nombre del fichero que se quiere cargar en la herramienta, acto seguido llama al método MatrizCubos::cargar (string name).

+ ExploradorWindowsGuardar()

Este método es llamado cuando se pulsa el botón del menú guardar, se encarga de abrir un explorador de Windows y guardar en un fichero XML la información del policubo y variables necesarias de la herramienta. Al obtener el nombre del fichero que se quiere guardar, se llama al siguiente método Matriz::guardar(string name).

+ frameEnded (FrameEvent evt)

Este método permite salir de la aplicación.

El pseudocódigo del método es:

```
Si mShutdownRequested entonces
        Devolver Falso
SiNo    Devolver Cierto

FinSi
```

+ frameRenderingQueued (FrameEvent evt)

Este método se encarga de capturar los eventos del teclado, ratón, ejecutar el movimiento de la cámara, rotación del policubo y cambiar el tamaño del primer cubo.

El pseudocódigo del método es:

```
Si mMouse entonces capturar ratón FinSi
Si mKyeboard entonces capturar teclado FinSi
mCamera->moveRelative(mDirection)
Si mKeyboard->isKeyDown (NUMPAD8) entonces CuboRoot->yaw(0.1) FinSi
Si mKeyboard->isKeyDown (NUMPAD2) entonces CuboRoot->yaw(-0.1)
FinSi
Si mKeyboard->isKeyDown (NUMPAD6) entonces CuboRoot->pitch(0.1)
FinSi
Si mKeyboard->isKeyDown (NUMPAD4) entonces CuboRoot->pitch(-0.1)
FinSi
Si mKeyboard->isKeyDown (NUMPAD9) entonces CuboRoot->roll(0.1) FinSi
Si mKeyboard->isKeyDown (NUMPAD7) entonces CuboRoot->roll(-0.1) FinSi
Si indice <= 1 entonces
        Si mKeyboard -> isKeyDown (SUBSTRACT) entonces CuboRoot ->
        setScale(CuboRoot -> getScale() – 0.0001) FinSi
        Si mKeyboard -> isKeyDown (ADD) entonces CuboRoot ->
        setScale(CuboRoot -> getScale() + 0.0001) FinSi
FinSi
```

Este método se ejecuta al final de cada frame, lo que permite efectuar movimientos. Si mantenemos pulsada la tecla 8 del teclado numérico, rotará hasta la tecla se deja de pulsar.

+ keyPressed (OIS::KeyEvent e)

Este método es el encargado de recoger las pulsaciones del teclado, exactamente cuando se oprime la tecla.

El pseudocódigo del método es:

```
Según tecla Hacer
        Caso ESCAPE
                Salir de la aplicación
        Caso UP o W
                Dirección.Z = -150
```

```
        Caso Down o S
                Dirección.Z = +150
        Caso Left o A
                Dirección.X = -150
        Caso Right o D
                Dirección.X = +150
        Caso PageDown o E
                Dirección.Y = -150
        Caso PageUp o Q
                Dirección.Y = +150
    FinSegún
    Devolver Cierto
```

+ keyReleased (OIS::Key Event e)

Este método es el encargado de recoger las pulsaciones del teclado, exactamente cuando se suelta la tecla.

El pseudocódigo del método es:

```
    Según e Hacer
        Caso UP o W
                Dirección.Z = 0
        Caso Down o S
                Dirección.Z = 0
        Caso Left o A
                Dirección.X = 0
        Caso Right o D
                Dirección.X = 0
        Caso PageDown o E
                Dirección.Y = 0
        Caso PageUp o Q
                Dirección.Y = 0
    FinSegún
    Devolver Cierto
```

+ mouseMoved(OIS::MouseEvent arg)

Este método es el encargado de recoger el desplazamiento del ratón, y si se oprime el botón derecho del ratón mover la cámara en la dirección del desplazamiento del ratón.

El pseudocódigo del método es:

```
arg =  Capturar movimiento del raton
Si BotonDerechoRaton entonces
        Camara->yaw(arg X)
        Camara->pitch(arg y)
FinSi
```

+mousePressed(OIS::MouseEvent, OIS::MouseButtomID id)

Este método controla que el botón del ratón se ha tocado, si el botón izquierdo a sido oprimido mira si se puede colocar un cubo, si es el botón derecho comprueba si se ha apretado la tecla espacio, y si es así comprueba si se puede borrar un cubo.

El pseudocódigo del método es el siguiente:

```
Si id = botón izquierdo hacer
        Si ResoluciónOFF = cierto hacer
                CEGUI::Point mousePos = Obtener posición del ratón en
                pantalla
                double t, u, v
                int resultado, ,i PosicionPantallaX, PosicionPantallaY
                Si PrimerCubo hacer
                        Índice = 0
                                PosicionPantallaX = mousePos.d_x
                        PosicionPantallaY = mousePos.d_y
                        PosicionReal = PosicionPantallaY * 800 * 4
                        PosicionPantallaX * 4

                        Poner al modelo base el material que obtiene las
                        distancias.

                        Actualizar el Render to Texture.
                        Float pBufferBoxFloat = Render to Texture
```

```
                    PosicionRaton.x = pBufferBoxFloat[PosicionReal]
                    PosicionRaton.y = pBufferBoxFloat[PosicionReal+1]
                    PosicionRaton.z = pBufferBoxFloat[PosicionReal+2]

                    Poner al modelo base el material original

                    Si se selecciono el modelo base hacer
                        Activar scrollbars del menú
                        Insertar cubo root en PosicionRaton
                        Índice = índice +1
                        PrimerCubo = Falso
                    FinSi
                FinSi
```

El pseudocódigo de arriba lo que hace es comprobar si se ha apretado el botón izquierdo, en ese caso como se quiere insertar un cubo se deshabilita la barra del cambió de resolución. El siguiente es paso es verificar si es el primer cubo, si lo es, se calcula la posición del ratón en pantalla y se traduce con el material del modelo base al entorno 3D. Una vez obtenida la posición 3D se inserta el cubo y se activa la barra de resolución.

Si el primer cubo ya esta puesto el método sigue con el siguiente pseudocódigo.

```
            Si índice = 1 entonces

                TempMalla = Calcular tamaño Malla
                Obtener Posición Cubo Root
                PosicionMalla = Calcular Posición Cubo Root en la Malla
                MallaMul = TempMalla + 1
                TamañoMallaInicial = MallaMul X * MallaMul Y * MallaMul Z
                MallaInicial = Crear Malla  de tamaño TamañoMallaInicial

                Por i = 0 hasta TamañoMallaInicial hacer
                    MallaInicial de i = falso
                FinPor
```

```
                MallaInicial[((PosicionMalla.z + MallaMulZ *
                (PosicionMalla.y - 1) + (MallaMulY * MallaMulZ) *
                (PosicionMalla.x - 1)) - 1)] = Cierto

                Poner CuboRoot en la posición correcta
        FinSi
```

En el pseudocódigo anterior se comprueba si se va a poner el segundo cubo, si es así, se calcula el tamaño de la matriz de booleanos y se reserva el espacio, también se coloca el cubo root en su posición.

Seguidamente se crea un iterador del RayoCamara con la posición del ratón y se comprueba todos los elementos con los que ha intersectado. Si encuentra un cubo, al estar ordenado por distancia se obtiene la cara para poder insertar más tarde el cubo.

```
Itr = Crear iterador  RayoCamara con la posición del ratón
Int cara = 0

Por itr inicio hasta itr final hacer
        Si itr es movible hacer
                Por i = 0 hasta NodoCubo Tamaño hacer
                        Si itr = NodoCubo[i] hacer
                                Cara = ChekRay (rayoraton,
                        NodoCubos[i])
                                Si Cara >= 1 hacer
                                        Break
                                FinSi
                        FinSi
                FinPor
        FinSi
FinPor
```

En el caso que se haya encontrada un cara sigue con este pseudocódigo.

```
Si cara es diferente de 0 hacer

        Posición = CalculaPosicion (NodoCubos [i], escala, cara,
        orientación)
        PosicionMalla = Calcular posición del cubo en malla

        Si TempMalla.x >= PosicionMalla.x && PosicionMalla.x >= 0
        &&
        TempMalla.y >= PosicionMalla.y && PosicionMalla.y >= 0 &&
        TempMalla.z >= PosicionMalla.z && PosicionMalla.z >= 0
hacer

        Insertar cubo en Posicion

        MallaInicial[((PosicionMalla.z + MallaMulZ * (PosicionMalla.y
        - 1) + (MallaMulY * MallaMulZ) * (PosicionMalla.x - 1)) - 1)] =
        Cierto

        Índice = Índice + 1

        Si (índice es igual a 2) hacer
                Scrollbar[0] = falso

        FinSi

Finsi

FinSi
```

En el pseudocódigo anterior se calcula la posición del cubo a insertar con el método *CalculaPosicion,* que se le pasan los siguientes valores: el cubo con el que se a intersectado, la escala, la cara con la que hubo intersección y la orientación. Con esto obtenemos la posición del nuevo cubo y lo podemos insertar tanto en pantalla como en la matriz de boleanos. También se bloquea la barra de cambio de tamaño del cubo root.

En el caso que se haya apretado el botón derecho del ratón, también se comprueba si se ha apretado la barra espaciadora como se puede ver en el siguiente pseudocódigo.

```
Si id = botón derecho hacer
        BotonDerechoRaton = cierto

        Itr = Crear iterador  RayoCamara con la posición del ratón
        Int cara = 0

        Si teclado es igual a espacio hacer


        Por itr inicio hasta itr final hacer
                Si itr es movible hacer
                        Por i = 0 hasta NodoCubo Tamaño hacer
                                Si itr = NodoCubo[i] hacer
                                        Cara = ChekRay (rayoraton,
                                NodoCubos[i])
                                        Quitar cubo de i de la escena.
                                        Si Cara >= 1 hacer
                                                Break
                                        FinSi
                                FinSi
                        FinPor
                FinSi
        FinPor
        FinSi
Finsi
```

En el pseudocódigo anterior se puede ver como se crea un iterador del RayoCamara con la posición del ratón y se comprueba todos los elementos con los que ha intersectado. Si encuentra un cubo se quita de la escena.

El procedimiento de este método se puede ver en el siguiente diagrama de estado.

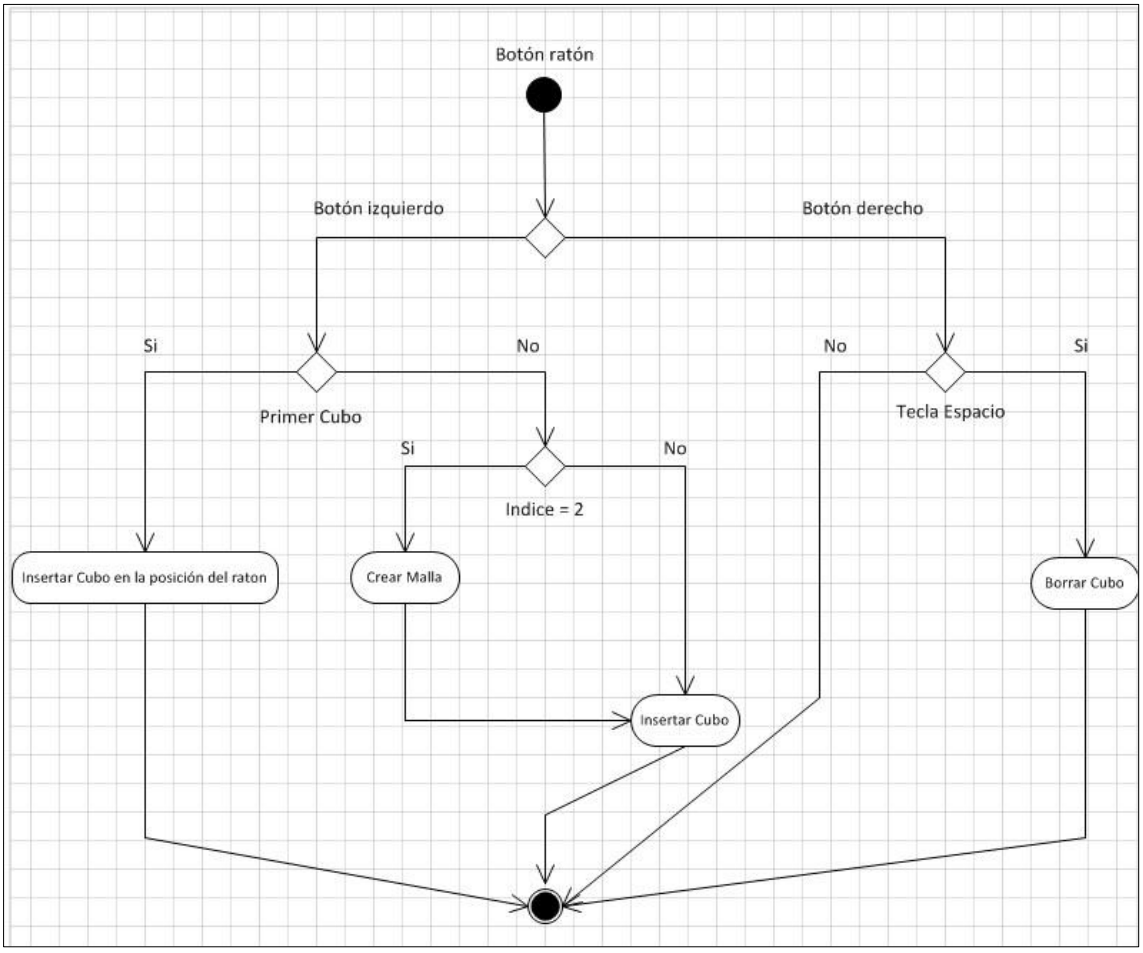


Figura 35 Diagrama de estado de MousePressed

+ mouseReleased (OIS::MouseEvent arg, OIS MouseButtom ID)

Este método actualiza la variable *BotonDerechoRoton = falso* cuando se deja de apretar el botón.

El pseudocódigo del método es:

```
Si id es igual a Boton Derecho entonces
        BotonDerechoRaton = falso
FinSi
```

+ **PFCListener (RenderWindow win, Camera cam, SceneManager sceneMgr, CEGUI::Renderer renderer)**

Este método es el constructor del PFCListener.

+ requestShutdown (FrameEvent evt)

Este método se encarga de salir de la herramienta si el usuario ha presionado el botón salir del menú.

+ScrollTamañoCubo (float posición, int índice)

Este método aplica los cambios se han ocasionado en las barras del menú por interacción del usuario.

El pseudocódigo del método es:

```
Si índice es igual a 0 hacer
        STamañoCubo = posición
        CuboRoot->escala = STamañoCubo
FinSi

Si índice es igual a 1 hacer
        MaterialPTR MaterialCubo = Obtener MaterialCubo
        ColourValue CuboTransparencia = 1 – Posición
        MaterialCubo = CuboTransparencia
FinSi

Si índice es igual a 2 hacer
        ResolucionOFF = falso
        STamañoObjeto = Posicion / 6
        STamañoObjeto = STamañoObjeto + 0.015

        Borrar policubo con la nueva resolución
        TempMallaNueva = Calcular tamaño Malla
        TamañoMallaNueva = TempMallaNueva.X * TempMallaNueva.Y *
        TempMallaNueva.Z
        MallaNueva = reservar espacio de booleanos con
        TamañoMallaNueva
        Inicializar MallaNueva a falso
        ConversorTabla(MallaInicial, MallaNueva, TempMalla,
        TempMallaNueva)
FinSi
```

Si índice es igual a 3 **hacer**
MaterialPTR MaterialMBase = *Obtener MaterialModeloBase*
ColourValue MBaseTransparencia = 1 – Posición
MaterialMBase = MBaseTransparencia
FinSi

8.11.2.3 Clase PFCApplication

Esta clase es la encargada de inicializar la herramienta (luces, cámaras, interfaz de usuario,…). En la siguiente Figura 36 se pueden ver sus atributos y métodos.

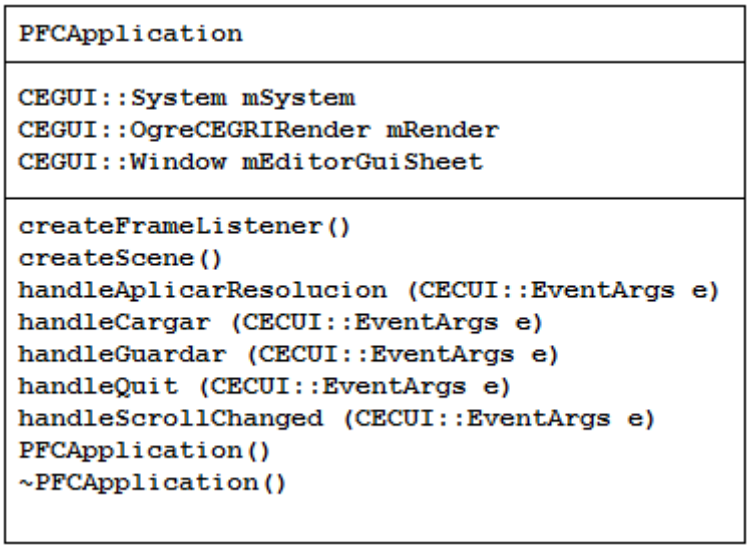


Figura 36 Clase PFCApplication

Atributos

+ CEGUI::System mSystem

+ CEGUI::OgreCEGUIRender mRender

+ CEGUI::Window mEditorGuiSheet

Métodos

+ createFrameListener ()

Este método crea un listener

+ createScene ()

Este método inicializa la escena, creando una cámara, una luz, interfaz de usuario, etc.

+ handleAplicarResolucion(CEGUI::EventArgs e)

En este método si se oprime el botón del menú Aplicar Resolución se encarga de hacer una llamada al PFCListener->AplicarResolucion()

+ handleCargar(CEGUI::EventArgs e)

En este método si se oprime el botón del menú Cargar se encarga de hacer una llamada al PFCListener->Cargar()

+ handleGuardar(CEGUI::EventArgs e)

En este método si se oprime el botón del menú Guardar se encarga de hacer una llamada al PFCListener->Guardar()

+ handleQuit(CEGUI::EventArgs e)

En este método si se oprime el botón del menú Salir se encarga de hacer una llamada al PFCListener->requestShutdown()

+ handleScrollChanged(CEGUI::EventArgs e)

En este método si desliza una barra del menú se encarga de hacer una llamada al PFCListener->ScrollTamañoCubo()

El pseudocódigo del método es:

```
name = Obtener nombre de la barra que se deslizado
Por i = 0 hasta 3 hacer
        Si BarraNombre[i] es igual nombre entonces
                PFCListener (Barra[i]->Posicion, i)
        FinSi
FinPor
```

+PFCApplication ()

Este método es el constructor por defecto.

9 Implementación y pruebas

En esta sección se explicará cómo se ha implementado cada elemento del proyecto, para poder llegar al resultado final.

9.1 Interfaz de usuario

La interfaz de usuario esta hecha con las librerías CEGUI, la inicialización de la interfaz se hace en PFCApplication::createScene()

CEGUI Setup

El constructor que se utilizado para generar la interfaz ha sido del siguiente:

```
OgreCEGUIRenderer(Ogre::RenderWindow* window, Ogre::uint8 queue_id =
Ogre::RENDER_QUEUE_OVERLAY, bool post_queue = false, uint max_quads
= 0)
```

mWindows es un puntero al Objeto Ogre::RenderWindow
Ogre::RENDER_QUEUE_OVERLAY especifica cuando la interfaz se ejecuta en renderización de Ogre.
postqueue En False indica que se tiene que haber renderizado después del queue_id.
max_quads está obsoleto: no se usa y se tiene que poner en 0.

La clase System del CEGUI el la clase que proporciona acceso a todos los demás elementos de la clase CEGUI. Este objeto se tiene que crear por la aplicación, se le tiene que pasa un objeto Renderer para poder interactuar con el sistema y así poder mostrar las imágenes de la Interfaz.

```
mRenderer = new CEGUI::OgreCEGUIRenderer(mWindow,
Ogre::RENDER_QUEUE_OVERLAY, false, 0, mSceneMgr);
mSystem = new CEGUI::System(mRenderer);
```

Una vez inicializada la Interfaz de usuario, se ha cargado el esquema de visualización, para ello se ha hecho uso de la clase CEGUI::SchemeManager, esta clase se encarga de la creación, acceso y destrucción de los objetos scheme.

```
CEGUI::SchemeManager::getSingleton().loadScheme((CEGUI::utf8*)"Tahar
ezLookSkin.scheme");
```

En la figura siguiente se puede ver “skin” (piel) que se ha utilizado.

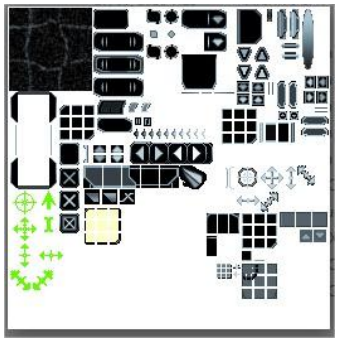


Figura 37 Taharez Skin

Con el siguiente código se especifica la forma del ratón y el tipo de letra de la interfaz.

```
mSystem->setDefaultMouseCursor((CEGUI::utf8*)"TaharezLook",
(CEGUI::utf8*)"MouseArrow");
mSystem->setDefaultFont((CEGUI::utf8*)"BlueHighway-12");
```

Una vez hecho esto, sólo queda cargar el menú. Se ha necesitado la clase CEGUI::WindowsManager que es la encargada de crear y destruir ventanas.

```
CEGUI::Window* sheet =
CEGUI::WindowManager::getSingleton().loadWindowLayout((CEGUI::utf8*)
"PolyCube.layout");
mSystem->setGUISheet(sheet);
```

En la siguiente Figura 38, se puede observar el menú cargado.

Figura 38 CECUI Menú

Para generar el menú se utiliza un fichero en XML que se puede ver en el apartado 14.1, para poder controlar los eventos que hay en el menú se ha hecho con la clase CEGUI::WindowsManager, donde se puede ver en el siguiente código.

```
//GUI Menu
//Boton Salir
CEGUI::WindowManager& wmgr = CEGUI::WindowManager::getSingleton();
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/Salir")
->subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleQuit, this));
//Boton Cargar
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/Cargar")
->subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleCargar, this));
//Boton Guardar
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/Guardar")
->subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleGuardar, this));
//Boton Aplicar Resolucion
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/AplicarResolucion")
->subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleAplicarResolucion,
this));
//ScrollBars
for (int i = 0; i < SI_COUNT; ++i)
{
      scrollbars[i] = static_cast<CEGUI::Scrollbar*>(
      wmgr.getWindow(scrollbarNames[i]));
      scrollbars[i]->subscribeEvent(
      CEGUI::Scrollbar::EventScrollPositionChanged,
      CEGUI::Event::Subscriber(&PFCApplication::handleScrollChanged,
      this));
      // disable to begin with
      scrollbars[i]->setEnabled(false);
}
```

Como se puede observar en el código, hay dos tipos eventos diferentes: primero los botones que se controlan con la clase CEGUI::PushButton y segundo las barras de desplazamiento que se controlan con la clase CEGUI::Scrollbar. Cuando se ocasiona un evento se llama al método encargado de resolver la petición.

9.2 Movimiento

En este apartado se mostrará el código que ha sido necesario para generar el movimiento de la cámara con el teclado y/o el ratón. También se mostrará como se consigue la rotación del policubo.

9.2.1 Movimiento de la cámara con el teclado

Gracias a las librerías ya implementadas en el sistema, el movimiento de la cámara resultó sencillo de programar. El teclado se encarga de mover la cámara por los ejes (x, y, z).

El código que se encarga de gestionar el movimiento de la cámara con el input de teclado es el siguiente:

```
bool keyPressed (const OIS::KeyEvent & e)
{
switch (e.key)
{
case OIS::KC_ESCAPE:
            mShutdownRequested = true;
            break;
      case OIS::KC_UP:
      case OIS::KC_W:
            mDirection.z = -mMove;
            break;
      case OIS::KC_DOWN:
      case OIS::KC_S:
            mDirection.z = mMove;
            break;
      case OIS::KC_LEFT:
      case OIS::KC_A:
            mDirection.x = -mMove;
            break;
      case OIS::KC_RIGHT:
      case OIS::KC_D:
            mDirection.x = mMove;
            break;
      case OIS::KC_PGDOWN:
      case OIS::KC_E:
            mDirection.y = -mMove;
            break;
      case OIS::KC_PGUP:
      case OIS::KC_Q:
            mDirection.y = mMove;
            break;
}
return true;
}
```

Como se puede ver en el código, cuando se detecta que se ha presionado una tecla que se tiene asignada para el movimiento de la cámara, se actualiza con la variable mMove previamente declarara en la clase e inicializada en el constructor a 150. El código para generar el movimiento es el siguiente:

```
mCamera->moveRelative(mDirection);
```

Una vez se deja de presionar la tecla se poner el valor de mDirection a 0 como se puede ver en el siguiente código:

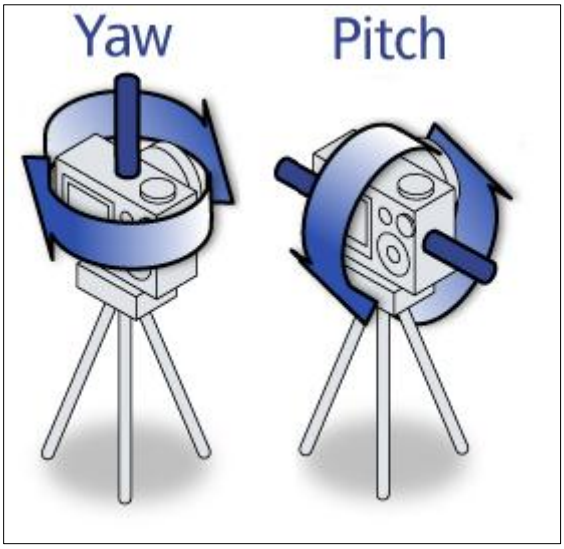
```
bool keyReleased (const OIS::KeyEvent & e)
{
	switch (e.key)
		{
		case OIS::KC_UP:
		case OIS::KC_W:
			mDirection.z = 0;
			break;
		case OIS::KC_DOWN:
		case OIS::KC_S:
			mDirection.z = 0;
			break;
		case OIS::KC_LEFT:
		case OIS::KC_A:
			mDirection.x = 0;
			break;
		case OIS::KC_RIGHT:
		case OIS::KC_D:
			mDirection.x = 0;
			break;
		case OIS::KC_PGDOWN:
		case OIS::KC_E:
			mDirection.y = 0;
			break;
		case OIS::KC_PGUP:
		case OIS::KC_Q:
			mDirection.y = 0;
			break;
		default:
			break;
		}
return true;
}
```

9.2.2 Movimiento de la cámara con el ratón

El código que se encarga de gestionar el movimiento de la cámara con ratón:

```
bool mouseMoved(const OIS::MouseEvent &arg)
{
	CEGUI::System::getSingleton().injectMouseMove(arg.state.X.rel,
	arg.state.Y.rel);
	if (BotonDerechoRaton)
	{
		mCamera->yaw(Degree(-arg.state.X.rel * mRotate));
		mCamera->pitch(Degree(-arg.state.Y.rel * mRotate));
	}
	return true;
}
```

Con este código podemos cambiar el grado de inclinación de la cámara. En la figura siguiente se muestra el tipo de movimiento que se puede realizar.

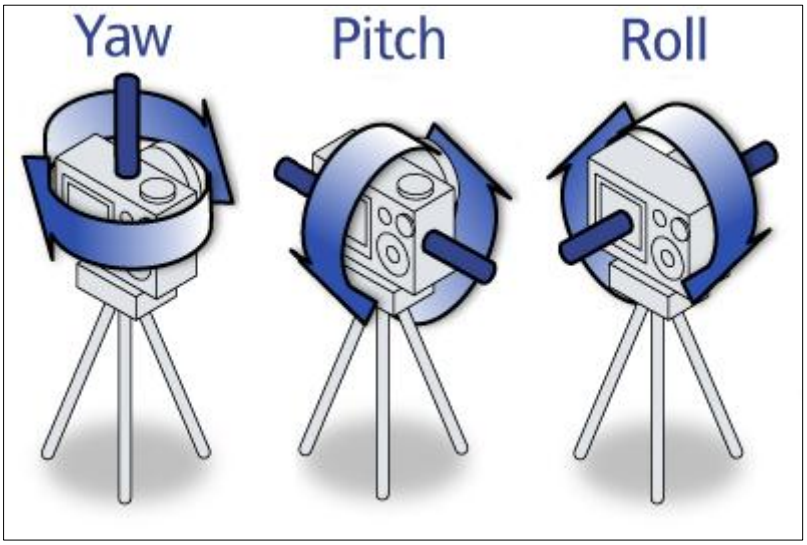


9.2.3 Movimiento de policubo con el teclado

El movimiento del policubo se basa en rotar el cubo root con esto se consigue que los hijos roten en la misma dirección. El código para generar el movimiento del policubo es el siguiente:

```
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD8)){mSceneMgr->getSceneNode
("CuboRoot")->yaw(Degree(0.1));
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD2)){mSceneMgr->getSceneNode
("CuboRoot")->yaw(Degree(-0.1));}
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD6)){mSceneMgr->getSceneNode
("CuboRoot")->pitch(Degree(0.1));}
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD4)){mSceneMgr->getSceneNode
("CuboRoot")->pitch(Degree(-0.1));}
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD9)){mSceneMgr->getSceneNode
("CuboRoot")->roll(Degree(0.1));}
if (mKeyboard->isKeyDown(OIS::KC_NUMPAD7)){mSceneMgr->getSceneNode
("CuboRoot")->roll(Degree(-0.1));}
```

En la figura siguiente se muestra el tipo de movimiento que se puede realizar.



9.3 Insertar cubos

En este apartado se hablara de las diferentes estrategias para colocar cubos según si es el primer cubo (cubo root), segundo cubo, o los siguientes.

9.3.1 Insertar Cubo Root

Para poder insertar cubo root en la herramienta el primer paso es obtener la posición del puntero en la pantalla, este paso es simple gracias a las librerías CEGUI que facilitan el proceso. El código utilizado es el siguiente:

```
CEGUI::Point mousePos = CEGUI::MouseCursor::getSingleton
().getPosition();
```

Con la instrucción anterior se obtiene la posición pantalla, pero lo que realmente se quiere conseguir es la posición dentro de la escena para saber si el clic del ratón ha impactado en algún sitio del modelo base y a su vez insertar el cubo.

El método para conseguir esta posición se basa en crear un material para el modelo base que lo pintará con el valor de las distancias que hay dentro de la escena. Para poder crear este material se ha hecho uso del lenguaje de programación Cg que permite programar los Shaders de la GPU. El código para cargar el material es el siguiente:

```
Entity* Objeto = (Entity*)ObjetoRoot->getAttachedObject("Objeto");
Objeto->setMaterialName("Examples/PFCDistanciaWorld");
```

Una vez que el modelo base tiene el material se procede a actualizar el Render to Texture y así obtener la imagen de la GPU en una variable del tipo float. Después se vuelve a poner el material visible al modelo base.

```
//Actualización de lo que se ven pantalla al textura RTT
Ogre::Texture *mTexture;
mTexture =
(Ogre::Texture*)Ogre::TextureManager::getSingleton().getByName("Rend
er_To_Texture").getPointer();
int mWidth = mTexture->getSrcWidth();
int mHeight = mTexture->getSrcHeight();
RenderTarget *RenderToTexture02 = mTexture->getBuffer()-
>getRenderTarget();
RenderToTexture02->update(true);

//Baja la textura del la GPU a la CPU
unsigned char* pBufferBox = new unsigned char[mTexture-
>getBuffer(0,0)->getSizeInBytes()];
float* pBufferBoxFloat = (float*)pBufferBox;
Ogre::PixelBox mPixelBox01 = Ogre::PixelBox(mTexture->getWidth(),
mTexture->getHeight(), 1, mTexture->getFormat(), pBufferBox);
```

```
mTexture->getBuffer(0, 0)->blitToMemory(mPixelBox01);

//Pone el material del robot "que se ve"
Objeto->setMaterialName("PFC2/Objeto");
```

Con esto tenemos en pBufferBoxFloat la posición de cada pixel del modelo base, con el siguiente código se puede obtener la posición en el mundo 3D.

```
PosicionPantallaX = mousePos.d_x;
PosicionPantallaY = mousePos.d_y;
PosicionReal = PosicionPantallaY*800*4+PosicionPantallaX*4;

PosicionRaton.x = pBufferBoxFloat[PosicionReal];
PosicionRaton.y = pBufferBoxFloat[PosicionReal+1];
PosicionRaton.z = pBufferBoxFloat[PosicionReal+2];
```

Una vez obtenida la PosicionRaton se pude poner el cubo root con el siguiente código:

```
Cubos.push_back(mSceneMgr->createEntity("Cubo" +
StringConverter::toString(indice), "cube.mesh"));
NodoCubos.push_back (mSceneMgr->getSceneNode("CuboRoot")-
>createChildSceneNode("CuboNodo" +
StringConverter::toString(indice)));
NodoCubos[NodoCubos.size()-1]->attachObject(Cubos[Cubos.size()-1]);
mSceneMgr->getSceneNode("CuboRoot")->setPosition(PosicionRaton);
Cubos[NodoCubos.size()-1]->setMaterialName("PFC2/Cubo");
```

9.3.2 Insertar segundo cubo

Una vez insertado el cubo root, para insertar el segundo cubo ya se puede calcular la malla virtual (rejilla) en el modelo base y colocar el cubo root en la posición mas cercana de la rejilla como, se muestra en el apartado 8.7, y el segundo cubo en la cara que se ha seleccionado del cubo root.

Malla virtual

Primero se calculan cuántos cubos caben en la malla gracias al siguiente código:

```
TempMalla = CalculoMalla (mSceneMgr->getSceneNode("ObjetoRoot")-
>_getWorldAABB(), mSceneMgr->getSceneNode("CuboNodo0")-
>getAttachedObject("Cubo0")->getBoundingBox(), mSceneMgr-
>getSceneNode("CuboRoot")->getScale());
```

Con el siguiente código obtenemos la posición (x, y, z) del cubo root en la malla. En la Figura 39 se puede ver un ejemplo gráfico de cómo se calcula la posición.

```
AxisAlignedBox CajaRoot = mSceneMgr->getSceneNode("ObjetoRoot")-
>_getWorldAABB();
PosicionMalla = (mSceneMgr->getSceneNode("CuboRoot")->getPosition()
- CajaRoot.getMinimum()) / (100 * mSceneMgr-
>getSceneNode("CuboRoot")->getScale());
```

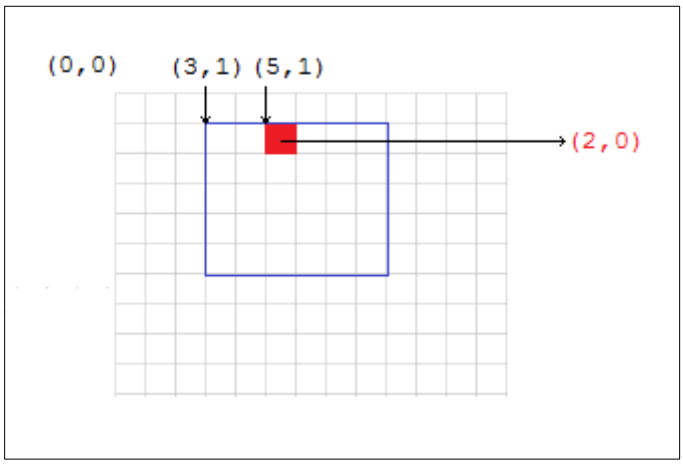


Figura 39 Ejemplo de como se calcula la posición del cubo en la malla.

El paso siguiente es calcular cuantos cubos caben en la malla para poder reservar el espació de la tabla de booleanos.

```
//Calculo el tamaño de la malla y reservo el espacio
TamañoMallaInicial = (MallaMulX * MallaMulY * MallaMulZ);
MallaInicial = new bool [TamañoMallaInicial];
```

Con el siguiente código se calcula posición del cubo en la tabla de booleanos y se pone la entrada correspondiente a cierto.

```
MallaInicial[(long int)((PosicionMalla.z+MallaMulZ*(PosicionMalla.y-
1)+(MallaMulY*MallaMulZ)*(PosicionMalla.x-1))-1)] = true;
```

El siguiente paso es colocar en cubo root en la posición mas cercana de la rejilla como se muestra en el apartado 8.7. El código es el siguiente.

```
mSceneMgr->getSceneNode("CuboRoot")-
>setPosition(CajaRoot.getMinimum() + ((PosicionMalla) * 100 *
mSceneMgr->getSceneNode("CuboRoot")->getScale()) + (50 * mSceneMgr-
>getSceneNode("CuboRoot")->getScale()));
```

La fórmula para calcular la posición es la siguiente:

$$Posición = (PosiciónMalla) * (TamañoCubo * Escala) + (MedioCubo * Escala)$$

Inserción del segundo cubo

Para colocar el segundo cubo una vez creada la malla, el procedimiento es el siguiente: se crea rayo con la posición de la cámara. Este rayo nos devuelve todos los objetos en los que han impactado, el siguiente paso es ordenarlos por distancia como se puede ver en el siguiente código:

```
Ray mouseRay = mCamera-
>getCameraToViewportRay(mousePos.d_x/float(arg.state.width),
mousePos.d_y/float(arg.state.height));
mRaySceneQuery->setRay(mouseRay);
mRaySceneQuery->setSortByDistance(true); //Ordena por distancia
```

Una vez obtenido el rayo se crea un iterador para recorrer el contenido y verificar si es un cubo. En el caso de que se encuentre un cubo, se llama al método checkRay para obtener la cara del cubo. Sólo se mira el cubo más cercano a la cámara.

```
// Execute query
RaySceneQueryResult &result = mRaySceneQuery->execute();
RaySceneQueryResult::iterator itr;
//Cojo el primer objeto
int cara = 0;
for (itr = result.begin(); itr != result.end(); itr++)
      {
      if (itr->movable)
            {
            for (i = 0; i < NodoCubos.size(); i++)
            {
                  if (NodoCubos[i] == itr->movable-
                  >getParentSceneNode())
                  {
                        cara = CheckRay (mouseRay, NodoCubos[i]);
                              if (cara >= 1)
                        {
                              break;
                        }
                  }
            }
      }
```

Con esto, si la cara es más grande de 0, se puede colocar el cubo controlando que no se salga de la malla, como se puede ver en el siguiente código.

```
Posicion = CalculaPosicion(NodoCubos[i]->getPosition() , mSceneMgr-
>getSceneNode("CuboRoot")->getScale(), cara, mSceneMgr-
>getSceneNode("CuboRoot")->getOrientation());
//Calcula la posición del cubo en la malla
AxisAlignedBox CajaRoot = mSceneMgr->getSceneNode("ObjetoRoot")-
>_getWorldAABB();
PosicionMalla = (Posicion * mSceneMgr->getSceneNode("CuboRoot")-
>getScale() + mSceneMgr->getSceneNode("CuboRoot")->getPosition()
- CajaRoot.getMinimum()) / (100 * mSceneMgr-
>getSceneNode("CuboRoot")->getScale());

PosicionMalla.x = floor (PosicionMalla.x);
PosicionMalla.y = floor (PosicionMalla.y);
PosicionMalla.z = floor (PosicionMalla.z);

//Controla que el cubo no se salga de la malla tempmalla es tamaño
maximo de cubo en (x,y,z)
if (TempMalla.x >= PosicionMalla.x && PosicionMalla.x >= 0 &&
TempMalla.y >= PosicionMalla.y && PosicionMalla.y >= 0 &&
TempMalla.z >= PosicionMalla.z && PosicionMalla.z >= 0)
      {
      //Vector de cubos puestos en pantalla
      Cubos.push_back(mSceneMgr->createEntity("Cubo" +
      StringConverter::toString(indice), "cube.mesh"));
      NodoCubos.push_back (mSceneMgr->getSceneNode("CuboRoot")-
      >createChildSceneNode("CuboNodo" +
      StringConverter::toString(indice)));
      NodoCubos[NodoCubos.size()-1]-
      >attachObject(Cubos[Cubos.size()-1]);
      NodoCubos[NodoCubos.size()-1]->setPosition(Posicion);
      Cubos[NodoCubos.size()-1]->setMaterialName("PFC2/Cubo");
      int MallaMulX= (int)TempMalla.x + 1;
      int MallaMulY= (int)TempMalla.y + 1;
      int MallaMulZ= (int)TempMalla.z + 1;
      PosicionMalla = PosicionMalla +1;

      MallaInicial[(long
      int)((PosicionMalla.z+MallaMulZ*(PosicionMalla.y-
      1)+(MallaMulY*MallaMulZ)*(PosicionMalla.x-1))-1)] = true;
      }
```

9.3.3 Inserción de los próximos cubos

El procedimiento para insertar los siguientes cubos es el mismo que el de insertar el segundo cubo pero sin hacer el cálculo de la malla.

9.4 Borrar Cubo

Para borrar un cubo del policubo, el procedimiento es el siguiente: se comprueba que se está presionando el botón derecho del ratón y al mismo tiempo si se presiona la tecla del espacio. En ese caso se crea rayo con la posición de la cámara.

Este rayo nos devuelve todos los objetos en los que impactado. El paso siguiente es ordenarlos por distancia como se puede ver en el siguiente código:

```
Ray mouseRay = mCamera-
>getCameraToViewportRay(mousePos.d_x/float(arg.state.width),
mousePos.d_y/float(arg.state.height));
mRaySceneQuery->setRay(mouseRay);
mRaySceneQuery->setSortByDistance(true); //Ordena por distancia
```

Una vez obtenido el rayo se crea un iterador para recorrer el contenido de él y buscar el cubo, si se encuentra el cubo se actualiza la tabla de boléanos y se quita el cubo de la escena.

```
// Execute query
RaySceneQueryResult &result = mRaySceneQuery->execute();
RaySceneQueryResult::iterator itr;
//Cojo el primer objeto
int cara = 0;
for (itr = result.begin(); itr != result.end(); itr++)
      {
      if (itr->movable)
            {
            for (i = 0; i < NodoCubos.size(); i++)
            {
                  if (NodoCubos[i] == itr->movable-
                  >getParentSceneNode())
                  {
                  MallaInicial[(long
                  int)((PosicionMalla.z+MallaMulZ*(PosicionMalla.y-
                  1)+(MallaMulY*MallaMulZ)*(PosicionMalla.x-1))-1)]
                  = false;
                  NodoCubos[i]->detachObject(Cubos[i]);
                  break;
                  }
            }
      `
```

9.5 Guardar el policubo

Para guardar el policubo se hace mediante el evento explicado anteriormente en el apartado 9.1 concretamente con el siguiente código:

```
//Boton Cargar
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/Guardar")
>subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleGuardar, this));
```

Este evento se encargará de llamar al método handleCargar y este avisa al PFCListener de que se tiene que ejecutar el método ExploradorWindowsGuardar(), que abrirá una ventana del explorador de windows y obtendrá el nombre que se le quiere poner al fichero XML. El código de abrir el explorador del windows se puede ver en el anexo. Para guardar los valores se utiliza el siguiente código:

- Del modelo base se guarda la posición y la escala.

```
TiXmlElement * eObjetoRoot = new TiXmlElement( "ObjetoRoot" );
root->LinkEndChild( eObjetoRoot );
Real gx,gy,gz;
//Guarda la posicion del objeto
Ogre::Vector3 gv3 = mSceneMgr->getSceneNode("ObjetoRoot")-
>getPosition();
gx = gv3.x; gy = gv3.y; gz = gv3.z;
eObjetoRoot-
>SetAttribute("PositionX",StringConverter::toString(gx).c_str());
eObjetoRoot-
>SetAttribute("PositionY",StringConverter::toString(gy).c_str());
eObjetoRoot-
>SetAttribute("PositionZ",StringConverter::toString(gz).c_str());
//Guarda la escala del objeto
gv3 = mSceneMgr->getSceneNode("ObjetoRoot")->getScale();
gx = gv3.x; gy = gv3.y; gz = gv3.z;
eObjetoRoot-
>SetAttribute("ScaleX",StringConverter::toString(gx).c_str());
eObjetoRoot-
>SetAttribute("ScaleY",StringConverter::toString(gy).c_str());
eObjetoRoot-
>SetAttribute("ScaleZ",StringConverter::toString(gz).c_str());
```

- Del cubo root se guarda, posición, escala y orientación.

```
TiXmlElement * eCuboRoot = new TiXmlElement( "CuboRoot" );
root->LinkEndChild( eCuboRoot );
//Guarda la posicion del cubo inicial
gv3 = mSceneMgr->getSceneNode("CuboRoot")->getPosition();
gx = gv3.x; gy = gv3.y; gz = gv3.z;
eCuboRoot-
>SetAttribute("PositionX",StringConverter::toString(gx).c_str());
eCuboRoot-
>SetAttribute("PositionY",StringConverter::toString(gy).c_str());
eCuboRoot-
>SetAttribute("PositionZ",StringConverter::toString(gz).c_str());
//Guarda la escala del cubo inicial
gv3 = mSceneMgr->getSceneNode("CuboRoot")->getScale();
gx = gv3.x; gy = gv3.y; gz = gv3.z;
eCuboRoot-
>SetAttribute("ScaleX",StringConverter::toString(gx).c_str());
eCuboRoot-
>SetAttribute("ScaleY",StringConverter::toString(gy).c_str());
eCuboRoot-
>SetAttribute("ScaleZ",StringConverter::toString(gz).c_str());
//Guarda la orientacion del cubo inicial
Ogre::Quaternion gq3 = mSceneMgr->getSceneNode("CuboRoot")-
>getOrientation();
gx = gq3.x; gy = gq3.y; gz = gq3.z; Real gw = gq3.w;
eCuboRoot-
>SetAttribute("OrientationX",StringConverter::toString(gx).c_str());
eCuboRoot-
>SetAttribute("OrientationY",StringConverter::toString(gy).c_str());
eCuboRoot-
>SetAttribute("OrientationZ",StringConverter::toString(gz).c_str());
eCuboRoot-
>SetAttribute("OrientationW",StringConverter::toString(gw).c_str());
```

- De la malla de booleanos se guardan los cubos que caben en la malla, el tamaño de la tabla de booleanos y sus valores en la tabla. Esto se puede ver en el siguiente código.

```
//Guarda la malla virtual de booleanos
TiXmlElement * eMallaInicial = new TiXmlElement( "MallaInicial" );
root->LinkEndChild( eMallaInicial );
eMallaInicial->SetAttribute("TMallaInicial",TamañoMallaInicial);
TempMallaNueva = CalculoMalla (mSceneMgr-
>getSceneNode("ObjetoRoot")->_getWorldAABB(),
mSceneMgr->getSceneNode("CuboNodo0")->getAttachedObject("Cubo0")-
>getBoundingBox(),
mSceneMgr->getSceneNode("CuboRoot")->getScale());
gx = TempMallaNueva.x; gy = TempMallaNueva.y; gz = TempMallaNueva.z;
eMallaInicial-
>SetAttribute("TMallaX",StringConverter::toString(gx).c_str());
eMallaInicial-
>SetAttribute("TMallaY",StringConverter::toString(gy).c_str());
eMallaInicial-
>SetAttribute("TMallaZ",StringConverter::toString(gz).c_str());

for (int i = 0; i <= TamañoMallaInicial; i++)
      {
      char test[50];
      strcpy(test,"P");
      strcat(test, StringConverter::toString(i).c_str());
      eMallaInicial->SetAttribute(test, MallaInicial[i]);
      }
```

9.6 Cargar el policubo

Para cargar el policubo se hace mediante el evento explicado anteriormente en el apartado 9.1, concretamente con el siguiente código:

```
//Boton Cargar
wmgr.getWindow((CEGUI::utf8*)"PolyCubo/Cubo/Cargar")
->subscribeEvent(CEGUI::PushButton::EventClicked,
CEGUI::Event::Subscriber(&PFCApplication::handleCargar, this));
```

Este evento se encargará de llamar al método handleCargar y este avisa al PFCListener de que se tiene que ejecutar el método ExploradorWindowsCargar(), que abrirá una ventana del explorador de windows y obtendrá el nombre del fichero XML. El código de abrir el explorador del windows se puede ver en el anexo. Para cargar los valores primero se limpia la escena y se inicializan las variables a 0, ver en el siguiente código:

```
//Limpia la escena
mSceneMgr->clearScene();
//Inicializacion de variables
Cubos.clear();
NodoCubos.clear();
CubosNew.clear();
NodoCubosNew.clear();
indice = 0;
```

Una vez inicializada las variables se cargan las luces y el modelo base. El siguiente paso es recuperar la información del fichero XML, el proceso es básicamente el mismo que de guardar, sólo que ahora se recupera la información. En el siguiente código se ve un ejemplo de cómo se recupera la información de una variable.

```
//Carga la posicion del Objeto
TiXmlElement* child = pElem->FirstChildElement("ObjetoRoot");

temp = child->Attribute("PositionX");
cv3.x = atof(temp.c_str());
temp = child->Attribute("PositionY");
cv3.y = atof(temp.c_str());
temp = child->Attribute("PositionZ");
cv3.z = atof(temp.c_str());
ObjetoRoot->setPosition(cv3);
```

Una vez recuperada toda la información, sólo queda llamar al método CargaTablaBoleanos para que reconstruya el policubo. Esto se muestra en el siguiente código.

```
CargaTablaBoleanos (MallaInicial, TempMalla, mSceneMgr-
>getSceneNode("CuboRoot")->getScale());
```

9.7 Cambiar transparencia del policubo y del Modelo base.

Para asignar transparencia al policubo y el modelo base, se hace mediante el evento explicado anteriormente en el apartado 9.1, concretamente con el siguiente código:

```
//ScrollBars
for (int i = 0; i < SI_COUNT; ++i)
      {
      scrollbars[i] = static_cast<CEGUI::Scrollbar*>(
      wmgr.getWindow(scrollbarNames[i]));
      scrollbars[i]->subscribeEvent(
      CEGUI::Scrollbar::EventScrollPositionChanged,
      CEGUI::Event::Subscriber(&PFCApplication::handleScrollChanged,
      this));
      }
```

Este evento se encargará de llamar al método handleScrollChanged y este avisa al PFCListener que se tiene que ejecutar el método ScrollTamañoCubo(float posición, int

i), donde la posición es un valor de 0 a 1 que corresponde a la posición de la barra y i es el identificador de la barra que se ha desplazado.

El siguiente paso es obtener el material del policubo o modelo base y aplicarle el cambio. Seguidamente podemos ver el código para aplicar la transparencia al policubo.

```
MaterialPtr MaterialCubo =
MaterialManager::getSingleton().getByName("PFC2/Cubo");
MaterialCubo->getTechnique(0)->getPass(0)-
>setDepthWriteEnabled(false);
ColourValue CuboColor = MaterialCubo->getTechnique(0)->getPass(0)-
>getDiffuse();
CuboColor.a = 1 - Posicion;
MaterialCubo->getTechnique(0)->getPass(0)-
>setSceneBlending(SBT_TRANSPARENT_ALPHA);
MaterialCubo->getTechnique(0)->getPass(0)->setDiffuse(CuboColor);
```

En el anexo se puede ver los materiales creados para el proyecto.

10 Resultados

Se han realizado todos los apartados planificados inicialmente. A continuación se pueden ver algunas capturas del trabajo realizado.

10.1 Capturas de la herramienta

A continuación se muestran varias capturas de la herramienta realizada:

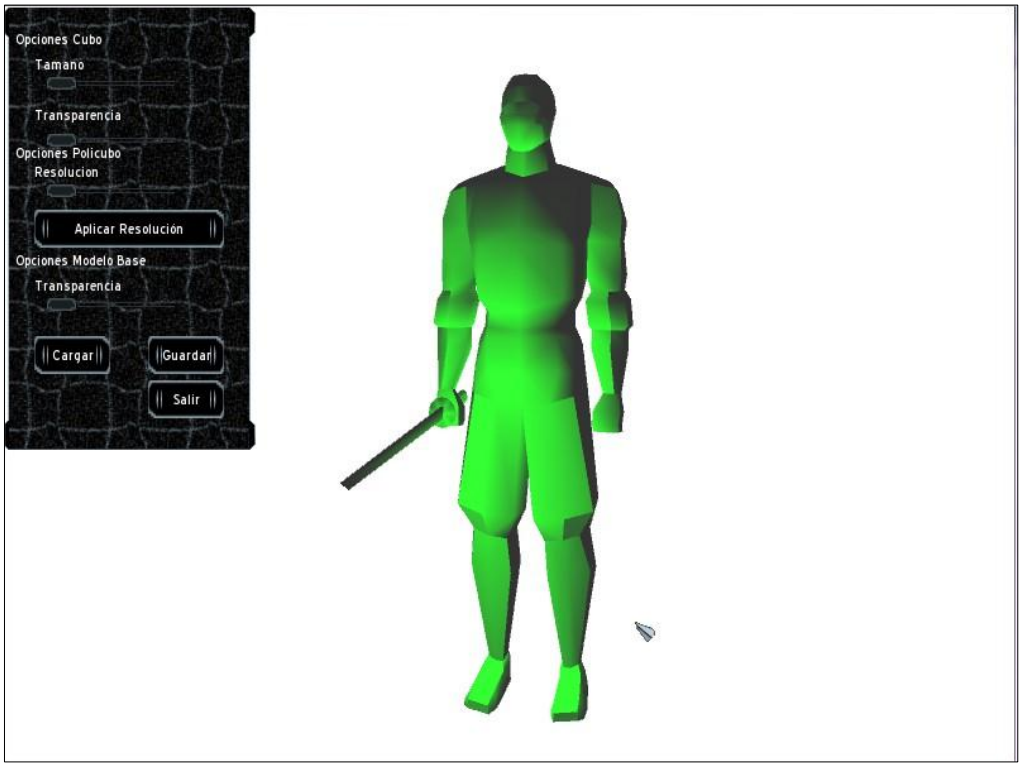


Figura 40 Captura de la herramienta

En la figura anterior se puede ver el aspecto general de la herramienta, con la mesh del ninja cargado.

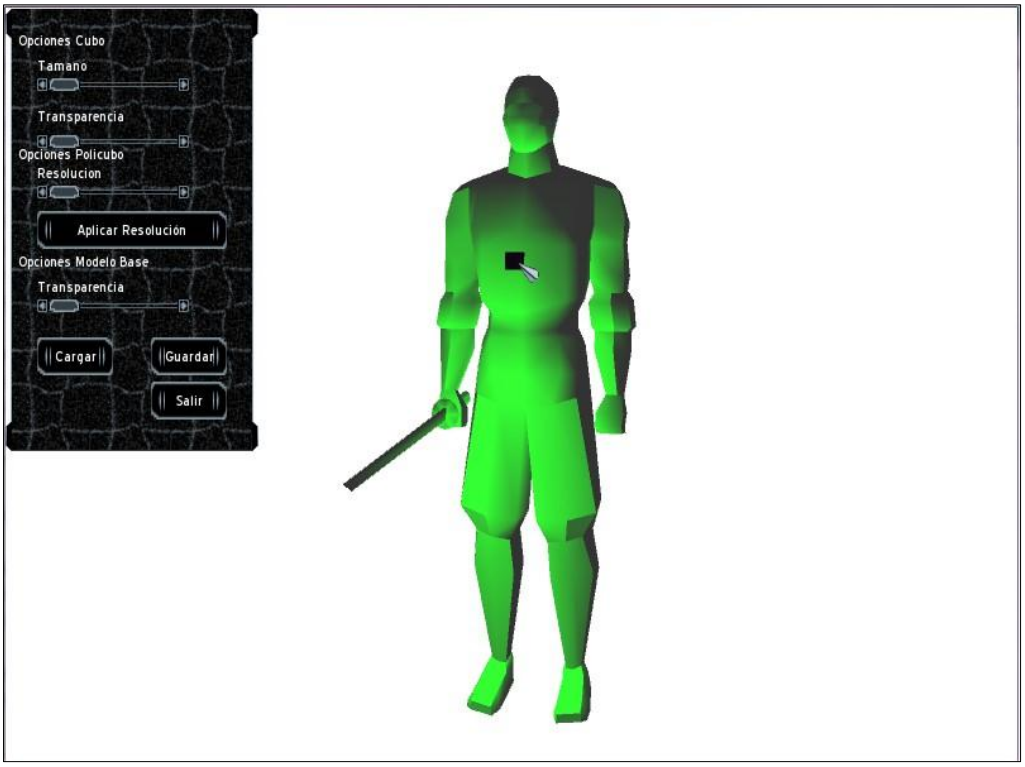


Figura 41 Captura de la inserción del primer cubo

En la figura anterior se puede ver la inserción del primer cubo (cubo root).

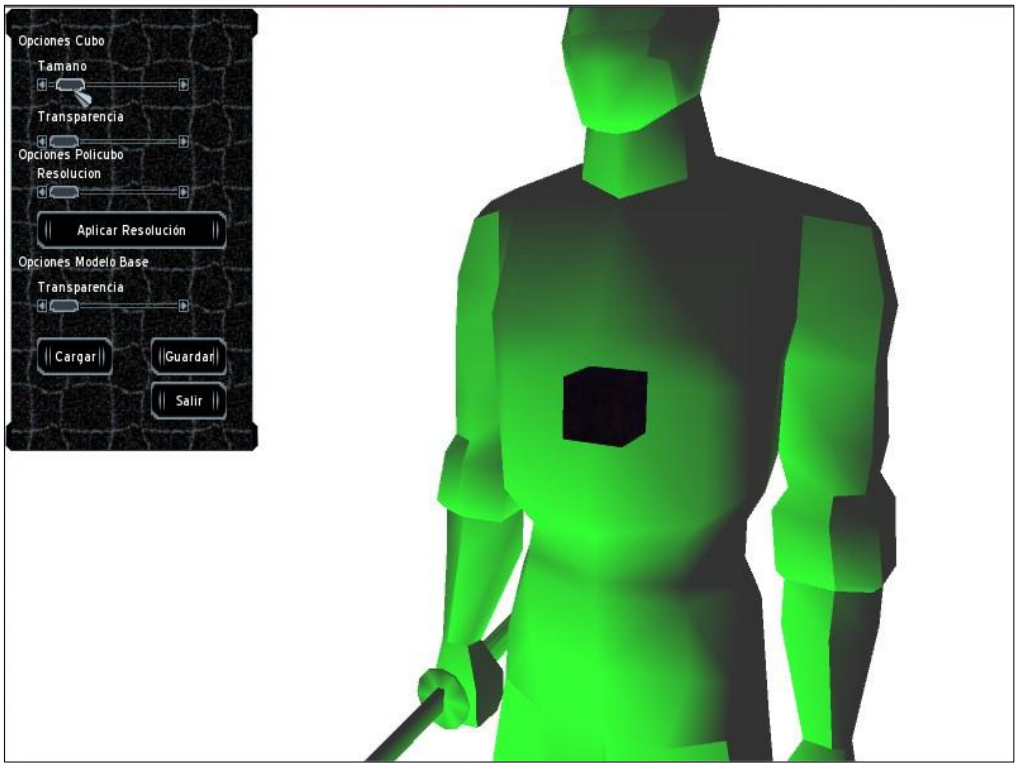


Figura 42 Captura de haciendo un cambio de tamaño al cubo inicial

En la figura anterior se puede ver como se cambia el tamaño del primer cubo (cubo root).

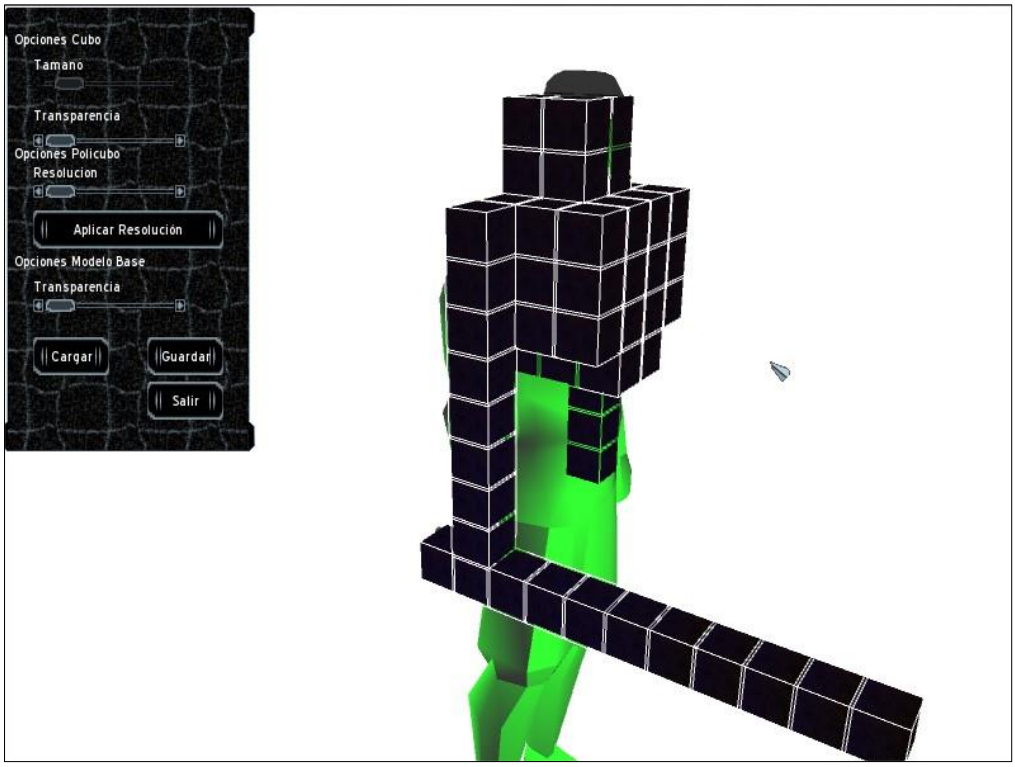


Figura 43 Captura de un ejemplo de policubo con modelo base de un ninja

En la figura anterior se puede ver como va tomando forma el policubo gracias a la herramienta.

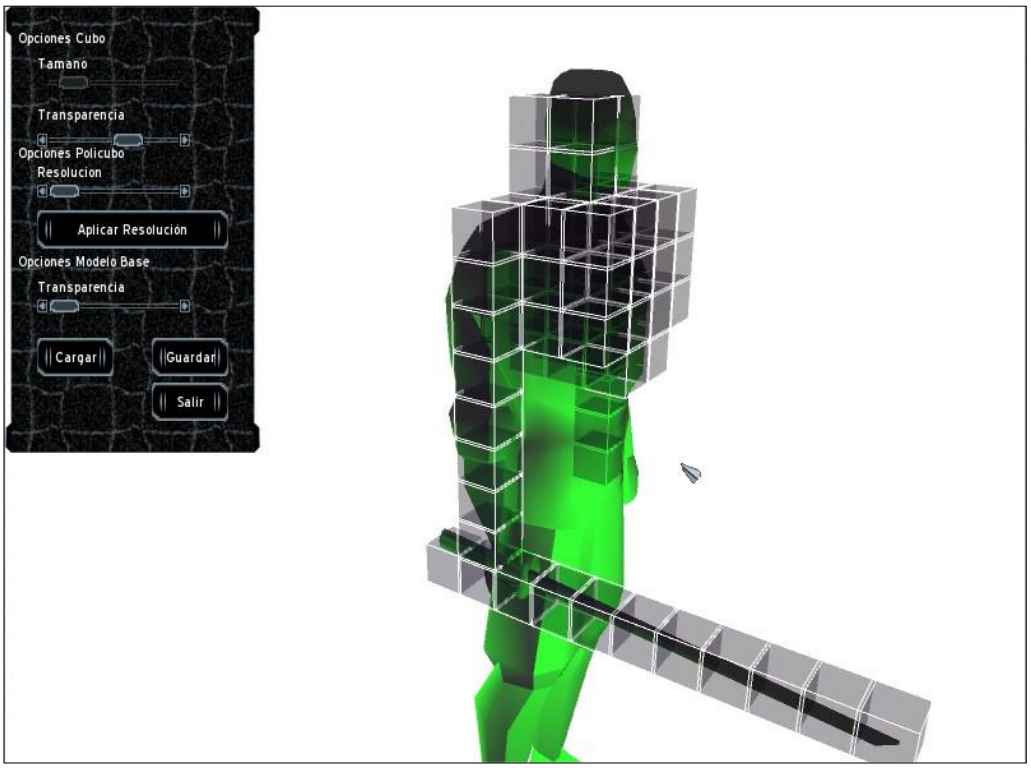


Figura 44 Captura de un cambio de transparencia al policubo

En la figura anterior se puede ver como se aplica transparencia al policubo.

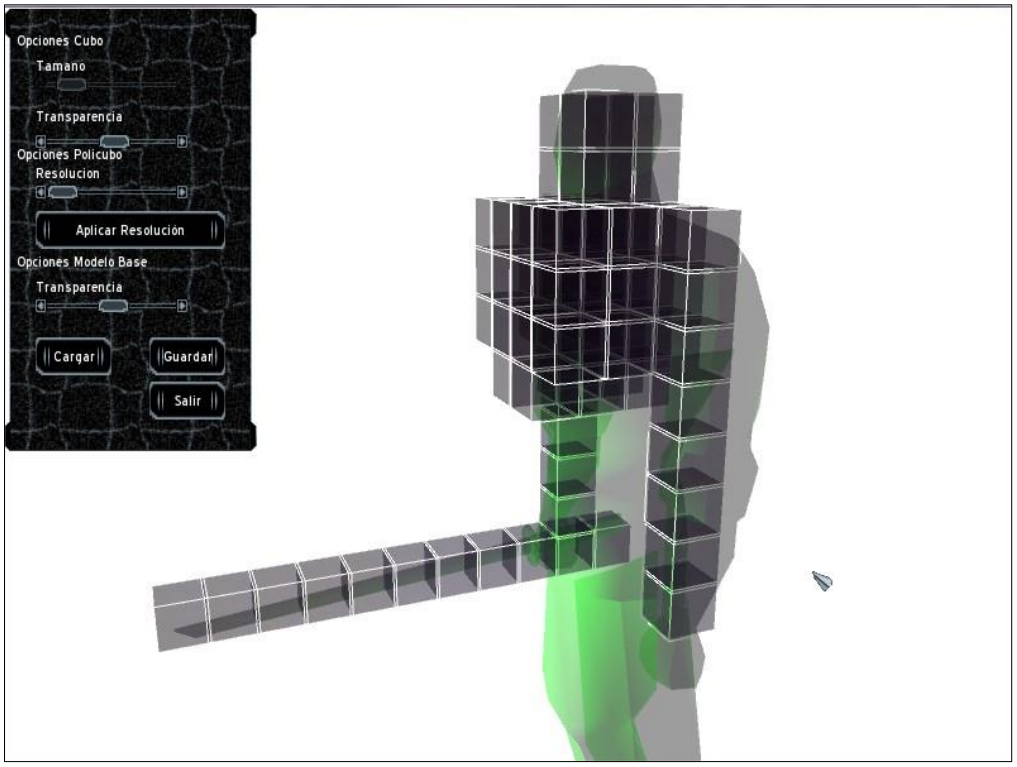


Figura 45 Captura de un cambio de transparencia al modelo base

En la figura anterior se puede observar como se le aplica transparencia al modelo base.

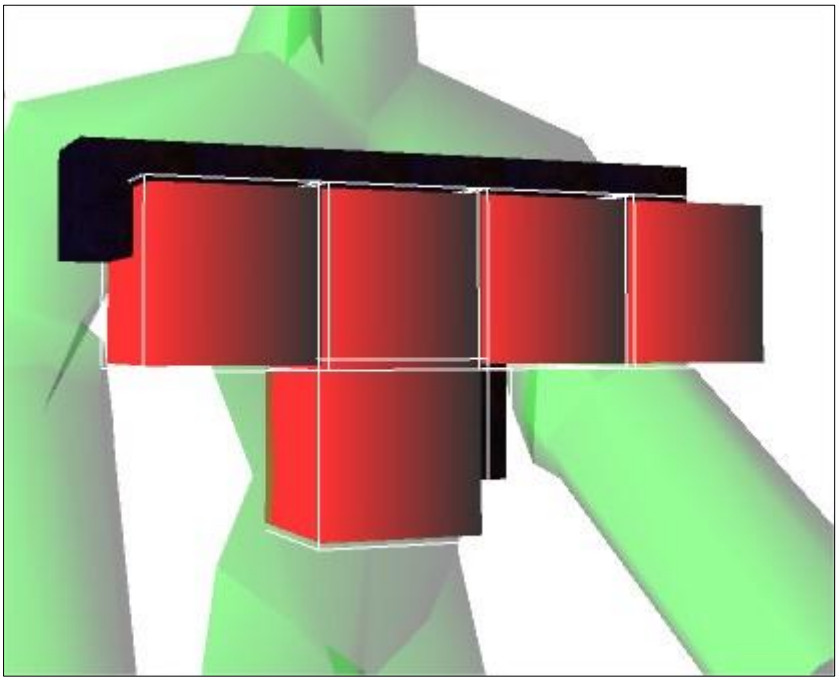


Figura 46 Captura de un cambio de resolución del policubo

En la figura anterior se puede observar en color negro el policubo creado por el usuario y en rojo el policubo resultante creado por la herramienta al aplicar la resolución.

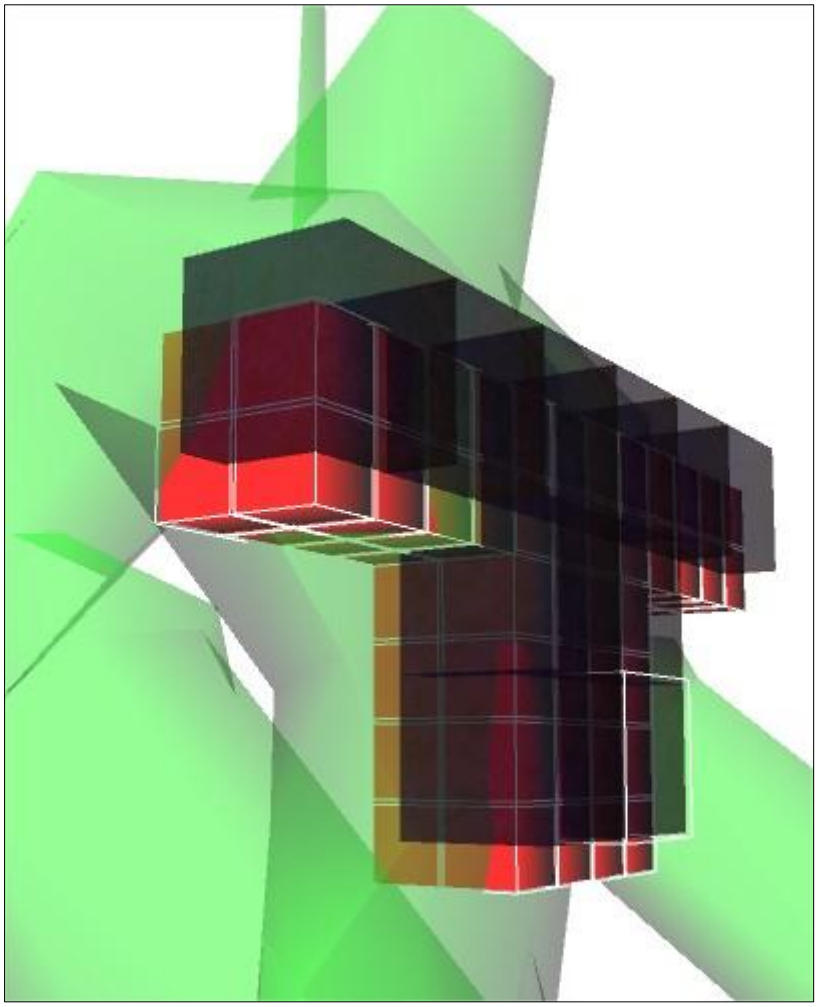


Figura 47 Captura de un cambio de resolución del policubo

En la figura anterior se puede ver otro ejemplo de un cambio de resolución al policubo.

10.2 Distribución

En cuanto a la distribución, se han considerado la página de SourceForce.

SourceForge es un sitio web de colaboración para proyectos de software. SourceForge es comercializado por VA Software. Provee una portada para un amplio rango de servicios útiles para los proceso de desarrollo de software e integra un amplio número de aplicaciones de software libre (tales como PostgreSQL y CVS).

SourceForge es una central de desarrollos de software que controla y gestiona varios proyectos de software libre y actúa como un repositorio de código fuente. SourceForge.net es hospedado por VA Software y corre en una versión del software SourceForge.

Ofrece alojamiento a proyectos tales como Ares Galaxy, FileZilla, 7-Zip, phpMyAdmin, etc.

11 Conclusiones

11.1 Temporización

Pocas veces la temporización planificada inicialmente en un proyecto corresponde fielmente con lo que se obtiene en la realidad. La temporización real del tiempo dedicado es tal como muestra la Figura 48 siguiente.

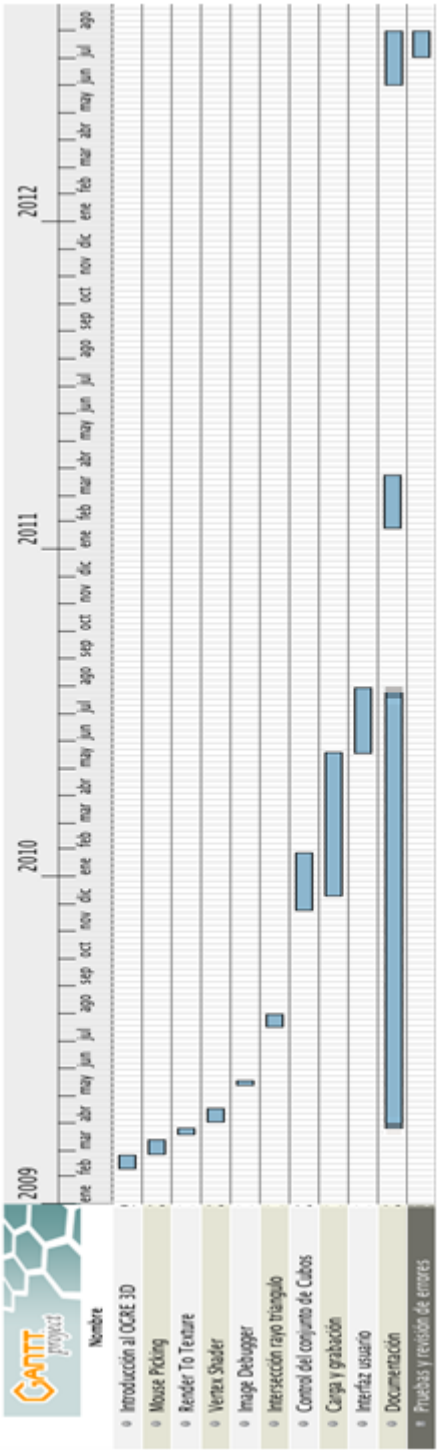


Figura 48 Temporización real del proyecto

El motivo principal por el cual se alargó el proyecto fue por el comienzo de un trabajo y poco a poco fue quedando apartado del día a día.

11.2 Cambios realizados

Gracias a la planificación con la ayuda del director y tutor no se han tenido que realizar cambios en la realización del proyecto, desde el primer momento quedo muy claro la meta de lo que se pretendía hacer y con la ayuda del director/tutor se ha llevado a cabo.

11.3 Conclusiones

Durante la elaboración del proyecto final de carrera se han llegado a varias conclusiones de las cuales destaco:

- A lo hora de hacer el proyecto es fundamental tener un buena planificación.
- Hoy en día existen numerosas librerías que facilitan mucho el trabajo a realizar, y además mejoran el rendimiento.
- El tener acceso a internet es fundamental ya que se puede obtener mucha información y además rápidamente.
- El objetivo inicial de crear la herramienta ha sido completado, pero al mismo tiempo se abre una nueva vía en el que se pueden aplicar mejoras tanto de funcionalidad como rendimiento.

12 Trabajo futuro

Hay que saber elegir hasta dónde se quiere llegar con un proyecto de este tipo, ya que siempre se puede ampliar más y más, aunque sí que es cierto que hay aspectos que me gustaría mejorar:

- Agregar un botón para cargar diferentes modelos base.
- Agregar la funcionalidad de hacer y deshacer
- Poder exportar el policubo a una mesh de Ogre u otro formato (3dsmax, maya,…)
- Mejorar la interfaz, como por ejemplo poner menú despegable.
- Mejorar el rendimiento de herramienta.

13 Bibliografía

Libros:

Junker, Gregory. *Pro Ogre 3D Programming*. *Expert's Voice in Open Source*. Apress, 2006.

Paginas Web:

Ogre3D Wiki Basic Tutorial 1: An introduction to the most basic Ogre constructs: SceneManager, SceneNode, and Entity objects. Ogre Team. 2001. Ogre3D. Febrero 2009. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+1&structure=Tutorials

Ogre3D Wiki Basic Tutorial 2: Cameras, Lights, and Shadows. Ogre Team. 2001. Ogre3D. Febrero 2009. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+2&structure=Tutorials

Ogre3D Wiki Basic Tutorial 3: CEGUI and Ogre. Ogre Team. 2001. Ogre3D. Agosto 2012. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+7&structure=Tutorials

Ogre3D Wiki Intermediate Tutorial 3: Mouse Picking (3D Object Selection). Ogre Team. 2001. Ogre3D. Junio 2010. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Intermediate+Tutorial+3&structure=Tutorials

Ogre3D Intermediate Tutorial 7: Render to texture (RTT). Ogre Team. 2001. Ogre3D. Marzo 2009. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+7&structure=Tutorials

Ogre3D Intermediate Tutorial 7: Render to texture (RTT). Ogre Team. 2001. Ogre3D. Marzo 2009. http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Basic+Tutorial+7&structure=Tutorials

Ogre3D API Reference Start Page. Ogre Team. 2001. Ogre3D. Marzo 2009. http://www.udg.edu/LaBibliotecaforma/Comcitardocuments/Comcitarpaginesweb/tabid/11969/language/ca-ES/Default.aspx

Inmensia Ray Tracing: Plano, Triángulo y Cubo. Juan Mellado. 3 Noviembre 2005. Inmensia. Julio 2010. http://www.inmensia.com/articulos/raytracing/planotrianguloycubo.html?pag=2

PolyCubeMaps. Marco Tarini. 11 setiembre 2004. Istituto di Scienza e Tecnologie dell'Informazione. Febrero 2009. http://vcg.isti.cnr.it/polycubemaps/

14 Anexos

14.1 PolyCube.layout

```
<?xml version="1.0" ?>
<GUILayout>
<Window Type="DefaultGUISheet" Name="root">
      <Window Type="DefaultGUISheet" Name="PolyCubo3D">
      <Property Name="UnifiedSize" Value="{{0.25,0},{1,0}}" />

      <Window Type="TaharezLook/FrameWindow" Name="PolyCubo/Cubo">

      <Property Name="UnifiedSize" Value="{{0,200},{0,350}}" />
      <Property Name="UnifiedPosition" Value="{{0,0},{0,0.52}}" />
      <Property Name="TitlebarEnabled" Value="false" />
      <Property Name="CloseButtonEnabled" Value="false" />
      <Property Name="Text" Value="Interfaz Usuario" />

      <Window Type="TaharezLook/StaticText"
Name="PolyCubo/Cubo/Cubo">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,10},{0,5}}" />
      <Property Name="UnifiedSize" Value="{{0,150},{0,40}}" />
      <Property Name="Text" Value="Opciones Cubo" />
      </Window>

      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/Tamano">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,25},{0,25}}" />
      <Property Name="UnifiedSize" Value="{{0,60},{0,40}}" />
      <Property Name="Text" Value="Tamano" />
      </Window>
      <Window Type="TaharezLook/HorizontalScrollbar"
      Name="PolyCubo/Cubo/Tamano_Scroll">
      <Property Name="UnifiedPosition" Value="{{0,25},{0,55}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,10}}" />
      <Property Name="DocumentSize" Value="1.1" />
      <Property Name="PageSize" Value="0.1" />
      <Property Name="StepSize" Value="0.005" />
      <Property Name="OverlapSize" Value="0.0" />
      </Window>

      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/Transparencia">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,25},{0,65}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,40}}" />
      <Property Name="Text" Value="Transparencia" />
      </Window>

      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/OpcionesPolicubo">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
```

```
<Property Name="UnifiedPosition" Value="{{0,10},{0,95}}" />
      <Property Name="UnifiedSize" Value="{{0,150},{0,40}}" />
      <Property Name="Text" Value="Opciones Policubo" />
      </Window>

      <Window Type="TaharezLook/HorizontalScrollbar"
      Name="PolyCubo/Cubo/Transparencia_Scroll">
      <Property Name="UnifiedPosition" Value="{{0,25},{0,100}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,10}}" />
      <Property Name="DocumentSize" Value="1.1" />
      <Property Name="PageSize" Value="0.1" />
      <Property Name="StepSize" Value="0.05" />
      <Property Name="OverlapSize" Value="0.05" />
      </Window>

      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/Resulucion">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,25},{0,110}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,40}}" />
      <Property Name="Text" Value="Resolucion" />
      </Window>
      <Window Type="TaharezLook/HorizontalScrollbar"
      Name="PolyCubo/Cubo/Resulucion_Scroll">
      <Property Name="UnifiedPosition" Value="{{0,25},{0,140}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,10}}" />
      <Property Name="DocumentSize" Value="1.1" />
      <Property Name="PageSize" Value="0.1" />
      <Property Name="StepSize" Value="0.05" />
      <Property Name="OverlapSize" Value="0.05" />
      </Window>

      <Window Type="TaharezLook/Button"
      Name="PolyCubo/Cubo/AplicarResolucion">
      <Property Name="ID" Value="12" />
      <Property Name="UnifiedSize" Value="{{0,150},{0,30}}" />
      <Property Name="UnifiedPosition" Value="{{0,25},{0,160}}" />
      <Property Name="Text" Value="Aplicar Resolución" />
      </Window>

      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/Objeto">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,10},{0,180}}" />
      <Property Name="UnifiedSize" Value="{{0,150},{0,40}}" />
      <Property Name="Text" Value="Opciones Modelo Base" />
      </Window>


      <Window Type="TaharezLook/StaticText"
      Name="PolyCubo/Cubo/ObjetoTransparencia">
      <Property Name="FrameEnabled" Value="False"/>
      <Property Name="BackgroundEnabled" Value="False"/>
      <Property Name="Font" Value="BlueHighway-12"/>
      <Property Name="UnifiedPosition" Value="{{0,25},{0,200}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,40}}" />
      <Property Name="Text" Value="Transparencia" />
      </Window>
```

```
<Window Type="TaharezLook/HorizontalScrollbar"
Name="PolyCubo/Cubo/ObjetoTransparencia_Scroll">
      <Property Name="UnifiedPosition" Value="{{0,25},{0,230}}" />
      <Property Name="UnifiedSize" Value="{{0,120},{0,10}}" />
      <Property Name="DocumentSize" Value="1.1" />
      <Property Name="PageSize" Value="0.1" />
      <Property Name="StepSize" Value="0.05" />
      <Property Name="OverlapSize" Value="0.05" />
      </Window>

      <Window Type="TaharezLook/Button" Name="PolyCubo/Cubo/Cargar">
      <Property Name="ID" Value="12" />
      <Property Name="UnifiedSize" Value="{{0,60},{0,30}}" />
      <Property Name="UnifiedPosition" Value="{{0,25},{0,260}}" />
      <Property Name="Text" Value="Cargar" />
      </Window>

      <Window Type="TaharezLook/Button"
Name="PolyCubo/Cubo/Guardar">
      <Property Name="ID" Value="12" />
      <Property Name="UnifiedSize" Value="{{0,60},{0,30}}" />
      <Property Name="UnifiedPosition" Value="{{0,115},{0,260}}" />
      <Property Name="Text" Value="Guardar" />
      </Window>

      <Window Type="TaharezLook/Button" Name="PolyCubo/Cubo/Salir">
      <Property Name="ID" Value="12" />
      <Property Name="UnifiedSize" Value="{{0,60},{0,30}}" />
      <Property Name="UnifiedPosition" Value="{{0,115},{0,295}}" />
      <Property Name="Text" Value="Salir" />
      </Window>
      </Window>
      </Window>
</Window>
</GUILayout>
```

14.2 Código ExploradorWindowsCargar

```
//Abre un explorador de Windows y obtiene el nombre del archivo que
seleccionamos
void ExploradorWindowsCargar()
{
      OPENFILENAME fname;
      ZeroMemory(&fname, sizeof(fname));
      char strfile[200] = "*.xml";
      char szFilter[] = TEXT ("XML Files (*.XML)\0*.xml\0")  \
      TEXT ("All Files (*.*)\0*.*\0\0") ;

      fname.lStructSize = sizeof(OPENFILENAME);
      fname.hwndOwner = NULL;
      fname.hInstance = NULL;
      fname.lpstrFilter = szFilter;
      fname.lpstrCustomFilter = NULL;
      fname.nFilterIndex = 0;
      fname.nMaxCustFilter = 0;
      fname.lpstrFile = strfile;
      fname.nMaxFile = 200;
      fname.lpstrFileTitle = NULL;
```

```
        fname.nMaxFileTitle = 0;
        fname.lpstrTitle = NULL;
        fname.Flags = OFN_HIDEREADONLY | OFN_CREATEPROMPT;
        fname.nFileOffset = 0;
        fname.nFileExtension = 0;
        fname.lpstrDefExt = 0;
        fname.lCustData  = NULL;
        fname.lpfnHook = NULL;
        fname.lpTemplateName  = NULL;
        fname.lpstrInitialDir = NULL;
        GetOpenFileName(&fname);
        }
```

14.3 Código ExploradorWindowsGuardar

```
void ExploradorWindowsGuardar()
        {
        OPENFILENAME fname;
        ZeroMemory(&fname, sizeof(fname));
        char strfile[200] = "*.xml";
        char szFilter[] = TEXT ("XML Files (*.XML)\0*.xml\0")  \
        TEXT ("All Files (*.*)\0*.*\0\0") ;
        fname.lStructSize = sizeof(OPENFILENAME);
        fname.hwndOwner = NULL;
        fname.hInstance = NULL;
        fname.lpstrFilter = szFilter;
        fname.lpstrCustomFilter = NULL;
        fname.nFilterIndex = 0;
        fname.nMaxCustFilter = 0;
        fname.lpstrFile = strfile;
        fname.nMaxFile = 200;
        fname.lpstrFileTitle = NULL;
        fname.nMaxFileTitle = 0;
        fname.lpstrTitle = NULL;
        fname.Flags = OFN_HIDEREADONLY | OFN_CREATEPROMPT;
        fname.nFileOffset = 0;
        fname.nFileExtension = 0;
        fname.lpstrDefExt = 0;
        fname.lCustData  = NULL;
        fname.lpfnHook = NULL;
        fname.lpTemplateName  = NULL;
        fname.lpstrInitialDir = NULL;
        GetSaveFileName(&fname);
        guardar(fname.lpstrFile);
        }
```

14.4 PFC2.material

```
material PFC2/Objeto
{
	technique
	{
		pass
		{
			diffuse 0.0 1.0 0.0 1.0
		}
	}
}

material PFC2/Cubo
{
	technique
	{
		pass
		{
			ambient 1.0 1.0 1.0
			diffuse 0.0 0.0 1.0 1.0
			//polygon_mode wireframe
			texture_unit
			{
				texture RustySteel.jpg
			}
		}
	}
}

material PFC2/Cubo2
{
	technique
	{
		pass
		{
			depth_write on
			ambient 1.0 1.0 1.0
			diffuse 1.0 0.0 0.0 1.0
		}
	}
}
```

15 Agradecimientos

Para acabar, hace falta mencionar a dos grandes personas que me han dado todo su apoyo y ayuda para poder completar el proyecto, mi director de proyecto Ismael García y tutor Gustavo Patow. También quiero dar gracias a todas las personas que me han dado apoyo (familia, amigos y pareja).