DESENVOLUPAMENT D'UN VIDEOJOC EN 3D

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
EDUARD RICART LÓPEZ
i dirigit per
XAVIER ORIOLS PLADEVALL
Bellaterra, 18 de JUNY de 2009

Índex

1 Introducció

Aquest projecte fi de carrera de la titulació d'Enginyeria Informàtica tracta sobre el disseny, la programació i la realització d'un videojoc. És un projecte diferent del que és habitual, ja que es tracte d'un projecte auto-proposat per mi mateix. El motiu per el qual he escollit fer un videojoc com a treball fi de carrera de la titulació de Enginyeria Informàtica es que ja des de petit m'agradaven i fascinaven molt conèixer la estructura i funcionament dels videojocs. De fet uns dels motius per els quals vaig escollir estudiar la carrera d'Enginyer Informàtic Superior va ser el poder veure com funcionen els videojocs amb tot detall. En aquest sentit, en l'assignatura Gràfics per Computador II, vaig poder experimentar les bases per poder crear un videojoc a nivell de programació i més endavant vaig realitzar un altre assignatura sobre com començar a dissenyar i programar un videojoc en 3D. Per tant, la possibilitat de realitzar un videojoc i poder donar utilitat a les matèries que he après a la carrera ha estat una motivació molt especial per la realització d'aquest projecte de fi de carrera.

Importància del la industria dels videojocs en la nostre societat actual

Els videojocs cada vegada tenen més importància en la nostre activitat d'oci actual, i per tant adquireixen una importància creixent dins dels camps de treball de la Informàtica. Anem a veure exemples de la importància que esta adquirint la industria dels videojocs en la nostre societat actual. Per exemple, l'ultima pel·lícula de "Piratas del Caribe" va ser la pel·lícula que més va recaptar durant l'any 2007 amb un volum de facturació de 406 milions de dollars. Sorprenentment, aquell mateix any, el videojoc conegut com **Grand Theft Auto IV (GTA IV)** desenvolupat per ordinadors i diverses consoles va facturar 500 milions de dollars.

Un altre exemple el trobem en la industria dels videojocs en els Estats Units. Durant el mes de juny de 2007 l'industria dels videojocs va registrar vendes per més de 1100

milions de dollars només dintre dels Estats Units. En aquest sentit, el nombre de llars dels Estats Units que ha adquirit una consola de Videojocs ha augmentat un 18% segons un estudi realitzat a finals de 2006, 46 milions de llars de EU tenen una consola de videojocs.

Segons càlculs de experts en aquest àmbit, la industria dels videojocs mou al voltant de 30,000 milions de dòlars durant l'últim any en tot el mon i el volum de diners involucrats no para de créixer cada any.

En aquest mateix sentit, la revista espanyola "Consumer" prediu que durant l'any 2010 el valor de la industria de videojocs creixerà un 100 %, és a dir 60 mil milions de dòlars aproximadament. Per tal de tenir una idea de l'impacta d'aquestes xifres, podem mencionar que, per exemple, (i) la industria Pornogràfica registra ventes mundials de 57 mil milions de dòlars, (ii) Hollywood produeix al voltant de 500 films al any i guanya 9 mil milions de dòlars, o bé, (iii) Microsoft ven anualment 40 Mil milions de dòlars en Software.

Fins ara hem descrit la importància de la industria dels videojocs a nivell global, si ens centrem dins del territori espanyol, la situació també es d'una importància creixent d'aquest sector. Veure figura 1.

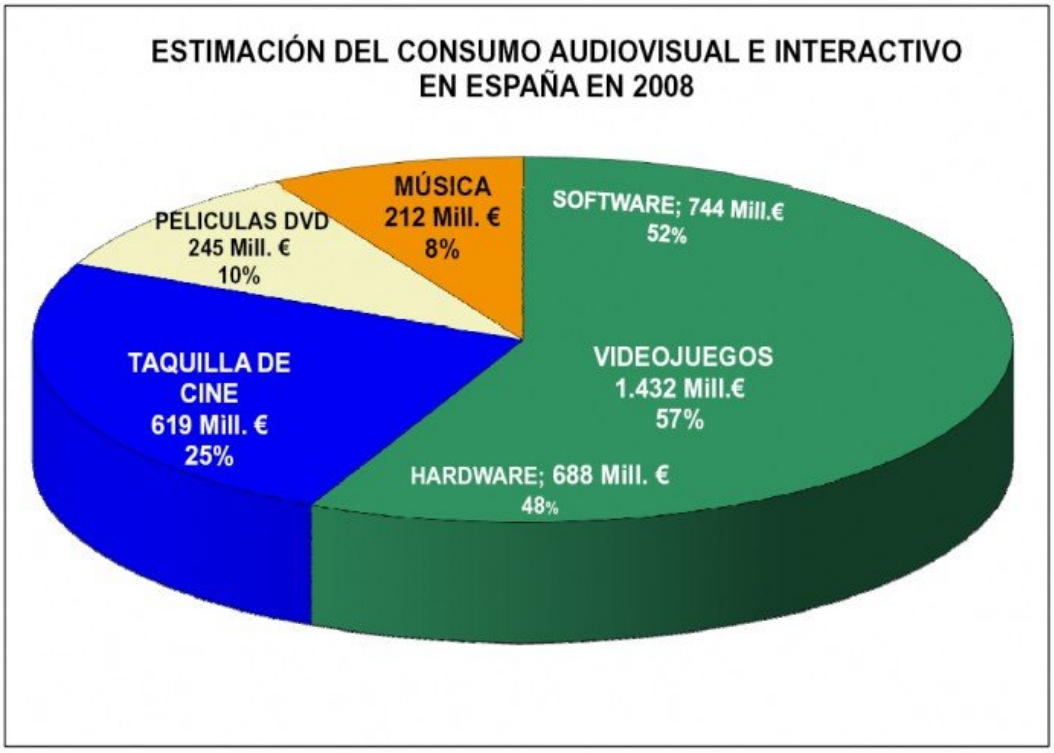


Figura 1: Estimació del consum audiovisual e interactiu a Espanya durant l'any 2008

Durant l'any 2008, com poder veure a la *Figura 1,* els videojocs s'emporten gairebé el 60% de les vendes del consum audiovisual e interactiu del país. Més del doble que el cinema, més de 5 vegades el consum de musica i mes de 7 vegades les pel·lícules en DVD.

Per tant, com a conclusió, podem dir que la industria de videojocs avui en dia és una de els més importants dins de l'industria del oci i el volum de recurs que mou augmenta espectacularment cada any que passa. En aquets sentit, la programació de videojocs que s'ha plantejat en aquest projecte fi de carrera es una activitat d'una importància creixent dins de l'àmbit de la informàtica i una bona font de activitat professional degut a la alta demanda, per part del públic, d'aquest sector en tot el mon.

2 Objectius

Després de la introducció del anterior capítol on hem explicat la motivació especial d'aquest projecte fi de carrera, a continuació, passem a descriure els objectius concrets d'aquest projecte fi de carrera.

El principal objectiu, que persegueix aquest projecte fi de carrera és el següent:

- **Dissenyar la lògica i desenvolupar el software d'un videojoc en 3D per Ordinador.**

Com ja s'ha comentat en la introducció, durant la carrera com a Enginyer Informàtic, s'han tractat alguns aspectes relacionats amb la programació i disseny d'un videojoc, i la possibilitat de realitzar un projecte de fi de carrera en aquest àmbit m'ofereix la possibilitat d'aprofundir en aquesta matèria.

Com s'explicarà en el tercer capítol, l'entorn de programació per a realitzar aquest projecte fi de carrera es el llenguatge C++ en un entorn de Microsoft Visual Studio 2008 Express, les llibreries de Ogre3D i el programa Blender per a realitzar objectes i textures 3D.

He escollit Microsoft Visual Studio 2008 Express com a entorn de programació ja que les seves propietats són les necessàries per poder realitzar un videojoc amb les característiques que m'he plantejat i a més la seva llicencia es gratuïta.

Per altre banda, he escollit C++ com a llenguatge de programació ja que segons l'experiència que tinc la majoria de jocs del mercat estan realitzar amb aquets llenguatge, a mes és un dels llenguatges que mes he vist durant la carrera i amb el que hem sento més còmode treballant.

He escollit Ogre3D com a motor gràfic (llibreries gràfiques per poder desenvolupar un videojoc en 3D) ja que és més potent que altres motors com OpenGL i DirectX ja que els inclou al dos i alhora abstrau les seves dificultats a més a més la seva llicència és gratuïta i està escrit en C++. Per una altre banda és un dels motor gràfics més usats avui en dia, per tant m'ha semblat interessant aprendre a dominar-lo de cara a un futur.

Finalment, he escollit Blender per dissenyar els models 3D del videojoc ja que també durant la carrera vaig poder realitzar un curs d'iniciació, per tant era el software de dibuix que millor conec a més la seva llicència és gratuïta.

Per tant, hi ha també una sèrie d'objectius secundaris que s'han d'assolir per tal de dissenyar el mencionat videojoc 3D. Aquest objectius secundaris del present projecte fi de carrera són:

- Dissenyar models 3D amb Blender.
- Programar el videojoc amb C++.
- Fer que C++ interpreti els models de Blender.
- Comprendre el funcionament d'Ogre3D.
- Realitzar un Moviment 3D basat en una malla virtual pròpia.
- Interactuar el codi Ogre3D amb el codi original.
- Afegir So i Música al videojoc.
- Afegir una Interfície Gràfica al videojoc.

La correcta realització del present projecte requereix l'assoliment dels mencionats objectius.

3 Desenvolupament

En aquest capítol discutirem el desenvolupament del treball realitzat per l'elaboració del videojoc 3D. Primerament discutirem el software basic utilitzat per a la realització del videojoc, posteriorment la lògica del joc i finalment la programació del mateix.

3.1 Software Bàsic

Per poder realitzar aquest videojoc en 3D es necessiten 4 paquets de software bàsic que a continuació passem a descriure:

3.1.1 Microsoft Visual Studio Express 2008

El Microsoft Visual Studio Express 2008 es l'entorn global on es desenvoluparà el gruix del codi desenvolupat en aquest projecte, ja que aquí és on es programarà tot el codi C++ i on serà compilat i executat el programa, per tal de poder "debbugar" el programa per detectar possibles errors i per veure el resultat final.

3.1.2 Llenguatge C++

Llenguatge de programació que utilitzarem per desenvolupar la major part del projecte es el C++ . El codi C++ que desenvolupem, un cop compilat i executat en l'entorn de programació mencionat anteriorment, ens servirà per unir les dades que ens donin diferents paquets de software que hem utilitzat (com per exemple, el motor gràfic, l'entorn de modelatge i el codi original).

El motiu d'escollir el C++ es basa en que es un llenguatge de programació molt potent que permet una estructuració del codi molt professional.

3.1.3 Ogre3D

El Ogre 3D (Object-Oriented Graphics Rendering Engine) és un conjunt de llibreries C++ que s'han d'incloure al entorn de programació per tal de poder tenir les bases per crear el videojoc 3D.

Figura 2: Logotip Ogre3D

En particular, l'Ogre 3D (el logotip es mostra en la *Figura 2*) és un "motor" que permet obtenir llibreries amb "subrutines" especifiques per dibuixar objectes i escenes 3D en constant moviment. El conjunt de llibreries (que anomenem "motor") està escrit en el llenguatge de programació C++. Aquestes llibreries eviten la dificultat de la utilització de capes inferiors de llibreries gràfiques com OpenGL i Direct3D, i a més, proveeixen una interfície basada en classes d'alt nivell. El motor és programari lliure, llicenciat sota LGPL i amb una comunitat de usuaris molt activa. Fins i tot, aquest mateix "motor" s'ha utilitzat professionalment en alguns jocs comercials.

3.1.4 Blender

El software Blender es l'encarregat de crear els gràfics i escenaris 3D que volem fer amb el nostre videojoc (veure un exemple del entorn del programa Blender en la *Figura 3*). Un cops creats els objectes 3D amb el programa de dibuix 3D Blender, els hi podem donar olors, forma, textures i moltes altres propietats.

Després, per tal de poder fe servir aquest objectes 3D al nostre entorn de programació, els haurem d'exportar fent us d'un exportador especial del Blender que es compatible amb OGRE3D que hem mencionat anteriorment.

El software Blender és, de fet, un programa multi plataforma, dedicat especialment al modelatge, animació i creació de gràfics tridimensionals. El programa va ser inicialment distribuït de forma gratuïta però sense el codi font, amb un manual disponible per a la venda, encara que posteriorment va passar a ser programari lliure. Actualment és compatible amb totes les versions de Windows, Mac US X, Linux, Solaris, FreeBSD i IRIX. Té una molt peculiar interfície gràfica d'usuari que permet la configuració personalitzada de la distribució dels menús i vistes de càmera.

En aquesta memòria no s'entrarà en detalls de com es realitza un model 3D a traves del a Blender, ja que no és el objectiu principal del projecte, però si que comentarem breument dues de les opcions que s'han utilitzat i que serà rellevant per entendre altres parts d'aquesta memòria:
En particular, discutirem les següents dues opcions:

- OgreMeshes.
- Textures UV Mapping.

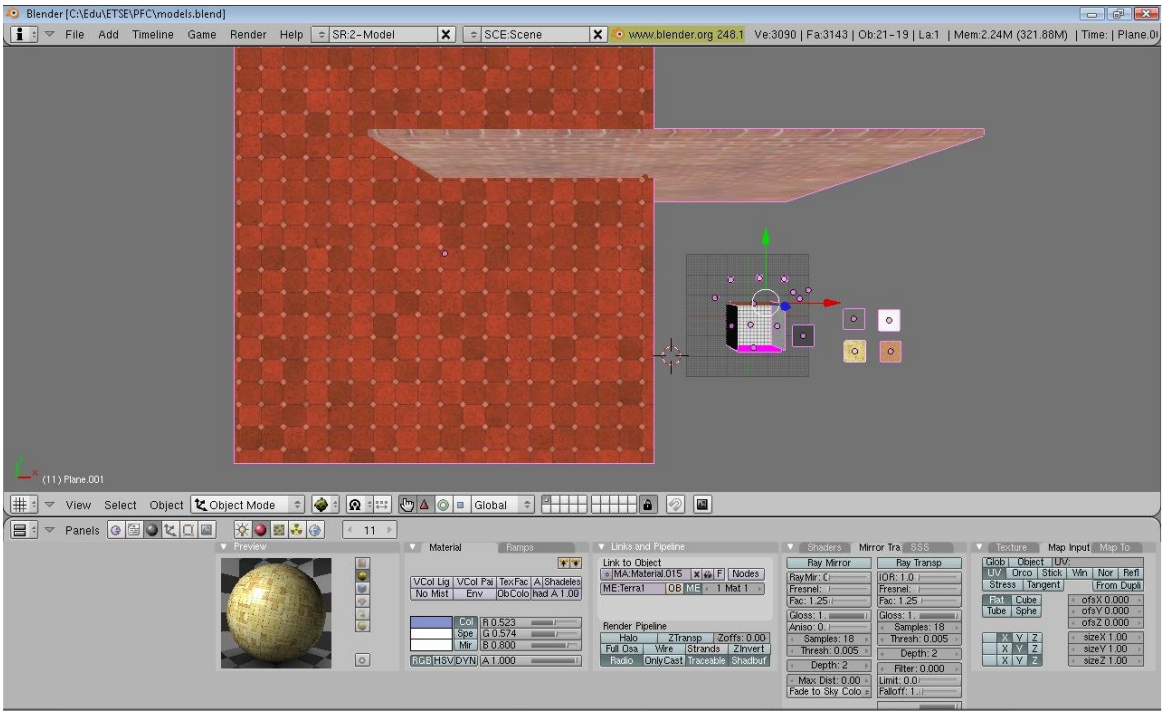


Figura 3: Models 3D en Blender

3.1.4.1 OgreMeshes

Aquesta opció del Blender és la que ens permet fer els objectes 3D compatibles amb el Ogre3D que hem comentat anteriorment. Un cop creats els models 3D amb Blender, necessitem exportar-los en un format que el nostre entorn de programació C++ pugui llegir e interpretar i que alhora siguin compatibles amb Ogre3D, en el nostre cas exportem els models 3D a fitxers .mesh. Per fer-ho farem ús d'un script ja creat anomenat OgreMeshes (*Figura 4*). Per fer servir aquest script necessitem tenir instal·lat Python amb la versió 2.6 com a mínim en el ordinador on estem executant Blender.

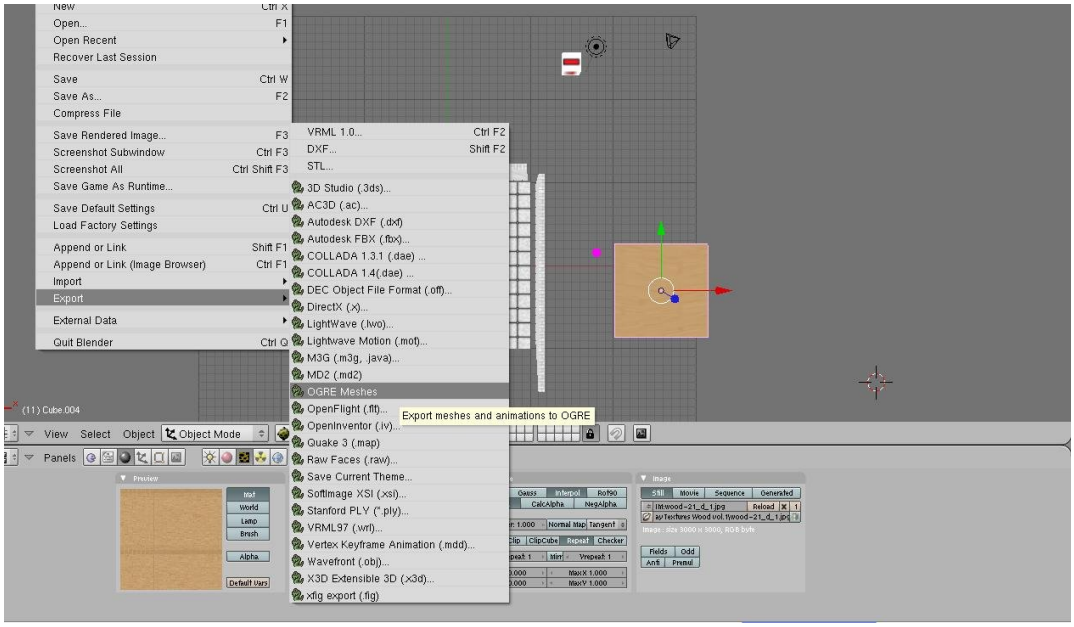


Figura 4: Blender, exportació models amb ogremeshes.

El OgreMeshes en realitat ens torna fitxers en format XML, però te l'opció de configurar un programa anomenat OgreXMLConverter que permet passar automàticament un fitxer XML a Mesh per facilitar-nos la feina de tenir que convertir-los nosaltres.

El Script també es retorna un altre fitxer amb extensió “material” on tenim tots el materials de tots el models que hem exportat i també les textures associades a aquets materials.

Tots aquests fitxers són directament interpretables pel Ogre3D.

OgreMeshes té algunes opcions extres com per exemple (*Figura 5*):

Copy Textures: Ens permet copiar les textures associades al Material a la mateixa ruta que els Mesh.

Fix Up Axis to Y: Aquesta opció ens ha estat molt útil ja que Ogre3D usa com Up Axis (Eix Cap amunt) el Y, però en canvi Blender usa Z per tant si féssim servir aquesta opció tindríem problemes de incompatibilitat alhora de moure els models en Ogre3D.

Skeleton Name Follow Mesh: opció útil per identificar els fitxer Mesh amb quin model es corresponen ja que assigna el nom del model a nom del fitxer.

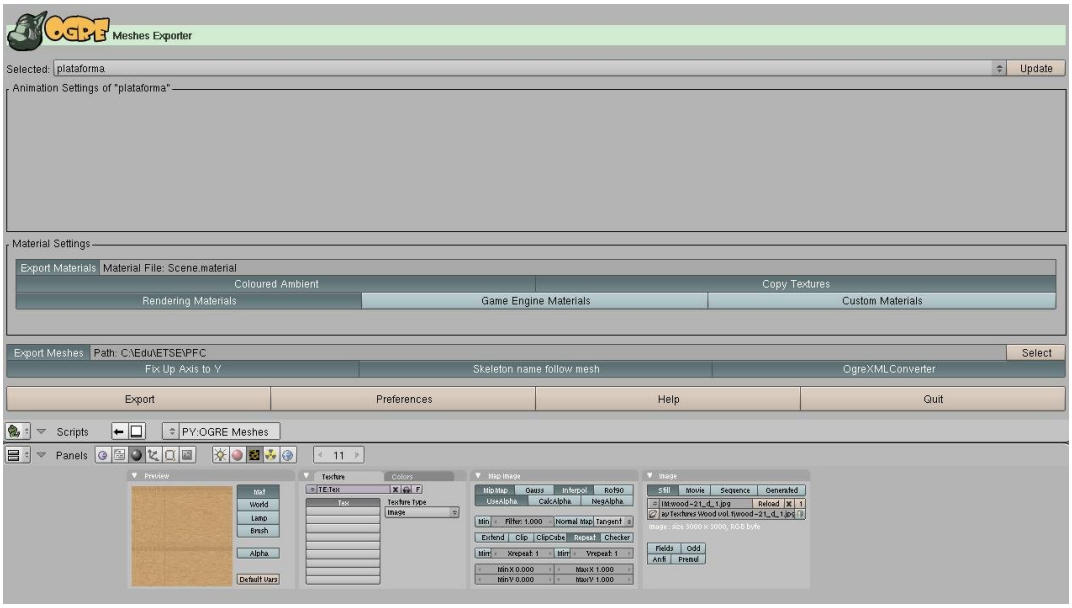


Figura 5: Blender, menú exportació ogremeshes

3.1.4.2 Textures UV Mapping

Una altre opció important del Blender es la textura UV Mapping. La forma més flexible de crear un “mapping” (adaptar estructura lògica) d'una textura 2D en un objecte 3D és un procés anomenat “UV Mapping” (*Figura 6 i 7*). Aquest procés, pren les tres dimensions (X, Y i Z) de la malla i les aplica en un plànol de dos dimensions (X , Y). Els colors de la imatge són assignats a la malla, i es mostren com el colors de les cares de la malla. L’ús de textures UV proporciona realisme als objectes, que els procediments de materials i textures no poden aconseguir, i millora els detalls que pot proporcionar Vertex Painting (una altre tècnica per aplicar textures a models).

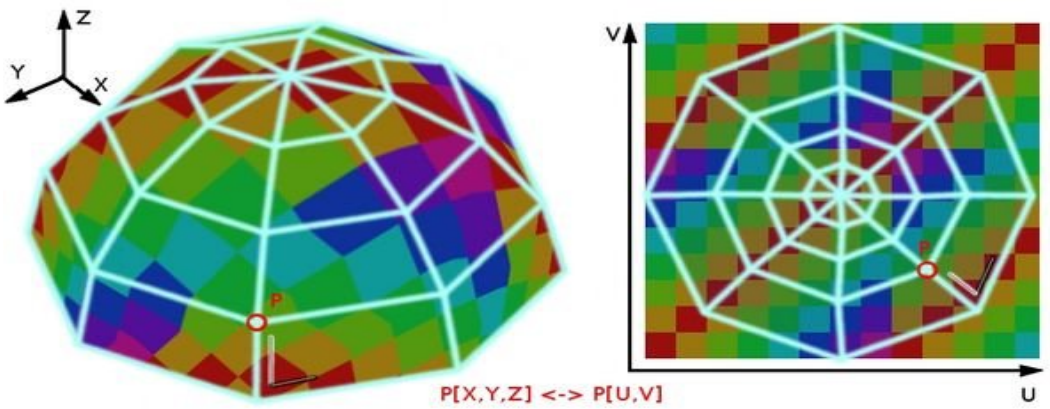


Figura 6: Exemple UV Mapping

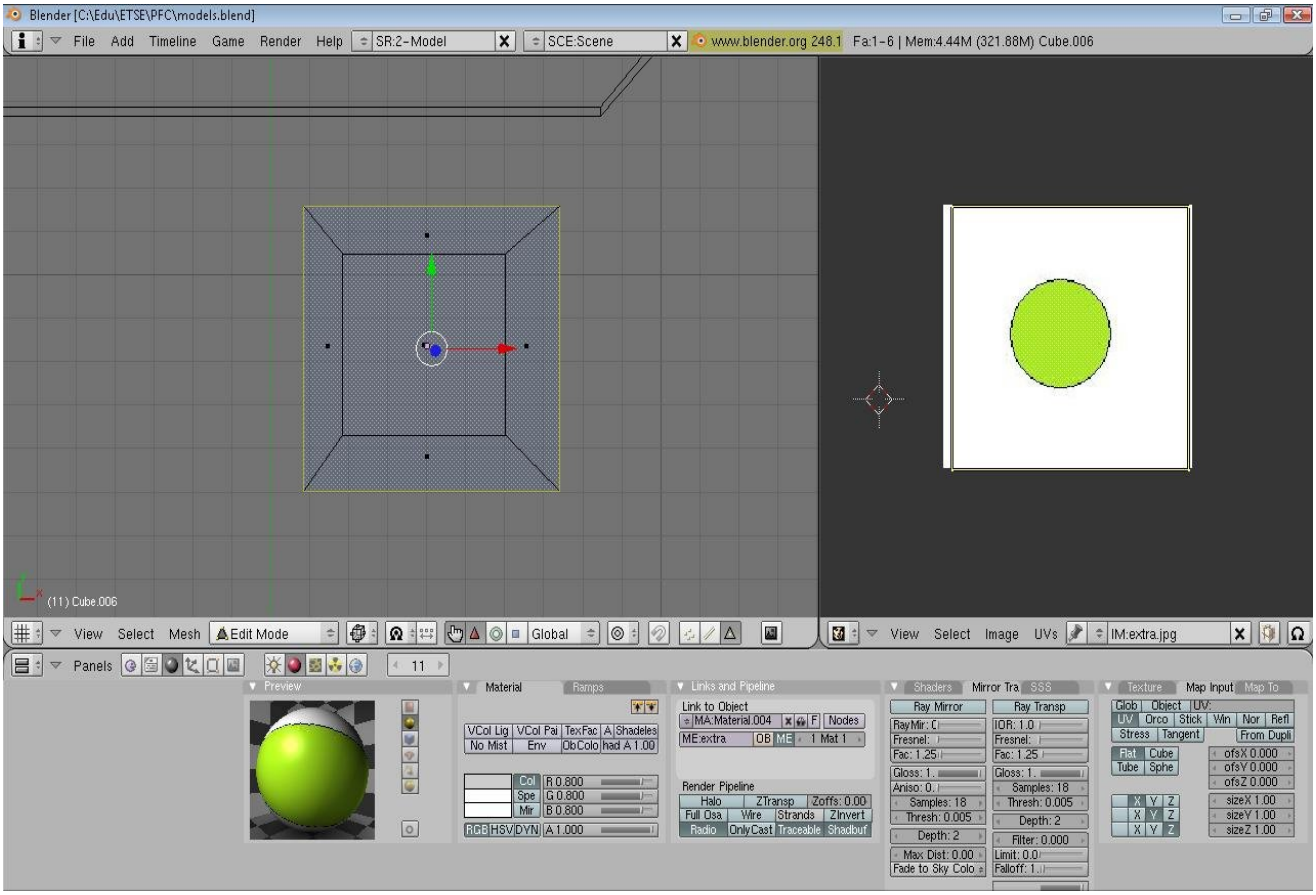


Figura 7: Blender, Comparació Vista 3D (esquerra) amb vista UV Mapping (dreta)

3.2 Lògica

Aquí parlarem de tot el que té a veure amb la funcionalitat del videojoc. És a dir, en les idees bàsiques que sustenten el joc 3D, però sense entrar en detalls de programació que els veurem en un altre subapartat.

3.2.1 Argument del Videojoc

Com ja s'ha comentat en la introducció, el interès principal per desenvolupar un videojoc era desenvolupar les tècniques de disseny 3D. Per tant, l'argument del joc no deixà de ser una part anecdòtica del projecte. En altres paraules, el mateix desenvolupament que s'ha fet per aquest videojoc particular, serveix idènticament per a multitud d'altres arguments de videojocs. En particular, l'argument del videojocs està basat en el famós videojoc Arkanoid (veure *Figura 8*). Però en aquest cas serà, en 3D, es clar.

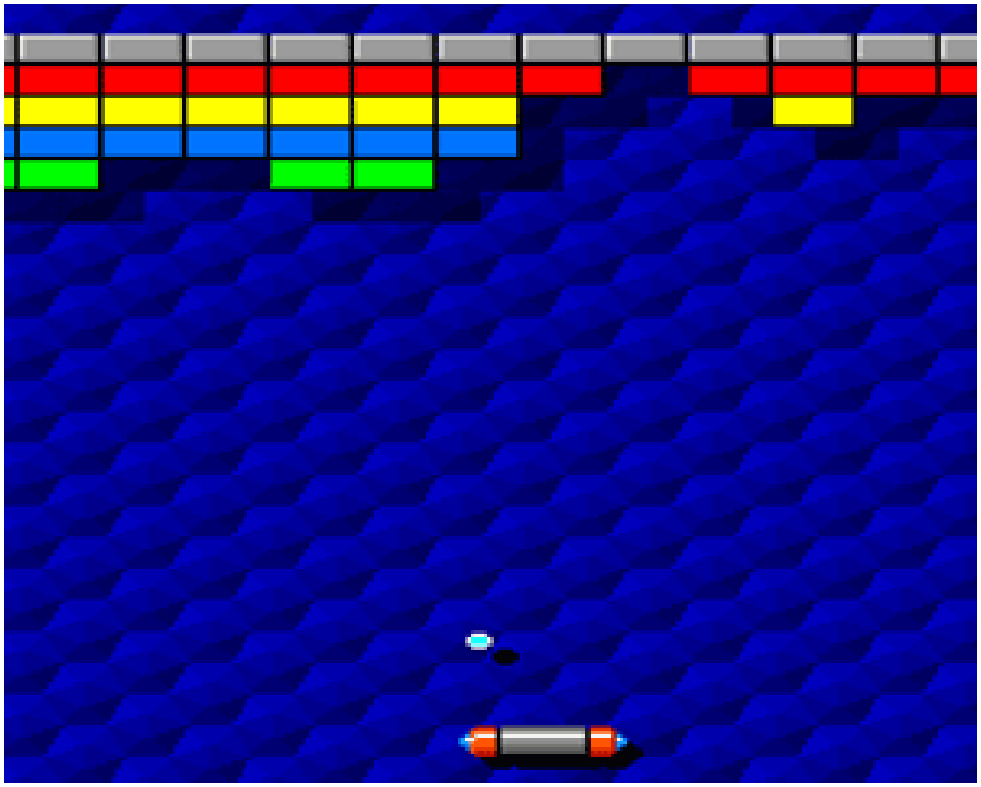


Figura 8: Arkanoid

Per tant, l'usuari tindrà el control d'una plataforma que podrà moure per l'escenari 3D (dreta-esquerra, endavant-enderrera, a dalt i a baix). Mitjançat la plataforma haurà de controlar les boles que aniran movent-se per el escenari. L'Escenari serà una habitació

on en el sostre, o la part alta del escenari, hi haurà tot un conjunt de blocs. El terra estarà dividit en 4 parts que s'aniran trencant a mesura que les boles impactin contra el terra. Un cop trencades aquestes parta, les boldes podran sortir de l'habitació per els forats que hauran deixat, si totes les boles cauen per els forats el jugador perd la partida.

L'objectiu del jugador és fer que les boles impactin contra la plataforma de manera que aquestes surtin rebotades en direcció als blocs impactant-los i així destruint-los.
Un cop destruïts tots el blocs el jugador ha guanyat la partida.

3.2.2 Plataforma

La plataforma (veure *Figura 9*) és l'objecte dinàmic que el jugador utilitza per interactuar amb el videojoc. És l'encarregat de assegurar-se que les boles no xoquen amb el terra trencant-lo, sinó que aquestes impacten contra la plataforma de manera que reboten cap enlaire i destrueix els blocs amb els que impacten.

La plataforma es controlada per el jugador amb el teclat de la següent manera:

Tecles esquerra,dreta (fletxes): mou la plataforma en l'eix X a l'esquerra i a la dreta respectivament.

Tecles amunt,avall (fletxes) mou la plataforma en l'eix Y amunt i avall respectivament.

Tecles repag,avpag: mou la plataforma en l'eix Z endavant i endarrere respectivament.

Figura 9: Plataforma

3.2.3 Blocs

Els blocs (*Figura 10*) són els objectes que el jugador té que destruir mitjançant les boles.
Tots els blocs tenen la mateixa mida. Un cop un bloc és destruït desapareix de l'escena. Alguns dels blocs un cop destruïts deixen caure un objecte nou a l'escena, d'aquests objectes parlarem al següent apartat.

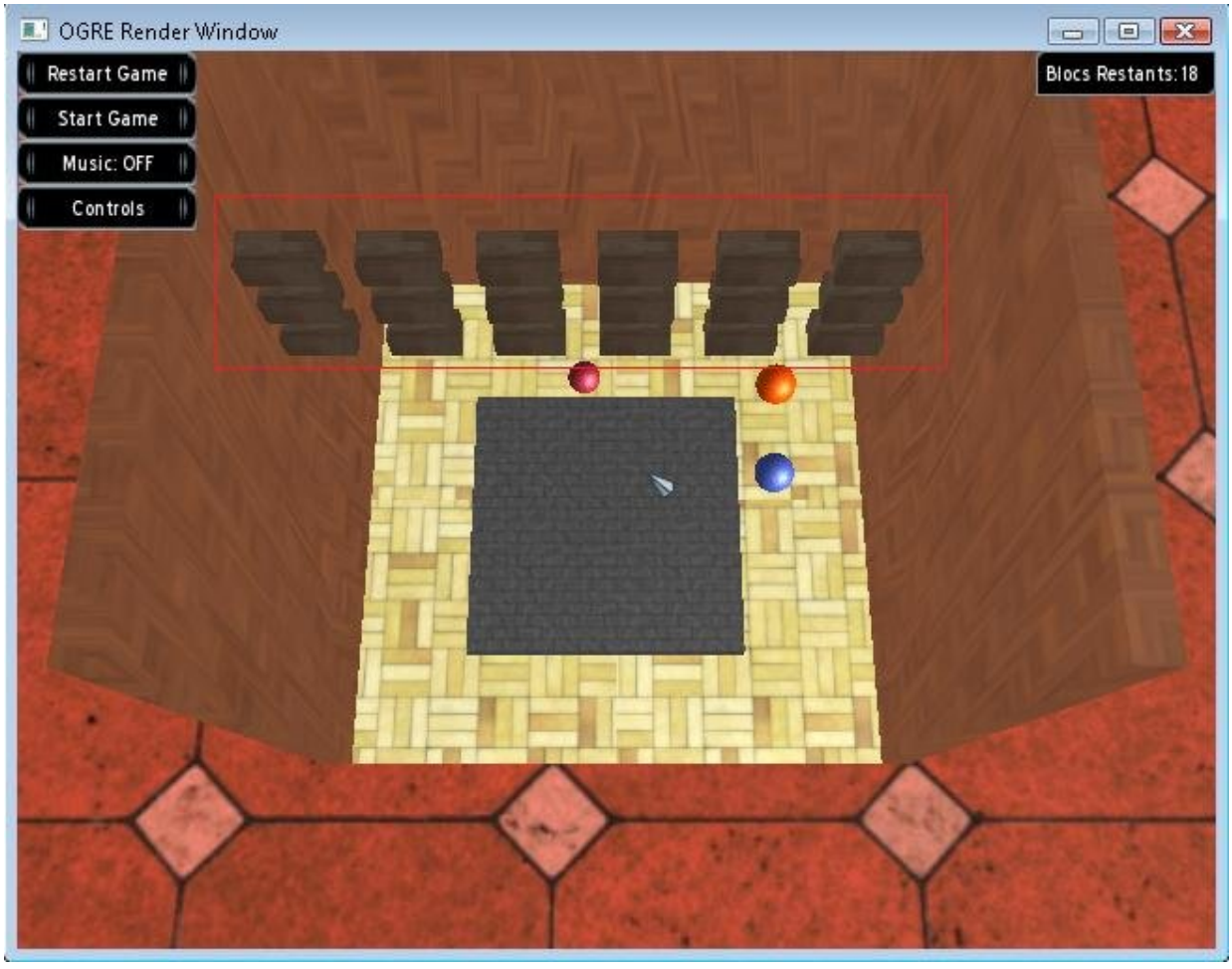


Figura 10: Blocs

3.2.3.1 Objectes

Com ja hem dit, en alguns dels blocs, quan són destruïts, quan són destruïts apareixen objectes amb propietats originals pel jugador. Aquestes objectes nomes interactuen amb la plataforma, és a dir ni xoquen amb les paret ni amb les boles ni amb altres blocs.
Si la plataforma toca el objectes es dispara un esdeveniment i sinó el toca i el objecte surt del cub (es a dir l'habitació) i simplement desapareix. Depenent del tipus d'objecte el esdeveniment que es "dispara" (activa) és diferent.

Hi ha de tres tipus d'objecte segons l' esdeveniment que disparen:

Plus: La plataforma augmenta de mida.

Minus: La plataforma disminueix de mida.

Extra: S'afegeix una altre bola al joc.

3.2.3.2 Malla Virtual

Per tal de tenir controlat en tot moment el gran nombre de objectes que interactuen en el espai real 3D del videojoc simultàniament ens hem creat un mallat virtual que ens permet conèixer, en tot moment, les posicions dels objectes i les seves possibles interaccions. .
Aquesta malla esta formada per cubs, cada cub té la mateixa mida i tots junts formen un cub gran que es l'escenari global que, anteriorment, hem anomenant com a cub, habitació o escenari. Evidentment com es un mallat virtual no es visible, sinó només aplicable a la lògica del videojoc.

La construcció d'aquest mallat virtual s'ha aconseguit de la següent manera: Cada cub del mallat virtual té un índex tridimensional que l'identifica en cada eix de l'escena, és a dir, un nx,ny i nz. Per tant cada objecte pot ocupar un o més cubs en cada frame depenent de la mida dels cubs i de la mida del objecte. D'aquesta manera tenim controlat per on estan els objectes en tot moment i podem controlar que en el mateix instant dos objectes no estan en el mateix cub o s'estan tocant, ja que ocupen el mateix cub.
Per tal de controlar la interacció entre els objectes presents al escenari, els cubs tenen una altre propietat que anomenem "color". El "color" és un índex que identifica quin tipus d'objecte està tocant el cub, per exemple, el color 0 vol dir que cap objecte està toquen el cub i el color -1 vol dir que un objecte de tipus paret esta tocant el cub.

P er exemple, si un objecte dinàmic va a canviar de cub i veu que el color del cub on va a parar es -1 sap que es una paret per tant abans de tocar el cub xocarà i la seva velocitat prendrà una direcció oposada a la que tenia inicialment l'objecte dinàmic.

Cal a dir que, quan més petita és la mida dels cubs més precisos son el moviments, es a dir les lleis físiques que determina el moviment dels objectes i les seves possibles interaccions.

3.3 Programació

En aquesta secció, parlarem del codi del videojoc pròpiament. Diferenciarem entre codi generat per llibreries externes és a dir de codi importat (com poden ser els fitxers de Blender o les llibreries de Ogre) i el codi generat per nosaltres per desenvolupar el videojoc.

En la figura 11 veiem un diagrama de flux de com funciona el videojoc, i les diferents rutines que controlen tots els aspectes del joc. En aquest diagrama es barregen tant codi propi com codi importat.

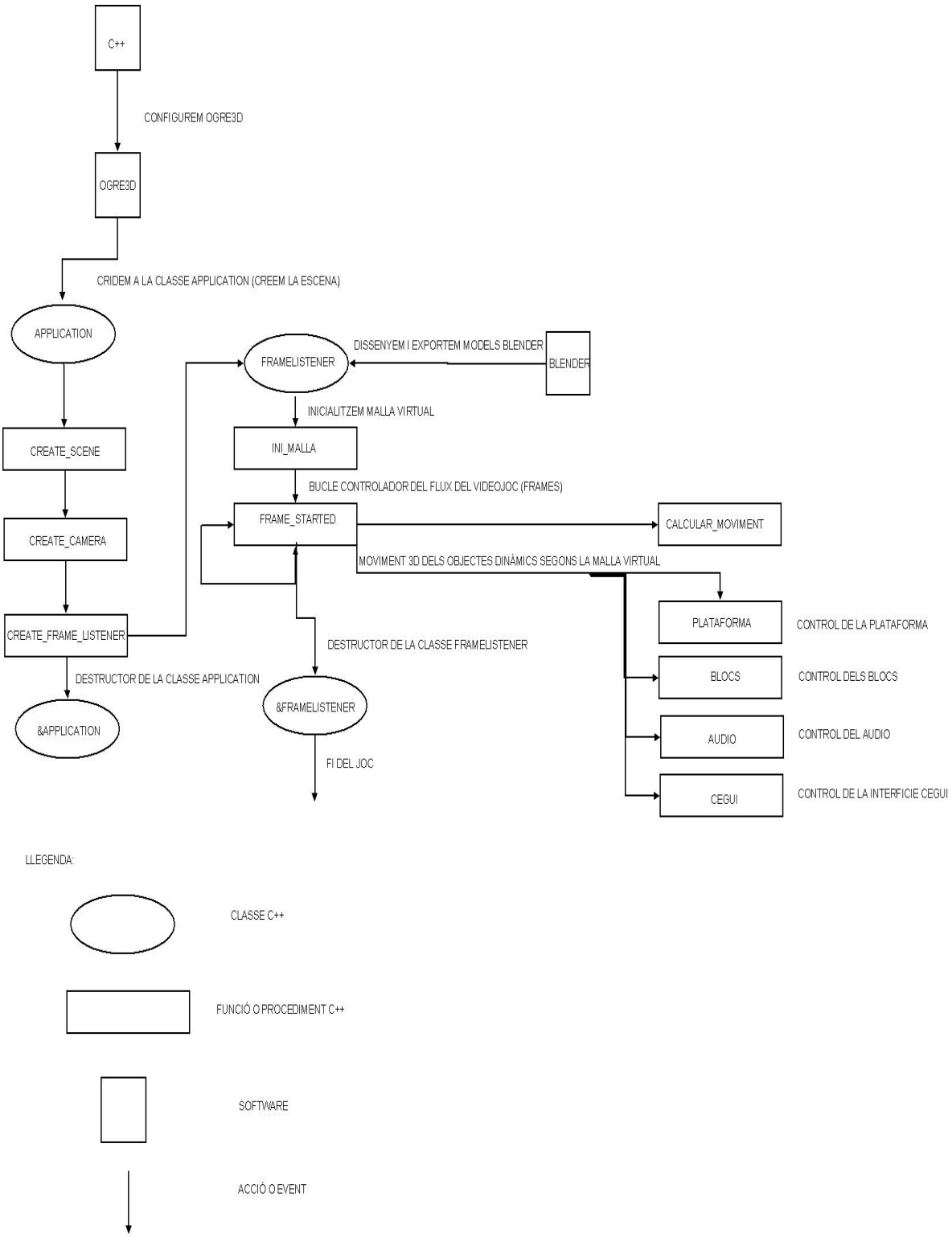


Figura 11: Diagrama de Flux del Videojoc

3.3.1 Generació de codi importat

A continuació, detallem els aspectes mes importants del codi importat que utilitzarem en l'aplicació final. Òbviament, aquest codi farà referència principalment a les llibreries de Ogre3D mencionades anteriorment, i en part també a la utilització del Blender per el dibuix d'objectes 3D.

Configuració Bàsica Ogre3D

A continuació destaquem tres propietats bàsiques que s'han de configurar del motor gràfic Ogre3D abans de poder veure una escena 3D en pantalla.

3.3.1.1 Scene manager

Normalment el Scene manager a Ogre3D es responsable de moltes propietats que determinen l'escena 3D, aquí citarem les propietats que hem fet servir nosaltres durant el projecte:

a.- Crear i situar objecte mòbils, llums i càmeres a l'escena d'una manera que puguem accedir a elles eficientment.

b.- Implementar consultes a l'escena per obtenir respostes a preguntes com: "Quins objectes estan continguts en una esfera centrat en un punt determinat de l'escena ? "

c.- Obviar objectes no visibles i posar objecte visibles en cues de render per tal de poder visibles en cada frame.

d.- Organitzar i distribuir les llums no direccionals des de la perspectiva del frame actual.

e.- Configurar i fer render totes les ombres de l'escena.

f.- Configurar i fer render tots els altres objectes de l'escena, com backgrounds i skyboxes.

g.- Passar tot aquest contingut al sistema de render per ser visible a cada frame.

3.3.1.1.1 Scene Nodes

El Scene manager es també l'origen dels nodes usats per definir l'estructura del graf de l'escena.

Els Scene Nodes estan organitzats amb el Scene Manager amb jerarquia: un scene node té un sol pare i pot tenir cero o més fills nodes. Pots assignar i desvincular scene nodes des de el scene manager a la teva voluntat, el scene node no es destruït fins que tu li dius al scene manager que el destrueixi. Això va molt bé per fer accés directes, és a dir si vols fer no visible tota una secció d'objectes simplement desvincules el scene node root que es el primer pare de la jerarquia de scene nodes i tant ell com tots els demes no seran visibles.

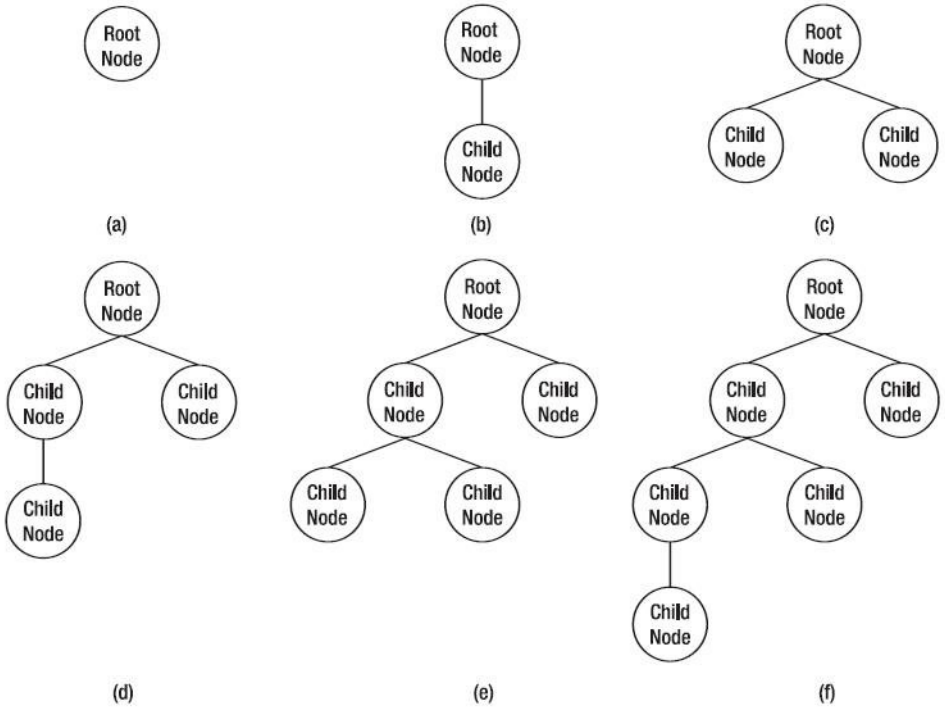


Figura 12: Jerarquia Arbres Scene Nodes

En aquesta imatge (*Figura 12*) veiem grafs de l'escena on des de el "a" fins al "f" anem afegint fills nodes al node pare (root node).

El Scene manager et garanteix que quan crea un objecte, el crea com a mínim amb un scene node, el root scene node. Aquest és el únic node en el graf de la escena que esta exclòs de la regla només un pare, el root scene node per definició no té pare, tu no pots destruir el root scene node.

3.3.1.2 Càmera

Una Cámara és el mateix que el seu anàleg en el mon real: fa una foto de la escena en cada "frame" (fotograma del videojoc), des de un punt particular (és a dir té una posició i una orientació). No és un objecte que es pugui fer "render" (dibuixar en l'escena) , és a dir, encara que tu tinguis una càmera en el camp de visió d'una altre càmera, tu no podràs veure-la. Les Càmeres (com les Llums, després veurem com són) també poden ser associades a un SceneNodes i d'aquesta manera poder ser controlades per l'aplicació o existir en un espai lliure (que vol dir que tu les pots moure manualment si tu vols canviar la seva posició o orientació en cada "frame"). Les Càmeres tenen camps de visió amb "near i far planes" (plans de visió propers o llunys a la càmera). Aquesta Geometria defineix el que es conegut com a "frustum", que es un tipus de piràmide (*Figura 13*).

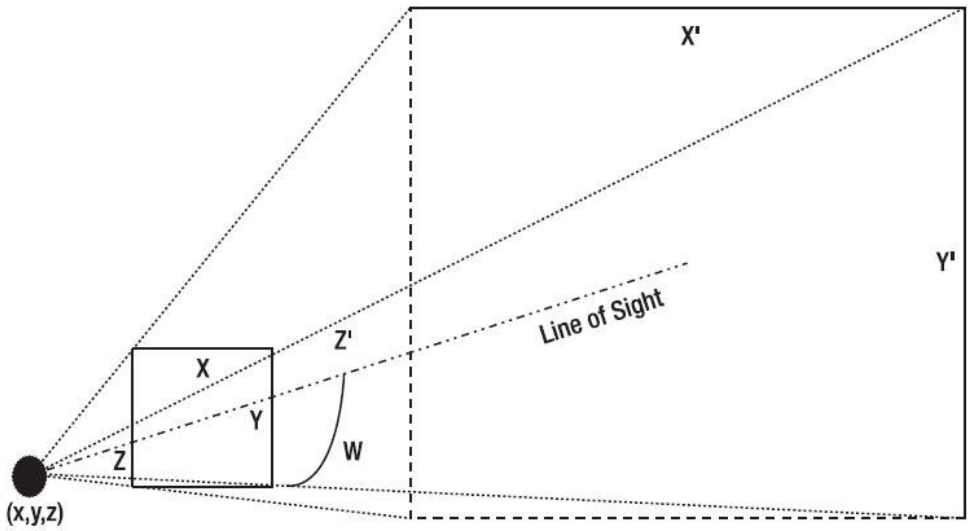


Figura 13: Visió de la Camara

En aquesta figura, (x,y,z) indica la localització de la Camara. X i Y són la mida del near clip plane, i són una funció de distància Z des de la Càmera fins al near plane. X' i Y' són la mida del far plane, i són una funció de distància (Z+Z') des de la Càmera fins al far plane. Tu informes les distàncies del near i el far plane, el aspect ratio de la càmera (definit com X/Y), i el angle vertical W entre la línia de visió i el més baix (i alt) "frustrum" bounding plane (que es el camp de Visio del angle Y), i la classe de la càmera calcula el horitzontal, el angle i la mida dels near i els far planes. Posem que tu vols tenir una càmera amb el Standard 4:3 "aspect ratio" (relació del aspecte. És a dir proporció entre la amplitud i altura de la pantalla), amb la distància del near plane a 5 unitats de la càmera i la distància far plane a 1000 unitats, amb un angle de 30-graus, entre la línia de visió i el mes baixos (i alts) "frustum bounding planes" (límits del "frustrum"). El següent Codi crea una càmera amb aquestes característiques:

```
// sceneMgr és una instància de una implementació de un SceneManager.
//Estem creant una càmera amb nom "MainCam" aquí.

Camera *camera = sceneMgr->createCamera("MainCam");

// normalment calculem això amb la mida del viewport
camera->setAspectRatio(1.33333f);
```

// 30 graus ens donaran una vista telescòpica

camera->setFOVy(30.0f);
camera->setNearClipDistance(5.0f);
camera->setFarClipDistance(1000.0f);

3.3.1.3 Llum

Ogre3D permet determinar 3 tipus de llums diferents:

- Points Lights.
- Spots Lights.
- Directional Lights.

Nosaltres hem fet servir Point Lights:

Point Lights són molt comunes en una escena 3D. Tenen les característiques de una llum radiant en totes les direccions des de un punt determinat en l'escena.
Les Point Lights són molt útils per simular la radiació d'una llum.

Exemple de codi de una llum Point Light:

```
mSceneMgr->setAmbientLight(ColourValue(1, 1, 1));

Light *light = mSceneMgr->createLight("Light1");
light->setType(Light::LT_POINT);
light->setPosition(Vector3(250, 150, 250));
light->setDiffuseColour(ColourValue::White);
light->setSpecularColour(ColourValue::White);
```

3.3.1.4 Entrada Dispositius

Per poder realitzar i jugar al videojoc necessitem tenir en compte i poder controlar l'entrada de dispositius externs, en el nostre cas el mouse i el teclat, per això farem servir una altre llibreria externa anomenada OIS.
Gràcies a OIS podrem saber coses com quant una tecla ha estat pitjada o quant un boto ha estat pitjat o en quina posició de la pantalla està situat el Mouse.

3.3.1.5 Interfície

La interfície del videojoc (*Figura 14 marcat en vermell*) està feta amb una llibreria externa anomenada CEGUI que expliquem a continuació:

Figura 14: Interfície Gràfica (CEGUI)

CEGUI

Crazy Eddie's GUI (CEGUI) és una llibreria de C++ per interfícies gràfiques d'usuari. Esta dissenyada particularment per les necessitats dels videojocs, però la llibreria també es útil pers tasques no relacionades amb videojocs. Esta dissenyada per ser flexible i també es adaptable a la decisió dels usuaris en quant a sistemes operatius.

En el nostre cas particular, de totes les possibilitats de CEGUI (que són moltes), hem fet servir els botons i les caixes de text.

3.3.1.6 So

Per reproduir musica i sorolls al videojoc utilitzem una llibreria externa de C++ anomenada Fmod.

FMOD

FMOD es una llibreria d'àudio comercial dissenyada per "Firelight Technologies", que reprodueix fitxers de musica de diversos formats i plataformes. Es usat en jocs i aplicacions de software per proveir una funcionalitat d'àudio. Fmod suporta un ampli ventall de formats d'àudio i nombroses plataformes de sistemes operatius**.**

Fins la versió 3.75, la llibreria era anomenada simplement com FMOD. Des de llavors FMOD ha sigut re dissenyada i ara conté 3 grans parts:

- FMOD Ex, el nivell d'àudio mes baix

- FMOD Event System, més abstracta, una capa d'aplicació de més nivell per simplificar el contingut del play back creat amb FMOD Designer
- FMOD Designer, l'eina per dissenyar àudio usada per sorolls més complexes i musica per play back.

FMOD té una avançada arquitectura de plugins, que pot ser usada per ampliar el suport dels formats d'àudio i per desenvolupar nous tipus com per exemple: Streaming.

3.3.2 Generació de codi original

Un cop hem desglossat tot el codi extern que utilitzarem en el nostre videojoc, passem a descriure el codi original que hem desenvolupat en aquest projecte.

3.3.2.1 Classes

Hem dividit tot el codi C++ del projecte en dos grans classes com es veu en la figura 10.

3.3.2.1.1 Creació Escena

És la classe on creem els components bàsics de l'escena i fem la crida a FrameListener passant-li tota la informació necessària.

3.3.2.1.2 FrameListener

En aquesta classe hi ha el gruix del codi. Com el seu nom indica, aquest classe es dedica a escoltar i controlar tot el que passa a cada “frame” del joc, la podríem veure com el bucle principal del videojoc.

3.3.2.2 Rutines

Aquí parlarem de les rutines pròpies més importants del codi.

3.3.2.2.1 Càlcul Moviment

És una de les funcions més importants del codi, juntament amb FrameStarted que explicarem al següent apartat. Aquesta funció és l’encarregada de moure els objectes per la Malla Virtual que hem mencionat en l’apartat 3.2.3.2, per tant, té la missió de controlar moltes variables per tal de que tots els objectes es mogui e interactuïn seguint les regles preestablertes.

Durant cada frame, aquesta funció es crida un cop per cada objecte dinàmic.

Entre els molts càlculs que fa aquesta funció, es troba el saber si el objecte en qüestió que està movent-se es mantindrà en el cub de la malla virtual on es troba actualment, canviarà de cub o per el contrari entrarà amb contacte amb algun altre objecte dinàmic degut a que en el cub de la malla virtual on l’objecte volia moure’s es troba ocupat per un altre objecte.

Apart de calcular lògicament on anirà a parar el objecte que està tractant també es dedica a situar-lo físicament dintre de l’escena i d’actualitzar totes les dades referents al objecte en si, com per exemple, en quin cub de la malla virtual es troba o les velocitats i

posicions del objecte o el canvi de color del cub de la malla que el objecte que està movent esta tocant etc.

3.3.2.2.2 FrameStarted

És la funció més important de la classe FrameListener, és l'única funció de tot el codi que s'executa en cada frame, per tant es la que es dedica a cridar a les altres funcions i controlar el flux del videojoc.
Des de aquí es crida a cada frame a tots el objectes dinàmics (Boles i objectes dels blocs) com hem comentat abans amb la funció Càlcul Moviment.
També controla quants blocs resten per finalitzar el jocs, és a dir, quants blocs resten per ser destruïts.

3.3.2.3 Estructures

Per tal de tenir controlades totes les dades que es mouen en el videojoc ens hem creat dues estructures per emmagatzemar les dades de forma organitzada. Hem anomenat a aquestes estructures, "objectes" i "espai".

3.3.2.3.1 Objecte:

Aquí guardem tota d'informació referent als objectes dinàmics del videojoc, és a dir la plataforma i la bola o boles que puguin estar movent-se per l'escena.

Dintre de l'estructura hi ha les següents dades:

X: Posició del objecte a l'eix X.

Y: Posició del objecte a l'eix Y.

Z: Posició del objecte a l'eix Z.

VX: Velocitat del objecte a l'eix X.

VY: Velocitat del objecte a l'eix Y.

VZ: Velocitat del objecte a l'eix Z.

NX: Posició del objecte a l'eix X de la malla virtual.

NY: Posició del objecte a l'eix Y de la malla virtual.

NZ: Posició del objecte a l'eix Z de la malla virtual.

M: Massa del objecte.

Índex: Identificador únic del objecte.

Activa: Booleà per saber si el objecte està actiu o no.

3.3.2.3.2 Espai

Aquí guardem tota l' informació referent a la malla virtual de l'escena.

Dintre de l'estructura hi ha les següents dades:

Inix: Posició Inicial del Cub de la malla a l'eix X.

Iniy: Posició Inicial del Cub de la malla a l'eix Y.

Iniz: Posició Inicial del Cub de la malla a l'eix Z.

dx: Mida del Cub de la malla a l'eix X.

dy: Mida del Cub de la malla a l'eix Y.

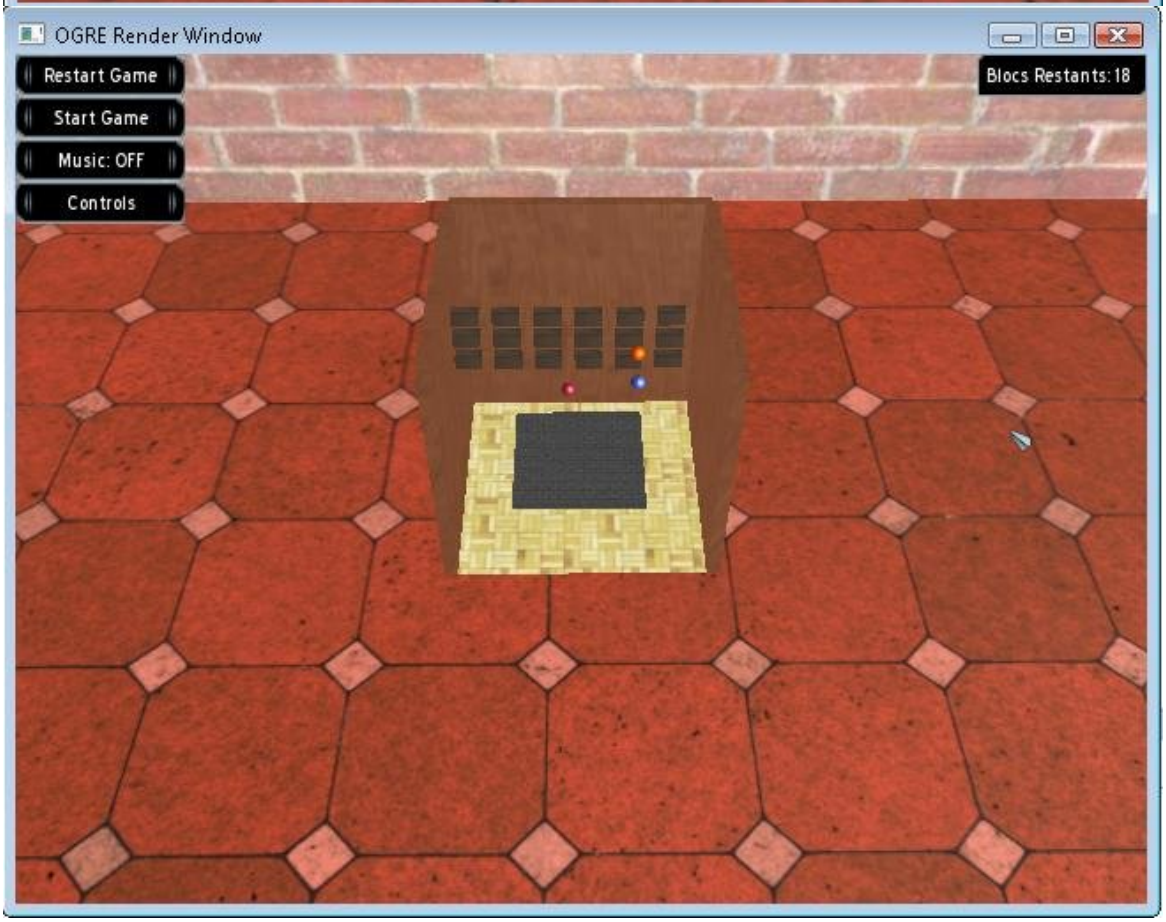
dz: Mida del Cub de la malla a l'eix Z.

Color: Integer usat per identificar quin esdeveniment es dispara quant un objecte impacta el cub virtual.

4 Resultats

Aquí veurem algunes imatges que mostren els resultats del projecte.


OGRE Render Window
Restart Game
Start Game
Music: OFF
Controls
Blocs Restants: 18


OGRE Render Window
Restart Game
Start Game
Music: OFF
Controls
Blocs Restants: 18


OGRE Render Window
Restart Game
Pause
Music: ON
Controls
Blocs Restants: 17
Controls:
Plataforma:
Eix X: LEFT, RIGHT
Eix Z: UP, DOWN
Eix Y: Repag, Avpag
Càmera:
Rotar X: A, D
Rotar Y: T, G
Zoom: W, S


OGRE Render Window
Restart Game
Pause
Music: OFF
Controls
Blocs Restants: 16

5 Expectatives de Futur

Com s'ha comentat en la introducció d'aquest projecte, els videojocs passen per molts bons moments en quan a volum de vendes i de usuaris. Per tant, hi ha una gran industria al darrera dels videojocs que genera una demanda important de llocs de treball, tot i que la demanda a Espanya no es tan bona com el numero de vendes.

El temps de realització d'aquest projecte, des de els seu origen fins a la conclusió final, és limitat. Per tant, hi ha una gran part de les expectatives que s'ha generat durant la realització d'aquest projecte que no s'ha pogut dur a terme. En un futur immediat, es pretén millorar i ampliar les capacitats del videojoc (noves pantalles i lògiques diferents...) utilitzant l'"esquelet" de programació basic pers jocs 3D generat durant aquest projecte.

La intenció última d'ampliar aquest projecte es poder fer-lo servir com a carta de presentació per poder obtenir una entrevista per un lloc de treball dins de la industria de programadors de videojocs.

6 Conclusions

Un projecte final de carrera, consisteix en plantejar-te uns objectius i dur-los a terme mitjançant investigacions i desenvolupaments basats en els coneixements fonamentals obtinguts durant la carrera, i/o ampliant-los convenientment, per tal d' assolir els objectius establerts al inici del projecte.

Com s'ha comentat en l'apartat d'objectius, la idea original d'aquest projecte ha estat proposada per mi mateix, com alumne, per donar sortida a les meves pròpies inquietuds personals. L'objectiu era realitzar un videojoc en un espai real 3D. Per tal de dur a terme aquest objectiu, primerament s'ha hagut de decidir quin software específic, ja desenvolupat, s'utilitzava per facilitar el treballar en l'espai 3D. En el meu cas he triat Blender per dissenyar els objectes 3D que apareixeran en el videojoc, C++ com a llenguatge de programació necessari per desenvolupar el videojoc i Ogre3D que és un conjunt de llibreries bàsiques, comunament anomenat com "Motor Gràfic".

Tot i que més anecdòtic per avaluar el projecte, també era importar decidir la temàtica del videojoc, i en aquest cas es va decidir realitzar un joc amb una temàtica senzilla, del estil dels clàssics com per exemple el "Tetris". Es a dir, un joc que accentua les habilitats del jugador per assolir uns objectius fent ús de la lògica, y que per contra generalment implica uns gràfics més lleugers, una durada del joc més curta i una estil de joc més repetitiu. En el meu cas he triat realitzar la conversió del famós joc "Arkanoid" desenvolupat originàriament en un pla (espai 2D) a un espai real 3D.

La particularitat més important en la programació d'aquest videojoc ha estat el fet de que hem diferit de la el que la majoria de programadors que utilitzen llibreries gratuïtes que s'encarreguen de calcular els moviments i interaccions entre objectes, comunament anomenat com "Motor de Física" (per analogia amb el "Motor Gràfic" que hem mencionat durant el projecte). Nosaltres varem decidir des de bon inici implementar nosaltres mateixos tota la lògica de programació referent al xoc entre partícules, sabent però de les avantatges i desavantatges que això comporta. Les desavantatges són el fet

de que els conjunts de llibreries d'avui en dia son molt complets i pots obtenir una funcionalitat molt alta i molt eficaç i en pocs temps. Per altre banda, les avantatges són que no hem estat lligats a codificar com unes llibreries ens imposen i hem estat lliures de decidir tots els detalls del moviment de les partícules, aconseguint un "realisme" en el càlcul del moviment i dels xocs molt elevat.

En aquest sentit, també volem destacar que hem hagut de fer coincidir (i col·laborar) el codi extern de les llibreries del Motor Gràfic (OGRE3D) amb el codi intern que nosaltres hem desenvolupat , tant la part dels xocs de partícules com en la dinàmica de cada objecte individual.

També hem après a incloure models dissenyats amb Blender al nostre llenguatge de programació i fer-los interactuar amb els nostres dos mons: Codi Propi i Motor Gràfic.

Personalment, crec que tots els objectius que ens varem plantejar en un origen han estat assolits satisfactòriament i la realització d'aquest projecte ha estat una experiència molt positiva, tant a nivell acadèmic, com personal.

Finalment, com ja he dit en el capítol 5, aquesta memòria no es un punt i final, sinó un punt i seguit en el desenvolupament del videojoc 3D que s'ha desenvolupat. La intenció última és ampliar aquest projecte en un futur immediat per, si s'escau, poder fer-lo servir com a carta de presentació en la industria de videojocs.

7 Bibliografia

7.1 Url's

http://www.ogre3d.org/wiki/index.php/Ogre_Tutorials

http://www.aserrano.net/2007/12/19/blender-exportar-modelos-a-ogre/

http://www.youtube.com/watch?v=d2U0gOoMtkc

7.2 Llibres

Pro Ogre3D programming Gregory Junker.

Blender Curso de Iniciación Mercè Galán

8 Annexes

Codi per la col·lisió elàstica de dues boles

```
void FL::collision3D(double m1, double m2, double r1, double r2,int aux,int index)     {


double pi,r12,m21,d,v,theta2,phi2,st,ct,sp,cp,vx1r,vy1r,vz1r,thetav,phiv,
            dr,alpha,beta,sbeta,cbeta,t,a,dvz2,vx2r,vy2r,vz2r,
            x21,y21,z21,vx21,vy21,vz21,x1,y1,z1,x2,y2,z2,vx1,vy1,vz1,vx2,vy2,vz2;



              x1 = o[index].x;
              y1 = o[index].y;
              z1 = o[index].z;
              x2 = x1-(dx0/1000);
              y2 = y1-(dy0/1000);
              z2 = z1-(dz0/1000);
              vx1 = o[index].vx;
              vy1 = o[index].vy;
              vz1 = o[index].vz;
              vx2 = o[aux].vx;
              vy2 = o[aux].vy;
              vz2 = o[aux].vz;



//     **** initialize some variables ****
       pi=acos(-1.0E0);
       r12=r1+r2;
       m21=m2/m1;
       x21=x2-x1;
       y21=y2-y1;
       z21=z2-z1;
       vx21=vx2-vx1;
       vy21=vy2-vy1;
       vz21=vz2-vz1;

//     **** calculate relative distance and relative speed ***
       d=sqrt(x21*x21 +y21*y21 +z21*z21);
       v=sqrt(vx21*vx21 +vy21*vy21 +vz21*vz21);
```

```
//    **** shift coordinate system so that ball 1 is at the origin ***
      x2=x21;
      y2=y21;
      z2=z21;

//    **** boost coordinate system so that ball 2 is resting ***
      vx1=-vx21;
      vy1=-vy21;
      vz1=-vz21;

//    **** find the polar coordinates of the location of ball 2 ***
      theta2=acos(z2/d);
      if(aux==104)
                  phi2=0;
            else{
                        if (x2==0 && y2==0)
                        {
                              phi2=0;
                        }
                        else
                        {
                              phi2=atan2(y2,x2);
                        }
            }
      st=sin(theta2);
      ct=cos(theta2);
      sp=sin(phi2);
      cp=cos(phi2);


//    **** express the velocity vector of ball 1 in a rotated coordinate
//       system where ball 2 lies on the z-axis ******
      vx1r=ct*cp*vx1+ct*sp*vy1-st*vz1;
      vy1r=cp*vy1-sp*vx1;
      vz1r=st*cp*vx1+st*sp*vy1+ct*vz1;
      thetav=acos(vz1r/v);
            if(aux==104)
                  phiv=0;
```

```
        else{
                if (vx1r==0 && vy1r==0)
                {
                        phiv=0;
                }
                else
                {
                        phiv=atan2(vy1r,vx1r);
                }
        }


//      **** calculate the normalized impact parameter ***
        dr=d*sin(thetav)/r12;

//      **** calculate impact angles if balls do collide ***
        alpha=asin(-dr);
        beta=phiv;
        sbeta=sin(beta);
        cbeta=cos(beta);


//      **** calculate time to collision ***
        t=(d*cos(thetav) -r12*sqrt(1-dr*dr))/v;

//  ***  update velocities ***

        a=tan(thetav+alpha);

        dvz2=2*(vz1r+a*(cbeta*vx1r+sbeta*vy1r))/((1+a*a)*(1+m21));

        vz2r=dvz2;
        vx2r=a*cbeta*dvz2;
        vy2r=a*sbeta*dvz2;
        vz1r=vz1r-m21*vz2r;
        vx1r=vx1r-m21*vx2r;
        vy1r=vy1r-m21*vy2r;


//      **** rotate the velocity vectors back and add the initial velocity
```

```
//          vector of ball 2 to retrieve the original coordinate system ****

      vx1=ct*cp*vx1r-sp*vy1r+st*cp*vz1r +vx2;
      vy1=ct*sp*vx1r+cp*vy1r+st*sp*vz1r +vy2;
      vz1=ct*vz1r-st*vx1r               +vz2;
      vx2=ct*cp*vx2r-sp*vy2r+st*cp*vz2r +vx2;
      vy2=ct*sp*vx2r+cp*vy2r+st*sp*vz2r +vy2;
      vz2=ct*vz2r-st*vx2r               +vz2;


            o[index].vx = vx1;
            o[index].vy = vy1;
            o[index].vz = vz1;
            o[aux].vx = vx2;
            o[aux].vy = vy2;
            o[aux].vz = vz2;

}
```

Estructures

Objecte

```
struct object {
                  double x,y,z,vx,vy,vz,m;
                  int index,nx,ny,nz;
                  bool activa;
};
```

Espai

```
struct cub{
                  double inix,iniy,iniz,dx,dy,dz;
                  int color;
};
```

Resum

Castellà

La industria del los videojuegos crece exponencialmente y está ya superando a otras industrias punteras del ocio. En este proyecto, nos hemos planteado la realización de un videojuego con visualización en el espacio real 3D. Para la realización del videojuego se ha usado el siguiente Software: Blender para diseñar los modelos 3D, C++ como lenguaje de programación para desarrollar el código i un conjunto de librerías básicas para desarrollar un videojuego llamadas Ogre3d (Motor Gráfico).La lógica del movimiento 3D y los choques entre las partículas del juego ha sido diseñada enteramente en este proyecto acorde con las necesidades del videojuego, y de forma compatible a los ficheros de Blender y a las librerías OGRE3D.

Català

La indústria dels videojocs creix exponencialment i està ja superant a altres indústries punteres de l'oci. En aquest projecte, ens hem plantejat la realització d'un videojoc amb visualització en l'espai real 3D . Per a la realització del videojoc s'ha usat el següent Programari: Blender per a dissenyar els models 3D, C++ com llenguatge de programació per a desenvolupar el codi i un conjunt de llibreries bàsiques per a desenvolupar un videojoc cridades Ogre3d (Motor Gràfic). La lògica del moviment 3D i els xocs entre les partícules del joc ha estat dissenyada enterament en aquest projecte d'acord amb les necessitats del videojoc, i de forma compatible als fitxers de Blender i a les llibreries OGRE3D .

Anglès

The gaming industry is growing exponentially and is now outselling other edge of the leisure industries. In this project, we have considered the realization of a video game display in real 3D space.
To achieve the game has used the following software: Blender 3D models for design, C + + programming language as the code and a core set of libraries to develop a video game called Ogre3d (Graphic Engine).
The logic of the 3D motion and collisions between particles of the game has been entirely designed in this project consistent with the needs of the game, and compatible with Blender and file libraries OGRE3D.