

Tàrgaco

CREACIÓ D'UN JOU
D'ORDINADOR

DAVID GUILLEN FANDOS
TUTOR: JAUME SABATER
CURS 2006-2007

Les intel·ligències poc capaces s'interessen per l'extraordinari,
les intel·ligències poderoses per les coses ordinàries.

Victor Hugo

PRESENTACIÓ

Un joc d'ordinador és un programa el qual, per mitjà d'imatges i so, ens distreu d'una manera interactiva. És a dir, podem canviar, moure i modificar aspectes del transcurs del joc d'una manera activa.

N'hi ha de molts tipus: en dues dimensions, en tres dimensions, en primera persona, en tercera, històrics, futuristes, etc. Si triem un joc que succeeix en un "món virtual" haurem d'aconseguir que tots aquells elements que apareixen en la pantalla funcionin de la mateixa manera que ho fan en la vida real.

Però, com es crea un joc d'ordinador? Aquest és precisament l'objectiu que persegueix aquest treball: explicar com es crea un joc d'ordinador de principi a final. I per què triar aquest tema com a treball de recerca? Doncs perquè crear un joc en tres dimensions ens pot aportar moltes coses noves, tant en el camp informàtic com en el camp escolar. Podríem dir que un joc és el màxim exponent de la programació d'ordinadors i una de les millors maneres d'aplicar coneixements físics i matemàtics.

Per totes aquestes raons i d'altres que veurem més endavant he decidit que crear un joc d'ordinador pot ser una bona experiència i que val la pena experimentar-la. Un cop finalitzat aquest treball hem de ser capaços de crear jocs per a ordinador de manera professional.

ÍNDEX

Secció	Pàgina
1 Introducció	
1.1 Els videojocs en l'actualitat	1
1.2 Una mica d'història	2
2 Objectius	3
3 Metodologia	
3.1 Motor 3D i compilador	4
3.2 Programes de modelat 3D	4
3.3 Programes de creació d'executables o compiladors	7
3.4 Metodologia de treball	9
4 Els jocs d'ordinador	
4.1 Història dels videojocs	10
4.2 L'argument en un joc	11
4.3 Creació d'un argument	12
5 Fonaments de la geometria	
5.1 Sistema de representació	14
5.2 Els models en l'espai	15
5.3 Accés a la física	18
6 Simulació de la realitat	
6.1 Moviment i col·isió	19
6.2 Càmera	25
6.3 Cel	28
6.4 Il·luminació	29
6.5 Ombres	31
7 Música i so	
7.1 Música	33
7.2 So 3D	35

Secció	Pàgina
8 Optimitzacions	
8.1 Col·lisió	36
8.2 Repetició de textures	36
8.3 Compressió de textures	37
8.4 Objectes dinàmics	37
8.5 Adjacència i reordenació de triangles	39
9 Finalització del joc	
9.1 Flux del programa	40
9.2 Dibuix amb Direct3D	41
9.3 Funcions i codi del Direct3D	43
9.4 Resultat final	44
10 Conclusió	45
11 Valoració	46
12 Bibliografia	47

Annexos

A Glossari	1
B Ombres	4
C Argument	70
D Disseny	73
E Codi font	77
F CD-ROM: Programes, recursos, codi font i joc final.	

INTRODUCCIÓ

Per a introduir el tema s'ha realitzat una petita explicació de com és ara i com ha estat la creació de videojocs des dels seus inicis. L'objectiu que es persegueix és imitar el funcionament d'una empresa de disseny de jocs creant-ne un. Podeu trobar la definició de les paraules subratllades a l'**annex A**.

1.1 Els videojocs en l'actualitat

Actualment existeixen nombroses empreses creadores de jocs. Els videojocs són un producte de moda que s'adapta a qualsevol públic. Les estadístiques ho demostren; a Espanya es van facturar 863 milions d'Euros l'any 2005 en aquesta indústria.

Però al darrere de qualsevol empresa de creació de jocs hi ha una gran quantitat de treballadors. Es troben dissenyadors, animadors, provadors de jocs, programadors, artistes i guionistes. Tot aquest grup és necessari només per a crear el joc, ja que després es necessitarà un grup de persones que el comercialitzin i el promocionin.

Un exemple d'empresa d'aquest tipus és *Id-Software*. Aquesta empresa només comercialitza quatre videojocs actualment. Tot i això està formada per trenta-dos components entre els quals es troben vuit programadors i tretze dissenyadors.

Com es pot veure, crear un joc no és una tasca fàcil. És un treball en grup que, en aquest cas, es realitzarà de forma individual. Com a conseqüència, els objectius i les expectatives no són les d'un joc professional, però s'intentarà apropar-s'hi al màxim.

Una altra alternativa (possiblement més factible) és l'adaptació o utilització d'un motor de joc ja existent. Hi ha nombrosos jocs que permeten la modificació total o parcial de les seves possibilitats per tal de crear un nou joc basat en l'anterior. Aquesta opció hauria permès, amb tota seguretat, un joc molt millor, però amb la mateixa seguretat es pot afirmar que l'experiència i els coneixements que s'haurien après no haurien estat el mateix.

També cal advertir de les possibilitats de cada plataforma. És molt diferent crear jocs per a una videoconsola que per a un ordinador. La programació per a ordinador acostuma a ser molt més senzilla.

Totes aquestes possibilitats i alhora dificultats que apareixen creen la necessitat de comunicació entre els equips de programadors i dissenyadors. És per això que periòdicament se celebren congressos sobre aquest tema arreu del món.

1.2 Una mica d'història

Per què es necessita utilitzar un motor tridimensional per a crear un joc? És difícil donar una resposta fàcilment comprensible. Si es fa una mica d'història es veurà clar la necessitat de crear-ne un.

En temps del *MS-DOS*, aquell sistema operatiu únic per a tots els ordinadors de pantalla negra i lletres blanques, el concepte d'ordinador era molt diferent. Tothom utilitzava el mateix sistema operatiu, les mateixes targetes gràfiques, el mateix teclat, el mateix ratolí... Tot eren estàndards. Amb això es vol dir que tot el maquinari funcionava de la mateixa manera i qualsevol petit canvi era impossible.

En aquella època, per a crear un joc cada programador havia de crear el seu motor 3D, un programa que mostrés objectes en perspectiva. També havia de posar-se en contacte directe amb la targeta gràfica, és a dir, comunicar-se sense intermediaris, sense que el *MS-DOS* aparegués pel mig. Era difícil fer jocs ja que es requeria d'un gran nivell de matemàtiques. També és cert que tots els jocs funcionaven en qualsevol ordinador i que no depenien de cap sistema operatiu ni programa.

El problema va arribar amb la creació de targetes més avançades i l'arribada del *Windows 95*. Les targetes més noves, amb funcions més complicades i més resolució trencaven els estàndards establerts. Això va迫car als dissenyadors de maquinari a crear el que ara es coneix com a *drivers* o controladors. El que fan els controladors és informar al sistema operatiu de com ha d'utilitzar cada component de l'ordinador. Si fins aleshores tot això era igual per a tothom, ara havia canviat. Com és lògic, els programadors no podien fer tots els jocs compatibles amb totes les targetes de so, les de vídeo, tots els ratolins, teclats... Per a solucionar-ho el *Windows* va passar a controlar tot el maquinari. A partir d'aquell moment cada cop que s'havia de dibuixar quelcom a la pantalla o detectar un botó que s'acabava de pitjar era el *Windows* l'encarregat de saber-ho i donar-ho a conèixer a tots els programes oberts en aquell moment.

Semblava que el problema estava solucionat, però no va ser així. El fet que el *Windows* ho controlés tot feia que l'ordinador anés més lent. Això, per a la creació de jocs, era una catàstrofe, ja que els jocs anaven encara més lents que abans tot i tenir ordinadors més potents. La solució definitiva seria un sistema que prescindís del *Windows* per a totes les tasques de jocs i que fos compatible amb tot el maquinari que es venia en aquell moment.

Aquí és on va néixer el *DirectX*. El *DirectX* és una llibreria que, integrada amb el *Windows*, és capaç de dibuixar, calcular i controlar tot el maquinari d'una forma ràpida i compatible amb la majoria de models de maquinari. No és l'única en el seu gènere. Altres llibreries van aparèixer amb el temps. La més coneguda i competidora del *DirectX* és *OpenGL*. Aquesta va néixer en la plataforma *Linux*. Cal recordar que *DirectX* va ser creada per *Microsoft* i només està disponible sota *Windows*.

OBJETIUS

L'objectiu d'aquest treball és el de crear un joc 3D. Però per arribar a aquest objectiu i explicar el procés que s'ha realitzat es requereix d'altres objectius.

- Crear un argument per a un joc. Aquest argument estarà basat en un joc històric sobre la Tàrraco romana.
- Explicar els fonaments bàsics del 3D i la seva relació amb les matemàtiques.
- Definir com es juga al joc, el tipus de joc i com interactuarà amb l'usuari.
- Triar un motor gràfic i explicar les funcions bàsiques per a utilitzar-lo.
- Justificar tots els programes emprats.
- Explicar com s'implementa la realitat en la informàtica basant-se en la matemàtica i la física i totes les eines que proporciona l'ordinador.
 - S'implementaran efectes com l'ombra i el cel.
 - Es crearà un motor de col·lisió.
 - Es crearà un motor de càmera i perspectiva.

Amb tots aquests objectius es vol aconseguir un de global que, com ja s'ha dit anteriorment, és la creació d'un joc d'ordinador en tres dimensions.

METODOLOGIA

En aquest apartat s'avaluaran les necessitats que es tindran a l'hora de crear el joc: programes, eines i informació que es necessitarà. També s'exposarà el mètode de treball i els passos que es donaran per a la creació d'aquest.

3.1 Motor 3D i compilador

Com ja s'ha explicat, per a crear un joc es requereix un motor 3D. Aquest permet un accés ràpid a l'ordinador i funcions específiques per al dibuix tridimensional. Com també s'ha dit en la introducció del tema existeixen diferents alternatives. Entre elles destaquen *DirectX* (només per a *Windows* però fàcilment adaptable a *Xbox*) i *OpenGL* (*Windows* i *Linux*, tot i que és portable a qualsevol dispositiu).

DirectX ha estat la llibreria triada. Les raons són diverses; des de la facilitat d'ús fins a la immensa ajuda que es pot trobar a internet, però la més important és que es pot utilitzar sota *Visual Basic 6.0*, que és el compilador que s'utilitzarà en aquest joc.

La tria del *Visual Basic* com a compilador es deu principalment a la facilitat d'ús d'aquest i a l'experiència que es té en aquest llenguatge. Com és lògic crear un joc en un llenguatge que ens és poc conegut significa un increment de la dificultat. Comporta aprendre a programar alhora que es crea el joc, cosa que no és gens recomanable.

3.2 Programes de Modelat 3D

Aquests programes serveixen per a crear escenaris i personatges 3D. Tots aquells objectes inclosos en el joc han estat creats o modificats amb aquests programes.

3.2.1 Autodesk 3D Studio Max 8.0

El *3D Studio Max* és un programa molt potent per a la creació i disseny de 3D. Pot crear des de pel·lícules fins a fotografies realistes totalment inventades. També s'utilitza en la creació de jocs, ja que permet exportar models tridimensionals i carregar-los des del joc que s'està creant.

Bàsicament s'ha utilitzat per a crear tot l'escenari de Tàrraco romana així com els personatges i objectes que apareixen al joc. Ofereix opcions de modelat (tot tipus d'eines, modificadors i, fins i tot, simulacions de física) i sobretot de textures. Més endavant s'explicaran els fonaments geomètrics dels mons tridimensionals.

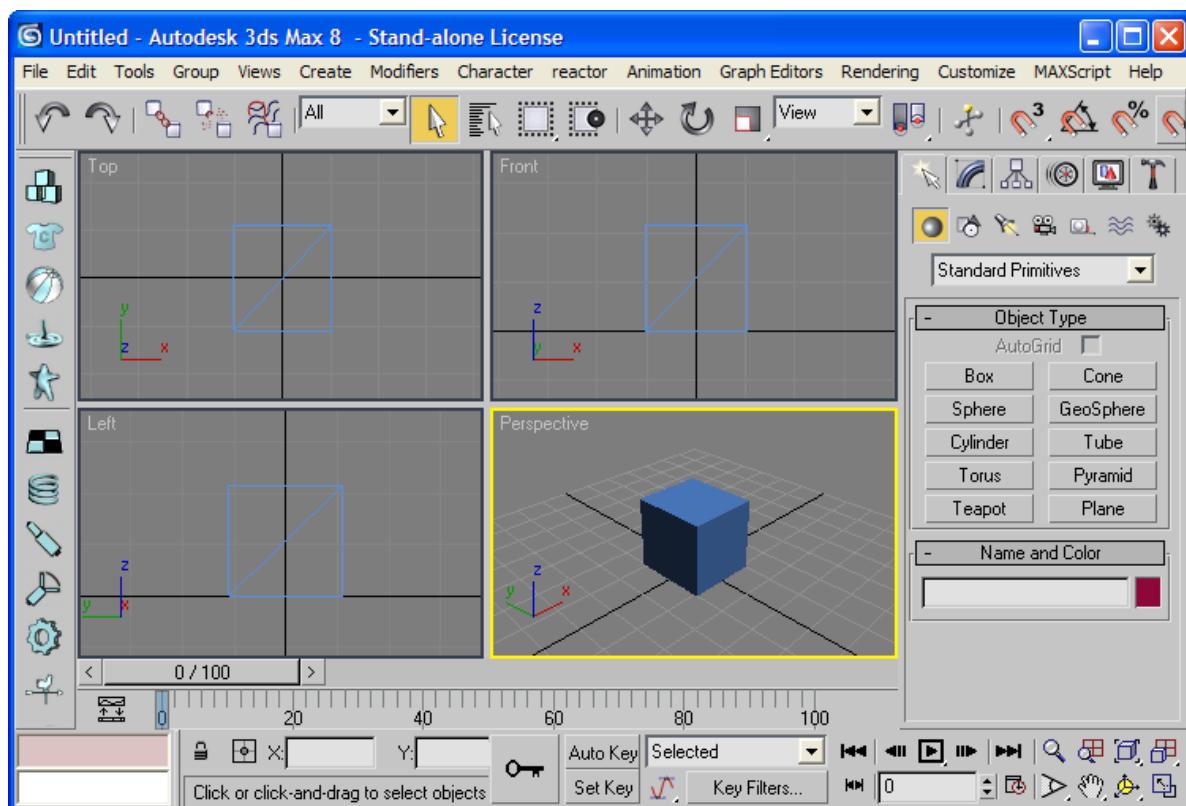


Figura 3.A Finestra principal del 3D Studio Max 8 amb un model d'un cub

Com es pot apreciar s'assembla bastant a la manera de dibuixar clàssica. Una vista tridimensional (axonomètric) i tres vistes (se'n poden triar més) en dues dimensions (dièdric).

3.2.2 Okino PolyTrans 4.1

El *PolyTrans* és un programa molt bo per a la conversió entre formats 3D. Donada la gran quantitat de programes de disseny 3D que existeixen en el mercat aquest programa permet convertir entre aquests formats. A més és ideal per a convertir models des del *3D Studio Max* cap al format de *Microsoft*, que és l'utilitzat en el joc que es vol crear.

Tot i que sembli que la seva funció és fàcil i senzilla no és així, ja que s'ha de passar el model 3D a un format que contingui tota la informació que es necessitarà per a carregar-lo correctament des del *Visual Basic*.

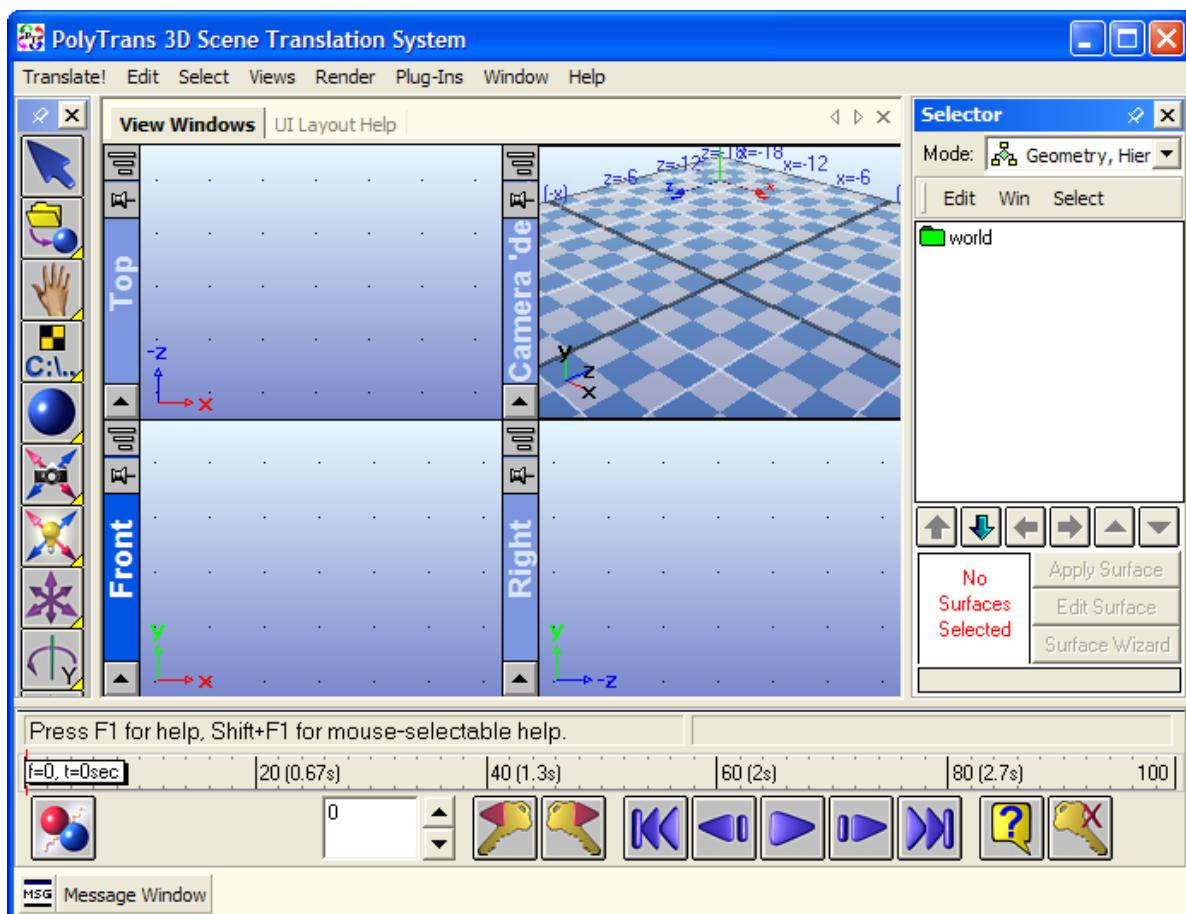


Figura 3.B Finestra principal del PolyTrans

3.2.3 Panda DirectX Exporter 4.8.63 (per a 3DS MAX 8)

Aquesta petita utilitat és un *plug-in** que permet exportar arxius des del *3D Studio Max* cap al format de *DirectX*. Tot i que no té tantes opcions com el *PolyTrans* va molt bé per a exportar animacions (cosa que l'anterior no pot fer) però sempre i quan siguin arxius petits. Combinant aquest programa i l'anterior es pot extreure molta informació dels models del *3DS*. El programa s'utilitza integrat amb el *3D Studio Max*.

* Petit programa que funciona dins d'un altre programa més gran i que n'amplia les opcions i les funcions.

3.3 Programes de creació d'executables o compiladors

Aquests programes serveixen per a crear altres programes. Com és lògic també jocs. Se'n utilitzaran bàsicament dos, que són aptes per a la utilització del *DirectX*.

3.3.1 Microsoft Visual Studio 6.0

El *Visual Studio* és una col·lecció de programes per a programar altres programes (també anomenats compiladors). Inclou, entre d'altres, el *Visual Basic* i el *Visual C++*. El primer utilitza una evolució del BASIC com a llenguatge i el segon utilitza el C++. El fabrica *Microsoft* per a tots els programadors de *Windows*.

El *Visual Basic* és el programa que s'ha utilitzat com a base del joc. Serà el que generarà el programa principal del joc. S'encarregarà de la representació dels gràfics i de la gestió de la memòria i dels recursos. El seu fàcil accés a tots els recursos de l'ordinador el fa ideal per a gestionar tot el joc en si.

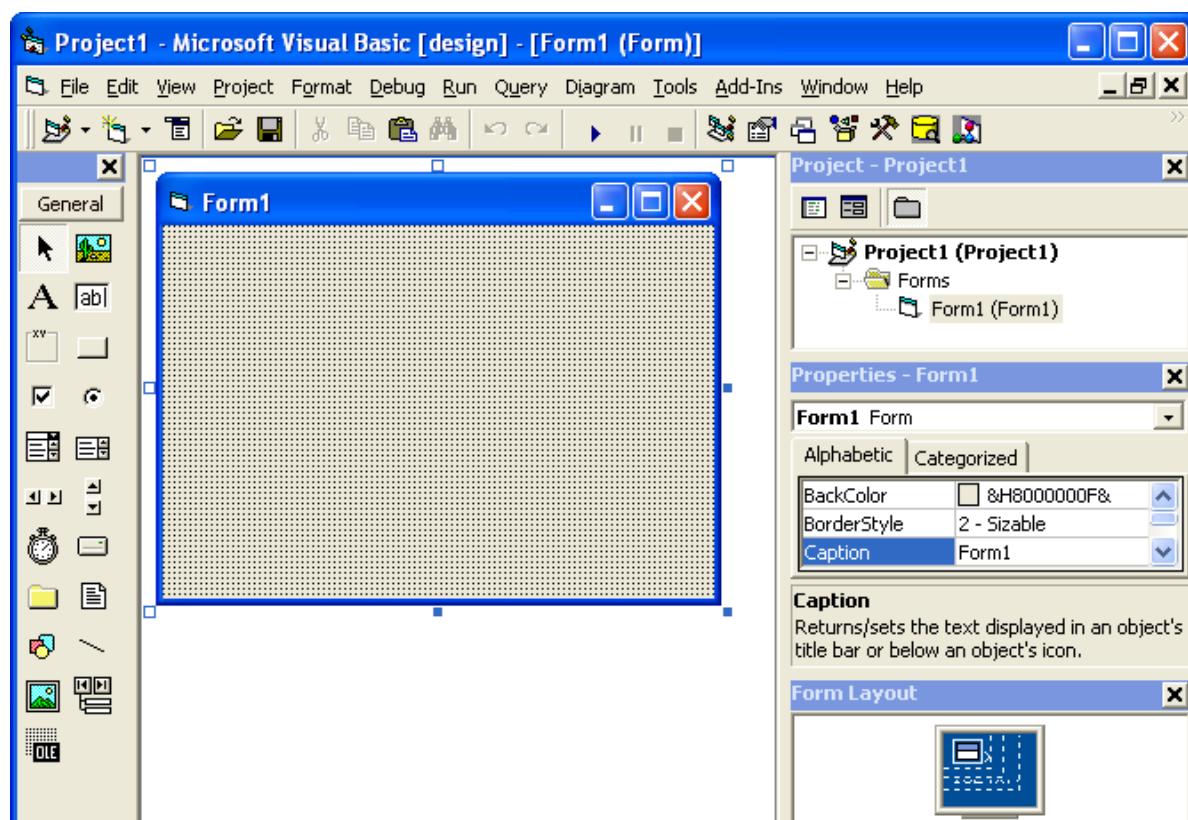


Figura 3.C Finestra gràfica principal del Visual Basic 6

Pel que fa al *Visual C++* s'ha utilitzat per a generar una llibreria auxiliar. El fet és que el *Visual Basic* és lent per al càlcul (s'està parlant de mil·lèsimes de segon), cosa necessària per a calcular la física. És per això que un arxiu extern C++ s'encarregarà de totes les operacions matemàtiques i de física. Així s'obté un programa unes dotze o quinze vegades més ràpid, cosa que el farà jugable en ordinadors més vells.

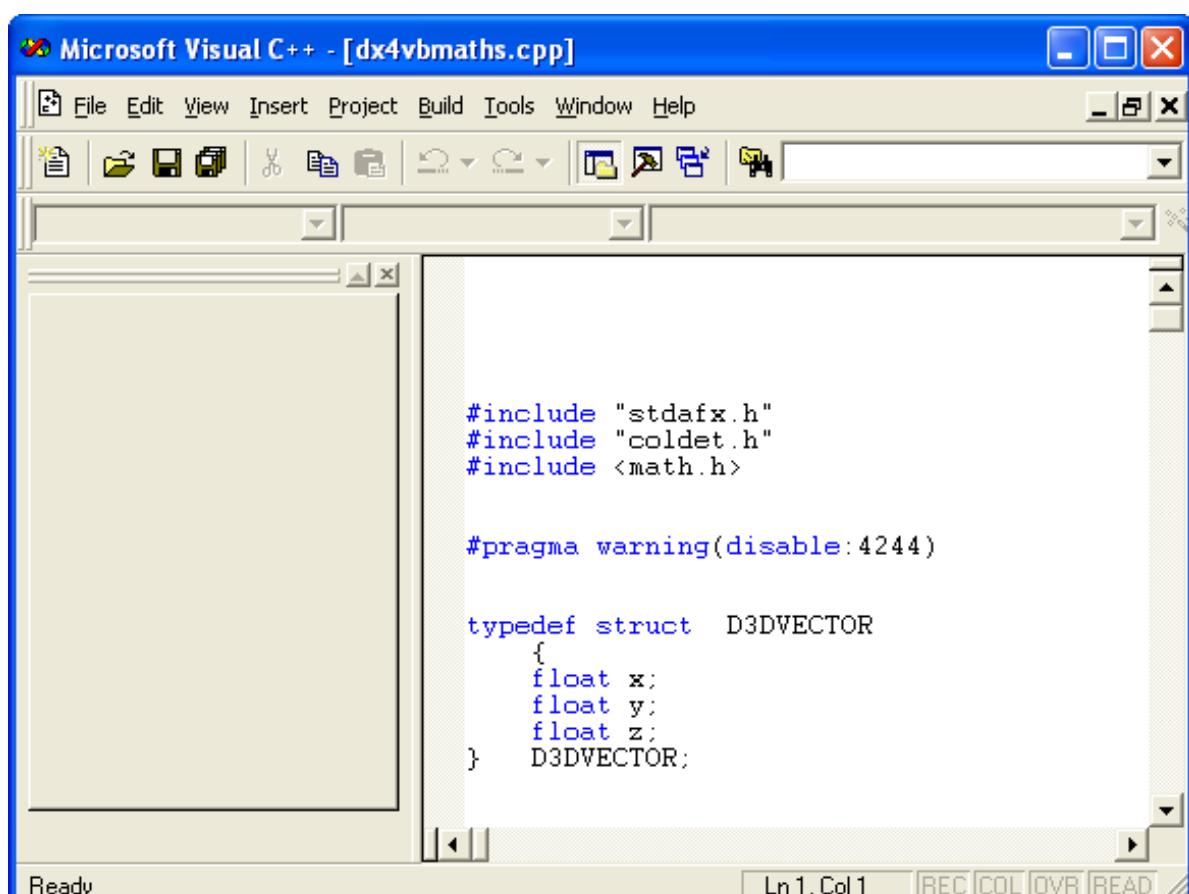


Figura 3.D Finestra de programació principal del Visual C++ 6

3.4 Metodologia de treball

Bàsicament es distingeixen tres parts ben diferenciades del joc. La creació de l'argument, el disseny dels elements gràfics (textures, personatges, món virtual, etc.) i la implementació dels efectes visuals i la interacció amb l'usuari (física, moviment, càmera, etc.).

La metodologia de treball es basa en la divisió màxima de les parts de treball i els processos. Hi ha una màxima en informàtica que diu: *Divideix i venceràs*. Amb això vol dir que per a processar càlculs llargs i repetitius cal dividir-los en subprocessos més petits i autònoms per a unir-los després en un de global. Això s'aplica de la següent forma:

Implementar cada un dels efectes per separat. Abans de crear el joc cal crear petits algorismes independents i autònoms que siguin capaços de realitzar una tasca determinada: dibuixarombres, simular cel, calcular física, etc. Més endavant s'uniran tots per a crear el joc.

Paral·lelament es dissenyarà un món virtual que pugui ser utilitzat de forma correcta en aquests efectes així com es crearà un argument d'acord amb les possibilitats disponibles.

Resumint els passos:

- Crear un argument.
- Crear un món virtual, uns personatges i uns objectes tridimensionals.
- Implementar per separat cada un dels efectes visuals.
- Programar la interacció amb l'usuari també per separat.
- Unir tots els elements anteriors en el joc final.

Però per a realitzar els passos anteriors es requereixen uns coneixements de disseny 3D i de *DirectX*. Aquests recursos s'han obtingut a través de tutorials de *3D Studio Max* i de l'edició de *DirectX* per a programadors (*DirectX SDK*). Per al disseny 3D de la Tàrraco romana s'ha buscitat plànols, vegeu l'**annex D**.

Un cop es disposa dels programes i manuals esmentats ja es pot començar a estudiar cada un dels temes per separat. Abans però s'explicarà la teoria del desenvolupament 3D, que ens és desconeguda.

ELS JOCS D'ORDINADOR

Un joc d'ordinador és un programa informàtic creat amb l'objectiu de divertir i entretenir a l'usuari. Es basa en la interacció de l'usuari amb la màquina per mitjà de recursos (vista, oïda, etc.) i sobre la qual l'usuari ha d'acomplir una sèrie d'objectius donades unes regles i uns recursos inicials. Per tant es pot reduir la creació d'un joc a dues parts: creació de l'argument i desenvolupament del joc utilitzant l'argument creat. A continuació es fa una mica d'història per veure l'evolució dels videojocs en relació a l'argument i els recursos utilitzats.

4.1 Història dels videojocs

Es pot dir que l'inici dels videojocs es va produir sobre un dels primers ordinadors, el EDSAC, amb el joc *Nought and crosses*, que era un tres en ratlla el qual permetia a l'usuari enfocar-se a la màquina. Va ser creat per Alexander S. Douglas el 1952.

Des d'aquest primer videojoc se'n van anar succeint de nous i cada cop més importants. Des d'un ping pong creat a partir d'un oscil·loscopi per a dos jugadors humans, fins a un joc de naus també entre dos jugadors anomenat *Space War*. El joc més famós però, va ser el *Pong*; un joc de ping pong contra la màquina o per a dos jugadors humans que es va instal·lar per primer cop en màquines recreatives. El sorgiment de les primeres videoconsoles domèstiques per part d'Atari, la primera empresa del sector, va fer que els videojocs passessin a formar part de la nostra vida.

A partir del sorgiment de les primeres videoconsoles, cap als anys 80, el món del videojoc es va diversificar. Per una banda hi ha els primers ordinadors domèstics programables que permetien crear jocs: *Spectrum*, *Commodore*, etc. I per l'altra banda hi ha les videoconsoles domèstiques les quals servien exclusivament per a jugar i no permetien a l'usuari crear els seus jocs. Permetien només jugar als jocs que es veien en forma de cartutx. Algunes van ser: *Atari*, *NES* i *Master System*. També es van començar a crear videoconsoles portàtils (com la famosa *Game Boy*).

El següent pas va ser l'arribada dels 16 bits (anys 90). Amb aquesta major potència es van crear videoconsoles i ordinadors que milloraven molt els gràfics. La capacitat d'emmagatzematge va augmentar amb l'arribada del CD a les videoconsoles. Es van començar a veure entorns 3D molt senzills o els anomenats prerenderitzats, és a dir, un entorn aparentment 3D però invariable, ja que només era una imatge de dues dimensions.

Finalment, amb l'arribada dels 32 bits, es va crear el 3D que es coneix ara (a partir del 1995). Es van crear les primeres targetes gràfiques per a ordinadors amb acceleració de gràfics 3D. Aquestes targetes eren especialitzades en el dibuix d'objectes en tres dimensions pel que s'aconseguien jocs tridimensionals en ordinadors poc potents. Aquests processadors gràfics (també anomenats *GPU*) creats pels ordinadors van passar a les videoconsoles creant la *Play Station*, la *Nintendo 64* i la *Dreamcast*. Aquesta generació de videoconsoles es basaven en la millora de l'emmagatzematge (Cd en el cas de la *Play Station* i la *Dreamcast*) i la millora del processador, però seguien essent simples màquines de jugar.

A partir de l'any 2000 van sorgir videoconsoles que incorporaven Sistemes Operatius propis, cosa que permetia ampliar les funcions d'aquesta fins al punt de convertir-se en un ordinador. A més es van ampliar els suports d'emmagatzematge amb l'arribada del DVD. En el camp dels ordinadors es van seguir millorant les targetes gràfiques fins al punt d'aconseguir que realitzessin la major part del treball. Les noves investigacions en el camp han incorporat processadors de física i moviment (*Aelia PhysX*) i, fins i tot, suport per a més d'una targeta en el mateix ordinador (tecnologies *SLI* i *Crossfire*).

4.2 L'argument en un joc

Aquí es podria donar per finalitzat el passeig per la història dels jocs d'ordinador i videoconsoles, però no és només el maquinari ni els recursos dels jocs el que interessa, sinó que es valora tant o més el fil argumental.

Una realitat que es vol deixar clara és que els millors jocs (o els més venuts) de la història no han estat els que tenien el millor argument o uns gràfics extraordinaris. Està clar que això ajuda, però no ho és tot. La prova la tenim en èxits com el (per tots conegut) *Tetris* (figura 4.A). Uns gràfics en blanc i negre a una resolució petitíssima amb un argument nefast: aconseguir punts i punts fins l'infinít. El fet però és que va ser un dels més venuts i jugats.

Amb això tampoc es vol dir que un joc sigui una loteria, al contrari, es vol deixar clar que s'ha de realitzar un esforç i donar un pas més enllà de la qualitat gràfica i del fil argumental. S'ha d'arribar a pensar què farà un joc atractiu i agradable a l'usuari. I si es fa una mica de repàs en la història dels jocs s'observa que han estat moltes les innovacions en aquest camp.



Figura 4.A

El Tetris, una prova que l'argument i els gràfics d'un joc no ho són tot.

Al principi els programadors creaven jocs ja existents, només els passaven a ordinador per dir-ho així. Incorporaven allò que es coneix com a intel·ligència artificial (en el cas dels jocs contra la màquina). I les posteriors innovacions van ser ben petites, ja que els jocs eren sempre en forma de bucle, és a dir, es repetien indefinidament i l'objectiu sempre era aconseguir més punts.

La revolució la va produir un joc que encara avui ens arriba: *Super Mario Bros*. Aquest joc tenia un fil argumental (encara que senzill) que proposava a l'usuari d'arribar al final del joc passant per una sèrie d'etapes on es trobaven escenaris diferents cada cop. Totes les pantalles són diferents, cosa que anima al jugador a descobrir noves etapes. Molts dels jocs posteriors es basarien en aquest com a model d'argument creant així el gènere de jocs conegut com "Joc de plataformes".

Finalment i, amb l'arribada del 3D, els arguments dels jocs es van diversificar. Tornarien els jocs simples i en forma de bucle com els jocs de carreres i de naus espacials i més complicats com l'aventura gràfica, successora dels jocs de plataformes en el nou món 3D.

El món 3D ofereix moltíssimes possibilitats. Cal saber coordinar les possibilitats gràfiques d'aquest amb la qualitat d'un bon argument. La combinació i el conjunt global seran molt més importants a l'hora de determinar l'èxit d'un joc. No es pot avaluar l'argument i les possibilitats de joc per separat, formen part d'un conjunt.

4.3 Creació d'un argument

Crear un argument per a un joc no és senzill. En primer lloc ha de ser sorprenent i atractiu pel jugador, s'ha d'innovar en el seu camp. Ha d'estar en bona combinació amb els gràfics i les possibilitats tècniques; no es pot fer un joc amb mol bon argument i que presenti uns gràfics i un so de baixa qualitat. Ha de ser senzill però alhora complex, és a dir, senzill en qüestió d'utilització i complex en qüestió de possibilitats i recursos utilitzats.

El joc realitzat és del tipus aventura gràfica en tercera persona. Això vol dir que serà un joc que es desenvoluparà en un món virtual tridimensional i que el jugador controlarà un personatge sempre des de fora, en tercera persona. Els motius que han impulsat l'elecció d'aquest tipus són: és ideal per a la idea de l'argument, és molt conegut i fàcil de jugar per a persones que mai hagin jugat a l'ordinador i s'ha aconseguit un motor de física bastant bo i que s'adapta a les necessitats d'aquest gènere de joc.

La idea de l'argument es basa en una persona de Tarragona de l'època actual és transportada a l'època romana i es troba perduda en la Tàrraco del Segle II dC. La persona haurà d'intentar tornar a casa i per a fer-ho haurà de superar una sèrie de problemes que se li plantegen. A continuació s'explica amb detall tots els problemes amb que es troba. Per a veure el guió del joc vegeu l'**annex C**.

En el joc es mesclen vídeos, seqüències tridimensionals i moments de joc. Els vídeos poden ser imatges actuals i reals gravades amb una càmera o creats amb programes de disseny. Les seqüències 3D són parts del joc animades automàticament en les quals no es requereix intervenció de l'usuari.

Es començarà amb un vídeo actual en el qual apareixeran uns adolescents que visiten el circ i el pretori. En fer-se de nit i aproximar-se l'hora del tancament s'amaguen per a passar la nit al pretori.

Un cop sols procedeixen a fer un ritual: el *devotio*. Aquest ritual d'origen romà consistia en el sacrifici d'una persona humana per al bé de la comunitat i mantenir contents als déus. Un dels nois es vol fer el valent i pronuncia les paraules del ritual. En aquell moment es produeix un fet sorprenent, viatja al passat. Aquest viatge el transporta al Segle II d.C. en el mateix lloc on es troba: el pretori. A partir d'aquí comença la seva aventura per la Tàrraco romana.

El seu objectiu serà retornar al present i aconseguir escapar del destí fatal en el qual s'ha involucrat. Amb aquest objectiu l'usuari haurà d'aprendre com funciona la societat romana i, des d'aquesta nova perspectiva, trobar la forma de desfer el ritual i tornar a casa.

El guió del joc és lineal, és a dir, es tracta d'un argument tancat que transcorre sempre de la mateixa forma. En alguns punts es dóna la possibilitat de triar certes coses, però això no influeix per a res en l'argument, només en varia una mica l'ordre.

Vegeu l'**annex C** per al guió sencer del joc.

FONAMENTS DE LA GEOMETRIA

Per a entendre el funcionament intern del 3D aplicat a ordinadors és necessari entendre com funciona la representació d'objectes. Després s'explicarà com modificar aquests objectes en temps real.

5.1 Sistema de representació

El *DirectX* es basa en la representació de triangles. No admet formes rodones ni còniques, només triangles. Però crear un personatge a base de línies de codi és molt difícil i no és viable. Per aquesta raó s'utilitzen els programes de modelat 3D (com el (ja explicat) *3DS Max*). A la figura 5.A es pot veure un model 3D format a base de triangles.

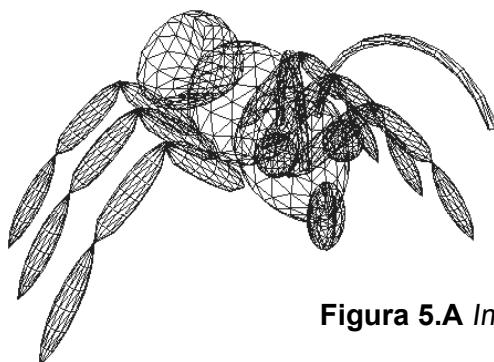


Figura 5.A Imatge d'un model creat a partir de triangles

Els triangles tenen associat un material, que és una petita informació de com seran dibuixats a la pantalla. La figura 5.A mostra com serà dibuixat un conjunt de triangles sense cap material associat. És important veure el treball dels materials; són capaços de “pintar” o omplir els triangles de color. La figura 5.B mostra un model amb materials.

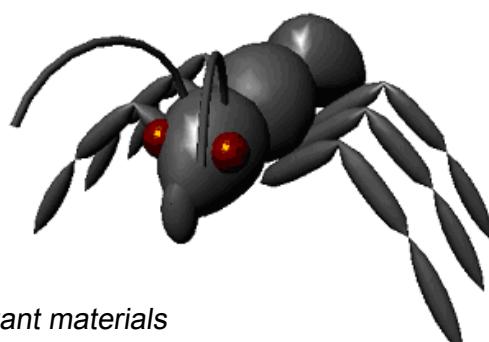


Figura 5.B Imatge del model anterior però utilitzant materials

I no només això, sinó que són capaços de carregar textures. Una textura és una imatge plana (2D) dibuixada a sobre d'un triangle per obtenir més realisme. Si es vol fer un arbre, enlloc d'un material de color verd es pot carregar una textura d'una fotografia d'una fulla i repetir-la de manera que sembli un arbre real.

5.2 Els models en l'espai

Un model tridimensional disposa d'un origen de coordenades. Aquest origen de coordenades és necessari ja que, donat que els vèrtexs són nombres, aquests han d'estar mesurats en referència a un altre punt, l'origen de coordenades. El món virtual que es crearà al joc també té un origen de coordenades, ja que s'ha de poder expressar totes les posicions respecte un punt de referència.

Com és lògic s'ha de poder moure un model dins del món virtual. Com es fa? Doncs canviant el sistema de referència. L'origen de coordenades que tenia fins ara ha de canviar al del món virtual. Això implica canviar un per un tots els vèrtexs del model. I si es vol girar el model? Doncs s'haurà de modificar també un per un tots els vèrtexs. Els càlculs que s'utilitzen per a transformar provenen de lleis matemàtiques ja existents i que s'han aplicat al disseny 3D. La més important és el teorema de rotació de Euler.

La transformació de vèrtexs se simplifica molt amb la introducció d'un mètode matemàtic: les matrius de transformació.

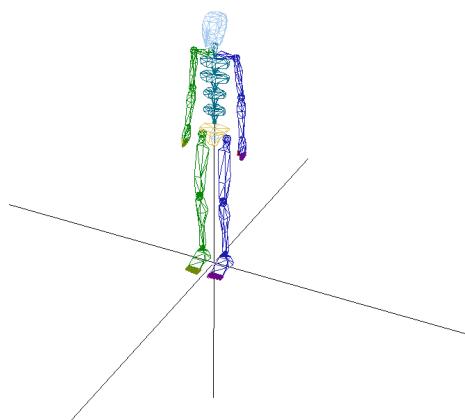


Figura 5.C Un model amb el seu origen de coordenades

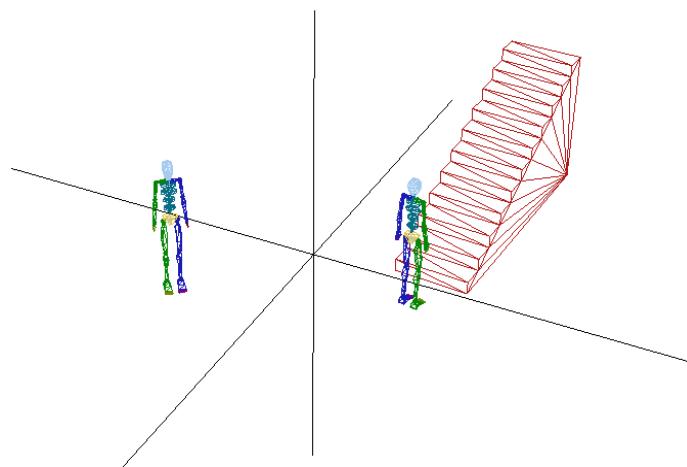


Figura 5.D
Un món 3D amb diferents models. S'ha aplicat un gir a cada model i se'ls ha posicionat en una nova posició

Es defineix una matriu de 4×4 com la de la figura. Aquesta matriu contindrà una transformació per a aplicar a un model.

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix}$$

Per a crear la matriu de transformació se segueix el procediment següent:

Matriu de translació: Desplaça un model en relació a l'origen de coordenades.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{pmatrix}$$

T_x , T_y i T_z representen el desplaçament en direcció dels eixos de coordenades.

Matriu d'escalat: Permet canviar la grandària d'un model.

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

S_x , S_y i S_z representen el factor pel qual s'ha de redimensionar el model. Si està entre 0 i 1 disminuirà la grandària; si és major de 1 augmentarà.

Matriu de rotació: Permet girar un model respecte de l'origen.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

α representa l'angle de rotació al voltant de l'eix **x** en radians

α representa l'angle de rotació al voltant de l'eix **y** en radians

$$\begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

α representa l'angle de rotació al voltant de l'eix **z** en radians

Un cop s'ha creat la matriu de transformació es multiplicarà aquesta per cada un dels vèrtexs del model. Els vèrtexs són representats com a vectors de 4 components:

$$(x, y, z, 1)$$

Es considera la quarta component com a 1. En cas que es treballi amb quaternions les matrius segueixen sent de 4 columnes i files però canvien lleugerament. El procés de transformació d'un vèrtex $(x, y, z, 1)$ utilitzant una matriu és:

$$(x, y, z, 1)' = (x, y, z, 1) \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix}$$

Unes notes abans de continuar. En primer lloc totes aquestes matrius només són vàlides per a *DirectX*; en altres motors com *OpenGL* poden canviar, encara que la funcionalitat sigui similar. També s'ha d'afegir que no hi ha unitats de mesura per a les distàncies, encara que sí pels angles, que són sempre en radians.

Un cop creada una matriu per a cada una de les transformacions es multiplicaran per tal de crear-ne una que contingui totes les transformacions. Aplicar múltiples transformacions als vèrtexs funciona igualment, però és una pèrdua dels valuosos recursos de processament.

Cal tenir en compte que la multiplicació de matrius no és commutativa, és a dir, l'ordre en multiplicar-les és important. Es multiplicarà en l'ordre a aplicar cada una de les transformacions. En cas d'un error es poden produir transformacions estranyes.

Per a expressar tot això en codi s'utilitzen les següents instruccions i objectes que prové el *DirectX*:

D3DVECTOR: Conté tres variables (x, y, z) que representen les components d'un vector.

D3DMATRIX: Conté 16 variables que defineixen una matriu de 4×4 .

D3DMatrixMultiply: Multiplica dues matrius i en torna una altra com a resultat.

D3DXVec3Transform: Transforma un vector multiplicant-lo per una matriu.

D3DXMatrixRotationX: Crea una matriu de rotació al voltant de l'eix x.

D3DXMatrixRotationY: Crea una matriu de rotació al voltant de l'eix y.

D3DXMatrixRotationZ: Crea una matriu de rotació al voltant de l'eix z.

D3DXMatrixScaling: Crea una matriu escalat donats S_x , S_y i S_z .

D3DXMatrixTranslation: Crea una matriu de translació donats T_x , T_y i T_z .

Un cop s'hagi creat la matriu i es vulgui aplicar la transformació al model s'indicarà al DirectX quina és la matriu transformació per a aquell objecte. Acte seguit es dibuixarà el model. La funció que s'utilitza és `SetTransform`.

5.3 Accés a la física

Quan s'implementi la física en el joc es necessitarà una llista amb tots els triangles que formen el món virtual. Per a extreure tota la informació del model s'ha d'utilitzar el concepte d'índex.

Per a explicar què és un índex s'utilitzarà un exemple. La creació d'un cub requereix 12 triangles (el DirectX només treballa amb triangles). 12 triangles són 36 vèrtexs. Si s'observa la majoria dels vèrtexs són repetits, ja que un cub només té 8 vèrtexs. És per això que els models 3D tenen una llista amb tots els vèrtexs utilitzats i una altra que indica quins vèrtexs s'han d'unir en la formació de triangles. Aquesta rep el nom d'índex.

Per aquest motiu si es vol extreure tots els triangles del model s'ha d'extreure primerament els vèrtexs i després els índexs de cada triangle. Una vegada extrets se'ls relaciona per a crear una única llista de triangles.

S'utilitza la funció `D3DIndexBuffer8GetData` per a extreure els índexs i la funció `D3DVertexBuffer8GetData` per als vèrtexs. Un cop es disposa de la informació es fa un bucle i es va creant una llista completa amb tots els vèrtexs. Cada grup de tres índexs es refereix numèricament a la posició dins la llista de vectors dels vèrtexs del triangle. Per exemple: si els tres primers índexs són 4, 5 i 8 estarà indicant que els vèrtexs que conformen el primer triangle són el vèrtex número 4, el número 5 i el número 8. Caldrà recórrer la llista de vèrtexs i trobar-los.

Aquest pas és molt important, ja que un cop es disposa de la llista de triangles pot ser processada en busca de col·lisions del personatge amb el món virtual.

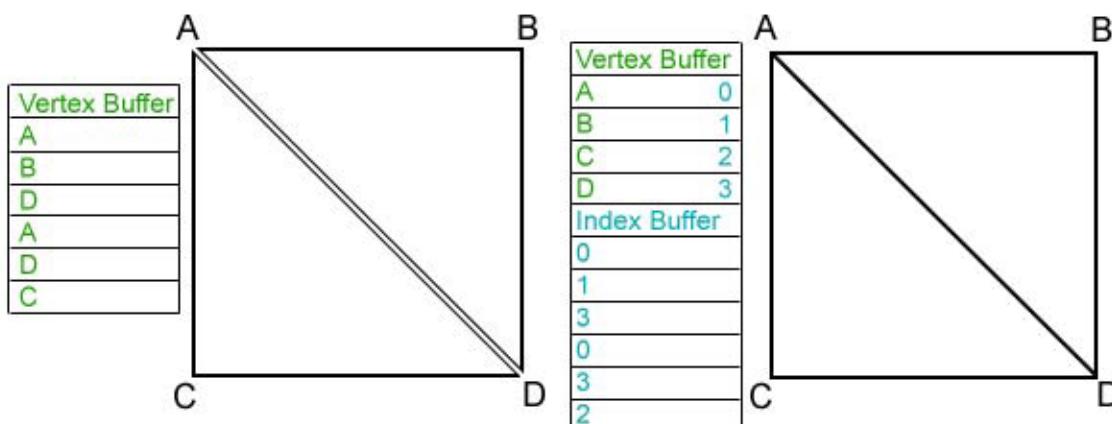


Figura 5.E Diferència entre la utilització de vèrtexs i de vèrtexs i índexs

SIMULACIÓ DE LA REALITAT

En aquest apartat s'explicarà com utilitzar les eines disponibles per a simular la realitat en un entorn virtual. Es parlarà de la gravetat, la col·lisió d'objectes, la perspectiva de la càmera, etc.

6.1 Moviment i col·lisió

El personatge 3D controlat per l'usuari ha de respondre a qualsevol tipus de moviment i col·lisió per part de l'usuari. En un joc en tercera persona com el que s'està realitzant aquest moviment serà sempre sobre una superfície (el terra) que pot ser més o menys plana. A més hi haurà parets, escales i altres obstacles.

Per tant cal pensar que la coordenada **Y** (l'altura) del personatge ha de respondre al moviment vertical aconseguit per mitjà d'escales, als salts que pugui fer l'usuari i a la gravetat de la terra, que l'atraurà sempre. Cal recordar que l'eix **Y** és l'eix vertical (alçada).

No s'utilitza cap principi de dinàmica de Newton, sinó que se supliran amb les equacions de cinemàtica bàsica. És a dir, conceptes com la inèrcia, la gravetat, energia cinètica o potencial quedaran reduïts a la cinemàtica del moviment rectilini. La raó és la major velocitat d'aquestes equacions així com la seva simplicitat.

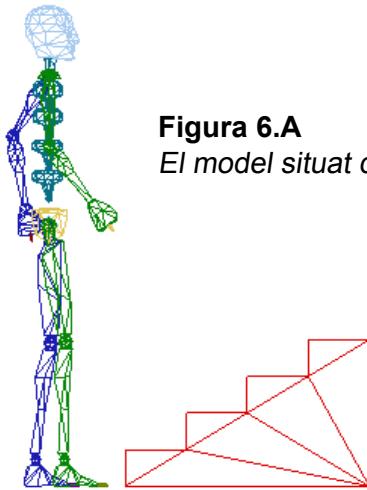


Figura 6.A
El model situat davant d'unes escales

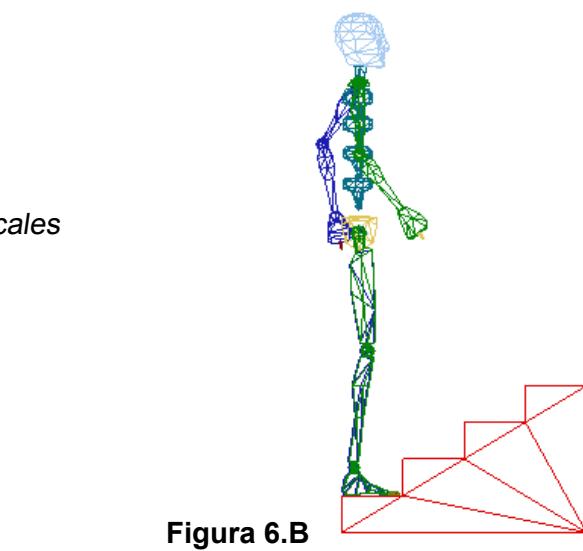


Figura 6.B
En apropar-se puja. Canvia la coordenada Y



6.1.1 Primer problema: moviment vertical

En apropar-se a un desnivell el personatge ha de pujar-lo.

- Situació inicial: El personatge és a punt de pujar una rampa (figura 6.C)
- Sense col·lisió: El personatge manté constant l'alçada i no puja la rampa (figura 6.D). Queda patent la necessitat de col·lisió.

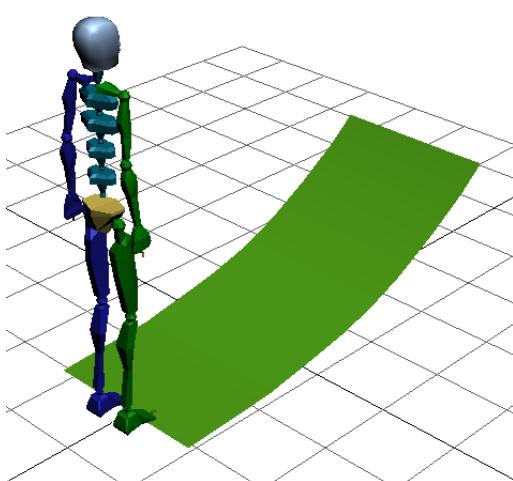


Figura 6.C Situació inicial

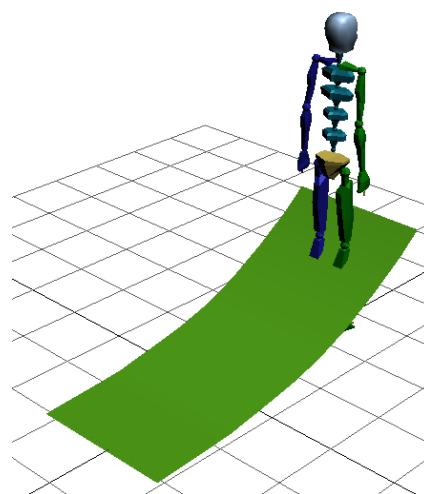


Figura 6.D Sense col·lisió

• Solució proposada

Per a processar la col·lisió es crea una semirecta d'origen el centre del personatge i sentit cap avall. En detectar la col·lisió amb el món es troba el punt d'alçada ideal per a aquesta posició.

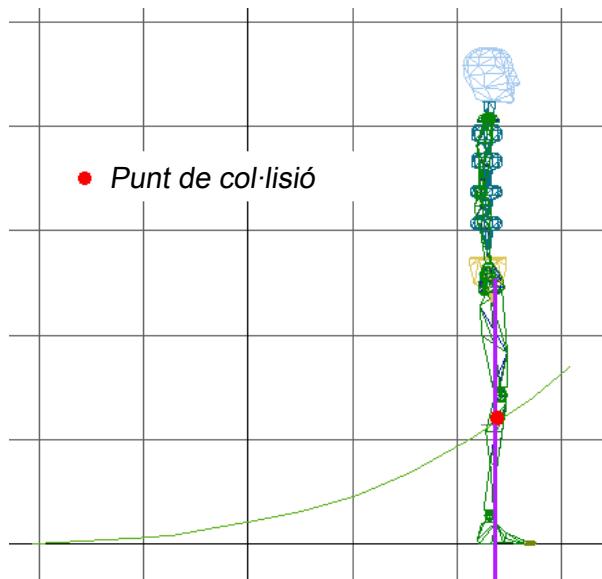
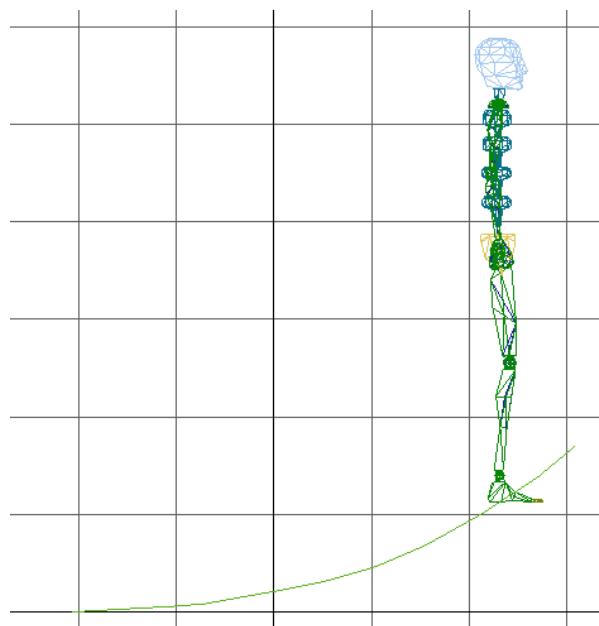


Figura 6.E Detectar el punt de col·lisió

En trobar el punt de col·lisió es processarà l'animació corresponent i s'ajustarà l'alçada del personatge.

S'utilitzarà la llibreria de col·lisió per a calcular la intersecció entre la llista de triangles i la semirecta.



Després d'aplicar el procediment anterior s'obté un moviment vàlid. El personatge ha pujat la rampa (figura 6.F).

A continuació es transforma en codi:

Figura 6.F Col·lisió correcta

```
_declspec( dllexport ) _stdcall hprocess(D3DVECTOR *tri, D3DVECTOR *point,
long numtri, salida *collide) {}
```

Aquesta funció torna el punt de col·lisió donada una coordenada d'origen, un vector direcció (usualment $(0, -1, 0)$) i una referència a la llista de triangles que tenim carregades prèviament a la memòria.

Per a calcular aquestes col·lisions s'utilitza la llibreria *ColDet* (*Collision Detection*) de la següent manera:

```
CollisionModel3D* model= newCollisionModel3D();
```

Es crea un model buit proporcionat per la llibreria.

```
model->addTriangle ();
```

S'omple el model amb la llista de triangles.

```
colisio=model->rayCollision ();
```

Es demana el punt de col·lisió (si existeix) i es torna al programa. Aquesta actuarà mostrant una animació o una altra i canviant les coordenades.

Com es pot apreciar aquest codi està escrit en C++ i es compilarà en l'arxiu *engine.dll*. El *Visual Basic* en serà dependent i l'utilitzarà a cada fotograma per a conèixer l'alçada ideal en aquell instant.

Per al codi detallat i més informació vegeu l'**annex E**.

Ara s'analitzarà la col·lisió amb objectes tals com parets i grans desnivells, així com la càmera.



6.1.2 Segon problema: xocs horitzontals

En apropar-se a un desnivell mol gran (o una paret) el personatge ha de xocar.

- Situació inicial: El personatge es dirigeix cap a un obstacle (figura 6.G).
- Xoc: En arribar a xocar amb l'obstacle el travessa com si no existís (figura 6.H).

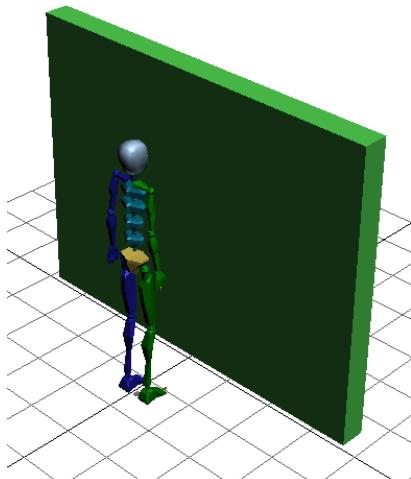


Figura 6.G Abans de xocar

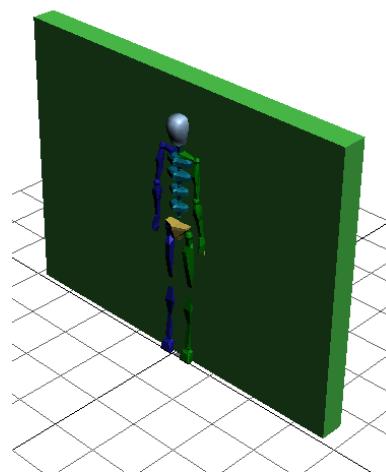


Figura 6.H No es produeix xoc

• Solució proposada

La solució proposada és comprovar la col·lisió entre el personatge i el món virtual. En cas que es produeixi col·lisió simplement no es deixa avançar més enllà de la coordenada actual. Per tant s'ha de guardar la posició de l'anterior fotograma i recuperar-la en cas de col·lisió.

6.1.3 Tercer problema: xocs verticals

Tot i que sembli que l'algorisme és perfecte aquest té un gran error. Si el personatge estigués situat a dalt d'una paret i caigués molt proper a la paret es quedaria literalment “enganxat” a aquesta, ja que detectaria sempre una col·lisió i recuperaria les coordenades **X** i **Z** antigues, que també produirien un error. El fet que les col·lisions verticals i horitzontals es processin per separat dóna lloc a un error.

Figura 6.I: El personatge està situat a la vora d'un gran desnivell com pot ser una paret.

Figura 6.J: Si s'apropa a la vora en el moment que el punt central sobrepassa l'obstacle començarà a caure. Això es produeix pel fet que l'alçada és calculada amb una semirecta.

Figura 6.K: En caure el personatge es queda enganxat a la paret i no en pot sortir, ja que segueix detectant col·lisió i carrega sempre l'última coordenada.

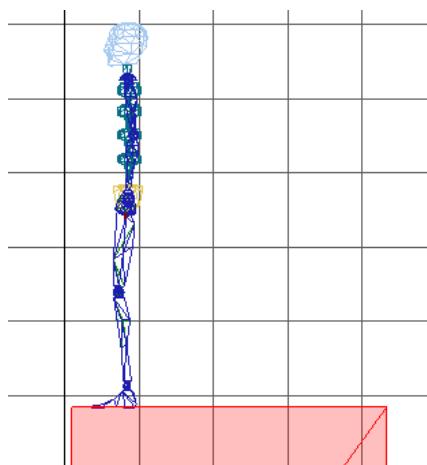


Figura 6.I

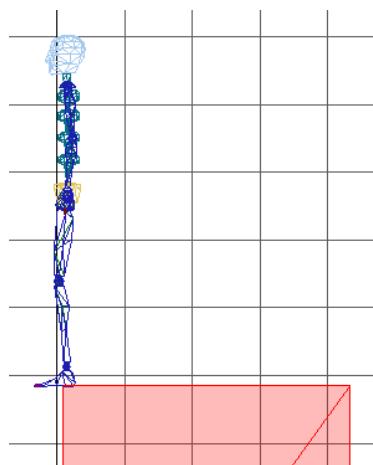


Figura 6.J

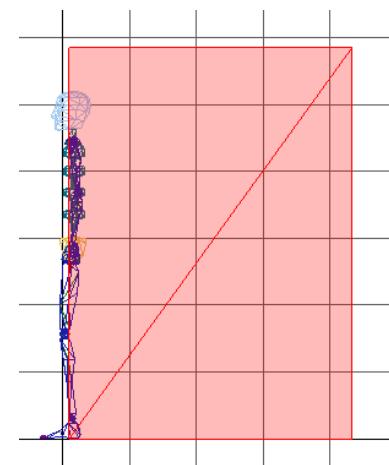


Figura 6.K

- Solució proposada

Per a resoldre aquest problema primerament es comprova si després de detectar la col·lisió per primer cop el personatge segueix xocant amb un obstacle. Si la segona prova és positiva s'haurà d'expulsar el personatge a una distància en la qual no es produueixi una nova col·lisió,

Per a fer-ho s'utilitzarà el mètode següent:

- Calcular el vector normal del triangle amb el que xoca.
- Trobar el punt de col·lisió i aplicar-hi el vector normal.
- Posicionar el personatge en la nova coordenada.

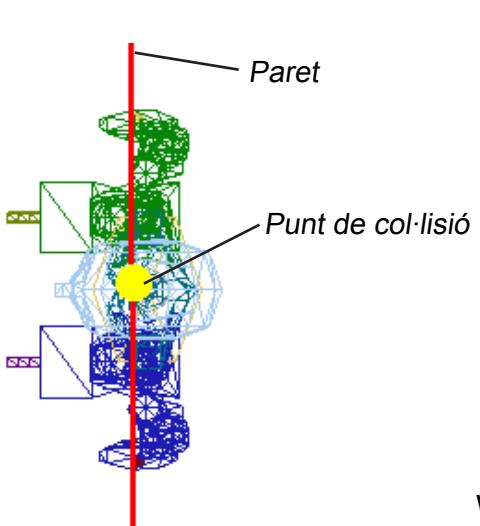


Figura 6.L

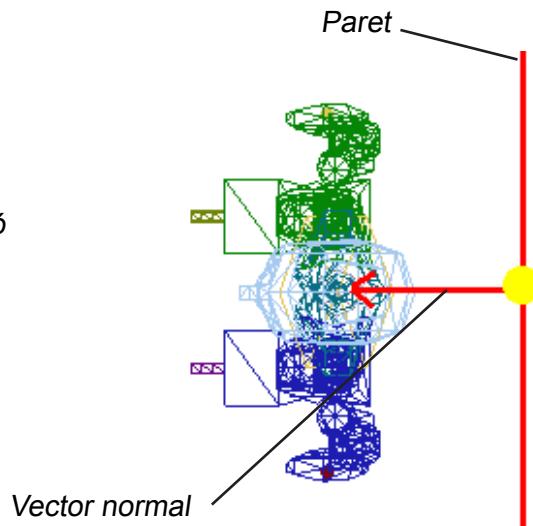


Figura 6.M



En la figura 6.L es pot veure la intersecció entre la paret i el personatge des d'un punt de vista vertical. Un cop s'obté el punt d'intersecció central es desplaça el personatge en direcció i sentit igual al del vector normal del triangle amb que col·lideix (figura 6.M).

- Càlcul matemàtic:

Per a calcular la normal del triangle s'utilitzarà: $\vec{N} = (\vec{B} - \vec{A}) \times (\vec{C} - \vec{A})$

On A, B i C són els vèrtexs del triangle i “x” expressa el producte vectorial.

Per a multiplicar-lo per una distància k es fa el vector unitari (dividir pel mòdul del vector) i es multiplica per k :

$$\vec{V} = \frac{\vec{N}}{|\vec{N}|} k$$

- Codi:

La funció que realitzarà tota la feina està situada també a la llibreria C++. També realitzarà el treball de càmera que s'explica en el següent apartat.

```
_declspec( dllexport ) _stdcall process(D3DVECTOR *cam, D3DVECTOR
*tri, long numtri, D3DVECTOR *pos, D3DVECTOR *pos2, D3DVECTOR *upv,
double distance, double angle, double angleh, float disfromcol, float
interpolationspeed, float midh, float hih, float avrframe, float speed, float
dimensions, long *tris) {}
```

Les variables `cam`, `pos`, `pos2` i `upv` marquen la posició de càmera, la posició del personatge, la posició d'aquest en l'anterior fotograma i un vector que indica l'orientació.

`angle`, `angleh`, `hih` i `midh` marquen l'angle vertical i horitzontal i la distància màxima i mitjana del personatge.

Les variables `tri`, `numtri` i `tris` són la llista de triangles, el nombre de triangles i una llista de nombres auxiliar (passada per referència per major velocitat) respectivament.

`avrframe` defineix la durada mitjana d'un fotograma i `interpolationspeed` la velocitat amb que es desplaça la càmera.

Finalment `distance`, `disfromcol`, `speed` i `dimensions` defineixen la distància de la càmera, la distància d'aquesta i els objectes amb que xoca, la velocitat del personatge en desplaçar-se i les dimensions circulars d'aquest.

6.2 Càmera

Per a crear un joc en tercera persona cal que la càmera estigui mirant el personatge i tot el que l'envolta, és a dir, una càmera exterior.

Aquesta càmera ha de complir els següents requisits:

- Mirar sempre al personatge principal.
- Respondre a l'usuari girant, canviant l'angle de depressió i la distància.
- Detectar el món on es troba i xocar amb les parets, escales i altres objectes.

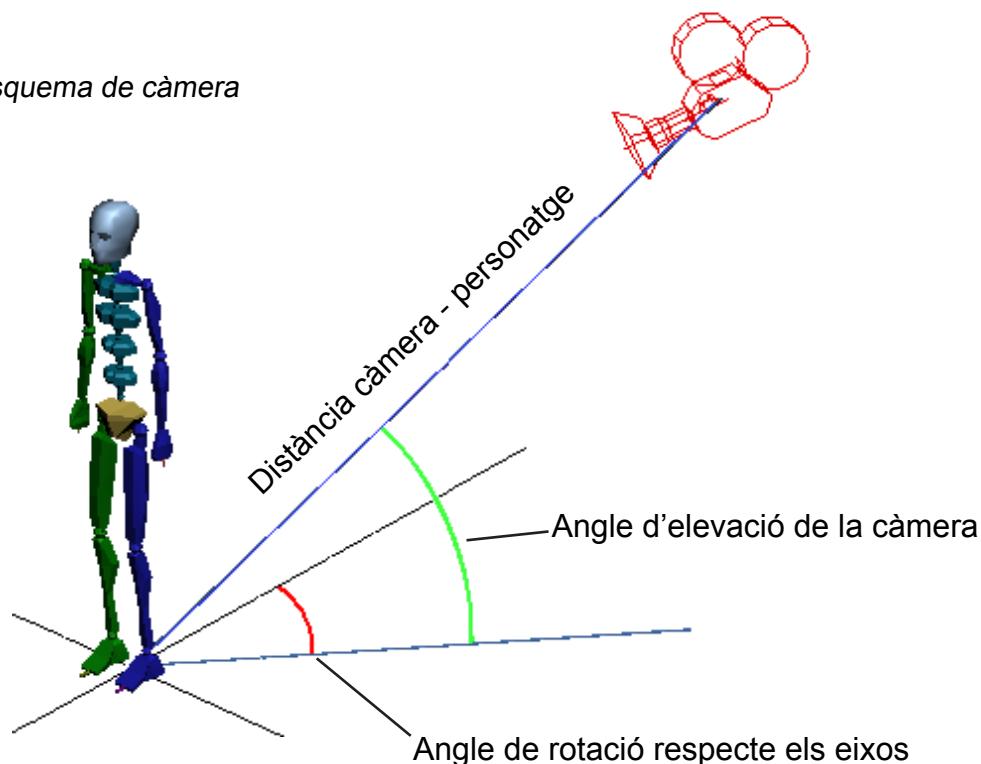
El *DirectX* proporciona la següent funció per a posicionar la càmera:

```
Function D3DXMatrixLookAtLH (MOut As D3DMATRIX, vEye As D3DVECTOR,
vAt As D3DVECTOR, vUp As D3DVECTOR) As Long
```

On **MOut** és la matriu que torna la funció, **vEye** és la posició de la càmera (o ull), **vAt** és el punt on es mira i **vUp** és el vector que marca què és dalt i què és baix.

No hi ha cap problema per a definir el punt on mira la càmera, però si en calcular la seva posició a partir dels dos angles i la distància. El vector **vUp** serà sempre (0,1,0) excepte si la càmera està inclinada cap a baix, ja que si segueix essent (0,1,0) la càmera es girarà. Caldrà de detectar-ho i canviar per (0,-1,0).

Figura 6.N Esquema de càmera



Per a trobar la coordenada de la càmera es realitza el següent càlcul:

$$\text{CoordY} = \text{distància} \cdot \sin(\text{anglev})$$

$$\text{CoordX} = (\text{distància} \cdot \cos(\text{anglev})) \cdot \sin(\text{angleh})$$

$$\text{CoordZ} = (\text{distància} \cdot \cos(\text{anglev})) \cdot \cos(\text{angleh})$$

Utilitzant trigonometria i, coneguts l'angle vertical i horitzontal així com la distància, es pot trobar les coordenades de la càmera. Després hauran de ser aplicades a la posició del personatge.

Però ara es planteja un nou problema: què passa si la càmera se situa a un lloc on traspassa una paret o un objecte? Doncs simplement la càmera s'escapa del món virtual i permet veure zones oclutes, a través de les portes, etc.

A qualsevol joc comercial això no succeeix, perquè la càmera xoca i s'adapta als obstacles. Si xoca amb una paret s'apropa per no deixar veure el que es troba al darrere.

6.2.1 Problema de càmera

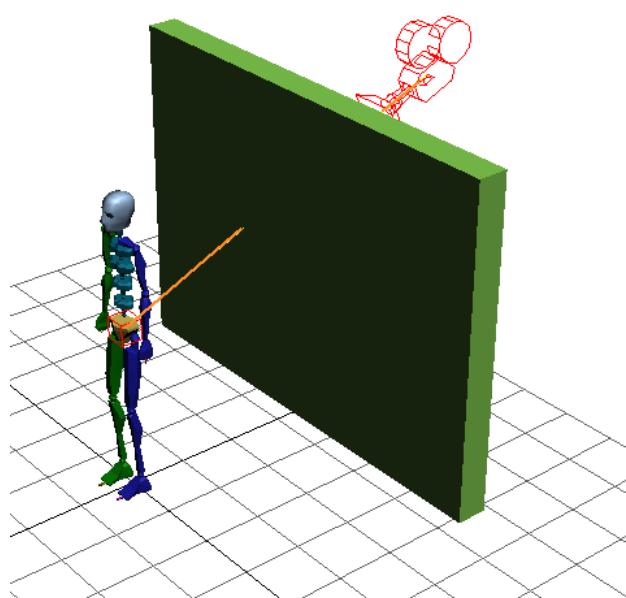


Figura 6.O

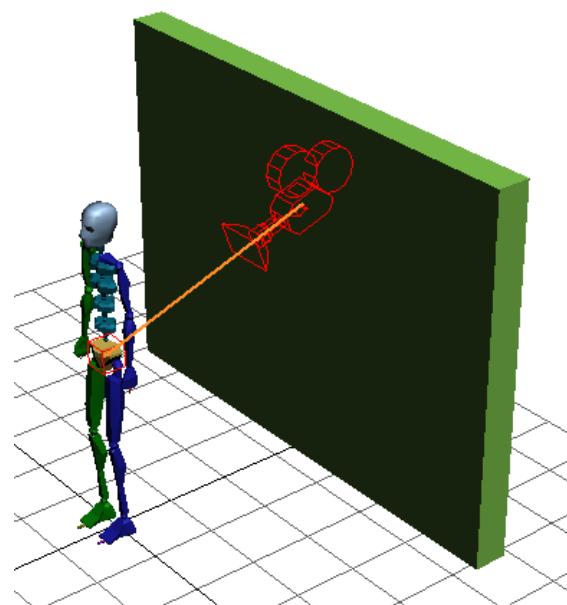


Figura 6.P

Com es pot veure a la figura 6.O l'escena queda tapada per una paret, cosa que no interessa. Per evitar-ho la càmera s'haurà d'adaptar a l'escena (figura 6.P).

- Solució proposada

Per aconseguir el resultat desitjat la càmera s'haurà d'apropar al personatge fins que no xoqui amb cap objecte. Per a fer-ho es pot crear un vector d'origen la coordenada de la càmera i d'extrem el personatge. En detectar la col·lisió amb el món s'obté el punt més proper en el qual no xoca.

Aquest codi s'incorpora en la llibreria C++ i en el procediment explicat anteriorment. A més es guardarà la coordenada de la càmera en l'anterior fotograma i s'interpolarà amb la nova. Amb això s'aconseguirà crear un moviment de càmera suau.

Un cop aconseguides les coordenades i la matriu punt de vista (o *view*) es procedeix a dir al *DirectX* des d'on es mirarà l'escena. És important fer això en començar a dibuixar, ja que si es fa al final o a meitat pel procés de dibuix s'obté un efecte retardat o encara pitjor, errors de dibuix.

Finalment cal afegir que serà la roleta del ratolí o les tecles “+” i “-” les que permetran reduir o augmentar la distància entre la càmera i el personatge.

6.2.2 Construcció d'una vista en perspectiva

També s'ha de definir la matriu perspectiva. Aquesta és la que fa que els objectes s'empeteixin proporcionalment a la distància de la càmera. Per calcular-la es poden utilitzar diverses funcions, però en aquest joc s'utilitza la següent:

```
Function D3DXMatrixPerspectiveFovLH (MOut As D3DMATRIX, fovy As
Single, aspect As Single, zn As Single, zf As Single) As Long
```

On **MOut** és la matriu que torna la funció, **fovy** és l'angle de visió vertical (normalment 45°) i **aspect** és la relació d'amplada entre alçada (útil per a pantalles panoràmiques).

zn i **zf** són els plans que determinen la profunditat mínima i màxima a dibuixar respectivament. Això és útil si el món virtual és molt extens, ja que permet reduir la quantitat de vèrtexs a dibuixar i accelerar el joc. En el joc s'ha incorporat una opció que permet canviar aquesta variable.

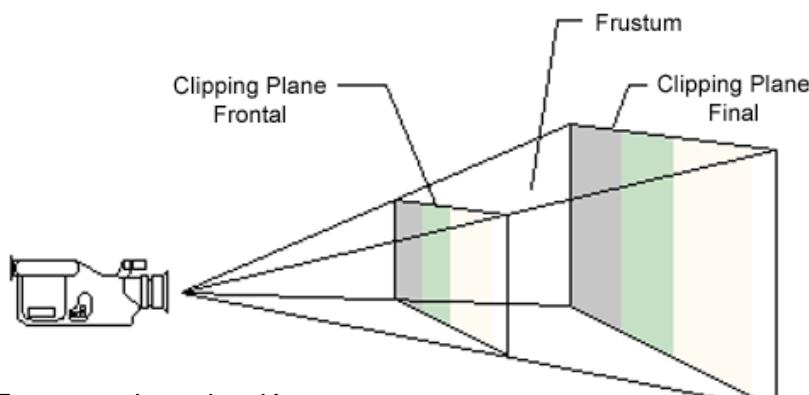


Figura 6.Q Esquema de projecció

6.3 Cel

Per a simular el cel s'utilitzarà una tècnica anomenada *sky box* o caixa de cel.

És una tècnica que s'ha utilitzat des de fa molts anys en el disseny de jocs. Consisteix en crear un cub de només 5 cares (sense la cara inferior) però amb els costats orientats cap a l'interior, és a dir, que les normals dels plans mirin a l'interior del cub.

Es texturitza el cub (ja sigui per codi com s'ha fet en el joc o per un programa de disseny) i es dibuixa aquest cub envoltant a la càmera, de manera que aquesta només vegi el cub.

Però cal tenir present que per que es pugui fer visible la resta d'objectes que estan a fora del cub s'ha de modificar el *buffer* de profunditat. És a dir, s'ha d'enganyar al *DirectX* fent-li creure que la caixa no està situada per davant de tots els altres objectes, però que la dibuixi com si ho estigués. Un cop s'hagi dibuixat la caixa es dibuixarà la resta d'objectes.

En codi seria així:

```
Device.SetRenderState D3DRS_ZWRITEENABLE, 0  
Device.SetRenderState D3DRS_LIGHTING, 0  
    'Dibuixar el cub del cel  
Device.SetRenderState D3DRS_ZWRITEENABLE, 1  
Device.SetRenderState D3DRS_LIGHTING, 1
```

D'aquesta manera el *DirectX* dibuixa la caixa com es desitja i no impedeix la visibilitat d'altres objectes.

Per a texturitzar el cub es necessiten textures que encaixin entre elles. Aquestes es poden trobar a Internet o crear-les amb programes especialitzats.

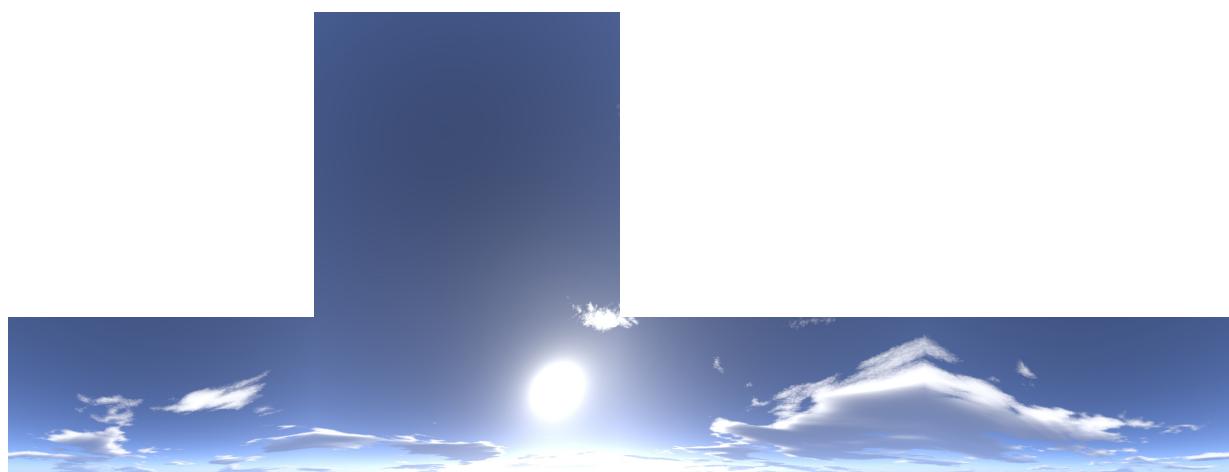


Figura 6.R Desplegament d'un cub texturitzat amb un cel

6.4 Il·luminació

Per a la simulació de la llum del sol, la lluna o qualsevol altre tipus cal entendre els fonaments de la il·luminació 3D.

Existeixen dos tipus d'il·luminació: l'ambiental i la produïda per una llum externa. Es pot trobar altres tipus d'il·luminació o efectes per a simular-la, però els més utilitzats són aquests.

La llum ambiental és un tipus de llum que no té ni direcció ni origen ni qualsevol tipus de component, només valor. Aquest tipus d'il·luminació es produeix a tots els polígons sense importar distància ni orientació. Simplement s'ha de definir quin grau d'il·luminació desitgem. El valor inclou llum per a cada un dels canals de manera que es pot il·luminar amb un to verdós, per exemple.

Les llums externes es divideixen en tres tipus: direccionals, focus i puntuals. El primer tipus es distingeix per tenir direcció i color. Els focus tenen a més posició, angle d'obertura del con focal, abast i atenuació. Les puntuals són focus que no tenen direcció ni angle, ja que il·luminen en totes direccions.

Pel que fa les direccionals es pot dir que van bé per a imitar la llum del sol. La seva trajectòria és sempre paral·lela a la direcció i tenen un abast infinit. Els focus projecten llum des d'un punt amb una direcció i amb una obertura determinada del con de projecció. Les puntuals simplement il·luminen en totes direccions en un abast concret. Pel que fa al factor d'atenuació determina com es debiliten els rajos illuminosos amb la distància. Cal dir que totes les llums poden ser de colors i és la blanca la que imita la llum natural.

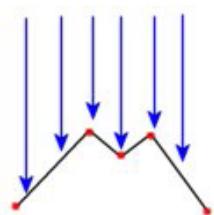


Figura 6.S Llum direccional

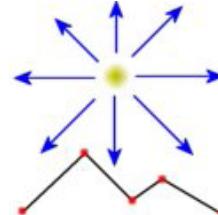


Figura 6.T Llum puntual

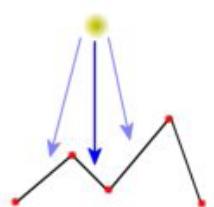


Figura 6.U Llum de tipus focal

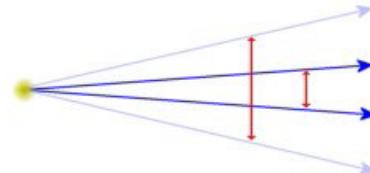


Figura 6.V Con d'obertura d'una llum focal



Una altra cosa important a remarcar és el fet que les ombres no són generades pel DirectX. Pel fet de crear una llum no es creen ombres. De fet, el tema de les ombres és un tema complex i difícil que es tractarà en un altre apartat.

Per a calcular el grau d'il·luminació d'un vèrtex el DirectX calcula la normal de les cares que forma aquest, de manera que la suma d'aquestes normals és la normal del vèrtex (figura 6.W). Un cop ha trobat la normal calcula l'angle que forma amb la llum. Si l'angle és petit, el grau d'il·luminació és màxim. A mesura que creix aquest angle el grau d'il·luminació decreix. En el cas d'una llum perpendicular (90°) el grau d'il·luminació és 0. També cal dir que en cas que la llum il·lumi un triangle de normal oposada (oposat a la llum) aquest no s'il·luminarà.

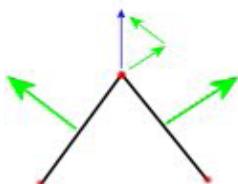


Figura 6.W Càcul de la normal d'un vèrtex

Hi ha casos on no es necessiten llums o molesten per a mostrar una imatge o escena concreta (el cas de menús, objectes especials, etc.). Per això es pot alternar entre el mode sense il·luminació i el mode amb il·luminació. També es poden afegir diverses llums i activar només aquelles que interessin en cada escena. S'utilitzaran les funcions i els mètodes següents:

```
Device.SetRenderState D3DRS_LIGHTING, 1    'llums activades
Device.SetRenderState D3DRS_LIGHTING, 0    'llums desactivades
```

Per afegir una llum utilitzem:

```
Device.SetLight LightIndex, LightStruct
```

Per activar-la s'utilitza:

```
Device.LightEnable LightIndex, 1
```

I finalment la llum ambiental que s'activa de la manera següent:

```
Device.SetRenderState D3DRS_AMBIENT, LightColor
```



6.5 Ombres

Les ombres són un tema molt complex que no es pot tractar en un apartat. De fet són el tema central de tesis i altres treballs. Per aquesta raó només s'explicaran els fonaments bàsics. Si voleu més informació vegeu l'**annex B**.

Bàsicament existeixen dos tipus d'ombres:

Ombres volumètriques (*Shadow Volumes* o *Stencil Shadows*)

Aquest tipus d'ombres es basen en el dibuix del volum que forma l'ombra. L'objecte que tapa la llum projecta un volum que provoca que tots els objectes que estiguin a dins quedin ombrejats. Existeixen dos algorismes per a la implementació d'aquest: el *ZPass* i el *ZFail*.

Ambdós mètodes es caracteritzen per generar un volum (a base de polígons) a partir de l'objecte que tapa la llum. Aquest volum és dibuixat al *Stencil Buffer* com un objecte més. Primerament es dibuixen les cares visibles i després les cares invisibles. Fent l'operació *visible - invisible* només queda a la pantalla l'ombra projectada al terra o a qualsevol objecte que estigui dins del volum. En el *ZFail* es produeix una petita variació en l'ordre de dibuix de les cares amb l'objectiu d'evitar un error quan la càmera entra dins del volum generat per l'ombra.

Mapa d'ombres (*Shadow Mapping*)

Aquestes ombres es caracteritzen per ser totalment fiables en la majoria dels casos. Es basen en el principi que, situats des del punt lluminós, tots els punts que es veuen són receptors de llum i que, qualsevol cosa darrere seu queda ombrejada.

Gràcies a aquest fet Lance Williams, el 1978, va crear un algorisme per a les ombres que solucionava el problema del *ZPass*. Primerament dibuixa l'escena des del punt de vista de la llum i guarda els valors de profunditat. Aquesta escena es transforma al punt de vista desitjat i, en dibuixar l'escena des d'aquest punt de vista, ombreja el *pixels* que tenen un valor més gran a l'enregistrat (més llunyans i, per tant, situats al darrere). El gran inconvenient d'aquest mètode és la baixa qualitat presentada degut que és processat a nivell de pixel.

Després d'analitzar i provar cada un dels mètodes descrits es va arribar a la conclusió que cap d'ells era adequat per al joc. Els motius per no utilitzar les ombres volumètriques és el seu alt cost de processador. Per altra banda els mapes d'ombres eren prou ràpids, però la qualitat era molt baixa ja que eren incapços de cobrir tot el mapa amb un bon acabat.

La solució que es va trobar en aquest cas va ser utilitzar un motor global per a la il·luminació i les ombres. Aquest motor es basa en la utilització de *Lightmaps*.



Lightmapping

Aquest és el nom que rep el procés pel qual la informació d'il·luminació és aplicada a textures anomenades *Lightmaps* i aquestes alhora són aplicades als polígons del model que es vol il·luminar.

Aquest sistema té uns grans avantatges:

- És precalculat, és a dir, no cal calcular la il·luminació a cada fotograma. Aquesta informació es calcula amb un programa anomenat *LightMapper* i es carrega en forma de textures i models 3D. Això fa que sigui molt ràpid en comparació a qualsevol tipus d'il·luminació.

- És d'alta definició. Les llums es calculen amb molts detalls:ombres, reflexos, il·luminació indirecta, etc. El límit de qualitat el determina el creador en un nombre de textures i la grandària d'aquestes. A més uneix el motor d'ombres i el de llums en un de sol i més complet (tot i que no els exclou).

Per contra té un gran desavantatge: el fet que tot sigui precalculat impedeix crearombres i llums dinàmiques realistes. Si s'utilitza aquest sistema lesombres en temps real són encara possibles però poc realistes i pesades. La solució serà doncs crear diferents il·luminacions (per exemple a diferents hores del dia) i alternar entre elles. Això només comportarà un augment d'espai requerit al disc dur de l'usuari i una mica de paciència entre els diferents nivells.

Figura 6.X

Exemple d'una textura de lightmap

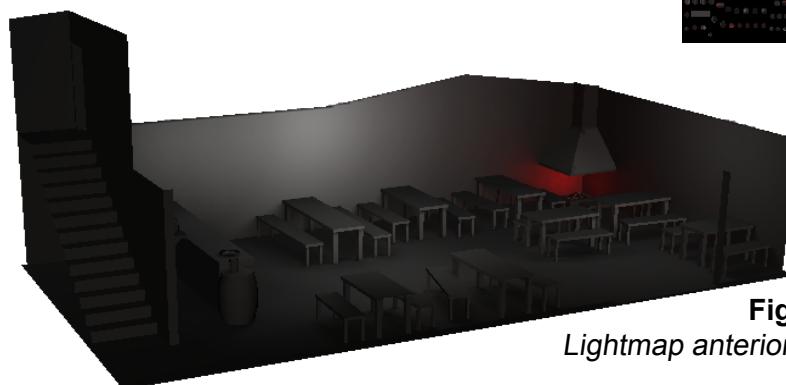
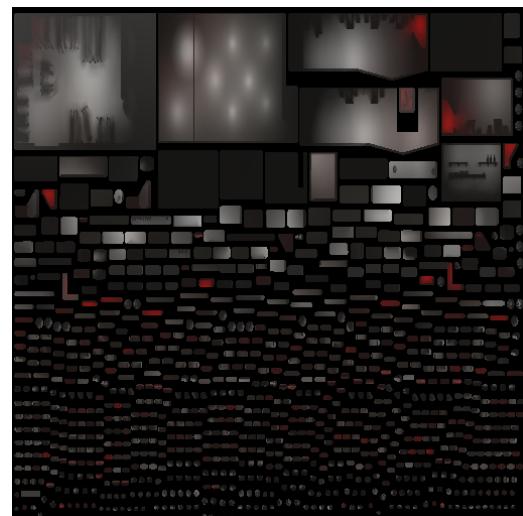


Figura 6.ZZ
Lightmap anterior aplicat al model 3D

MÚSICA I SO

Aquí es tractarà com s'implementa el so en *DirectX*. Primerament es farà una introducció als conceptes bàsics i després es parlarà de la música i del so tridimensional amb més profunditat.

Introducció

Per a reproduir so en *DirectX* es necessita un dispositiu (targeta de so) i un *buffer* de so. La llibreria que s'encarrega de proporcionar tot això és el *DirectSound*.

La tria del dispositiu es pot fer enumerant tots els dispositius i deixant que l'usuari seleccioni el que vol utilitzar. El *buffer* per la seva banda és un espai en la memòria on es carregarà el so sense compressió i des d'on serà reproduït pel *DirectSound*. Cal remarcar la importància que el so no estigui comprimit, ja que no és usual trobar sons als ordinadors d'avui dia que estiguin descomprimits.

A més a més s'ha d'indicar les propietats d'aquest so al *buffer* de so: freqüència, volum, quantitat de bits per segon, etc. Tot això dependrà del tipus de format utilitzat aquí com del mètode que s'utilitzi per a reproduir-lo.

7.1 Música

Abans de començar a explicar com funciona el motor de música cal dir que el format utilitzat en aquest joc és OGG *Vorbis*. Aquest format, desconegut per a la gran majoria d'usuaris, és segurament el més popular actualment en l'àmbit professional. La qualitat de so juntament amb el fet que és completament lliure i que no cal pagar cap tipus de llicència el fan un format molt estès en els jocs actuals.

Per a accedir a la música en aquest format s'han utilitzat les llibreries oficials que es poden trobar a la pàgina web oficial (www.vorbis.com).

En el moment de carregar la música sorgeix un problema. No es pot carregar tota la música al *buffer*, ja que una cançó d'uns cinc minuts ocuparia uns 50 Mbytes de memòria. Juntament amb tota la memòria que s'utilitza en el joc caldria un ordinador molt potent per jugar. Com es pot utilitzar música sense necessitat de carregar-la de cop a la memòria? Doncs utilitzant una tècnica anomenada *streaming*. Aquesta tècnica, que es basa en carregar la música poc a poc i a trossos petits, s'utilitza a internet per escoltar música sense haver de descarregar tota la cançó de cop.

A continuació s'exposa de forma esquemàtica com funciona aquesta tècnica.

Es crea un *buffer* d'una grandària determinada, en aquest cas 600 KB. Aquest es divideix en tres espais. El so serà llegit d'esquerra a dreta i, en arribar al final, es repetirà la reproducció (figura 7.A).

Aprofitant aquest fet es carregarà el *buffer* per zones un cop aquestes hagin estat reproduïdes. D'aquesta manera el *buffer* es repetirà, però la cançó no, ja que s'haurà actualitzat el contingut del *buffer* en cada zona.

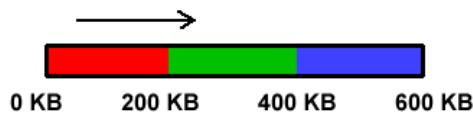


Figura 7.A Esquema del buffer utilitzat en el joc.

Inicialment es carrega el *buffer* amb les tres primeres parts de la cançó (figura 7.B).

Es comença a reproduir cap a la dreta. En sobrepassar el límit del primer bloc (figura 7.C) es pot dir que la primera part de la cançó no es tornarà a reproduir mai més. Per tant és substituïda per la quarta part de la cançó (figura 7.D). Es farà el mateix amb el segon i el tercer bloc. En arribar al final (figura 7.E) el *buffer* estarà carregat amb els trossos 4, 5 i 6 i, donat el fet que el *buffer* es repetirà indefinidament, es reproduiran aquests nous trossos de cançó. Es procedirà de manera anàloga fins al final de la cançó, on s'aturarà el *buffer*.



Figura 7.B

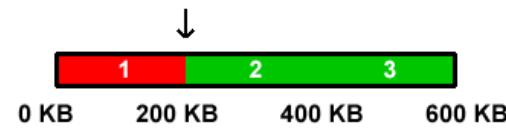


Figura 7.C

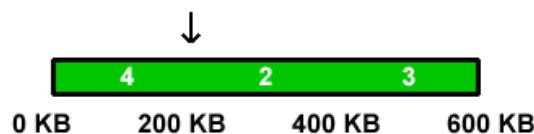


Figura 7.D

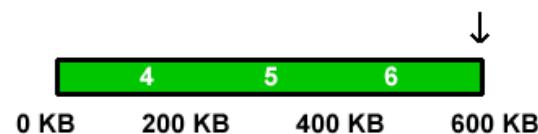


Figura 7.E

Utilitzant aquest procediment es pot reproduir una cançó de llarga durada utilitzant només 600 KB de memòria.

Per afegir aquest mètode al joc s'ha de comprovar a cada fotograma si s'ha sobrepassat alguna zona i omplir-la amb música en cas afirmatiu. D'aquesta manera l'execució del joc i el seu rendiment no es veuen afectats per la música (només s'han de carregar 150 KB de música aproximadament a cada segon).

7.2 So 3D

El so multicanal que ha aparegut últimament al mercat (en forma de jocs d'altaveus que envolten l'usuari) és un tema d'actualitat que abraça des de l'aparell de reproducció DVD del saló de casa nostra fins al món dels ordinadors i els jocs. Un bon so en un joc és essencial, ja que transmet moltes emocions que no es poden captar per la vista o qualsevol altre sentit.

El *DirectSound* té una llibreria que permet simular un so en l'espai. Es basa en la creació d'un objecte que simbolitza la persona que escolta el so i que té com a característiques la posició, l'orientació, la velocitat de percepció, etc. Ara cada vegada que es vulgui emetre un so es podran especificar paràmetres per a que automàticament el sistema emeti el so des d'un altaveu o un altre i aconseguir així sensació d'estar dins del joc.

L'esquema d'emissió de so és el mostrat a la figura 7.F.

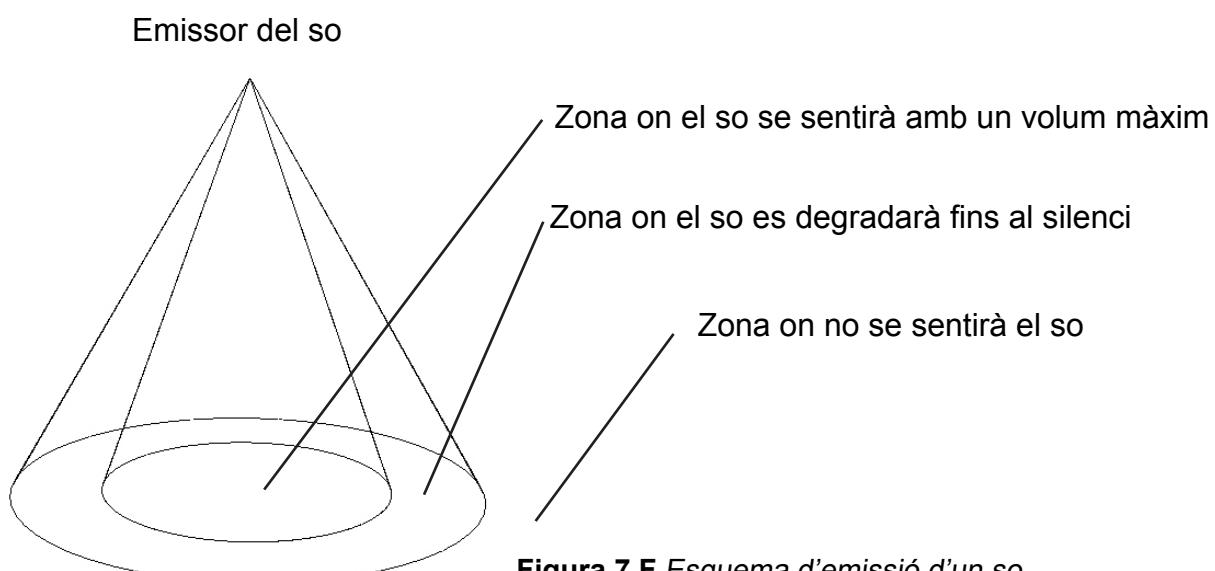


Figura 7.F Esquema d'emissió d'un so

També existeixen altres efectes que no s'utilitzaran en el joc. Aquests permeten la distorsió del so per a provocar nous sons. El motor d'un cotxe accelerant, un objecte passant a gran velocitat i un altre de més lent, etc. Hi ha efectes per a totes les necessitats, però s'ha d'anar amb compte de no requerir massa consum de processador, ja que el so pot ser molt complicat de calcular i mantenir a la memòria.

El receptor del so també té una sèrie de característiques que es poden personalitzar. No s'utilitzarà cap funció d'aquestes, ja que les predeterminades del *DirectX* són les millors per a simular un so percebut per una persona. Aquests efectes es basen en estudis biològics que determinen com les persones percepren el so i que ajuden a millorar el realisme.

OPTIMITZACIÓNS

En aquest punt es tractarà de diferents sistemes, mètodes i algorismes per a reduir el temps de càlcul i augmentar la velocitat del joc. Tot i que això es pugui considerar una cosa opcional o poc important el fet és que el joc que s'ha dissenyat en el moment actual no funcionaria en cap ordinador degut a la quantitat de càlculs que se li exigeixen. És per aquest motiu que l'optimització d'aquest és estrictament necessària.

8.1 Col·lisió

Optimitzar la col·lisió és imprescindible. Cada triangle processat requereix un gran nombre d'instruccions i per tant de temps. Per a reduir el nombre de triangles es realitzen dos processos:

- En primer lloc dividir el món virtual en zones. Les zones estan separades per portes o altres obstacles naturals del món. D'aquesta manera només es comprova la col·lisió amb una part del total dels triangles.
- En segon lloc abans de processar tots i cada un dels triangles es fa una tria. La tria consisteix en descartar tots els triangles que estiguin allunyats una certa distància del personatge. Amb això s'aconsegueix que el càlcul final es redueixi a uns pocs triangles del total d'aquella zona.

Aquesta optimització està a l'arxiu `engine.dll` programat en C++.

8.2 Repetició de textures

En el disseny del món virtual i dels personatges s'utilitzen nombroses textures. De vegades es repeteixen textures o són utilitzades en més d'un model. Si s'hagués de carregar la textura cada cop que es necessités es perdria un temps molt valuós i alhora es requeriria una quantitat enorme de memòria de vídeo.

És per això que s'ha creat un algorisme que abans de carregar les textures carrega els models. Un cop han estat carregats els models el programa busca i carrega un cop cada textura. Cada cop que s'ha de dibuixar una textura es busca la textura dins la memòria i es dibuixa. Amb això s'aconsegueix que si una textura està repetida el programa utilitza només un espai de memòria, estalvant-ne una gran part.



8.3 Compressió de textures

Per a estalviar memòria i augmentar velocitat en el bus *PCI/AGP** es pot aplicar compressió a les textures. La compressió es basa en el sacrifici de la qualitat de textura a canvi d'una velocitat de dibuix molt superior.

El problema és que no tots els ordinadors són compatibles amb la compressió de textures i abans de carregar el joc s'ha de comprovar si l'ordinador ho permet o no. Per al joc s'han utilitzat dos nivells de compressió bàsics:

D3DFMT_DXT1: Comprimeix textures sense transparència.

D3DFMT_DXT5: Comprimeix textures amb transparències.

Per a comprovar si l'ordinador on s'està iniciant el joc és compatible o no s'utilitza la funció següent:

```
Function CheckDeviceFormat (Adapter As Long, DeviceType As CONST_
D3DDEVTYPE, AdapterFormat As CONST_D3DFORMAT, Usage As Long, RType As
CONST_D3DRESOURCETYPE, CheckFormat As CONST_D3DFORMAT) As Long
```

Que torna **D3D_OK** si s'admeten textures comprimides.

8.4 Objectes dinàmics

Un cop creat el món virtual del joc cal afegir objectes. Aquests poden ser des d'una caixa de fusta fins a un arbre, passant per persones, animals, fonts, etc. Aquests objectes no poden anar lligats al món virtual ja que s'han de poder moure o canviar de lloc en funció de l'estat del joc.

És per això que els objectes lliures (o dinàmics) es carreguen per separat i s'uneixen al món virtual mitjançant codi.

Per a optimitzar els objectes es poden utilitzar diversos sistemes, però en aquest joc s'ha creat el següent:

- Un fitxer conté informació de cada objecte: posició, rotació, arxiu del model...
- Cada objecte carregat té dos models: el simple i el complex

En dibuixar un objecte es té en compte la distància entre aquest i el personatge i la visibilitat. Si un objecte està molt lluny no es dibuixa. Si es troba a una distància mitjana es dibuixa un model simple (pocs polígons i per tant més ràpid) i si es troba a prop es dibuixa el model més complex (més polígons i més detall, però més lent).

* El bus PCI/AGP és la connexió que transmet informació entre la memòria de vídeo (situada a la targeta gràfica) i la memòria del sistema i el processador (situats a la placa base). És important que hi hagi el mínim de trànsit possible i a la màxima velocitat per aconseguir un joc ràpid.



Amb això s'aconsegueix que un món amb molts objectes funcioni a una velocitat molt superior a la que es donaria si no s'implementés aquest algorisme.

Els objectes llunyans no es dibuixen i els propers es fan amb poc detall. En apropar-nos-hi més es mostren amb tot el seu detall. De tot això l'usuari no se n'adona, ja que les coses llunyanes es veuen molt petites.

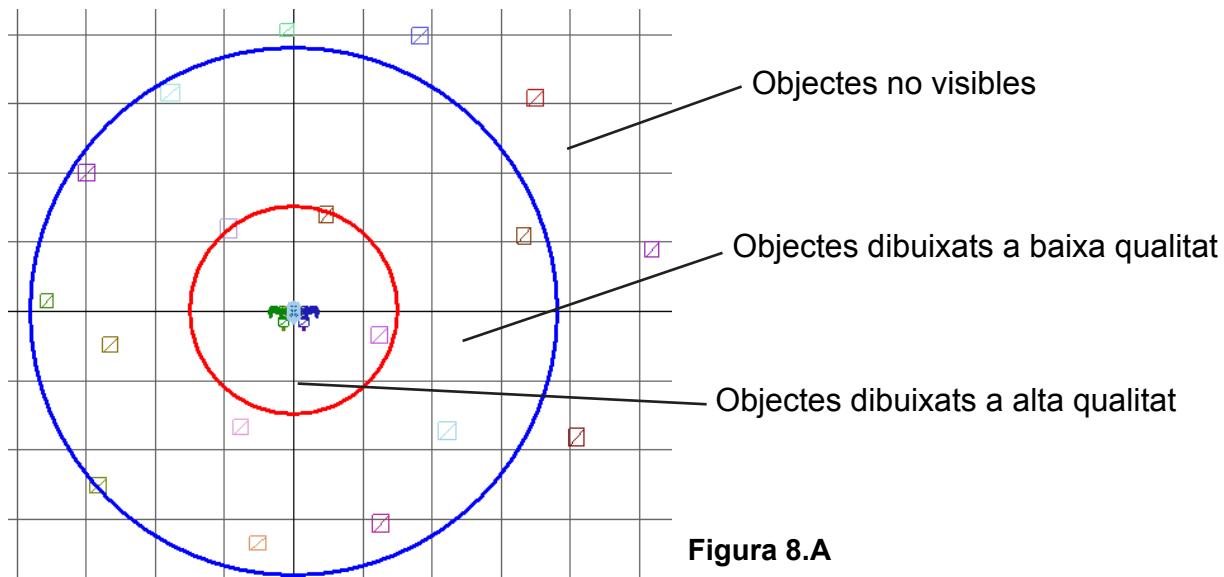


Figura 8.A

Algun programador digué una vegada que els triangles més ràpids a dibuixar són aquells que no es dibuixen. És per això que, encara que es dibuixin triangles que no apareixen en pantalla, en el fotograma final és convenient evitar funcions innecessàries. I és per això també que s'utilitzarà una prova de visibilitat anomenada *frustum culling* (o selecció de visibilitat).

Aquesta prova consisteix en descartar els objectes que queden fora de la nostra visió tot i estar molt propers (com poden ser objectes situats al darrere).

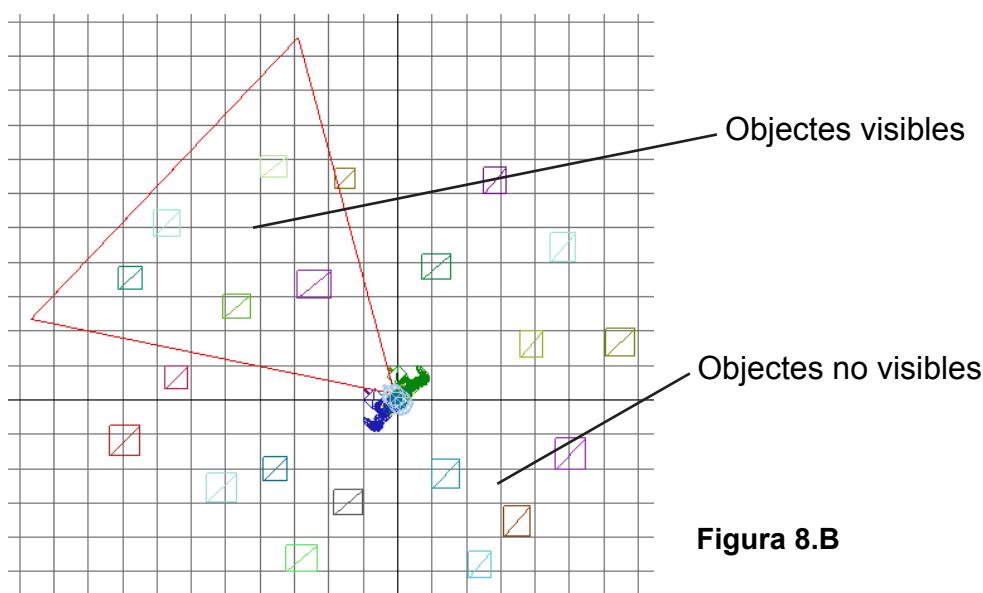


Figura 8.B

Per a calcular aquesta prova de visibilitat, primer de tot s'han de calcular els quatre plans que formen la piràmide de vista. Això es fa multiplicant la matriu vista per la projecció i calculant-ne la inversa. Un cop es disposa dels plans, fent simplement el producte escalar del vector normal d'un pla pel vector d'un punt qualsevol es troba a quin costat està el punt. Si el punt es troba a l'interior de la piràmide formada pels quatre plans voldrà dir que aquell punt és visible.

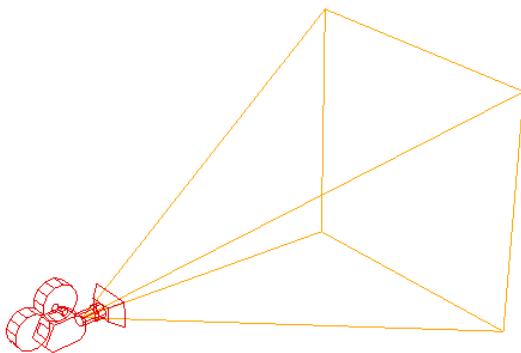


Figura 8.C

A cada fotograma s'ha d'actualitzar l'equació dels plans de la piràmide i comprovar cada un dels objectes. Però, si aquesta prova només comprova punts, com es poden calcular objectes sencers?

Doncs la resposta és creant una esfera circumscrita al model i en calcular el producte escalar se sabrà si l'esfera és a dins, a fora o queda tallada pels plans.

8.5 Adjacència i reordenació de triangles

Finalment cal parlar de l'optimització integrada que ofereix el *DirectX*. Aquest disposa d'una funció que reordena els triangles en funció del material que tenen per a agilitar el procés de dibuix. També disposa d'una opció la qual elimina vèrtexs repetits basant-se en el buffer de triangles adjacents (o *adjacency buffer*).

Aquest *buffer* és un conjunt de nombres que informa quins triangles adjacents té un triangle determinat. Pot ser que no tingui cap triangle adjacent o que en tingui un, dos o tres. Basant-se en això el *DirectX* ajunta, ordena i elimina vèrtexs repetits per aconseguir una velocitat de dibuix molt superior.

Abans de començar el joc s'utilitzarà aquesta funció per a optimitzar al màxim el nostre model de món virtual així com tots els objectes.

```
Function Optimize(flags As Long, adjacencyOut As Any, FaceRemap As Any, VertexRemapOut As D3DXBuffer) As D3DXMesh
```

FINALITZACIÓ DEL JOC

Aquest és l'últim apartat del treball que parla de programació. En aquest apartat s'explica com unir tot el que s'ha explicat en els apartats anteriors i crear un flux de programa que doni lloc al joc que es vol crear.

9.1 Flux del programa

Qualsevol joc està basat en una rutina molt important: el bucle principal del joc. En aquest bucle es processa un fotograma i la repetició d'aquest bucle dóna lloc al moviment de fotogrames. Aquest bucle es divideix en tres parts molt importants: la detecció de l'usuari, el processament físic i matemàtic i el dibuix i els altres efectes (música, so, etc.).

L'entrada de l'usuari és el procediment encarregat de la detecció del ratolí i del teclat per a conèixer què vol fer l'usuari. En funció d'aquestes entrades el procés físic evolucionarà d'una manera o d'una altra.

En el procés físic i matemàtic es calcula el moviment i la col·lisió així com es comproven certes condicions de joc: obrir i tancar portes, parlar amb personatges, agafar objectes, etc. Depèn en gran mesura de l'entrada de l'usuari.

En el procés de dibuix es renderitza tot allò preparat pels processos anteriors i es controla també el so i la música. Les animacions i els efectes especials van aquí també.

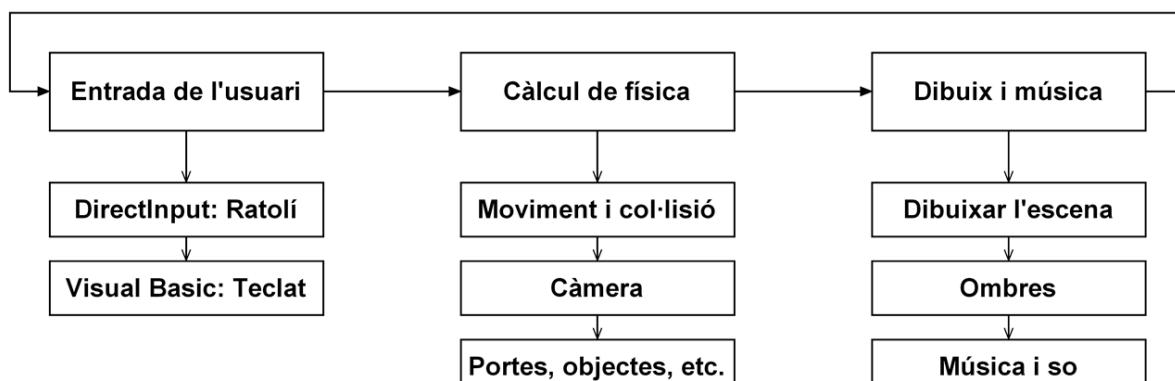


Figura 9.A Bucle principal del joc

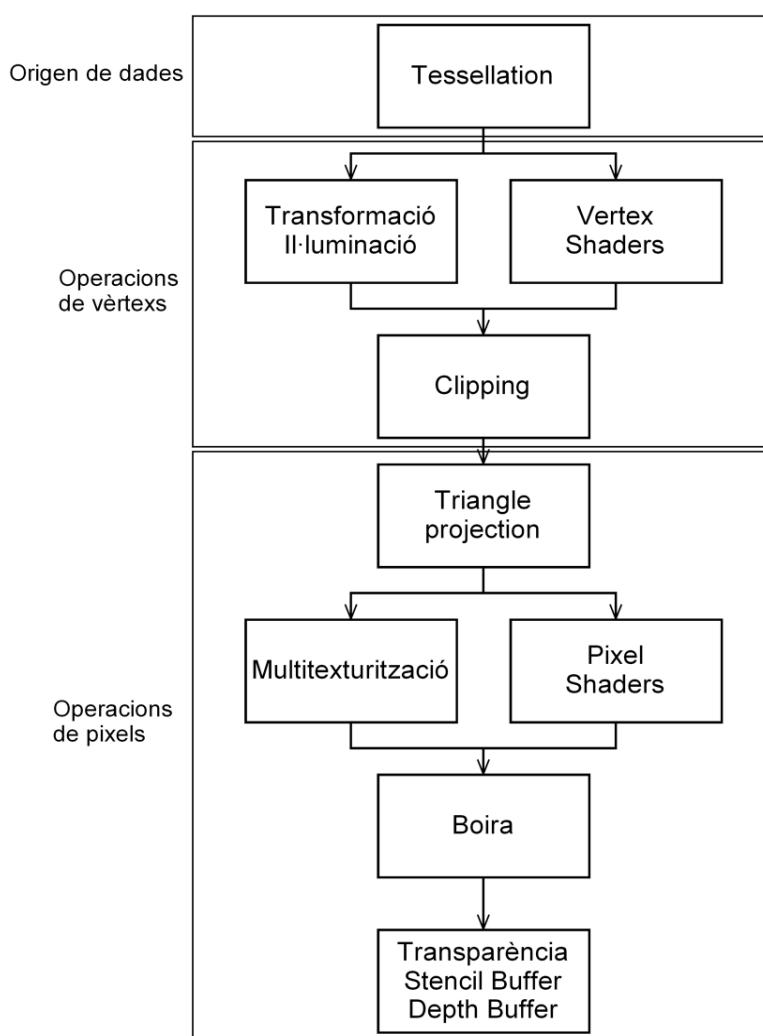


El procés d'entrada de dades es fa a través del *Visual Basic* en qüestió de teclat i amb *DirectInput* pel ratolí. El *DirectInput* és una part del *DirectX* que permet l'entrada de molts tipus de dispositius: ratolins, teclats, *joysticks*, etc.

Finalment cal comentar la funció del rellotge (o *timer*). Cada cop que cal fer un càlcul físic és necessari conèixer l'increment de temps des de l'anterior fotograma. Si l'ordinador processa a uns 75 fotogrames per segon i pot variar molt d'un ordinador a un altre, queda clar que es requereix d'una funció que sigui capaç d'informar del temps mitjà de càlcul d'una imatge. Si es disposa d'aquest valor és molt fàcil calcular moviment d'una manera perfecte. Si el *timer* no funciona bé s'obtindrà un joc que anirà fent "salts". De vegades anirà molt ràpid i de vegades massa lent.

9.2 Dibuix amb Direct3D

L'últim procés que falta per explicar és el de dibuix. A continuació es mostra el procés intern de dibuix del *Direct3D* (part del *DirectX* que s'encarrega del dibuix de gràfics 3D), l'anomenat *Direct3D pipeline*.



En primer lloc es produeix la lectura dels *buffers* de dades corresponents (vèrtexs i índexs).

Aquests vèrtexs són transformats i se'n calcula la il·luminació, així com els *Vertex Shaders* propis. Un cop fet, es talla la geometria amb els plans de projecció (el *view frustum*).

Els triangles són projectats en dues dimensions i es texturitzen. S'apliquen els *Pixel Shaders* propis i altres efectes com la transparència, la boira, etc.

En aquest pas es calcula la profunditat (*Depth Buffer*) que determina quin objecte oculta quin.



- L'origen de dades

Les dades d'origen són bàsicament els *buffers* de vèrtexs, índexs i materials. Com ja s'ha explicat, els primers i els segons defineixen la geometria que s'ha de dibuixar i els tercers defineixen l'aspecte final del dibuix. Les textures són referenciades pels materials, és a dir, els materials no contenen textures, només en contenen una referència a aquestes (usualment el nom d'arxiu). És per això que les textures es tracten per separat.

- Les operacions amb vèrtexs

En aquesta primera etapa els vèrtexs es transformen gràcies a les matrius. La targeta gràfica calcula els nous vèrtexs i aquests passen un procés pel qual són eliminats tots aquells vèrtexs que no es veuen a la pantalla. Es poden realitzar transformacions personalitzades a la targeta gràfica utilitzant la tecnologia *Vertex Shader*. Aquesta permet programar transformacions pròpies molt més avançades. Aquí es calcula també la il·luminació.

- Les operacions amb pixels

Un cop tota la geometria està llesta, es projecten els triangles. Utilitzant la matriu projecció es converteix l'escena en 3D en una imatge de dues dimensions, però amb sensació de profunditat. Es crea l'anomenat *Depth Buffer*, que no és res més que un espai on es guarda la profunditat de cada vèrtex en la imatge bidimensional. La necessitat d'aquest *buffer* és saber quins triangles queden ocults i quins visibles. També es poden crear operacions pròpies a nivell de pixel utilitzant *Pixel Shaders*. Efectes com la transparència i la boira són incorporats a la majoria de targetes actuals i no cal programar un *Pixel Shader* específic per cada un.

En el joc se segueix el mateix ordre de dibuix. En primer lloc es calcula quins objectes han de ser dibuixats. Després es crea una transformació per a cada un i es dibuixen. L'ordre de dibuix és: primer el cel, segon el món virtual, tercer els personatges i els objectes i finalment els efectes especials. Els efectes especials són lesombres i els objectes amb transparències, que requereixen un tractament especial.

En el cas de lesombres el *buffer* amb els triangles preparats és dibuixat al *Stencil Buffer* en lloc del *Depth Buffer* i al *Color Buffer*. Només el *Stencil Buffer* és capaç de realitzar operacions matemàtiques de forma eficaç. Es podria utilitzar una textura per a suprir-lo, però és massa lent. Aquesta tècnica s'utilitza en maquinari que no disposa de *Stencil Buffer*, com per exemple la *Play Station 2*.



9.3 Funcions i codi del Direct3D

Per a dibuixar quelcom es necessita tenir accés a la targeta gràfica. L'objecte que permet accés de forma directa és el `Direct3DDevice8`. Aquest objecte és anomenat dispositiu *HAL* (*Hardware Acceleration Layer*). Amb això es vol dir que utilitza un sistemes d'abstracció per comunicar-se amb la targeta gràfica; és indiferent quin model o marca de targeta s'utilitzi, ja que les funcions seran sempre les mateixes.

Tot i això s'ha d'anar amb compte de conèixer a fons el dispositiu que s'està emprant. No totes les targetes gràfiques tenen les mateixes possibilitats i, per a solucionar-ho, el *Direct3D* proporciona informació sobre les possibilitats de la targeta que s'està utilitzant amb l'estructura D3DCAPS8. Aquesta serà utilitzada en el joc per a determinar si es poden implementar ombres, entre d'altres coses.

Ara que ja s'ha explicat com funciona el *Direct3D* internament, s'explicaran les funcions bàsiques per a programar-ho. Aquestes funcions són derivades de l'objecte `Direct3DDevice8`.

- Funcions de dibuix

`DrawPrimitive`: Dibuixa triangles donada una llista de vèrtexs o un *vertex buffer*.

`DrawIndexedPrimitive`: Dibuixa triangles a partir d'una llista de vèrtexs i d'índexs.

`Clear`: Esborra els *buffers* de dibuix per a dibuixar un nou fotograma.

`Present`: Dibuixa el *buffer* creat a la pantalla per a que l'usuari el pugui veure.

- Funcions de dibuix avançades

`SetTexture`: Assigna una textura als triangles que es dibuixaran.

`SetMaterial`: Assigna un material als triangles que es vol dibuixar.

`SetLight`: Crea una llum a partir d'una estructura `D3DLIGHT8`.

`SetTransform`: Aplica una transformació als vèrtexs donada una matriu.

`SetRenderState`: Funció que permet canviar entre els diferents modes de dibuix. Permet també l'activació i desactivació de nombrosos efectes.

Aquestes funcions s'utilitzen normalment en aquest ordre: `SetRenderState` per a definir les opcions de dibuix prèvies, `SetLight` per a crear les llums necessàries, `Clear` per a esborrar el buffer de dibuix, `SetMaterial` i `SetTexture` per a preparar el dibuix dels triangles, `SetTransform` per aplicar una transformació al dibuix, `DrawPrimitive` i `DrawIndexedPrimitive` per a dibuixar els triangles i `Present` per a mostrar el que s'ha dibuixat en pantalla.



9.4 Resultat final

En aquest punt ja es pot donar per acabat el joc. Tot i no disposar dels models de personatges ni un argument perfectament acabat, el joc funciona perfectament i es consideren per assolits els objectius proposats. El treball que resta per fer es pot considerar com a millora de l'aspecte gràfic d'aquest. Per a veure el codi font final vegeu l'**annex E**, que inclou tot el codi utilitzat tant C++ com *Visual Basic*, i l'**annex F** (el CD-ROM), que inclou el joc final acabat i tot el material utilitzat per a crear-lo.

Ara només cal crear un petit instal·lador (un programa que copiï els arxius del joc a qualsevol ordinador i en permeti la seva utilització correcta) i gravar-lo en un CD-ROM. Un cop realitzat això el joc es podrà distribuir sense cap problema.

A continuació es mostren algunes fotografies finals del joc. Com es pot comprovar, el resultat obtingut és el desitjat.



CONCLUSIONS

En aquest punt el treball ja està finalitzat. Ha arribat el moment de revisar els objectius i extreure conclusions. Com és lògic, la visió inicial que es tenia del tema ha canviat de forma radical.

En primer lloc el projecte ha crescut d'una forma excepcional i s'ha allunyat del tot de la idea inicial. En un primer moment es pretenia un projecte molt més senzill que impliqués uns coneixements mínims de 3D i de programació. Però donada la gran ambició d'aconseguir un bon joc, el projecte s'ha convertit en un joc que es podria catalogar de quasi professional. La falta de temps ha estat un factor que ha produït alguns buits en el projecte final, que caldrà rematar posteriorment.

Per altra banda els problemes que es preveien inicialment han estat superats amb facilitat alhora que se n'han creat més en ampliar el joc. Tots però, han estat superats amb més o menys èxit.

Les conclusions a les que s'ha arribat són:

El *Visual Basic* no és apte per a fer jocs en 3D. En primer lloc per la seva limitació al *DirectX 8.1* i per altra banda el seu escàs potencial.

Abans de començar a fer un joc cal definir primer l'argument i conèixer els límits que es té. En el cas de fer-ho tot a l'inrevés (com ha succeït) el resultat és igualment bo, però molt més costós. El fet de no conèixer tot el món 3D i els propis límits provoca haver de canviar molts cops el codi font. És el preu que s'ha de pagar per aprendre d'una forma autodidàctica.

Al marge del tema, una conclusió molt important és que per a aprendre a fer jocs cal fer-ne molts. L'experiència en programació és molt més important que els coneixements.

És també molt important la divisió del treball. Està clar que ser polivalent i conèixer diferents àrees és beneficis, però un cert grau d'especialització és també necessari. Per aquesta raó aquest treball s'hauria d'haver realitzat en un petit grup, on cada membre del grup conegués una àrea amb més profunditat (programació, disseny, so, guió, etc.).

Finalment només cal afegir que la tria del *Visual Basic* ha estat donada pel fet que era l'únic llenguatge que es coneixia en profunditat. Un treball fet en C++ hagués estat més un tutorial per aprendre a programar en C++ que no pas el disseny d'un joc. Totes aquestes conclusions es tindran en compte en el pròxim joc que es faci.

VALORACIÓ

La valoració global del treball és molt positiva.

Durant aquest treball probablement he après moltes més coses que en tota la meva vida de programació. El fet és que dedicar un treball de l'escola a un tema que t'agrada provoca una major dedicació i un major esforç en aquest. I és això el que m'ha succeït: he pogut demostrar tots els meus coneixements de programació, matemàtiques, física, disseny, etc. d'una manera molt agraïda i pràcticament sense pensar que era una obligació.

M'he sorprès de ser capaç de realitzar una tasca tan gran en tan poc temps. De fet, un dels motius que em va impulsar a fer un joc pel treball de recerca, va ser precisament això: sabia que si no m'obligava a mi mateix a fer-lo, no el faria mai. Com que ja em coneix i sé que mai acabo el que començo preferia crear algun tipus d'obligació per a acabar-lo.

En referència al treball de recerca en general penso que està molt bé. Per una banda és una altra forma de demostrar el que has après, ja que es requereixen coneixements de l'escola, i alhora permet la tria d'un tema que ens agradi i estigui dintre de les nostres possibilitats.

Per finalitzar m'agradaria dir que mai m'ho havia passat tan bé fent un treball per l'escola, i trobo que això suposa una gran motivació per part de l'alumne.

BIBLIOGRAFIA

Pàgines webs utilitzades per a la realització del codi, la programació i la redacció:

- GameDev: <http://www.gamedev.net>
- DirectX4Vb: <http://directx4vb.vbgamer.com>
- Visual Basic Gamer: <http://www.vbgamer.com>
- ColDet: <http://coldet.sourceforge.net>
- FreeImage: <http://freeimage.sourceforge.net>
- Planet Source Code: <http://www.planet-source-code.com>
- FlipCode: <http://www.flipcode.com>
- El Guille: <http://www.elguille.info>
- OGG Vorbis: <http://www.vorbis.com>
- NVIDIA developer: <http://developer.nvidia.com>
- ATI developer: <http://ati.amd.com/developer>
- Gamasutra: <http://www.gamasutra.com>

Pàgines web utilitzades per a la redacció del treball:

- Wikipedia English: <http://en.wikipedia.org>
- Wikipedia Español: <http://es.wikipedia.org>

Llibres utilitzats per a la programació del joc:

- Game Programming Gems 2
Autor: Mark DeLoura
(ISBN: 1-58450-054-9)
Editorial: Charles River Media
- Game Programming Gems 4
Autor: Andrew Kirmse
(ISBN: 1-58450-295-9)
Editorial: Charles River Media