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

Secció Pàgina

Annexos

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 forçar 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*.

Objectius

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 móns tridimensionals.

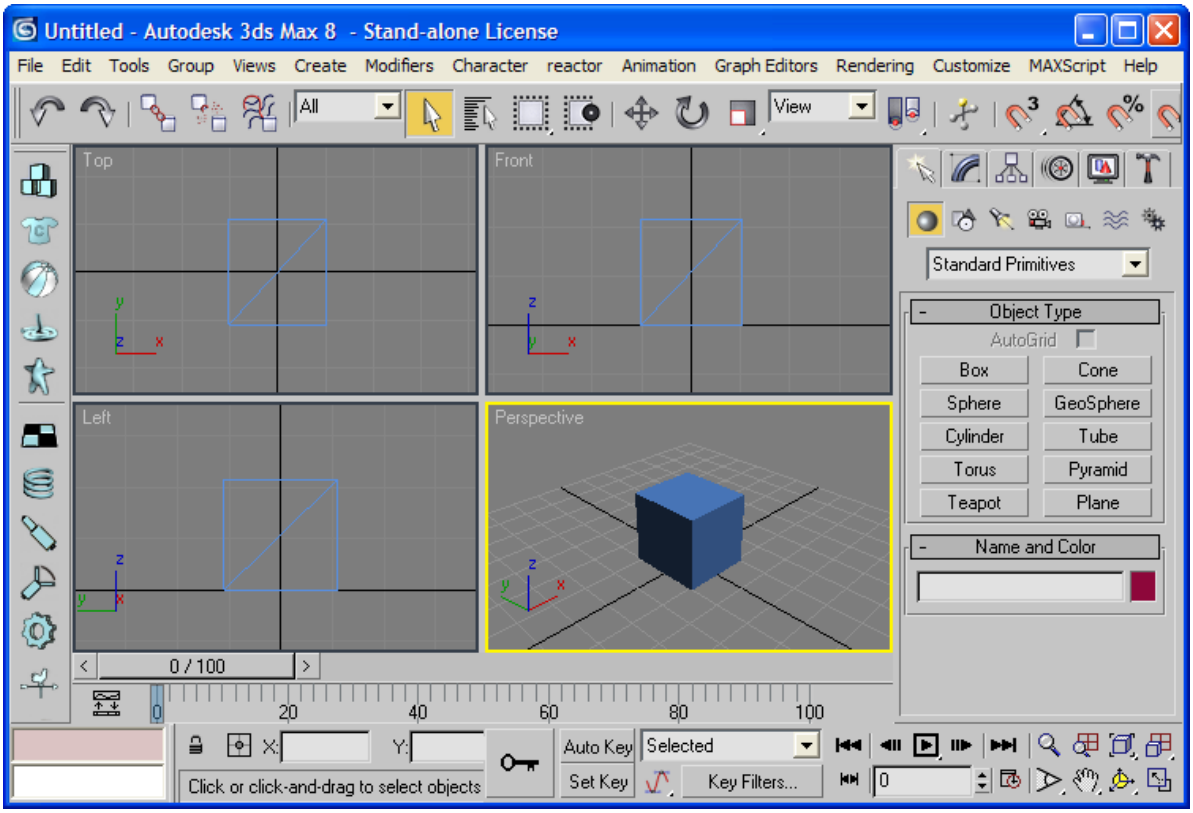


Figura 3.A *Finestra principal del 3D Studio Max 8 amb un model d'un cub*

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*.

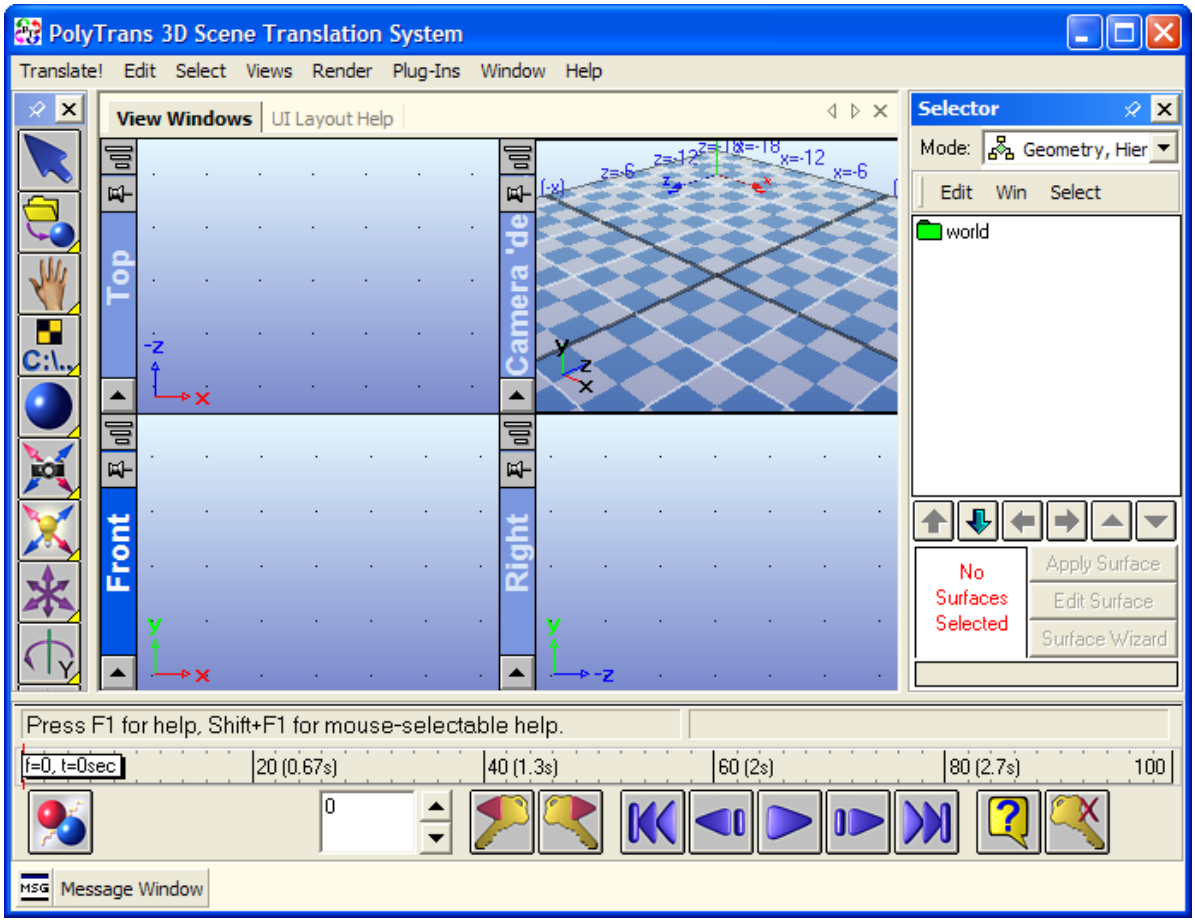


Figura 3.B *Finestra principal del PolyTrans*

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.

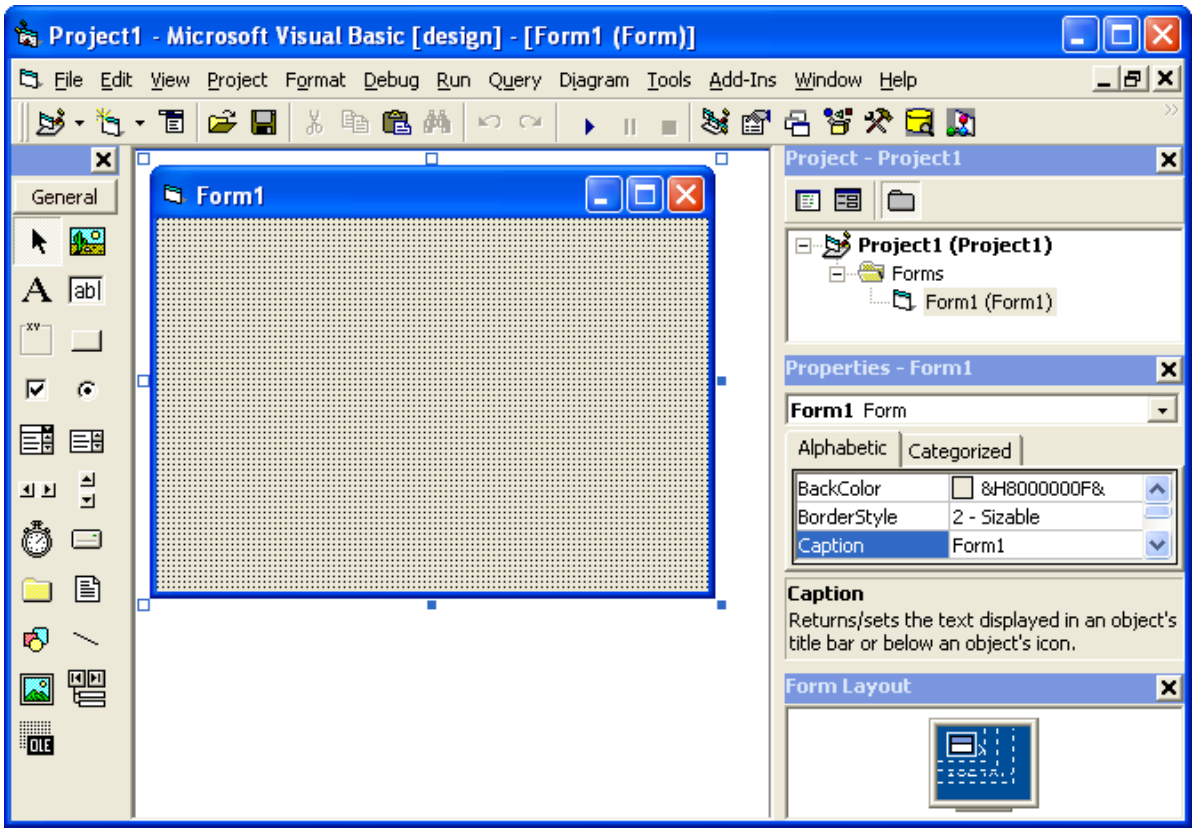


Figura 3.C *Finestra gràfica principal del Visual Basic 6*

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.

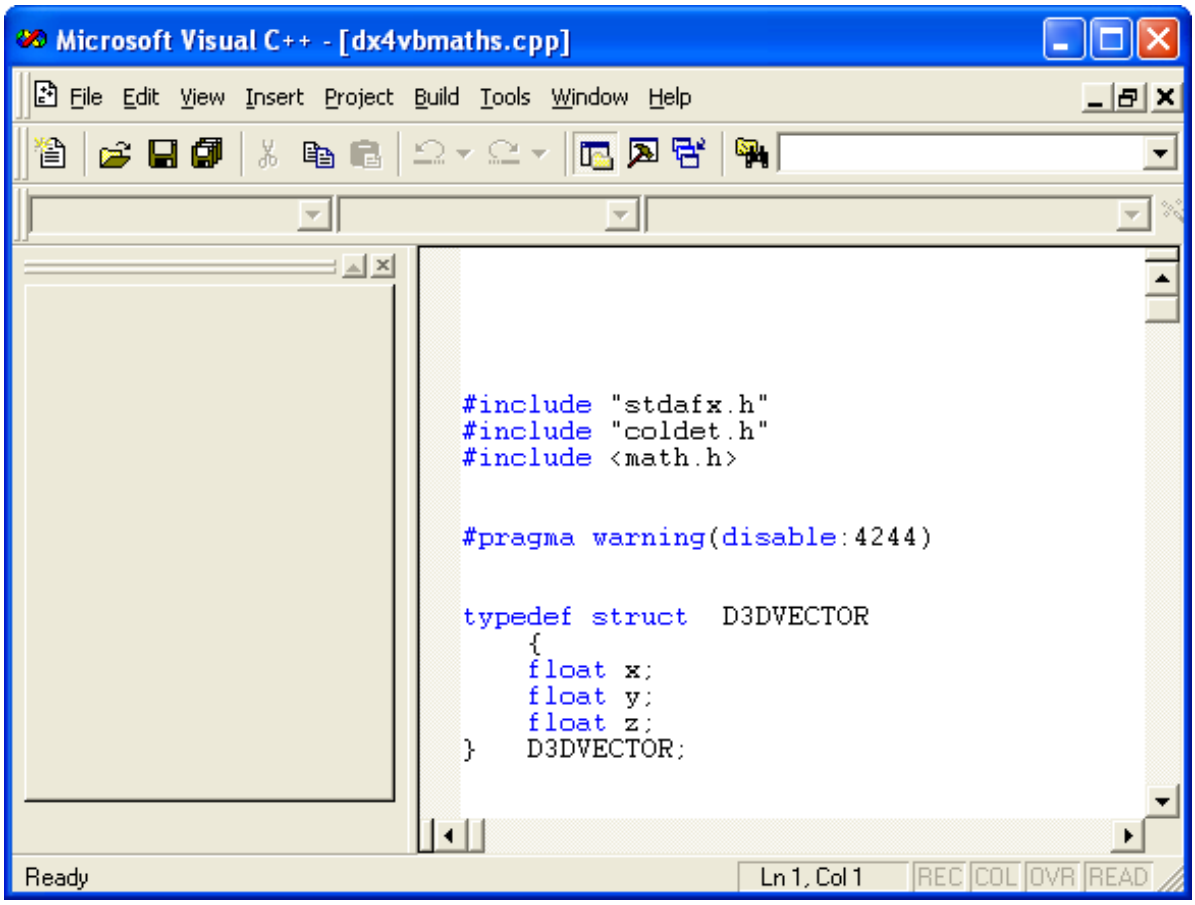


Figura 3.D *Finestra de programació principal del Visual C++ 6*

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: dibuixar ombres, 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 buscat 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 enfrontar-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 (*Aegia 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'infinit. 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.

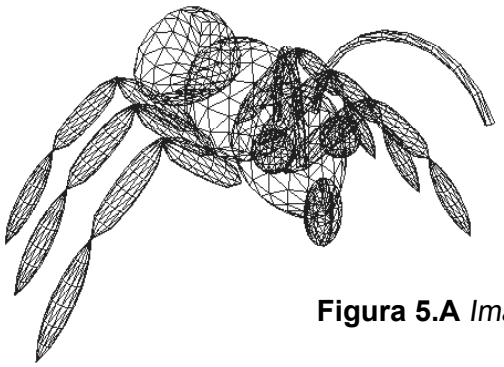


Figura 5.A *Imatge d'un model creat a partir de triangles*

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.

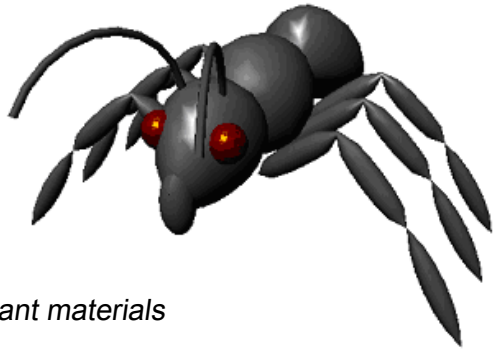


Figura 5.B *Imatge del model anterior però utilitzant materials*

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.

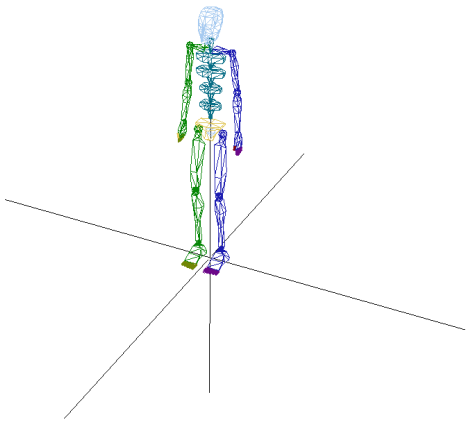


Figura 5.C *Un model amb el seu origen de coordenades*

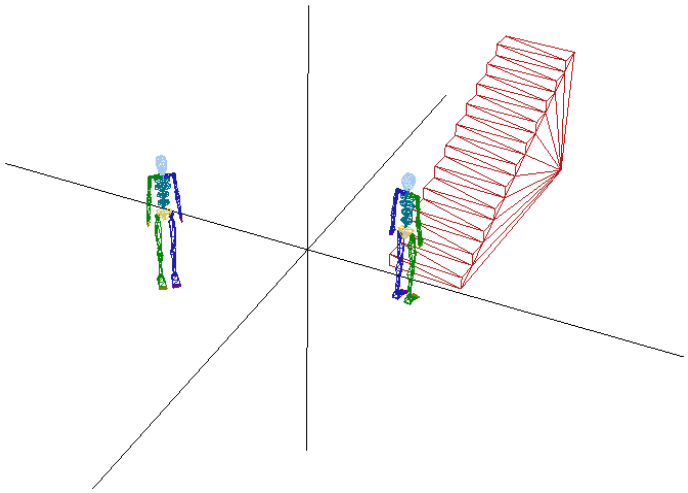


Figura 5.D
Un món 3D amb diferents models. S'ha aplicat un gir a cada model i se'ls ha posicionat en una nova posició

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

$$\begin{pmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{pmatrix}$$

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:

$$(x,y,z,1)' = (x,y,z,1)\begin{pmatrix} m11 & m12 & m13 & m14 \\ m21 & m22 & m23 & m24 \\ m31 & m32 & m33 & m34 \\ m41 & m42 & m43 & m44 \end{pmatrix}$$

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 x 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.

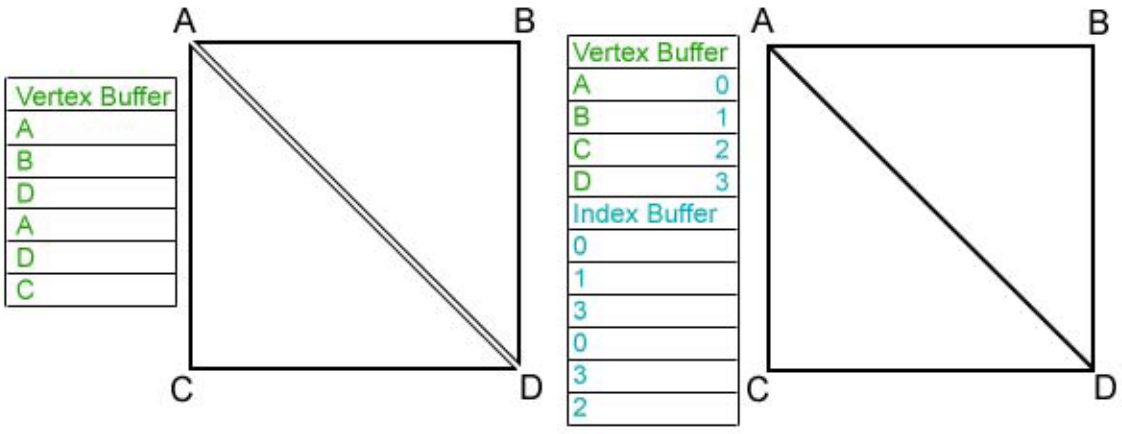


Figura 5.E *Diferència entre la utilització de vèrtexs i de vèrtexs i índexs*

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.

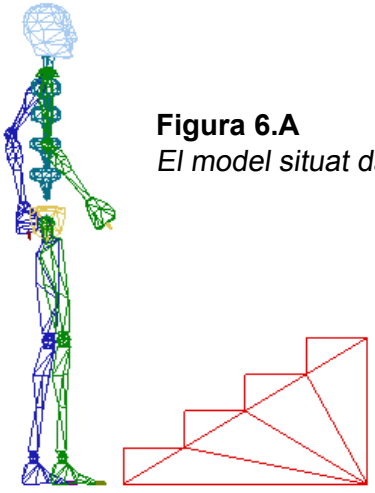


Figura 6.A
El model situat davant d'unes escales

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.

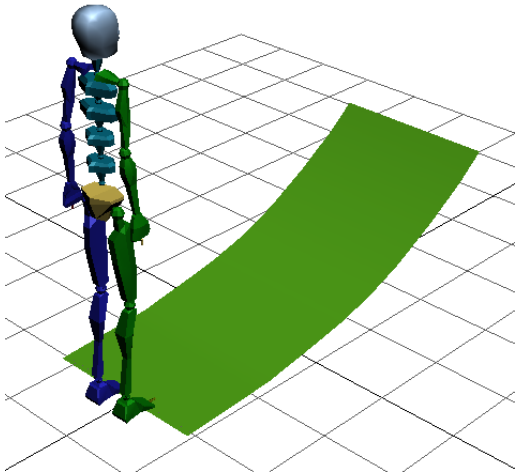


Figura 6.C *Situació inicial*

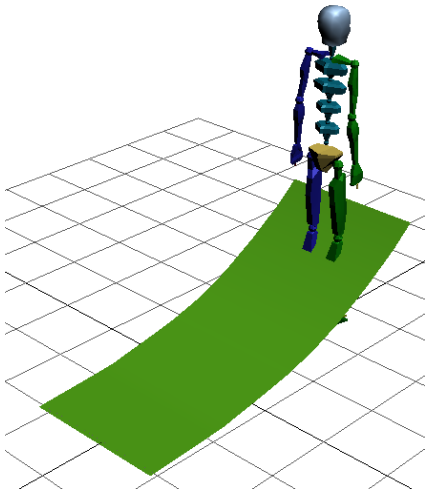


Figura 6.D *Sense col·lisió*

· *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.

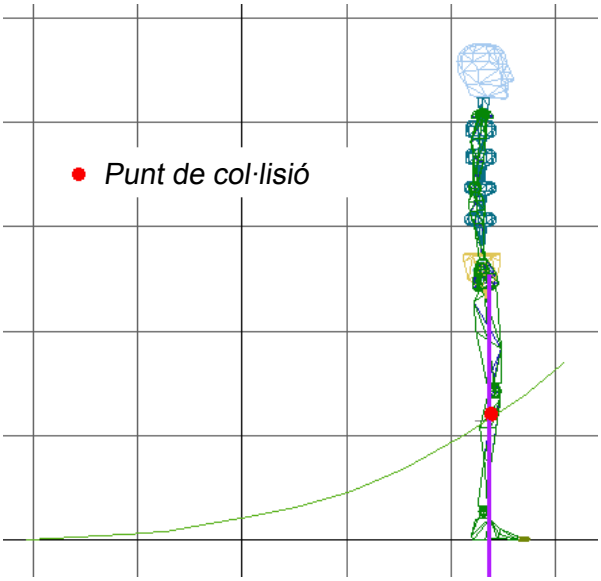
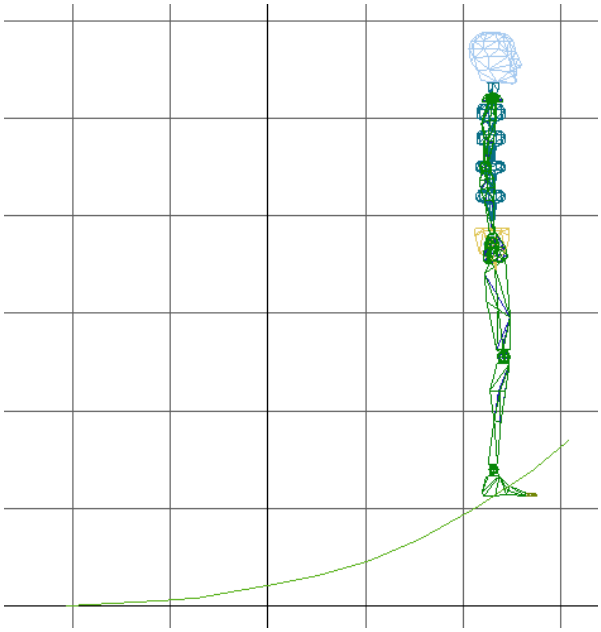


Figura 6.E *Detectar el punt de col·lisió*

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).

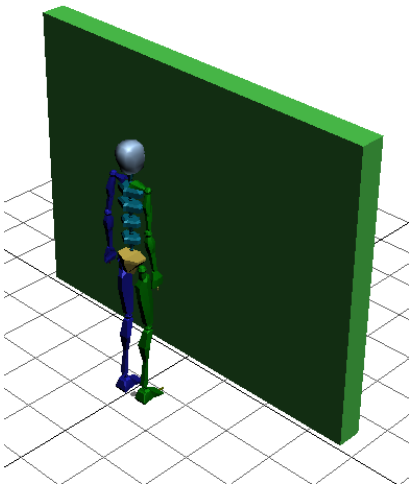


Figura 6.G *Abans de xocar*

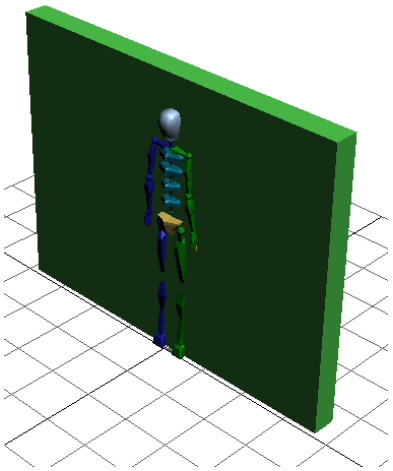


Figura 6.H *No es produeix xoc*

· *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 sobresurti de 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.

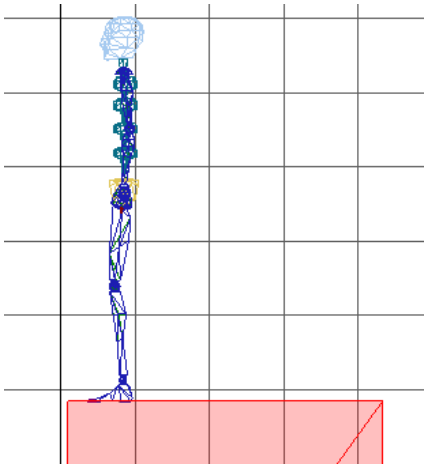


Figura 6.I

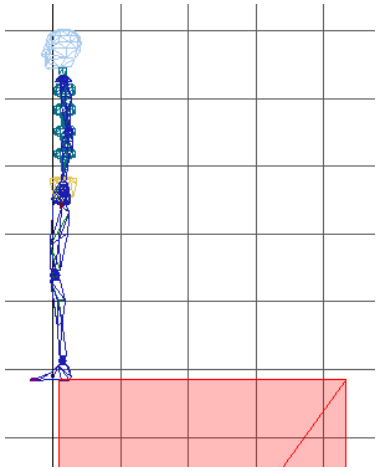


Figura 6.J

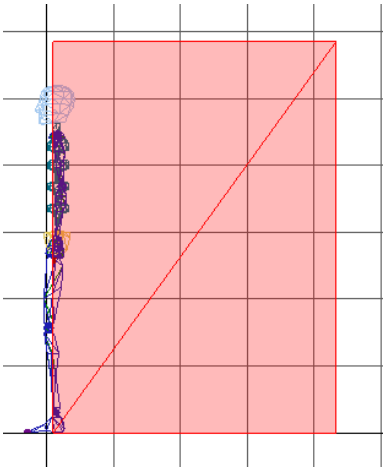


Figura 6.K

· *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 produeixi 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.

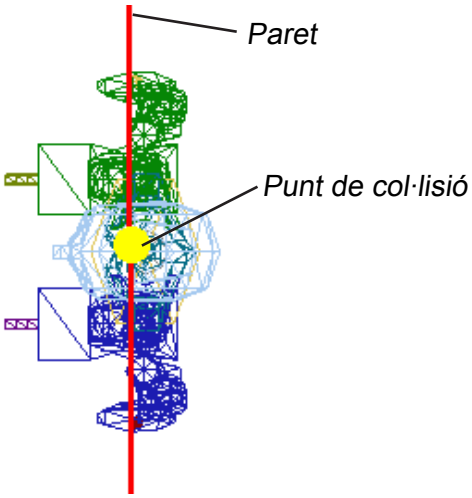


Figura 6.L

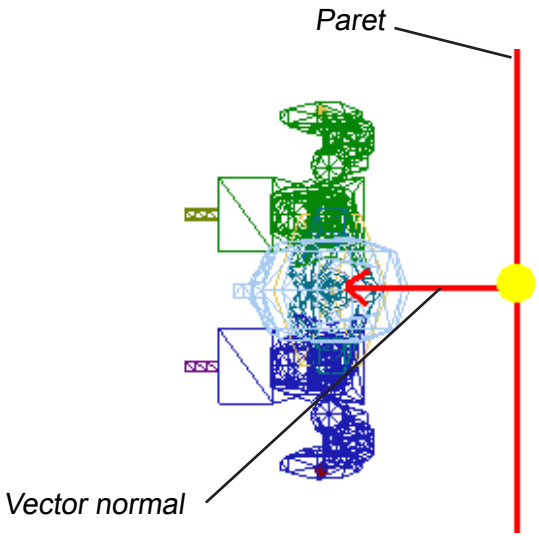


Figura 6.M

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: $\vec{N} = (\vec{B} - \vec{A})x(\vec{C} - \vec{A})$

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.

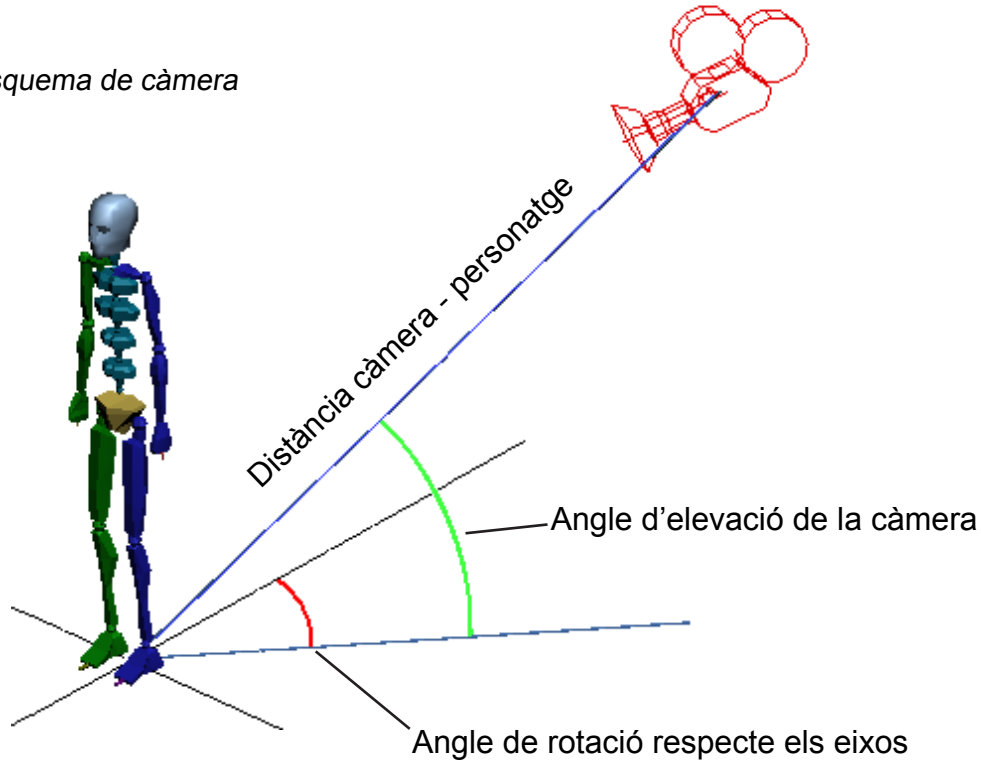
El *DirectX* proporciona la següent funció per a posicionar la càmera:

```
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:

CoordY = distancia · sin (anglev)

CoordX = (distancia · cos (anglev)) · sin (angleh)

CoordZ = (distancia · cos (anglev)) · cos (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 ocultes, 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

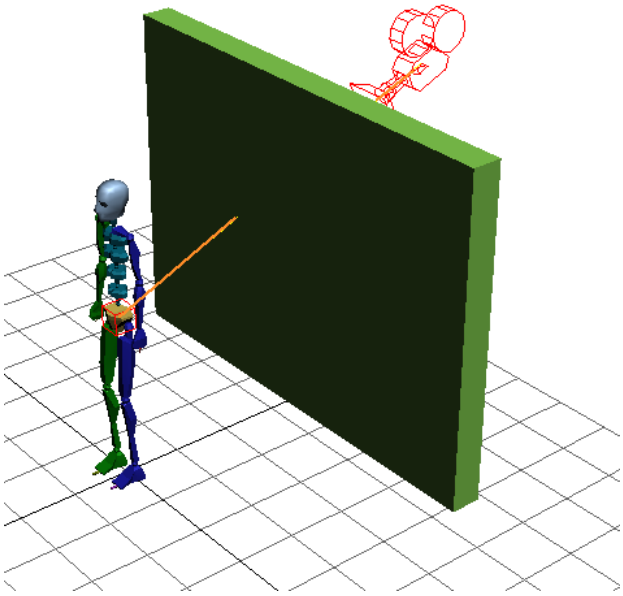


Figura 6.O

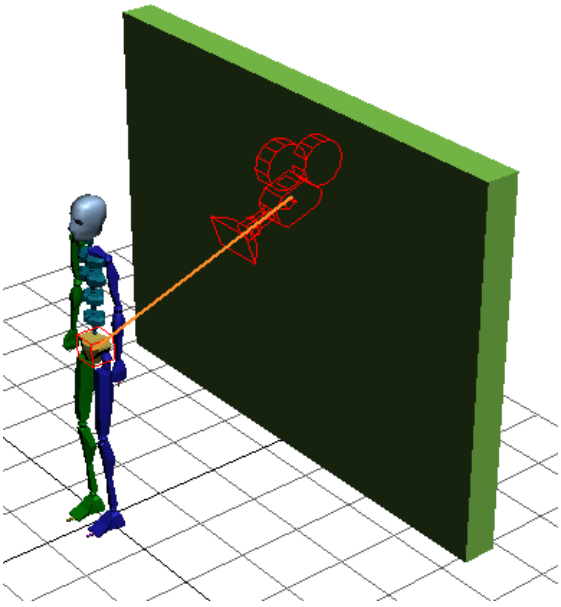


Figura 6.P

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 rodeta 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'empetiteixin 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 45º) 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.

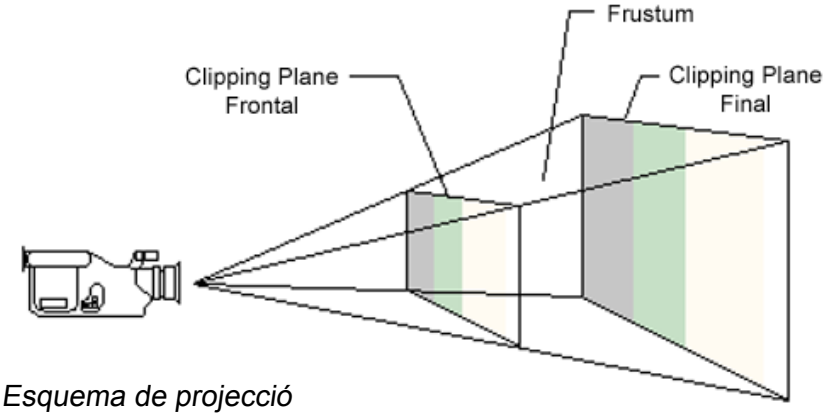


Figura 6.Q *Esquema de projecció*

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 lluminosos amb la distància. Cal dir que totes les llums poden ser de colors i és la blanca la que imita la llum natural.

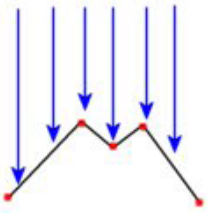


Figura 6.S *Llum direccional*

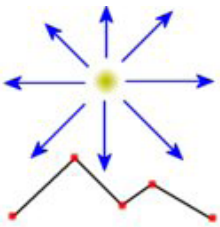


Figura 6.T *Llum puntual*

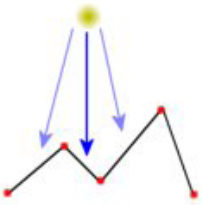


Figura 6.U *Llum de tipus focal*

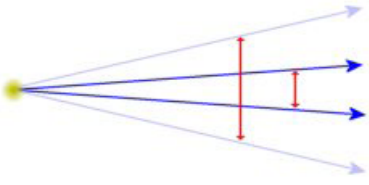


Figura 6.V *Con d'obertura d'una llum focal*

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 (90º) el grau d'il·luminació és 0. També cal dir que en cas que la llum il·lumini un triangle de normal oposada (oposat a la llum) aquest no s'il·luminarà.

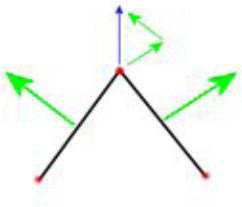


Figura 6.W *Càlcul de la normal d'un vèrtex*

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. Per a 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:

- *ZPass*: Va ser dissenyat per Frank Crow el 1977. Es caracteritza 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 visibles - invisibles només queda a la pantalla l'ombra projectada al terra o a qualsevol objecte que estigui dins del volum.

Els avantatges d'aquesta tècnica són la seva velocitat (mitjana, comparada amb la resta de tècniques) i la seva alta definició (ombres perfectament definides). Com a desavantatges cal destacar el més important de tots: si la càmera entra dins l'ombra, els polígons frontals no són dibuixats, pel que la resta anterior torna un valor invers i provoca el dibuix completament incorrecte de les ombres.

- *ZFail* (o *Carmack's Reverse*): La creació d'aquest mètode se li atribueix a John Carmack el 2000, uns dels creadors de videojocs més importants del món. Aquest mètode fa el mateix que el *ZPass* però a l'inrevés. Dibuixa les cares que no s'haurien de dibuixar en l'ordre invers de manera que aconsegueix el mateix resultat. El problema és que es necessita "tapar" amb polígons el volum d'ombra creat, ja que si no es fa, es produeix una petita inversió (similar a la del *ZPass*) quan la càmera se situa en la direcció de la llum. És més lent però té l'avantatge d'eliminar el problema de la càmera.

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, totes 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.

Per al joc s'ha triat la utilització de l'algorisme *ZFail* així com s'ha decidit que les ombres són opcionals i es poden triar a les Opcions del joc de forma lliure i voluntària (degut als seus requisits elevats).

Com ja s'ha explicat primerament s'ha de crear el volum d'ombra. Aquest volum serà estàtic pels edificis i el món en general, ja que no es mouen en cap moment. Per a crear aquest volum primer s'ha de triar la silueta de l'objecte des del punt de vista de la llum i després crear un volum "estirant" la silueta fins l'infinit.

Figura 6.X

A la figura 6.X es pot veure com es crea un volum que surt de la silueta fins l'infinit.

Per a determinar la silueta es crea una llista amb totes les arestes que pertanyen a triangles oposats a la llum i es trien aquelles que siguin úniques (que estiguin compartides per un triangle oposat i un de no oposat).

Un cop creat el volum (cal afegir una "tapa" al final i al principi per evitar errors) es dibuixarà seguint aquests passos:

- Dibuixar els triangles no visibles i que siguin oposats a la càmera. Incrementar el *Stencil Buffer*.

- Dibuixar els triangles no visibles però que mirin a la càmera. Decrementar el valor del *Stencil Buffer*.

- Omplir de color gris transparent la zona que en resulti de restar les dues anteriors.

Amb aquest procediment s'obté una ombra perfectament definida i sense cap tipus d'error.

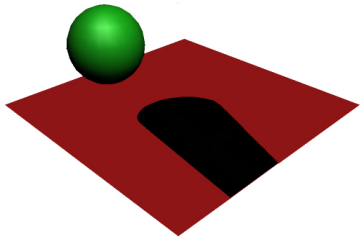


Figura 6.Y

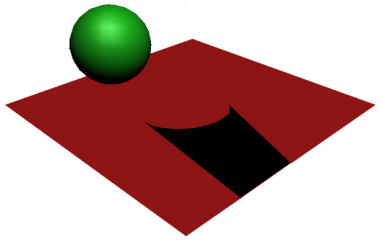


Figura 6.Z

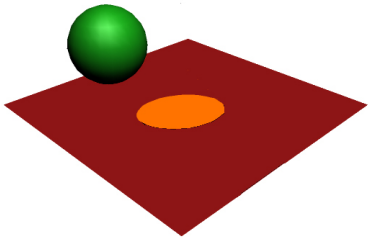


Figura 6.ZZ

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 així 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*.

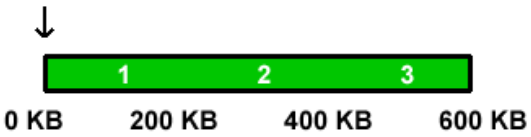


Figura 7.B

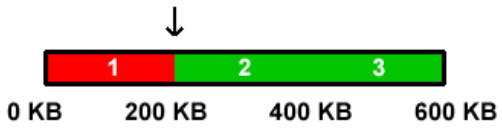


Figura 7.C

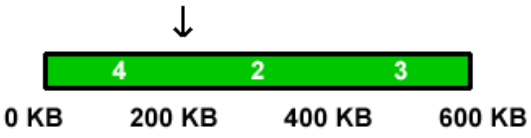


Figura 7.D

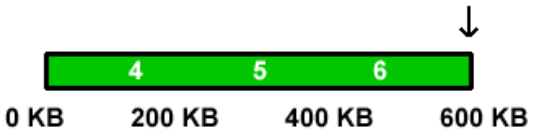


Figura 7.E

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.

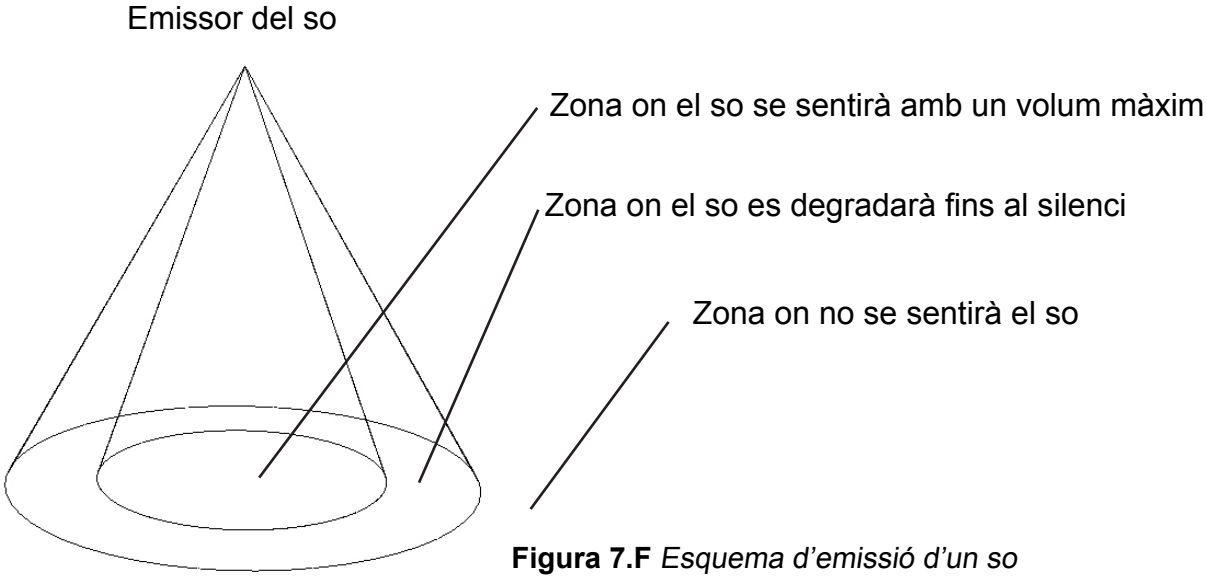


Figura 7.F *Esquema d'emissió d'un so*

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 perceben el so i que ajuden a millorar el realisme.

Optimitzacions

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, estalviant-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.

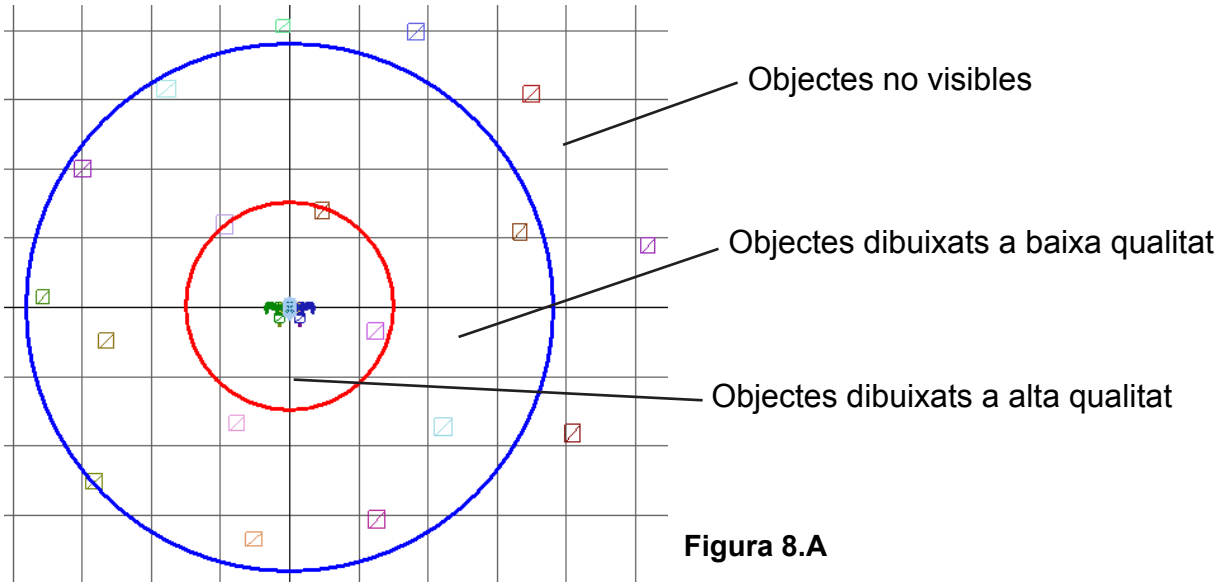


Figura 8.A

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).

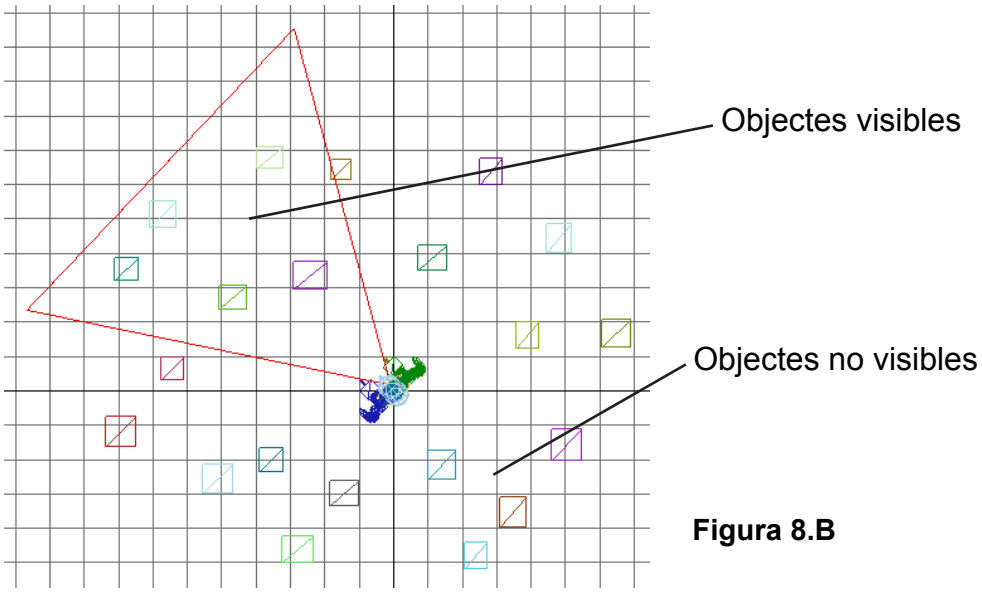


Figura 8.B

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.

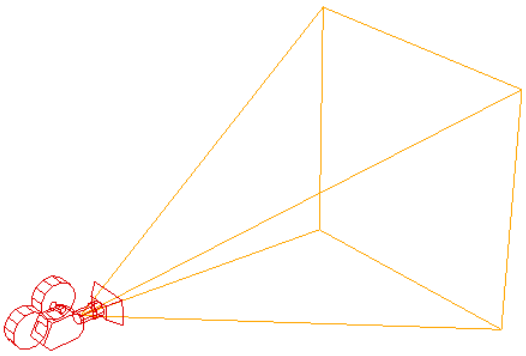


Figura 8.C

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.

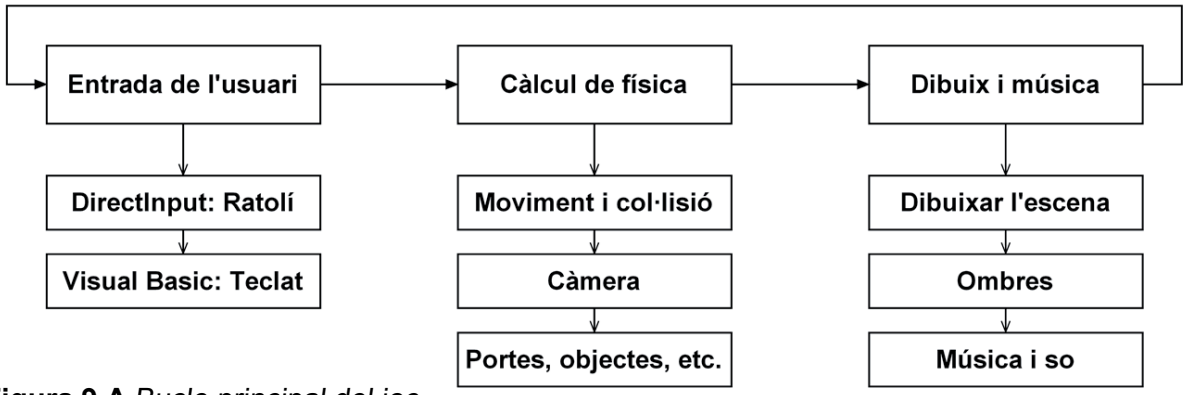


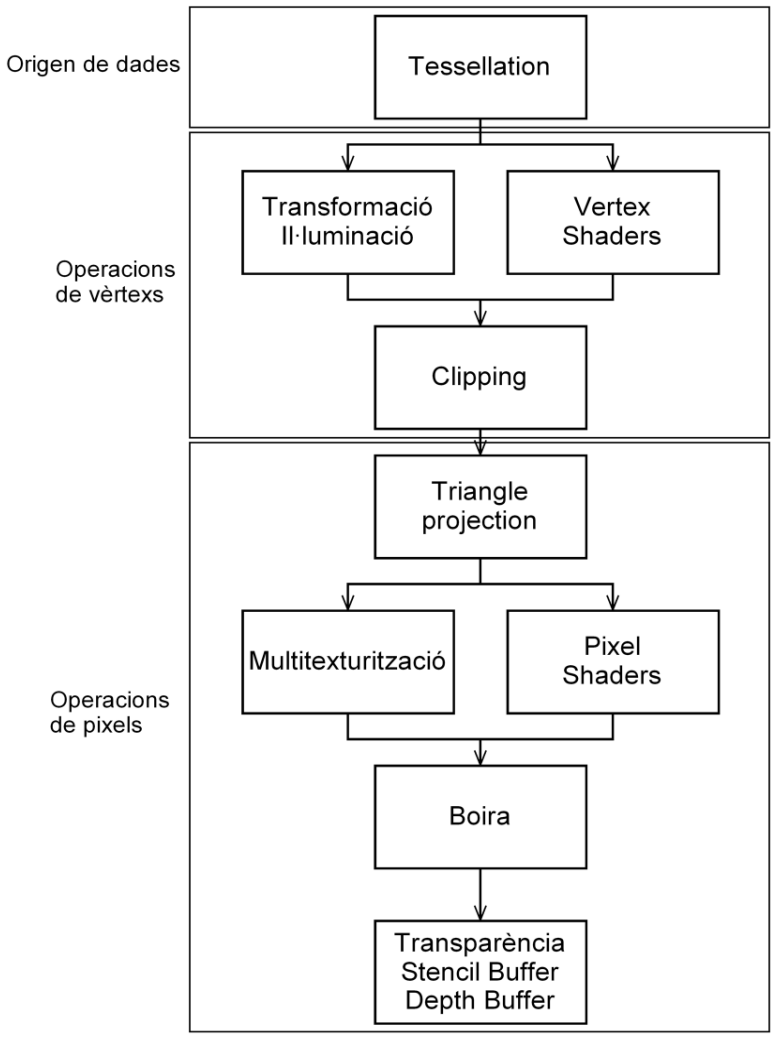
Figura 9.A *Bucle principal del joc*

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 *Vertexs 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 les ombres i els objectes amb transparències, que requereixen un tractament especial.

En el cas de les ombres el *buffer* amb els triangles preparats és dibuixat al *Stencil Buffer* enlloc 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 suplir-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 sistems 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.

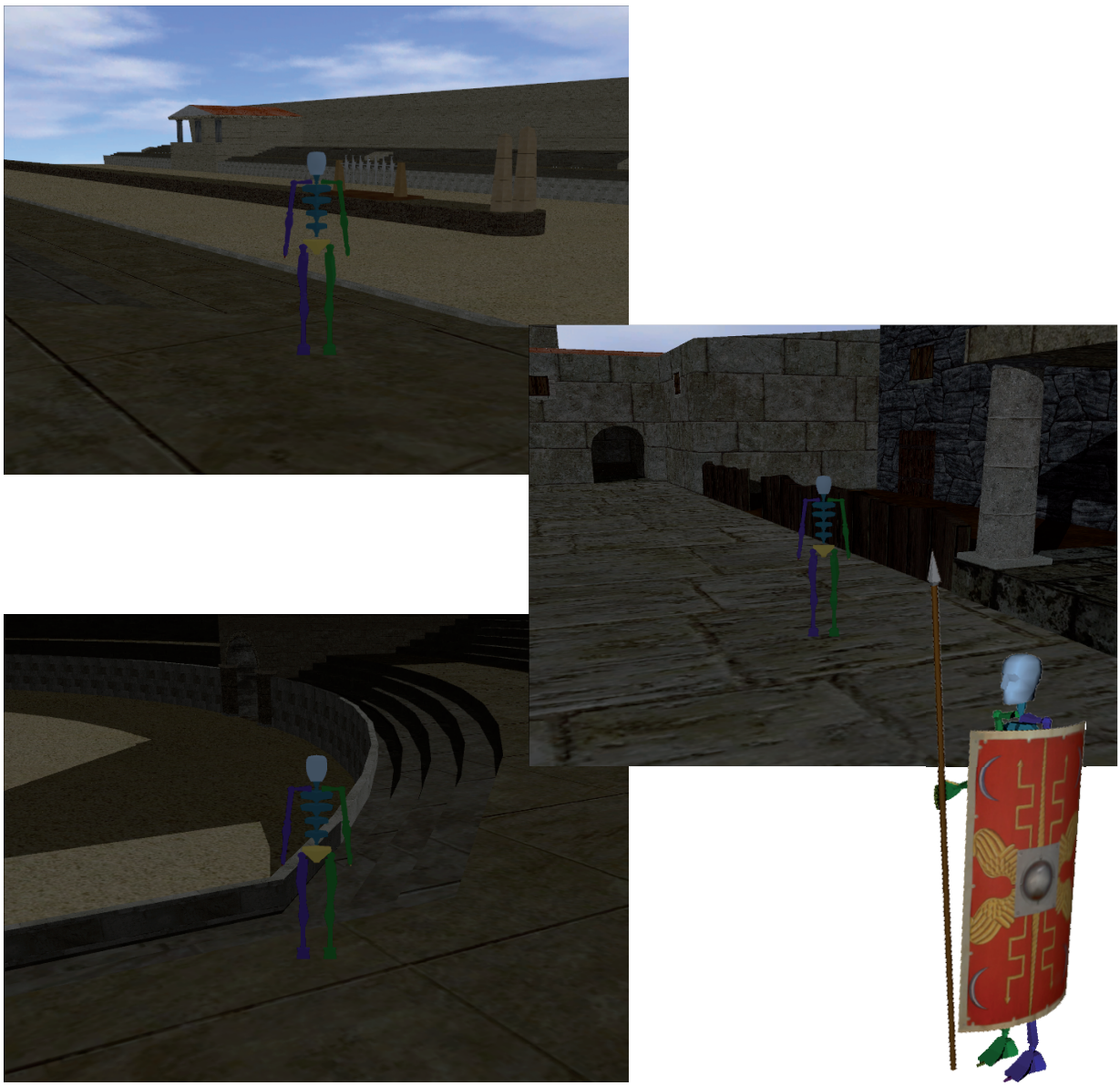
Aquestes funcions s'utilitzen normalment en aquest ordre: `SetRenderState` per a definir les opcions de dibuix prèvies, `SetLight` per a crear les llums necessàries, `Clear` per a esborrar el buffer de dibuix, `SetMaterial` i `SetTexture` per a preparar el dibuix dels triangles, `SetTransform` per aplicar 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 beneficiós, 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 conec 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:

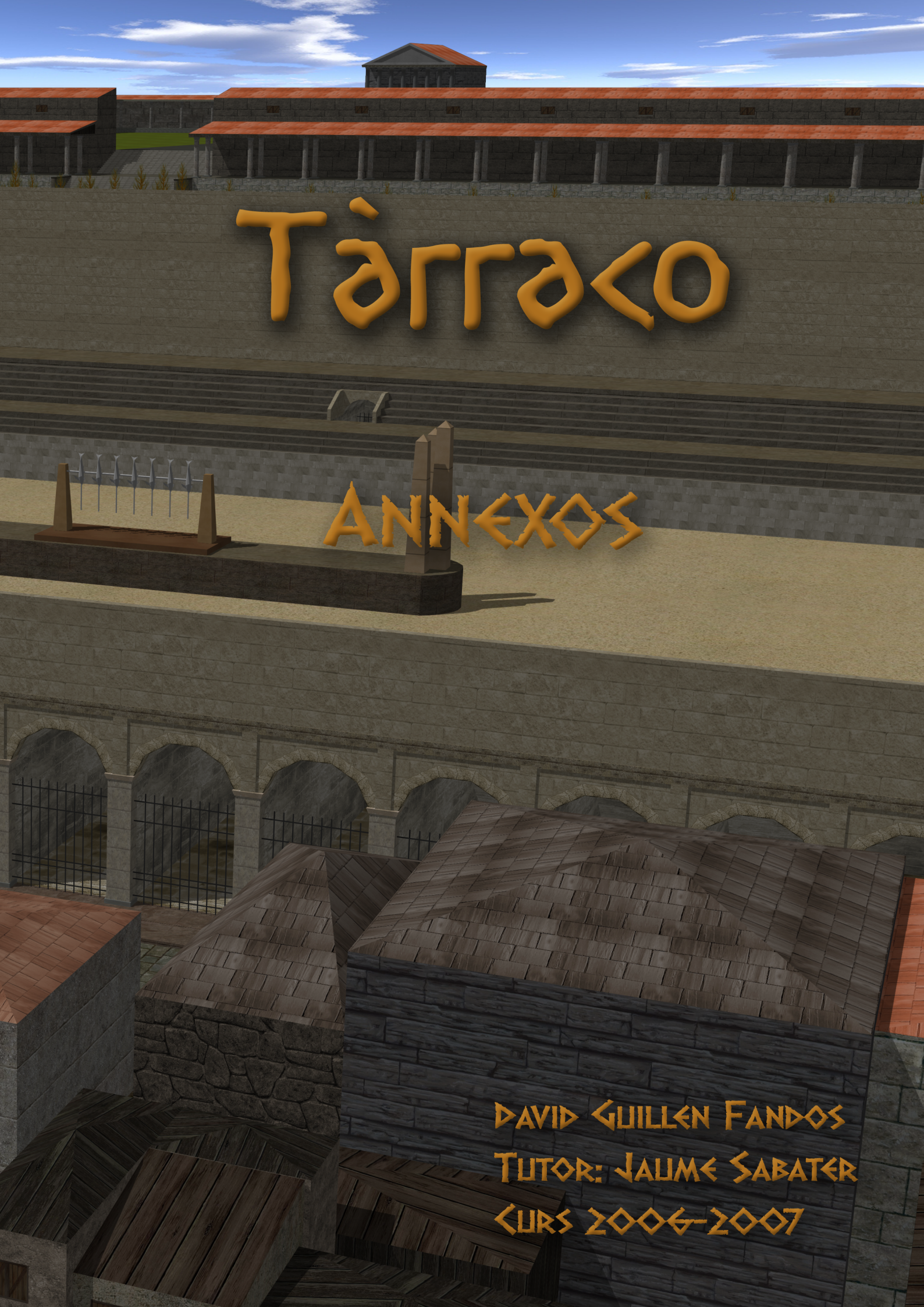
- GameDev: http://www.gamedev.net
- DirectX4Vb: http://directx4vb.vbgamer.com
- Visual Basic Gamer: http://www.vbgamer.com
- ColDet: http://coldet.sourceforge.net
- FreeImage: http://freeimage.sourceforge.net
- Planet Source Code: http://www.planet-source-code.com
- FlipCode: http://www.flipcode.com
- 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 (ISBN: 1-58450-054-9)
  Autor: Mark DeLoura Editorial: Charles River Media
- Game Programming Gems 4 (ISBN: 1-58450-295-9)
  Autor: Andrew Kirmse Editorial: Charles River Media



Tàrraco

Annexos

David Guillen Fandos
Tutor: Jaume Sabater
Curs 2006-2007

Annex: A
Glossari

Aquí s'ha realitzat un petit recull de termes relacionats amb la programació, la informàtica i el món del 3D per a clarificar totes aquelles paraules que puguin produir una confusió durant la lectura del treball.

Bucle: Procés que es repeteix un cert nombre de vegades o indefinidament fins que es compleixi una determinada condició.

Buffer: Espai de memòria utilitzat amb alguna finalitat concreta que es pot trobar tant a la memòria de l'ordinador com a la de la targeta de vídeo. S'utilitza amb diferents propòsits; des de guardar imatges i so fins a emmagatzemar vèrtexs i altres dades numèriques. Segons la utilitat que rep se l'anomena juntament amb el contingut: *Vertex Buffer*, *Index Buffer*, *Color Buffer*, etc.

Compressió: En termes informàtics, aplicar un algorisme a una sèrie de dades amb l'objectiu que ocupin un menor espai a la memòria o al disc.

Depth Buffer: També anomenat *buffer* de profunditat o *ZBuffer*. És un espai de memòria a nivell de pixel que emmagatzema la profunditat de l'objecte més proper a la càmera. La seva utilitat és ordenar el dibuix de polígons i determinar quins polígons són vistos i quins ocults (queden tapats per uns altres). L'espai reservat és variable però es compleix que a més espai reservat més precisió de dibuix. Comparteix memòria amb el *Stencil Buffer*.

Direct3D: Llibreria que forma part del *DirectX* i que s'encarrega del dibuix de gràfics a la pantalla. Usualment treballa en tres dimensions, tot i que també ho pot fer en dues dimensions.

DirectInput: Llibreria que forma part del *DirectX* i que s'encarrega de detectar entrades de l'usuari a través de tot tipus de dispositius: ratolí, teclat, *joystick*, etc.

DirectSound: Llibreria que forma part del *DirectX* i que s'encarrega de la reproducció de sons i música. Permet utilitzar so 3D, és a dir, amb sensació de profunditat.

Estructura (de dades): Forma que té un espai de memòria concret. Un exemple d'estructura seria l'estructura d'un vèrtex (*D3DVERTEX*) que conté tres components: x, y i z. Totes les dades estan estructurades d'alguna manera, amb alguna forma concreta, ja que si no el seu ús en seria impossible.

Funció: En programació a l'igual que en matemàtiques una funció és una petita part de codi o programa que, donats uns arguments, torna un valor. Un exemple seria la funció suma, que torna el resultat d'una suma donats dos sumands. En programació s'admeten arguments no numèrics tals com caràcters alfanumèrics i estructures de dades més complexes.

Índex: Un índex (quan es parla de geometria tridimensional) és un nombre que apunta a un vèrtex. És a dir, indica l'ordre d'un vèrtex en un dibuix enlloc d'indicar el vèrtex en si mateix. La gran utilitzat d'aquest és el fet que ocupa molta menys memòria que un vèrtex i permet eliminar grans quantitats de vèrtexs repetits.

Mètode: En programació es refereix a un conjunt de línies de codi que, a l'igual que una funció, realitzen un determinat treball. A diferència d'aquestes, però, no tornen cap valor. S'utilitzen per a executar processos que no requereixen cap valor a canvi.

Model: S'anomena així a un objecte o figura tridimensional creat a partir de vèrtexs i cares poligonals. És la peça bàsica en la visualització gràfica d'un joc. També es pot dir model 3D.

Pixel: Petit quadrat que forma part d'una imatge. Una agrupació de pixels en forma de quadrícula defineixen una imatge. Cada pixel té un color definit per les tres components de color: verd, vermell i blau.

Prerenderitzat: Es refereix a una imatge o qualsevol altra cosa que ha estat renderitzada prèviament (dibuixada en perspectiva prèviament), no en el mateix instant. Es parla d'imatges prerenderitzades quan es refereix a dibuixos tridimensionals fets prèviament a la seva utilització.

Renderitzar: En argot informàtic i de disseny és l'acció de dibuixar una escena tridimensional en una imatge bidimensional. Un sinònim seria dibuixar en perspectiva. Prové de l'anglès (*Render*).

Stencil Buffer: El *Stencil Buffer* és un *buffer* creat a l'espai de dibuix de la targeta gràfica i que comparteix memòria amb el buffer de profunditat (*Depth Buffer*). El seu ús és personalitzable. S'utilitza per a diverses tasques entre les quals destaca la creació d'ombres. És considerat a nivell de pixel pel fet que és compartit amb cada un dels pixels de la pantalla. Només algunes targetes gràfiques noves l'incorporen. Les més velles són incompatibles amb la seva utilització.

Targeta gràfica: Component electrònic que forma part de l'ordinador i que permet mostrar gràfics (imatges) a la pantalla d'aquest. Té moltes funcions que permeten el dibuix de gràfics tridimensionals i és essencial el seu ús en la creació d'un joc.

Textura: Imatge bidimensional que és dibuixada a sobre d'un polígon amb l'objectiu d'afegir realisme al model tridimensional que forma. Enlloc d'acolorir els polígons es poden utilitzar textures per a un major realisme.

Texturitzar: Acció d'aplicar una textura a un triangle, un model, o qualsevol objecte. De la mateixa manera l'adjectiu texturitzat designa un objecte que ha passat un procés de texturització.

Annex: B
ombres

En aquest annex es recullen articles i documents extrets de diferents pàgines webs on programadors i enginyers professionals exposen mètodes per a la creació d'ombres eficientment. Aquests documents són els que es van emprar per a crear l'algorisme que utilitza el joc.

- **The Theory of Stencil Shadow Volumes** de Hun Yen Kwoon.

Explica la teoria de les ombres creades a partir d'un volum creat per un objecte. Compara els algorismes *ZPass* i *ZFail*.

Extret de www.gamedev.net

- **Hardware Shadow Mapping** per Cass Everitt, Ashu Rege i Cem Cebenoyan.

Parla sobre les ombres creades a partir d'un mapa d'ombres i la seva implementació en *OpenGL* i *Direct3D*.

Extret de www.nvidia.com

The Theory of Stencil Shadow Volumes

by Hun Yen Kwoon

Introduction

Shadows used to be just a patch of darkened texture, usually round in shape, which is projected onto the floor below characters or objects in a game. One must be ill informed or naïve to think that we can still get away with this kind of sloppy "hacks" in future 3D games. There used to be a time where shadows are just too expensive to be rendered properly in real-time, but with the ever-increasing power of graphics hardware, failure to provide proper shadows no longer meant mediocre implementations, it borders on being guilty of criminally under-utilizing the graphics hardware available.

There are many differing shadowing techniques and approaches to implementing shadows and nailing down a "best" solution is difficult. In order to understand all the approaches and appreciate their differences, strengths and weakness, I strongly suggest reading anything and everything about doing shadows in 3D. We should not constrain ourselves to just studying shadow volume technique; any shadowing technique is worth a look. Chapter 6 of [13] has a wonderful high-level discussion on most of the known shadowing techniques. To limit the scope of this paper, we shall only discuss the theory and implementation issues of stencil shadow volumes with particular reference to using Microsoft's Direct3D API. It would also be good to understand that stencil shadow volume just isn't the "end all" shadowing technique. A discussion on the strengths of differing shadowing technique can be found at [4] with reference to a game setting. Just recently, Eric Lengyel [11] also presented a very complete article on implementing shadow volumes in OpenGL at the Gamasutra website [17]. The mathematical derivations of Lengyel's article can be found in [12]. Going back a few years, there was the famous "Carmack On Shadow Volumes" text file [6], which is nothing more that an email from John Carmack of id Software to Mark Kilgard of Nvidia about the derivation of the depth-fail shadow volume implementation. It's interesting to note that Carmack independently discovered the depth-fail method while Bill Bilodeau and Mike Songy [7] had also presented similar approach to shadow volumes. Consequently, the depth-fail method is now commonly known as "Carmack's Reverse".

Stencil Shadow Volume Concept

Frank Crow [8] first presented the idea of using shadow volumes for shadow casting in 1977. Tim Heidmann [5] of Silicon Graphics implemented Crow's shadow volume by cunningly utilizing the stencil buffer for shadow volume counting in IRIX GL. Lets take a look at how the original stencil shadow volume technique works.

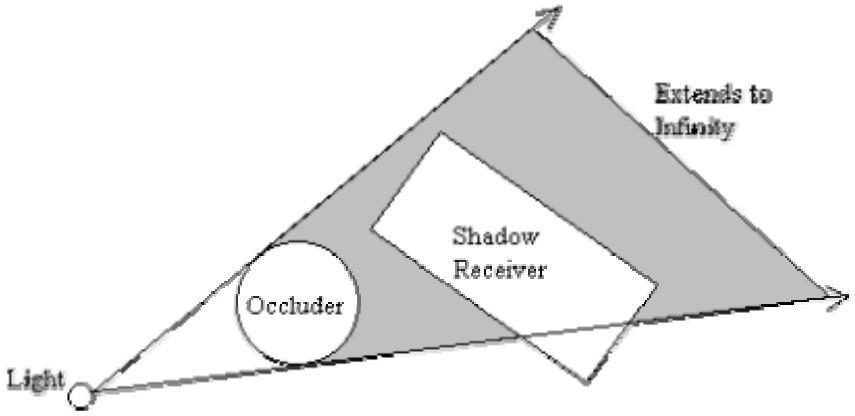


Figure 1: Occluder and shadow volume

As common convention goes, any objects in a scene that cast shadows are called **occluders**. As shown in Figure 1 above, we have a simplistic 2D view (top down) of a scene with a sphere as the occluder. The rectangle to the right of the sphere is the shadow receiver. For simplicity, we do not take into account the shadow volume created by the rectangle. The shaded region represents the shadow volume, in 2D, created by the occluder. The shadow volume is the result of extruding the silhouette edges from the point-of-view of the light source to a finite or infinite distance.

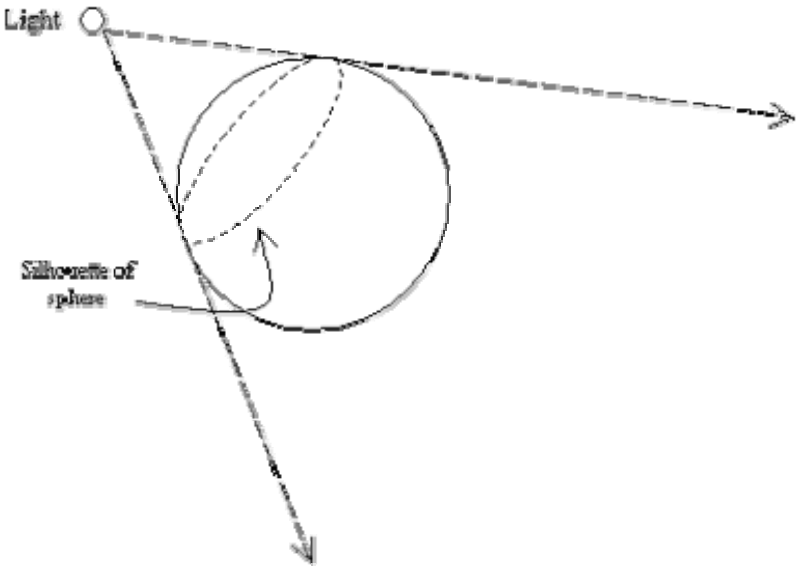


Figure 2: Silhouette of occluders

Figure 2 shows the probable silhouette of the sphere generated from the viewing position of the light source. The silhouette is simply made up of edges that consist of two vertices each. These edges are then extruded in the direction as shown by the broken arrows originating from the light source. By extruding the silhouette edges, we are effectively creating the shadow volume. It should be noted at this point in time that shadow volume extrusion differs for different light sources. For point light sources, the silhouette edges extrude exactly point for point. For infinite directional light sources, the silhouette edges extrude to a single point. We will go into the details of determining silhouette edges and the creation of the shadow volumes later. The magnitude of the extrusion can be

either finite or infinite. Thus, implementations that extrude silhouette edges to infinity are commonly known as Infinite Shadow Volumes.

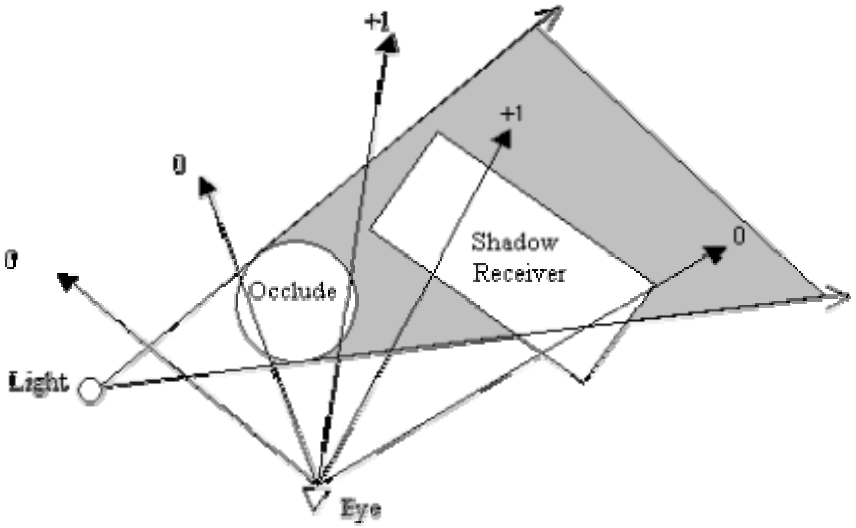


Figure 3: Depth-Pass stencil operation

Figure 3 shows the numerous possible viewing direction of a player in the scene. The numbers at the end of the arrows are the values left in the stencil buffer after rendering the shadow volume. Fragments with non-zero stencil values are considered to be in shadow. The generation of the values in the stencil buffer is the result of the following stencil operations:

1. Render **front** face of shadow volume. If depth test passes, **increment** stencil value, else does nothing. Disable draw to frame and depth buffer.
2. Render **back** face of shadow volume. If depth test passes, **decrement** stencil value, else does nothing. Disable draw to frame and depth buffer.

The above algorithm is also known as the Depth-Pass stencil shadow volume technique since we manipulate the stencil values only when depth test passes. Depth-pass is also commonly known as z-pass.

Let's assume that we had already rendered the objects onto the frame buffer prior to the above stenciling operations. This means that the depth buffer would have been set with the correct values for depth testing or z-testing if you like. The 2 leftmost ray originating from the eye position does not hit any part of the shadow volume (in gray), hence the resultant stencil values is 0, which means that the fragment represented by this two rays are not in shadow. Now lets trace the 3rd ray from the left. When we render the front face of the shadow volume, the depth test would pass and the stencil value would be incremented to 1. When we render the back face of the shadow volume, the depth test would fail since the back face of the shadow volume is behind the occluder. Thus the stencil value for the fragment represented by this ray remains at 1. This means that the fragment is in shadow since its stencil value is non-zero.

Does the shadow volume counting work for multiple shadow volumes? Yes it does.

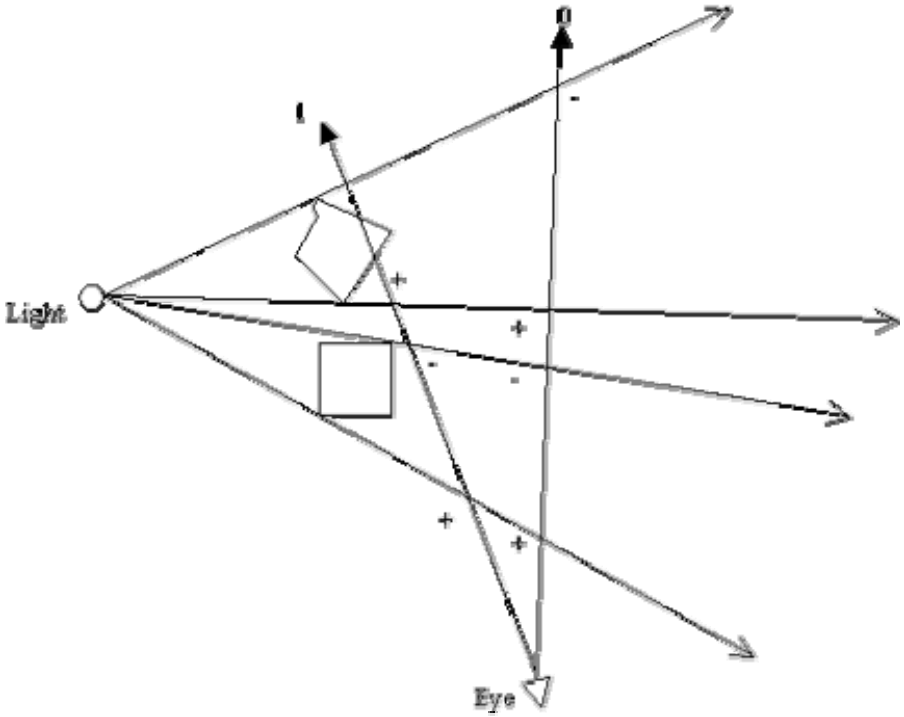


Figure 4: Multiple shadow volumes counting

Figure 4 above shows that the counting using the stencil buffer will still work even for multiple intersecting shadow volumes.

Finite Volume *vs* Infinite Volume

Referring back to Figure 1, you could see that the shadow volume is supposed to extrude to infinity. This is actually not strictly a requirement. We send the shadow volume to infinity in order to avoid the awkward situation whereby the light source is very close to an occluder.

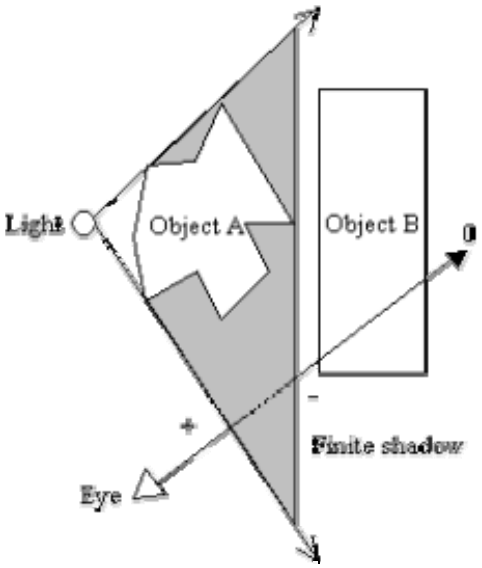


Figure 5: Finite shadow volume fails to shadow other objects

With the light close to object A, a finite shadow volume may not be enough to reach object B. The ray from the eye towards object B will end up with a fragment stencil value of 0 when in fact it should have been non-zero! An infinite shadow volume would ensure that no matter how close the object is to an occluder, the resultant shadow volume would cover all the objects in the scene. We will discuss how to extrude vertices to infinity shortly.

Carmack's Reverse

Why did John Carmack, Bill Bilodeau and Mike Songy even bother to crack their heads to come out with an alternative stencil algorithm since the depth-pass technique seems to work great? Depth-pass really works well, at least most of the time. But when the eye point enters the shadow volume, all hell break loose.

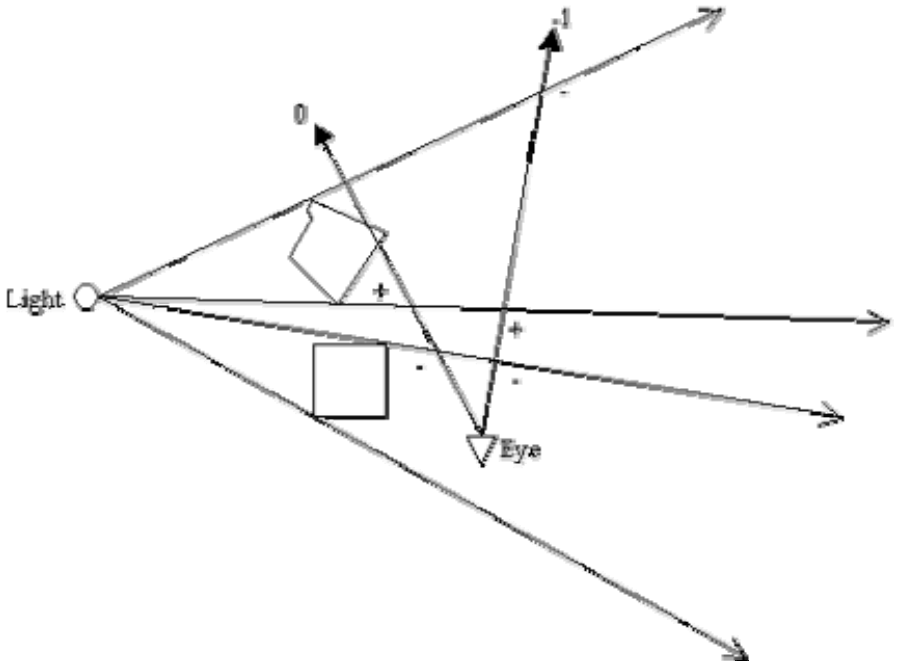


Figure 6: When eye point is within the shadow volume, depth-pass stencil operation fails

As shown in Figure 6 above, the depth-pass technique utterly fails when the eye point is within the shadow volume. This meant that we could not have that big bad horned reaper sneaking up from behind you while engulfing you in the enlarging darkness of his shadows. John Carmack would never have it this way! The following is the depth-fail (a.k.a Carmack's Reverse) algorithm:

1. Render **back** face of shadow volume. If depth test fails, **increment** stencil value, else does nothing. Disable draw to frame and depth buffer.
2. Render **front** face of shadow volume. If depth test fails, **decrement** stencil value, else does nothing. Disable draw to frame and depth buffer.

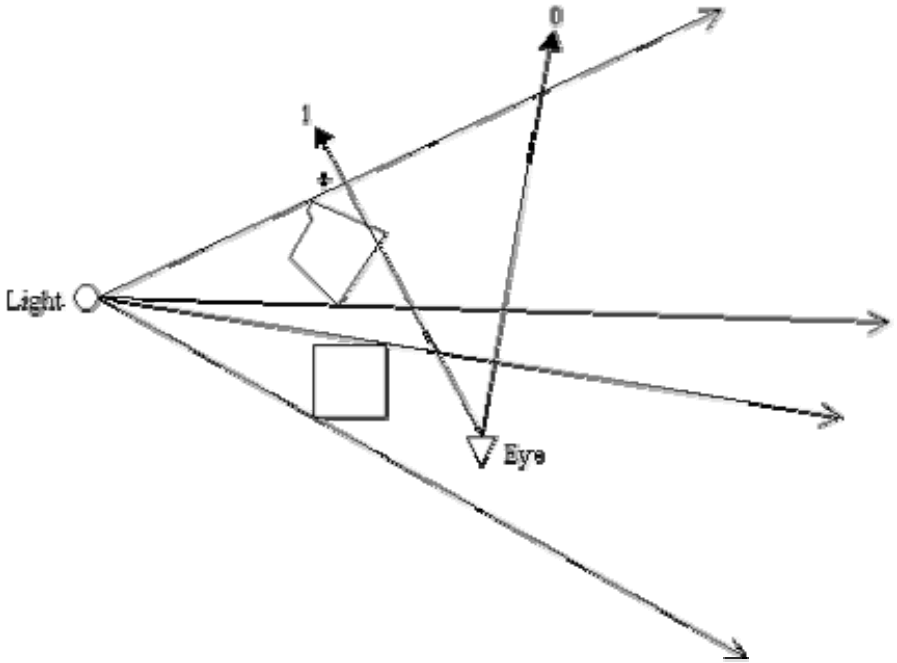


Figure 7: Depth-fail works even if eye point is in shadow

Depth-fail is also commonly referred to as z-fail. Figure 7 shows the depth-fail technique working even when the eye point is in shadow. If you think about the scenario where the eye position is outside the shadow volume, the depth-fail technique should work as well. But really, it fails in some cases. We shall discuss these scenarios soon; just remember for now that both the depth-pass and depth-fail techniques are not perfect. In fact, we would need a combination of different methods to come up with a robust solution for shadow volumes. [11] and [10] contains some very good discussion on robust stencil shadow volume solutions.

Capping For Depth-Fail

To put in non-zero values into the stencil buffer, the depth-fail technique depends on the failure to render the shadow volume's back faces with respect to the eye position. This meant that the shadow volume must be a closed volume; the shadow volume must be capped at both the front and back end (even if back end is at infinity). Without capping, the depth-fail technique would produce erroneous results. Amazing as it may sound, but yes, you can cap the shadow volume even at infinity.

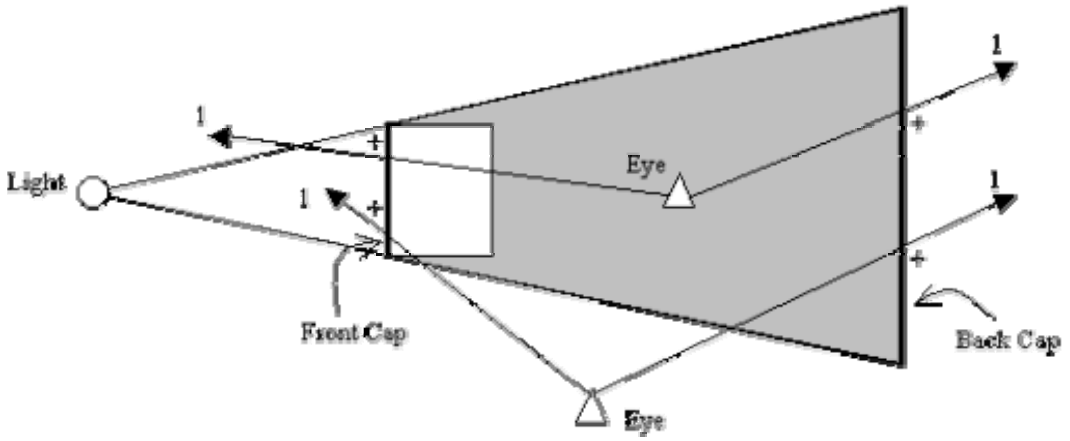


Figure 8: Capping for shadow volume

As shown in Figure 8, the front and back cap (bold lines) creates a closed shadow volume. Both the front and back caps are considered back face from the two eye positions. With depth-fail stenciling operations, the capping will create correct non-zero stencil values. There are a few ways to create the front and back capping. Mark Kilgard [2] described a non-trivial method of creating the front cap. The method basically involves the projection of the occluder's back facing geometries onto the near clip plane and uses these geometries as the front cap. Alternatively, we can build the front cap by reusing the front facing triangles with respect to the light source. The geometries used in the front cap can then be extruded, with their ordering reversed, to create the back cap. Reversing the ordering is to ensure that the back cap face outward from the shadow volume. In fact, we must always ensure that the primitives, in our case triangles, that define the entire shadow volume are outward facing as shown in Figure 9. It must be noted that rendering closed shadow volumes are somewhat more expensive than using depth-pass without shadow volume capping. Besides a larger primitive count for the shadow volume, additional computational resource are also needed to compute the front and back capping. We will go into the details of capping shadow volumes shortly.

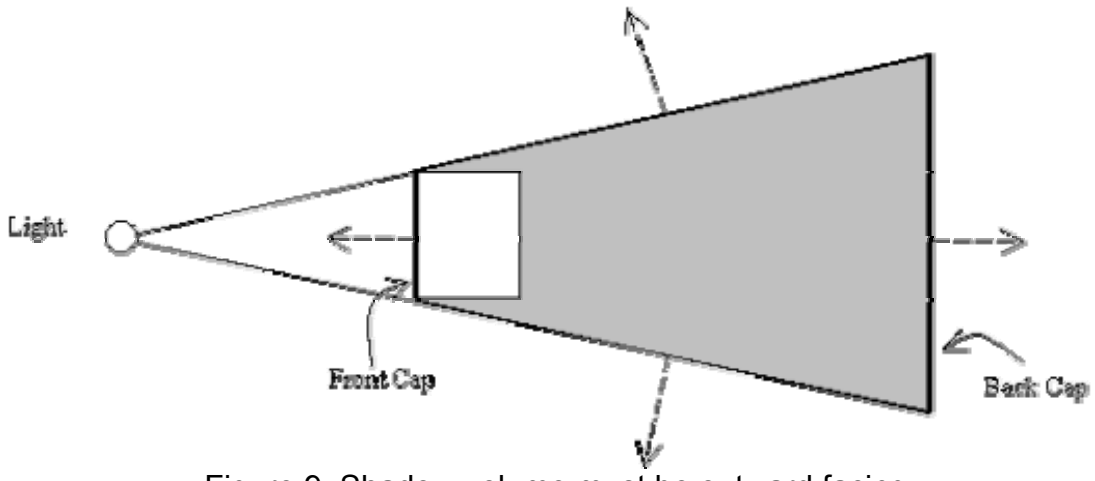


Figure 9: Shadow volume must be outward facing

Putting It Together

Let's collate what we have learned and try to come up with all the require steps to do stencil shadow volumes before we tackle all the deficiencies of the techniques discussed. A general list of steps to implement stencil shadow volumes would be:

1. Render all the objects using only ambient lighting and any other surface-shading attribute. Rendering should not depend on any particular light source. Make sure depth buffer is written.
2. Starting with a light source, clear the stencil buffer and calculate the silhouette of all the occluders with respect to the light source.
3. Extrude the silhouette away from the light source to a finite or infinite distance to form the shadow volumes and generate the capping if depth-fail technique was used. (Infinite shadow volume extrusion is not really compulsory)
4. Render the shadow volumes using the selected technique. Depth-pass or depth-fail.
5. Using the updated stencil buffer, do a lighting pass to shade (make it a tone darker) the fragments that corresponds to non-zero stencil values.
6. Repeat step 2 to 5 for all the lights in the scene.

From the above list of steps, it should be quite obvious that having more lights means having more passes, which can burn a nice hole in your frame rate pocket. In fact, we have to be very selective when deciding which lights should be used for casting shadows. The article [4] has a nice discussion on selecting shadow casting lights within a scene lit by multiple light sources. Imagine your game character standing in the middle of a stadium with four gigantic batteries of floodlights shining down the field. There should be at least 4 shadows of you game character on the floor forming a cross due to the shadow casting from 4 different directions. Selecting only 1 light source here is going to make the scene look weird. Having multiple lights allows you to get nice realistic soft shadows but there are other ways to fake it without resorting to multiple light sources. Soft shadow is a huge topic and is not within the scope of this paper, so let's just drop it from here. Rule of thumb: Always select the dominant light sources in the scene. Using the viewing frustum to select light sources can be very dangerous, since you may have a nice giant 1000-mega watt photon busting spot light right behind the top of your head. It's not in your view frustum, but it's going to be responsible for the most distinct shadows you would see in the scene. Just remember, the fewer the number of lights, the more cycles and rendering passes you can save for other visually more important effects. So choose with care!

From the released screen shots of the upcoming Doom3 engine, I estimate that id Software would have to limit the number of shadow casting lights in any scene to a maximum of say 4 or 5. Well, we will know when Doom3 hits the shelves next year.

Silhouette Determination

The very first step to constructing a shadow volume is to determine the silhouette of the occluder. The stencil shadow algorithm requires that the occluders be closed triangle meshes. This meant that every edge in the model must only be shared by 2 triangles thus disallowing any holes that would expose the interior of the model. We are only interested in the edges shared by a triangle that faces the light source and another triangle that face away from the light source. There are many ways to calculate the silhouette edges and every single one of these methods are CPU cycles hungry. Lets assume we are working with an indexed triangle mesh.

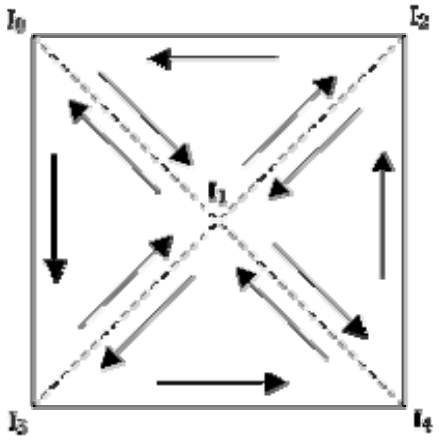


Figure 10: Edge elimination for silhouette determination

Figure 10 shows one side of a box that is made up of four triangles with a consistent counter-clockwise winding. The broken lines indicate the redundant internal edges since we are only interested in the solid line that forms the outline of the box. The redundant internal edges are indexed twice as they are shared by two triangles. We take advantage of this property to come up with a simple method to determine the silhouette edges.

1. Loop through all the model's triangles
2. If triangle faces the light source (dot product > 0)
3. Insert the three edges (pair of vertices), into an edge stack
4. Check for previous occurrence of each edges or it's reverse in the stack
5. If an edge or its reverse is found in the stack, remove both edges
6. Start with new triangle

The above algorithm will ensure that the internal edges would be eventually removed from the stack since they are indexed by more than one triangle.

Eric Lengyel [11] presented another silhouette determination algorithm that makes use of the consistent winding (counterclockwise) of vertices. The method requires 2 passes on all the triangles of the model to filter in all the edges shared by pairs of triangles. The resultant edges list then undergo the dot product operations to get the edges that are shared by a light facing triangle and a non light facing triangle.

It is important to note that silhouette determination is one of the two most expensive operations in stencil shadow volume implementation. The other is the shadow volume rendering passes to update the stencil buffer. These two areas are prime candidates for aggressive optimizations, which we will discuss in detail at the concluding sections of this paper.

Generating Shadow Volume Capping

Remember that shadow volume capping is only necessary for the depth-fail technique. The purpose of doing shadow volume capping is to ensure that our shadow volume is closed, and it must be closed even at infinity. Interestingly, the extrusion of geometries for point light sources and infinite directional light sources are different. Point light sources would extrude the silhouette edges exactly point for point while infinite directional light sources would extrude all silhouette edges to a single point at infinity. This would meant that the shadow volume's back capping would be redundant for infinite directional light sources as it is already closed.

The ideal time to generate the front and back capping would be during the silhouette generation since we are already generating the angles between the light vector and the edges. For the front cap, we just need to duplicate all front facing geometries and use these geometries for extrusion to form the back capping as well. Note that the back cap is only necessary for point light sources.

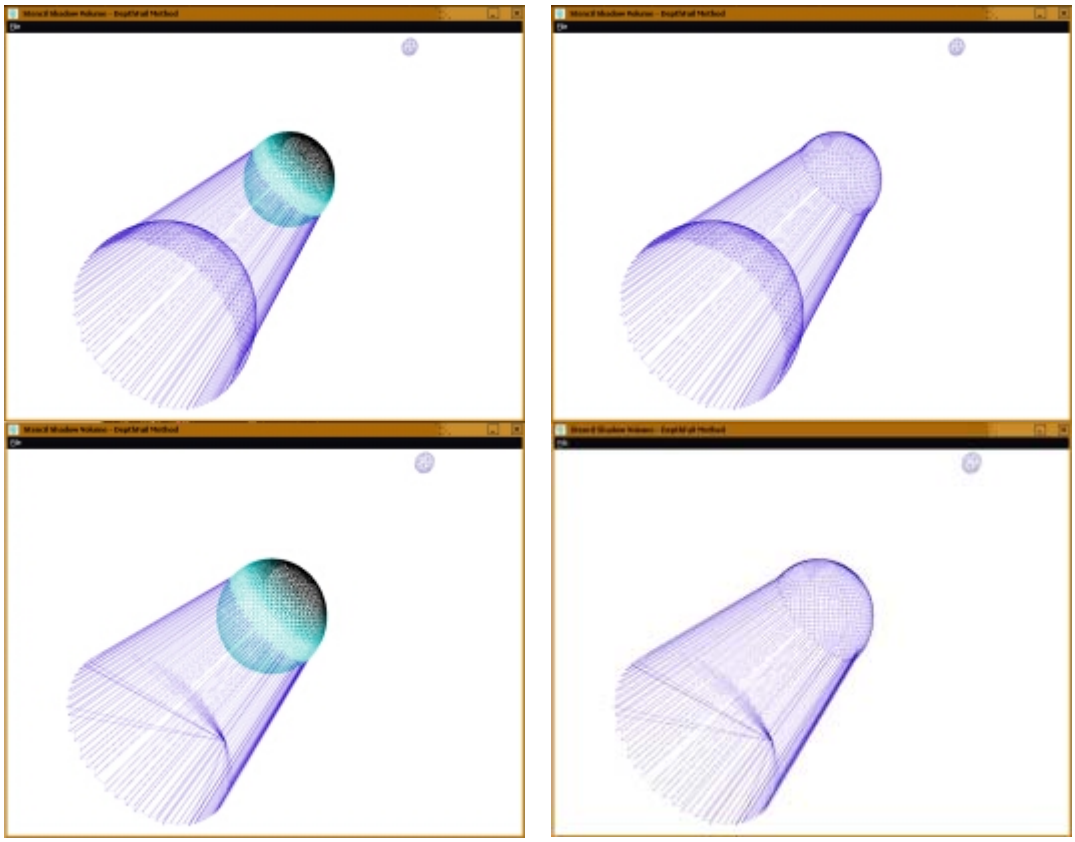


Figure 11: Closed shadow volume with point light source

Figure 11 shows two sets of images employing different geometries to close the shadow volume. The first row depicts a closed shadow volume formed by a front and back capping reusing light facing geometries. The second row shows a closed shadow volume with a front cap that reuses light facing geometries of the occluder and a triangle-fan back cap constructed from extruded silhouette edges. The triangle-fan back cap should be used as it results in less geometry and hence requires less memory and rendering time. When reusing the front facing geometries of the occluder, we should be extremely careful with regards to rendering the shadow volume since the shadow volume's front capping geometries are physically coplanar with the occluder's front facing geometries. Most often than not, precision problems will cause the front capping geometries of the shadow volume to be rendered in front of the occluder's front facing geometries causing the entire occluder to be engulfed in its own shadow volume. We can make use of the D3DRS_ZBIAS flag in Direct3D's D3DRENDERSTATETYPE to force the occluder's front facing geometries to be rendered in front of its shadow volume front cap. Simply use the D3DRS_ZBIAS flag when setting the render state (e.g. pd3dDevice->SetRenderState(D3DRS_ZBIAS, value)). We set the flag value to a higher value for the occluder's geometries and a lower value for its shadow volume. This will ensure that the front cap of the shadow volume is rendered behind the occluder's front facing geometries.

Extruding Geometries To Infinity

As discussed previously, we need to extrude the silhouette edges to infinity to avoid the situation shown in Figure 5 where a finite shadow volume extrusion fails to cover all the shadow receivers in a scene. However, it is not compulsory to extrude the silhouette edges to infinity if we can ensure that the situation in Figure 5 never happens in our scene. In practical cases, a large value would normally be more than adequate.

Mark Kilgard [2] introduced the trick of using the w value of homogenous coordinates to render semi-infinite vertices. In 4D homogenous coordinates, we represent a point or vector as (x, y, z, w) with w being the 4^{th} coordinate. For points, w is equal to 1.0. For vectors, w is equal to 0.0. The homogeneous notation is extremely useful for transforming both points and vectors. Since translation is only meaningful to points and not vectors, the value of w plays an important role in transforming only points and not vertices. This can be easily deduced since the translation values of a transformation matrix are on either the 4^{th} column or the 4^{th} row depending on the matrix convention. By setting the w value of the infinity-bound vertices to 0.0, we change the homogenous representation from that of a 3D point to a 3D vector. The rendering of a vector (w = 0.0) in clip space would be semi-infinite. It is important to note that we should only set the w values to 0.0 after transformation to clip space. In Direct3D, this would mean the combined transformation of the world, view and projection matrices. This is because when we set the flexible vertex format to D3DFVF_XYZRHW, we are bypassing Direct3D's transformation and lighting pipeline. Direct3D assumes that we had already transformed and lit the vertices. Ideally, the extrusion of geometries should be done in a vertex program since

we are already working in clip space in a vertex shader. In fact, vertex shaders and stencil shadow volumes is a match made in heaven. We will discuss the benefits of doing shadow volumes in a vertex program at the end of this paper.

While extruding geometries by a huge distance or to infinity helps to avoid the problem of finite shadow volume cover, it also generates another problem. Imagine two players in a dungeon First-Person-Shooter (FPS) game, roaming in adjacent rooms separated by a solid brick wall. The table lamp in one of the room causes one of the players to cast a shadow onto the brick separating the rooms. The player on the other room would see the shadow cast by the table lamp since the shadow volume extrudes out to infinity. The solid brick wall suddenly becomes like a thin piece of paper with a "ghost" shadow on it. Luckily, we can avoid this kind of situation in the very first place by culling away the shadow casting player's avatars using occlusion-culling techniques. Figure 12 shows a more awkward situation whereby the camera sees both the occluder and the occluder's ghost shadow on the other side of the terrain. This scenario is very possible especially for flight simulations or aerial combat games. The only possible solution to avoid both the finite shadow volume cover (Figure 5) and ghost shadow (Figure 12) is to impose limitations on the placing of light sources and occluders in a scene. If we can be sure that an occluder can never get closer than a certain distance of a shadow casting light source, then we can safely estimate the largest distance we would need to extrude the shadow volume in order to provide adequate shadow cover while not causing ghost shadows.

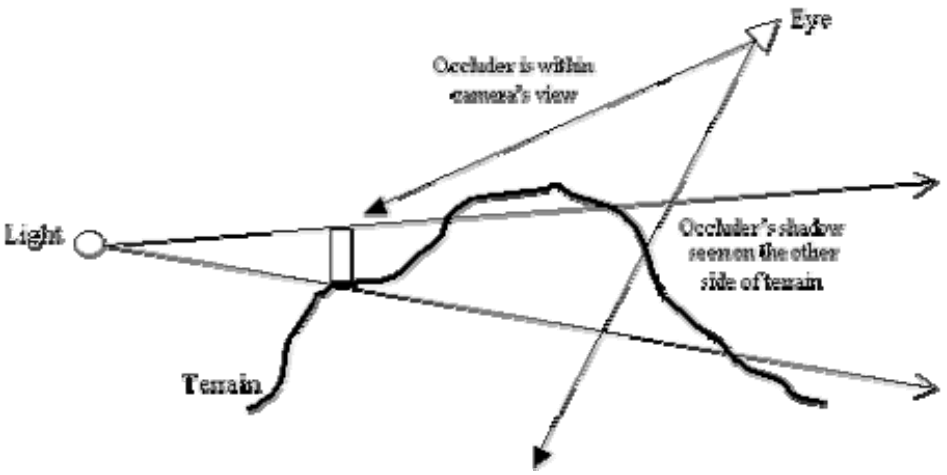


Figure 12: Ghost shadow effect due to large extrusion distance

View Frustum Clipping – The Ultimate Evil

It is time to confront the greatest evil in stencil shadow volumes: View frustum clipping. Clipping is a potential problem to any 3D rendering technique because we rely on a perspective projection view of our 3D worlds. The view frustum requires a near clipping distance and a far clipping distance, for the creation of a near clip plane and a far clip plane. Both the depth-pass and depth-fail

techniques suffer from view frustum clipping problem. Depth-pass technique suffers from errors when the shadow volume gets clipped after intersecting the near clip plane as shown in Figure 13. The red arrow represents one case whereby the stencil values for the associated fragment will be wrong due to the clipping of the shadow volume's front face.

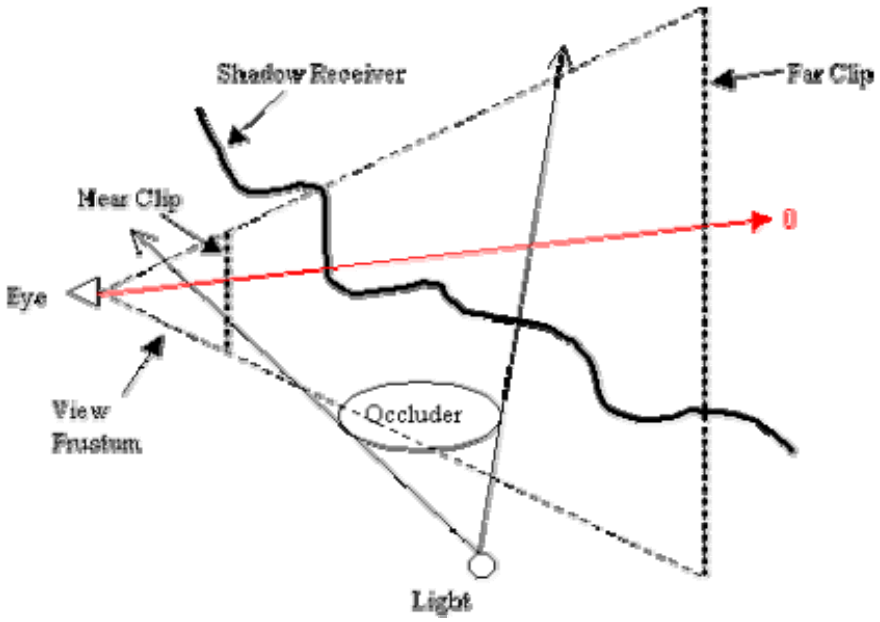


Figure 13: Shadow volume clipped at near clip plane causing depth-pass errors

On the other hand, depth-fail technique suffers from errors arising due to the clipping of the shadow volume with the far clip plane. Since the far clip plane is at a finite distance from the eye position, the depth-fail technique will almost certainly produce the wrong result when the shadow volume gets clipped at the far plane. The red arrow in Figure 14 represents a case whereby the depth-fail technique will generate errors since the back face of the shadow volume had been clipped at the far plane.

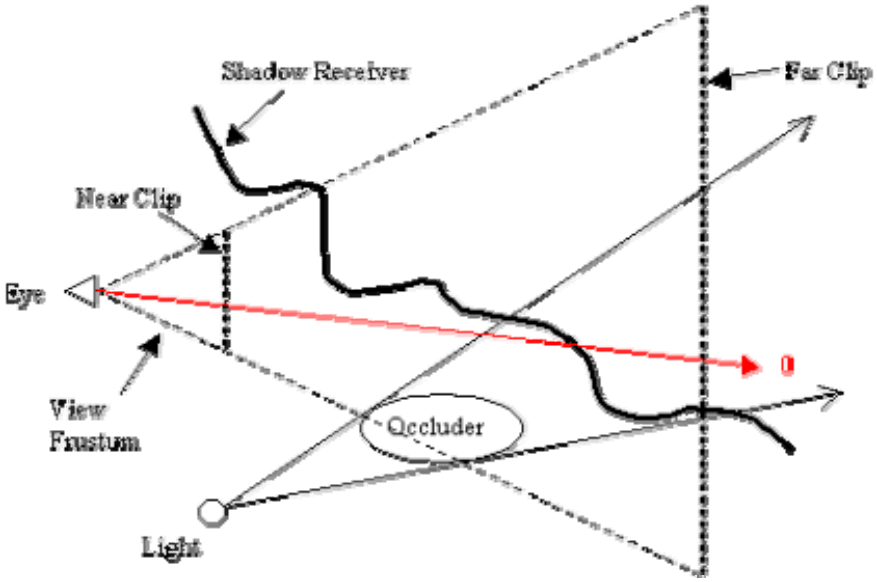


Figure 14: Shadow volume clipped at far clip plane causing depth-fail errors

We can solve the clipping problems by adjusting the clipping planes, but it is not always advisable to do so. For example, moving the near clip plane will greatly affect the depth precision and may have negative impacts on other operations that uses the depth buffer.

Mark Kilgard [2] presented an interesting idea of handling the two possible scenarios when shadow volumes intersect the near clip plane. The idea was to "cap" the shadow volume at the near clip plane, so that the previously clipped front facing geometries can now be rendered at the near clip plane. The first scenario is when all the vertices of the occluder's silhouette projects to the near clip plane. In this case, a quad strip loop is generated from all front facing vertices within the silhouette of the occluder. The quad strip loop is then projected onto the near clip plane thus forming a capping for the shadow volume.

The second scenario occurs when only part of the shadow volume projects onto the near clip plane. This proves to be very much more difficult to handle than the previous scenario. To his credit, Kilgard devised an elaborate system to filter out the vertices of triangles (facing away from the light) that should be projected onto the near clip plane in order to cap the shadow volume. The capping of shadow volumes at the near clip plane gave rise to another problem: depth precision. Rendering geometries at the near clip plane is analogous to rolling a coin along a razor's edge; the coin can drop down both sides easily. What this means is that the near plane may still clip the vertices that were meant to cap the shadow volume. To overcome this, Kilgard devised yet another method that builds a depth range "ledge" from the eye point to the near plane. The idea is to render the shadow volume from a depth range of 0.0 to 1.0, while normal scene rendering occurs within a depth range of 0.1 to 1.0. The ledge could be build into the view frustum by manipulating the perspective projection matrix. Once in place, the near clip plane capping of shadow volumes is done at a depth value of 0.05, which is half of the ledge. This idea is indeed original but it does not solve the problem totally. Cracks or "holes" in the near plane shadow cap occurs very frequently resulting in erroneous results. The conclusion with the near clip plane problem is that there are really no trivial solutions. At least, there is no known foolproof solution to the problem at the time of this writing. This makes the depth-pass technique very undesirable.

Fortunately, there is an elegant solution to the far plane clipping problem that plagues the depth-fail technique. The antidote to the problem is simply to use an infinite perspective view projection or simply an infinite view frustum. By projecting a far plane all the way to infinity, there is no mathematical chance of the shadow volume being clipped by the far plane when we are rendering the shadow volume. Even if the shadow volume were extruded to infinity, the far plane at infinity would still not clip it! Eric Lengyel presented the mathematic derivation for OpenGL perspective projection matrix in [11]. We are going to deal with Direct3D perspective projection matrix here. Lets start by looking at a standard left-handed perspective projection matrix in Direct3D:

$$P = \begin{bmatrix} \cot\left(\frac{fov_w}{2}\right) & 0 & 0 & 0 \\ 0 & \cot\left(\frac{fov_h}{2}\right) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & \frac{-fn}{f-n} & 0 \end{bmatrix} \quad (1)$$

Variables:
n: near plane distance
f: far plane distance
fov_w: horizontal field of view in radians
fov_h: vertical field of view in radians

A far plane at infinity means that the far plane distance needs to approach ∞. Hence, we get the following perspective projection matrix when the far plane distance goes towards the infinity limit:

$$P_\infty = \lim_{f \to \infty} P = \begin{bmatrix} \cot\left(\frac{fov_w}{2}\right) & 0 & 0 & 0 \\ 0 & \cot\left(\frac{fov_h}{2}\right) & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -n & 0 \end{bmatrix} \quad (2)$$

Equation (2) defines a perspective projection view that extends from the near plane to a far plane at infinity. But, are we absolutely sure that the vertices that we extruded to infinity using the 4D homogeneous vector does not get clipped at infinity? Sadly, we cannot be 100% sure of this due to limited hardware precision. In reality, graphics hardware sometimes produces points with a normalized z-coordinate marginally greater than 1. These values are then converted into integers for use in the depth buffer. This is going to weak havoc since our stencil operations depends wholly on the depth value testing. Fortunately, there is workaround for this problem. The solution is to map the z-coordinate values of our normalized device coordinates from a range of [0, 1] to [0, 1-e], where e is a small positive constant. What this means is that we are trying to map the z coordinate of a point at infinity to a value that is slightly less than 1.0 in normalized device coordinates. Let $\boldsymbol{D_z}$ be the original z-coordinate value and $\boldsymbol{D\varepsilon_z}$ be the mapped z-coordinate. The mapping can be achieved using equation (3) shown below:

$$D'_z = D_z(1-\varepsilon) \quad (3)$$

Now, let's make use of equation (2) to transform a point ***A*** from camera space (***A*****cam**) to clip space (***A*****clip**). Note that camera space is also commonly referred to as eye space.

$$A_{clip} = P_\infty A_{cam} = \begin{bmatrix} A_x & A_y & A_z & A_w \end{bmatrix} \begin{bmatrix} \cot\left(\frac{fov_w}{2}\right) & 0 & 0 & 0 \\ 0 & \cot\left(\frac{fov_h}{2}\right) & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -n & 0 \end{bmatrix}$$

Which would gives us:

$$A_{clip} = \begin{bmatrix} A_x \cot\left(\frac{fov_w}{2}\right) \\ A_y \cot\left(\frac{fov_h}{2}\right) \\ A_z - nA_w \\ A_z \end{bmatrix} \quad (4)$$

Let's factor the desired range mapping into equation (3) by replacing D_z with $\frac{(A_{clip})_z}{(A_{clip})_w}$ and $D\varepsilon_z$ with $\frac{(A'_{clip})_z}{(A_{clip})_w}$:

$$\frac{(A'_{clip})_z}{(A_{clip})_w} = \frac{(A_{clip})_z}{(A_{clip})_w}(1-\varepsilon) \quad (5)$$

Simplifying equation (5) by using the values given by equation (4), we get:

$$(A'_{clip})_z = A_z(1-\varepsilon) + nA_w(\varepsilon - 1) \quad (6)$$

Using equation (6), we can enforce our range mapping into the projection matrix P_∞ given by equation (2) to get the following:

$$P'_\infty = \begin{bmatrix} \cot\left(\frac{fov_w}{2}\right) & 0 & 0 & 0 \\ 0 & \cot\left(\frac{fov_h}{2}\right) & 0 & 0 \\ 0 & 0 & (1-\varepsilon) & 1 \\ 0 & 0 & n(\varepsilon - 1) & 0 \end{bmatrix} \quad (7)$$

Thus, we can use the perspective projection matrix given in equation (7) without fear of far plane clipping of shadow volumes occurring at infinity! You might

wonder whether stretching the view frustum volume all the way to infinity would impact the depth buffer precision. The answer is, yes it does affect precision, but the loss of precision is really negligible. The amount of numerical range lost when extending the far plane out to infinity is only $\frac{n}{f}$. Say our original near clip plane is at 0.1 meter and far clip plane is at 100 meters. This range corresponds to a depth range of [-1.0, 1.0]. We then extend the far plane distance to infinity. The range from 0.1 meter to 100 meters would now correspond to a depth range of [-1, 0.999]. The range from 100 meters to infinity would correspond to a depth range of [0.999, 1.0]. The loss in depth buffer precision is really not a big impact at all. The larger the difference between the *n* and *f* values, the smaller the loss in depth buffer precision. You can find the above derivations and many other related mathematical derivations in Eric Lengyel's book [12]. It should be noted that using an infinite view frustum meant that we have to draw more geometries. This may pose a potential performance problem.

The infinite view frustum projection is really just a software solution to the far plane clipping problem. Mark Kilgard and Cass Everitt [10] presented a hardware solution to the problem instead of using an infinite view frustum. Newer graphics hardware now supports a technique called "depth-clamping". In fact, the depth-clamping extension, NV_depth_clamp, was specifically added to Nvidia's GeForce3 and above graphics cards to solve the far plane clipping problem for shadow volumes. When active, depth-clamping would force all the objects beyond the far clip plane to be drawn at the far clip plane with the maximum depth value. This meant that we can project the closed shadow volume to any arbitrary distance without fear of it being clipped by the far plane as the hardware will handle the drawing properly. With such automatic support from graphics hardware, depth-fail shadow volumes become very easy to implement. We can extend the shadow volume to infinity while rendering with our finite view frustum and still get correct depth-fail stencil values! Well, the tradeoff is hardware dependence. If we want the depth-fail shadow volume to work for any graphics card (with stenciling support), we will have to use the infinite view frustum projection instead of the depth-clamping extension.

Depth-Pass or Depth-Fail

We had run through most of the method and implementation issues of both the depth-pass and depth-fail techniques for doing stencil shadow volumes. So which method should we use in our games? Lets take stock of the pros and cons of both techniques.

Depth-pass

- Advantages
  - Does not require capping for shadow volumes
  - Less geometry to render
  - Faster of the two techniques
  - Easier to implement if we ignore the near plane clipping problem

    - Does not require an infinite perspective projection
- Disadvantages
    - Not robust due to unsolvable near plane clipping problem

Depth-fail

- Advantages
    - Robust solution since far plane clipping problem can be solved elegantly
- Disadvantages
    - Requires capping to form closed shadow volumes
    - More geometry to render due to capping
    - Slower of the two techniques
    - Slightly more difficult to implement
    - Requires an infinite perspective projection

It seems that depth-pass is the better technique of the two, but we must remember that it will totally fail when our camera enters a shadow volume. Until there is a feasible solution for the near plane clipping problem, the depth-fail technique is still required if a robust implementation is desired. The selection between the two techniques depends heavily on the constraints of the games that we are developing. If shadow casting is required for a top down or isometric view game such as Diablo, the depth-pass technique will suffice. On the other hand, for FPS games, it would be almost impossible to avoid the situation of camera entering a shadow volume. In this case, depth-fail technique is the only feasible solution. Of course, we must also not forget about other shadowing techniques such as shadow mapping. In certain situations where the shadow casters in a scene are too small for any self-shadowing to be visible, it would be wiser to just use projective shadow mapping. For realistic soft shadows, it may also be done more cheaply using shadow maps.

On the whole, it is beneficial to combine other techniques with shadow volumes to achieve better quality shadows. One example of such hybrid implementation is the Power Render X [16] game engine that generates shadows using shadow volumes and then fade out the shadows with respect to distance from the occluder by using projective textures.

The Buzzwords: Robustness and Efficiency

Doing realistic and accurate shadows in games is no longer enough as the complexity of games had skyrocketed during the past 10 years. We need to provide robust and yet efficient implementations of stencil shadow volumes. In the case of robustness, using the depth-fail technique should suffice for almost any situations imaginable. However, hardware limitations and poor frame rates will sometimes push the depth-fail technique beyond our computation budget. There are many ways to optimize our shadow volume implementation so as to

create nice looking shadows and yet hold the frame rate above that all-important 20fps benchmark.

The real bottlenecks in a stencil shadow volume implementation are silhouette determination and shadow volume rendering. The former requires a huge amount of CPU cycles and it worsens if the occluders had high polygon counts. The latter is a huge consumer of invisible fill rate. One obvious way to alleviate the CPU crunch during silhouette determination is to use a lower polygon model of the occluder. Another effective way is to determine a new silhouette only every 2-4 frames. This is based on the assumption that the light's position or the occluder's position does not change very drastically within 2-4 frames. This assumption turns out to be pretty good for most cases.

Remember that the extra capping geometries used to form a closed shadow volume in the depth-fail technique contributed to depth-fail being a more expensive method? We can drastically reduce the capping geometries for occluders that have relatively little detail on the surfaces that frequently face the light. Little detail here means fewer geometric details, which implies that the surface is rather flat and would usually produce near or fully convex silhouette hulls. If that is the case, we can often create a triangle strip to be used as the front cap to close the shadow volume. We should note that this is an approximation and hence would result in shadows that are not correct at certain angles. However this approximation should work very well for small objects.

For Direct3D implementations, it is also advisable to use "welded" meshes. A welded mesh simply means that there are no duplicated vertices representing the exact same point. To see an example of an "unwelded" mesh, open the mesh viewer tool and create a cube. Look at the vertices information of the cube and you will see that there are 24 instead of just 8 vertices. This is unavoidable since Direct3D's version of a vertex contains color and normal information that cannot be shared by different faces referring to the same point; hence extra vertices are generated for different faces. The extra vertices are redundant but could not be removed during the silhouette calculation without considerable amount of comparison work. It is therefore wiser to use welded meshes for silhouette determination. The Direct3D mesh viewer utility provides a nifty option to do just that. Click MeshOps then Weld Vertices, check Remove Back To Back Triangles, Regenerate Adjacency and Weld All Vertices before welding. Alternatively, we can also make use of the mesh function D3DXWeldVertices to weld the mesh ourselves.

For Direct3D implementations, it is also advisable to use "welded" meshes. A welded mesh simply means that there are no duplicated vertices representing the exact same point. To see an example of an "unwelded" mesh, open the mesh viewer tool and create a cube. Look at the vertex information for the cube and you will see that there are 24 instead of just 8 vertices. This is unavoidable since Direct3D's version of a vertex contains color and normal information that cannot be shared by different faces referring to the same point; hence extra vertices are generated for different faces. The extra vertices are redundant but could not be removed during the silhouette calculation without considerable amount of comparison work. It is therefore wiser to use welded meshes for

silhouette determination. The Direct3D mesh viewer utility provides a nifty option to do just that. Click MeshOps then Weld Vertices, check Remove Back To Back Triangles, Regenerate Adjacency and Weld All Vertices before welding. Alternatively, we can also make use of the mesh function D3DXWeldVertices to weld the mesh ourselves.

Regarding the invisible fill rate, they are really unavoidable. However, we could probably lessen the impact by setting the D3DRS_COLORWRITEENABLE render state in Direct3D before rendering the shadow volume. We can use it to turn off the red, green, blue and alpha channel drawing since we are only interested in filling the stencil buffer.

Another area that we should take note of is the management of shadow casting lights in our 3D scene. Good management of light sources will invariantly benefit the shadow volume generation process. The rule of thumb is to keep the number of shadow casting light sources below a maximum of 4 at any one time. Future hardware or improved algorithms would nullify the previous statement, but for now it serves as a good guideline and would probably remain so for the next 2 years at least. The important aspect of light source management is the method used for selecting which light sources should be included in the shadow volume generation process. The main parameters that should be taken into considerations could be intensity, distance from viewer, relevance to current game play and lastly visual importance. Take a look at the excellent article [4] by Charles Bloom regarding the selection of light sources for shadow casting.

Let's discuss some high level optimization that we can employ to speed up our shadow volume enabled games further. We can actually make use of the depth-pass technique when we are sure that the camera is not within any shadow volumes. This can be done rather easily by forming a near-clip volume. The light source's position and the four sides of the near plane are used to define a pyramid. The near plane closes the pyramid and thus forms the near-clip volume. If an occluder lies completely outside this volume, we can safely employ the depth-pass technique since the occluder's shadow volume has no chance of intersecting the near plane. Eric Lengyel also described utilizing OpenGL scissor rectangle support to cut down the fill rate penalty for rendering the shadow volumes and the illuminated fragments. However, comprehensive high-level scissor rectangle support is not yet available in DirectX 8.1. For details of these two optimizations, please refer to [11].

Lastly, we should aggressively utilize any hardware supports that are available. Future GPUs would be expected to support two-sided stencil testing, which will allow us to render the front and back faces of shadow volumes together. This would provide great savings when rendering the shadow volume by halving the geometry setup cost, vertex transformation cost and geometry transfer cost since we would only need to push the shadow volume geometries through the pipeline once. The hardware will take care of the front face and back face culling automatically while doing 2 stenciling pass on the same set of geometries. Hardware depth-clamping support should also be used to clip the shadow volume geometries to the far plane at no extra costs. Finally, lets look

at one of the most important modern graphics hardware capability that we should take full advantage of: **Vertex Shaders**.

Shadow Volumes Powered By Vertex Shaders

Among the whole host of improvements to commercial graphics hardware, the introduction of programmable vertex processing pipeline (vertex shaders) is perhaps the best thing that can happen to anyone implementing shadow volumes. The biggest advantage of doing shadow volumes in a vertex program is that we do not need to upload the shadow volume geometries whenever it is generated. The entire shadow volume could reside on hardware memory as static vertex buffers. The data bandwidth saved can be quite substantial. Furthermore, floating point operations done in programmable vertex hardware are incredibly fast. However, we need to note here that implementing shadow volume fully using vertex program may actually degrade performance in certain circumstances. We will go into this at the end of this section.

To leverage the power of vertex shaders, we need to preprocess our occluder's geometry first. Current vertex shader hardware does not have the capability of generating new vertices on the fly. It is strictly a 1 vertex in and 1 vertex out pipeline. This poses a problem since we need to create new vertices from the silhouette edges in order to form a shadow volume. The solution is to create all the additional vertices that are needed during preprocessing. Once in the vertex shader, we generate the shadow volume using these additional vertices. Lets look at how this is done.

We need to create a quad for every edge (2 vertices) that is shared by exactly 2 faces. The quad can be viewed as a "degenerate" quad formed by the original edge shared by 2 different faces. Both the faces contribute the same edge to the degenerate quad. Since the edges from both faces are similar, positional wise, the degenerate quad is "zero length". The only difference is that the edges hold the normal information of their respective face. Once in the vertex program, we dot the light vector and the vertex normal. If the result is positive, the vertices pass through the vertex program untouched. If the result is negative, we extrude it in the direction of the light vector. This technique would elegantly generate a closed shadow volume as light facing geometries are left untouched to form the front capping while geometries that faces away from the light are extruded to form the sides of the shadow volume and the back capping.

If you are unsure about how it works, try this example. Imagine a sphere mesh with a point light source to its left. The entire left hemisphere faces the light and hence all the geometries that define the left hemisphere are left untouched to form the front capping. The entire right hemisphere however faces away from the light. Hence, all the geometries that define the right hemisphere are

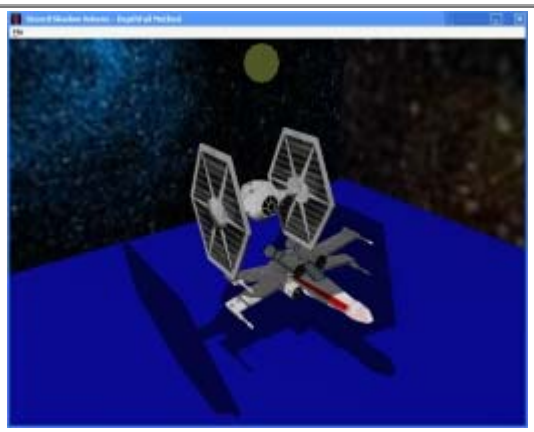
extruded to form the back capping. The sides of the shadow volume are formed auto-magically by the degenerated quads residing along the silhouette edges. In this case, the silhouette edges forms exactly a vertical line down the middle of the sphere. This works because exactly 1 edge per degenerate quad from the silhouette edges is extruded. The previously degenerate quad now becomes a normal quad that defines the shadow volume's sides. Chris Brennan presented a short article in [15] on implementing the extrusion of shadow volume in a vertex program.

We should note that the preprocessing needed creates a lot of additional geometries. In fact, only the degenerate quads along the silhouette edges are useful. The rest are simply dormant but are still being pushed through the processing pipeline. However, shadow volume generation can now be done completely on the graphics hardware and performance is generally much better than non-shader implementation in most cases.

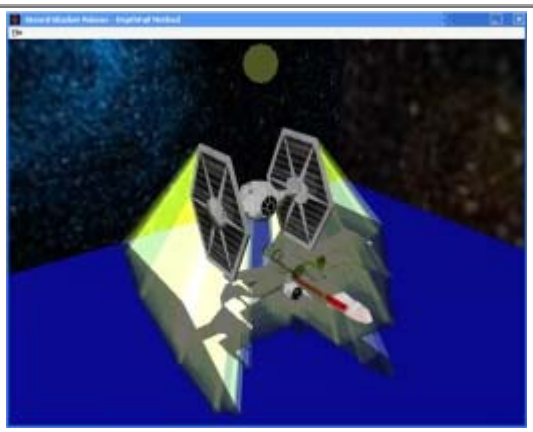
Recently Mark Kilgard pointed out that computing the silhouette edges within the vertex shader may be detrimental to performance if the occluders have high polygon counts or if there are a lot of shadows casting light sources. This assessment stern from the fact that we need to push more vertices into the pipeline and all of these have to passed through the silhouette edges testing within the vertex shader. Consequently, occluders with high polygon counts would generate large amount of wasted vertices (degenerate quads), and the cost of testing all these extra vertices may not cover the geometry upload savings we get by using vertex shaders! Having more light sources will obviously worsen such vertex shader implementation further. Hence, implementation of shadow volume on programmable vertex hardware should be thoroughly tested to ensure that we have a net performance gain over implementation utilizing the CPU. If the CPU is needed for heavy A.I. or game logic computation, a vertex shader implementation of shadow volumes may be more efficient. However, it might also be better in many cases to just use vertex shader as an assist instead of trying to do everything within the vertex shader. The moral of the story is: ***always remember to turn on everything (A.I., Physics, Sound, Input, Network, Renderer etc) in your game and benchmark, benchmark and benchmark again!***

Lastly, a more extensive and in-depth article on the stencil shadow volume technique will be available in the upcoming book ShaderX2 (www.shaderx2.com). The article in the book delves deeper into the algorithms involved in stencil shadow volume with detailed discussions of optimizations, workflow, and scene management and ‘cheats' employed in commercial 3D engines to speed up robust shadow volume implementations. There will also be 6 extensive samples that cover normal CPU, GPU implementation in assembly and GPU implementation using the new High Level Shader Language (DirectX9.0). The book is a compilation of many advance shader techniques by professionals and engineers working in the field. It will be available possibly in August 2003 and the editor is Mr Wolfgang Engel.

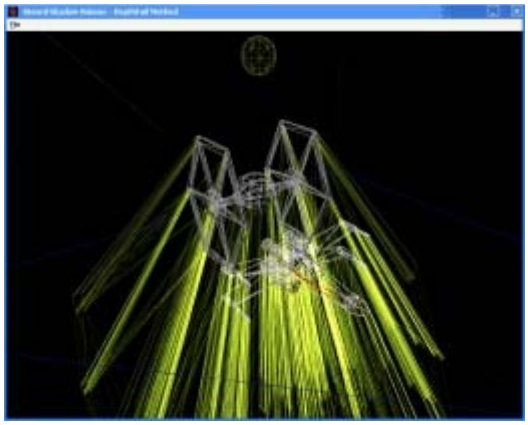
Shadow Volumes At Work



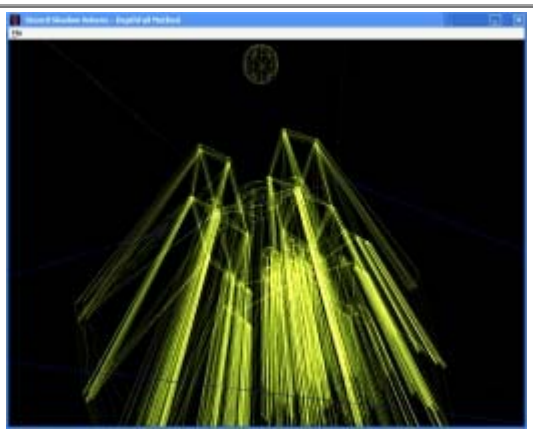
Depth-fail stencil shadow volume technique. Showcasing accurate self-shadowing and multiple occluder shadowing.



Front faces of shadow volume drawn to give a visual appreciation of the silhouette extrusion from the point light source.



Wire frame of the occluders and the shadow volume.



The occluder's geometries were omitted to show the front and back capping of the shadow volume. The light facing geometries forms the front cap.

Near plane clipping of shadow volume causes errors when the camera enters the shadow volume in the depth-pass technique.

Far plane clipping of shadow volume cause errors in the depth-fail technique.

Many thanks to Augustin Denis for providing the XWing and Tie fighter models.
(These models are fan art)

Reference

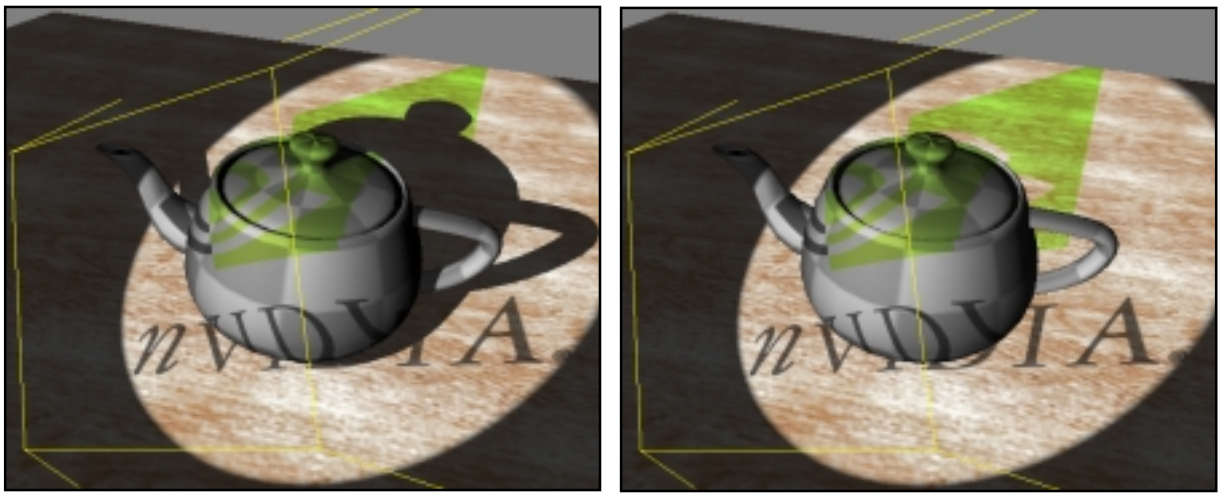
[1]: Mark Kilgard, http://developer.nvidia.com/docs/IO/1348/ATT/stencil.pdf
[2]: Mark Kilgard, http://developer.nvidia.com/docs/IO/1451/ATT/StencilShadows_CEDEC_E.pdf
[3]: http://developer.nvidia.com/view.asp?IO=inf_shadow_volumes
[4]: Charles Bloom, http://www.cbloom.com/3d/techdocs/shadow_issues.txt
[5]: Tim Heidmann, http://developer.nvidia.com/docs/IO/2585/ATT/RealShadowsRealTime.pdf
[6]: John Carmack, http://developer.nvidia.com/docs/IO/2585/ATT/CarmackOnShadowVolumes.txt
[7]: Bilodeau, Bill and Mike Songy. "Real Time Shadows", Creativity 1999, Creative Labs Inc. Sponsored game developer conferences, Los Angeles, California, and Surrey, England, May 1999.
[8]: Frank Crow. Shadows Algorithms for Computers Graphics. Computer Graphics, Vol. 11, No.3, Proceedings of SIGGRAPH 1977, July 1977.
[9]: Cass Everitt and Mark Kilgard, http://developer.nvidia.com/docs/IO/2585/ATT/RobustShadowVolumes.pdf
[10]: Cass Everitt and Mark Kilgard, http://developer.nvidia.com/docs/IO/2585/ATT/GDC2002_RobustShadowVolumes.pdf
[11]: Eric Lengyel, http://www.gamasutra.com/features/20021011/lengyel_01.htm
[12]: Eric Lengyel, "Mathematics for 3D Game Programming & Computer Graphics", Charles River Media, 2002
[13]: Tomas Moller, Eric Haines. "Realtime Rendering", 2nd Edition, A K Peters Ltd, 2002, ISBN: 1-56881-182-9.
[14]: Wolfgang F. Engel, Amir Geva and Andre LaMothe. "Beginning Direct3D Game Programming", Prima Publishing, 2001, ISBN: 0-7615-3191-2.
[15]: Wolfgang F. Engel. "Direct3D ShaderX Vertex and Pixel Shader Tips and Tricks", Wordware Publishing Inc, 2002.
[16]: Power Render X game engine. http://www.powerrender.com/prx/index.htm
[17]: Gamasutra website. http://www.gamasutra.com/

Hardware Shadow Mapping

Cass Everitt
cass@nvidia.com

Ashu Rege
arege@nvidia.com

Cem Cebenoyan
cem@nvidia.com



Introduction

Shadows make 3D computer graphics look better. Without them, scenes often feel unnatural and flat, and the relative depths of objects in the scene can be very unclear. The trouble with rendering high quality shadows is that they require a visibility test for each light source at each rasterized fragment. For ray tracers, adding an extra visibility test is trivial, but for rasterizers, it is not. Fortunately, there are a number of common cases where the light visibility test can be efficiently performed by a rasterizer. The two most common techniques for hardware accelerated complex shadowing are stenciled shadow volumes and shadow mapping. This document will focus on using shadow mapping to implement shadowing for spotlights.

Shadow mapping is an image-based shadowing technique developed by Lance Williams [8] in 1978. It is particularly amenable to hardware implementation because it makes use of existing hardware functionality – texturing and depth buffering. The only extra burden it places on hardware is the need to perform a high-precision scalar comparison for each texel fetched from the shadow map texture. Shadow maps are also attractive to application programmers because they are very easy to use, and unlike stenciled shadow volumes, they require no additional geometry processing.

Hardware accelerated shadow mapping [5] is available today on GeForce3 GPUs. It is exposed in OpenGL [4] through the SGIX_shadow and SGIX_depth_texture extensions [6], and in Direct3D 8 through a special texture format.

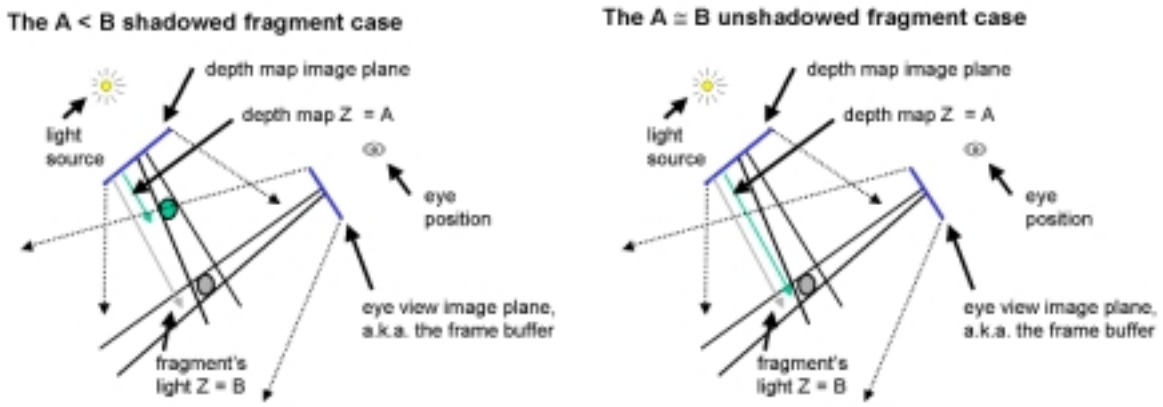


Figure 1. These diagrams were taken from Mark Kilgard's shadow mapping presentation at GDC 2001. They illustrate the shadowing comparison that occurs in shadow mapping.

How It Works

The clever insight of shadow mapping is that the depth buffer generated by rendering the scene from the light's point of view is a pre-computed light visibility test over the light's view volume. The light's depth buffer (a.k.a. the shadow map) partitions the view volume of the light into two regions: the shadowed region and the unshadowed region. The visibility test is of the form

$$p_z \leq shadow\_map(p_x, p_y)$$

where p is a point in the light's image space. Shadow mapping really happens in the texture unit, so the comparison actually looks like:

$$\frac{p_r}{p_q} \leq texture\_2D\left(\frac{p_s}{p_q}, \frac{p_t}{p_q}\right)$$

Note that this form of comparison is identical to the depth test used for visible surface determination during standard rasterization. The primary difference is that the rasterizer always generates fragments (primitive samples) on the regular grid of the eye's discretized image plane for depth test, while textures are sampled over a continuous space at irregular intervals. If we made an effort to sample the shadow map texture in the same way that we sample the depth buffer, there would be no difference at all. In fact, we can use shadow maps in this way to perform more than one depth test per fragment [2].

Figure 1 illustrates the depth comparison that takes place in shadow mapping. The eye position and image plane are shown, but they are not relevant to the visibility test because shadowing is view-independent.

Figure 2. A shadow mapped scene rendered from the eye's point of view (left), the scene as rendered from the light's point of view (center), and the corresponding depth/shadow map (right).

How To Do It

The basic steps for rendering with shadow maps are quite simple:

- render the scene from the light's point of view,
- use the light's depth buffer as a texture (shadow map),
- projectively texture the shadow map onto the scene, and
- use "texture color" (comparison result) in fragment shading.

Figure 2 shows an example scene with shadows, the same scene shown from the light's point of view, and the corresponding shadow map (or depth buffer). Note that samples that are closer to the light are darker than samples that are further away.

Since applications already have to be able to render the scene, rendering from the light's point of view is trivial. If it is available, polygon offset should be used to push fragments back slightly during this rendering pass.

Why Is Polygon Offset Needed?

If implemented literally, the light visibility test described in the previous section is prone to self-shadowing error due to it's razor's edge nature in the case of unshadowed objects. In the hypothetical "infinite resolution, infinite precision" case, surfaces that pass the visibility test would have depth *equal* to the depth value stored in the shadow map. In the real world of finite precision and finite resolution, precision and sampling issues cause problems. These problems can be solved by adding a small bias to the shadow map depths used in the comparison.

If the problem were only one of precision, a constant bias of all the shadow map depths would be sufficient, but there is also a less obvious sampling issue that affects the magnitude of bias necessary. Consider the case illustrated in Figure 3. When the geometry is rasterized from the eye's point of view, it will be sampled in different

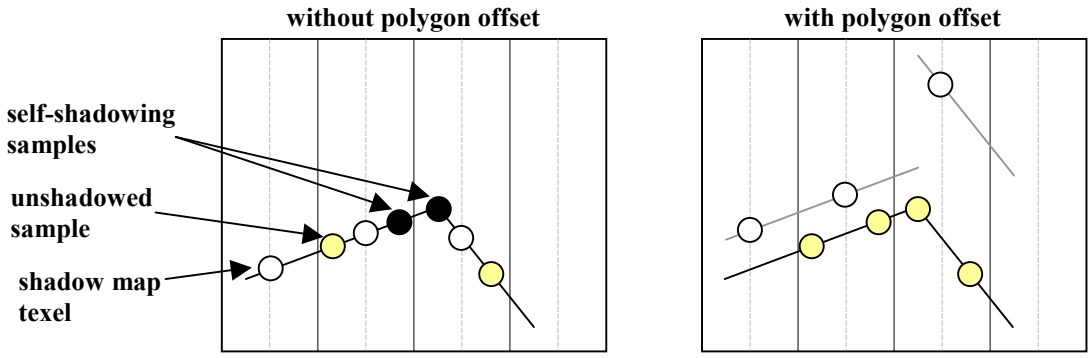


Figure 3. These figures illustrate the need for polygon offsetting to eliminate self-shadowing artifacts. The variable sampling location necessitates the use of z slope-based offset.

locations than when it was rasterized from the light's point of view. The difference in the depths of the samples is based on the slope of the polygon in light space, so in order to account for this we must supply a positive "slope factor" (typically about 1.0) to the polygon offset.

Direct3D does not expose polygon offset, so applications must provide this bias through matrix tweaks. This approach is workable, but because it fails to account for z slope, the uniform bias is generally much larger than it would otherwise need to be, which may introduce incorrectly unshadowed samples, or "light leaking".

The depth map as rendered from the light's point of view *is* the shadow map. With OpenGL, turning it into a real texture requires copying it into texture memory via glCopyTex{Sub}Image2D(). Even though the copy is card-local, it is still somewhat expensive. Direct3D's render-to-texture capability makes this copy unnecessary. You can render directly to the shadow map texture. This render-to-texture capability will also be available soon in OpenGL through extensions.

Once the shadow map texture is generated, it is projectively textured onto the scene. For shadow mapping, we compute 3D projective texture coordinates, where *r* is the sample depth in light space, and *s* and *t* index the 2D texture. Figure 4 shows these quantities, which are compared during rendering to determine light visibility.

Figure 4. A shadow mapped scene (left), the scene showing each sample's distance from the light source (center), and the scene with the shadow map shadow map projected onto it (right).

The final step in rendering shadows is to actually factor the shadow computation result into the shading equation. The result of the comparison is either 1 or 0, and it is returned as the texture color. If linear filtering is enabled, the comparison is performed at the four neighboring shadow map samples, and the results are bilinearly filtered just as if they had been colors.

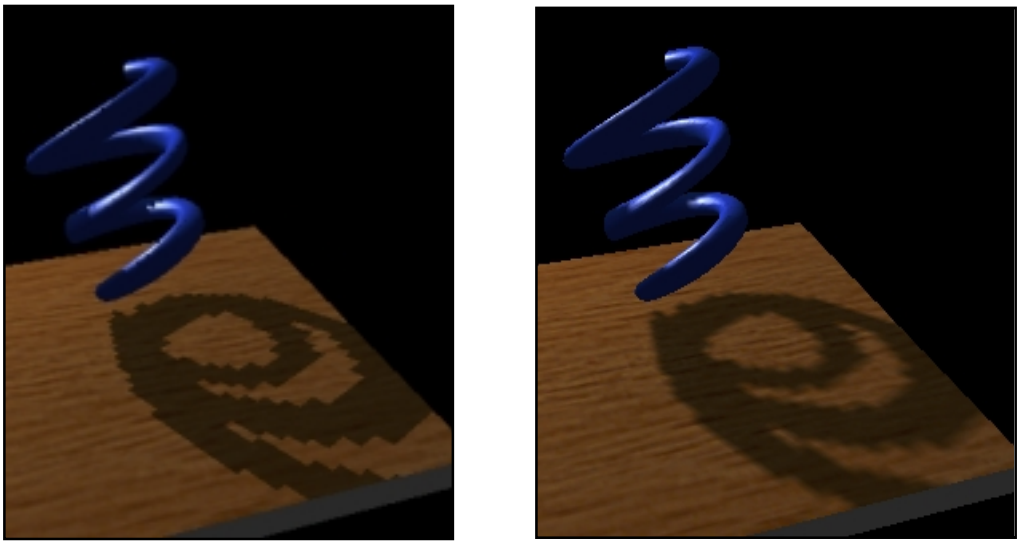


Figure 5. A very low resolution shadow map is used to demonstrate the difference between nearest (left) and linear (right) filtering for shadow maps. Credit: Mark Kilgard.

With GeForce3 hardware, it is easiest to use NV_register_combiners to implement the desired per-fragment shading based on the shadow comparison. One simple approach is to use the shadowing result directly to modulate the diffuse and specular intensity. Kilgard points out [3] that leaving some fraction of diffuse intensity in helps keep shadows areas from looking too "flat".

OpenGL API Details

Support for shadow mapping in OpenGL is provided by the SGIX_shadow and SGIX_depth_texture extensions. The SGIX_shadow extension exposes the per-texel comparison as a texture parameter, and SGIX_depth_texture defines a texture internal format of DEPTH_COMPONENT, complete with various bits-per-texel choices. It also provides semantics for glCopyTex{Sub}Image*() calls to read from the depth buffer when performing a copy.

Direct3D API Details

Support for shadow mapping in Direct3D is provided by special depth texture formats exposed in drivers version 21.81 and later. Support for both 24-bit (D3DFMT_D24S8) and 16-bit (D3DFMT_D16) shadow maps is included.

Setup

The following code snippet checks for hardware shadow map support on the default adapter in 32-bit color:

```
HRESULT hr = pD3D->CheckDeviceFormat(
        D3DADAPTER_DEFAULT,         //default adapter
        D3DDEVTYPE_HAL,             //HAL device
        D3DFMT_X8R8G8B8,            //display mode
        D3DUSAGE_DEPTHSTENCIL,      //shadow map is a depth/s surface
        D3DRTYPE_TEXTURE,           //shadow map is a texture
        D3DFMT_D24S8                //format of shadow map
);
```

Note that since shadow mapping in Direct3D relies on "overloading" the meaning of an existing texture format, the above check does not guarantee hardware shadow map support, since it's feasible that a particular hardware / driver combo could one day exist that supports depth texture formats for another purpose. For this reason, it's a good idea to supplement the above check with a check that the hardware is GeForce3 or greater.

Once shadow map support has been determined, you can create the shadow map using the following call:

```
pD3DDev->CreateTexture(texWidth, texHeight, 1,
        D3DUSAGE_DEPTHSTENCIL, D3DFMT_D24S8, D3DPOOL_DEFAULT,
        &pTex);
```

Note that you **must** create a corresponding color surface to go along with your depth surface since Direct3D requires you to set a color surface / z surface pair when doing a SetRenderTarget(). If you're not using the color buffer for anything, it's best to turn off color writes when rendering to it using the D3DRS_COLORWRITEENABLE renderstate to save bandwidth.

Rendering

Rendering uses the same ideas as in OpenGL: you render from the point of view of the light to the shadow map you created, then set the shadow map texture in a texture stage and set the texture coordinates in that stage to index into the shadow map at (s / q, t / q) and use the depth value (r / q) for the comparison. There are a few Direct3D-specific idiosyncrasies to be aware of, however:

- The (z / w) value used to compare with the value in the shadow map is in the range [0..$2^{bitdepth}$-1], not [0..1], where 'bitdepth' is the bitdepth of the shadowmap (24 or 16 bits). This means you have to put an additional scale factor into your texture matrix.
- Direct3D addresses pixels and texels in different ways [1], where integral screen coordinates address pixel centers and integral texture coordinates address texel boundaries. You need to take this into account when addressing the shadow map. There are two ways to do this: either offset the viewport by half a texel when rendering the shadow map, or offset by half a texel when addressing the shadow map.
- As stated earlier, there is no native polygon offset support in Direct3D. The closest thing is D3DRS_ZBIAS, but this doesn't help us when shadow mapping since it can only be used to bias depth a constant amount *towards* the camera, not away. Instead we can get similar functionality, albeit without taking into account polygon slope, by adding a small bias amount to our texture matrix.

Here is a sample texture matrix that takes into account these limitations:

```
float fOffsetX = 0.5f + (0.5f / fTexWidth);
float fOffsetY = 0.5f + (0.5f / fTexHeight);
D3DXMATRIX texScaleBiasMat( 0.5f,     0.0f,     0.0f,      0.0f,
                            0.0f,    -0.5f,     0.0f,      0.0f,
                            0.0f,     0.0f,     fZScale,   0.0f,
                            fOffsetX, fOffsetY, fBias,     1.0f );
```

Where fZScale is the ($2^{bitdepth}$-1) and fBias is a small negative value. Note that this matrix is applied post-projection, **not** in eye space.

Once the texture coordinates have been setup properly, the hardware will automatically compare (r / q) > shadowMap[s / q, t / q] and return zero to indicate in shadow or one to indicate in light (or potentially something in between if you're on the shadow edge and using D3DTEXF_LINEAR). The following pixel shader shows a simple use of shadow mapping (but note that you don't have to use pixel shaders to use shadow maps, DirectX7-style texture stage states work as well):

```
tex t0   // normal map
tex t1   // decal texture
tex t2   // shadow map
dp3_sat r0, t0_bx2, v0_bx2  //light vector is in v0
mul r0, r0, t2   //modulate lighting contribution by shadow result
mul r0, r0, t1   //modulate lighting contribution by decal
```

Advantages and Limitations

As with any technique, shadow mapping has certain advantages and limitations to be aware of. The fact that it is image-based turns out to be both an advantage and a limitation. It's advantageous, because it doesn't require additional application geometry processing, it works well with GPU-created and GPU-altered geometry and correctly handles fragment culling like alpha test. The associated limitation is that because it's image based, it works well for spotlights, but not point light sources. One could imagine a cube map –based shadow mapping system, but they would require six 90-degree frusta, which would each need to be fairly high resolution, and five more passes over the

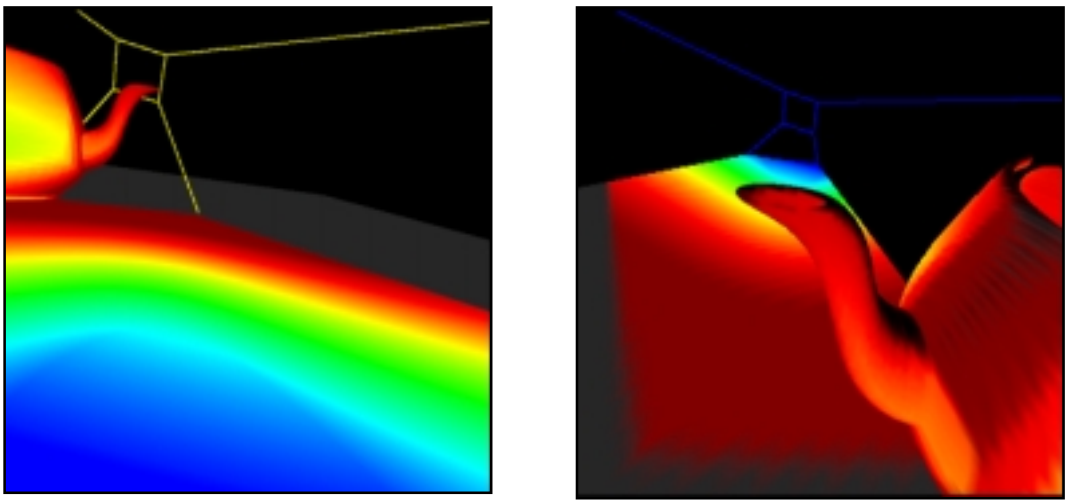


Figure 6. The "dueling frusta" problem occurs when the spotlight points toward the eye. The eye's view (left) shows the variation in sampling frequency of the shadow map, blue being the highest . The light's view (right) shows the very small portion of the light's image plane needs high frequency sampling.

geometry to generate the shadow map.

Along the same lines, the quality of shadow mapping depends on how well the shadow map sampling frequency matches the eye's sampling frequency. When the eye and light have similar location and orientation, the sampling frequencies match pretty well, but when the light and eye are looking toward each other, the sampling frequencies rarely match well. Figure 6 illustrates this "dueling frusta" situation.

Another problem that comes up with any projective texture mapping is the phantom "negative projection". This is actually pretty simple to remove at the cost of an additional texture unit, or per-vertex color. The goal is just to make sure that the shadow test always returns "shadowed" for surfaces behind the light.

Finally, the polygon offset fudge factor, while quite adequate for virtually all uses of shadow mapping, can be a bit dissatisfying. Andrew Woo [9] suggested an alternative shadow map generation that is produced from averaging the nearest and second-nearest depth layers from the light's point of view. This technique can actually be implemented as a two-pass technique on GeForce3 hardware using the depth peeling technique described in [2] and with a slight twist. In the second pass, the shadow map is used to peel away the nearest surfaces, but all depths are computed as the average of the

fragment's original depth and the nearest depth at that fragment's (x,y). The nearest surface (that is not peeled away), is then the average of the first and second nearest fragments!

Wang and Molnar introduce another technique to reduce the need for polygon offset [7]. Their technique works by rendering only back-faces into the shadow map, relying on the observation that back-face z-values and front-face z-values are likely far enough apart in z to not falsely self-shadow. This only helps front-faces, of course, but back-faces (with respect to the light) are, by definition, not in light, which helps hide artifacts. Note that this algorithm only works for closed polygonal objects.

Computing Transformations for Shadow Mapping

Computing the transformations required for shadow mapping can be somewhat tricky. This section provides details on the various transformations that need to be applied during the two render passes. While this section provides details for the OpenGL case, the transformations required for Direct3D are very similar with the main exception being that the texture coordinate generation is done directly via a matrix instead of the *texgen* planes. Also, keep in mind that the scale-bias matrix in Direct3D requires an additional offset to account for the discrepancy between pixel and texel coordinates as mentioned earlier, and that *eye linear* texgen is called D3DTSS_TCI_CAMERASPACEPOSITION.

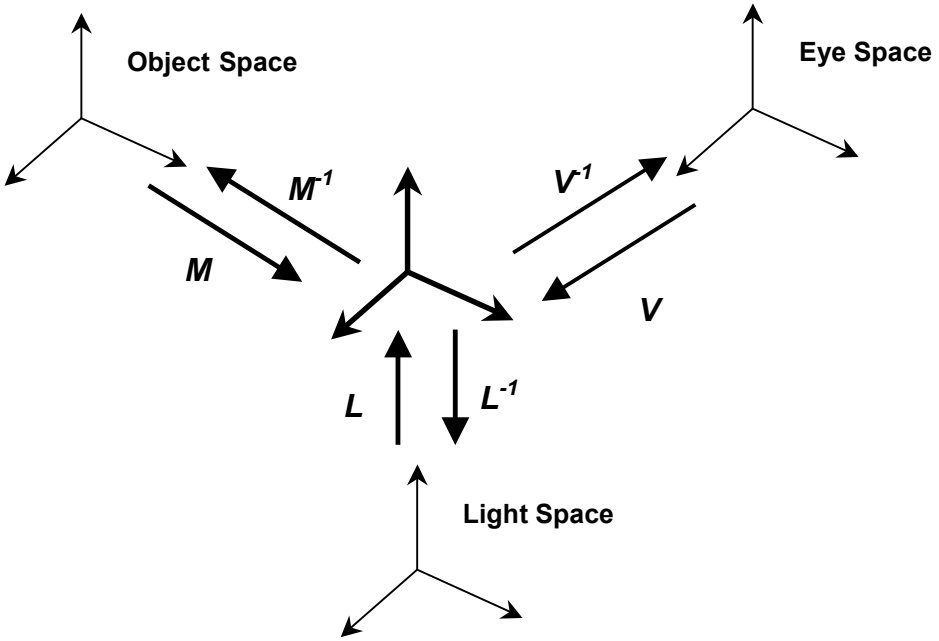


Figure 7: Schematic view of the basic transformations involved in shadow mapping.

Figure 7 shows the three primary transformations (and inverses) used in shadow mapping. Note that we use the convention of using the *forward* transforms as going *to* world coordinates. The standard 'modelview' matrix using the above notation will therefore be: $\boldsymbol{V^{1}M}$. In addition to the above transformations, we also have to account for the projections involved in the two passes – these could be different depending on the frusta for the light and eye. The projection transformation will also be applied during the texture coordinate generation phase which is depicted in Figure 8 for OpenGL. As shown

in the figure, two transformations are applied to the eye coordinates – the *texgen* planes, and the *texture matrix*. For *eye linear* texgen planes, OpenGL will automatically multiply the eye coordinates with the inverse of the modelview matrix *in effect when the planes are specified*. (See Appendix A for a more detailed explanation of the texgen planes in the eye linear case.)

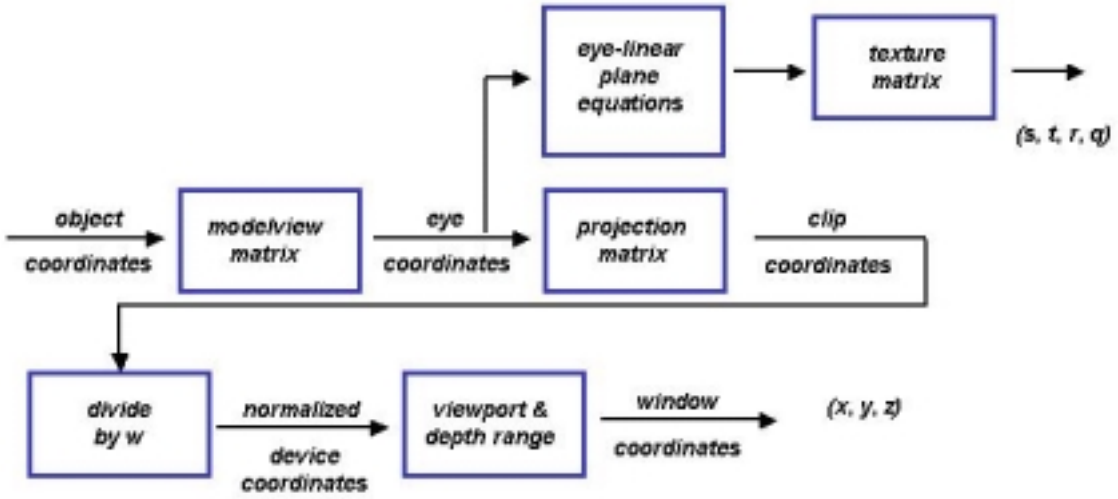


Figure 8: OpenGL Transformation Pipeline

The resulting texture coordinates are therefore computed as:

$$[x_e,y_e,z_e,w_e]^T = (\text{modelview})\ [x_o,y_o,z_o,w_o]^T$$

$$E_e = E_{po}(\text{modelview}_{po})^{-1}$$

$$[s,t,r,q]^T = T\ E_e\ [x_e,y_e,z_e,w_e]^T$$

Equation 1

Here the subscript '**o**' denotes object space coordinates, and the subscript '**e**' refers to eye space coordinates, $\textbf{modelview}_{po}$ is the modelview matrix in effect when the eye linear texgen plane equations are specified, E_{po} is the matrix composed of the eye linear plane equations *as specified to OpenGL* (i.e. in their own object space), E_e is the matrix composed of the transformed plane equations (these are the plane equations that are

actually stored by OpenGL), **T** is the texture matrix, and **modelview** is the modelview matrix when rendering the scene geometry.

Setting Up the Transformations

We want to set the transformations in Equation 1 to compute texture coordinates **(s,t,r,q)** such that **(s/q,t/q)** will be the fragment's location within the depth texture, and **r/q** will be the window-space z of the fragment relative to the light's frustum. In other words, we want to compute:

$$[s,t,r,q]^T = S\ P_{light}\ L^{-1}\ M\ [x_o,y_o,z_o,w_o]^T$$

Equation 2

Here, **S** is the scale-bias matrix, given by:

$$\begin{bmatrix} ½ & 0 & 0 & ½ \\ 0 & ½ & 0 & ½ \\ 0 & 0 & ½ & ½ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{P_{light}}$ is the projection matrix for the light frustum. The "texgen matrix" ($\mathbf{E_e}$), however, is applied to *eye* coordinates $[x_e,y_e,z_e,w_e]^T$ but we want to generate the coordinates in *light* space, since that is where the depth map computation takes place. So we need to take $[x_e,y_e,z_e,w_e]^T$ back into world space by applying the transform **V**. That is, we want to compute $[s,t,r,q]^T$ as:

$$[s,t,r,q]^T = S\ P_{light}\ L^{-1}\ V\ [x_e,y_e,z_e,w_e]^T$$

Equation 3

Note that the right hand side of Equation 3 reduces to $S\ P_{light}\ L^{-1}\ M\ [x_o,y_o,z_o,w_o]$, precisely what we want. A straightforward way to compute Equation 3 is to set $\mathbf{modelview_{po}}$ to *identity* and set:

$$T\ E_{po} = S\ P_{light}\ L^{-1}\ V$$

Equation 4

The first observation is that we have two matrices **T** (the texture matrix) and $\mathbf{E_{po}}$ (the eye linear texgen planes specified to OpenGL) so we can compute Equation 4 in several

ways. Since we are going to have to set the eye linear planes in any case, the less expensive thing to do is to not set the texture matrix at all, and use the texgen matrix **G** for the entire computation[†], i.e., set

$$\mathbf{E}_{po} = \mathbf{S}\,\mathbf{P}_{light}\,\mathbf{L}^{-1}\,\mathbf{V}$$

Equation 5

This assumes that the modelview matrix, $\mathbf{modelview}_{po}$, was identity at the time the texgen planes are set. Another improvement is to make use of the fact that OpenGL automatically multiplies $[x_e,y_e,z_e,w_e]^T$ with $(\mathbf{modelview}_{po})^{-1}$ for eye linear texgen. The sole purpose of using **V** in Equation 5 is to eliminate $\mathbf{V}^{-1}$. If we set $\mathbf{modelview}_{po} = \mathbf{V}^{-1}$, then OpenGL will do the elimination for us and we can avoid having to compute **V**, the inverse of the view matrix. The steps can be summarized as follows:

First Pass (Depth Map Generation)

- Render from light's point of view. Set projection matrix to $\mathbf{P}_{light}$. Set the view portion of the modelview matrix to $\mathbf{L}^{-1}$.
- Render scene (with appropriate modeling transform(s) **M**).

Second Pass (Depth Map Comparison)

- Render from eye's point of view. Set projection matrix to be $\mathbf{P}_{eye}$. Set the view portion of the modelview matrix to be $\mathbf{V}^{-1}$.
- Set texgen to be EYE_LINEAR. Specify texgen planes as $\mathbf{E}_{po} = \mathbf{S}\,\mathbf{P}_{light}\,\mathbf{L}^{-1}$
- Render scene (with appropriate modeling transform(s) **M**)

Conclusions

Shadow mapping is an easy-to-use shadowing technique that makes 3D rendering just look better. It enjoys hardware acceleration on GeForce3 GPUs. There is example source code in the NVSDK (hw_shadowmaps_simple, hw_woo_shadowmaps) that demonstrate the technique, and the corresponding OpenGL extensions. Please direct questions or comments to cass@nvidia.com.

References

[1] Craig Duttweiler. Mapping Texels to Pixels in Direct3D. http://developer.nvidia.com/view.asp?IO=Mapping_texels_Pixels.

[†] Note that this technique of collapsing the texture and texgen matrices works in our case because we are setting all four planes, and using the same mode for all four planes. In general, each coordinate can have a different mode (eye linear, object linear, sphere map…) and the technique may not be applicable.

[2] Cass Everitt. Interactive Order-Independent Transparency. Whitepaper: http://developer.nvidia.com/view.asp?IO=Interactive_Order_Transparency.

[3] Mark Kilgard. Shadow Mapping with Today's Hardware. Technical presentation: http://developer.nvidia.com/view.asp?IO=cedec_shadowmap.

[4] Mark Segal and Kurt Akeley. The OpenGL Graphics System: A Specification (Version 1.2.1). www.opengl.org

[5] Mark Segal, et al. Fast shadows and lighting effects using texture mapping. In *Proceedings of SIGGRAPH '92*, pages 249-252, 1992.

[6] OpenGL Extension Registry. http://oss.sgi.com/projects/ogl-sample/registry/.

[7] Yulan Wang and Steven Molnar. Second-Depth Shadow Mapping. UNC-CS Technical Report TR94-019, 1994.

[8] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH '78*, pages 270-274, 1978.

[9] Andrew Woo, P. Poulin, and A. Fournier. "A Survey of Shadow Algorithms," IEEE Computer Graphics and Applications: vol 10(6), pages 13-32, 1990.

Appendix A: Another Way to Think about EYE_LINEAR planes in OpenGL

An unfortunate thing about EYE_LINEAR texgen in OpenGL is that the name implies that the plane equations are specified in eye space, when they are, in fact, specified in their own object space. There are two ways one can think about the planes specified in EYE_LINEAR texgen. As mentioned earlier, OpenGL will automatically multiply the planes specified with $(\mathbf{modelview}_{\mathbf{po}})^{-1}$, i.e. the inverse of the modelview matrix in effect when the planes are specified. From Equation 1 we see that the net effect is to map the vertex position in eye coordinates $[x_e,y_e,z_e,w_e]^T$ back to the 'object space' defined by $(\mathbf{modelview}_{\mathbf{po}})^{-1}$. The transformed coordinates are then evaluated at each plane in this object space to get the texture coordinates. An alternate way to think about the texgen planes is to consider the matrix $\mathbf{E}_e = \mathbf{E}_{po}(\mathbf{modelview})^{-1}$, which defines a map whose domain is *eye* space, with the planes $\mathbf{E}_{po}$ being specified in object space. $\mathbf{E}_e$ therefore defines the *transformed* planes in eye space. In either case, the planes are being *specified* in the 'object space' defined by $(\mathbf{modelview}_{\mathbf{po}})^{-1}$ and *not* in eye space.

In the shadow mapping case described earlier, the modelview matrix is set to $\mathbf{V}^{-1}$ when the texgen plane equations are specified. This is the same thing as saying that we are specifying the plane equations in *world* space. If the modelview matrix were set to identity, then we would be specifying the equations in *eye* space. The same is true if we were specifying vertex positions.

We could set the modelview matrix to $\mathbf{V}^{-1}\mathbf{L}$, and specify the plane equations in *light* space. This might be handy, because we would only need to update our plane equations if the light's projection ($\mathbf{P}_{\mathbf{light}}$) changed. We could even put the *whole* transformation into the modelview matrix as $\mathbf{V}^{-1}\mathbf{L}\mathbf{P}_{\mathbf{light}}^{-1}\mathbf{S}^{-1}$. In this case, the texgen planes are *always* just specified as identity ($\mathbf{E}_{\mathbf{po}} = \mathbf{I}$)!

Annex: argument

Com ja s'ha dit, l'argument es basa en un viatge al passat provocat per un antic ritual romà, el *devotio*. El personatge haurà de trobar la manera de tornar al present i escapar dels déus romans. En tota aquesta aventura succeiran nombroses missions i proves que l'usuari haurà de superar. A continuació tot l'argument detallat:

- Vídeo inicial: Uns nois entren al pretori i es queden fins que es fa fosc. Realitzen el ritual romà del *devotio* i un d'ells és transportat al passat.

- Arribada: El noi arriba al pretori i ha de sortir d'aquest sense que cap soldat romà el vegi (aprofitant quan estiguin d'esquena o passin per un passadís fosc).

- Arribar fins al punt de guardar partida. Aquí s'explicarà a l'usuari com guardar una partida i carregar-la posteriorment.

- En aquest punt es fa de dia i el noi ha d'aconseguir roba adequada a l'època per a no cridar l'atenció. També ha d'aconseguir informació sobre *augurs* i savis que el puguin ajudar a tornar a casa.

- Descobrirà que la persona més sàvia és el conseller i ajudant privat del governador de Tàrraco, un home vell i malalt. Per arribar al fòrum i al recinte de culte (espai reservat per a persones d'alta categoria) s'haurà de guanyar la confiança d'un soldat romà. I per a fer-ho haurà de completar les següents missions que li demana el romà.

  - Aconseguir (robar) un contracte de compra d'unes terres que va vendre el soldat i que vol recuperar per la força. El noi s'haurà de fer passar per venedor i entrar a la casa de la víctima per robar-li.

  - Entrar al pretori de nit i amagar-se per a sentir una conversa entre un noble que vol trair a un altre noble i un soldat que està disposat a ajudar-lo.

  - En aquest punt el soldat es posa molt content per la informació que el noi ha aconseguit i li diu que l'acompanyarà fins al fòrum provincial; però s'adona que ha perdut les claus de la porta i li demana al noi que l'ajudi a buscar-les.

- A continuació li demana un nou favor. Entrar a casa d'un funcionari i falsificar uns documents que demostren que el soldat és fill d'esclaus. En aquesta missió el noi descobreix per casualitat que és família d'un noble romà. També recorda, de la conversa escoltada, que aquest noble està en perill de mort.

- Aquesta vegada, quan el noi es troba de nou amb el soldat, aquest li posa molts problemes per a entrar al fòrum. Li suggereix que vagi al *castrum* (caserna de l'exèrcit) i s'allisti per a obtenir el grau de soldat. El noi fa cas però les proves d'accés a l'exèrcit són massa dures i no és acceptat.

- Davant el problema de no poder entrar al fòrum el noi decideix portar al soldat a una taverna i emborratxar-lo. En primer lloc ha de recollir diners per pagar la beguda i després portar al soldat a la taverna.

- Un cop el soldat està borratxo li pren la roba i s'escapoleix dins del fòrum.

- A partir d'aquest punt el noi té accés al fòrum. Descobrirà que la persona més sàvia de tot Tàrraco és el conseller i ajudant privat del governador, un home vell i malalt. A partir d'aquí es divideix l'argument en dues parts:

Per una banda el noi ha de trobar la manera de salvar el seu avantpassat (per evitar la seva desaparició en el present).

- En aquesta missió el noi a d'obtenir informació de com intentaran assassinar al seu avantpassat. Un cop ho ha fet ha de robar les armes que s'utilitzaran en el crim així com escapar d'una persecució.

- En un segon intent d'assassinar al seu avantpassat el noi li explica que el volen matar i el noi i ell hauran de fugir i amagar-se dels assassins durant un període de temps.

Per altra banda ha de posar-se en contacte amb el conseller del governador per a trobar la forma de tornar a casa.

- En un primer intent d'entrar en contacte amb aquest home, el noi haurà de trobar quin és el seu criat i parlar amb ell. Aquest li demanarà diners a canvi, que haurà d'aconseguir. Un cop pot parlar amb el savi aquest no li aclareix res, però el noi s'adona que la casa està plena de llibres màgics.

- Decideix entrar per la força i roba els llibres del savi amb l'intenció de realitzar el ritual pel seu compte, però s'adona que necessita uns objectes màgics per a realitzar-lo.

- Es troben els dos fils anteriors. El noi i el seu avantpassat denunciaran els fets a la justícia i els assassins seran detinguts. Com a mostra d'agraïment el noi rebrà uns objectes màgics (els necessaris per fer el ritual per tornar a casa) i la possibilitat de viure a casa del seu avantpassat (nou punt de guardat).

- El noi realitzarà el ritual per a tornar a casa, però no ho aconseguirà. Es mourà uns anys en el temps fins a la decadència de Tàrraco.

- En arribar al període de decadència de Tàrraco el noi s'assabenta que la ciutat està en perill. Tribus de bàrbars intenten atacar-la. El conseller del governador ha mort, i no hi ha gaire gent que el pugui ajudar a tornar a casa.

També sent que el governador enviarà un missatger a Roma per a sol·licitar reforços contra els bàrbars. Però ell sap que per a que la història continuï de forma correcta aquests reforços no han d'arribar. És per això que aquí l'argument es divideix de nou.

Un dels seus objectius és, altra vegada, tornar a casa.

- Parlant amb la gent sent que existeix un objecte màgic que va pertànyer a un emperador de Roma i al qual se li atribueixen nombrosos poders. Decideix provar sort i anar fins al recinte de culte per a robar-lo.

- El problema que té a continuació és que no sap com s'utilitza aquest objecte. Per a intentar trobar la manera d'utilitzar-lo entra a les diferents cases i oficines de funcionaris buscant algun llibre on n'expliqui el seu funcionament.

L'altre objectiu és trobar una manera de fer que la història no canviï.

- En un primer intent d'evitar una petició de reforços a Roma intenta subornar un funcionari, però no ho aconsegueix. Aquest el tracta de traïdor a Tàrraco i envia uns soldats a la seva persecució. El noi es veu obligat a amagar-se en una casa mol humil que l'acull (nou punt de guardat).

- Aquest cop intenta per la força cremar l'oficina encarregada de la missatgeria entre les ciutats romanes.

- Però de nou falla i el seu últim recurs és localitzar el missatger i canviar-li el missatge. Aquesta és una missió arriscada on ha d'evitar que algú el vegi canviar el missatge, així com trobar algú que li falsifiqui.

- Un cop complerts els objectius anteriors està llest per marxar a casa. El ritual funciona aquest cop i pot marxar de casa salvant la vida.

- Finalment en l'últim vídeo es veu com el noi torna al present i intenta convèncer als seus amics del que ha viscut, però ells no el creuen. Finalment s'adona que l'objecte màgic amb el que han fet el ritual del *devotio* ha desaparegut, cosa que li dóna la raó. Els seus amics vacil·len i ell, que no ha escarmentat, els proposa un nou viatge al passat tots junts. Aquí es dóna peu a una nova part del joc.

Annex: Disseny

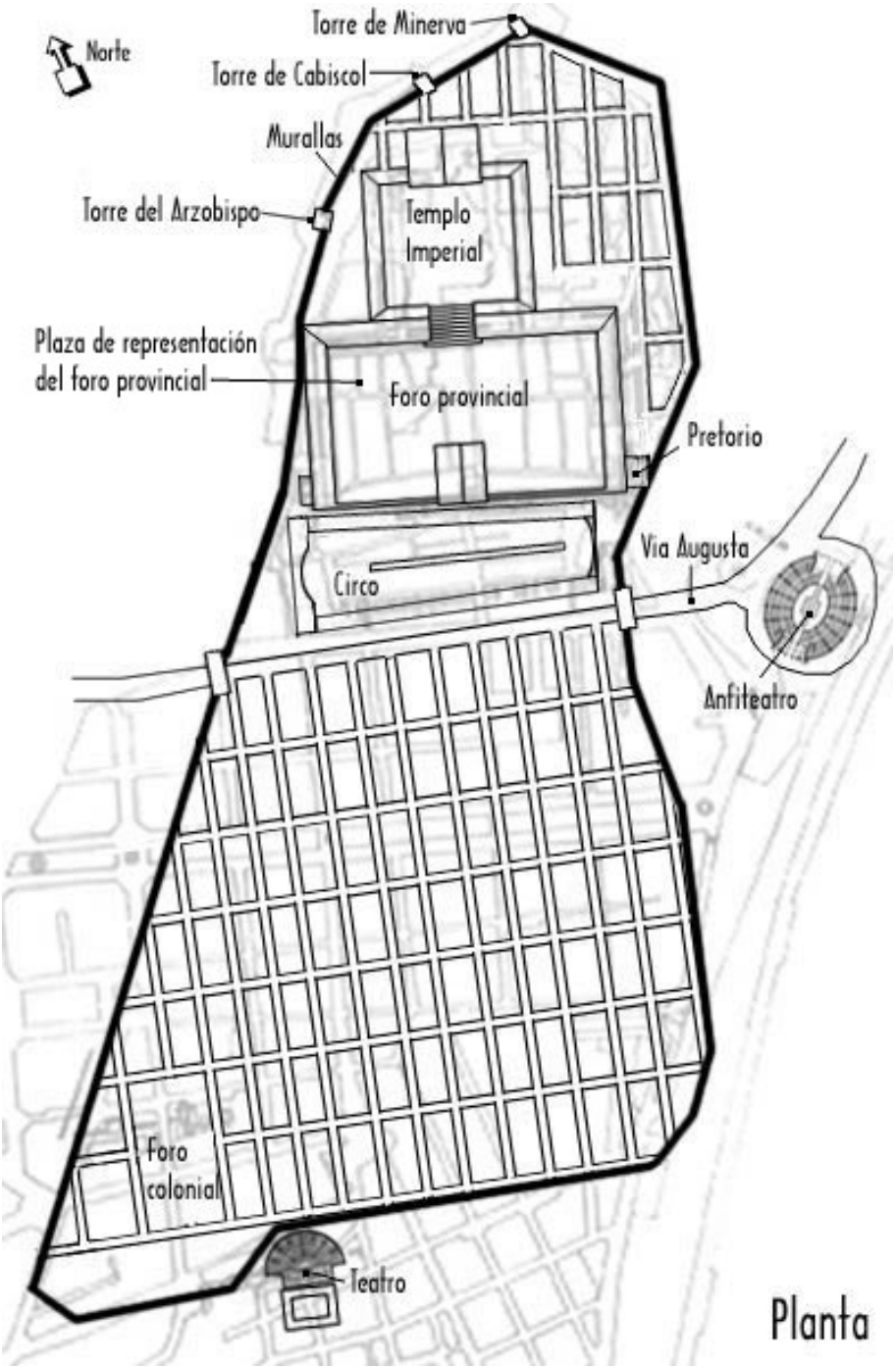


Figura C.A *Pla general de la ciutat de Tàrraco el S II d.C.*

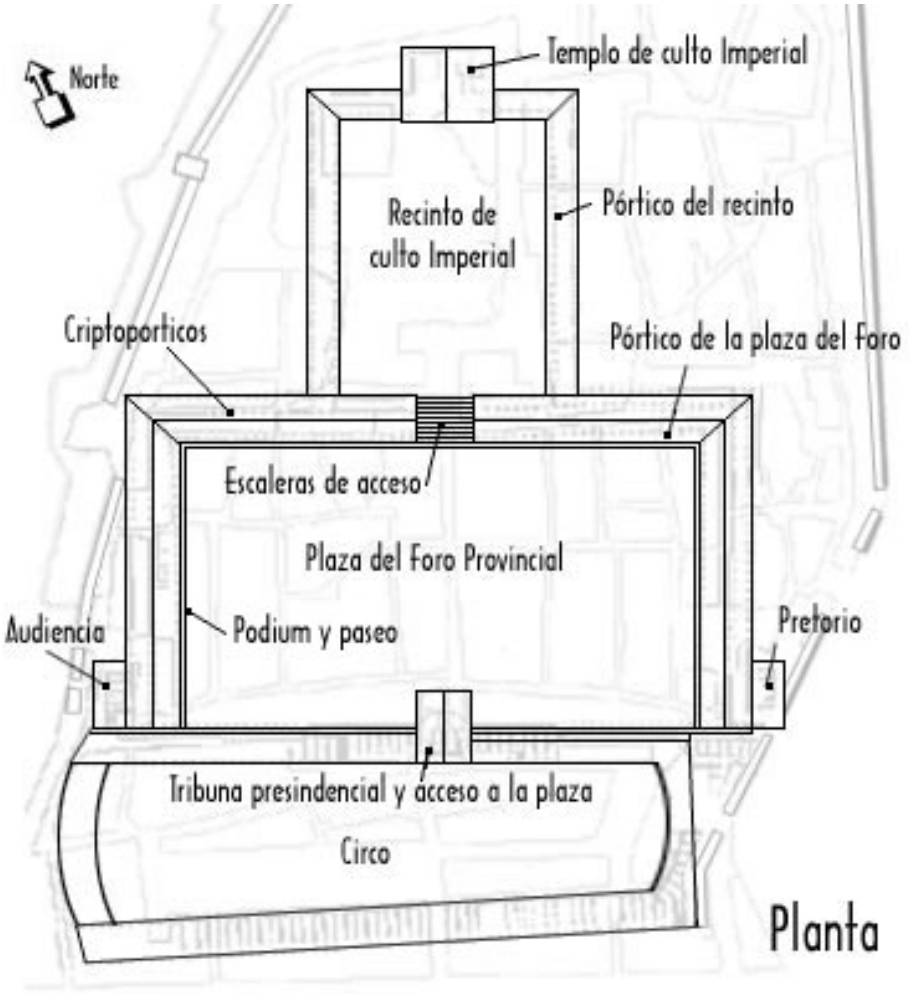


Figura C.B *Pla general de la part superior de Tàrraco el S II d.C.*

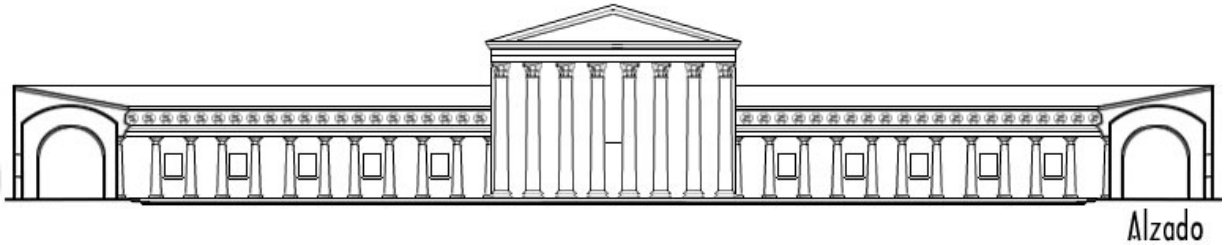


Figura C.C *Alçat del recinte de culte*

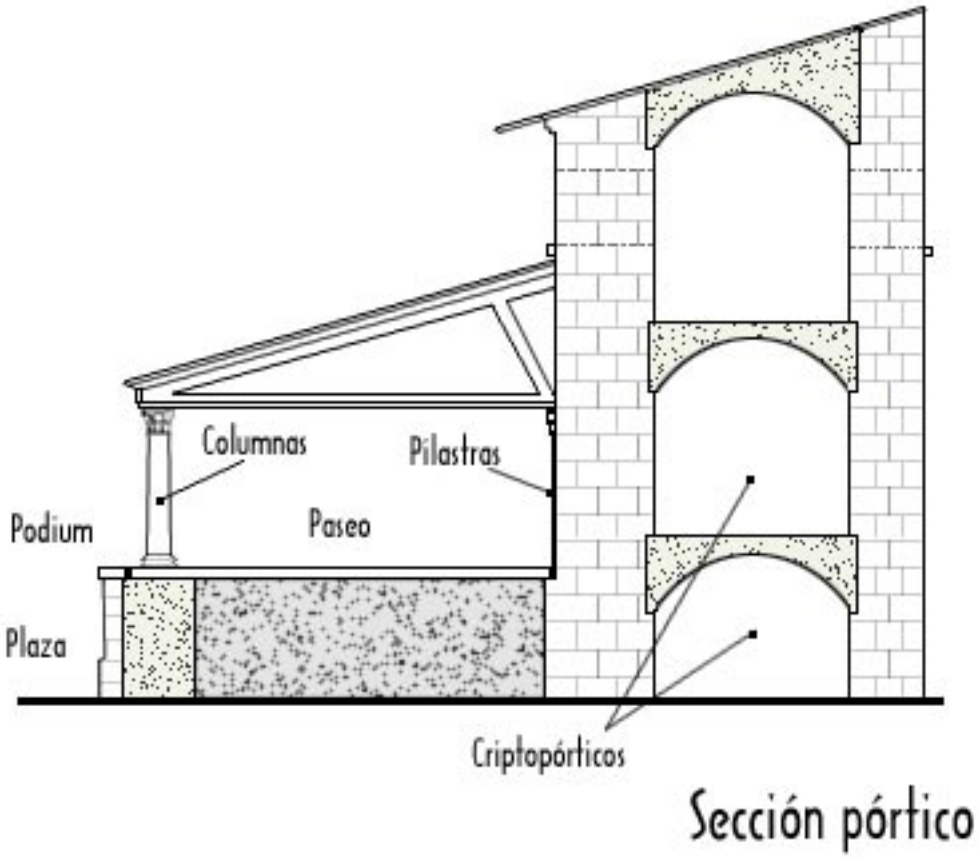


Figura C.D *Secció del porticat del fòrum*

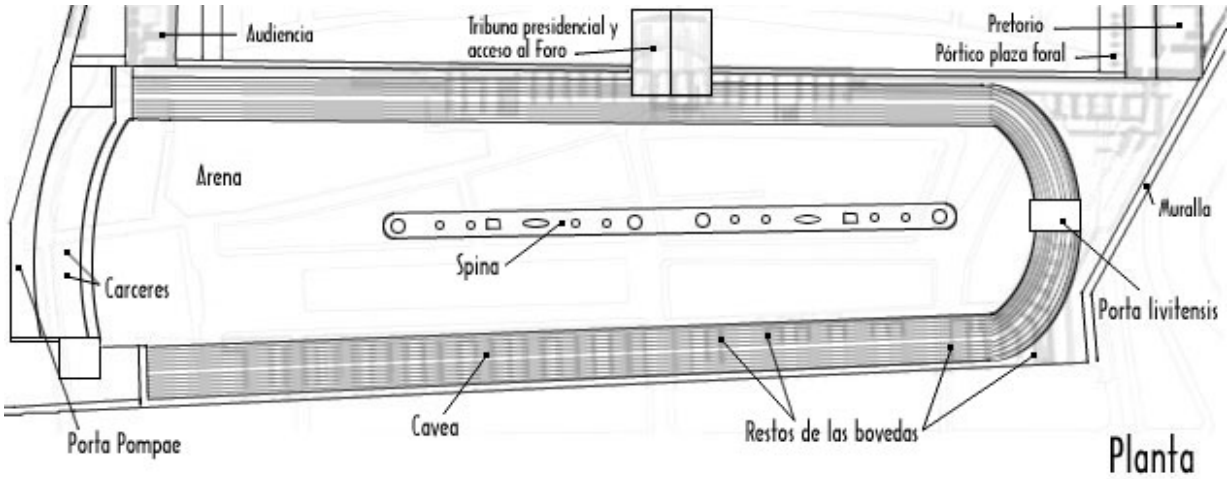


Figura C.E *Vista aèria del circ*

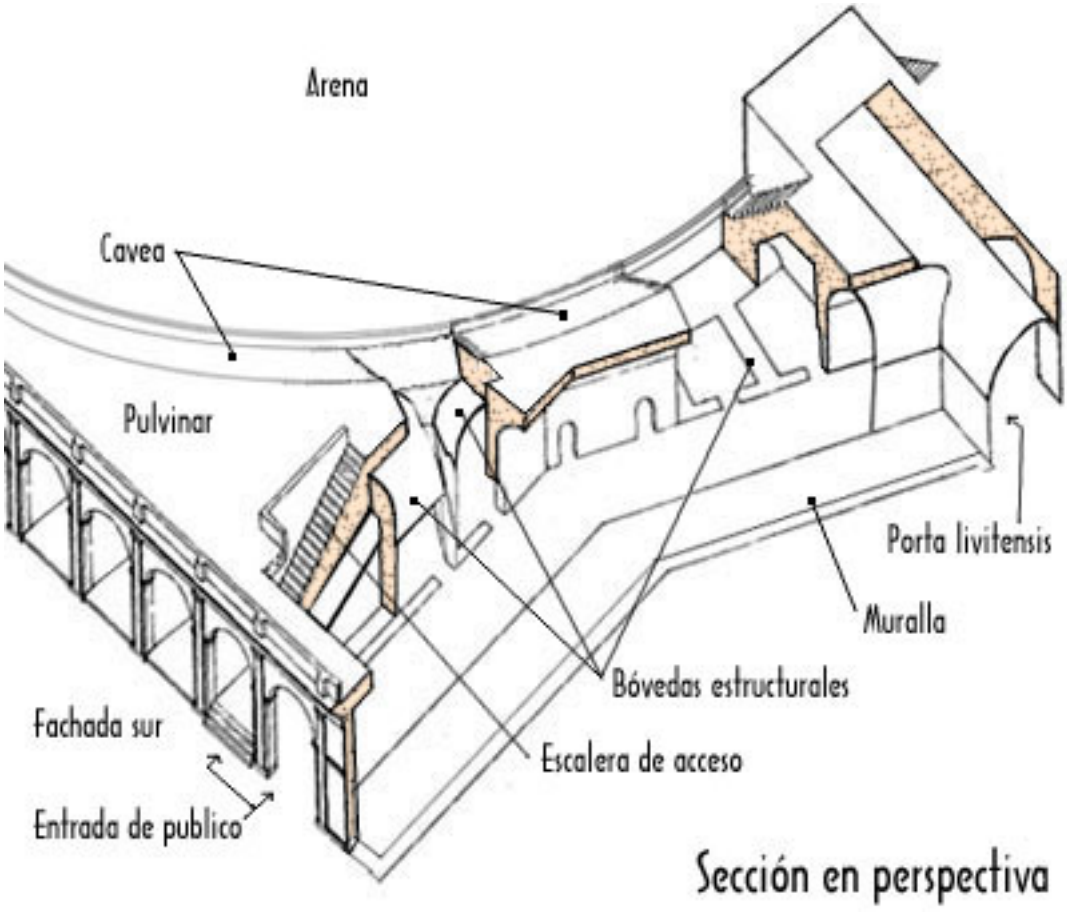


Figura C.F *Secció de la volta del circ*

Aquests plànols han permès la reconstrucció de la Tàrraco romana. Extrets de www.spanisharts.com, on es pot trobar tot tipus de plànols de ciutats romanes

Annex: Codi

Codi font del joc. Per una banda el codi C++ i per l'altra el Visual Basic. No s'ha inclòs tot el codi, aquí només trobareu el codi creat pel treball. Altres llibreries en ús estan disponibles a l'annex F.

frmGraphics - 1

```
Option Explicit

Implements DirectXEvent8

Private Sub DirectXEvent8_DXCallback(ByVal eventid As Long)

If Not (eventid = DI_hevent) Then Exit Sub

    Dim DevData(1 To 50) As DIDEVICEOBJECTDATA
    Dim nEvents As Long
    Dim i As Long

    On Local Error Resume Next    'handle errors???
    nEvents = MouseDevice.GetDeviceData(DevData, DIGDD_DEFAULT)

    For i = 1 To nEvents
        Select Case DevData(i).lOfs
            Case DIMOFS_X
                MouseX = MouseX + DevData(i).lData
            Case DIMOFS_Y
                MouseY = MouseY + DevData(i).lData
            Case DIMOFS_Z
                MouseZ = MouseZ + DevData(i).lData
            Case DIMOFS_BUTTON0
                If DevData(i).lData = 0 Then
                    If MouseB0 = True Then MouseClick0 = True
                    MouseB0 = False
                Else
                    MouseB0 = True
                End If
            Case DIMOFS_BUTTON1
                If DevData(i).lData = 0 Then
                    If MouseB1 = True Then MouseClick1 = True
                    MouseB1 = False
                    If MainRender Then Movement = 0
                Else
                    MouseB1 = True
                    If MainRender Then Movement = 1
                End If
        End Select
    Next

DoEvents
End Sub

Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
If MainRender Then          'we are in the game loop
    If Shift = 0 Then       'NO shifts keys
        If KeyCode = vbKeySpace And Jumping = False Then Jumping = True
        If KeyCode = vbKeyW Or KeyCode = vbKeyUp Then Movement = 1
        If KeyCode = vbKeyEscape Then AuxMenu = True
    End If
End If
End Sub

Private Sub Form_KeyUp(KeyCode As Integer, Shift As Integer)
If MainRender Then          'we are in the game loop
    If Shift = 0 Then       'NO shifts keys
        If KeyCode = vbKeyW Or KeyCode = vbKeyUp Then Movement = 0
    End If
```

```
End If
End Sub
```

AuxFunctions - 1

```
Option Explicit

'-------------------------------------------------------------------------
--------------
'---------------------------------- Register Functions -------------------
--------------
'-------------------------------------------------------------------------
--------------

Public Function RegRead(ByVal ValueName As String) As String
RegRead = Reg.GetRegString("HKLM\Software\" & RegName & "\", ValueName)
End Function

Public Sub RegSave(ByVal ValueName As String, ByVal Value As String)
Reg.SetReg "HKLM\Software\" & RegName & "\", ValueName, Value
End Sub

Public Function ParseDM(ByVal DM As String) As D3DDISPLAYMODE
ParseDM.width = Val(left(DM, 4))
ParseDM.height = Val(mid(DM, 5, 4))
ParseDM.RefreshRate = Val(mid(DM, 9, 4))
ParseDM.Format = Val(right(DM, 4))

Dim x As Long, Ok As Boolean
For x = 0 To Direct3D.GetAdapterModeCount(0) - 1
    If ParseDM.RefreshRate <> 0 Then           'refresh rate defined, chek
if it is valid
        If DModesArray(x).width = ParseDM.width And _
            DModesArray(x).height = ParseDM.height And _
            DModesArray(x).RefreshRate = ParseDM.RefreshRate And _
            DModesArray(x).Format = ParseDM.Format Then
            Ok = True
            Exit For
        End If
    Else         'refresh rate = 0, auto rr, no check
        If DModesArray(x).width = ParseDM.width And _
            DModesArray(x).height = ParseDM.height And _
            DModesArray(x).Format = ParseDM.Format Then
            Ok = True
            Exit For
        End If
    End If
Next

If Ok = False Then
    Direct3D.GetAdapterDisplayMode 0, ParseDM
End If
End Function

'-------------------------------------------------------------------------
--------------
'---------------------------------- Maths Functions ----------------------
--------------
'-------------------------------------------------------------------------
--------------

Public Function VecDistFast(v1 As D3DVECTOR, v2 As D3DVECTOR) As Double
VecDistFast = (v1.x - v2.x) ^ 2 + (v1.y - v2.y) ^ 2 + (v1.z - v2.z) ^ 2
End Function
```

```
AuxFunctions - 2

Public Function CalculateDistance(v1 As D3DVECTOR, v2 As D3DVECTOR) As Sing
le
CalculateDistance = Sqr((v1.x - v2.x) ^ 2 + (v1.y - v2.y) ^ 2 + (v1.z - v2.
z) ^ 2)
End Function

Public Function v3(ByVal x As Single, ByVal y As Single, ByVal z As Single)
 As D3DVECTOR
v3.x = x
v3.y = y
v3.z = z
End Function

Public Function Normalize(v As D3DVECTOR) As D3DVECTOR
On Local Error Resume Next
Dim module As Double
module = Sqr(v.x ^ 2 + v.y ^ 2 + v.z ^ 2)
If module = 0 Then Exit Function
Normalize.x = v.x / module
Normalize.y = v.y / module
Normalize.z = v.z / module
End Function

'----- Color function ------

Public Function ColorValue(ByVal r As Single, ByVal g As Single, ByVal b As
 Single, ByVal a As Single) As D3DCOLORVALUE
ColorValue.r = r
ColorValue.g = g
ColorValue.b = b
ColorValue.a = a
End Function

'-------------------------------------------------------------------------
--------------
'---------------------------------- Unpacking Functions ------------------
--------------
'-------------------------------------------------------------------------
--------------

Public Function ExtractFile(ByVal file As String, ByVal Dir As String) As B
oolean
'On Local Error GoTo Out
If FileExists(file) = False Then Exit Function
Dim filepath As String
filepath = Dir
If right(filepath, 1) <> "\" Then filepath = filepath & "\"
Dim ff As Integer, ff2 As Integer, x As Long
Dim cad As String, ln As Long, Max As Long, num As Long, cad2 As String
ff = FreeFile
Open file For Binary As #ff
    cad = Space(14)
    Get #ff, , cad
    Get #ff, , num

    For x = 1 To num
        Get #ff, , ln
        cad = Space(ln)
        Get #ff, , cad
        cad = Code(cad)
```

```
        ff2 = FreeFile
        Open filepath & cad For Binary As #ff2
            Get #ff, , ln
            If ln <= 1127 Then
                cad = Space(ln)
                Get #ff, , cad
                cad = Code(cad)
                Put #ff2, , cad
            Else
                cad2 = Space(ln - 1127)
                Get #ff, , cad2
                cad = Space(1127)
                Get #ff, , cad
                cad = Code(cad)
                Put #ff2, , cad
                Put #ff2, , cad2
            End If
        Close #ff2
    Next
Close #ff
ExtractFile = True
Out:
End Function

Public Function Code(var As String) As String
Dim n As Integer
Code = ""
For n = 1 To Len(var)
    Code = Code & Chr(255 - Asc(mid$(var, n, 1)))
Next
End Function

Public Function DecodeFile(ByVal file As String, ByVal file2 As String) As
Boolean
On Local Error Resume Next
Kill file2
On Local Error GoTo Out:
Dim ff As Integer, ff2 As Integer
Dim tam As Long, x As Long
Dim by As Byte, by2 As Long
ff = FreeFile
Open file For Binary As #ff
ff2 = FreeFile
Open file2 For Binary As #ff2
tam = FileLen(file)
For x = 1 To tam
    If (x Mod 2) = 1 Then
        Get #ff, , by
        by = 255 - by
        Put #ff2, , by
    Else
        Get #ff, , by
        by2 = by
        by2 = by2 - 121
        If by2 < 0 Then by2 = by2 + 256
        by = by2
        Put #ff2, , by
    End If
Next
Close #ff
```

```
Close #ff2
DecodeFile = True
Out:
End Function


'--------------------------------------------------------------------------
---------------
'---------------------------------- Frustum Culling ----------------------
---------------
'--------------------------------------------------------------------------
---------------

Public Sub ComputeClipPlanes()
    Dim vecsf(7) As D3DVECTOR, mat As D3DMATRIX, x As Integer

    D3DXMatrixMultiply mat, viewMatrix, projMatrix
    D3DXMatrixInverse mat, 0, mat

    vecsf(0) = v3(-1, -1, 0)
    vecsf(1) = v3(1, -1, 0)
    vecsf(2) = v3(-1, 1, 0)
    vecsf(3) = v3(1, 1, 0)
    vecsf(4) = v3(-1, -1, 1)
    vecsf(5) = v3(1, -1, 1)
    vecsf(6) = v3(-1, 1, 1)
    vecsf(7) = v3(1, 1, 1)

    For x = 0 To 7
        D3DXVec3TransformCoord vecsf(x), vecsf(x), mat
    Next

    D3DXPlaneFromPoints FrustumPlanes(0), vecsf(0), vecsf(1), vecsf(2)
    D3DXPlaneFromPoints FrustumPlanes(1), vecsf(6), vecsf(7), vecsf(5)
    D3DXPlaneFromPoints FrustumPlanes(2), vecsf(2), vecsf(6), vecsf(4)
    D3DXPlaneFromPoints FrustumPlanes(3), vecsf(7), vecsf(3), vecsf(5)
    D3DXPlaneFromPoints FrustumPlanes(4), vecsf(2), vecsf(3), vecsf(6)
    D3DXPlaneFromPoints FrustumPlanes(5), vecsf(1), vecsf(0), vecsf(4)
End Sub


'--------------------------------------------------------------------------
----------------------
'---------------------------------- FIXED OBJECTS FUNCTIONS -------------
----------------------
'--------------------------------------------------------------------------
----------------------

Public Sub DrawFixedObjects()
Dim x As Integer, y As Long
For x = 1 To UBound(FixedObjects)
    If FixedObjects(x).WorldID = TheGameSlot.WorldID Then
        If Abs(CharPos.x - FixedObjects(x).Position.x) < DistanceFOAppear A
nd _
           Abs(CharPos.z - FixedObjects(x).Position.z) < DistanceFOAppear T
hen
            If CheckSphere(FixedObjects(x)) = True Then
                D3DXMatrixTranslation TMatrix, FixedObjects(x).Position.x,
FixedObjects(x).Position.y, FixedObjects(x).Position.z
                Device.SetTransform D3DTS_WORLD, TMatrix
```

```
                If Abs(CharPos.x - FixedObjects(x).Position.x) < DistanceFO
Detail And _
                    Abs(CharPos.z - FixedObjects(x).Position.z) < DistanceF
ODetail Then
                    RenderModel FOComplex(FixedObjects(x).MeshID)
                Else
                    If FixedObjects(x).unimesh = 1 Then
                        RenderModel FOComplex(FixedObjects(x).MeshID)
                    Else
                        RenderModel FOSimple(FixedObjects(x).MeshID)
                    End If
                End If
            End If
        End If
    End If
Next
End Sub

Public Sub DrawFixedObjectsStage()
Dim x As Integer, y As Long
For x = 1 To UBound(FixedObjectsStage)
    If FixedObjectsStage(x).WorldID = TheGameSlot.WorldID Then
        If Abs(CharPos.x - FixedObjectsStage(x).Position.x) < DistanceFOApp
ear And _
           Abs(CharPos.z - FixedObjectsStage(x).Position.z) < DistanceFOApp
ear Then
            If CheckSphere(FixedObjectsStage(x)) = True Then
                D3DXMatrixTranslation TMatrix, FixedObjectsStage(x).Positio
n.x, FixedObjectsStage(x).Position.y, FixedObjectsStage(x).Position.z
                Device.SetTransform D3DTS_WORLD, TMatrix
                If Abs(CharPos.x - FixedObjectsStage(x).Position.x) < Dista
nceFODetail And _
                    Abs(CharPos.z - FixedObjectsStage(x).Position.z) < Dist
anceFODetail Then
                    RenderModel FOComplexStage(FixedObjectsStage(x).MeshID)
                Else
                    If FixedObjectsStage(x).unimesh = 1 Then
                        RenderModel FOComplexStage(FixedObjectsStage(x).Mes
hID)
                    Else
                        RenderModel FOSimpleStage(FixedObjectsStage(x).Mesh
ID)
                    End If
                End If
            End If
        End If
    End If
Next
End Sub

Public Function CheckSphere(object As FixedObject) As Boolean
Dim i As Long

For i = 0 To 5
    If D3DXPlaneDotCoord(FrustumPlanes(i), object.transformedBoundingSphere
.center) < -object.transformedBoundingSphere.radius Then
        CheckSphere = False
        Exit Function
    End If
Next
```

```
CheckSphere = True
End Function

Public Function LoadFixedObject(Position As D3DVECTOR, ByVal meshfile As St
ring, ByVal meshfilesimple As String, AutoY As Boolean, RotationY As Single
, WorldID As Integer) As FixedObject
Dim res As salida, x As Long, id As Long, y As Long
LoadFixedObject.Position = Position

For x = 1 To UBound(FOComplex)
    If FOComplex(x).MeshName = UCase(GetFileName(meshfile)) And _
       FOSimple(x).MeshName = UCase(GetFileName(meshfilesimple)) Then
        id = x
        GoTo SkipLoadingFO
    End If
Next

ReDim Preserve FOComplex(UBound(FOComplex) + 1)
ReDim Preserve FOSimple(UBound(FOSimple) + 1)
id = UBound(FOSimple)

Set FOComplex(id).Mesh = Direct3DX.LoadMeshFromX(meshfile, D3DXMESH_MANAGED
, Device, FOComplex(id).tmp_adj, FOComplex(id).tmp_mat, FOComplex(id).NumMa
terials)
ReDim FOComplex(id).Materials(FOComplex(id).NumMaterials - 1)
ReDim FOComplex(id).TexturesNames(FOComplex(id).NumMaterials - 1)
OptimizeMesh FOComplex(id).Mesh, FOComplex(id).tmp_adj
Set FOComplex(id).tmp_adj = Nothing

For y = 0 To FOComplex(id).NumMaterials - 1
    Direct3DX.BufferGetMaterial FOComplex(id).tmp_mat, y, FOComplex(id).Mat
erials(y)
    FOComplex(id).Materials(y).Ambient = FOComplex(id).Materials(y).diffuse
     'aspect patch!
    FOComplex(id).TexturesNames(y) = UCase(GetFileName(Direct3DX.BufferGetT
extureName(FOComplex(id).tmp_mat, y)))
    If FOComplex(id).TexturesNames(y) <> "" Then Call AddTexture(FOComplex(
id).TexturesNames(y))
Next
Set FOComplex(id).tmp_mat = Nothing

If FileExists(meshfilesimple) = True Then
    Set FOSimple(id).Mesh = Direct3DX.LoadMeshFromX(meshfilesimple, D3DXMES
H_MANAGED, Device, FOSimple(id).tmp_adj, FOSimple(id).tmp_mat, FOSimple(id)
.NumMaterials)
    ReDim FOSimple(id).Materials(FOSimple(id).NumMaterials - 1)
    ReDim FOSimple(id).TexturesNames(FOSimple(id).NumMaterials - 1)
    OptimizeMesh FOSimple(id).Mesh, FOSimple(id).tmp_adj
    Set FOSimple(id).tmp_adj = Nothing

    For y = 0 To FOSimple(id).NumMaterials - 1
        Direct3DX.BufferGetMaterial FOSimple(id).tmp_mat, y, FOSimple(id).M
aterials(y)
        FOSimple(id).Materials(y).Ambient = FOSimple(id).Materials(y).diffu
se     'aspect patch!
        FOSimple(id).TexturesNames(y) = UCase(GetFileName(Direct3DX.BufferG
etTextureName(FOSimple(id).tmp_mat, y)))
        If FOSimple(id).TexturesNames(y) <> "" Then Call AddTexture(FOSimpl
e(id).TexturesNames(y))
    Next
```

```
    Set FOSimple(id).tmp_mat = Nothing
End If

SkipLoadingFO:
LoadFixedObject.MeshID = id
LoadFixedObject.WorldID = WorldID
If FileExists(meshfilesimple) Then FOSimple(id).MeshName = UCase(GetFileNam
e(meshfilesimple))
FOComplex(id).MeshName = UCase(GetFileName(meshfile))

If FOSimple(id).MeshName = "" Then LoadFixedObject.unimesh = 1

Dim Out As salida, center As D3DVECTOR, radius As Single

If Position.y = -9999 Then
    segintersectfast v3(Position.x, 500, Position.z), v3(0, -1, 0), Collisi
onFloats(WorldID).vertices(0), UBound(CollisionFloats(WorldID).vertices) /
3, 1, Out

    LoadFixedObject.Position.y = Out.puntocolision.y
End If

Direct3DX.ComputeBoundingSphereFromMesh FOComplex(LoadFixedObject.MeshID).M
esh, center, radius
D3DXMatrixTranslation TMatrix, LoadFixedObject.Position.x, LoadFixedObject.
Position.y, LoadFixedObject.Position.z
D3DXVec3TransformCoord center, center, TMatrix

LoadFixedObject.transformedBoundingSphere.center = center
LoadFixedObject.transformedBoundingSphere.radius = radius
End Function

Public Function LoadFixedObjectStage(Position As D3DVECTOR, ByVal meshfile
As String, ByVal meshfilesimple As String, AutoY As Boolean, RotationY As S
ingle, WorldID As Integer) As FixedObject
Dim res As salida, x As Long, id As Long, y As Long
LoadFixedObjectStage.Position = Position

For x = 1 To UBound(FOComplexStage)
    If FOComplexStage(x).MeshName = UCase(GetFileName(meshfile)) And _
       FOSimpleStage(x).MeshName = UCase(GetFileName(meshfilesimple)) Then
        id = x
        GoTo SkipLoadingFOS
    End If
Next

ReDim Preserve FOComplexStage(UBound(FOComplexStage) + 1)
ReDim Preserve FOSimpleStage(UBound(FOSimpleStage) + 1)
id = UBound(FOSimpleStage)

Set FOComplexStage(id).Mesh = Direct3DX.LoadMeshFromX(meshfile, D3DXMESH_MA
NAGED, Device, FOComplexStage(id).tmp_adj, FOComplexStage(id).tmp_mat, FOCo
mplexStage(id).NumMaterials)
ReDim FOComplexStage(id).Materials(FOComplexStage(id).NumMaterials - 1)
ReDim FOComplexStage(id).TexturesNames(FOComplexStage(id).NumMaterials - 1)
OptimizeMesh FOComplexStage(id).Mesh, FOComplexStage(id).tmp_adj
Set FOComplexStage(id).tmp_adj = Nothing

For y = 0 To FOComplexStage(id).NumMaterials - 1
    Direct3DX.BufferGetMaterial FOComplexStage(id).tmp_mat, y, FOComplexSta
```

```
ge(id).Materials(y)
    FOComplexStage(id).Materials(y).Ambient = FOComplexStage(id).Materials(
y).diffuse     'aspect patch!
    FOComplexStage(id).TexturesNames(y) = UCase(GetFileName(Direct3DX.Buffe
rGetTextureName(FOComplexStage(id).tmp_mat, y)))
    If FOComplexStage(id).TexturesNames(y) <> "" Then Call AddTexture(FOCom
plexStage(id).TexturesNames(y))
Next
Set FOComplexStage(id).tmp_mat = Nothing

If FileExists(meshfilesimple) = True Then
    Set FOSimpleStage(id).Mesh = Direct3DX.LoadMeshFromX(meshfilesimple, D3
DXMESH_MANAGED, Device, FOSimpleStage(id).tmp_adj, FOSimpleStage(id).tmp_ma
t, FOSimpleStage(id).NumMaterials)
    ReDim FOSimpleStage(id).Materials(FOSimpleStage(id).NumMaterials - 1)
    ReDim FOSimpleStage(id).TexturesNames(FOSimpleStage(id).NumMaterials -
1)
    OptimizeMesh FOSimpleStage(id).Mesh, FOSimpleStage(id).tmp_adj
    Set FOSimpleStage(id).tmp_adj = Nothing

    For y = 0 To FOSimpleStage(id).NumMaterials - 1
        Direct3DX.BufferGetMaterial FOSimpleStage(id).tmp_mat, y, FOSimpleS
tage(id).Materials(y)
        FOSimpleStage(id).Materials(y).Ambient = FOSimpleStage(id).Material
s(y).diffuse     'aspect patch!
        FOSimpleStage(id).TexturesNames(y) = UCase(GetFileName(Direct3DX.Bu
fferGetTextureName(FOSimpleStage(id).tmp_mat, y)))
        If FOSimpleStage(id).TexturesNames(y) <> "" Then Call AddTexture(FO
SimpleStage(id).TexturesNames(y))
    Next
    Set FOSimpleStage(id).tmp_mat = Nothing
End If

SkipLoadingFOS:
LoadFixedObjectStage.MeshID = id
LoadFixedObjectStage.WorldID = WorldID
If FileExists(meshfilesimple) Then FOSimpleStage(id).MeshName = UCase(GetFi
leName(meshfilesimple))
FOComplexStage(id).MeshName = UCase(GetFileName(meshfile))

If FOSimpleStage(id).MeshName = "" Then LoadFixedObjectStage.unimesh = 1

Dim Out As salida, center As D3DVECTOR, radius As Single

If Position.y = -9999 Then
    segintersectfast v3(Position.x, 500, Position.z), v3(0, -1, 0), Collisi
onFloats(WorldID).vertices(0), UBound(CollisionFloats(WorldID).vertices) /
3, 1, Out

    LoadFixedObjectStage.Position.y = Out.puntocolision.y
End If

Direct3DX.ComputeBoundingSphereFromMesh FOComplexStage(LoadFixedObjectStage
.MeshID).Mesh, center, radius
D3DXMatrixTranslation TMatrix, LoadFixedObjectStage.Position.x, LoadFixedOb
jectStage.Position.y, LoadFixedObjectStage.Position.z
D3DXVec3TransformCoord center, center, TMatrix

LoadFixedObjectStage.transformedBoundingSphere.center = center
LoadFixedObjectStage.transformedBoundingSphere.radius = radius
```

```
AuxFunctions - 9

End Function

'--------------------------------------------------------------------------
----------
'-------------------------------- MESH EXTENDED FUNCTIONS -----------------
----------
'--------------------------------------------------------------------------
----------

Public Sub ExtractTriVec(Mesh As D3DXMesh, tris() As D3DVECTOR)
On Local Error Resume Next
Dim hresult As Long
Dim vertices() As D3DVERTEX
Dim desc As D3DINDEXBUFFER_DESC
Dim IBuf As Direct3DIndexBuffer8
Dim indices() As Integer, x As Long, y As Long, ignore As Long, indices32()
 As Long

ReDim vertices(Mesh.GetNumVertices)
ReDim tris(Mesh.GetNumFaces * 3 - 1)

hresult = D3DXMeshVertexBuffer8GetData(Mesh, 0, Len(vertices(0)) * Mesh.Get
NumVertices, 0, vertices(0))

Set IBuf = Mesh.GetIndexBuffer()
IBuf.Lock 0, 0, ignore, 16
IBuf.GetDesc desc
IBuf.Unlock
If (Mesh.GetOptions And D3DXMESH_32BIT) Then
    '32 bit index mesh
    ReDim indices32(desc.Size / 4)   '4 as we use 32 bit mesh , 2 if we use
 16 bit
    D3DXMeshIndexBuffer8GetData Mesh, 0, desc.Size, 0, indices32(0)

    For y = 0 To Mesh.GetNumFaces * 3 - 1 Step 3
        tris(y).x = vertices(indices32(y)).x
        tris(y).y = vertices(indices32(y)).y
        tris(y).z = vertices(indices32(y)).z
        tris(y + 1).x = vertices(indices32(y + 1)).x
        tris(y + 1).y = vertices(indices32(y + 1)).y
        tris(y + 1).z = vertices(indices32(y + 1)).z
        tris(y + 2).x = vertices(indices32(y + 2)).x
        tris(y + 2).y = vertices(indices32(y + 2)).y
        tris(y + 2).z = vertices(indices32(y + 2)).z
    Next
Else
    '16 bit index mesh
    ReDim indices(desc.Size / 2)
    D3DXMeshIndexBuffer8GetData Mesh, 0, desc.Size, 0, indices(0)

    For y = 0 To Mesh.GetNumFaces * 3 - 1 Step 3
        tris(y).x = vertices(indices(y)).x
        tris(y).y = vertices(indices(y)).y
        tris(y).z = vertices(indices(y)).z
        tris(y + 1).x = vertices(indices(y + 1)).x
        tris(y + 1).y = vertices(indices(y + 1)).y
        tris(y + 1).z = vertices(indices(y + 1)).z
        tris(y + 2).x = vertices(indices(y + 2)).x
        tris(y + 2).y = vertices(indices(y + 2)).y
        tris(y + 2).z = vertices(indices(y + 2)).z
```

```
    Next
End If

ReDim indices(0)
ReDim indices32(0)
ReDim vertices(0)           'destroy all temp objects!!
Set IBuf = Nothing
End Sub

Public Sub ComputeNormals(verts() As D3DVECTOR, normals() As D3DVECTOR)
ReDim normals((UBound(verts) + 1) / 3)
lib_compute_normals verts(0), UBound(verts) + 1, normals(0)
End Sub

Public Sub AddToArray(array1() As D3DVECTOR, outarray() As D3DVECTOR)
Dim x As Long, start As Long
If UBound(outarray) = 0 Then
    'the array where add is empty
    ReDim Preserve outarray(UBound(array1))

    For x = 0 To UBound(array1)
        outarray(x) = array1(x)
    Next
Else
    start = UBound(outarray) + 1
    ReDim Preserve outarray(UBound(outarray) + UBound(array1) + 1)

    For x = 0 To UBound(array1)
        outarray(start + x) = array1(x)
    Next
End If
End Sub

Public Function DetermineMaxX(array1() As D3DVECTOR) As Single
Dim Max As Long, x As Long
For x = 0 To UBound(array1)
    If array1(x).x > Max Then Max = array1(x).x
Next
DetermineMaxX = Max
End Function

Public Function DetermineMaxZ(array1() As D3DVECTOR) As Single
Dim Max As Long, x As Long
For x = 0 To UBound(array1)
    If array1(x).z > Max Then Max = array1(x).z
Next
DetermineMaxZ = Max
End Function

Public Function DetermineMinX(array1() As D3DVECTOR) As Single
Dim Min As Long, x As Long
For x = 0 To UBound(array1)
    If array1(x).x < Min Then Min = array1(x).x
Next
DetermineMinX = Min
End Function

Public Function DetermineMinZ(array1() As D3DVECTOR) As Single
Dim Min As Long, x As Long
For x = 0 To UBound(array1)
```

```
    If array1(x).z < Min Then Min = array1(x).z
Next
DetermineMinZ = Min
End Function

Public Sub TransformMesh(Mesh As D3DXMesh, mat As D3DMATRIX)
'-- extract all the vertices to form triangles ------
Dim hresult As Long
Dim vertices() As D3DVERTEX
Dim avector As D3DVECTOR, y As Long

ReDim vertices(Mesh.GetNumVertices)
hresult = D3DXMeshVertexBuffer8GetData(Mesh, 0, Len(vertices(0)) * Mesh.Get
NumVertices, 0, vertices(0))

For y = 0 To Mesh.GetNumVertices - 1
    avector.x = vertices(y).x
    avector.y = vertices(y).y
    avector.z = vertices(y).z
    D3DXVec3TransformCoord avector, avector, mat
    vertices(y).x = avector.x
    vertices(y).y = avector.y
    vertices(y).z = avector.z
Next

D3DXMeshVertexBuffer8SetData Mesh, 0, Len(vertices(0)) * Mesh.GetNumVertice
s, 0, vertices(0)
End Sub

Public Sub TransformVerts(vertices() As D3DVECTOR, mat As D3DMATRIX)
'-- extract all the vertices to form triangles ------
Dim y As Long
For y = 0 To UBound(vertices)
    D3DXVec3TransformCoord vertices(y), vertices(y), mat
Next
End Sub

Public Sub OptimizeMesh(Mesh As D3DXMesh, adjbuffer As D3DXBuffer)
    Dim s As Long
    Dim adjBuf1() As Long
    Dim adjBuf2() As Long
    Dim facemap() As Long
    Dim vertexMap As D3DXBuffer

    s = adjbuffer.GetBufferSize
    ReDim adjBuf1(s / 4)
    ReDim adjBuf2(s / 4)

    s = Mesh.GetNumFaces
    ReDim facemap(s)

    Direct3DX.BufferGetData adjbuffer, 0, 4, s * 3, adjBuf1(0)
    Mesh.OptimizeInplace D3DXMESHOPT_COMPACT Or D3DXMESHOPT_ATTRSORT Or D3D
XMESHOPT_VERTEXCACHE, adjBuf1(0), adjBuf2(0), facemap(0), vertexMap

    ReDim adjBuf1(0)
    ReDim adjBuf2(0)
End Sub

Public Sub CreateVertexAndIndexBuffer(vertices() As D3DVECTOR, Indexbuffer
```

```
As Direct3DIndexBuffer8, VertexBuffer As Direct3DVertexBuffer8)
Dim newv() As D3DVECTOR
Dim numv As Long
Dim newi() As Integer
Dim numi As Long
Dim x As Long, y As Long

For x = 0 To UBound(vertices)
    For y = 0 To numv - 1
        If newv(y).x = vertices(x).x And newv(y).y = vertices(x).y And newv
(y).z = vertices(x).z Then
            'this vertex is repeated!
            'skip it!
            GoTo SkipThatVert
        End If
    Next
    'add the vertex to the list
    ReDim Preserve newv(numv)
    newv(numv) = vertices(x)
    numv = numv + 1
SkipThatVert:
Next

For x = 0 To UBound(vertices)
    For y = 0 To numv - 1
        If newv(y).x = vertices(x).x And newv(y).y = vertices(x).y And newv
(y).z = vertices(x).z Then
            ReDim Preserve newi(numi)
            newi(numi) = y
            numi = numi + 1
            Exit For
        End If
    Next
Next

Set Indexbuffer = Device.CreateIndexBuffer(2 * numi, D3DUSAGE_WRITEONLY, D3
DFMT_INDEX16, D3DPOOL_MANAGED)
D3DIndexBuffer8SetData Indexbuffer, 0, 2 * numi, 0, newi(0)
Set VertexBuffer = Device.CreateVertexBuffer(12 * numv, D3DUSAGE_WRITEONLY
Or D3DUSAGE_DONOTCLIP, D3DFVF_XYZ, D3DPOOL_MANAGED)
D3DVertexBuffer8SetData VertexBuffer, 0, 12 * numv, 0, newv(0)

ReDim newi(0)
ReDim newv(0)
End Sub

'-------------------------------------------------------------------------
----------
'-------------------------------- TEXTURE MANAGING ------------------------
----------
'-------------------------------------------------------------------------
----------

Public Sub AddTexture(ByVal file As String)
Dim x As Integer
For x = 1 To UBound(PendingTextures)
    If PendingTextures(x) = file Then Exit Sub
Next

ReDim Preserve PendingTextures(UBound(PendingTextures) + 1)
```

```
PendingTextures(UBound(PendingTextures)) = file
End Sub

Public Sub LoadTexture(ByVal file As String)
'--- adds a new texture into the texture array ----
Dim x As Integer
For x = 1 To UBound(GameTextures)
    If UCase(GetFileName(GameTextures(x).filename)) = UCase(GetFileName(fil
e)) Then Exit Sub
Next
ReDim Preserve GameTextures(UBound(GameTextures) + 1)

Set GameTextures(UBound(GameTextures)).texture = LoadTextureAndReturn(file)
GameTextures(UBound(GameTextures)).filename = UCase(GetFileName(file))
End Sub

Public Sub LoadTextureSec(ByVal file As String)
'--- adds a new texture into the secondary texture array ----
Dim x As Integer
For x = 1 To UBound(GameTexturesSec)
    If UCase(GetFileName(GameTexturesSec(x).filename)) = UCase(GetFileName(
file)) Then Exit Sub
Next
ReDim Preserve GameTexturesSec(UBound(GameTexturesSec) + 1)

Set GameTexturesSec(UBound(GameTexturesSec)).texture = LoadTextureAndReturn
(file)
GameTexturesSec(UBound(GameTexturesSec)).filename = UCase(GetFileName(file)
)
End Sub

Public Function LoadTextureAndReturn(ByVal file As String, Optional ByVal N
C As Boolean = False) As Direct3DTexture8
'--- loads a texture file and returnns a Direct3DTexture8 class ---
Dim TransparentImage As Boolean
Dim imagew As Long, imageh As Long

GetImageProperties file, imagew, imageh, TransparentImage

If caps.MaxTextureWidth < imagew Then imagew = caps.MaxTextureWidth
If caps.MaxTextureHeight < imageh Then imageh = caps.MaxTextureHeight
If caps.TextureCaps And D3DPTEXTURECAPS_SQUAREONLY Then
    If imagew > imageh Then
        imagew = imageh
    Else
        imageh = imagew
    End If
End If

CheckTextureDimensions imagew, imageh

If TransparentImage = False Then
    If Direct3D.CheckDeviceFormat(0, D3DDEVTYPE_HAL, D3DM.Format, 0, D3DRTY
PE_TEXTURE, D3DFMT_DXT1) = D3D_OK And NC = False Then
        Set LoadTextureAndReturn = Direct3DX.CreateTextureFromFileEx(Device
, file, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_DXT1, D3DPOOL_MANAGED, D3DX
_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    Else
        Set LoadTextureAndReturn = Direct3DX.CreateTextureFromFileEx(Device
, file, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D
```

```
3DX_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    End If
Else
    If Direct3D.CheckDeviceFormat(0, D3DDEVTYPE_HAL, D3DM.Format, 0, D3DRTY
PE_TEXTURE, D3DFMT_DXT5) = D3D_OK And NC = False Then
        Set LoadTextureAndReturn = Direct3DX.CreateTextureFromFileEx(Device
, file, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_DXT5, D3DPOOL_MANAGED, D3DX
_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    Else
        Set LoadTextureAndReturn = Direct3DX.CreateTextureFromFileEx(Device
, file, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D
3DX_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    End If
End If
End Function

Public Sub LoadPendingTextures(Optional ByVal Pi As Integer, Optional ByVal
 PE As Integer)
On Local Error GoTo TexError
Dim mypath As String, x As Integer
ReDim GameTextures(0)
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "tex_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
MkDir mypath
If ExtractFile(EXE & "maps\main.dat", mypath) = True Then
    For x = 1 To UBound(PendingTextures)
        Call LoadTexture(mypath & PendingTextures(x))
        If Pi <> 0 Then ShowLoadPercent Trim(Str(Int(((PE - Pi) * x / UBoun
d(PendingTextures) + Pi))))
    Next
Else
    MsgBox "El programa no està correctament instal·lat. Falten un o més ar
xius o no són correctes", vbCritical, "Error intern"
End If

DeleteDir mypath
ReDim PendingTextures(0)
Exit Sub

TexError:
DeleteDir mypath
On Local Error Resume Next
MouseDevice.Unacquire
Set Device = Nothing
frmGraphics.Hide
Unload frmGraphics
DoEvents

MsgBox "Error greu carregant el joc!", vbCritical, "Error intern"
End
End Sub

Public Sub LoadPendingTexturesSec(Optional ByVal Pi As Integer, Optional By
Val PE As Integer)
On Local Error GoTo TexError
Dim mypath As String, x As Integer
ReDim GameTexturesSec(0)
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
```

```
mypath = mypath & "tex_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
MkDir mypath
If ExtractFile(EXE & "maps\sec.dat", mypath) = True Then
    For x = 1 To UBound(PendingTextures)
        Call LoadTextureSec(mypath & PendingTextures(x))
        If Pi <> 0 Then ShowLoadPercentStage Trim(Str(Int(((PE - Pi) * x /
UBound(PendingTextures) + Pi))))
    Next
End If

DeleteDir mypath
ReDim PendingTextures(0)
Exit Sub

TexError:
DeleteDir mypath
On Local Error Resume Next
MouseDevice.Unacquire
Set Device = Nothing
frmGraphics.Hide
Unload frmGraphics
DoEvents

MsgBox "Error greu carregant el joc!", vbCritical, "Error intern"
End
End Sub

Public Sub DeviceSetTexture(ByVal file As String)
On Local Error Resume Next
Dim x As Integer
For x = 1 To UBound(GameTextures)
    If GameTextures(x).filename = file Then
        Device.SetTexture 0, GameTextures(x).texture
        Exit Sub
    End If
Next
For x = 1 To UBound(GameTexturesSec)
    If GameTexturesSec(x).filename = file Then
        Device.SetTexture 0, GameTexturesSec(x).texture
        Exit Sub
    End If
Next
End Sub

'--------------------------------------------------------------------------
----------
'---------------------------------- MISSION TARGETS -----------------------
----------
'--------------------------------------------------------------------------
----------

Public Sub AddMissionTarget(ByVal mid As String, pos As D3DVECTOR, ByVal he
ight As Single, ByVal radius As Single, ByVal WID As Long)
Dim x As Long
x = UBound(MissionTargets) + 1
ReDim Preserve MissionTargets(x)
MissionTargets(x).Position = pos
MissionTargets(x).height = height
MissionTargets(x).radius = radius
MissionTargets(x).WorldID = WID
```

```
MissionTargets(x).id = mid

If pos.y = -9999 Then
    Dim Out As salida
    segintersectfast v3(pos.x, 500, pos.z), v3(0, -1, 0), CollisionFloats(W
ID).vertices(0), UBound(CollisionFloats(WID).vertices) / 3, 1, Out
    MissionTargets(x).Position.y = Out.puntocolision.y
End If
End Sub

'-------------------------------------------------------------------------
----------
'---------------------------------------- OTHERS -------------------------
----------
'-------------------------------------------------------------------------
----------

Public Function CheckPointInCube(vert1 As D3DVECTOR, vert2 As D3DVECTOR, po
int As D3DVECTOR) As Boolean
If vert1.x < vert2.x Then
    If Not (point.x > vert1.x And point.x < vert2.x) Then Exit Function
Else
    If Not (point.x < vert1.x And point.x > vert2.x) Then Exit Function
End If
If vert1.z < vert2.z Then
    If Not (point.z > vert1.z And point.z < vert2.z) Then Exit Function
Else
    If Not (point.z < vert1.z And point.z > vert2.z) Then Exit Function
End If
If vert1.y < vert2.y Then
    If Not (point.y > vert1.y And point.y < vert2.y) Then Exit Function
Else
    If Not (point.y < vert1.y And point.y > vert2.y) Then Exit Function
End If
CheckPointInCube = True
End Function

Public Function TempPath() As String
Dim var As String
var = Space(512)
GetTempPath 512, var
TempPath = left(var, InStr(1, var, Chr(0)) - 1)
End Function

Public Function AssignMV(ByVal x As Single, ByVal y As Single, ByVal z As S
ingle, ByVal tu As Single, ByVal tv As Single) As myVertex
AssignMV.x = x
AssignMV.y = y
AssignMV.z = z
AssignMV.tu = tu
AssignMV.tv = tv
AssignMV.rhw = 1
AssignMV.Color = D3DColorARGB(0, 255, 255, 255)
End Function

Public Function AssignMVS(ByVal x As Single, ByVal y As Single, ByVal z As
Single, ByVal tu As Single, ByVal tv As Single) As myVertexSimple
AssignMVS.x = x
AssignMVS.y = y
AssignMVS.z = z
```

```
AssignMVS.tu = tu
AssignMVS.tv = tv
AssignMVS.Color = D3DColorARGB(0, 255, 255, 255)
End Function

Public Function AssignMVA(ByVal x As Single, ByVal y As Single, ByVal z As
Single, ByVal Color As Long) As myVertexAlpha
AssignMVA.x = x
AssignMVA.y = y
AssignMVA.z = z
AssignMVA.Color = Color
AssignMVA.rhw = 1
End Function

Public Sub MouseSetUp()
MouseDevice.Unacquire
Set MouseDevice = Nothing
Set MouseDevice = DirectInput.CreateDevice("guid_SysMouse")

MouseDevice.SetCommonDataFormat DIFORMAT_MOUSE
MouseDevice.SetCooperativeLevel frmGraphics.hWnd, DISCL_EXCLUSIVE Or DISCL_
FOREGROUND

Dim diProp As DIPROPLONG
diProp.lHow = DIPH_DEVICE
diProp.lObj = 0
diProp.lData = 50      '50 Buffer for the mouse!!!

Call MouseDevice.SetProperty("DIPROP_BUFFERSIZE", diProp)

DI_hevent = DirectX.CreateEvent(frmGraphics)
MouseDevice.SetEventNotification DI_hevent

MouseDevice.Acquire
End Sub

Public Function GetAllResolutions(ByVal MinWidth As Single, ByVal MinHeight
 As Single, Out As Variant)
Dim x As Integer, y As Integer
ReDim Out(0)
For x = 0 To UBound(DModesArray) - 1
    If DModesArray(x).width >= MinWidth And DModesArray(x).height >= MinHei
ght Then
        For y = 0 To UBound(Out)
            If (Out(y) = DModesArray(x).width & " x " & DModesArray(x).heig
ht) Then GoTo MyOut
        Next
        ReDim Preserve Out(UBound(Out) + 1)
        Out(UBound(Out)) = DModesArray(x).width & " x " & DModesArray(x).he
ight
    End If
MyOut:
Next
End Function

Public Sub DrawFont(ByVal text As String, ByVal x As Single, ByVal y As Sin
gle, Optional ByVal totalwidth As Single = 0, Optional ByVal height As Sing
le = 0, Optional ByVal Fontwidth As Single = 0)
If text = "" Then Exit Sub
Dim f As Integer, char As Byte, g As Integer
```

```
Dim xpos As Integer, ypos As Integer
Dim width As Single
Dim tu As Single, tu2 As Single, tv As Single, tv2 As Single

Device.SetTexture 0, FontTexture

For f = 1 To Len(text)
    ypos = 0: xpos = 0
    char = Asc(mid(text, f, 1))
    If (char >= 65 And char <= 90) Or (char >= 97 And char <= 122) Then
'letras
        If char < 90 Then char = char - 65
        If char > 90 Then char = char - 97
        Do While char >= 8
            ypos = ypos + 1
            char = char - 8
        Loop
        xpos = char
    ElseIf char >= 48 And char <= 57 Then
        char = char - 48
        ypos = 4
        If char > 7 Then
            ypos = ypos + 1
            char = char - 8
        End If
        xpos = char
    ElseIf char >= 44 And char <= 47 Then
        xpos = 2 + (char - 44)
        ypos = 3
    ElseIf char = 92 Then
        xpos = 6: ypos = 3
    ElseIf char = 59 Then
        xpos = 7: ypos = 3
    ElseIf char = 33 Then
        xpos = 2: ypos = 5
    ElseIf char = 34 Then
        xpos = 3: ypos = 5
    ElseIf char = 39 Then
        xpos = 4: ypos = 5
    ElseIf char = 186 Then            'º --->  flecha abajo
        xpos = 1: ypos = 6
    ElseIf char = 170 Then             'ª --->  flecha arriba
        xpos = 0: ypos = 6
    End If

    If char <> 32 Then
        If totalwidth = 0 Then
            width = Fontwidth * Len(text)
            If width = 0 Then width = Len(text) * 32
        Else
            width = totalwidth
        End If
        If height = 0 Then height = 32

        tu = (xpos * 32) / 256
        tu2 = ((xpos + 1) * 32) / 256
        tv = (ypos * 32) / 256
        tv2 = ((ypos + 1) * 32) / 256

        FontVertices(0) = AssignMV(x + (f - 1) * width / Len(text), y, 0, t
```

```
u, tv)
        FontVertices(1) = AssignMV(x + f * width / Len(text), y, 0, tu2, tv
)
        FontVertices(2) = AssignMV(x + (f - 1) * width / Len(text), y + hei
ght, 0, tu, tv2)
        FontVertices(3) = AssignMV(x + f * width / Len(text), y + height, 0
, tu2, tv2)

        Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, FontVertices(0), Len
(FontVertices(0))
    End If
Next
End Sub

Public Function ParseResolution(ByVal var As String) As D3DDISPLAYMODE
Dim x As Integer
x = InStr(1, var, "x")
ParseResolution.width = Val(left(var, x - 2))
ParseResolution.height = Val(mid(var, x + 2))
End Function

Public Function Is32bit(DMode As D3DDISPLAYMODE) As Boolean
If DMode.Format = D3DFMT_X8R8G8B8 Then
    Is32bit = True
Else
    Is32bit = False
End If
End Function

Public Sub SaveRS()
RenderStates(0) = Device.GetRenderState(D3DRS_ALPHABLENDENABLE)
RenderStates(1) = Device.GetRenderState(D3DRS_ZENABLE)
RenderStates(2) = Device.GetRenderState(D3DRS_ZWRITEENABLE)
RenderStates(3) = Device.GetRenderState(D3DRS_LIGHTING)
RenderStates(4) = Device.GetRenderState(D3DRS_DESTBLEND)
RenderStates(5) = Device.GetRenderState(D3DRS_SRCBLEND)
RenderStates(6) = Device.GetRenderState(D3DRS_DIFFUSEMATERIALSOURCE)
RenderStates(7) = Device.GetTextureStageState(0, D3DTSS_MAGFILTER)
RenderStates(8) = Device.GetTextureStageState(0, D3DTSS_MINFILTER)
End Sub

Public Sub RestoreRS()
Device.SetRenderState D3DRS_ALPHABLENDENABLE, RenderStates(0)
Device.SetRenderState D3DRS_ZENABLE, RenderStates(1)
Device.SetRenderState D3DRS_ZWRITEENABLE, RenderStates(2)
Device.SetRenderState D3DRS_LIGHTING, RenderStates(3)
Device.SetRenderState D3DRS_DESTBLEND, RenderStates(4)
Device.SetRenderState D3DRS_SRCBLEND, RenderStates(5)
Device.SetRenderState D3DRS_DIFFUSEMATERIALSOURCE, RenderStates(6)
Device.SetTextureStageState 0, D3DTSS_MAGFILTER, RenderStates(7)
Device.SetTextureStageState 0, D3DTSS_MINFILTER, RenderStates(8)
End Sub

Public Sub RefreshDrawDepth()
Select Case DrawDepth
Case 1
    DistanceFOAppear = 30
    DistanceFODetail = 8
    FarViewPlane = 120
Case 2
```

```
    DistanceFOAppear = 50
    DistanceFODetail = 12
    FarViewPlane = 190
Case 3
    DistanceFOAppear = 80
    DistanceFODetail = 16
    FarViewPlane = 250
End Select
ShadowCullSize = (FarViewPlane * 0.8) * Cos(Pi / 4) / 4
'DistanceFOAppear   '30 low,  50 medium,  80 high
'DistanceFODetail   '8 low, 12 medium, 16 high
'FarViewPlane       '120 low, 190 medium, 250 high
End Sub

Public Sub LoadSettings()
Dim var As String
var = RegRead("drawdepth")
Select Case Val(var)
Case 1, 2, 3
    DrawDepth = Val(var)
Case Else
    DrawDepth = 2
End Select

var = RegRead("cursorspeed")
Select Case Val(var)
Case 1, 2, 3, 4, 5
    CursorSpeed = Val(var)
Case Else
    CursorSpeed = 3
End Select

var = RegRead("chardetail")
Select Case Val(var)
Case 1, 2, 3
    CharDetail = Val(var)
Case Else
    CharDetail = 2
End Select

var = RegRead("texq")
Select Case Val(var)
Case 1, 2
    TexQuality = Val(var)
Case Else
    TexQuality = 2
End Select
End Sub

Public Sub SaveSettings()
RegSave "drawdepth", Trim(Str(DrawDepth))
RegSave "cursorspeed", Trim(Str(CursorSpeed))
RegSave "chardetail", Trim(Str(CharDetail))
RegSave "texq", Trim(Str(TexQuality))
End Sub

Public Sub CheckTextureDimensions(width As Long, height As Long)
If TexQuality = 1 Then
    If width > 256 Then width = 256
    If height > 256 Then height = 256
```

```
End If
End Sub

'------------------------------------------------------------------------
-------
'-------------------------------- FILE FUNCTIONS -------------------------
-------
'------------------------------------------------------------------------
-------

Public Sub DeleteDir(ByVal directory As String)
'---- Removes all the content of an existing folder (including subfolders)
----
On Local Error Resume Next
If directory = "" Then Exit Sub
Dim cad As String
If right(directory, 1) <> "\" Then directory = directory & "\"
Kill directory & "*.*"

Again:
cad = Dir(directory, vbDirectory)
Do While cad <> ""
    If cad <> "." And cad <> ".." Then
        DeleteDir directory & cad
        GoTo Again
    End If
    cad = Dir$
Loop
RmDir directory
End Sub

Public Sub GetImageProperties(ByVal file As String, width As Long, height A
s Long, transparent As Boolean)
Dim dib As Long
Select Case LCase(right(file, 3))
Case "jpg", "jpeg"
    dib = FreeImage_Load(FIF_JPEG, file)
    transparent = False
Case "bmp"
    dib = FreeImage_Load(FIF_BMP, file)
    transparent = False
Case "tga"
    dib = FreeImage_Load(FIF_TARGA, file)
    transparent = True
End Select
width = FreeImage_GetWidth(dib)
height = FreeImage_GetHeight(dib)
Call FreeImage_Unload(dib)
End Sub

Public Sub LoadAndParseFile(ByVal file As String)
ReDim PNames(0)
ReDim PValues(0)
Dim f As Integer, cad As String
f = FreeFile()
Open file For Input As #f
Do While Not EOF(f)
    Line Input #f, cad
    If left(cad, 1) <> "#" And cad <> "" And InStr(1, cad, "=") <> 0 Then
        ReDim Preserve PNames(UBound(PNames) + 1)
```

```
        ReDim Preserve PValues(UBound(PValues) + 1)
        PNames(UBound(PNames)) = left(cad, InStr(1, cad, "=") - 1)
        PValues(UBound(PValues)) = mid(cad, InStr(1, cad, "=") + 1)
    End If
Loop
Close #f
End Sub

Public Function PFGetProperty(ByVal PropertyName As String) As String
Dim x As Integer
For x = 1 To UBound(PNames)
    If UCase(PNames(x)) = UCase(PropertyName) Then
        PFGetProperty = PValues(x)
        Exit Function
    End If
Next
End Function

Public Function GetFileName(ByVal file As String) As String
If file = "" Then Exit Function
Dim xx As Long, x As Long
For x = Len(file) To 1 Step -1
    xx = InStr(x, file, "\")
    If xx <> 0 Then Exit For
Next
GetFileName = mid$(file, xx + 1, Len(file) - xx + 1)
End Function

Public Function FileExists(ByVal filename As String) As Boolean
'------ Returns if the file exists -------
On Local Error Resume Next
Dim temp As VbFileAttribute, fff As Integer
temp = GetAttr(filename)
fff = FreeFile()
Open filename For Binary As #fff
Close #fff
If Err.number <> 0 Then
    FileExists = False
Else
    FileExists = True
End If
Err.Clear
Err.number = 0
End Function

'-------------------------------------------------------------------------
-------
'-------------------------------- ANIM3D FUNCTIONS ------------------------
-------
'-------------------------------------------------------------------------
-------

Public Sub LoadAnim3D(Anim As Anim3D, ByVal path As String, ByVal root As S
tring)
Dim ignore As Long, x As Long
LoadAndParseFile path & root & ".txt"

Anim.NumMeshes = Val(PFGetProperty("nummeshes"))
Anim.DurationMS = Val(PFGetProperty("duration"))
ReDim Anim.Meshes(1 To Anim.NumMeshes)
```

```
For x = 1 To Anim.NumMeshes
    If x = 1 Then
        Set Anim.Meshes(x) = Direct3DX.LoadMeshFromX(path & root & "_" & Fo
rmat(x, "0000") & ".x", D3DXMESH_MANAGED, Device, Anim.tmp_adj, Anim.tmp_ma
t, Anim.NumMaterials)
    Else
        Set Anim.Meshes(x) = Direct3DX.LoadMeshFromX(path & root & "_" & Fo
rmat(x, "0000") & ".x", D3DXMESH_MANAGED, Device, Nothing, Nothing, ignore)
    End If
    OptimizeMesh Anim.Meshes(x), Anim.tmp_adj
Next

ReDim Anim.Materials(Anim.NumMaterials - 1)
ReDim Anim.TexturesNames(Anim.NumMaterials - 1)
For x = 0 To Anim.NumMaterials - 1
    Direct3DX.BufferGetMaterial Anim.tmp_mat, x, Anim.Materials(x)
    Anim.Materials(x).Ambient = Anim.Materials(x).diffuse     'aspect patch
!
    Anim.TexturesNames(x) = UCase(GetFileName(Direct3DX.BufferGetTextureNam
e(Anim.tmp_mat, x)))
    If Anim.TexturesNames(x) <> "" Then Call AddTexture(Anim.TexturesNames(
x))
Next

Set Anim.tmp_adj = Nothing
Set Anim.tmp_mat = Nothing
End Sub

Public Sub DestroyAnim3D(Anim As Anim3D)
Dim x As Integer
For x = 1 To Anim.NumMeshes
    Set Anim.Meshes(x) = Nothing
    Set Anim.tmp_adj = Nothing
    Set Anim.tmp_mat = Nothing
    ReDim Anim.Materials(0)
    ReDim Anim.TexturesNames(0)
Next
End Sub


'-------------------------------------------------------------------------
-------
'------------------------------- C++ Functions --------------------------
-------
'-------------------------------------------------------------------------
-------

Public Function ProcessCameraCoords(Position As D3DVECTOR, ByVal AngleH As
Single, ByVal AngleV As Single, ByVal distance As Single) As D3DVECTOR
Dim CameraDistanceProj As Single, CameraRelX As Single, CameraRelZ As Singl
e
CameraDistanceProj = distance * Cos(AngleV * Pi / 180)
CameraRelX = CameraDistanceProj * sin(AngleH * Pi / 180)
CameraRelZ = CameraDistanceProj * Cos(AngleH * Pi / 180)
ProcessCameraCoords.x = CameraRelX + Position.x
ProcessCameraCoords.z = CameraRelZ + Position.z
ProcessCameraCoords.y = (distance * sin(AngleV * Pi / 180)) + Position.y
End Function
```

Declarations - 1

```
Option Explicit

Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
Public Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long)
As Integer
Public Declare Function GetTickCount Lib "kernel32" () As Long
Public Declare Function ShowCursor Lib "user32" (ByVal bShow As Long) As Lo
ng
Public Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" (By
Val nBufferLength As Long, ByVal lpBuffer As String) As Long

'--- FreeImage Lib
Public Declare Function FreeImage_Load Lib "freeimage.dll" Alias "_FreeImag
e_Load@12" (_
           ByVal fif As FREE_IMAGE_FORMAT, _
           ByVal filename As String, _
  Optional ByVal Flags As Long = 0) As Long

Public Declare Function FreeImage_GetWidth Lib "freeimage.dll" Alias "_Free
Image_GetWidth@4" (ByVal dib As Long) As Long
Public Declare Function FreeImage_GetHeight Lib "freeimage.dll" Alias "_Fre
eImage_GetHeight@4" (ByVal dib As Long) As Long
Public Declare Sub FreeImage_Unload Lib "freeimage.dll" Alias "_FreeImage_U
nload@4" (ByVal dib As Long)

Public Enum FREE_IMAGE_FORMAT
   FIF_UNKNOWN = -1: FIF_BMP = 0: FIF_ICO = 1: FIF_JPEG = 2: FIF_JNG = 3: F
IF_KOALA = 4
   FIF_LBM = 5: FIF_IFF = FIF_LBM: FIF_MNG = 6: FIF_PBM = 7: FIF_PBMRAW = 8
: FIF_PCD = 9
   FIF_PCX = 10: FIF_PGM = 11: FIF_PGMRAW = 12: FIF_PNG = 13: FIF_PPM = 14:
 FIF_PPMRAW = 15
   FIF_RAS = 16: FIF_TARGA = 17: FIF_TIFF = 18: FIF_WBMP = 19: FIF_PSD = 20
: FIF_CUT = 21
   FIF_XBM = 22: FIF_XPM = 23: FIF_DDS = 24: FIF_GIF = 25: FIF_HDR = 26: FI
F_FAXG3 = 27: FIF_SGI = 28
End Enum
'----------------

Public Const Pi As Double = 3.14159265358979    'or magic formula: atn(1) *
 4
Public Const MaxCameraDistance As Single = 6
Public Const MinCameraDistance As Single = 4
Public Const MinVerticalAngle As Single = 20
Public Const MaxVerticalAngle As Single = 45

Public Const WorldAcceleration As Single = 9.8

Public Type MEMORYSTATUS
    dwLength As Long
    dwMemoryLoad As Long
    dwTotalPhys As Long
    dwAvailPhys As Long
    dwTotalPageFile As Long
    dwAvailPageFile As Long
    dwTotalVirtual As Long
    dwAvailVirtual As Long
End Type

Public Type Model3D
```

Declarations - 2

```
    MeshName As String

    Mesh As D3DXMesh
    Materials() As D3DMATERIAL8
    NumMaterials As Long
    TexturesNames() As String

    tmp_adj As D3DXBuffer
    tmp_mat As D3DXBuffer
End Type

Public Type CollisionFloat
    vertices() As D3DVECTOR
End Type

Public Type ShadowedWorld
    MinWidth As Single
    MinHeight As Single
    HeightBlocks As Integer
    WidthBlocks As Integer
End Type

Public Type Anim3D
    NumMeshes As Long
    DurationMS As Single
    Meshes() As D3DXMesh

    Materials() As D3DMATERIAL8
    NumMaterials As Long
    TexturesNames() As String

    tmp_adj As D3DXBuffer
    tmp_mat As D3DXBuffer

    Tag As String       'used by some special functions
    TagLong As Long
End Type

Public Type WorldProp
    SkyBox As Integer
    RenderSky As Boolean
    AmbientValues As D3DVECTOR  'color as d3dvector :P
End Type

Public Type myVertex        'texture  and transformed vert
    x As Single
    y As Single
    z As Single
    rhw As Single
    Color As Long
    tu As Single
    tv As Single
End Type
Public Const myVertexFVF = D3DFVF_XYZRHW Or D3DFVF_DIFFUSE Or D3DFVF_TEX1

Public Type myVertexSimple  'texture but not transformes
    x As Single
    y As Single
    z As Single
    Color As Long
```

```
Declarations - 3

    tu As Single
    tv As Single
End Type
Public Const myVertexFVFSimple = D3DFVF_XYZ Or D3DFVF_DIFFUSE Or D3DFVF_TEX
1

Public Type myVertexAlpha       'not texture, bu transformed and alpha
    x As Single
    y As Single
    z As Single
    rhw As Single
    Color As Long
End Type
Public Const myVertexAlphaFVF = D3DFVF_XYZRHW Or D3DFVF_DIFFUSE

Public Type salida
    respuesta As Integer
    puntocolision As D3DVECTOR
End Type

Public Declare Sub hprocess Lib "engine.dll" (ByRef Tri As D3DVECTOR, seg A
s D3DVECTOR, ByVal numtri As Long, ByRef collide As salida)
Public Declare Sub process Lib "engine.dll" (ByRef cam As D3DVECTOR, Tri As
 D3DVECTOR, ByVal numtri As Long, ByRef pos As D3DVECTOR, ByRef pos2 As D3D
VECTOR, ByRef upv As D3DVECTOR, ByVal distance As Double, ByVal angle As Do
uble, ByVal AngleH As Double, ByVal disfromcol As Single, ByVal interpolati
onspeed As Single, ByVal midh As Single, ByVal hih As Single, ByVal avrfram
e As Single, ByVal speed As Single, ByVal dimensions As Single, ByRef tris
As Long)
Public Declare Sub segintersect Lib "engine.dll" (ByRef origin As D3DVECTOR
, ByRef Direction As D3DVECTOR, ByRef Tri As D3DVECTOR, ByVal numtris As Lo
ng, ByVal numsegs As Long, ByRef collide As salida)
Public Declare Sub segintersectfast Lib "engine.dll" (ByRef origin As D3DVE
CTOR, ByRef Direction As D3DVECTOR, ByRef Tri As D3DVECTOR, ByVal numtris A
s Long, ByVal numsegs As Long, ByRef collide As salida)
Public Declare Sub extrudeshadow Lib "engine.dll" (ByVal proj As Long, ByRe
f tris As D3DVECTOR, ByVal numtri As Long, ByRef light As D3DVECTOR, ByRef
trisout As D3DVECTOR, ByRef numtrisout As Long)
Public Declare Sub lib_compute_normals Lib "engine.dll" Alias "computenorma
ls" (ByRef verts As D3DVECTOR, ByVal numverts As Long, ByRef normals As D3D
VECTOR)
Public Declare Sub createindices Lib "engine.dll" (ByRef vertices As D3DVEC
TOR, ByVal numverts As Long, ByRef vertsout As D3DVECTOR, ByRef numvertsout
 As Long, ByRef indices As Long, ByRef numindicesout As Long)

Public Type Sphere
    center As D3DVECTOR
    radius As Single
End Type

Public Type DMode
    Ancho As Long
    Alto As Long
    Frecuencia As Long
    Formato As Long
End Type

Public Type Sound
    file As String
    soundname As String
```

```
Declarations - 4

    buffer3d As DirectSound3DBuffer8
    buffersec As DirectSoundSecondaryBuffer8
    desc As DSBUFFERDESC
End Type

Public Type SoundStandard
    file As String
    soundname As String
    buffer As DirectSoundSecondaryBuffer8
    desc As DSBUFFERDESC
End Type

Public Type FixedObject
    MeshID As Long

    Position As D3DVECTOR
    transformedBoundingSphere As Sphere
    unimesh As Byte
    WorldID As Integer

    id As String
End Type

Type WorldShadow
    ShadowsArray() As clsShadow
    NumShadows As Long
End Type

Type WorldShadowCull
    CullArray() As clsCullNode
    NumCull As Long
End Type

Public Type gamecondition
    type As String
    varg As D3DVECTOR
    varg2 As D3DVECTOR
    varg3 As D3DVECTOR
    larg As Long
    larg2 As Long
    sarg As String
    sarg2 As String
End Type

Public Type gameeffect
    type As String
    varg As D3DVECTOR
    varg2 As D3DVECTOR
    varg3 As D3DVECTOR
    larg As Long
    larg2 As Long
    sarg As String
    sarg2 As String
End Type

Public Type Scene
    Conditions() As gamecondition
    Effects() As gameeffect
    Enabled As Boolean
End Type
```

Declarations - 5

```
Public Type MissionTarget
    id As String

    Visible As Boolean

    Position As D3DVECTOR
    WorldID As Integer
    radius As Single
    height As Single
End Type

Public Type TextureEx
    filename As String
    texture As Direct3DTexture8
End Type

Public Enum LightType
    Directional
    Target
    Omni
End Enum

Public Type myLight
    type As LightType
    Position As D3DVECTOR
    Direction As D3DVECTOR
    CastShadows As Boolean
    ShadowPoint As D3DVECTOR
    Range As Single
    Color As D3DCOLORVALUE
    Phi As Single           'outer cone
    Theta As Single         'inner cone   (theta<phi)
    WorldID As Long
End Type

'-------------- SAVING GAMES --------------
Public Type GameSlot
    MissionState1 As Long
    MissionState2 As Long
    MissionState3 As Long
    MissionState4 As Long

    Health As Long

    ObjectsID() As Long
    NumObjects As Long

    Position As D3DVECTOR   'position of the save point / char
    RotationH As Single     'best angle rotation for the save point
 '0 camera abaix, 270 camera a l'esquerra 180 dal 90 dreta

    WorldID As Integer      'which world is the char
End Type

'--------- MUSIC!!! -------

Public Declare Function FreeLibrary Lib "kernel32" (ByVal hModule As Long)
As Long
Public Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (By
```

```
Val lpLibFileName As String) As Long
Public Declare Function GetProcAddress Lib "kernel32" (ByVal hModule As Lon
g, ByVal lpProcName As String) As Long
Public Declare Function CallWindowProc Lib "user32" Alias "CallWindowProcA"
 (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal
 wParam As Long, ByVal lParam As Long) As Long
Public Declare Function IsBadReadPtr Lib "kernel32" (ptr As Any, ByVal ucb
As Long) As Long
Public Declare Function IsBadWritePtr Lib "kernel32" (ptr As Any, ByVal ucb
 As Long) As Long
Public Declare Sub CpyMem Lib "kernel32" Alias "RtlMoveMemory" (pDst As Any
, pSrc As Any, ByVal cBytes As Long)

Public Enum SND_RESULT
    SND_ERR_SUCCESS = 0
    SND_ERR_INVALID_SOURCE
    SND_ERR_INTERNAL
    SND_ERR_OUT_OF_RANGE
    SND_ERR_END_OF_STREAM
End Enum

Public Enum SND_SEEK_MODE
    SND_SEEK_PERCENT = 0
    SND_SEEK_SECONDS
End Enum

Public Type DynamicObject
    Position As D3DVECTOR

End Type

'--------- VIDEO ENGINE ----------
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA
" (ByVal lpstrCommand As String, ByVal lpstrReturnString As String, ByVal u
ReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function GetShortPathName Lib "kernel32" Alias "GetShortPath
NameA" (ByVal lpszLongPath As String, ByVal lpszShortPath As String, ByVal
cchBuffer As Long) As Long
```

GameLoop - 1

```
Option Explicit

'------- Game Loop Module -----------
'--- All the hard rendering here! ---

Public Sub GameLoop()
Dim WhatToClear As CONST_D3DCLEARFLAGS

'----------- CLEAN UP VARS ----------
FrameAverage = 0: TimeCounter = 0
AccelerationTimer = 0: Stopped = True
MakeFade2 = True: MakeFade1 = False

'---------- RENDER STATES BEFORE RENDERING ---------
Call SetCorrectRenderStates(RGB(WorldProperties(TheGameSlot.WorldID).Ambien
tValues.x, WorldProperties(TheGameSlot.WorldID).AmbientValues.y, WorldPrope
rties(TheGameSlot.WorldID).AmbientValues.z))

'------- SET up the corresponding lights --------
Call UpdateLights

'-------- SET UP PROJ MATRIX --------
D3DXMatrixPerspectiveFovLH projMatrix, 45 * Pi / 180, 1, 0.1, FarViewPlane
D3DXMatrixPerspectiveFovLH projMatrixShadows, 45 * Pi / 180, 1, 0.1, FarVie
wPlane - FarViewPlane / 200
Device.SetTransform D3DTS_PROJECTION, projMatrix

MainRender = True
Do While MainRender
    TimeCounter = GetTickCount()

    Call UpdateListenerSettings

    Call MusicEngine.RenderTime

    Call UpdateScene            'check the argument flow

    Call ComputeDoors           'be sure of the change of world

    '-------------------- INPUT --------------------
    ' - Mouse
        If MouseX > FrameAverage * 2.5 Then MouseX = FrameAverage * 2.5
        If MouseX < FrameAverage * -2.5 Then MouseX = FrameAverage * -2.5
        CharAngleH = CharAngleH + MouseX * CursorSpeed / 15
        If CharAngleH > 360 Then CharAngleH = CharAngleH - 360
        If CharAngleH < 0 Then CharAngleH = CharAngleH + 360

        CharAngleV = CharAngleV + MouseY * CursorSpeed / 25
        If CharAngleV > MaxVerticalAngle Then CharAngleV = MaxVerticalAngle
        If CharAngleV < MinVerticalAngle Then CharAngleV = MinVerticalAngle

        CharDistance = CharDistance + MouseZ / 2000 * CursorSpeed
        If CharDistance > MaxCameraDistance Then CharDistance = MaxCameraDi
stance
        If CharDistance < MinCameraDistance Then CharDistance = MinCameraDi
stance
        MouseX = 0: MouseY = 0: MouseZ = 0
    ' - Keyboard        --> configured in frmGraphics / ProcessCharMoves Su
b to increase speed
```

```
GameLoop - 2

    Call ProcessCharMoves

    Call PhysicsModule

    Call ProcessDynamicObjects

    '------------------- MATRIX VIEW ---------------
    D3DXMatrixLookAtLH viewMatrix, v3(CameraPos.x, CameraPos.y - 0.2, Camer
aPos.z), v3(CharPos.x, CharPos.y + 1, CharPos.z), vectorUP
    Device.SetTransform D3DTS_VIEW, viewMatrix
    Call ComputeClipPlanes

    '---------------- SOUND CONTROL -------------
    Call SoundCtrl

    '------------------- RENDER! -------------------
    WhatToClear = D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER
    If EnableShadows And ShadowsAvaliable Then WhatToClear = WhatToClear Or
 D3DCLEAR_STENCIL
    Call Device.Clear(0, ByVal 0, WhatToClear, 0, 1#, 0)
    Device.BeginScene

    Device.SetRenderState D3DRS_ALPHABLENDENABLE, 0
    Call DrawSkyBox

    Device.SetRenderState D3DRS_ALPHABLENDENABLE, 1

    Call DrawFixedObjects
    Call DrawFixedObjectsStage    'objects for this stage
    Call DrawMainChar
    Call RenderDynamicObjects

    Call DrawMainMesh

    If EnableShadows And ShadowsAvaliable Then Call DrawShadows

    Call DrawMissionTargets

    Call MakeFade       'controls the fading options, must be the last draw
ing
    Call SceneFading    'controls the level / XML fading

    Device.EndScene
    On Local Error Resume Next
    Call Device.Present(ByVal 0, ByVal 0, 0, ByVal 0)
    DoEvents
    '-----------------------------------------------

    TimeCounter = GetTickCount() - TimeCounter
    If FrameAverage = 0 Then FrameAverage = TimeCounter
    FrameAverage = (FrameAverage * 4 + TimeCounter) / 5

    If AuxMenu = True Then Call AuxMenuSystem

    If VideoOn <> "" Then
        DoEvents
        VideoEngine.OpenVideo VideoOn
        DoEvents
        VideoEngine.PlayVideo
        DoEvents
```

```
GameLoop - 3

        Do While VideoEngine.VideoStatus() = "playing"
            Call UpdateScene            'check the argument flow
            Sleep 5
            DoEvents
        Loop

        Call MouseSetUp

        VideoOn = ""
    End If
Loop

MusicEngine.StopMusic
End Sub

Public Sub GameLoadLevel()
Call SetLevelAtributes(TheGameSlot.MissionState1)
Call LoadAllSounds

'----------- Load Cam & Char Positions and coords --------------
CharAngleH = TheGameSlot.RotationH
CharAngleV = (MaxVerticalAngle + MinVerticalAngle) / 2
CharPos = TheGameSlot.Position
CharDistance = (MaxCameraDistance + MinCameraDistance) / 2

CameraPos = ProcessCameraCoords(CharPos, CharAngleH, CharAngleV, CharDistan
ce)

'-------------- SET UP THE SETTINGS INTO VARIABLES ----------
vectorUP = v3(0, 1, 0)
End Sub

Public Sub LoadAllSounds()
Dim mypath As String, x As Integer
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "sound_tmp_" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0

ExtractFile EXE & "sound\" & Trim(Str(TheGameSlot.WorldID)) & ".dat", mypat
h

For x = 1 To UBound(PendingSounds)
    SoundEngine.CreateSound mypath & PendingSounds(x), LCase(GetFileName(Pe
ndingSounds(x)))
Next

DeleteDir mypath

'---------------- MUSIC ------------------
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "music_" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
MusicDir = mypath
```

```
GameLoop - 4

ExtractFile EXE & "sound\music.dat", MusicDir
End Sub

Public Function MeshOfWorld(ByVal MeshID As Integer, ByVal WorldID As Integ
er) As Boolean
'---- MeshID ----
    '-- 01 - Circ + Ciutat   |
    '-- 02 - Arcs            |   WorldID1               --> Cull this two
    '-- 03 - Circ + Pulvinar |                          -->

    '-- 04 - Forum           |   WorldID 2

    '-- 05 - Pretori         |   WorldID 3

Select Case MeshID
Case 1, 2, 3
    If WorldID = 1 Then MeshOfWorld = True
Case 4
    If WorldID = 2 Then MeshOfWorld = True
Case 5
    If WorldID = 3 Then MeshOfWorld = True
End Select
End Function

Public Sub SetLights(ByVal WID As Integer)
Dim Light1 As D3DLIGHT8, Light2 As D3DLIGHT8, Light3 As D3DLIGHT8
Device.LightEnable 0, 0
Device.LightEnable 1, 0
Device.LightEnable 2, 0

Select Case WID
Case 1, 2
    Light1.type = D3DLIGHT_DIRECTIONAL
    Light1.Direction = Normalize(v3(-1, -1, -1))
    Light1.diffuse = ColorValue(1, 1, 1, 1)

    Device.SetLight 0, Light1
    Device.LightEnable 0, 1
Case 3

End Select
End Sub

Public Sub DrawSkyBox()
D3DXMatrixTranslation TMatrix, CameraPos.x, CameraPos.y - 0.22 - 0.05, Came
raPos.z
Device.SetTransform D3DTS_WORLD, TMatrix

Device.SetRenderState D3DRS_ZWRITEENABLE, 0
Device.SetRenderState D3DRS_LIGHTING, 0

Device.SetVertexShader myVertexFVFSimple
Device.SetTexture 0, SkyBoxTextures((WorldProperties(TheGameSlot.WorldID).S
kyBox - 1) * 4)
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, SkyBoxVertices(0), Len(SkyBo
xVertices(0))
Device.SetTexture 0, SkyBoxTextures((WorldProperties(TheGameSlot.WorldID).S
kyBox - 1) * 4 + 1)
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, SkyBoxVertices(4), Len(SkyBo
xVertices(0))
```

GameLoop - 5

```
Device.SetTexture 0, SkyBoxTextures((WorldProperties(TheGameSlot.WorldID).S
kyBox - 1) * 4 + 2)
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, SkyBoxVertices(8), Len(SkyBo
xVertices(0))
Device.SetTexture 0, SkyBoxTextures((WorldProperties(TheGameSlot.WorldID).S
kyBox - 1) * 4 + 3)
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, SkyBoxVertices(12), Len(SkyB
oxVertices(0))

Device.SetRenderState D3DRS_ZWRITEENABLE, 1
Device.SetRenderState D3DRS_LIGHTING, 1
End Sub

Public Sub DrawMainMesh()
D3DXMatrixIdentity TMatrix
Device.SetTransform D3DTS_WORLD, TMatrix
Select Case TheGameSlot.WorldID
Case 1
    RenderModel RenderModels(1)
    If CameraPos.z < 0 Then
        RenderModel RenderModels(2)
    Else
        RenderModel RenderModels(3)
    End If
Case 2
    RenderModel RenderModels(4)
Case 3
    RenderModel RenderModels(5)
End Select
End Sub

Public Sub RenderModel(model As Model3D)
Dim x As Integer
For x = 0 To model.NumMaterials - 1
    If model.TexturesNames(x) = "" Then
        Device.SetTexture 0, Nothing
    Else
        DeviceSetTexture model.TexturesNames(x)
    End If
    Device.SetMaterial model.Materials(x)
    model.Mesh.DrawSubset x
Next
End Sub

Public Sub RenderAnim(Anim As Anim3D, ByVal frame As Integer)
Dim x As Integer
For x = 0 To Anim.NumMaterials - 1
    If Anim.TexturesNames(x) = "" Then
        Device.SetTexture 0, Nothing
    Else
        DeviceSetTexture Anim.TexturesNames(x)
    End If
    Device.SetMaterial Anim.Materials(x)
    Anim.Meshes(frame).DrawSubset x
Next
End Sub

Public Sub DrawMainChar()
Dim mat As D3DMATRIX, Mat2 As D3DMATRIX
D3DXMatrixRotationY mat, CharAngleH / 180 * Pi
```

GameLoop - 6

```
D3DXMatrixTranslation Mat2, CharPos.x, CharPos.y, CharPos.z
D3DXMatrixMultiply mat, mat, Mat2
Device.SetTransform D3DTS_WORLD, mat

Select Case CurrentCharAnimation
Case "standing"
    RenderAnim MainCharStanding, Int(CurrentCharAnimationFrame)
Case "walking"
    RenderAnim MainCharWalking, Int(CurrentCharAnimationFrame - MainCharWal
kingStart.NumMeshes)
Case "walking_end"
    RenderAnim MainCharWalkingEnd, Int(CurrentCharAnimationFrame)
Case "walking_start"
    RenderAnim MainCharWalkingStart, Int(CurrentCharAnimationFrame)
End Select
End Sub


Public Sub PhysicsModule()
Dim vertex1 As D3DVECTOR, Fallen As Boolean
Static VSpeed As Single
vertex1.x = CharPos.x
vertex1.y = CharPos.y + 1
vertex1.z = CharPos.z

Dim res As salida, temp As Double, NextY As Double

hprocess CollisionFloats(TheGameSlot.WorldID).vertices(0), vertex1, (UBound
(CollisionFloats(TheGameSlot.WorldID).vertices) / 3), res

If AccelerationTimer <> 0 Then
    temp = ((GetTickCount() - AccelerationTimer) / 1000)
    NextY = InitY + (VSpeed - 7) * temp + 0.5 * -WorldAcceleration * temp ^
 2

    Fallen = True

    If res.respuesta = 1 And (NextY <= CharPos.y) Then
    If res.puntocolision.y >= NextY Then
        temp = (res.puntocolision.y - CharPos.y) * 25 * FrameAverage / 1000
        If temp < 0.01 Then temp = (res.puntocolision.y - CharPos.y)
        CharPos.y = CharPos.y + temp
        Fallen = False
    Else
        If NextY > res.puntocolision.y Then
            If (NextY - res.puntocolision.y) < 0.1 And VSpeed = 0 Then
                CharPos.y = res.puntocolision.y
                Fallen = False
            Else
                CharPos.y = NextY
            End If
        Else
            CharPos.y = res.puntocolision.y
            Fallen = False
        End If
    End If
    Else
        CharPos.y = NextY
    End If
End If
```

GameLoop - 7

```
If Fallen = False Then
    VSpeed = 0
    InitY = CharPos.y
    AccelerationTimer = GetTickCount()
    If Jumping = True Then VSpeed = 11
Else
    Jumping = False
End If

CharPosBefore = CharPos

If MouseB0 = True Then
    process CameraPos, CollisionFloats(TheGameSlot.WorldID).vertices(0), (U
Bound(CollisionFloats(TheGameSlot.WorldID).vertices) / 3), CharPos, CharPos
Before, vectorUP, CharDistance, CharAngleV, CharAngleH, 0.5, 5, 0.7, 1.2, F
rameAverage, CharSpeed * -1 * 10, 0.2, CollisionFloatsAux(0)
Else
    process CameraPos, CollisionFloats(TheGameSlot.WorldID).vertices(0), (U
Bound(CollisionFloats(TheGameSlot.WorldID).vertices) / 3), CharPos, CharPos
Before, vectorUP, CharDistance, CharAngleV, CharAngleH, 0.5, 5, 0.7, 1.2, F
rameAverage, CharSpeed * -1 * 1, 0.2, CollisionFloatsAux(0)
End If
End Sub

Public Sub ProcessCharMoves()
Static LastState As Integer, LoopDone As Boolean

ReEnter:
If ResetAllMoves Then
    LastState = 0: LoopDone = False
    EndingAnim = False
    ResetAllMoves = False
    Movement = 0
End If

If Movement = 0 Then
    If EndingAnim = False Then
        If LastState = 1 Then
            'ready to brake
            EndingAnim = True
            CharState = 0
            CharSpeed = 1
            LoopDone = False
            GoTo ReEnter
        Else
            CurrentCharAnimation = "standing"
            CurrentCharAnimationFrame = 1
            CharState = 0
            CharSpeed = 0
        End If
    Else
        If LoopDone = False Then
            CurrentCharAnimationFrame = CurrentCharAnimationFrame + (MainCh
arWalking.NumMeshes + MainCharWalkingStart.NumMeshes) / (MainCharWalking.Du
rationMS + MainCharWalkingStart.DurationMS) * FrameAverage
            If CurrentCharAnimationFrame < 1 Then CurrentCharAnimationFrame
 = 1

            If CurrentCharAnimationFrame > (MainCharWalking.NumMeshes + Mai
```

```
nCharWalkingStart.NumMeshes) Then
                LoopDone = True
                GoTo ReEnter
            End If

            If CurrentCharAnimationFrame < MainCharWalkingStart.NumMeshes T
hen
                CurrentCharAnimation = "walking_start"
            Else
                If CurrentCharAnimation = "walking_start" Then
                    CurrentCharAnimationFrame = 0
                    LoopDone = True
                    GoTo ReEnter
                End If
                CurrentCharAnimation = "walking"
            End If
        Else
            CurrentCharAnimationFrame = CurrentCharAnimationFrame + MainCha
rWalkingEnd.NumMeshes / MainCharWalkingEnd.DurationMS * FrameAverage
            If CurrentCharAnimationFrame < 1 Then CurrentCharAnimationFrame
 = 1
            If CurrentCharAnimationFrame > MainCharWalkingEnd.NumMeshes The
n
                EndingAnim = False
                CharState = 0
                LastState = 0
                GoTo ReEnter
            End If
            CurrentCharAnimation = "walking_end"
        End If
        CharState = 0
        CharSpeed = 1
    End If
Else
    CharState = 1
    CharSpeed = 1

    CurrentCharAnimationFrame = CurrentCharAnimationFrame + (MainCharWalkin
g.NumMeshes + MainCharWalkingStart.NumMeshes) / (MainCharWalking.DurationMS
 + MainCharWalkingStart.DurationMS) * FrameAverage

    If CurrentCharAnimationFrame < 1 Then CurrentCharAnimationFrame = 1
    If CurrentCharAnimationFrame > (MainCharWalking.NumMeshes + MainCharWal
kingStart.NumMeshes) Then CurrentCharAnimationFrame = MainCharWalkingStart.
NumMeshes

    If Int(CurrentCharAnimationFrame) <= MainCharWalkingStart.NumMeshes The
n
        CurrentCharAnimation = "walking_start"
    Else
        CurrentCharAnimation = "walking"
    End If
End If
LastState = CharState
End Sub

Public Sub MakeFade()
'makes a fade between the worlds (see computedoors sub)
Static mTimer As Double
Dim alpha As Single
```

GameLoop - 9

```
InitAgain:

If MakeFade1 Then
    If mTimer = 0 Then mTimer = GetTickCount()
    If mTimer + 750 < GetTickCount() Then
        mTimer = 0
        EnterNewWorld 0, 0
        GoTo InitAgain
    End If

    alpha = 255 * (GetTickCount() - mTimer) / 750
    If alpha > 255 Then alpha = 255
    If alpha < 0 Then alpha = 0
    fadeVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(alpha, 0, 0, 0))
    fadeVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(alpha, 0, 0,
 0))
    fadeVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(alpha, 0, 0
, 0))
    fadeVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColorARGB(al
pha, 0, 0, 0))

    Device.SetTexture 0, Nothing
    Device.SetVertexShader myVertexAlphaFVF
    Device.SetRenderState D3DRS_ZENABLE, 0
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, fadeVertices(0), Len(fad
eVertices(0))
    Device.SetRenderState D3DRS_ZENABLE, 1
ElseIf MakeFade2 Then
    If mTimer = 0 Then mTimer = GetTickCount()
    If mTimer + 750 < GetTickCount() Then
        MakeFade2 = False
        MakeFade1 = False
        mTimer = 0
        Exit Sub
    End If

    alpha = 255 - 255 * (GetTickCount() - mTimer) / 750
    If alpha > 255 Then alpha = 255
    If alpha < 0 Then alpha = 0
    fadeVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(alpha, 0, 0, 0))
    fadeVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(alpha, 0, 0,
 0))
    fadeVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(alpha, 0, 0
, 0))
    fadeVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColorARGB(al
pha, 0, 0, 0))

    Device.SetTexture 0, Nothing
    Device.SetVertexShader myVertexAlphaFVF
    Device.SetRenderState D3DRS_ZENABLE, 0
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, fadeVertices(0), Len(fad
eVertices(0))
    Device.SetRenderState D3DRS_ZENABLE, 1
Else
    mTimer = 0
End If
End Sub

Public Sub SceneFading()
```

```
GameLoop - 10

Static FTimer As Double
Dim alpha As Single

SceneFading_Init:

Select Case FadeState
Case 0
    'no fade
    FTimer = 0
    Exit Sub
Case 1
    'fade in
    If FTimer = 0 Then FTimer = GetTickCount()
    If FTimer + FadeTimeMS < GetTickCount() Then
        FadeState = 0
        FTimer = 0
        GoTo SceneFading_Init
    End If
    alpha = 255 - 255 * (GetTickCount() - FTimer) / FadeTimeMS
Case 2
    'fade out
    If FTimer = 0 Then FTimer = GetTickCount()
    If FTimer + FadeTimeMS < GetTickCount() Then
        FadeState = 3   'black
        FTimer = 0
        GoTo SceneFading_Init
    End If
    alpha = 255 * (GetTickCount() - FTimer) / FadeTimeMS
Case 3
    alpha = 255         'opac
    FTimer = 0
End Select

If alpha > 255 Then alpha = 255
If alpha < 0 Then alpha = 0
fadeVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(alpha, 0, 0, 0))
fadeVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(alpha, 0, 0, 0))
fadeVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(alpha, 0, 0, 0)
)
fadeVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColorARGB(alpha,
 0, 0, 0))

Device.SetTexture 0, Nothing
Device.SetVertexShader myVertexAlphaFVF
Device.SetRenderState D3DRS_ZENABLE, 0
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, fadeVertices(0), Len(fadeVer
tices(0))
Device.SetRenderState D3DRS_ZENABLE, 1
End Sub

Public Sub SoundCtrl()
Select Case CurrentCharAnimation
Case "standing"
    If IsPlayingStSound("pasos_mainchar") Then StopStandardSound "pasos_mai
nchar"
Case "walking"

Case "walking_end"

Case "walking_start"
```

```
    PlayStandardSound "pasos_mainchar", True, False
End Select
End Sub

Public Sub DrawMissionTargets()
If UBound(MissionTargets) = 0 Then Exit Sub
Dim x As Long, y As Long
Dim SMatrix As D3DMATRIX, TMatrix As D3DMATRIX, RMatrix As D3DMATRIX, TheMa
trix As D3DMATRIX

Device.SetRenderState D3DRS_ZWRITEENABLE, 0
Device.SetRenderState D3DRS_LIGHTING, 0

For x = 1 To UBound(MissionTargets)
    If MissionTargets(x).Visible Then
        D3DXMatrixScaling SMatrix, MissionTargets(x).radius + Cos(GetTickCo
unt() / 250) / 20, MissionTargets(x).height + sin(GetTickCount() / 250) / 2
0, MissionTargets(x).radius + Cos(GetTickCount() / 250) / 20
        D3DXMatrixRotationY RMatrix, GetTickCount() / 1000
        D3DXMatrixTranslation TMatrix, MissionTargets(x).Position.x, Missio
nTargets(x).Position.y, MissionTargets(x).Position.z
        D3DXMatrixMultiply TheMatrix, SMatrix, RMatrix
        D3DXMatrixMultiply TheMatrix, TheMatrix, TMatrix
        Device.SetTransform D3DTS_WORLD, TheMatrix
        For y = 0 To MissionTargetModel.NumMaterials - 1
            Device.SetMaterial MissionTargetModel.Materials(y)
            DeviceSetTexture MissionTargetModel.TexturesNames(y)
            Call MissionTargetModel.Mesh.DrawSubset(y)
        Next
    End If
Next


Device.SetRenderState D3DRS_ZWRITEENABLE, 1
Device.SetRenderState D3DRS_LIGHTING, 1
End Sub

Public Sub DrawShadows()
Dim light As D3DVECTOR, Inv As D3DMATRIX
Dim AbsPosition As D3DVECTOR, XRatio As Single, ZRatio As Single, Ratio As
Long
light = v3(0, 1, 2)

Dim mat As D3DMATRIX, Mat2 As D3DMATRIX, mati As D3DMATRIX
D3DXMatrixIdentity mati
D3DXMatrixRotationY mat, CharAngleH / 180 * Pi
D3DXMatrixTranslation Mat2, CharPos.x, CharPos.y, CharPos.z
D3DXMatrixMultiply mat, mat, Mat2

D3DXMatrixInverse Inv, 0, mat
D3DXVec3TransformNormal light, light, Inv

Static a As Boolean

'MainCharStandingShadowClass.LightProj = 10
'MainCharStandingShadowClass.Build MainCharStandingShadowVerts, light, Main
CharStandingShadowNormals

Device.SetRenderState D3DRS_ZWRITEENABLE, False
Device.SetRenderState D3DRS_STENCILENABLE, 1
```

GameLoop - 12

```
Device.SetRenderState D3DRS_SHADEMODE, D3DSHADE_FLAT
Device.SetRenderState D3DRS_LIGHTING, 0

Device.SetRenderState D3DRS_STENCILFUNC, D3DCMP_ALWAYS
Device.SetRenderState D3DRS_STENCILZFAIL, D3DSTENCILOP_INCR
Device.SetRenderState D3DRS_STENCILFAIL, D3DSTENCILOP_KEEP
Device.SetRenderState D3DRS_STENCILPASS, D3DSTENCILOP_KEEP

Device.SetRenderState D3DRS_STENCILREF, &H1
Device.SetRenderState D3DRS_STENCILMASK, &HFFFFFFFF
Device.SetRenderState D3DRS_STENCILWRITEMASK, &HFFFFFFFF

Device.SetRenderState D3DRS_SRCBLEND, D3DBLEND_ZERO
Device.SetRenderState D3DRS_DESTBLEND, D3DBLEND_ONE
Device.SetTransform D3DTS_PROJECTION, projMatrixShadows
Device.SetTexture 0, Nothing
Device.SetVertexShader D3DFVF_XYZ

Device.SetRenderState D3DRS_CULLMODE, D3DCULL_CW

Device.SetTransform D3DTS_WORLD, mati
AbsPosition.x = CharPos.x - ShadowedWorlds(TheGameSlot.WorldID).MinWidth
AbsPosition.z = CharPos.z - ShadowedWorlds(TheGameSlot.WorldID).MinHeight
XRatio = Int(AbsPosition.x / (ShadowCullSize * 2)) + 1
ZRatio = Int(AbsPosition.z / (ShadowCullSize * 2)) + 1
Ratio = ZRatio + (XRatio - 1) * ShadowedWorlds(TheGameSlot.WorldID).HeightB
locks
On Local Error Resume Next
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio).RenderGeometry
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio - 1).RenderGeometry
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio + 1).RenderGeometry
On Local Error GoTo 0

Device.SetRenderState D3DRS_CULLMODE, D3DCULL_CCW
Device.SetRenderState D3DRS_STENCILZFAIL, D3DSTENCILOP_DECR

Device.SetTransform D3DTS_WORLD, mati
On Local Error Resume Next
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio).RenderGeometry
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio - 1).RenderGeometry
WorldShadows(TheGameSlot.WorldID).ShadowsArray(Ratio + 1).RenderGeometry
On Local Error GoTo 0

Device.SetRenderState D3DRS_SHADEMODE, D3DSHADE_GOURAUD
Device.SetRenderState D3DRS_ZWRITEENABLE, True

Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_SRCBLEND, D3DBLEND_SRCALPHA
Device.SetRenderState D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA

Device.SetRenderState D3DRS_STENCILREF, &H1
Device.SetRenderState D3DRS_STENCILFUNC, D3DCMP_LESSEQUAL
Device.SetRenderState D3DRS_STENCILPASS, D3DSTENCILOP_KEEP

Device.SetTransform D3DTS_PROJECTION, projMatrix

Device.SetVertexShader myVertexAlphaFVF
ShadowVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(128, 0, 0, 0))
ShadowVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(128, 0, 0, 0))
ShadowVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(128, 0, 0, 0)
```

```
)
ShadowVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColorARGB(128,
 0, 0, 0))
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, ShadowVertices(0), Len(Shado
wVertices(0))

Device.SetRenderState D3DRS_ZENABLE, 1
Device.SetRenderState D3DRS_STENCILENABLE, 0
Device.SetRenderState D3DRS_LIGHTING, 1
End Sub

Public Sub ProcessDynamicObjects()
Dim x As Long, y As Long
Dim Directions() As D3DVECTOR
Dim Salidas() As salida

ReDim DynObjPositions(0)
NumDynObjPositions = -1

For x = 1 To UBound(MovingCharacters)
    If Not (MovingCharacters(x) Is Nothing) And MovingCharacters(x).Visible
 And MovingCharacters(x).Paused = False Then Call MovingCharacters(x).Rende
rTime
Next

If NumDynObjPositions = -1 Then GoTo Skip
ReDim Directions(NumDynObjPositions)
ReDim Salidas(NumDynObjPositions)
For x = 0 To NumDynObjPositions
    Directions(x).y = -1
Next
segintersectfast DynObjPositions(0), Directions(0), CollisionFloats(TheGame
Slot.WorldID).vertices(0), UBound(CollisionFloats(TheGameSlot.WorldID).vert
ices) / 3, NumDynObjPositions + 1, Salidas(0)

Skip:

For x = 1 To UBound(MovingCharacters)
    If Not (MovingCharacters(x) Is Nothing) And MovingCharacters(x).Visible
 And MovingCharacters(x).Paused = False Then
        If MovingCharacters(x).AutoY = False Then
            Call MovingCharacters(x).RenderTime2
        Else
            MovingCharacters(x).CoordY = Salidas(y).puntocolision.y
            Call MovingCharacters(x).RenderTime2
            y = y + 1
        End If
    End If
    If Not (MovingCharacters(x) Is Nothing) And MovingCharacters(x).Visible
 Then
        'check if paused too
        Call MovingCharacters(x).RenderSound
    End If
Next
End Sub

Public Sub RenderDynamicObjects()
Dim x As Long
For x = 1 To UBound(MovingCharacters)
    If Not (MovingCharacters(x) Is Nothing) Then Call MovingCharacters(x).R
```

```
enderNow
Next
End Sub

Public Sub AuxMenuSystem()
'the menu system which appears when pressing escape
Dim coordX As Single, CoordY As Single
Dim Out As Boolean
Device.SetRenderState D3DRS_ALPHABLENDENABLE, True
Device.SetVertexShader myVertexFVF
Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_SRCBLEND, D3DBLEND_SRCALPHA
Device.SetRenderState D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA
Device.SetRenderState D3DRS_LIGHTING, 0

Call FreezeAll

Do While Out = False
    '----------- MOUSE COORDS PROCESS ---------------
    coordX = coordX + MouseX * (CursorSpeed / 3)
    CoordY = CoordY + MouseY * (CursorSpeed / 3)
    If coordX < 0 Then coordX = 0
    If coordX > D3DM.width Then coordX = D3DM.width
    If CoordY < 0 Then CoordY = 0
    If CoordY > D3DM.height Then CoordY = D3DM.height
    MouseVerts(0) = AssignMV(coordX, CoordY, 0, 0, 0)
    MouseVerts(1) = AssignMV(coordX + CursorW, CoordY, 0, 1, 0)
    MouseVerts(2) = AssignMV(coordX, CoordY + CursorH, 0, 0, 1)
    MouseVerts(3) = AssignMV(coordX + CursorW, CoordY + CursorH, 0, 1, 1)
    MouseX = 0: MouseY = 0
    '---------------- SEND EVENTS TO MENUS ------------
    AuxiliarMenu.ProcessMouseMove coordX, CoordY
    If MouseClick0 = True Then
        AuxiliarMenu.ProcessClick coordX, CoordY
        MouseClick0 = False
    End If

    If AuxiliarMenu.isEvent Then
        Select Case AuxiliarMenu.EventName
        Case "returngame"
            Out = True
        Case "returnmenu"
            Out = True
            MainRender = False
        End Select
    End If
    AuxiliarMenu.isEvent = False

    Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
    Device.BeginScene

    '------------ RENDER THE MENU --------
    AuxiliarMenu.RenderMenu

    '----------- RENDER THE CURSOR OVER ALL -----------
    Device.SetTexture 0, MouseTexture
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, MouseVerts(0), Len(Mouse
Verts(0))

    Device.EndScene
```

GameLoop - 15

```
    Device.Present ByVal 0, ByVal 0, 0, ByVal 0

    Sleep 5
    DoEvents
Loop

If MainRender = True Then
    'only return to the game
    Call SetCorrectRenderStates(RGB(WorldProperties(TheGameSlot.WorldID).Am
bientValues.x, WorldProperties(TheGameSlot.WorldID).AmbientValues.y, WorldP
roperties(TheGameSlot.WorldID).AmbientValues.z))

    Call UnFreezeAll
Else
    'exit to the menu

End If
AuxMenu = False
End Sub

Public Sub FreezeAll()
Dim x As Long
For x = 1 To UBound(MovingCharacters)
    If Not (MovingCharacters(x) Is Nothing) Then
        MovingCharacters(x).FreezeTimer
    End If
Next
End Sub

Public Sub UnFreezeAll()
Dim x As Long
For x = 1 To UBound(MovingCharacters)
    If Not (MovingCharacters(x) Is Nothing) Then
        MovingCharacters(x).UnFreezeTimer
    End If
Next
End Sub
```

GameScenes - 1

```
Option Explicit

Public Sub UpdateScene()
'check for objectives or changes in the missions / world
'es comproven les condicions i s'executen les accions corresponents
'escrit en XML a stages/
Dim x As Long, y As Long, Ok As Boolean, z As Long
For x = 1 To UBound(LevelScenes)
    If LevelScenes(x).Enabled Then
        Ok = True
        For y = 1 To UBound(LevelScenes(x).Conditions)
            'check for the condition
            Select Case LevelScenes(x).Conditions(y).type
            Case "isinarea"
                If Not (CharPos.x > LevelScenes(x).Conditions(y).varg.x And
 CharPos.x < LevelScenes(x).Conditions(y).varg2.x And _
                        CharPos.z > LevelScenes(x).Conditions(y).varg.z And
 CharPos.z < LevelScenes(x).Conditions(y).varg2.z) Then
                    Ok = False
                End If
            Case "isintarget"
                For z = 1 To UBound(MissionTargets)
                    If MissionTargets(z).id = LevelScenes(x).Conditions(y).
sarg Then
                        If Not VecDistFast(CharPos, MissionTargets(z).Posit
ion) < MissionTargets(z).radius ^ 2 Then
                            Ok = False
                        End If
                        Exit For
                    End If
                Next
            Case "fadestate"
                If LevelScenes(x).Conditions(y).sarg = "clear" Then
                    If FadeState <> 0 Then Ok = False
                Else    'black
                    If FadeState <> 3 Then Ok = False
                End If
            Case "videofinished"
                If VideoEngine.VideoStatus <> "stopped" Then
                    Ok = False
                End If
            End Select
        Next
        If Ok = True Then
            'conditions ok, so execute effects
            For y = 1 To UBound(LevelScenes(x).Effects)
                Select Case LevelScenes(x).Effects(y).type
                Case "setcharpos"
                    CharPos = LevelScenes(x).Effects(y).varg
                    ResetAllMoves = True
                    InitY = CharPos.y
                    Movement = 0
                Case "disablescene"
                    LevelScenes(LevelScenes(x).Effects(y).larg).Enabled = F
alse
                Case "enablescene"
                    LevelScenes(LevelScenes(x).Effects(y).larg).Enabled = T
rue
                Case "enableobject"
                    For z = 1 To UBound(MissionTargets)
```

GameScenes - 2

```
                        If MissionTargets(z).id = LevelScenes(x).Effects(y)
.sarg Then
                            MissionTargets(z).Visible = True
                            Exit For
                        End If
                    Next
                    For z = 1 To UBound(MovingCharacters)
                        If MovingCharacters(z).id = LevelScenes(x).Effects(
y).sarg Then
                            MovingCharacters(z).Visible = True
                            MovingCharacters(z).SetPos 0
                            Exit For
                        End If
                    Next
                Case "disableobject"
                    For z = 1 To UBound(MissionTargets)
                        If MissionTargets(z).id = LevelScenes(x).Effects(y)
.sarg Then
                            MissionTargets(z).Visible = False
                            Exit For
                        End If
                    Next
                    For z = 1 To UBound(MovingCharacters)
                        If MovingCharacters(z).id = LevelScenes(x).Effects(
y).sarg Then
                            MovingCharacters(z).Visible = False
                            Exit For
                        End If
                    Next
                Case "doors"
                    If LevelScenes(x).Effects(y).larg = 0 Then
                        DisableDoors = True
                    Else
                        DisableDoors = False
                    End If
                Case "setfade"
                    If LevelScenes(x).Effects(y).sarg = "in" Then
                        FadeState = 1
                    ElseIf LevelScenes(x).Effects(y).sarg = "out" Then
                        FadeState = 2
                    Else
                        FadeState = 3
                    End If
                Case "setfadetime"
                    FadeTimeMS = LevelScenes(x).Effects(y).larg   'sets the
 duration of a fade
                Case "setworld"
                    TheGameSlot.WorldID = LevelScenes(x).Effects(y).larg
                    Call SetCorrectRenderStates(RGB(WorldProperties(TheGame
Slot.WorldID).AmbientValues.x, WorldProperties(TheGameSlot.WorldID).Ambient
Values.y, WorldProperties(TheGameSlot.WorldID).AmbientValues.z))
                    Call UpdateLights
                Case "setcamera"
                    'reposiciona la càmera donat l'angle horitzontal
                    CameraPos = ProcessCameraCoords(CharPos, LevelScenes(x)
.Effects(y).larg, (MaxVerticalAngle + MinVerticalAngle) / 2, (MaxCameraDist
ance + MinCameraDistance) / 2)
                Case "playvideo"
                    VideoOn = EXE & "video\" & LevelScenes(x).Effects(y).sa
rg
```

GameScenes - 3

```
                    'VideoEngine.OpenVideo EXE & "video\" & LevelScenes(x).
Effects(y).sarg
                    'VideoEngine.PlayVideo
                Case "endvideo"
                    VideoEngine.StopVideo
                    VideoEngine.CloseVideo
                    ResetDevice D3DM, AntiAliasLevel
                End Select
            Next
        End If
    End If
Next
End Sub

Public Sub RenderScene()

End Sub

Public Sub ResetCurrentLevel()
On Local Error Resume Next
Dim x As Long
LevelScenes(1).Enabled = True
For x = 2 To UBound(LevelScenes)
    LevelScenes(x).Enabled = False
Next
For x = 1 To UBound(MissionTargets)
    MissionTargets(x).Visible = False
Next
For x = 1 To UBound(MovingCharacters)
    MovingCharacters(x).SetPos 0
    MovingCharacters(x).Visible = False
Next
End Sub

Public Sub SetLevelAtributes(ByVal Level As Long)
'load the current level
Dim mypath As String, CharPath As String
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
CharPath = mypath & "chars_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
mypath = mypath & "stages_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath:  MkDir CharPath
On Local Error GoTo 0
LoadingStageScreen 0
ExtractFile EXE & "stages\stage" & Trim(Str(Level)) & ".dat", mypath
ExtractFile EXE & "graphics\otherchars.dat", CharPath
LoadingStageScreen 20

ReDim FixedObjectsStage(0)
LoadParseExecuteXML mypath & "stage" & Trim(Str(Level)) & ".dat", mypath, C
harPath, 20, 50

Call ResetCurrentLevel
Call LoadWorldShadows2

DeleteDir mypath: DeleteDir CharPath
End Sub

Public Sub LoadParseExecuteXML(ByVal file As String, ByVal ResPath As Strin
```

```
g, ByVal CharPath As String, ByVal barmin As Integer, ByVal barmax As Integ
er)
Dim ff As Integer, cad As String, cad2 As String
Dim lines() As String, x As Long, y As Long, z As Long
Dim R1 As Long, R2 As Long

Dim Position As D3DVECTOR, RotationY As Single
Dim Mesh As String, Meshsimple As String, AutoY As Boolean
Dim WorldID As Integer

ff = FreeFile()

Open file For Binary As #ff
    cad = Space(LOF(ff))
    Get #ff, , cad
    lines = Split(cad, vbCrLf)
Close #ff

Call DestroyThisScene

Do While x <= UBound(lines)
    If left(lines(x), 1) = "<" Then
        Position = v3(0, 0, 0): RotationY = 0: Mesh = "": Meshsimple = "":
AutoY = False
        Select Case lines(x)
        Case "<ambient>"
            WorldProperties(WorldID).AmbientValues = ParsePos(lines(x + 1))
            x = x + 2
        Case "<night>"
            WorldProperties(WorldID).RenderSky = True
            WorldProperties(WorldID).SkyBox = 3
        Case "<morning>"
            WorldProperties(WorldID).RenderSky = True
            WorldProperties(WorldID).SkyBox = 1
        Case "<afternoon>"
            WorldProperties(WorldID).RenderSky = True
            WorldProperties(WorldID).SkyBox = 2
        Case "<nosky>"
            WorldProperties(WorldID).RenderSky = False
            WorldProperties(WorldID).SkyBox = 0
        Case "<target>"
            TargetFromString lines(x + 1), WorldID
            x = x + 2
        Case "<fixedobject>"
            ReDim Preserve FixedObjectsStage(UBound(FixedObjectsStage) + 1)
            Do While lines(x) <> "</fixedobject>"
                x = x + 1
                cad = lines(x)
                R1 = InStr(1, cad, "=")
                cad2 = mid(cad, R1 + 1)
                If R1 <> 0 Then
                Select Case LCase(left(cad, R1 - 1))
                Case "pos"
                    Position = ParsePos(cad2)
                Case "rotationy"
                    RotationY = Val(cad2)
                Case "mesh"
                    Mesh = cad2
                Case "meshsimple"
                    Meshsimple = cad2
```

```
                Case "autoy"
                    If Val(cad) <> 0 Then AutoY = True
                End Select
                End If
            Loop
            FixedObjectsStage(UBound(FixedObjectsStage)) = LoadFixedObjectS
tage(Position, ResPath & Mesh, ResPath & Meshsimple, AutoY, RotationY, Worl
dID)
        Case "<char>"
            y = UBound(MovingCharacters) + 1
            ReDim Preserve MovingCharacters(y)
            Set MovingCharacters(y) = New clsDynObj
            MovingCharacters(y).WorldID = WorldID
            Do While lines(x) <> "</char>"
                x = x + 1
                cad = lines(x)
                R1 = InStr(1, cad, "=")
                cad2 = mid(cad, R1 + 1)
                If R1 <> 0 Then
                Select Case LCase(left(cad, R1 - 1))
                Case "quiet"
                    MovingCharacters(y).SetAnimation Quiet, CharPath, cad2
                Case "moving"
                    MovingCharacters(y).SetAnimation Moving, CharPath, cad2
                Case "startmoving"
                    MovingCharacters(y).SetAnimation StartMoving, CharPath,
 cad2
                Case "endmoving"
                    MovingCharacters(y).SetAnimation EndMoving, CharPath, c
ad2
                Case "y"
                    If cad2 = "true" Then
                        MovingCharacters(y).AutoY = True
                    ElseIf cad2 = "false" Then
                        MovingCharacters(y).AutoY = False
                    Else
                        MovingCharacters(y).AutoY = False
                        MovingCharacters(y).CoordY = Val(cad2)
                    End If
                Case "addpoint"
                    AddPointToObject MovingCharacters(y), cad2
                Case "id"
                    MovingCharacters(y).id = cad2
                Case "sound"
                    MovingCharacters(y).SetSpeak cad2
                Case "radius"
                    MovingCharacters(y).StopRadius = Val(cad2)
                End Select
                End If
            Loop
            MovingCharacters(y).SetPos 0
            MovingCharacters(y).SetLimit -1
        Case "<light>"
            y = UBound(Lights) + 1
            ReDim Preserve Lights(y)
            Lights(y).WorldID = WorldID
            Do While lines(x) <> "</light>"
                x = x + 1
                cad = lines(x)
                R1 = InStr(1, cad, "=")
```

```
                cad2 = mid(cad, R1 + 1)
                If R1 <> 0 Then
                Select Case LCase(left(cad, R1 - 1))
                Case "type"
                    Select Case cad2
                    Case "omni"
                        Lights(y).type = Omni
                    Case "target"
                        Lights(y).type = Target
                    Case "directional"
                        Lights(y).type = Directional
                    End Select
                Case "pos"
                    Lights(y).Position = ParsePos(cad2)
                Case "dir"
                    Lights(y).Direction = ParsePos(cad2)
                Case "icone"
                    Lights(y).Theta = Val(cad2)
                Case "ocone"
                    Lights(y).Phi = Val(cad2)
                Case "range"
                    Lights(y).Range = Val(cad2)
                Case "shadowpos"
                    Lights(y).ShadowPoint = ParsePos(cad2)
                    Lights(y).CastShadows = True
                Case "color"
                    Lights(y).Color = ParseColor(cad2)
                End Select
                End If
            Loop
        Case "<sounds>"
            Do While lines(x) <> "</sounds>"
                x = x + 1
                If lines(x) <> "" And lines(x) <> "</sounds>" Then
                    y = UBound(PendingSounds) + 1
                    ReDim Preserve PendingSounds(y)
                    PendingSounds(y) = lines(x)
                End If
            Loop
        Case Else
            If left(lines(x), Len("<world")) = "<world" Then
                WorldID = Val(mid(lines(x), Len("<world=x")))
            ElseIf left(lines(x), Len("<scene")) = "<scene" Then
                y = UBound(LevelScenes) + 1
                ReDim Preserve LevelScenes(y)
                ReDim LevelScenes(y).Conditions(0)
                ReDim LevelScenes(y).Effects(0)
                If y = 1 Then LevelScenes(y).Enabled = True
                Do While lines(x) <> "</scene>"
                    x = x + 1
                    cad = lines(x)
                    R1 = InStr(1, cad, "=")
                    cad2 = mid(cad, R1 + 1)
                    If R1 <> 0 Then
                    Select Case LCase(left(cad, R1 - 1))
                    Case "isinarea"
                        z = UBound(LevelScenes(y).Conditions) + 1
                        ReDim Preserve LevelScenes(y).Conditions(z)
                        ParseArea cad2, LevelScenes(y).Conditions(z).varg,
LevelScenes(y).Conditions(z).varg2
```

```
                        LevelScenes(y).Conditions(z).type = LCase(left(cad,
 R1 - 1))
                    Case "isintarget", "videofinished"
                        z = UBound(LevelScenes(y).Conditions) + 1
                        ReDim Preserve LevelScenes(y).Conditions(z)
                        LevelScenes(y).Conditions(z).sarg = cad2
                        LevelScenes(y).Conditions(z).type = LCase(left(cad,
 R1 - 1))
                    Case "fadestate"
                        z = UBound(LevelScenes(y).Conditions) + 1
                        ReDim Preserve LevelScenes(y).Conditions(z)
                        LevelScenes(y).Conditions(z).sarg = cad2
                        LevelScenes(y).Conditions(z).type = LCase(left(cad,
 R1 - 1))
                    Case "disablescene", "enablescene", "setworld", "setcam
era", "doors", "setfadetime"
                        z = UBound(LevelScenes(y).Effects) + 1
                        ReDim Preserve LevelScenes(y).Effects(z)
                        LevelScenes(y).Effects(z).larg = Val(cad2)
                        LevelScenes(y).Effects(z).type = LCase(left(cad, R1
 - 1))
                    Case "setcharpos"
                        z = UBound(LevelScenes(y).Effects) + 1
                        ReDim Preserve LevelScenes(y).Effects(z)
                        LevelScenes(y).Effects(z).varg = ParsePos(cad2)
                        LevelScenes(y).Effects(z).type = LCase(left(cad, R1
 - 1))
                    Case "enableobject", "disableobject", "playvideo", "end
video"
                        z = UBound(LevelScenes(y).Effects) + 1
                        ReDim Preserve LevelScenes(y).Effects(z)
                        LevelScenes(y).Effects(z).sarg = cad2
                        LevelScenes(y).Effects(z).type = LCase(left(cad, R1
 - 1))
                    Case "setfade"
                        z = UBound(LevelScenes(y).Effects) + 1
                        ReDim Preserve LevelScenes(y).Effects(z)
                        LevelScenes(y).Effects(z).sarg = cad2
                        LevelScenes(y).Effects(z).type = LCase(left(cad, R1
 - 1))
                    End Select
                    End If
                Loop
            End If
        End Select
    End If
    LoadingStageScreen barmin + x * (barmax - barmin) / UBound(lines)
    x = x + 1
Loop

Call LoadPendingTexturesSec(0, 100)
End Sub

Public Sub TargetFromString(ByVal cad As String, ByVal World As Long)
'sintaxi: id, alçada, radi, posició
Dim id As String, pos As D3DVECTOR, height As Single, radius As Single
Dim R1 As Long, R2 As Long
R1 = InStr(1, cad, ",")
id = Trim(left(cad, R1 - 1))
R2 = InStr(R1 + 1, cad, ",")
```

```
height = Val(mid(cad, R1 + 1, R2 - R1 - 1))
R1 = InStr(R2 + 1, cad, ",")
radius = Val(mid(cad, R2 + 1, R1 - R2 - 1))
pos = ParsePos(mid(cad, R1 + 1))

AddMissionTarget id, pos, height, radius, World
End Sub

Public Function ParsePos(ByVal cad As String) As D3DVECTOR
Dim R1 As Long, R2 As Long, cad1 As String
R1 = InStr(1, cad, "{")
R2 = InStr(1, cad, ",")
ParsePos.x = Val(Trim(mid(cad, R1 + 1, R2 - R1 - 1)))
R1 = InStr(1, cad, ",")
R2 = InStr(R1 + 1, cad, ",")
ParsePos.y = Val(Trim(mid(cad, R1 + 1, R2 - R1 - 1)))
ParsePos.z = Val(Trim(mid(cad, R2 + 1)))
End Function

Public Sub ParseArea(ByVal cad As String, v1 As D3DVECTOR, v2 As D3DVECTOR)
Dim R1 As Long
R1 = InStr(InStr(1, cad, "{") + 1, cad, "{")
v1 = ParsePos(left(cad, R1))
v2 = ParsePos(mid(cad, R1))
End Sub

Public Function ParseColor(ByVal cad As String) As D3DCOLORVALUE
Dim R1 As Long, R2 As Long, cad1 As String
R1 = InStr(1, cad, "(")
R2 = InStr(1, cad, ",")
ParseColor.r = Val(Trim(mid(cad, R1 + 1, R2 - R1 - 1)))
R1 = InStr(1, cad, ",")
R2 = InStr(R1 + 1, cad, ",")
ParseColor.g = Val(Trim(mid(cad, R1 + 1, R2 - R1 - 1)))
ParseColor.b = Val(Trim(mid(cad, R2 + 1)))
End Function

Public Sub AddPointToObject(object As clsDynObj, ByVal st As String)
Dim l As Long
Dim speed As Long, sleept As Long, pos As D3DVECTOR
l = InStr(1, st, "}")
pos = ParsePos(left(st, l))
l = InStr(l, st, ",")
speed = Val(Trim(mid(st, l + 1)))
l = InStr(l + 1, st, ",")
sleept = Val(Trim(mid(st, l + 1)))
If Not (object Is Nothing) Then object.AddPoint pos, speed, sleept
End Sub

Public Sub ShowLoadPercentStage(ByVal Percent As String)
Dim myPercent As String
myPercent = Percent
If Len(myPercent) < 3 Then myPercent = Space(3 - Len(myPercent)) & myPercen
t
Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_LIGHTING, False
Device.SetVertexShader myVertexFVF
Device.SetTextureStageState 0, D3DTSS_MAGFILTER, D3DPTFILTERCAPS_MAGFANISOT
ROPIC
Device.SetTextureStageState 0, D3DTSS_MINFILTER, D3DPTFILTERCAPS_MINFANISOT
```

GameScenes - 9

ROPIC

```
Dim width As Single, height As Single
Dim top As Single, left As Single
Dim tu As Single, tu2 As Single
Dim bottom As Single, right As Single
Dim char As Integer
width = 290 * D3DM.width / 800
height = 42 * D3DM.height / 600

LVertices(0) = AssignMV((D3DM.width - width) / 2, (D3DM.height - height) /
2, 0, 0, 0)     'arriba izq
LVertices(1) = AssignMV((D3DM.width + width) / 2, (D3DM.height - height) /
2, 0, 1, 0)     'arriba der
LVertices(2) = AssignMV((D3DM.width - width) / 2, (D3DM.height + height) /
2, 0, 0, 1)     'abajo izq
LVertices(3) = AssignMV((D3DM.width + width) / 2, (D3DM.height + height) /
2, 0, 1, 1)     'abajo der

width = 30 * D3DM.width / 800
height = 30 * D3DM.height / 600
top = D3DM.height - height * 2
left = D3DM.width - width * 5
bottom = D3DM.height - height
right = D3DM.width - width

Dim f As Integer
For f = 1 To 3
    char = Val(mid(myPercent, f, 1))
    If char = 0 Then char = 10
    tu = ((char - 1) * 30) / 330
    tu2 = (char * 30) / 330
    LVertices(f * 4) = AssignMV(left + (30 * D3DM.width / 800) * (f - 1), t
op, 0, tu, 0)
    LVertices(f * 4 + 1) = AssignMV(left + (30 * D3DM.width / 800) * f, top
, 0, tu2, 0)
    LVertices(f * 4 + 2) = AssignMV(left + (30 * D3DM.width / 800) * (f - 1
), bottom, 0, tu, 1)
    LVertices(f * 4 + 3) = AssignMV(left + (30 * D3DM.width / 800) * f, bot
tom, 0, tu2, 1)
Next

tu = ((10) * 30) / 330
LVertices(16) = AssignMV(left + (30 * D3DM.width / 800) * 3, top, 0, tu, 0)
LVertices(17) = AssignMV(right, top, 0, 1, 0)
LVertices(18) = AssignMV(left + (30 * D3DM.width / 800) * 3, bottom, 0, tu,
 1)
LVertices(19) = AssignMV(right, bottom, 0, 1, 1)

DoEvents
Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
Device.BeginScene

Device.SetTexture 0, LTextMain
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, LVertices(0), Len(LVertices(
0))

Device.SetTexture 0, LNum
If mid(myPercent, 1, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(4), Len(LVertices(0))
```

```
If mid(myPercent, 2, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(8), Len(LVertices(0))
If mid(myPercent, 3, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(12), Len(LVertices(0))
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, LVertices(16), Len(LVertices
(0))

Device.EndScene
Device.Present ByVal 0, ByVal 0, 0, ByVal 0
DoEvents
End Sub

Public Sub SetCorrectRenderStates(ByVal AmbientRGB As Long)
'---------- RENDER STATES BEFORE RENDERING ---------
Device.SetRenderState D3DRS_LIGHTING, 1
Device.SetRenderState D3DRS_ZWRITEENABLE, 1
Device.SetRenderState D3DRS_ZENABLE, 1
Device.SetRenderState D3DRS_ALPHABLENDENABLE, 0
Device.SetRenderState D3DRS_SRCBLEND, D3DBLEND_SRCALPHA
Device.SetRenderState D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA

Device.SetRenderState D3DRS_AMBIENT, AmbientRGB

Device.SetTextureStageState 0, D3DTSS_MAGFILTER, D3DTEXF_ANISOTROPIC
Device.SetTextureStageState 0, D3DTSS_MINFILTER, D3DTEXF_ANISOTROPIC
End Sub

Public Sub LoadingStageScreen(ByVal LoadValue As Integer)
If LoadValue > 0 Then LoadValue = 100
Dim width As Single, height As Single, top As Single, left As Single
Dim tu As Single, tv As Single
width = 256 * D3DM.width / 800
height = 50 * D3DM.height / 600
tv = Int((LoadValue / 100) * 8)
If tv > 4 Then
    tv = tv - 4
    tu = 0.5
Else
    tu = 0
End If
tv = tv * 16 / 64

Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_LIGHTING, False
Device.SetVertexShader myVertexFVF
Device.SetTextureStageState 0, D3DTSS_MAGFILTER, D3DPTFILTERCAPS_MAGFANISOT
ROPIC
Device.SetTextureStageState 0, D3DTSS_MINFILTER, D3DPTFILTERCAPS_MINFANISOT
ROPIC

DoEvents
Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
Device.BeginScene

top = D3DM.height - height * 1.75
left = (D3DM.width - width) / 2
LSVert(0) = AssignMV(left, top, 0, tu, tv)
LSVert(1) = AssignMV(left + width, top, 0, tu + 0.5, tv)
LSVert(2) = AssignMV(left, top + height, 0, tu, tv + 16 / 64)
LSVert(3) = AssignMV(left + width, top + height, 0, tu + 0.5, tv + 16 / 64)
```

```
Device.SetTexture 0, LSBar
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, LSVert(0), Len(LSVert(0))

Device.EndScene
Device.Present ByVal 0, ByVal 0, 0, ByVal 0
DoEvents
End Sub

Public Sub DestroyThisScene()
Dim x As Long, y As Long, z As Long
Static Init As Boolean
On Local Error Resume Next

If Init = True Then
    For x = 0 To UBound(FOComplexStage)
        Set FOComplexStage(x).Mesh = Nothing
        Set FOComplexStage(x).tmp_adj = Nothing
        Set FOComplexStage(x).tmp_mat = Nothing
    Next
    For x = 0 To UBound(FOSimpleStage)
        Set FOSimpleStage(x).Mesh = Nothing
        Set FOSimpleStage(x).tmp_adj = Nothing
        Set FOSimpleStage(x).tmp_mat = Nothing
    Next
    For x = 0 To UBound(MovingCharacters)
        Set MovingCharacters(x) = Nothing
    Next
    For x = 0 To UBound(OtherCharAnimations)
        For y = 0 To OtherCharAnimations(x).NumMeshes
            Set OtherCharAnimations(x).Meshes(y) = Nothing
        Next
        Set OtherCharAnimations(x).tmp_adj = Nothing
        Set OtherCharAnimations(x).tmp_mat = Nothing
    Next
End If
ReDim FOSimpleStage(0)
ReDim FOComplexStage(0)
ReDim FixedObjectsStage(0)
ReDim MovingCharacters(0)
ReDim OtherCharAnimations(0)

ReDim LevelScenes(0)
ReDim MissionTargets(0)
ReDim PendingSounds(0)
ReDim Lights(0)

Call SoundEngine.DestroySounds
ReDim DS_Sounds(0)
On Local Error GoTo 0
Init = True
End Sub
```

```
GameStage - 1

Option Explicit

'- This module controls the game missions and stages ---

'--- First mission stage -----
Public Sub ResetGS(gs As GameSlot)
gs.Health = 100
gs.MissionState1 = 1
gs.MissionState2 = 0
gs.MissionState3 = 0
gs.MissionState4 = 0
gs.NumObjects = 0
ReDim gs.ObjectsID(0)
gs.Position = v3(-38, 0, -30)
gs.RotationH = 180
gs.WorldID = 1
End Sub

Public Sub EnterNewWorld(ByVal WorldID As Integer, ByVal Door As Integer)
Static MemWorldID As Integer, MemDoor As Integer
If MakeFade1 = False Then
    MemWorldID = WorldID
    MemDoor = Door
    MakeFade1 = True
    Exit Sub
Else
    WorldID = MemWorldID
    Door = MemDoor
    MakeFade1 = False
    MakeFade2 = True
    ResetAllMoves = True
End If

With TheGameSlot

    Select Case WorldID
    Case 1
        Select Case Door
        Case 1
            .Position = v3(29, 6.5, 86)         'circ pretori pis 1
            .RotationH = 25
        Case 2
            .Position = v3(28, 0.1, 83)         'circ pretori baixos
            .RotationH = 25
        Case 3
            .Position = v3(-292, 6.5, 87)         'circ forum pis 1 porta e
squerra auxiliar
            .RotationH = 270
        Case 4
            .Position = v3(-247, 4, 77)        'circ boca ultima esquerra
            .RotationH = 0
        End Select
    Case 2
        Select Case Door
        Case 1
            .Position = v3(20, 12, 100)         'forum porta dreta (pretori
)
            .RotationH = 90
        Case 2
            .Position = v3(-292, 12, 100)         'forum porta esquerra
```

GameStage - 2

```
            .RotationH = 270
        End Select
    Case 3
        Select Case Door
        Case 1
            .Position = v3(-6, 0, -11)         'baixa
            .RotationH = 180
        Case 2
            .Position = v3(-8, 6, -11)         'mitjana
            .RotationH = 180
        Case 3
            .Position = v3(-8, 11.5, -8)        'alta
            .RotationH = 270
        End Select
    End Select

End With

TheGameSlot.WorldID = WorldID
CharAngleH = TheGameSlot.RotationH
CharAngleV = (MaxVerticalAngle + MinVerticalAngle) / 2
CharPos = TheGameSlot.Position
CharDistance = (MaxCameraDistance + MinCameraDistance) / 2
CameraPos = ProcessCameraCoords(CharPos, CharAngleH, CharAngleV, CharDistan
ce)
InitY = CharPos.y
Movement = 0
Call SetCorrectRenderStates(RGB(WorldProperties(TheGameSlot.WorldID).Ambien
tValues.x, WorldProperties(TheGameSlot.WorldID).AmbientValues.y, WorldPrope
rties(TheGameSlot.WorldID).AmbientValues.z))
Call UpdateLights
End Sub

Public Sub UpdateLights()
Dim x As Long, cont As Long
Dim light As D3DLIGHT8
For x = 0 To 7
    Device.LightEnable x, 0
Next
For x = 1 To UBound(Lights)
    If Lights(x).WorldID = TheGameSlot.WorldID Then
        Select Case Lights(x).type
        Case Directional
            light.type = D3DLIGHT_DIRECTIONAL
            light.Direction = Lights(x).Direction
        Case Omni
            light.type = D3DLIGHT_POINT
            light.Position = Lights(x).Position
            light.Attenuation0 = 0.1
            light.Range = Lights(x).Range
        Case Target
            light.type = D3DLIGHT_SPOT
            light.Position = Lights(x).Position
            light.Direction = Lights(x).Direction
            light.Range = Lights(x).Range
            light.Phi = Lights(x).Phi
            light.Theta = Lights(x).Theta
        End Select
        light.diffuse = Lights(x).Color
        Device.SetLight cont, light
```

GameStage - 3

```
        Device.LightEnable cont, 1
        cont = cont + 1
    End If
Next
End Sub

Public Sub ComputeDoors()
If MakeFade1 Then Exit Sub      'if we are changing do not test doors or we
'll get errors or a inf loop
Dim NewDoor As Integer, NewWorld As Integer
Select Case TheGameSlot.WorldID
Case 1
    If CheckPointInCube(v3(29.1, 6, 91), v3(31.7, 9, 89.6), CharPos) Then
        EnterNewWorld 3, 2
        Exit Sub
    End If
    If CheckPointInCube(v3(27, -1, 88), v3(32, 3, 86), CharPos) Then
        EnterNewWorld 3, 1
        Exit Sub
    End If
Case 2
    If CheckPointInCube(v3(23.5, 11, 101.5), v3(25, 15, 98.5), CharPos) The
n
        EnterNewWorld 3, 3
        Exit Sub
    End If
    If CheckPointInCube(v3(-297, 11, 101.5), v3(-296, 15, 99), CharPos) The
n
        EnterNewWorld 1, 3
        Exit Sub
    End If
Case 3
    If CheckPointInCube(v3(-7.5, -1, -14), v3(-4, 4, -15), CharPos) Then
        EnterNewWorld 1, 2
        Exit Sub
    End If
    If CheckPointInCube(v3(-9, 5.5, -14), v3(-6.5, 9, -15), CharPos) Then
        EnterNewWorld 1, 1
        Exit Sub
    End If
    If CheckPointInCube(v3(-13, 11, -7), v3(-11.75, 15, -9.5), CharPos) The
n
        EnterNewWorld 2, 1
        Exit Sub
    End If
End Select
End Sub
```

MainModule - 1

```
Option Explicit

Dim debugskip As Boolean

Sub Main()
frmLoading.Show
If Command() = "debug" Then debugskip = True
'------------ Aplication essentials --------------
EXE = App.path
If right(EXE, 1) <> "\" Then EXE = EXE & "\"
Randomize Timer
'------------ Program initial comprobations ----------
Call InitialComprobations
'------------ Init DirectX -----------------
Call InitDx
'------------ Load resolution modes ----------
Call LoadDM
'------------ Create Device -----------------
Unload frmLoading
Call CreateDev
'-- Load Loading screen and percent numbers --
Call LoadPrev
'-------------- Load Menus -------------
Call LoadMenu
'---------------- LoadSettings --------
Call LoadSettings
'-------------- Load Models & Obj & Col --------
ReDim PendingTextures(0)
Call LoadWorld
Call RefreshDrawDepth
Call LoadWorldShadows
Call LoadCollisionWorld
ReDim FOComplex(0): ReDim FOSimple(0)
Call LoadFixedObjects
Call LoadCollisionFO
'-------------- Load Chars & Anims --------
Call LoadMainCharAnims

'------------- Load Generic Sounds -----------
Call LoadCommonSounds
'-------------- Start Music ----------
Call InitMusic
'------------ Load Textures ----------
Call LoadSkyBox
Call LoadSM
Call LoadPendingTextures(70, 100)
'--------------- Capture Mouse ---------
Call MouseSetUp
'--------------- MenuShow ---------------
'(and destroy load screen as we wont use it any more)
Set LTextMain = Nothing
Set LNum = Nothing

MenuAgain:
Call MenuSystem
'----- After the Menu Start Game or Exit -----
If ExitToWin = False Then
    '--- GET INTO THE GAME LOOP ---
    Select Case GlobalMenuOption
    Case 1  'start new game
```

MainModule - 2

```
        ResetGS TheGameSlot
        Call GameStart    'Start the game, the function will retun when exi
ting
        Call GameEnd      'The game has exited, delete world objects such a
s sound
    Case 2  'Load saved game

    End Select
End If
'---- End of the game --- Show menu or quit
If ExitToWin = False Then
    GoTo MenuAgain
End If

'--------- SAVE SETTINGS BEFOR EXITING -------
Call SaveSettings

'------- END (al fin) ------
On Local Error Resume Next
MouseDevice.Unacquire
Set MouseDevice = Nothing
Call MusicEngine.DestroyMusic
Set Device = Nothing
Set Direct3D = Nothing
Set Direct3DX = Nothing
Set DirectSound = Nothing
Set DirectInput = Nothing
Set DirectX = Nothing
DeleteDir MusicDir

Unload frmGraphics
End
End Sub

Public Sub InitialComprobations()
Set Reg = New clsWinReg

If FileExists(EXE & "engine.dll") = False Then GoTo Error

Reg.CreateKey "HKLM\Software\" & RegName & "\"

Exit Sub
Error:
MsgBox "El programa no està correctament instal·lat. Falten un o més arxius
", vbCritical, "Error intern"
End
End Sub

Public Sub InitDx()
On Local Error GoTo ErrExit
Dim ret As Long
Set DirectX = New DirectX8
Set Direct3D = DirectX.Direct3DCreate()
Set DirectInput = DirectX.DirectInputCreate()
Set MouseDevice = DirectInput.CreateDevice("guid_SysMouse")
Set Direct3DX = New D3DX8
Direct3D.GetDeviceCaps 0, D3DDEVTYPE_HAL, caps

On Local Error Resume Next
```

MainModule - 3

```
Dim SoundDevice As String
Set DirectSound = DirectX.DirectSoundCreate("")
If Err.number <> 0 Then
    ret = MsgBox("Els controladors de so no estan correctament instalats. V
ols continuar sense so?", vbExclamation + vbYesNo, "Error de so")
    If ret = vbYes Then
        DisableSound = True
    Else
        End
    End If
End If
ReDim DS_Sounds_Plain(0)
ReDim DS_Sounds(0)

Exit Sub
ErrExit:

MsgBox "El programa no ha pogut iniciar DirectX correctament. Comproba la v
ersió instal·lada. Ha de ser 8.1 o superior", vbCritical, "Error intern"
End
End Sub

Public Sub LoadDM()
Dim x As Long, DModeTemp As D3DDISPLAYMODE
ReDim DModesArray(Direct3D.GetAdapterModeCount(0) - 1)
For x = 0 To Direct3D.GetAdapterModeCount(0) - 1
    Direct3D.EnumAdapterModes 0, x, DModeTemp
    DModesArray(x) = DModeTemp
Next
GetAllResolutions 800, 600, ResolutionArray
End Sub

Public Sub CreateDev()
Dim Param As D3DPRESENT_PARAMETERS
Dim LastDM As String

LastDM = RegRead("Resolution")
AntiAliasLevel = Val(RegRead("antialias"))

If LastDM = "" Then
    D3DM = CreateDM(800, 600, True)
Else
    D3DM = ParseDM(LastDM)
End If

If Direct3D.CheckDeviceMultiSampleType(0, D3DDEVTYPE_HAL, D3DM.Format, 0, A
ntiAliasLevel) <> D3D_OK Then
    'incorrect multisample!!!
    AntiAliasLevel = 0
End If

SaveResolution D3DM, AntiAliasLevel

frmGraphics.Show

With Param
    '.Windowed = 1
    .SwapEffect = D3DSWAPEFFECT_DISCARD
    .BackBufferFormat = D3DM.Format
    .EnableAutoDepthStencil = 1
```

```
    .BackBufferCount = 1
    .BackBufferWidth = D3DM.width
    .BackBufferHeight = D3DM.height
    .AutoDepthStencilFormat = D3DFMT_D24S8
    .MultiSampleType = AntiAliasLevel
End With

On Local Error Resume Next
'------- 24 bit depth + 8 bit stencil  ---- HARDWARE ACCELERATION --------
Err.number = 0
Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd, D3D
CREATE_HARDWARE_VERTEXPROCESSING, Param)
'------- 24 bit depth + 4 bit stencil + 4 unused bits ---- HARDWARE ACCELER
ATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X4S4
    Err.Clear: Err.number = 0
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
End If

'------- 24 bit depth + 8 unused bits ---- HARDWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X8
    Err.Clear: Err.number = 0
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
End If
'------- 16 bit depth ---- HARDWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D16
    Err.Clear: Err.number = 0
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
End If
'------- 24 bit depth ---- SOFTWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X8
    Err.Clear: Err.number = 0
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_SOFTWARE_VERTEXPROCESSING, Param)
End If
'------- 16 bit depth ---- SOFTWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D16
    Err.Clear: Err.number = 0
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_SOFTWARE_VERTEXPROCESSING, Param)
End If
On Local Error GoTo 0

Select Case Param.AutoDepthStencilFormat
Case D3DFMT_D24S8, D3DFMT_D24X4S4
    ShadowsAvaliable = TestStencilCaps()
Case Else
    ShadowsAvaliable = False
End Select
End Sub

Public Sub SaveResolution(DM As D3DDISPLAYMODE, ByVal AntiAlias As Long)
```

```
RegSave "Resolution", Format(DM.width, "0000") & Format(DM.height, "0000")
& Format(DM.RefreshRate, "0000") & Format(DM.Format, "0000")
RegSave "AntiAlias", Trim(Str(AntiAlias))
End Sub

Public Sub CursorV(ByVal IsVisible As Boolean)
ShowCursor IsVisible
End Sub

Public Sub LoadPrev()
Dim mypath As String, x As Integer
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "tex_tmp_prev" & Format(Int(Rnd * 5000) + 1, "0000") & "\
"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "maps\prev.dat", mypath

Dim imagew As Long, imageh As Long
Dim imagew2 As Long, imageh2 As Long

Set LTextMain = LoadTextureAndReturn(mypath & "loading.bmp")

Set LNum = LoadTextureAndReturn(mypath & "num.bmp")

Set MouseTexture = Direct3DX.CreateTextureFromFileEx(Device, mypath & "curs
or.tga", 40, 40, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_FIL
TER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
CursorW = 40: CursorH = 40

Set FontTexture = Direct3DX.CreateTextureFromFileEx(Device, mypath & "font.
tga", 256, 256, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX_FILT
ER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)

DeleteDir mypath

Call ShowLoadPercent("0")
End Sub

Public Sub ShowLoadPercent(ByVal Percent As String)
Dim myPercent As String
myPercent = Percent
If Len(myPercent) < 3 Then myPercent = Space(3 - Len(myPercent)) & myPercen
t
Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_LIGHTING, False
Device.SetVertexShader myVertexFVF
Device.SetTextureStageState 0, D3DTSS_MAGFILTER, D3DPTFILTERCAPS_MAGFANISOT
ROPIC
Device.SetTextureStageState 0, D3DTSS_MINFILTER, D3DPTFILTERCAPS_MINFANISOT
ROPIC

Dim width As Single, height As Single
Dim top As Single, left As Single
Dim tu As Single, tu2 As Single
Dim bottom As Single, right As Single
Dim char As Integer
width = 290 * D3DM.width / 800
```

MainModule - 6

```
height = 42 * D3DM.height / 600

LVertices(0) = AssignMV((D3DM.width - width) / 2, (D3DM.height - height) /
2, 0, 0, 0)     'arriba izq
LVertices(1) = AssignMV((D3DM.width + width) / 2, (D3DM.height - height) /
2, 0, 1, 0)     'arriba der
LVertices(2) = AssignMV((D3DM.width - width) / 2, (D3DM.height + height) /
2, 0, 0, 1)     'abajo izq
LVertices(3) = AssignMV((D3DM.width + width) / 2, (D3DM.height + height) /
2, 0, 1, 1)     'abajo der

width = 30 * D3DM.width / 800
height = 30 * D3DM.height / 600
top = D3DM.height - height * 2
left = D3DM.width - width * 5
bottom = D3DM.height - height
right = D3DM.width - width

Dim f As Integer
For f = 1 To 3
    char = Val(mid(myPercent, f, 1))
    If char = 0 Then char = 10
    tu = ((char - 1) * 30) / 330
    tu2 = (char * 30) / 330
    LVertices(f * 4) = AssignMV(left + (30 * D3DM.width / 800) * (f - 1), t
op, 0, tu, 0)
    LVertices(f * 4 + 1) = AssignMV(left + (30 * D3DM.width / 800) * f, top
, 0, tu2, 0)
    LVertices(f * 4 + 2) = AssignMV(left + (30 * D3DM.width / 800) * (f - 1
), bottom, 0, tu, 1)
    LVertices(f * 4 + 3) = AssignMV(left + (30 * D3DM.width / 800) * f, bot
tom, 0, tu2, 1)
Next

tu = ((10) * 30) / 330
LVertices(16) = AssignMV(left + (30 * D3DM.width / 800) * 3, top, 0, tu, 0)
LVertices(17) = AssignMV(right, top, 0, 1, 0)
LVertices(18) = AssignMV(left + (30 * D3DM.width / 800) * 3, bottom, 0, tu,
 1)
LVertices(19) = AssignMV(right, bottom, 0, 1, 1)

DoEvents
Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
Device.BeginScene

Device.SetTexture 0, LTextMain
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, LVertices(0), Len(LVertices(
0))

Device.SetTexture 0, LNum
If mid(myPercent, 1, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(4), Len(LVertices(0))
If mid(myPercent, 2, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(8), Len(LVertices(0))
If mid(myPercent, 3, 1) <> " " Then Device.DrawPrimitiveUP D3DPT_TRIANGLEST
RIP, 2, LVertices(12), Len(LVertices(0))
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, LVertices(16), Len(LVertices
(0))

Device.EndScene
```

```
Device.Present ByVal 0, ByVal 0, 0, ByVal 0
DoEvents
End Sub

'--------------------------------------------------------------------------
----------------
'------- Here I create the menus for the game. All the textures included in
 ---------------
'------- Menu.dat inside graphics folder. For the events watch MenuSystem
 ---------------
'--------------------------------------------------------------------------
----------------
Public Sub LoadMenu()
Dim mypath As String, x As Integer, number As Integer
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "tex_tmp_prev" & Format(Int(Rnd * 5000) + 1, "0000") & "\
"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "maps\menu.dat", mypath
ShowLoadPercent "2"

Set AuxiliarMenu = New cGameMenu
AuxiliarMenu.TexPath = mypath
AuxiliarMenu.BackGround = "auxmenu.bmp"
AuxiliarMenu.AddButton 250, 250, 300, 90, "returngame.tga", "returngame_ove
r.tga", "returngame", 0
AuxiliarMenu.AddButton 250, 400, 300, 90, "returnmenu.tga", "returnmenu_ove
r.tga", "returnmenu", 0
AuxiliarMenu.BuildMenu

Set MainMenu = New cGameMenu
MainMenu.TexPath = mypath
MainMenu.BackGround = "mainmenu.bmp"
MainMenu.AddButton 600, 500, 150, 50, "exit.tga", "exit_over.tga", "exitwin
", 0
MainMenu.AddButton 500, 150, 200, 60, "newgame.tga", "newgame_over.tga", "n
ewgame", 0
MainMenu.AddButton 500, 250, 200, 60, "loadgame.tga", "loadgame_over.tga",
"loadgame", 0
MainMenu.AddButton 500, 350, 200, 60, "options.tga", "options_over.tga", "o
ptions", 0
MainMenu.BuildMenu
ShowLoadPercent "5"

Set ConfigMenu = New cGameMenu
ConfigMenu.TexPath = mypath
ConfigMenu.BackGround = "optionsmenu.bmp"
ConfigMenu.AddButton 600, 500, 150, 50, "return.tga", "return_over.tga", "a
pplyconfig", 0
ConfigMenu.AddList 450, 180, 250, 30, 20, 4, "resolution", 1

For x = 1 To UBound(ResolutionArray)
    ConfigMenu.AddElementToList "resolution", ResolutionArray(x)
Next

ConfigMenu.AddLabel "Resolucio de pantalla", 100, 187, 16, 15
ConfigMenu.AddLabel "profunditat de color", 100, 227, 16, 15
```

```
ConfigMenu.AddLabel "suavitzar imatge", 100, 267, 16, 15
ConfigMenu.AddLabel "Produnditat de dibuix", 100, 347, 16, 15
ConfigMenu.AddLabel "Velocitat del cursor", 100, 387, 16, 15
ConfigMenu.AddLabel "Ombres", 100, 427, 16, 15

ConfigMenu.AddList 450, 220, 250, 30, 20, 4, "bits", 1
ConfigMenu.AddElementToList "bits", "16"
ConfigMenu.AddElementToList "bits", "32"
ConfigMenu.AddList 450, 260, 250, 30, 20, 4, "alias", 1
ConfigMenu.AddElementToList "alias", "no"
For x = 2 To 16
    If Direct3D.CheckDeviceMultiSampleType(0, D3DDEVTYPE_HAL, D3DM.Format,
0, x) = D3D_OK Then ConfigMenu.AddElementToList "alias", "x" & Trim(Str(x))
Next

ConfigMenu.AddList 450, 340, 250, 30, 20, 4, "zdetail", 1
ConfigMenu.AddElementToList "zdetail", "baixa"
ConfigMenu.AddElementToList "zdetail", "mitjana"
ConfigMenu.AddElementToList "zdetail", "alta"
ConfigMenu.AddList 450, 380, 250, 30, 20, 2, "vcursor", 1
ConfigMenu.AddElementToList "vcursor", "molt baixa"
ConfigMenu.AddElementToList "vcursor", "baixa"
ConfigMenu.AddElementToList "vcursor", "mitjana"
ConfigMenu.AddElementToList "vcursor", "alta"
ConfigMenu.AddElementToList "vcursor", "molt alta"
If ShadowsAvaliable Then
    ConfigMenu.AddList 450, 420, 250, 30, 20, 2, "shadows", 1
    ConfigMenu.AddElementToList "shadows", "desactivades"
    ConfigMenu.AddElementToList "shadows", "activades"
End If

ConfigMenu.ListTextureOpen = "combo2.tga"
ConfigMenu.ListTextureOpenHover = "combo3.tga"
ConfigMenu.ListTexture = "combo1.tga"

ConfigMenu.BuildMenu
ShowLoadPercent "9"

DeleteDir mypath
End Sub

Public Sub MenuSystem()
Device.SetRenderState D3DRS_ALPHABLENDENABLE, True
Device.SetVertexShader myVertexFVF
Device.SetRenderState D3DRS_ZENABLE, False
Device.SetRenderState D3DRS_SRCBLEND, D3DBLEND_SRCALPHA
Device.SetRenderState D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA

Dim Out As Boolean
Dim MenuIndex As Integer, ActualMenu As cGameMenu, LastMenuIndex As Integer
Dim mTimer As Double
Dim coordX As Single, CoordY As Single
Dim alpha As Single, x As Integer
Dim DMode As D3DDISPLAYMODE, NewAlias As Long

coordX = D3DM.width / 2
CoordY = D3DM.height / 2

GlobalMenuOption = 0
Do While Out = False
```

```
    Set ActualMenu = MenuFromIndex(MenuIndex)
    '----------- MOUSE COORDS PROCESS ---------------
    coordX = coordX + MouseX * (CursorSpeed / 3)
    CoordY = CoordY + MouseY * (CursorSpeed / 3)
    If coordX < 0 Then coordX = 0
    If coordX > D3DM.width Then coordX = D3DM.width
    If CoordY < 0 Then CoordY = 0
    If CoordY > D3DM.height Then CoordY = D3DM.height
    MouseVerts(0) = AssignMV(coordX, CoordY, 0, 0, 0)
    MouseVerts(1) = AssignMV(coordX + CursorW, CoordY, 0, 1, 0)
    MouseVerts(2) = AssignMV(coordX, CoordY + CursorH, 0, 0, 1)
    MouseVerts(3) = AssignMV(coordX + CursorW, CoordY + CursorH, 0, 1, 1)
    MouseX = 0: MouseY = 0
    '---------------- SEND EVENTS TO MENUS ------------
    ActualMenu.ProcessMouseMove coordX, CoordY
    If MouseClick0 = True Then
        ActualMenu.ProcessClick coordX, CoordY
        MouseClick0 = False
    End If
    '--------------- Look for events ---------------
    If ActualMenu.isEvent Then
        Select Case ActualMenu.EventName
        Case "exitwin"
            ExitToWin = True
            Out = True
        Case "newgame"
            Out = True
            GlobalMenuOption = 1
        Case "loadgame"
            Out = True
            GlobalMenuOption = 2
        Case "options"
            If Is32bit(D3DM) Then
                ConfigMenu.SetSelectedListItem "bits", 2
            Else
                ConfigMenu.SetSelectedListItem "bits", 1
            End If

            ConfigMenu.SetSelectedListItemByText "resolution", D3DM.width &
 " x " & D3DM.height
            ConfigMenu.SetSelectedListItem "zdetail", DrawDepth
            ConfigMenu.SetSelectedListItem "vcursor", CursorSpeed
            If AntiAliasLevel <> 0 Then
                ConfigMenu.SetSelectedListItemByText "alias", "x" & Trim(St
r(AntiAliasLevel))
            Else
                ConfigMenu.SetSelectedListItem "alias", 0
            End If
            ConfigMenu.SetSelectedListItem "shadows", EnableShadows + 1
            MenuIndex = 1
        Case "applyconfig"
            MenuIndex = 0
            DMode = ParseResolution(ConfigMenu.GetListValue("resolution"))
            NewAlias = Val(mid(ConfigMenu.GetListValue("alias"), 2))
            If ConfigMenu.GetListValue("bits") = "32" Then
                DMode = CreateDM(DMode.width, DMode.height, True)
            Else
                DMode = CreateDM(DMode.width, DMode.height, False)
            End If
            If DMode.Format <> D3DM.Format Or _
```

```
                DMode.width <> D3DM.width Or _
                DMode.height <> D3DM.height Or _
                NewAlias <> AntiAliasLevel Then _
                'the user has changed de DM
                    SaveRS
                    If ResetDevice(DMode, NewAlias) Then
                        D3DM = DMode
                        AntiAliasLevel = NewAlias
                        MainMenu.RefreshDM
                        ConfigMenu.RefreshDM
                    End If
                    SaveResolution D3DM, AntiAliasLevel
                    RestoreRS
            End If
            DrawDepth = ConfigMenu.GetListIndex("zdetail")
            CursorSpeed = ConfigMenu.GetListIndex("vcursor")
            CharDetail = ConfigMenu.GetListIndex("pdetail")
            TexQuality = ConfigMenu.GetListIndex("texq")
            If ShadowsAvaliable Then EnableShadows = ConfigMenu.GetListInde
x("shadows") - 1
        End Select
        ActualMenu.isEvent = False
    End If
    Set ActualMenu = MenuFromIndex(MenuIndex)

    '----------------------- MAKE A FADE OUT / IN BETWEEN MENUS -----------
---------
    If LastMenuIndex <> MenuIndex Or (Out = True And GlobalMenuOption <> 0)
 Then
        Device.SetVertexShader myVertexAlphaFVF
        mTimer = GetTickCount()
        Do While (mTimer + 1000) > GetTickCount()
            alpha = 255 * (GetTickCount() - mTimer) / 1000
            If alpha > 255 Then alpha = 255
            If alpha < 0 Then alpha = 0
            fadeVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(alpha, 0, 0,
0))
            fadeVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(alph
a, 0, 0, 0))
            fadeVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(alp
ha, 0, 0, 0))
            fadeVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColo
rARGB(alpha, 0, 0, 0))

            Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0
, 1#, 0
            Device.BeginScene

            Device.SetVertexShader myVertexFVF
            MenuFromIndex(LastMenuIndex).RenderMenu

            Device.SetTexture 0, Nothing
            Device.SetVertexShader myVertexAlphaFVF
            Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, fadeVertices(0),
 Len(fadeVertices(0))

            Device.EndScene
            Device.Present ByVal 0, ByVal 0, 0, ByVal 0
            DoEvents
        Loop
```

```
    End If
    If LastMenuIndex <> MenuIndex Then
        mTimer = GetTickCount()
        Do While (mTimer + 1000) > GetTickCount()
            alpha = 255 - (255 * (GetTickCount() - mTimer) / 1000)
            If alpha > 255 Then alpha = 255
            If alpha < 0 Then alpha = 0
            fadeVertices(0) = AssignMVA(0, 0, 0, D3DColorARGB(alpha, 0, 0,
0))
            fadeVertices(1) = AssignMVA(D3DM.width, 0, 0, D3DColorARGB(alph
a, 0, 0, 0))
            fadeVertices(2) = AssignMVA(0, D3DM.height, 0, D3DColorARGB(alp
ha, 0, 0, 0))
            fadeVertices(3) = AssignMVA(D3DM.width, D3DM.height, 0, D3DColo
rARGB(alpha, 0, 0, 0))

            Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0
, 1#, 0
            Device.BeginScene

            Device.SetVertexShader myVertexFVF
            MenuFromIndex(MenuIndex).RenderMenu

            Device.SetTexture 0, Nothing
            Device.SetVertexShader myVertexAlphaFVF
            Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, fadeVertices(0),
 Len(fadeVertices(0))

            Device.EndScene
            Device.Present ByVal 0, ByVal 0, 0, ByVal 0
            DoEvents
        Loop
        LastMenuIndex = MenuIndex
        Device.SetVertexShader myVertexFVF
        MouseClick0 = False: MouseClick1 = False
        MouseX = 0: MouseY = 0
    End If
    '-------------------------------------------------------------------
-----------------------

    If GlobalMenuOption <> 0 Then Exit Sub

    Device.Clear 0, ByVal 0, D3DCLEAR_TARGET Or D3DCLEAR_ZBUFFER, 0, 1#, 0
    Device.BeginScene

    '------------ RENDER THE MENU --------
    ActualMenu.RenderMenu           'the class does all the work

    '----------- RENDER THE CURSOR OVER ALL -----------
    Device.SetTexture 0, MouseTexture
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, MouseVerts(0), Len(Mouse
Verts(0))

    Device.EndScene
    Device.Present ByVal 0, ByVal 0, 0, ByVal 0

    Sleep 5
    DoEvents
Loop
End Sub
```

```
Public Function MenuFromIndex(ByVal index As Integer) As cGameMenu
Select Case index
Case 0
    Set MenuFromIndex = MainMenu
Case 1
    Set MenuFromIndex = ConfigMenu
End Select
End Function

Public Function CreateDM(ByVal width As Single, ByVal height As Single, ByV
al bit32 As Boolean) As D3DDISPLAYMODE
Dim x As Integer
CreateDM.width = width
CreateDM.height = height
CreateDM.RefreshRate = 0
If bit32 Then
    'chek for 32 bits format in fullscreen  (D3DFMT_X8R8G8B8)
    For x = 0 To UBound(DModesArray)
        If DModesArray(x).width = width And DModesArray(x).height = height
Then
        If (DModesArray(x).Format And D3DFMT_X8R8G8B8) = D3DFMT_X8R8G8B8 Th
en
            CreateDM.Format = D3DFMT_X8R8G8B8
            Exit Function
        End If
        End If
    Next
    'if we arrive here it means that there isnt any DMode in 32 bits
    'so we have to try with 16 bits.....
    GoTo Try32Bits
Else
Try32Bits:
    'check for 16 bits format in fullscreen  (D3DFMT_R5G6B5)
    For x = 0 To UBound(DModesArray)
        If DModesArray(x).width = width And DModesArray(x).height = height
Then
        If (DModesArray(x).Format And D3DFMT_R5G6B5) = D3DFMT_R5G6B5 Then
            CreateDM.Format = D3DFMT_R5G6B5
            Exit Function
        End If
        End If
    Next
    'if not, check for another possible format   ('chek for 16 bits format
in fullscreen  (D3DFMT_X1R5G5B5)
    For x = 0 To UBound(DModesArray)
        If DModesArray(x).width = width And DModesArray(x).height = height
Then
        If (DModesArray(x).Format And D3DFMT_X1R5G5B5) = D3DFMT_X1R5G5B5 Th
en
            CreateDM.Format = D3DFMT_X1R5G5B5
            Exit Function
        End If
        End If
    Next
End If
End Function

Public Function ResetDevice(DMode As D3DDISPLAYMODE, ByVal AntiAlias As Lon
```

```
g) As Boolean
Dim Param As D3DPRESENT_PARAMETERS, OriginalDM As D3DDISPLAYMODE, failedonc
e As Boolean
Dim TestDM As D3DDISPLAYMODE
Dim Create As Boolean

If Device Is Nothing Then
    Create = True
End If

If Not Create Then Device.GetDisplayMode OriginalDM
TestDM = DMode
With Param
    .SwapEffect = D3DSWAPEFFECT_DISCARD
    .BackBufferFormat = TestDM.Format
    .EnableAutoDepthStencil = 1
    .BackBufferCount = 1
    .BackBufferWidth = TestDM.width
    .BackBufferHeight = TestDM.height
    .AutoDepthStencilFormat = D3DFMT_D24S8
    .MultiSampleType = AntiAlias
End With

On Local Error Resume Next
'------- 24 bit depth + 8 bit stencil ---- HARDWARE ACCELERATION --------
Err.number = 0
If Create Then
    Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.hWnd,
 D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
Else
    Device.Reset Param
End If
'------- 24 bit depth + 4 bit stencil ---- HARDWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X4S4
    Err.Clear: Err.number = 0
    If Create Then
        Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.h
Wnd, D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
    Else
        Device.Reset Param
    End If
End If

'------- 24 bit depth + 8 unused bits ---- HARDWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X8
    Err.Clear: Err.number = 0
    If Create Then
        Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.h
Wnd, D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
    Else
        Device.Reset Param
    End If
End If
'------- 16 bit depth  ---- HARDWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D16
    Err.Clear: Err.number = 0
    If Create Then
```

```
        Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.h
Wnd, D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
    Else
        Device.Reset Param
    End If
End If

'------- 24 bit depth ---- SOFTWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D24X8
    Err.Clear: Err.number = 0
    If Create Then
        Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.h
Wnd, D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
    Else
        Device.Reset Param
    End If
End If
'------- 16 bit depth ---- SOFTWARE ACCELERATION --------
If Err.number <> 0 Then
    Param.AutoDepthStencilFormat = D3DFMT_D16
    Err.Clear: Err.number = 0
    If Create Then
        Set Device = Direct3D.CreateDevice(0, D3DDEVTYPE_HAL, frmGraphics.h
Wnd, D3DCREATE_HARDWARE_VERTEXPROCESSING, Param)
    Else
        Device.Reset Param
    End If
End If
If Err.number <> 0 And failedonce = False Then
    failedonce = True
    TestDM = OriginalDM
ElseIf failedonce = True And Err.number <> 0 Then
    ResetDevice = False
    Exit Function
End If
ResetDevice = True
On Local Error GoTo 0

Select Case Param.AutoDepthStencilFormat
Case D3DFMT_D24S8, D3DFMT_D24X4S4
    ShadowsAvaliable = TestStencilCaps()
Case Else
    ShadowsAvaliable = False
End Select
End Function

Public Function TestStencilCaps() As Boolean
If caps.StencilCaps & D3DSTENCILOP_KEEP Then
If caps.StencilCaps & D3DSTENCILOP_ZERO Then
If caps.StencilCaps & D3DSTENCILOP_REPLACE Then
If caps.StencilCaps & D3DSTENCILOP_INCR Then
If caps.StencilCaps & D3DSTENCILOP_DECR Then
    TestStencilCaps = True
    Exit Function
End If
End If
End If
End If
End If
```

```
MainModule - 15

TestStencilCaps = False
End Function

Public Sub GameStart()
'------ Start the Game --------
'-- 1. Load the current sounds / animations / ambients and other stuff for
the current part
'-- 2. Jump to GameLoop Module
'------------------------------------------------------------------------
----------------------

Call GameLoadLevel
Call GameLoop.GameLoop

End Sub

Public Sub GameEnd()
Call DestroyStandardSound
End Sub

Public Sub LoadWorld()
Dim mypath As String, x As Long, number As Integer, y As Long
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "models_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "graphics\main.dat", mypath
ShowLoadPercent "13"

For x = 1 To 5
    Set RenderModels(x).Mesh = Direct3DX.LoadMeshFromX(mypath & "main_" & T
rim(Str(x)) & ".x", D3DXMESH_MANAGED Or D3DXMESH_32BIT, Device, RenderModel
s(x).tmp_adj, RenderModels(x).tmp_mat, RenderModels(x).NumMaterials)
    ReDim RenderModels(x).Materials(RenderModels(x).NumMaterials - 1)
    ReDim RenderModels(x).TexturesNames(RenderModels(x).NumMaterials - 1)
Next
ShowLoadPercent "14"

For x = 1 To 5
    OptimizeMesh RenderModels(x).Mesh, RenderModels(x).tmp_adj
    Set RenderModels(x).tmp_adj = Nothing
Next
ShowLoadPercent "15"

For x = 1 To 5
    For y = 0 To RenderModels(x).NumMaterials - 1
        Direct3DX.BufferGetMaterial RenderModels(x).tmp_mat, y, RenderModel
s(x).Materials(y)
        RenderModels(x).Materials(y).Ambient = RenderModels(x).Materials(y)
.diffuse     'aspect patch!
        RenderModels(x).TexturesNames(y) = UCase(GetFileName(Direct3DX.Buff
erGetTextureName(RenderModels(x).tmp_mat, y)))
        If RenderModels(x).TexturesNames(y) <> "" Then Call AddTexture(Rend
erModels(x).TexturesNames(y))
    Next
Next

DeleteDir mypath
```

```
End Sub

Public Sub LoadCollisionWorld()
Dim mypath As String, x As Long, number As Integer, y As Long, ignore As Lo
ng
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "geo_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "geometry\main.dat", mypath
ShowLoadPercent "17"

For x = 1 To 3
    Set CollisionModels(x) = Direct3DX.LoadMeshFromX(mypath & "main_collisi
on_" & Trim(Str(x)) & ".x", D3DXMESH_MANAGED + D3DXMESH_32BIT, Device, Noth
ing, Nothing, ignore)
Next
ShowLoadPercent "18"

'-- extract all the vertices to form triangles ------
Dim hresult As Long
Dim vertices() As D3DVERTEX
Dim desc As D3DINDEXBUFFER_DESC
Dim IBuf As Direct3DIndexBuffer8
Dim indices() As Long

For x = 1 To 3
    ReDim vertices(CollisionModels(x).GetNumVertices)
    hresult = D3DXMeshVertexBuffer8GetData(CollisionModels(x), 0, Len(verti
ces(0)) * CollisionModels(x).GetNumVertices, 0, vertices(0))

    Set IBuf = CollisionModels(x).GetIndexBuffer()
    IBuf.Lock 0, 0, ignore, 16
    IBuf.GetDesc desc
    IBuf.Unlock
    ReDim indices(desc.Size / 4)   '4 as we use 32 bit mesh , 2 if we use 1
6 bit

    D3DXMeshIndexBuffer8GetData CollisionModels(x), 0, desc.Size, 0, indice
s(0)

    ReDim CollisionFloats(x).vertices(CollisionModels(x).GetNumFaces * 3 -
1)

    For y = 0 To CollisionModels(x).GetNumFaces * 3 - 1 Step 3
        CollisionFloats(x).vertices(y).x = vertices(indices(y)).x
        CollisionFloats(x).vertices(y).y = vertices(indices(y)).y
        CollisionFloats(x).vertices(y).z = vertices(indices(y)).z
        CollisionFloats(x).vertices(y + 1).x = vertices(indices(y + 1)).x
        CollisionFloats(x).vertices(y + 1).y = vertices(indices(y + 1)).y
        CollisionFloats(x).vertices(y + 1).z = vertices(indices(y + 1)).z
        CollisionFloats(x).vertices(y + 2).x = vertices(indices(y + 2)).x
        CollisionFloats(x).vertices(y + 2).y = vertices(indices(y + 2)).y
        CollisionFloats(x).vertices(y + 2).z = vertices(indices(y + 2)).z
    Next

    ReDim indices(0)
    ReDim vertices(0)           'destroy all temp objects!!
```

```
    Set IBuf = Nothing
    Set CollisionModels(x) = Nothing
Next

ShowLoadPercent "23"

DeleteDir mypath
End Sub

Public Sub LoadCollisionFO()
'loads all the collision Fxd obj and adds the floats to the respectives wor
lds
Dim x As Long, y As Long, temparray() As D3DVECTOR, Max As Long

For x = 1 To UBound(CollisionFloats)
    ReDim CollisionFloatsObj(x).vertices(0)
    For y = 1 To UBound(FixedObjects)
        If FixedObjects(y).WorldID = x Then
            ReDim temparray(0)
            If FixedObjects(y).unimesh = 1 Then     'choose the low-poly on
e when possible
                ExtractTriVec FOComplex(FixedObjects(y).MeshID).Mesh, tempa
rray
            Else
                ExtractTriVec FOSimple(FixedObjects(y).MeshID).Mesh, tempar
ray
            End If
            'transform vertices to world space!
            D3DXMatrixTranslation TMatrix, FixedObjects(y).Position.x, Fixe
dObjects(y).Position.y, FixedObjects(y).Position.z
            TransformVerts temparray, TMatrix
            AddToArray temparray, CollisionFloatsObj(x).vertices
        End If
    Next

    'now we have filled  CollisionFloatsObj(worldid) so add to the global a
rray
    AddToArray CollisionFloatsObj(x).vertices, CollisionFloats(x).vertices
    If UBound(CollisionFloats(x).vertices) > Max Then Max = UBound(Collisio
nFloats(x).vertices)
    ReDim CollisionFloatsObj(x).vertices(0)     'return memory
Next
ReDim temparray(0)
ReDim CollisionFloatsAux(Max)
End Sub

Public Sub LoadMainCharAnims()
Dim mypath As String
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "mainchar_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\
"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "graphics\mainchar.dat", mypath
ShowLoadPercent "25"

Dim cad As String
```

```
'------ Walking ------
LoadAnim3D MainCharWalking, mypath, "mainchar_walking"
ShowLoadPercent "29"

LoadAnim3D MainCharWalkingStart, mypath, "mainchar_startwalking"
ShowLoadPercent "32"

LoadAnim3D MainCharWalkingEnd, mypath, "mainchar_endwalking"
ShowLoadPercent "33"

'----- Standing -----
LoadAnim3D MainCharStanding, mypath, "mainchar_standing"
ShowLoadPercent "34"

DeleteDir mypath
End Sub

Public Sub LoadMainCharAnimsShadows()
ExtractTriVec MainCharStanding.Meshes(1), MainCharStandingShadowVerts
ComputeNormals MainCharStandingShadowVerts, MainCharStandingShadowNormals
Set MainCharStandingShadowClass = New clsShadow
End Sub

Public Sub LoadWorldShadows()
'create vertices and triangles and split them into boxes
Dim ShadowNode As clsCullNode
Dim CX As Single, CZ As Single
Dim x As Double, y As Double, z As Double
Dim yy As Double, zz As Double
Dim Min As Double
Dim CurrentW() As D3DVECTOR, tarray1() As D3DVECTOR, normals() As D3DVECTOR
Dim center As D3DVECTOR, conta As Long, conta2 As Long, light As D3DVECTOR

For x = 1 To 3          'for each world
    ReDim CurrentW(0)
    ReDim WorldShadowsCull(x).CullArray(0)
    WorldShadowsCull(x).NumCull = 0
    For y = 1 To UBound(RenderModels)       'for each mesh to render
        If MeshOfWorld(y, x) Then
            ReDim tarray1(0)
            ExtractTriVec RenderModels(y).Mesh, tarray1
            AddToArray tarray1, CurrentW
        End If
    Next

    'we have generated the array
    yy = DetermineMinX(CurrentW)
    Min = DetermineMinZ(CurrentW)
    CX = DetermineMaxX(CurrentW)
    CZ = DetermineMaxZ(CurrentW)
    ShadowedWorlds(x).MinHeight = Min
    ShadowedWorlds(x).MinWidth = yy

    conta2 = 0
    Do While yy <= CX
        zz = Min
        conta = 0
        Do While zz <= CZ
            conta = conta + 1
            WorldShadowsCull(x).NumCull = WorldShadowsCull(x).NumCull + 1
```

```
            ReDim Preserve WorldShadowsCull(x).CullArray(WorldShadowsCull(x
).NumCull)
            Set WorldShadowsCull(x).CullArray(WorldShadowsCull(x).NumCull)
= New clsCullNode
            WorldShadowsCull(x).CullArray(WorldShadowsCull(x).NumCull).Cube
Position = v3(yy + ShadowCullSize, 0, zz + ShadowCullSize)
            WorldShadowsCull(x).CullArray(WorldShadowsCull(x).NumCull).heig
ht = ShadowCullSize
            WorldShadowsCull(x).CullArray(WorldShadowsCull(x).NumCull).widt
h = ShadowCullSize
            WorldShadowsCull(x).CullArray(WorldShadowsCull(x).NumCull).Buil
dNodeFromVerts CurrentW
            zz = zz + ShadowCullSize * 2
        Loop
        yy = yy + ShadowCullSize * 2
        conta2 = conta2 + 1
    Loop
    ShadowedWorlds(x).HeightBlocks = conta
    ShadowedWorlds(x).WidthBlocks = conta2
Next
ReDim tarray1(0)
ReDim CurrentW(0)
End Sub

Public Sub LoadWorldShadows2()
Dim x As Long, y As Long, light As D3DVECTOR
Dim arrayv() As D3DVECTOR
For x = 1 To 2
    For y = 1 To UBound(Lights)
        If Lights(y).CastShadows = True Then
            light = Lights(y).ShadowPoint
        End If
    Next
    For y = 1 To WorldShadows(x).NumShadows
        Set WorldShadows(x).ShadowsArray(y) = Nothing
    Next
    WorldShadows(x).NumShadows = WorldShadowsCull(x).NumCull
    ReDim WorldShadows(x).ShadowsArray(WorldShadows(x).NumShadows)
    For y = 1 To WorldShadowsCull(x).NumCull
        If WorldShadowsCull(x).CullArray(y).NumOfVertices <> 0 Then
            Set WorldShadows(x).ShadowsArray(y) = New clsShadow
            WorldShadowsCull(x).CullArray(y).GetVertices arrayv
            WorldShadows(x).ShadowsArray(y).LightProj = 25
            WorldShadows(x).ShadowsArray(y).Build arrayv, light
            ReDim arrayv(0)
        End If
    Next
Next
End Sub

Public Sub LoadSkyBox()
Dim mypath As String
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "sky_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "maps\sky.dat", mypath
ShowLoadPercent "50"
```

```
'assume no compression is activated in order to avoid some view issues in t
he sky
Set SkyBoxTextures(0) = LoadTextureAndReturn(mypath & "day_front.bmp", Fals
e)
Set SkyBoxTextures(1) = LoadTextureAndReturn(mypath & "day_right.bmp", Fals
e)
Set SkyBoxTextures(2) = LoadTextureAndReturn(mypath & "day_back.bmp", False
)
Set SkyBoxTextures(3) = LoadTextureAndReturn(mypath & "day_left.bmp", False
)

Set SkyBoxTextures(4) = LoadTextureAndReturn(mypath & "aft_front.bmp", True
)
Set SkyBoxTextures(5) = LoadTextureAndReturn(mypath & "aft_right.bmp", True
)
Set SkyBoxTextures(6) = LoadTextureAndReturn(mypath & "aft_back.bmp", True)
Set SkyBoxTextures(7) = LoadTextureAndReturn(mypath & "aft_left.bmp", True)

Set SkyBoxTextures(8) = LoadTextureAndReturn(mypath & "night_front.bmp", Tr
ue)
Set SkyBoxTextures(9) = LoadTextureAndReturn(mypath & "night_right.bmp", Tr
ue)
Set SkyBoxTextures(10) = LoadTextureAndReturn(mypath & "night_back.bmp", Tr
ue)
Set SkyBoxTextures(11) = LoadTextureAndReturn(mypath & "night_left.bmp", Tr
ue)

DeleteDir mypath

ShowLoadPercent "55"

SkyBoxVertices(0) = AssignMVS(-0.5, 0.6, 0.5, 0.001, 0.001)
SkyBoxVertices(1) = AssignMVS(0.5, 0.6, 0.5, 0.999, 0.001)
SkyBoxVertices(2) = AssignMVS(-0.5, 0, 0.5, 0.001, 0.999)
SkyBoxVertices(3) = AssignMVS(0.5, 0, 0.5, 0.999, 0.999)

SkyBoxVertices(12) = AssignMVS(-0.5, 0.6, -0.5, 0.001, 0.001)
SkyBoxVertices(13) = AssignMVS(-0.5, 0.6, 0.5, 1, 0.001)
SkyBoxVertices(14) = AssignMVS(-0.5, 0, -0.5, 0.001, 0.999)
SkyBoxVertices(15) = AssignMVS(-0.5, 0, 0.5, 0.999, 0.999)

SkyBoxVertices(8) = AssignMVS(0.5, 0.6, -0.5, 0.001, 0.001)
SkyBoxVertices(9) = AssignMVS(-0.5, 0.6, -0.5, 1, 0.001)
SkyBoxVertices(10) = AssignMVS(0.5, 0, -0.5, 0.001, 0.999)
SkyBoxVertices(11) = AssignMVS(-0.5, 0, -0.5, 0.999, 0.999)

SkyBoxVertices(4) = AssignMVS(0.5, 0.6, 0.5, 0.001, 0.001)
SkyBoxVertices(5) = AssignMVS(0.5, 0.6, -0.5, 0.999, 0.001)
SkyBoxVertices(6) = AssignMVS(0.5, 0, 0.5, 0.001, 0.999)
SkyBoxVertices(7) = AssignMVS(0.5, 0, -0.5, 0.999, 0.999)

SkyBoxVertices(16) = AssignMVS(-0.5, 0.6, -0.5, 0.01, 0.01)
SkyBoxVertices(17) = AssignMVS(0.5, 0.6, -0.5, 0.999, 0.01)
SkyBoxVertices(18) = AssignMVS(-0.5, 0.6, 0.5, 0.01, 0.999)
SkyBoxVertices(19) = AssignMVS(0.5, 0.6, 0.5, 0.999, 0.999)
End Sub

Public Sub LoadFixedObjects()
Dim mypath As String
```

MainModule - 21

```
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "fobj_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "graphics\obj.dat", mypath

Dim f As Integer, cad As String, lon As Long, lon2 As Long, sin As Single,
x As Long, WorldID As Integer
Dim posv As D3DVECTOR, file1 As String, file2 As String, AutoY As Boolean,
roty As Single
f = FreeFile
Open mypath & "obj.dat" For Binary As #f
    cad = Space(Len("FODATAFILE"))
    Get #f, , cad
    Get #f, , lon
    ReDim FixedObjects(lon)
    For x = 1 To lon
        Get #f, , sin
        posv.x = sin
        Get #f, , sin
        posv.y = sin
        If sin = -9999 Then
            AutoY = True
        Else
            AutoY = False
        End If
        Get #f, , sin
        posv.z = sin
        Get #f, , sin
        roty = sin
        Get #f, , lon2
        cad = Space(lon2)
        Get #f, , cad
        file1 = mypath & cad
        Get #f, , lon2
        cad = Space(lon2)
        Get #f, , cad
        file2 = mypath & cad
        Get #f, , WorldID
        FixedObjects(x) = LoadFixedObject(posv, file1, file2, AutoY, roty,
WorldID)
    Next
Close #f

Set MissionTargetModel.Mesh = Direct3DX.LoadMeshFromX(mypath & "target.x",
D3DXMESH_MANAGED, Device, MissionTargetModel.tmp_adj, MissionTargetModel.tm
p_mat, MissionTargetModel.NumMaterials)
OptimizeMesh MissionTargetModel.Mesh, MissionTargetModel.tmp_adj

ReDim MissionTargetModel.Materials(MissionTargetModel.NumMaterials - 1)
ReDim MissionTargetModel.TexturesNames(MissionTargetModel.NumMaterials - 1)

For x = 0 To MissionTargetModel.NumMaterials - 1
    Direct3DX.BufferGetMaterial MissionTargetModel.tmp_mat, x, MissionTarge
tModel.Materials(x)
    MissionTargetModel.Materials(x).Ambient = MissionTargetModel.Materials(
x).diffuse
    MissionTargetModel.TexturesNames(x) = UCase(GetFileName(Direct3DX.Buffe
```

```
rGetTextureName(MissionTargetModel.tmp_mat, x)))
    If MissionTargetModel.TexturesNames(x) <> "" Then Call AddTexture(Missi
onTargetModel.TexturesNames(x))
Next
Set MissionTargetModel.tmp_mat = Nothing
Set MissionTargetModel.tmp_adj = Nothing

DeleteDir mypath
End Sub

Public Sub LoadCommonSounds()
'-------------- INIT DIRECT SOUND ----------
DirectSound.SetCooperativeLevel frmGraphics.hWnd, DSSCL_PRIORITY
DSPrimaryBufferDesc.lFlags = DSBCAPS_CTRL3D Or DSBCAPS_PRIMARYBUFFER
Set DSPrimaryBuffer = DirectSound.CreatePrimarySoundBuffer(DSPrimaryBufferD
esc)
Set DSListener = DSPrimaryBuffer.GetDirectSound3DListener()
'-------------------------------------------
Dim mypath As String, x As Integer
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "sound_tmp_" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "sound\main.dat", mypath

'----- Global Sounds ------
CreateStandardSound mypath & "pasos.wav", "pasos_mainchar"

DeleteDir mypath
End Sub


Public Sub LoadSM()
Dim mypath As String
mypath = TempPath()
If right(mypath, 1) <> "\" Then mypath = mypath & "\"
mypath = mypath & "fobj_tmp" & Format(Int(Rnd * 5000) + 1, "0000") & "\"
On Local Error Resume Next
MkDir mypath
On Local Error GoTo 0
ExtractFile EXE & "maps\stage.dat", mypath

Set LSBar = LoadTextureAndReturn(mypath & "bar.bmp")

DeleteDir mypath
End Sub
```

MusicEngine - 1

```
'------ Plays OGG streams using StreamOGG and clsDLL classes ------
'------ Uses Direct Sound 8 interface and buffer streaming --------
'------ by davidgf. Thanks to DX9 SDK and [rm_code] ---------

Public Volume As Long
Public MusicStatus As Integer      '1 play, 0 stop, 2 pause

Private Const MYBUFFERSIZE As Long = 600000
Private Const MYBUFFERPARTSIZE As Long = 200000
Private Const B1_B2 As Long = 200000
Private Const B2_B3 As Long = 400000

Private MusicBuffer As DirectSoundSecondaryBuffer8
Private bytes(MYBUFFERSIZE) As Byte
Private ebytes(MYBUFFERPARTSIZE) As Byte

Private BufferCursor As DSCURSORS
Private Out As SND_RESULT
Private Read As Long
Private PauseCursor As Long

Private OGGLib As StreamOGG

Private w1 As Boolean, w2 As Boolean, w3 As Boolean
Private CountOut As Integer

Public Sub PlayMusic(ByVal filename As String)
OGGLib.StreamClose

Out = OGGLib.StreamOpen(filename)

MusicBuffer.SetFrequency OGGLib.SamplesPerSecond

Out = OGGLib.StreamRead(VarPtr(bytes(0)), MYBUFFERSIZE, Read)
MusicBuffer.WriteBuffer 0, MYBUFFERSIZE, bytes(0), DSBLOCK_DEFAULT
MusicBuffer.SetCurrentPosition 0
MusicBuffer.Play DSBPLAY_LOOPING

MusicStatus = 1

w1 = False
w2 = True
w3 = True
End Sub

Public Sub StopMusic()
If Not MusicBuffer Is Nothing Then MusicBuffer.Stop
If Not OGGLib Is Nothing Then OGGLib.StreamClose
MusicStatus = 0
End Sub

Public Sub RenderTime()
'this function MUST be called at least once a second to allow the prog to r
efresh
'the sound buffer. if not some sound spikes or bad music looping will be he
ard
'       splits the buffer into 3 areas for fast music streaming
'       loads the music as it is played
Static lastVol As Long
```

MusicEngine - 2

```
If Not MusicBuffer Is Nothing And MusicStatus = 1 Then
    If Volume <> lastVol Then MusicBuffer.SetVolume Volume

    MusicBuffer.GetCurrentPosition BufferCursor

    If BufferCursor.lPlay > B1_B2 And BufferCursor.lPlay < B2_B3 And w1 = F
alse Then
        If ContOut <> 0 Then
            ContOut = ContOut + 1
            MusicBuffer.WriteBuffer 0, MYBUFFERPARTSIZE, ebytes(0), DSBLOCK
_DEFAULT
            If ContOut = 4 Then GoTo SoundEnd
        Else
            Out = OGGLib.StreamRead(VarPtr(bytes(0)), MYBUFFERPARTSIZE, Rea
d)
            MusicBuffer.WriteBuffer 0, MYBUFFERPARTSIZE, bytes(0), DSBLOCK_
DEFAULT
            If Read < MYBUFFERPARTSIZE Then
                MusicBuffer.WriteBuffer Read, (MYBUFFERPARTSIZE - Read), eb
ytes(0), DSBLOCK_DEFAULT
                ContOut = ContOut + 1
            End If
        End If
        w1 = True
        w2 = False
        w3 = False
    End If

    If BufferCursor.lPlay > B2_B3 And w2 = False Then
        If ContOut <> 0 Then
            ContOut = ContOut + 1
            MusicBuffer.WriteBuffer B1_B2, MYBUFFERPARTSIZE, ebytes(0), DSB
LOCK_DEFAULT
            If ContOut = 4 Then GoTo SoundEnd
        Else
            Out = OGGLib.StreamRead(VarPtr(bytes(0)), MYBUFFERPARTSIZE, Rea
d)
            MusicBuffer.WriteBuffer B1_B2, MYBUFFERPARTSIZE, bytes(0), DSBL
OCK_DEFAULT
            If Read < MYBUFFERPARTSIZE Then
                MusicBuffer.WriteBuffer Read + B1_B2, (MYBUFFERPARTSIZE - R
ead), ebytes(0), DSBLOCK_DEFAULT
                ContOut = ContOut + 1
            End If
        End If

        w1 = False
        w2 = True
        w3 = False
    End If

    If BufferCursor.lPlay < B1_B2 And w3 = False Then
        If ContOut <> 0 Then
            ContOut = ContOut + 1
            MusicBuffer.WriteBuffer B2_B3, MYBUFFERPARTSIZE, ebytes(0), DSB
LOCK_DEFAULT
            If ContOut = 4 Then GoTo SoundEnd
        Else
            Out = OGGLib.StreamRead(VarPtr(bytes(0)), MYBUFFERPARTSIZE, Rea
```

```
d)
            MusicBuffer.WriteBuffer B2_B3, MYBUFFERPARTSIZE, bytes(0), DSBL
OCK_DEFAULT
            If Read < MYBUFFERPARTSIZE Then
                MusicBuffer.WriteBuffer Read + B2_B3, (MYBUFFERPARTSIZE - R
ead), ebytes(0), DSBLOCK_DEFAULT
                ContOut = ContOut + 1
            End If
        End If
        w1 = False
        w2 = False
        w3 = True
    End If
End If
Exit Sub

SoundEnd:
MusicBuffer.Stop
MusicStatus = 0
End Sub

Public Sub InitMusic()
Dim soundDESC As DSBUFFERDESC
soundDESC.lBufferBytes = MYBUFFERSIZE
soundDESC.lFlags = DSBCAPS_CTRLVOLUME + DSBCAPS_CTRLFREQUENCY
Set MusicBuffer = DirectSound.CreateSoundBuffer(soundDESC)

Set OGGLib = New StreamOGG
End Sub

Public Sub Pause()
MusicBuffer.Stop
MusicStatus = 2
End Sub

Public Sub ResumePlay()
MusicBuffer.Play DSBPLAY_LOOPING
MusicStatus = 1
End Sub

Public Sub DestroyMusic()
On Local Error Resume Next
OGGLib.StreamClose
Set OGGLib = Nothing
Set MusicBuffer = Nothing
End Sub

Public Sub LoadOGGBuffer(ByVal fileogg As String, ByRef buffer As DirectSou
ndSecondaryBuffer8)
Dim SOgg As StreamOGG
Dim hr As SND_RESULT
Dim bbytes() As Byte
hr = SOgg.StreamOpen(fileogg)
ReDim bbytes(SOgg.BufferSizeBytes)
SOgg.StreamRead VarPtr(bbytes(0)), SOgg.BufferSizeBytes, 0

Dim soundDESC As DSBUFFERDESC
soundDESC.lBufferBytes = SOgg.BufferSizeBytes
soundDESC.lFlags = DSBCAPS_CTRLVOLUME + DSBCAPS_CTRLFREQUENCY
Set buffer = DirectSound.CreateSoundBuffer(soundDESC)
```

```
buffer.WriteBuffer 0, SOgg.BufferSizeBytes, bbytes(0), DSBLOCK_ENTIREBUFFER
buffer.SetFrequency SOgg.BitsPerSample
End Sub
```

SoundEngine - 1

```
Option Explicit

'------ 3D SOUND ----------

Public Sub UpdateListenerSettings()
DSListener.SetPosition CharPos.x, CharPos.y, CharPos.z, DS3D_DEFERRED
DSListener.SetOrientation sin(CharAngleH * Pi / 180), 0, Cos(CharAngleH * P
i / 180), 0, 1, 0, DS3D_DEFERRED
DSListener.CommitDeferredSettings
End Sub

Public Sub CreateSound(ByVal file As String, ByVal soundname As String)
Dim index As Integer
index = UBound(DS_Sounds) + 1
ReDim Preserve DS_Sounds(index)

DS_Sounds(index).desc.lFlags = DSBCAPS_CTRL3D + DSBCAPS_CTRLVOLUME
DS_Sounds(index).desc.guid3DAlgorithm = GUID_DS3DALG_DEFAULT
Set DS_Sounds(index).buffersec = DirectSound.CreateSoundBufferFromFile(file
, DS_Sounds(index).desc)
DS_Sounds(index).buffersec.SetVolume 0

Set DS_Sounds(index).buffer3d = DS_Sounds(index).buffersec.GetDirectSound3D
Buffer()
DS_Sounds(index).buffer3d.SetConeAngles 90, 160, DS3D_IMMEDIATE
DS_Sounds(index).buffer3d.SetConeOutsideVolume -50, DS3D_IMMEDIATE
DS_Sounds(index).buffer3d.SetMaxDistance 15, DS3D_IMMEDIATE
DS_Sounds(index).buffer3d.SetMinDistance 3, DS3D_IMMEDIATE

DS_Sounds(index).file = file
DS_Sounds(index).soundname = soundname
End Sub

Public Sub PlaySound(ByVal soundname As String, pos As D3DVECTOR, orient As
 D3DVECTOR)
Dim x As Integer, orient2 As D3DVECTOR
orient2 = Normalize(orient)
For x = 1 To UBound(DS_Sounds)
    If DS_Sounds(x).soundname = soundname Then
        DS_Sounds(x).buffer3d.SetConeOrientation orient2.x, orient2.y, orie
nt2.z, DS3D_IMMEDIATE
        DS_Sounds(x).buffer3d.SetPosition pos.x, pos.y, pos.z, DS3D_IMMEDIA
TE

        DS_Sounds(x).buffersec.Stop
        DS_Sounds(x).buffersec.SetCurrentPosition 0
        DS_Sounds(x).buffersec.Play DSBPLAY_DEFAULT
        Exit Sub
    End If
Next
End Sub

Public Sub StopSound(ByVal soundname As String)
Dim x As Integer
For x = 1 To UBound(DS_Sounds)
    If DS_Sounds(x).soundname = soundname Then
        DS_Sounds(x).buffersec.Stop
        Exit Sub
    End If
Next
```

```
SoundEngine - 2

End Sub

Public Function SoundStopped(ByVal soundname As String) As Boolean
Dim x As Integer, cur As DSCURSORS
For x = 1 To UBound(DS_Sounds)
    If DS_Sounds(x).soundname = soundname Then
        DS_Sounds(x).buffersec.GetCurrentPosition cur
        If cur.lPlay = 0 Then SoundStopped = True
        Exit Function
    End If
Next
End Function

Public Sub StopAllSounds()
Dim x As Integer
For x = 1 To UBound(DS_Sounds)
    DS_Sounds(x).buffersec.Stop
Next
End Sub

Public Sub DestroySounds()
On Local Error Resume Next
Call StopAllSounds
Dim x As Integer
For x = 1 To UBound(DS_Sounds)
    Set DS_Sounds(x).buffer3d = Nothing
    Set DS_Sounds(x).buffersec = Nothing
Next
ReDim DS_Sounds(0)
End Sub

'------ Standard SOUND ----------

Public Sub CreateStandardSound(ByVal file As String, ByVal soundname As Str
ing)
Dim index As Integer
index = UBound(DS_Sounds_Plain) + 1
ReDim Preserve DS_Sounds_Plain(index)

DS_Sounds_Plain(index).desc.lFlags = DSBCAPS_CTRLPAN Or DSBCAPS_CTRLVOLUME
Set DS_Sounds_Plain(index).buffer = DirectSound.CreateSoundBufferFromFile(f
ile, DS_Sounds_Plain(index).desc)
DS_Sounds_Plain(index).file = file
DS_Sounds_Plain(index).soundname = soundname
End Sub

Public Sub PlayStandardSound(ByVal soundname As String, ByVal Looping As Bo
olean, Optional ByVal Reset As Boolean = True)
Dim x As Integer
For x = 1 To UBound(DS_Sounds_Plain)
    If DS_Sounds_Plain(x).soundname = soundname Then
        If Looping = False Then
            DS_Sounds_Plain(x).buffer.Play DSBPLAY_DEFAULT
        Else
            DS_Sounds_Plain(x).buffer.Play DSBPLAY_LOOPING
        End If
        Exit Sub
    End If
Next
End Sub
```

```
Public Sub StopStandardSound(ByVal soundname As String)
Dim x As Integer
For x = 1 To UBound(DS_Sounds_Plain)
    If DS_Sounds_Plain(x).soundname = soundname Then
        DS_Sounds_Plain(x).buffer.Stop
    End If
Next
End Sub

Public Sub DestroyStandardSound()
Dim x As Integer
For x = 1 To UBound(DS_Sounds_Plain)
    Set DS_Sounds_Plain(x).buffer = Nothing
Next
Erase DS_Sounds_Plain
ReDim DS_Sounds_Plain(0)
End Sub

Public Function IsPlayingStSound(ByVal sndname As String) As Boolean
Dim x As Integer, st As Long
For x = 1 To UBound(DS_Sounds_Plain)
    If DS_Sounds_Plain(x).soundname = sndname Then
        st = DS_Sounds_Plain(x).buffer.GetStatus()
        If (st & DSBSTATUS_PLAYING) <> 0 Then IsPlayingStSound = True
        Exit Function
    End If
Next
End Function
```

Variables - 1

```
Option Explicit

'---------------- GLOBALS --------------
Global EXE As String
Global Reg As clsWinReg

Global ExitToWin As Boolean

Public Const RegName = "tarraco"

'------------- FILE PARSING ------------
Global PNames() As String
Global PValues() As String

'------------------- DX ---------------
Global DirectX As DirectX8
Global Direct3D As Direct3D8
Global Direct3DX As D3DX8
Global DirectInput As DirectInput8
Global DirectSound As DirectSound8
Global caps As D3DCAPS8

'---------------- DS ---------------------
Global DS_Sounds() As Sound
Global PendingSounds() As String
Global DS_Sounds_Plain() As SoundStandard

Global DisableSound As Boolean
Global MusicDir As String

Global DSPrimaryBuffer As DirectSoundPrimaryBuffer8
Global DSPrimaryBufferDesc As DSBUFFERDESC

Global DSListener As DirectSound3DListener8

'----------------- DI -------------------
Global Device As Direct3DDevice8
Global MouseDevice As DirectInputDevice8
Global DI_hevent As Long

Global MouseX As Single, MouseY As Single, MouseZ As Single
Global MouseB0 As Boolean, MouseB1 As Boolean
Global MouseClick0 As Boolean, MouseClick1 As Boolean
Global MouseTexture As Direct3DTexture8, MouseVerts(3) As myVertex
Global CursorW As Single, CursorH As Single

'-------------------- D3D ---------------
Global MainRender As Boolean
Global AuxMenu As Boolean

Global D3DM As D3DDISPLAYMODE
Global AntiAliasLevel As Long
Global DModesArray() As D3DDISPLAYMODE
Global ResolutionArray As Variant
Global RenderStates(20) As Long

Global ShadowsAvaliable As Boolean
Global EnableShadows As Integer

'----------------- DRAW SETTINGS --------------
```

Variables - 2

```
Global DrawDepth As Integer  '1, 2, 3
Global DistanceFOAppear As Single
Global DistanceFODetail As Single
Global FarViewPlane As Single
Global CursorSpeed As Integer  '1, 2, 3
Global CharDetail As Integer  '1, 2, 3
Global TexQuality As Integer    '1 Low, 2 Normal

'------------------- MATHS -----------------
Global projMatrix As D3DMATRIX
Global projMatrixShadows As D3DMATRIX
Global viewMatrix As D3DMATRIX
Global FrustumPlanes(7) As D3DPLANE

'---------------------- RESOURCES -----------------
Global GameTextures() As TextureEx
Global GameTexturesSec() As TextureEx
Global PendingTextures() As String
Global FontTexture As Direct3DTexture8
Global FontVertices(3) As myVertex
Global SkyBoxTextures(11) As Direct3DTexture8
Global SkyBoxVertices(19) As myVertexSimple
Global MissionTargets() As MissionTarget
Global MissionTargetModel As Model3D

'---------------------- PREV ----------------------
Global LTextMain As Direct3DTexture8
Global LNum As Direct3DTexture8
Global LVertices(19) As myVertex

'-------------------- MENU ------------------
Global MainMenu As cGameMenu
Global ConfigMenu As cGameMenu
Global LoadGameMenu As cGameMenu
Global AuxiliarMenu As cGameMenu
Global GlobalMenuOption As Integer
Global fadeVertices(3) As myVertexAlpha

'------------------- VIDEO -------------------
Global VideoOn As String

'-------------- LOAD STAGE SCREEN --------------
Global LSBar As Direct3DTexture8
Global LSVert(3) As myVertex

'-------------------- GAME ----------------------
    '-------- GAME FLOW ---------
    Global SlotID As Long
    Global TheGameSlot As GameSlot

    Global LevelScenes() As Scene

    Global DisableDoors As Boolean
    Global FadeState As Integer
    Global FadeTimeMS As Single
    '----------------------- WORLD -----------------
    Global RenderModels(1 To 5) As Model3D
    Global WorldProperties(1 To 3) As WorldProp                 'world desc
riptor
    Global CollisionModels(1 To 3) As D3DXMesh                  'temp model
```

```
Variables - 3

s
    Global CollisionFloats(1 To 3) As CollisionFloat            'collision
triangles
    Global CollisionFloatsAux() As Long                         'long aux a
rray for engine.dll
    Global CollisionFloatsObj(1 To 3) As CollisionFloat         'temp array
 for fo

    Global MakeFade1 As Boolean, MakeFade2 As Boolean
    Global ResetAllMoves As Boolean

        'fixed objects arrays
        Global FixedObjects() As FixedObject        'at startup
        Global FixedObjectsStage() As FixedObject   'each scene

        'array for the startup
        Global FOComplex() As Model3D
        Global FOSimple() As Model3D

        'array for the scene (celaned and reloaded each scene)
        Global FOComplexStage() As Model3D
        Global FOSimpleStage() As Model3D


    Global WorldShadows(1 To 3) As WorldShadow
    Global WorldShadowsCull(1 To 3) As WorldShadowCull
    Global ShadowedWorlds(1 To 3) As ShadowedWorld
    Global ShadowVertices(3) As myVertexAlpha
    Global ShadowCullSize As Single

    Global Lights() As myLight

    '-------------------- MAINCHAR --------------------
    Global CharPos As D3DVECTOR             'position now
    Global CharPosBefore As D3DVECTOR       'posistion last frame
    Global CharAngleH As Single             'rotation angle for the char
    Global CharAngleV As Single             'cam vertical angle
    Global CharDistance As Single           'camera - char distance
    Global CharOrientation As D3DVECTOR     'where is looking to
    Global CharState As Integer             'state animation (walk, run...)

    Global MainCharStanding As Anim3D
    Global MainCharWalking As Anim3D
    Global MainCharWalkingStart As Anim3D
    Global MainCharWalkingEnd As Anim3D
    Global MainCharRunning As Anim3D
    Global MainCharStairsUp As Anim3D

    Global MainCharStandingShadowVerts() As D3DVECTOR
    Global MainCharStandingShadowClass As clsShadow
    Global MainCharStandingShadowNormals() As D3DVECTOR

    '--------------- OTHER CHARS ---------------
    Global CharSoldiers(5) As clsDynObj
    Global MovingCharacters() As clsDynObj

    Global OtherCharAnimations() As Anim3D

    '--------------- CAMERA -------------------
    Global CameraPos As D3DVECTOR
```

Variables - 4

```
    Global vectorUP As D3DVECTOR

    '--------------- PHYSICS ---------------
    Global Movement As Single
    Global Jumping As Boolean
    Global InitY As Single
    Global AccelerationTimer As Double

    Global TMatrix As D3DMATRIX

    Global TimeCounter As Double
    Global FrameAverage As Double

    Global CharSpeed As Single
    Global Stopped As Boolean
    Global EndingAnim As Boolean

    Global CurrentCharAnimation As String
    Global CurrentCharAnimationFrame As Single

    Global DynObjPositions() As D3DVECTOR
    Global NumDynObjPositions As Long
```

VideoEngine - 1

```
'-----------------------------------------------------------------
' Video engine by davidgf for use with theora ogg video format '
' Requires Theora Direct Show Filter (ex: illiminable/ogg)     '
' Uses Direct Show                                             '
' only 1 video at the same time                                '
'-----------------------------------------------------------------

Private Started As Boolean
Private StartTime As Double

Const WS_CHILD = &H40000000
Const WS_CLIPSIBLINGS = &H4000000

Private m_objBasicAudio  As IBasicAudio      'Basic Audio Object
Private m_objBasicVideo As IBasicVideo       'Basic Video Object
Private m_objMediaEvent As IMediaEvent       'MediaEvent Object
Private m_objVideoWindow As IVideoWindow     'VideoWindow Object
Private m_objMediaControl As IMediaControl   'MediaControl Object
Private m_objMediaPosition As IMediaPosition 'MediaPosition Object

Public Sub OpenVideo(ByVal VideoFile As String)
Set m_objMediaControl = New FilgraphManager
Call m_objMediaControl.RenderFile(VideoFile)

Set m_objBasicAudio = m_objMediaControl
m_objBasicAudio.Volume = 0
m_objBasicAudio.Balance = 0

Set m_objVideoWindow = m_objMediaControl
m_objVideoWindow.WindowStyle = CLng(&H10000000) Or CLng(WS_CHILD)

m_objVideoWindow.left = 0
m_objVideoWindow.top = 0
Dim left As Long, top As Long, width As Long, height As Long
m_objVideoWindow.GetWindowPosition left, top, width, height
m_objVideoWindow.SetWindowPosition left, top, D3DM.width, D3DM.height
m_objVideoWindow.Owner = frmGraphics.hWnd

Set m_objMediaEvent = m_objMediaControl

Set m_objMediaPosition = m_objMediaControl

m_objMediaPosition.rate = 1

Started = False
End Sub

Private Function LongToShort(ByVal Str As String) As String
Dim buffer As String, ret As Long
buffer = Space(512)
ret = GetShortPathName(Str, buffer, 512)
LongToShort = left(buffer, InStr(1, buffer, vbNullChar) - 1)
End Function

Public Sub PlayVideo()
m_objMediaPosition.CurrentPosition = 0

Call m_objMediaControl.Run

Started = True
```

```
StartTime = GetTickCount()
End Sub

Public Sub StopVideo()
m_objMediaControl.Stop
End Sub

Public Function VideoStatus() As String
If Started Then
    If GetTickCount() - StartTime > m_objMediaPosition.Duration * 1000 Then
        VideoStatus = "stopped"
    Else
        VideoStatus = "playing"
    End If
Else
    VideoStatus = "playing"
End If
End Function

Public Sub CloseVideo()
m_objVideoWindow.Owner = 0          'sets the Owner to NULL
m_objVideoWindow.Visible = False

    'clean-up & dereference
If Not m_objBasicAudio Is Nothing Then Set m_objBasicAudio = Nothing
If Not m_objBasicVideo Is Nothing Then Set m_objBasicVideo = Nothing
If Not m_objMediaControl Is Nothing Then Set m_objMediaControl = Nothing
If Not m_objVideoWindow Is Nothing Then Set m_objVideoWindow = Nothing
If Not m_objMediaPosition Is Nothing Then Set m_objMediaPosition = Nothing
End Sub
```

cGameMenu - 1

```
Option Explicit

'-------- All coordinates in a menu base of 800 x 600 ---------

Public isEvent As Boolean
Public EventName As String
Public EventValue As String
Public BackGround As String
Public ListTexture As String
Public ListTextureOpen As String
Public ListTextureOpenHover As String
Public TexPath As String

Private Textures() As Direct3DTexture8
Private Textures2() As Direct3DTexture8
Private Textures3() As Direct3DTexture8
Private BGTex As Direct3DTexture8
Private ListTex As Direct3DTexture8
Private ListTexOpen As Direct3DTexture8
Private ListTexOpenHover As Direct3DTexture8

Private Textures3Names() As String

Private Type myButton
    orX As Single
    orY As Single
    orWidth As Single
    orHeight As Single
    '---- transformed ----
    x As Single
    y As Single
    width As Single
    height As Single
    '--- other ----
    texture As String
    texturehover As String
    ColorK As Long
    b_name As String
    isover As Boolean
End Type

Private Type myList
    orX As Single
    orY As Single
    orWidth As Single
    orHeight As Single
    orFontHeight As Single
    '---- transformed ----
    x As Single
    y As Single
    width As Single
    height As Single
    Fontheight As Single
    '----- other ------
    list() As String
    listover() As Byte
    l_name As String
    maxrows As Integer
    startelement As Integer
    selected As Integer
```

```
    scroll As Boolean
End Type

Private Type myLabel
    orX As Single
    orY As Single
    orFontWidth As Single
    orFontHeight As Single
    '---- transformed ----
    x As Single
    y As Single
    Fontwidth As Single
    Fontheight As Single
    '------------------
    text As String
End Type

Private Type myImage
    orX As Single
    orY As Single
    orWidth As Single
    orHeight As Single
    '---- transformed ----
    x As Single
    y As Single
    width As Single
    height As Single
    '---------------------
    texture As String
    ColorK As Long
End Type

Private bgvertices(3) As myVertex
Private numLists As Integer
Private arrayLists() As myList
Private numButtons As Integer
Private arrayButtons() As myButton
Private numLabels As Integer
Private arrayLabels() As myLabel
Private numImages As Integer
Private arrayImages() As myImage

Private aListIsOpened As Boolean
Private ListOpenedIndex As Integer

Private PublicVertices(3) As myVertex

Public Sub AddButton(ByVal x As Single, ByVal y As Single, ByVal width As S
ingle, ByVal height As Single, ByVal texture As String, ByVal texturehover
As String, ByVal ButtonName As String, ByVal ColorK As Long)
numButtons = numButtons + 1
Dim current As Integer
current = numButtons - 1
ReDim Preserve arrayButtons(current)
arrayButtons(current).orX = x
arrayButtons(current).orY = y
arrayButtons(current).orWidth = width
arrayButtons(current).orHeight = height
arrayButtons(current).texture = texture
arrayButtons(current).texturehover = texturehover
```

```
arrayButtons(current).b_name = ButtonName
arrayButtons(current).ColorK = ColorK
End Sub

Public Sub AddList(ByVal x As Single, ByVal y As Single, ByVal width As Sin
gle, ByVal height As Single, ByVal Fontheight As Single, ByVal maxrows As I
nteger, ByVal listname As String, ByVal selected As Integer)
numLists = numLists + 1
Dim current As Integer
current = numLists - 1
ReDim Preserve arrayLists(current)
arrayLists(current).orX = x
arrayLists(current).orY = y
arrayLists(current).orWidth = width
arrayLists(current).orHeight = height
arrayLists(current).orFontHeight = Fontheight
arrayLists(current).l_name = listname
arrayLists(current).maxrows = maxrows
arrayLists(current).selected = selected

ReDim arrayLists(current).listover(arrayLists(current).maxrows + 2)

ReDim arrayLists(current).list(0)
End Sub

Public Sub AddLabel(ByVal text As String, ByVal x As Single, ByVal y As Sin
gle, ByVal Fontheight As Single, Optional ByVal Fontwidth As Single = 0)
numLabels = numLabels + 1
ReDim Preserve arrayLabels(numLabels - 1)
arrayLabels(numLabels - 1).text = text
arrayLabels(numLabels - 1).orX = x
arrayLabels(numLabels - 1).orY = y
arrayLabels(numLabels - 1).orFontHeight = Fontheight
If Fontwidth = 0 Then
    arrayLabels(numLabels - 1).orFontWidth = Fontheight
Else
    arrayLabels(numLabels - 1).orFontWidth = Fontwidth
End If
End Sub

Public Sub AddImage(ByVal file As String, ByVal x As Single, ByVal y As Sin
gle, ByVal width As Single, ByVal height As Single, ByVal ColorK As Long)
numImages = numImages + 1
ReDim Preserve arrayImages(numImages - 1)
arrayImages(numImages - 1).orX = x
arrayImages(numImages - 1).orY = y
arrayImages(numImages - 1).orWidth = width
arrayImages(numImages - 1).orHeight = height
arrayImages(numImages - 1).texture = file
arrayImages(numImages - 1).ColorK = ColorK
End Sub

Public Sub ProcessClick(ByVal x As Single, ByVal y As Single)
'-------------- THIS COORDINATES ARE IN D3DM resolution!!!!!! ----------
-
Dim rows As Integer, number As Integer, bottom  As Boolean
If Not aListIsOpened Then
    Dim f As Integer
    For f = 0 To numButtons - 1
    With arrayButtons(f)
```

```
        If x > (.x) And x < (.x + .width) Then
        If y > (.y) And y < (.y + .height) Then
            isEvent = True
            EventName = .b_name
            Exit For
        End If
        End If
    End With
    Next

    For f = 0 To numLists - 1
    With arrayLists(f)
        If x > (.x) And x < (.x + .width) Then
        If y > (.y) And y < (.y + .height) Then
            'Open the list!
            aListIsOpened = True
            ListOpenedIndex = f
            Exit For
        End If
        End If
    End With
    Next
Else
    With arrayLists(ListOpenedIndex)
        rows = UBound(.list)
        If rows > .maxrows Then rows = .maxrows
        rows = rows + 2

        If x > (.x) And x < (.x + .width) And _
           y > (.y + .height) And y < (.y + .height + rows * .height) Then
            'The user has chosen a list element
            number = (y - .y - .height) \ .height + 1
            If .scroll = False Then
                If number = 1 Or number = UBound(.list) + 2 Then Exit Sub
                number = number - 1
                .selected = number
                aListIsOpened = False
            Else
                If number = 1 And .startelement <> 1 Then .startelement = .
startelement - 1
                If number = .maxrows + 2 And .startelement <> (UBound(.list
) - .maxrows + 1) Then .startelement = .startelement + 1
                If number <> 1 And number <> .maxrows + 2 Then
                    .selected = .startelement + number - 2
                    aListIsOpened = False
                End If
            End If
        Else
            aListIsOpened = False
        End If
    End With
End If
End Sub

Public Sub ProcessMouseMove(ByVal x As Single, ByVal y As Single)
'-------------- THIS COORDINATES ARE IN D3DM resolution!!!!!! -------------
-
Dim f As Integer, rows As Integer

If Not aListIsOpened Then
```

```
    For f = 0 To numButtons - 1
        arrayButtons(f).isover = False
    Next
    For f = 0 To numButtons - 1
    With arrayButtons(f)
        If x > (.x) And x < (.x + .width) Then
        If y > (.y) And y < (.y + .height) Then
            .isover = True
            Exit For
        End If
        End If
    End With
    Next
Else
    With arrayLists(ListOpenedIndex)
        For f = 1 To UBound(.listover)
            .listover(f) = 0
        Next

        rows = UBound(.list)
        If rows > .maxrows Then rows = .maxrows
        rows = rows + 2

        If x > (.x) And x < (.x + .width) And _
           y > (.y + .height) And y < (.y + .height + rows * .height) Then
            rows = (y - .y - .height) \ .height + 1
            If rows > .maxrows + 2 Then rows = .maxrows
            .listover(rows) = 1
        End If
    End With
End If
End Sub

'--------------- LOADS THE TEXTURES AND CALCULATES THE COORDS. ------------
------
Public Sub BuildMenu()
ReDim Textures(numButtons - 1)
ReDim Textures2(numButtons - 1)
Dim f As Integer, g As Integer, number As Integer
For f = 0 To numButtons - 1
With arrayButtons(f)
    Set Textures(f) = LoadTex(.texture, .ColorK)
    Set Textures2(f) = LoadTex(.texturehover, .ColorK)
End With
Next

Set BGTex = LoadTex(BackGround, 0)

For f = 0 To numLists - 1
With arrayLists(f)
    .startelement = 1
    .scroll = .maxrows < UBound(.list)
End With
Next

If ListTexture <> "" Then Set ListTex = LoadTex(ListTexture, 0)
If ListTextureOpen <> "" Then Set ListTexOpen = LoadTex(ListTextureOpen, 0)
If ListTextureOpenHover <> "" Then Set ListTexOpenHover = LoadTex(ListTextu
reOpenHover, 0)
```

```
ReDim Textures3Names(0)
ReDim Textures3(0)
For f = 0 To numImages - 1
    For g = 0 To UBound(Textures3Names) - 1
        If Textures3Names(g) = arrayImages(f).texture Then GoTo Out
    Next
    ReDim Preserve Textures3Names(UBound(Textures3Names) + 1)
    ReDim Preserve Textures3(UBound(Textures3) + 1)
    Textures3Names(UBound(Textures3Names) - 1) = arrayImages(f).texture
    Set Textures3(UBound(Textures3) - 1) = LoadTex(arrayImages(f).texture,
arrayImages(f).ColorK)
Out:
Next

Call RefreshDM
End Sub

'----------------- RECALCULATES ALL THE COORDS. FOR THE NEW DM ------------
----
Public Sub RefreshDM()
Dim f As Integer
For f = 0 To numButtons - 1
With arrayButtons(f)
    .x = .orX * D3DM.width / 800
    .y = .orY * D3DM.height / 600
    .width = .orWidth * D3DM.width / 800
    .height = .orHeight * D3DM.height / 600
End With
Next

For f = 0 To numLists - 1
With arrayLists(f)
    .x = .orX * D3DM.width / 800
    .y = .orY * D3DM.height / 600
    .width = .orWidth * D3DM.width / 800
    .height = .orHeight * D3DM.height / 600
    .Fontheight = .orFontHeight * D3DM.height / 600
End With
Next

For f = 0 To numImages - 1
With arrayImages(f)
    .x = .orX * D3DM.width / 800
    .y = .orY * D3DM.height / 600
    .width = .orWidth * D3DM.width / 800
    .height = .orHeight * D3DM.height / 600
End With
Next

For f = 0 To numLabels - 1
With arrayLabels(f)
    .x = .orX * D3DM.width / 800
    .y = .orY * D3DM.height / 600
    .Fontwidth = .orFontWidth * D3DM.width / 800
    .Fontheight = .orFontHeight * D3DM.height / 600
End With
Next
End Sub

'----------------- SHOWS THE MENU WITH ALL THE EFECTS --------------
```

cGameMenu - 7

```
Public Sub RenderMenu()
bgvertices(0) = AssignMV(0, 0, 0, 0.001, 0.001)
bgvertices(1) = AssignMV(D3DM.width, 0, 0, 0.999, 0.001)
bgvertices(2) = AssignMV(0, D3DM.height, 0, 0.001, 0.999)
bgvertices(3) = AssignMV(D3DM.width, D3DM.height, 0, 0.999, 0.999)
Device.SetTexture 0, BGTex
Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, bgvertices(0), Len(bgvertice
s(0))
Dim f As Integer, g As Integer, offset As Single, offset2 As Single

For f = 0 To numImages - 1
With arrayImages(f)
    PublicVertices(0) = AssignMV(.x, .y, 0, 0.001, 0.001)
    PublicVertices(1) = AssignMV(.x + .width, .y, 0, 0.999, 0.001)
    PublicVertices(2) = AssignMV(.x, .y + .height, 0, 0, 0.999)
    PublicVertices(3) = AssignMV(.x + .width, .y + .height, 0, 0.999, 0.999
)
    Device.SetTexture 0, GetTex3(.texture)
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, PublicVertices(0), Len(P
ublicVertices(0))
End With
Next

For f = 0 To numLabels - 1
With arrayLabels(f)
    DrawFont .text, .x, .y, 0, .Fontheight, .Fontwidth
End With
Next

For f = 0 To numButtons - 1
With arrayButtons(f)
    If .isover = False Then
        Device.SetTexture 0, Textures(f)
    Else
        Device.SetTexture 0, Textures2(f)
    End If
    PublicVertices(0) = AssignMV(.x, .y, 0, 0.001, 0.001)
    PublicVertices(1) = AssignMV(.x + .width, .y, 0, 0.999, 0.001)
    PublicVertices(2) = AssignMV(.x, .y + .height, 0, 0.001, 0.999)
    PublicVertices(3) = AssignMV(.x + .width, .y + .height, 0, 0.999, 0.999
)
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, PublicVertices(0), Len(P
ublicVertices(0))
End With
Next

For f = 0 To numLists - 1
With arrayLists(f)
    offset = (.width - Len(.list(.selected)) * .Fontheight) / 2
    offset2 = (.height - .Fontheight) / 2
    PublicVertices(0) = AssignMV(.x, .y, 0, 0.001, 0.001)
    PublicVertices(1) = AssignMV(.x + .width, .y, 0, 0.999, 0.001)
    PublicVertices(2) = AssignMV(.x, .y + .height, 0, 0.001, 0.999)
    PublicVertices(3) = AssignMV(.x + .width, .y + .height, 0, 0.999, 0.999
)
    Device.SetTexture 0, ListTex
    Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, PublicVertices(0), Len(P
ublicVertices(0))
    DrawFont .list(.selected), .x + offset, .y + offset2, 0, .Fontheight, .
Fontheight
```

```
End With
Next

If aListIsOpened Then
    With arrayLists(ListOpenedIndex)
    offset2 = (.height - .Fontheight) / 2
    If .scroll = True Then
        For f = 0 To .maxrows + 1
            PublicVertices(0) = AssignMV(.x, .y + .height * (f + 1), 0, 0.0
01, 0.001)
            PublicVertices(1) = AssignMV(.x + .width, .y + .height * (f + 1
), 0, 0.999, 0.001)
            PublicVertices(2) = AssignMV(.x, .y + .height * (f + 2), 0, 0.0
01, 0.999)
            PublicVertices(3) = AssignMV(.x + .width, .y + .height * (f + 2
), 0, 0.999, 0.999)
            If .listover(f + 1) = 1 Then
                If (f <> 0 And f <> .maxrows + 1) Or _
                   ((f = 0) And (.startelement <> 1)) Or _
                   ((f = .maxrows + 1) And .startelement <> (UBound(.list)
- .maxrows + 1)) Then
                    Device.SetTexture 0, ListTexOpenHover
                Else
                    Device.SetTexture 0, ListTexOpen
                End If
            Else
                Device.SetTexture 0, ListTexOpen
            End If
            Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, PublicVertices(0
), Len(PublicVertices(0))

            If f <> .maxrows + 1 And f <> 0 Then
                offset = (.width - Len(.list(.startelement + f - 1)) * .Fon
theight) / 2#
                DrawFont .list(.startelement + f - 1), .x + offset, .y + (f
 + 1) * .height + offset2, 0, .Fontheight, .Fontheight
            ElseIf f = 0 And .startelement <> 1 Then
                offset = (.width - .Fontheight) / 2#
                DrawFont "ª", .x + offset, .y + (f + 1) * .height + offset2
, 0, .Fontheight, .Fontheight
            ElseIf (.startelement + .maxrows - 1) <> UBound(.list) And f =
.maxrows + 1 Then
                offset = (.width - .Fontheight) / 2#
                DrawFont "º", .x + offset, .y + (f + 1) * .height + offset2
, 0, .Fontheight, .Fontheight
            End If
        Next
    Else
        For f = 0 To UBound(.list) + 1
            PublicVertices(0) = AssignMV(.x, .y + .height * (f + 1), 0, 0.0
01, 0.001)
            PublicVertices(1) = AssignMV(.x + .width, .y + .height * (f + 1
), 0, 0.999, 0.001)
            PublicVertices(2) = AssignMV(.x, .y + .height * (f + 2), 0, 0.0
01, 0.999)
            PublicVertices(3) = AssignMV(.x + .width, .y + .height * (f + 2
), 0.001, 0.999, 0.999)
            If .listover(f + 1) = 1 And f <> UBound(.list) + 1 And f <> 0 T
hen
                Device.SetTexture 0, ListTexOpenHover
```

```
            Else
                Device.SetTexture 0, ListTexOpen
            End If
            Device.DrawPrimitiveUP D3DPT_TRIANGLESTRIP, 2, PublicVertices(0
), Len(PublicVertices(0))

            If f <> UBound(.list) + 1 And f <> 0 Then
                offset = (.width - Len(.list(f)) * .Fontheight) / 2#
                DrawFont .list(f), .x + offset, .y + (f + 1) * .height + of
fset2, 0, .Fontheight, .Fontheight
            End If
        Next
    End If
    End With
End If
End Sub

Private Function LoadTex(ByVal file As String, ByVal ColorK As Long) As Dir
ect3DTexture8
Dim transparent As Boolean
Dim imagew As Long, imageh As Long

GetImageProperties file, imagew, imageh, transparent

If caps.MaxTextureWidth < imagew Then imagew = caps.MaxTextureWidth
If caps.MaxTextureHeight < imageh Then imageh = caps.MaxTextureHeight
If caps.TextureCaps And D3DPTEXTURECAPS_SQUAREONLY Then
    If imagew > imageh Then
        imagew = imageh
    Else
        imageh = imagew
    End If
End If

If transparent = False Then
    If Direct3D.CheckDeviceFormat(0, D3DDEVTYPE_HAL, D3DM.Format, 0, D3DRTY
PE_TEXTURE, D3DFMT_DXT1) = D3D_OK Then
        Set LoadTex = Direct3DX.CreateTextureFromFileEx(Device, TexPath & f
ile, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_DXT1, D3DPOOL_MANAGED, D3DX_FI
LTER_LINEAR, D3DX_FILTER_LINEAR, ColorK, ByVal 0, ByVal 0)
    Else
        Set LoadTex = Direct3DX.CreateTextureFromFileEx(Device, TexPath & f
ile, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX
_FILTER_LINEAR, D3DX_FILTER_LINEAR, ColorK, ByVal 0, ByVal 0)
    End If
Else
    If Direct3D.CheckDeviceFormat(0, D3DDEVTYPE_HAL, D3DM.Format, 0, D3DRTY
PE_TEXTURE, D3DFMT_DXT5) = D3D_OK Then
        Set LoadTex = Direct3DX.CreateTextureFromFileEx(Device, TexPath & f
ile, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_DXT5, D3DPOOL_MANAGED, D3DX_FI
LTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    Else
        Set LoadTex = Direct3DX.CreateTextureFromFileEx(Device, TexPath & f
ile, imagew, imageh, D3DX_DEFAULT, 0, D3DFMT_UNKNOWN, D3DPOOL_MANAGED, D3DX
_FILTER_LINEAR, D3DX_FILTER_LINEAR, 0, ByVal 0, ByVal 0)
    End If
End If
End Function

Public Sub AddElementToList(ByVal listname As String, ByVal element As Stri
```

```
ng)
Dim f As Integer
For f = 0 To numLists
    If arrayLists(f).l_name = listname Then GoTo Found:
Next
Exit Sub
Found:
ReDim Preserve arrayLists(f).list(UBound(arrayLists(f).list) + 1)
arrayLists(f).list(UBound(arrayLists(f).list)) = element
End Sub

Public Sub ReInitialize()
Dim f As Integer
For f = 0 To numLists - 1
    arrayLists(f).startelement = 1
Next
End Sub

Private Function GetTex3(ByVal file As String) As Direct3DTexture8
Dim f As Integer
For f = 0 To UBound(Textures3Names) - 1
    If Textures3Names(f) = file Then
        Set GetTex3 = Textures3(f)
        Exit For
    End If
Next
End Function

Public Function GetListValue(ByVal listname As String) As String
Dim f As Integer
For f = 0 To numLists - 1
    If arrayLists(f).l_name = listname Then
        GetListValue = arrayLists(f).list(arrayLists(f).selected)
        Exit Function
    End If
Next
End Function

Public Function GetListIndex(ByVal listname As String) As Integer
Dim f As Integer
For f = 0 To numLists - 1
    If arrayLists(f).l_name = listname Then
        GetListIndex = arrayLists(f).selected
        Exit Function
    End If
Next
End Function

Public Sub SetSelectedListItem(ByVal listname As String, ByVal selected As
Integer)
Dim f As Integer
For f = 0 To numLists - 1
    If arrayLists(f).l_name = listname Then
        arrayLists(f).selected = selected
        Exit Sub
    End If
Next
End Sub

Public Sub SetSelectedListItemByText(ByVal listname As String, ByVal select
```

```
ed As String)
Dim f As Integer, x As Integer
For f = 0 To numLists - 1
    If arrayLists(f).l_name = listname Then
        For x = 1 To UBound(arrayLists(f).list)
            If arrayLists(f).list(x) = selected Then
                arrayLists(f).selected = x
                Exit Sub
            End If
        Next
    End If
Next
End Sub
```

clsCullNode - 1

```
'##################################################################
'##################################################################
'#############       SCENE DIVISION BY QUADTREES     #############
'#############                                       #############
'#############       by davidgf                      #############
'##################################################################
'##################################################################

'Slices a mesh given a pos and the dimensions of the cube (height is
'considered as infinite, 2D)

Option Explicit

Public width As Single
Public height As Single

Private VertexBuffer As Direct3DVertexBuffer8

Private Position As D3DVECTOR
Private vertices() As D3DVECTOR
Public NumOfVertices As Long
Private SlicePlanes(3) As D3DPLANE

Public Property Get CubePosition() As D3DVECTOR
CubePosition.x = Position.x
CubePosition.y = Position.y
CubePosition.z = Position.z
End Property

Public Property Let CubePosition(vNewValue As D3DVECTOR)
Position.x = vNewValue.x
Position.y = vNewValue.y
Position.z = vNewValue.z
End Property

Public Sub BuildNode(InputMesh As D3DXMesh)
Call pGenereatPlanes

Dim TempVertices() As D3DVECTOR, x As Long, y As Long
ExtractTriVec InputMesh, TempVertices
ReDim vertices(299)
Dim TempVertices2() As D3DVECTOR


For x = 0 To UBound(TempVertices) Step 3
    If CheckPointInCube(TempVertices(x)) And _
       CheckPointInCube(TempVertices(x + 1)) And _
       CheckPointInCube(TempVertices(x + 2)) Then
        'tri in cube
        vertices(NumOfVertices) = TempVertices(x)
        vertices(NumOfVertices + 1) = TempVertices(x + 1)
        vertices(NumOfVertices + 2) = TempVertices(x + 2)
        NumOfVertices = NumOfVertices + 3
        If NumOfVertices > UBound(vertices) Then
            ReDim Preserve vertices(NumOfVertices + 299)
        End If
        GoTo NextTri
    End If
```

```
    If IsTriOut(TempVertices(x), TempVertices(x + 1), TempVertices(x + 2))
Then
        GoTo NextTri
    End If
    Call CutTriangle(TempVertices(x), TempVertices(x + 1), TempVertices(x +
 2), TempVertices2)
    If UBound(TempVertices2) = 0 Then GoTo NextTri
    If UBound(TempVertices2) + 1 >= UBound(vertices) - NumOfVertices + 1 Th
en
        ReDim Preserve vertices(UBound(vertices) + UBound(TempVertices2) +
301)
    End If
    For y = 0 To UBound(TempVertices2)
        vertices(NumOfVertices + y) = TempVertices2(y)
    Next
    NumOfVertices = NumOfVertices + UBound(TempVertices2) + 1
NextTri:
Next

If NumOfVertices = 0 Then Exit Sub
If UBound(vertices) > NumOfVertices Then
    ReDim Preserve vertices(NumOfVertices - 1)
End If
ReDim TempVertices(0)
ReDim TempVertices2(0)
End Sub

Public Sub BuildNodeFromVerts(TempVertices() As D3DVECTOR)
Call pGenereatPlanes

Dim x As Long, y As Long
ReDim vertices(299)
Dim TempVertices2() As D3DVECTOR

For x = 0 To UBound(TempVertices) Step 3
    If CheckPointInCube(TempVertices(x)) And _
       CheckPointInCube(TempVertices(x + 1)) And _
       CheckPointInCube(TempVertices(x + 2)) Then
        'tri in cube
        vertices(NumOfVertices) = TempVertices(x)
        vertices(NumOfVertices + 1) = TempVertices(x + 1)
        vertices(NumOfVertices + 2) = TempVertices(x + 2)
        NumOfVertices = NumOfVertices + 3
        If NumOfVertices > UBound(vertices) Then
            ReDim Preserve vertices(NumOfVertices + 299)
        End If
        GoTo NextTri
    End If

    If IsTriOut(TempVertices(x), TempVertices(x + 1), TempVertices(x + 2))
Then
        GoTo NextTri
    End If

    Call CutTriangle(TempVertices(x), TempVertices(x + 1), TempVertices(x +
 2), TempVertices2)
    If UBound(TempVertices2) = 0 Then GoTo NextTri
    If UBound(TempVertices2) + 1 >= UBound(vertices) - NumOfVertices + 1 Th
en
        ReDim Preserve vertices(UBound(vertices) + UBound(TempVertices2) +
```

```
clsCullNode - 3

301)
    End If
    For y = 0 To UBound(TempVertices2)
        vertices(NumOfVertices + y) = TempVertices2(y)
    Next
    NumOfVertices = NumOfVertices + UBound(TempVertices2) + 1
NextTri:
Next

If UBound(vertices) > NumOfVertices Then
    If NumOfVertices <> 0 Then
        ReDim Preserve vertices(NumOfVertices - 1)
    End If
End If
ReDim TempVertices2(0)
End Sub

Private Sub pGenereatPlanes()

'|--------------|
'|      |       |
'|      ¡       |
'|-----><------|
'|      ^       |
'|      |       |
'|--------------|

' Up plane
D3DXPlaneFromPointNormal SlicePlanes(0), v3(Position.x, Position.y, Positio
n.z + height), v3(0, 0, -1)

' Down plane
D3DXPlaneFromPointNormal SlicePlanes(1), v3(Position.x, Position.y, Positio
n.z - height), v3(0, 0, 1)

' Left plane
D3DXPlaneFromPointNormal SlicePlanes(2), v3(Position.x - width, Position.y,
 Position.z), v3(1, 0, 0)

' Right plane
D3DXPlaneFromPointNormal SlicePlanes(3), v3(Position.x + width, Position.y,
 Position.z), v3(-1, 0, 0)
End Sub

Private Function CheckPointInCube(point As D3DVECTOR) As Boolean
If point.z <= Position.z + height Then
If point.z >= Position.z - height Then
If point.x <= Position.x + width Then
If point.x >= Position.x - width Then
    CheckPointInCube = True
End If
End If
End If
End If
End Function

Public Sub GetVertices(myArray() As D3DVECTOR)
If NumOfVertices = 0 Then Exit Sub
ReDim myArray(NumOfVertices - 1)
Dim x As Long
```

```
clsCullNode - 4

For x = 0 To NumOfVertices - 1
    myArray(x) = vertices(x)
Next
End Sub

Private Function LinePlaneCollide(vec1 As D3DVECTOR, vec2 As D3DVECTOR, pla
ne As D3DPLANE) As Boolean
On Local Error Resume Next
Err.Clear: Err.number = 0
Dim k As Double, vdir As D3DVECTOR
vdir.x = vec2.x - vec1.x
vdir.y = vec2.y - vec1.y
vdir.z = vec2.z - vec1.z
k = -(plane.a * vec1.x + plane.b * vec1.y + plane.c + vec1.z + plane.d) / (
vdir.x * plane.a + vdir.y * plane.b + vdir.z * plane.c)
If k > 0 And k < 1 And Err.number = 0 Then LinePlaneCollide = True
End Function

Private Function LinePlaneIntersect(vec1 As D3DVECTOR, vec2 As D3DVECTOR, p
lane As D3DPLANE) As D3DVECTOR
Dim k As Double, vdir As D3DVECTOR
Dim var As Double
vdir.x = vec2.x - vec1.x
vdir.y = vec2.y - vec1.y
vdir.z = vec2.z - vec1.z
k = -(plane.a * vec1.x + plane.b * vec1.y + plane.c * vec1.z + plane.d) / (
vdir.x * plane.a + vdir.y * plane.b + vdir.z * plane.c)
LinePlaneIntersect.x = vec1.x + k * vdir.x
LinePlaneIntersect.y = vec1.y + k * vdir.y
LinePlaneIntersect.z = vec1.z + k * vdir.z
End Function

Private Sub CutTriangle(v1 As D3DVECTOR, v2 As D3DVECTOR, v3 As D3DVECTOR,
VOut() As D3DVECTOR)
Dim x As Integer, y As Long, z As Long
Dim vector1 As D3DVECTOR, vector2 As D3DVECTOR, vector3 As D3DVECTOR
Dim point1 As D3DVECTOR, point2 As D3DVECTOR
Dim i1 As Boolean, i2 As Boolean
Dim side1 As Boolean, side2 As Boolean, side3 As Boolean

ReDim VOut(2)
VOut(0) = v1: VOut(1) = v2: VOut(2) = v3
TryAgain:
For x = 0 To 3         'for each plane
    For y = 0 To UBound(VOut) Step 3
        side1 = PointAtRightSide(VOut(y), SlicePlanes(x))
        side2 = PointAtRightSide(VOut(y + 1), SlicePlanes(x))
        side3 = PointAtRightSide(VOut(y + 2), SlicePlanes(x))

        If (Not side1 And (side2 And side3)) Then          'side1 is out
            point1 = LinePlaneIntersect(VOut(y), VOut(y + 1), SlicePlanes(x
))
            point2 = LinePlaneIntersect(VOut(y), VOut(y + 2), SlicePlanes(x
))
            ReDim Preserve VOut(UBound(VOut) + 3)
            VOut(y) = point2
            VOut(UBound(VOut) - 2) = point1
            VOut(UBound(VOut) - 1) = VOut(y + 1)
            VOut(UBound(VOut)) = point2
        ElseIf (Not side2 And (side3 And side1)) Then           'side2 is
```

```
out
            point1 = LinePlaneIntersect(VOut(y + 1), VOut(y + 2), SlicePlan
es(x))
            point2 = LinePlaneIntersect(VOut(y + 1), VOut(y), SlicePlanes(x
))
            ReDim Preserve VOut(UBound(VOut) + 3)
            VOut(y + 1) = point2
            VOut(UBound(VOut) - 2) = point1
            VOut(UBound(VOut) - 1) = VOut(y + 2)
            VOut(UBound(VOut)) = point2
        ElseIf (Not side3 And (side2 And side1)) Then          'side3 is ou
t
            point1 = LinePlaneIntersect(VOut(y + 2), VOut(y), SlicePlanes(x
))
            point2 = LinePlaneIntersect(VOut(y + 2), VOut(y + 1), SlicePlan
es(x))
            ReDim Preserve VOut(UBound(VOut) + 3)
            VOut(y + 2) = point2
            VOut(UBound(VOut) - 2) = point1
            VOut(UBound(VOut) - 1) = VOut(y)
            VOut(UBound(VOut)) = point2
        ElseIf (Not (side1 And side2) And side3) Then           'side1 and
side2 are out
            point1 = LinePlaneIntersect(VOut(y + 2), VOut(y), SlicePlanes(x
))
            point2 = LinePlaneIntersect(VOut(y + 2), VOut(y + 1), SlicePlan
es(x))
            VOut(y) = point1
            VOut(y + 1) = point2
        ElseIf (Not (side3 And side2) And side1) Then           'side2 and
side3 are out
            point1 = LinePlaneIntersect(VOut(y), VOut(y + 1), SlicePlanes(x
))
            point2 = LinePlaneIntersect(VOut(y), VOut(y + 2), SlicePlanes(x
))
            VOut(y + 1) = point1
            VOut(y + 2) = point2
        ElseIf (Not (side3 And side1) And side2) Then           'side3 and
side1 are out
            point1 = LinePlaneIntersect(VOut(y + 1), VOut(y), SlicePlanes(x
))
            point2 = LinePlaneIntersect(VOut(y + 1), VOut(y + 2), SlicePlan
es(x))
            VOut(y) = point1
            VOut(y + 2) = point2
        ElseIf Not (side3 And side1 And side2) Then
            'tri is out! so we must delete it!
            For z = y To UBound(VOut) - 3 Step 3
                VOut(z) = VOut(z + 3)
                VOut(z + 1) = VOut(z + 4)
                VOut(z + 2) = VOut(z + 5)
            Next
            If UBound(VOut) - 3 > 0 Then
                ReDim Preserve VOut(UBound(VOut) - 3)
            Else
                ReDim VOut(0)
                Exit Sub
            End If
        Else
            GoTo NextVert
```

```
        End If
        GoTo TryAgain
NextVert:
    Next y
Next x
End Sub

Private Function PointAtRightSide(point As D3DVECTOR, plane As D3DPLANE) As
 Boolean
'If D3DXPlaneDotCoord(plane, point) >= 0 Then PointAtRightSide = True
If D3DXPlaneDotCoord(plane, point) >= -0.05 Then PointAtRightSide = True
End Function

Private Function PointAtRightSideE(point As D3DVECTOR, plane As D3DPLANE) A
s Boolean
'If D3DXPlaneDotCoord(plane, point) >= 0 Then PointAtRightSide = True
If D3DXPlaneDotCoord(plane, point) >= 0 Then PointAtRightSideE = True
End Function

Private Function IsTriOut(v1 As D3DVECTOR, v2 As D3DVECTOR, v3 As D3DVECTOR
) As Boolean
Dim x As Integer
Dim side1 As Boolean, side2 As Boolean, side3 As Boolean

If v1.z >= Position.z + height Then
If v2.z >= Position.z + height Then
If v3.z >= Position.z + height Then
    IsTriOut = True
End If
End If
End If

If v1.z <= Position.z - height Then
If v2.z <= Position.z - height Then
If v3.z <= Position.z - height Then
    IsTriOut = True
End If
End If
End If

If v1.x >= Position.x + width Then
If v2.x >= Position.x + width Then
If v3.x >= Position.x + width Then
    IsTriOut = True
End If
End If
End If

If v1.x <= Position.x - width Then
If v2.x <= Position.x - width Then
If v3.x <= Position.x - width Then
    IsTriOut = True
End If
End If
End If
End Function

Private Function PointsAligned(v1 As D3DVECTOR, v2 As D3DVECTOR, v3 As D3DV
ECTOR) As Boolean
'extreure l'equació de la recta entre v1 i v2 i v1 i v3 per comprobar si es
```

clsCullNode - 7

```
tan alineats
'compute linear equations to know if the points are aligned
Dim director As D3DVECTOR, director2 As D3DVECTOR
director.x = v3.x - v1.x
director.y = v3.y - v1.y
director.z = v3.z - v1.z
director2.x = v2.x - v1.x
director2.y = v2.y - v1.y
director2.z = v2.z - v1.z

director = Normalize(director)
director2 = Normalize(director2)

If CompareAprox(director, director2) Then
    'very similar directors
    PointsAligned = True
End If
End Function

Private Function CompareAprox(v1 As D3DVECTOR, v2 As D3DVECTOR) As Boolean
If v2.x <= v1.x + 0.001 And v2.x >= v1.x - 0.001 Then
If v2.y <= v1.y + 0.001 And v2.y >= v1.y - 0.001 Then
If v2.z <= v1.z + 0.001 And v2.z >= v1.z - 0.001 Then
    CompareAprox = True
End If
End If
End If
End Function

Private Sub JoinTriangles(triarray() As D3DVECTOR)
If UBound(triarray) = 2 Then Exit Sub
Dim x As Long, y As Long, z As Long
Do While x < UBound(triarray)
    y = x + 3
    Do While y < UBound(triarray)
        If CompareAprox(triarray(x + 2), triarray(y + 2)) And CompareAprox(
triarray(x + 1), triarray(y)) Then
            ' 22, 10
            If PointsAligned(triarray(x), triarray(x + 2), triarray(y + 1))
 Then
                triarray(x + 2) = triarray(y + 1)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x + 2), triarray(y + 1)) And CompareAp
rox(triarray(x + 1), triarray(y + 2)) Then
            ' 21, 12
            If PointsAligned(triarray(x), triarray(x + 2), triarray(y)) The
n
                triarray(x + 2) = triarray(y)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x + 2), triarray(y)) And CompareAprox(
triarray(x + 1), triarray(y + 1)) Then
            ' 20, 11
```

```
            If PointsAligned(triarray(x), triarray(x + 2), triarray(y + 2))
 Then
                triarray(x + 2) = triarray(y + 2)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x + 1), triarray(y + 2)) And CompareAp
rox(triarray(x), triarray(y)) Then
            ' 12, 00
            If PointsAligned(triarray(x + 2), triarray(x + 1), triarray(y +
 1)) Then
                triarray(x + 1) = triarray(y + 1)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x + 1), triarray(y + 1)) And CompareAp
rox(triarray(x), triarray(y + 2)) Then
            ' 11, 02
            If PointsAligned(triarray(x + 2), triarray(x + 1), triarray(y))
 Then
                triarray(x + 1) = triarray(y)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x + 1), triarray(y)) And CompareAprox(
triarray(x), triarray(y + 1)) Then
            ' 10, 01
            If PointsAligned(triarray(x + 2), triarray(x + 1), triarray(y +
 2)) Then
                triarray(x + 1) = triarray(y + 2)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x), triarray(y + 2)) And CompareAprox(
triarray(x + 2), triarray(y)) Then
            ' 02, 20
            If PointsAligned(triarray(x + 1), triarray(x), triarray(y + 1))
 Then
                triarray(x) = triarray(y + 1)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x), triarray(y + 1)) And CompareAprox(
triarray(x + 2), triarray(y + 2)) Then
            ' 01, 22
            If PointsAligned(triarray(x + 1), triarray(x), triarray(y)) The
n
                triarray(x) = triarray(y)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
```

```
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        ElseIf CompareAprox(triarray(x), triarray(y)) And CompareAprox(tria
rray(x + 2), triarray(y + 1)) Then
            ' 00, 21
            If PointsAligned(triarray(x + 1), triarray(x), triarray(y + 2))
 Then
                triarray(x) = triarray(y + 2)
                For z = y To UBound(triarray) - 3
                    triarray(z) = triarray(z + 3)
                Next
                ReDim Preserve triarray(UBound(triarray) - 3)
            End If
        End If
        y = y + 3
    Loop
    x = x + 3
Loop
End Sub
```

clsDLL - 1

```
Option Explicit

'-------- clsDLL --> Loads and uses DLLS

Private hMod        As Long
Private blnIsCDECL  As Boolean

Public Property Get ModuleHandle() As Long
    ModuleHandle = hMod
End Property

Public Property Get IsCDECL() As Boolean
    IsCDECL = blnIsCDECL
End Property

Public Property Let IsCDECL(bln As Boolean)
    blnIsCDECL = bln
End Property

Public Sub UnloadDLL()
    FreeLibrary hMod
    hMod = 0
End Sub

Public Function LoadDLL(ByVal strDLL As String) As Boolean
    blnIsCDECL = False
    hMod = LoadLibrary(strDLL)
    LoadDLL = hMod <> 0
End Function

Public Function CallFunc(ByVal fnc As String, ParamArray args() As Variant)
 As Long
    Dim hFnc    As Long

    hFnc = GetProcAddress(hMod, fnc)
    If hFnc = 0 Then
        Err.Raise 1, , "Export not found!"
        Exit Function
    End If

    If Not IsCDECL Then
        CallFunc = CallStd(hFnc, args)
    Else
        CallFunc = CallCdecl(hFnc, args)
    End If
End Function

Private Function CallStd(ByVal fnc As Long, ParamArray Params() As Variant)
 As Long
    Dim btASM(&HEC00& - 1)  As Byte
    Dim pASM                As Long
    Dim i                   As Integer

    pASM = VarPtr(btASM(0))

    AddByte pASM, &H58                  ' POP EAX
    AddByte pASM, &H59                  ' POP ECX
    AddByte pASM, &H59                  ' POP ECX
    AddByte pASM, &H59                  ' POP ECX
    AddByte pASM, &H59                  ' POP ECX
```

```
    AddByte pASM, &H50                  ' PUSH EAX

    If UBound(Params) = 0 Then
        If IsArray(Params(0)) Then
            For i = UBound(Params(0)) To 0 Step -1
                AddPush pASM, CLng(Params(0)(i))    ' PUSH dword
            Next
        Else
            For i = UBound(Params) To 0 Step -1
                AddPush pASM, CLng(Params(i))       ' PUSH dword
            Next
        End If
    Else
        For i = UBound(Params) To 0 Step -1
            AddPush pASM, CLng(Params(i))           ' PUSH dword
        Next
    End If

    AddCall pASM, fnc                   ' CALL rel addr
    AddByte pASM, &HC3                  ' RET

    CallStd = CallWindowProc(VarPtr(btASM(0)), 0, 0, 0, 0)
End Function

Private Function CallCdecl(ByVal lpfn As Long, ParamArray args() As Variant
) As Long
    Dim btASM(&HEC00& - 1)  As Byte
    Dim pASM                As Long
    Dim btArgSize           As Byte
    Dim i                   As Integer

    pASM = VarPtr(btASM(0))

    If UBound(args) = 0 Then
        If IsArray(args(0)) Then
            For i = UBound(args(0)) To 0 Step -1
                AddPush pASM, CLng(args(0)(i))    ' PUSH dword
                btArgSize = btArgSize + 4
            Next
        Else
            For i = UBound(args) To 0 Step -1
                AddPush pASM, CLng(args(i))       ' PUSH dword
                btArgSize = btArgSize + 4
            Next
        End If
    Else
        For i = UBound(args) To 0 Step -1
            AddPush pASM, CLng(args(i))           ' PUSH dword
            btArgSize = btArgSize + 4
        Next
    End If

    AddByte pASM, &HB8
    AddLong pASM, lpfn
    AddByte pASM, &HFF
    AddByte pASM, &HD0
    AddByte pASM, &H83
    AddByte pASM, &HC4
    AddByte pASM, btArgSize
    AddByte pASM, &HC2
```

clsDLL - 3

```
    AddByte pASM, &H10
    AddByte pASM, &H0

    CallCdecl = CallWindowProc(VarPtr(btASM(0)), 0, 0, 0, 0)
End Function

Private Sub AddPush(pASM As Long, lng As Long)
    AddByte pASM, &H68
    AddLong pASM, lng
End Sub

Private Sub AddCall(pASM As Long, addr As Long)
    AddByte pASM, &HE8
    AddLong pASM, addr - pASM - 4
End Sub

Private Sub AddLong(pASM As Long, lng As Long)
    CpyMem ByVal pASM, lng, 4
    pASM = pASM + 4
End Sub

Private Sub AddByte(pASM As Long, Bt As Byte)
    CpyMem ByVal pASM, Bt, 1
    pASM = pASM + 1
End Sub

Private Sub Class_Terminate()
    UnloadDLL
End Sub
```

clsDynObj - 1

```
Option Explicit

Public WorldID As Long
Public AutoY As Boolean
Public CoordY As Single
Public Visible As Boolean
Public id As String
Public StopRadius As Single
Public Paused  As Boolean

Private Const OrientationOffset = 0.5
Private Type point
    point As D3DVECTOR
    speed As Single
    mSleep As Single
    StopSeconds As Single
End Type

Private Points() As point
Private NumPoints As Long

Private ObjStartMoving As Long          'this numbers represent the indices
 in the
Private ObjMoving As Long               'othercharanims array which contain
 the anims
Private ObjEndMoving As Long
Private ObjQuiet As Long
Private ObjQuietEffect() As Long
Private NumObjQuietEffect As Long
Private Sounds() As String
Private CurrentSound As String

Public Enum animtype
    StartMoving
    EndMoving
    Moving
    Quiet
    QuietWithEffects
End Enum

Private Enum AnimEstatusEnum
    CharQuiet
    CharMoving
    StartingWalking
    EndWalking
End Enum

Private Orientation As D3DVECTOR
Private Position As D3DVECTOR
Private MatrixTransform As D3DMATRIX
Private RotationY As D3DMATRIX
Private Translation As D3DMATRIX
Private LastPoint As Long
Private NextPoint As Long
Private LimPoint As Long

Private NoUpdate As Boolean

Private AnimStatus As AnimEstatusEnum
Private PointTimer As Double
```

clsDynObj - 2

```
Private SleepingTime As Double
Private PauseTime As Double
Private AnimTimer As Double
Private FreezeTime As Double

Private CRadius As Single
Private CCenter As D3DVECTOR
Private Do2ndRender As Boolean

Public Sub PauseAnim()
Paused = True
PauseTime = GetTickCount()
AnimStatus = CharQuiet
End Sub

Public Sub ResumeAnim()
PauseTime = GetTickCount() - PauseTime
PointTimer = PointTimer + PauseTime
If SleepingTime <> 0 Then SleepingTime = SleepingTime + PauseTime
PauseTime = 0
AnimStatus = StartingWalking
Paused = False
End Sub

Public Sub FreezeTimer()
FreezeTime = GetTickCount()
End Sub

Public Sub UnFreezeTimer()
FreezeTime = GetTickCount() - FreezeTime

If PauseTime <> 0 Then PauseTime = PauseTime + FreezeTime
If SleepingTime <> 0 Then SleepingTime = SleepingTime + FreezeTime
If PointTimer <> 0 Then PointTimer = PointTimer + FreezeTime
FreezeTime = 0
End Sub

Public Sub SetPos(ByVal NumPoint As Long)
Position = Points(NumPoint).point
LastPoint = NumPoint

NextPoint = NumPoint + 1
If NextPoint > NumPoints - 1 Then NextPoint = 0

PointTimer = 0
NoUpdate = False
AnimStatus = StartingWalking
End Sub

Public Sub SetLimit(ByVal NumPoint As Long)
LimPoint = NumPoint
PointTimer = 0
NoUpdate = False
End Sub

Public Sub AddPoint(ByRef v As D3DVECTOR, ByVal SpeedAtThisPoint As Single,
 Optional ByVal SleepTime As Single)
ReDim Preserve Points(NumPoints)
Points(NumPoints).point = v
Points(NumPoints).speed = SpeedAtThisPoint
```

```
Points(NumPoints).mSleep = SleepTime
NumPoints = NumPoints + 1
End Sub

Private Function CheckPointInTrajectory(point As D3DVECTOR, ByVal radius As
 Single) As Boolean
Dim plane As D3DPLANE
If VecDistFast(point, Position) < radius ^ 2 Then
    'we are in the affect radius, check if we ara in front of the char or b
ack
    D3DXPlaneFromPointNormal plane, Position, Orientation
    If D3DXPlaneDotCoord(plane, point) > 0 Then
        CheckPointInTrajectory = True
    Else
        CheckPointInTrajectory = False
    End If
End If
End Function

Public Sub RenderSound()
Dim RanNum As Long
If AnimStatus = CharQuiet Then
    'stoped: check point out
    If CheckPointInTrajectory(CharPos, StopRadius * 2) = False Then
        'continue anim
        Me.ResumeAnim
        Exit Sub
    End If
Else
    'walking
    If CheckPointInTrajectory(CharPos, StopRadius) Then
        'stop anim
        RanNum = Int(Rnd() * UBound(Sounds)) + 1
        PauseAnim
        SoundEngine.PlaySound Sounds(RanNum), Position, Orientation
        Exit Sub
    End If
End If
End Sub

Public Sub SetSpeak(ByVal soundfile As String)
Dim x As Long
x = UBound(Sounds) + 1
ReDim Preserve Sounds(x)
Sounds(x) = soundfile
End Sub

Public Sub SetAnimation(animtype As animtype, ByVal path As String, ByVal p
attern As String)
Dim x As Long
On Local Error Resume Next
x = UBound(OtherCharAnimations)
If OtherCharAnimations(0).DurationMS = 0 Then Err.number = 666   'if the ar
ray has been erased then redim 0
If Err.number <> 0 Then GoTo Zero
For x = 0 To UBound(OtherCharAnimations)
    If OtherCharAnimations(x).Tag = pattern Then
        Select Case animtype
        Case StartMoving
            ObjStartMoving = x
```

```
        Case EndMoving
            ObjEndMoving = x
        Case Moving
            ObjMoving = x
        Case Quiet
            ObjQuiet = x
        Case QuietWithEffects
            NumObjQuietEffect = x
        End Select
        If OtherCharAnimations(x).TagLong > CRadius Then
            CRadius = OtherCharAnimations(x).TagLong
        End If
        Exit Sub
    End If
Next
ReDim Preserve OtherCharAnimations(UBound(OtherCharAnimations) + 1)
Zero:
If Err.number <> 0 Then
    Err.Clear: Err.number = 0
    ReDim OtherCharAnimations(0)
End If
On Local Error GoTo 0

Dim TCenter As D3DVECTOR, tradius As Single
LoadAnim3D OtherCharAnimations(UBound(OtherCharAnimations)), path, pattern
OtherCharAnimations(UBound(OtherCharAnimations)).Tag = pattern
Select Case animtype
Case StartMoving
    ObjStartMoving = UBound(OtherCharAnimations)
    For x = 1 To OtherCharAnimations(ObjStartMoving).NumMeshes
        Direct3DX.ComputeBoundingSphereFromMesh OtherCharAnimations(ObjStar
tMoving).Meshes(x), TCenter, tradius
        If tradius > CRadius Then
            CRadius = tradius
            CCenter = TCenter
        End If
    Next
    OtherCharAnimations(ObjStartMoving).TagLong = CRadius
Case EndMoving
    ObjEndMoving = UBound(OtherCharAnimations)
    For x = 1 To OtherCharAnimations(ObjEndMoving).NumMeshes
        Direct3DX.ComputeBoundingSphereFromMesh OtherCharAnimations(ObjEndM
oving).Meshes(x), TCenter, tradius
        If tradius > CRadius Then
            CRadius = tradius
            CCenter = TCenter
        End If
    Next
    OtherCharAnimations(EndMoving).TagLong = CRadius
Case Moving
    ObjMoving = UBound(OtherCharAnimations)
    For x = 1 To OtherCharAnimations(ObjMoving).NumMeshes
        Direct3DX.ComputeBoundingSphereFromMesh OtherCharAnimations(ObjMovi
ng).Meshes(x), TCenter, tradius
        If tradius > CRadius Then
            CRadius = tradius
            CCenter = TCenter
        End If
    Next
    OtherCharAnimations(ObjMoving).TagLong = CRadius
```

```
Case Quiet
    ObjQuiet = UBound(OtherCharAnimations)
    For x = 1 To OtherCharAnimations(ObjQuiet).NumMeshes
        Direct3DX.ComputeBoundingSphereFromMesh OtherCharAnimations(ObjQuie
t).Meshes(x), TCenter, tradius
        If tradius > CRadius Then
            CRadius = tradius
            CCenter = TCenter
        End If
    Next
    OtherCharAnimations(ObjQuiet).TagLong = CRadius
Case QuietWithEffects
    NumObjQuietEffect = UBound(OtherCharAnimations)
    For x = 1 To OtherCharAnimations(NumObjQuietEffect).NumMeshes
        Direct3DX.ComputeBoundingSphereFromMesh OtherCharAnimations(NumObjQ
uietEffect).Meshes(x), TCenter, tradius
        If tradius > CRadius Then
            CRadius = tradius
            CCenter = TCenter
        End If
    Next
    OtherCharAnimations(QuietWithEffects).TagLong = CRadius
End Select


End Sub

Private Sub Class_Initialize()
ReDim Points(0)
ReDim Speeds(0)
ReDim ObjQuietEffect(0)
ReDim Sounds(0)
End Sub

Private Sub Class_Terminate()
ReDim Points(0)
ReDim Speeds(0)
ReDim ObjQuietEffect(0)
DestroyAnim3D OtherCharAnimations(ObjQuiet)
DestroyAnim3D OtherCharAnimations(ObjMoving)
DestroyAnim3D OtherCharAnimations(ObjStartMoving)
DestroyAnim3D OtherCharAnimations(ObjEndMoving)
End Sub

Public Sub RenderTime(Optional ByVal ForceRender As Boolean = False)
'calculate orientation, position and bounding box for collsion
If Visible = False Then Exit Sub

'if this isn't the actual world don't render. In case we force the renderin
g, continue.
Do2ndRender = False
If ForceRender = False And WorldID <> TheGameSlot.WorldID Then Exit Sub

If PointTimer = 0 Then PointTimer = GetTickCount()
Dim interpolation As Double
Dim desp As Double
Dim prev_prev As D3DVECTOR, next_next As D3DVECTOR, tempvec As D3DVECTOR
Dim tmp_long As Long

If NoUpdate Then Exit Sub
```

```
If PauseTime <> 0 Then Exit Sub

If SleepingTime <> 0 Then
    If GetTickCount() - SleepingTime > Points(LastPoint).mSleep Then
        SleepingTime = 0
        PointTimer = GetTickCount()
        AnimStatus = StartingWalking
    Else
        AnimStatus = CharQuiet
        Exit Sub
    End If
End If

desp = (GetTickCount() - PointTimer) * Points(LastPoint).speed / 1000

    If VecDistFast(Points(LastPoint).point, Points(NextPoint).point) <> 0 T
hen
        interpolation = desp / VecDist(Points(LastPoint).point, Points(Next
Point).point)
    Else
        interpolation = desp / 0.01
    End If
    If interpolation > 1 Or VecDistFast(Points(LastPoint).point, Points(Nex
tPoint).point) = 0 Then
        'we have reached and passed the next point, change points
        If NextPoint = LimPoint Then
            NoUpdate = True
            AnimStatus = CharQuiet
            Exit Sub
        End If
        LastPoint = NextPoint
        NextPoint = NextPoint + 1
        If NextPoint > NumPoints - 1 Then NextPoint = 0
        PointTimer = 0
        interpolation = 0
        If Points(LastPoint).mSleep <> 0 Then
            SleepingTime = GetTickCount()
            AnimStatus = CharQuiet
            Exit Sub
        End If
    End If
    Position = LinearInterpolation(Points(LastPoint).point, Points(NextPoin
t).point, interpolation)

    '--------- Calculate Orientation of the model --------
    If PointTimer = 0 Then desp = 0

    If VecDistFast(Points(LastPoint).point, Points(NextPoint).point) <> 0 T
hen
        interpolation = (desp + OrientationOffset) / VecDist(Points(LastPoi
nt).point, Points(NextPoint).point)
    Else
        interpolation = interpolation + 0.001
    End If
    If interpolation > 1 Then
        tmp_long = NextPoint
        Do While interpolation > 1
            If (tmp_long + 1) > (NumPoints - 1) Then
                prev_prev = Points(tmp_long).point
                next_next = Points(0).point
```

```
                tmp_long = 0
            Else
                prev_prev = Points(tmp_long).point
                next_next = Points(tmp_long + 1).point
                tmp_long = tmp_long + 1
            End If
            If tmp_long = 0 Then
                tempvec = Points(NumPoints - 2).point
            ElseIf tmp_long = 1 Then
                tempvec = Points(NumPoints - 1).point
            Else
                tempvec = Points(tmp_long - 2).point
            End If
            desp = desp - VecDist(tempvec, prev_prev)
            interpolation = (desp + OrientationOffset) / VecDist(prev_prev,
 next_next)
        Loop
        Orientation = LinearInterpolation(prev_prev, next_next, interpolati
on)
        Orientation.x = Orientation.x - Position.x
        Orientation.y = Orientation.y - Position.y
        Orientation.z = Orientation.z - Position.z
    Else
        Orientation = LinearInterpolation(Points(LastPoint).point, Points(N
extPoint).point, interpolation)
        Orientation.x = Orientation.x - Points(LastPoint).point.x
        Orientation.y = Orientation.y - Points(LastPoint).point.y
        Orientation.z = Orientation.z - Points(LastPoint).point.z
    End If

    '---- Create Translation And RotationY Matrices from position and orien
tation ------
    '-- Combine them into a transforma Matrix

    ' RotationY matrix is static, so if the model is on the next point we c
an multiply
    ' basing on the last matrix and there's no problem
    If Orientation.x > 0 And Orientation.z > 0 Then
        D3DXMatrixRotationY RotationY, Atn(Orientation.x / Orientation.z) +
 Pi
    ElseIf Orientation.x < 0 And Orientation.z > 0 Then
        D3DXMatrixRotationY RotationY, Atn(Orientation.x / Orientation.z) +
 Pi
    ElseIf Orientation.x > 0 And Orientation.z < 0 Then
        D3DXMatrixRotationY RotationY, Atn(Orientation.x / Orientation.z)
    ElseIf Orientation.x < 0 And Orientation.z < 0 Then
        D3DXMatrixRotationY RotationY, Atn(Orientation.x / Orientation.z)
    ElseIf Orientation.z = 0 And Orientation.x > 0 Then
        D3DXMatrixRotationY RotationY, -Pi / 2
    ElseIf Orientation.z = 0 And Orientation.x < 0 Then
        D3DXMatrixRotationY RotationY, Pi / 2
    ElseIf Orientation.x = 0 And Orientation.z > 0 Then
        D3DXMatrixRotationY RotationY, Pi
    ElseIf Orientation.x = 0 And Orientation.z < 0 Then
        D3DXMatrixRotationY RotationY, 0
    End If

    If AutoY = True Then
        NumDynObjPositions = NumDynObjPositions + 1
        ReDim Preserve DynObjPositions(NumDynObjPositions)
```

```
        DynObjPositions(NumDynObjPositions).x = Position.x
        DynObjPositions(NumDynObjPositions).z = Position.z
        DynObjPositions(NumDynObjPositions).y = DynObjPositions(NumDynObjPo
sitions).y + 2
    End If

    Do2ndRender = True
End Sub

Public Sub RenderTime2()
If Do2ndRender = False Then Exit Sub
If Visible = False Then Exit Sub

If AutoY = False Then
    D3DXMatrixTranslation Translation, Position.x, Position.y, Position.z
    D3DXMatrixMultiply MatrixTransform, RotationY, Translation
Else
    D3DXMatrixTranslation Translation, Position.x, CoordY, Position.z
    D3DXMatrixMultiply MatrixTransform, RotationY, Translation
End If
End Sub

Public Sub RenderNow()
'draw the character on the screen
If Visible = False Then Exit Sub

Dim frame As Long
Dim TCenter As D3DVECTOR, Transl As D3DMATRIX
Static LastStatus As AnimEstatusEnum

Device.SetTransform D3DTS_WORLD, MatrixTransform
D3DXMatrixTranslation Transl, Position.x, Position.y, Position.z
D3DXVec3TransformCoord TCenter, CCenter, Transl

If LastStatus <> AnimStatus Then
    'state changed!!!
    AnimTimer = GetTickCount()
End If
LastStatus = AnimStatus

Select Case AnimStatus
Case CharMoving
    If GetTickCount() - AnimTimer > OtherCharAnimations(ObjMoving).Duration
MS Then
        AnimTimer = GetTickCount()
        If Points(NextPoint).mSleep <> 0 Then
        'calculate if we have to brake
            If VecDist(Position, Points(NextPoint).point) < OtherCharAnimat
ions(ObjEndMoving).DurationMS * Points(LastPoint).speed / 1000 Then
                AnimStatus = EndWalking
            End If
        End If
    End If
    frame = (GetTickCount() - AnimTimer) / OtherCharAnimations(ObjMoving).D
urationMS * OtherCharAnimations(ObjMoving).NumMeshes
    If frame < 1 Then frame = 1
    If frame > OtherCharAnimations(ObjMoving).NumMeshes Then frame = OtherC
harAnimations(ObjMoving).NumMeshes
    If CheckSphere(TCenter, CRadius) Then RenderAnim OtherCharAnimations(Ob
jMoving), frame
```

```
Case CharQuiet
    If CheckSphere(TCenter, CRadius) Then RenderAnim OtherCharAnimations(Ob
jQuiet), 1
Case EndWalking
    If GetTickCount() - AnimTimer > OtherCharAnimations(ObjEndMoving).Durat
ionMS Then
        frame = OtherCharAnimations(ObjEndMoving).NumMeshes
        AnimTimer = GetTickCount()
        AnimStatus = CharQuiet
    Else
        frame = (GetTickCount() - AnimTimer) / OtherCharAnimations(ObjEndMo
ving).DurationMS * OtherCharAnimations(ObjEndMoving).NumMeshes
    End If
    If frame < 1 Then frame = 1
    If frame > OtherCharAnimations(ObjEndMoving).NumMeshes Then frame = Oth
erCharAnimations(ObjEndMoving).NumMeshes
    If CheckSphere(TCenter, CRadius) Then RenderAnim OtherCharAnimations(Ob
jEndMoving), frame
Case StartingWalking
    If GetTickCount() - AnimTimer > OtherCharAnimations(ObjStartMoving).Dur
ationMS Then
        frame = OtherCharAnimations(ObjStartMoving).NumMeshes
        AnimTimer = GetTickCount()
        AnimStatus = CharMoving
    Else
        frame = (GetTickCount() - AnimTimer) / OtherCharAnimations(ObjStart
Moving).DurationMS * OtherCharAnimations(ObjStartMoving).NumMeshes
    End If
    If frame < 1 Then frame = 1
    If frame > OtherCharAnimations(ObjStartMoving).NumMeshes Then frame = O
therCharAnimations(ObjStartMoving).NumMeshes
    If CheckSphere(TCenter, CRadius) Then RenderAnim OtherCharAnimations(Ob
jStartMoving), frame
End Select
End Sub

'-------- PRIVATE AUXILIAR FUNCTIONS --------
'--------------------------------------------

Private Function VecDist(v1 As D3DVECTOR, v2 As D3DVECTOR) As Double
VecDist = Sqr((v1.x - v2.x) ^ 2 + (v1.y - v2.y) ^ 2 + (v1.z - v2.z) ^ 2)
End Function

Private Function LinearInterpolation(v1 As D3DVECTOR, v2 As D3DVECTOR, ByVa
l interpolation As Double) As D3DVECTOR
LinearInterpolation.x = v1.x + interpolation * (v2.x - v1.x)
LinearInterpolation.y = v1.y + interpolation * (v2.y - v1.y)
LinearInterpolation.z = v1.z + interpolation * (v2.z - v1.z)
End Function

Private Function CheckSphere(center As D3DVECTOR, radius As Single) As Bool
ean
Dim i As Long

For i = 0 To 5
    If D3DXPlaneDotCoord(FrustumPlanes(i), center) < -radius Then
        CheckSphere = False
        Exit Function
    End If
Next
```

```
clsDynObj - 10

CheckSphere = True
End Function
```

clsShadow - 1

```
Option Explicit

'----------------- SHADOW MODULE ---------------
'  by davidgf -- translated from C++ DX8 SDK
'  generates a shadow volume and renders it
'  using Z-Fail technique
'-----------------------------------------------

Public LightProj As Long

Private ShadowVertices() As D3DVECTOR
Private NumberVertices

Private ShadowEdges() As Long
Private NumberEdges As Long

Private Position As D3DVECTOR

Private VertexBuffer As Direct3DVertexBuffer8

Public Sub RenderGeometry()
Device.SetStreamSource 0, VertexBuffer, Len(ShadowVertices(0))

Device.DrawPrimitive D3DPT_TRIANGLELIST, 0, NumberVertices / 3
End Sub

Public Sub Build(VertexList() As D3DVECTOR, light As D3DVECTOR)
On Local Error Resume Next
NumberEdges = 0
Dim NumberIndices As Long
ReDim ShadowVertices(UBound(VertexList) * 16)

extrudeshadow LightProj, VertexList(0), UBound(VertexList) + 1, light, Shad
owVertices(0), NumberVertices

Set VertexBuffer = Device.CreateVertexBuffer(NumberVertices * 12, D3DUSAGE_
WRITEONLY Or D3DUSAGE_DONOTCLIP, D3DFVF_XYZ, D3DPOOL_MANAGED)
D3DVertexBuffer8SetData VertexBuffer, 0, NumberVertices * 12, 0, ShadowVert
ices(0)
ReDim ShadowVertices(0)
End Sub

Private Sub AddEdge(ByVal v1 As Long, ByVal v2 As Long)
Dim i As Long
For i = 0 To NumberEdges - 1
    If (ShadowEdges(2 * i) = v1 And ShadowEdges(2 * i + 1) = v2) _
    Or (ShadowEdges(2 * i) = v2 And ShadowEdges(2 * i + 1) = v1) Then
        If NumberEdges > 1 Then
            ShadowEdges(2 * i) = ShadowEdges(2 * (NumberEdges - 1))
            ShadowEdges(2 * i + 1) = ShadowEdges(2 * (NumberEdges - 1) + 1)
        End If

        NumberEdges = NumberEdges - 1
        Exit Sub
    End If
Next

ShadowEdges(2 * NumberEdges) = v1
ShadowEdges(2 * NumberEdges + 1) = v2
NumberEdges = NumberEdges + 1
```

```
clsShadow - 2

End Sub

Private Function resta(v1 As D3DVECTOR, v2 As D3DVECTOR) As D3DVECTOR
resta.x = v1.x - v2.x
resta.y = v1.y - v2.y
resta.z = v1.z - v2.z
End Function

Private Function multiplica(v1 As D3DVECTOR, ByVal factor As Single) As D3D
VECTOR
multiplica.x = v1.x * factor
multiplica.y = v1.y * factor
multiplica.z = v1.z * factor
End Function

'--- Properties Tag ----

Public Property Get CubePosition() As D3DVECTOR
CubePosition.x = Position.x
CubePosition.y = Position.y
CubePosition.z = Position.z
End Property

Public Property Let CubePosition(vNewValue As D3DVECTOR)
Position.x = vNewValue.x
Position.y = vNewValue.y
Position.z = vNewValue.z
End Property
```

```
clsWinReg - 1

Option Explicit

Public Enum eHKEY
    HKEY_CLASSES_ROOT = &H80000000
    HKEY_CURRENT_USER = &H80000001
    HKEY_LOCAL_MACHINE = &H80000002
    HKEY_USERS = &H80000003
    HKEY_PERFORMANCE_DATA = &H80000004
    HKEY_CURRENT_CONFIG = &H80000005
    HKEY_DYN_DATA = &H80000006
    HKEY_FIRST = HKEY_CLASSES_ROOT
    HKEY_LAST = HKEY_DYN_DATA
End Enum
Public Enum eHKEYError
    ERROR_SUCCESS = 0
    ERROR_NONE = 0
    ERROR_FILE_NOT_FOUND = 2&
    ERROR_ACCESS_DENIED = 5&
    ERROR_OUTOFMEMORY = 6&
    ERROR_INVALID_PARAMETER = 7&
    ERROR_INVALID_PARAMETERS = 87&
    ERROR_MORE_DATA = 234&
    ERROR_NO_MORE_ITEMS = 259&
    ERROR_BADKEY = 1010&
End Enum

Public Enum eHKEYDataType
    REG_NONE = 0&
    REG_SZ = 1&
    REG_EXPAND_SZ = 2
    REG_BINARY = 3
    REG_DWORD = 4
    REG_DWORD_LITTLE_ENDIAN = 4
    REG_DWORD_BIG_ENDIAN = 5
    REG_LINK = 6
    REG_MULTI_SZ = 7
    REG_RESOURCE_LIST = 8
    REG_FULL_RESOURCE_DESCRIPTOR = 9
    REG_RESOURCE_REQUIREMENTS_LIST = 10
End Enum

Const SYNCHRONIZE = &H100000
Const READ_CONTROL = &H20000
Const STANDARD_RIGHTS_ALL = &H1F0000
Const STANDARD_RIGHTS_REQUIRED = &HF0000
Const STANDARD_RIGHTS_EXECUTE = (READ_CONTROL)
Const STANDARD_RIGHTS_READ = (READ_CONTROL)
Const STANDARD_RIGHTS_WRITE = (READ_CONTROL)

Public Enum eREGSAM
    KEY_QUERY_VALUE = &H1
    KEY_SET_VALUE = &H2
    KEY_CREATE_SUB_KEY = &H4
    KEY_ENUMERATE_SUB_KEYS = &H8
    KEY_NOTIFY = &H10
    KEY_CREATE_LINK = &H20
    KEY_READ = ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_S
UB_KEYS Or KEY_NOTIFY) And (Not SYNCHRONIZE))
    KEY_WRITE = ((STANDARD_RIGHTS_WRITE Or KEY_SET_VALUE Or KEY_CREATE_SUB_
KEY) And (Not SYNCHRONIZE))
```

```
    KEY_EXECUTE = ((KEY_READ) And (Not SYNCHRONIZE))
    KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or KEY_QUERY_VALUE Or KEY_SET_VA
LUE Or KEY_CREATE_SUB_KEY Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY Or KEY_CR
EATE_LINK) And (Not SYNCHRONIZE))
End Enum

Private Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Private Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias "RegQuery
InfoKeyA" _
    (ByVal hKey As Long, ByVal lpClass As String, lpcbClass As Long, _
    ByVal lpReserved As Long, lpcSubKeys As Long, lpcbMaxSubKeyLen As Long,
 _
    lpcbMaxClassLen As Long, lpcValues As Long, lpcbMaxValueNameLen As Long
, _
    lpcbMaxValueLen As Long, lpcbSecurityDescriptor As Long, _
    lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyE
xA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal ulOptions As Long, ByVal samDesired As Long, _
    phkResult As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32.dll" _
    (ByVal hKey As Long) As Long
Private Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValu
eA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpValueName As String, lpcbValueName As Long, _
    ByVal lpReserved As Long, lpType As Long, lpData As Any, _
    lpcbData As Long) As Long
Private Declare Function RegCreateKey Lib "advapi32.dll" Alias "RegCreateKe
yA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String, _
    phkResult As Long) As Long
Private Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKe
yA" _
    (ByVal hKey As Long, ByVal lpszSubKey As String) As Long

Private Declare Function RegDeleteValue Lib "advapi32.dll" Alias "RegDelete
ValueA" _
    (ByVal hKey As Long, ByVal szValueName As String) As Long
Private Declare Function RegEnumKey Lib "advapi32.dll" Alias "RegEnumKeyA"
_
    (ByVal hKey As Long, ByVal iSubKey As Long, _
    ByVal lpszName As String, ByVal cchName As Long) As Long
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyE
xA" _
    (ByVal hKey As Long, ByVal dwIndex As Long, _
    ByVal lpName As String, lpcbName As Long, _
    ByVal lpReserved As Long, ByVal lpClass As String, _
    lpcbClass As Long, lpftLastWriteTime As FILETIME) As Long
Private Declare Function RegQueryValue Lib "advapi32.dll" Alias "RegQueryVa
lueA" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal lpValue As String, lpcbValue As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQuery
ValueExA" _
```

```
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, lpdwType As Long, _
    lpbData As Any, cbData As Long) As Long
Private Declare Function RegSetValue Lib "advapi32.dll" Alias "RegSetValueA
" _
    (ByVal hKey As Long, ByVal lpSubKey As String, _
    ByVal dwType As Long, ByVal lpData As String, _
    ByVal cbData As Long) As Long
Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValu
eExA" _
    (ByVal hKey As Long, ByVal lpszValueName As String, _
    ByVal dwReserved As Long, ByVal fdwType As Long, _
    lpbData As Any, ByVal cbData As Long) As Long
Private Declare Function RegSaveKeyA Lib "advapi32.dll" _
    (ByVal hKey As Long, ByVal lpFile As String, _
    lpSecurityAttributes As Long) As Long

Public Function CreateKey(ByVal sKey As String) As eHKEYError
    Dim lRet As eHKEYError
    Dim hKey2 As Long
    Dim hKey As Long
    hKey = ParseKey(sKey, hKey)
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)
    If lRet <> ERROR_SUCCESS Then
        lRet = RegCreateKey(hKey, sKey, hKey2)
    End If
    Call RegCloseKey(hKey2)
    CreateKey = lRet
End Function

Public Function ExistKey(ByVal sKey As String) As Boolean
    Dim ret As eHKEYError
    Dim hKey2 As Long
    Dim hKey As eHKEY
    hKey = HKEY_LOCAL_MACHINE
    hKey = ParseKey(sKey, hKey)
    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)
    If ret = ERROR_SUCCESS Then
        ExistKey = True
        Call RegCloseKey(hKey2)
    Else
        ExistKey = False
    End If
End Function

Private Function DeleteKeyNT(hParentKey As Long, szKey As String) As Long
    Dim hKey As Long, lRet As eHKEYError, cSubKeys As Long, cbMaxSubKeyLen
As Long
    Dim cbSubKeyLen As Long, dwIndex As Long, ft As FILETIME, szTempSubKey
As String
    Dim szSubKey As String

    lRet = RegOpenKeyEx(hParentKey, szKey, 0, KEY_ALL_ACCESS, hKey)
    If Not lRet = ERROR_SUCCESS Then
        DeleteKeyNT = lRet
        Exit Function
    End If

    lRet = RegQueryInfoKey(hKey, vbNullString, 0&, 0, _
                           cSubKeys, cbMaxSubKeyLen, _
```

```
                            0&, 0&, 0&, 0&, 0&, ft)
    If Not lRet = ERROR_SUCCESS Then
        DeleteKeyNT = lRet
        Call RegCloseKey(hKey)
        Exit Function
    End If

    If cSubKeys > 0 Then
        dwIndex = cSubKeys - 1                          ' start at the end
        cbMaxSubKeyLen = cbMaxSubKeyLen + 1             ' +1 for the null term
inator
        szTempSubKey = String(cbMaxSubKeyLen, "*")    ' buffer to get name b
ack in
        Do
            cbSubKeyLen = cbMaxSubKeyLen * 2

            lRet = RegEnumKeyEx(hKey, dwIndex, szTempSubKey, cbSubKeyLen, 0
&, vbNullString, 0&, ft)
            If lRet = ERROR_SUCCESS Then
                szSubKey = left(szTempSubKey, cbSubKeyLen)
                Call DeleteKeyNT(hKey, szSubKey)
            End If
            dwIndex = dwIndex - 1                       ' enumerate backwards
        Loop While dwIndex >= 0
    End If

    Call RegCloseKey(hKey)

    lRet = RegDeleteKey(hParentKey, szKey)
    DeleteKeyNT = lRet
End Function

Public Function GetReg(ByVal sKey As String, Optional ByVal sValue As Strin
g = "", Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER, Optional ByVal bA
sString As Boolean = False) As Variant
    Dim lRet As Long, hKey2 As Long, rDT As eHKEYDataType
    Dim retDT As eHKEYDataType, lSize As Long, sData As String
    Dim aData() As Byte, lDWord As Long, i As Long, sTmp As String
    hKey = ParseKey(sKey, hKey)
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)
    ReDim aData(0)
    lDWord = 0
    sData = ""

    If lRet = ERROR_SUCCESS Then
        lRet = RegQueryValueEx(hKey2, sValue, 0&, retDT, 0&, lSize)
        Select Case retDT
        Case REG_DWORD
            lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, lDWord, lSize)
        Case REG_EXPAND_SZ, REG_SZ, REG_MULTI_SZ
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData,
 lSize)
            End If
        Case Else
            If lSize Then
                ReDim aData(lSize)
                lRet = RegQueryValueEx(hKey2, sValue, 0&, rDT, aData(0), lS
ize)
```

```
            End If
        End Select
        RegCloseKey hKey2
    End If
    Select Case retDT
    Case REG_DWORD
        If bAsString Then
            GetReg = "0x" & right$("00000000" & Hex$(lDWord), 8) & " (" & l
DWord & ")"
        Else
            GetReg = lDWord
        End If
    Case REG_EXPAND_SZ, REG_SZ
        GetReg = sData
    Case REG_MULTI_SZ
        GetReg = RTrimZero(sData, True)
    Case REG_BINARY
        If bAsString Then
            For i = 0 To UBound(aData) - 1
                sTmp = sTmp & right$("00" & Hex$(aData(i)), 2) & " "
            Next
            GetReg = sTmp
        Else
            GetReg = aData
        End If
    End Select
End Function

Private Function ParseKey(sKey As String, _
                          Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER
_
                          ) As eHKEY
    Dim i As Long
    Dim sRootKey As String
    sKey = Trim$(sKey)
    If right$(sKey, 1) = "\" Then
        sKey = left$(sKey, Len(sKey) - 1)
    End If
    i = InStr(sKey, "HKEY_")
    If i Then
        i = InStr(sKey, "\")
        If i Then
            sRootKey = left$(sKey, i - 1)
            sKey = mid$(sKey, i + 1)
        Else
            sRootKey = sKey
            sKey = ""
        End If
    ElseIf left$(sKey, 5) = "HKCR\" Then
        sRootKey = "HKEY_CLASSES_ROOT"
        sKey = mid$(sKey, 6)
    ElseIf left$(sKey, 5) = "HKCU\" Then
        sRootKey = "HKEY_CURRENT_USER"
        sKey = mid$(sKey, 6)
    ElseIf left$(sKey, 5) = "HKLM\" Then
        sRootKey = "HKEY_LOCAL_MACHINE"
        sKey = mid$(sKey, 6)
    ElseIf left$(sKey, 4) = "HKU\" Then
        sRootKey = "HKEY_USERS"
        sKey = mid$(sKey, 5)
```

```
    ElseIf left$(sKey, 5) = "HKCC\" Then
        sRootKey = "HKEY_CURRENT_CONFIG"
        sKey = mid$(sKey, 6)
    ElseIf left$(sKey, 5) = "HKDD\" Then
        sRootKey = "HKEY_DYN_DATA"
        sKey = mid$(sKey, 6)
    ElseIf left$(sKey, 5) = "HKPD\" Then
        sRootKey = "HKEY_PERFORMANCE_DATA"
        sKey = mid$(sKey, 6)
    Else
        Select Case hKey
        Case HKEY_FIRST To HKEY_LAST
        Case Else
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If
    If Len(sRootKey) Then
        Select Case sRootKey
        Case "HKEY_CLASSES_ROOT"
            hKey = HKEY_CLASSES_ROOT
        Case "HKEY_CURRENT_USER"
            hKey = HKEY_CURRENT_USER
        Case "HKEY_LOCAL_MACHINE"
            hKey = HKEY_LOCAL_MACHINE
        Case "HKEY_USERS"
            hKey = HKEY_USERS
        Case "HKEY_CURRENT_CONFIG"
            hKey = HKEY_CURRENT_CONFIG
        Case "HKEY_DYN_DATA"
            hKey = HKEY_DYN_DATA
        Case "HKEY_PERFORMANCE_DATA"
            hKey = HKEY_PERFORMANCE_DATA
        Case Else
            hKey = HKEY_CLASSES_ROOT
        End Select
    End If

    ParseKey = hKey
End Function

Public Function OpenKeyEx(ByVal hKey As Long, ByVal lpSubKey As String, _
                      ByVal ulOptions As Long, _
                      ByVal samDesired As eREGSAM, phkResult As Long) As
Long
    OpenKeyEx = RegOpenKeyEx(hKey, lpSubKey, 0&, samDesired, phkResult)
End Function

Public Function OpenKeyQuery(ByVal hKey As Long, ByVal lpSubKey As String,
ByVal ulOptions As Long, ByVal samDesired As eREGSAM, phkResult As Long) As
 Long
    OpenKeyQuery = RegOpenKeyEx(hKey, lpSubKey, 0&, KEY_QUERY_VALUE, phkRes
ult)
End Function

Public Function EnumValue(ByVal hKey As Long, ByVal dwIndex As Long, _
    lpValueName As String, lpcbValueName As Long, _
    lpReserved As Long, lpType As Long, lpData As Byte, _
    lpcbData As Long) As Long
    EnumValue = RegEnumValue(hKey, dwIndex, _
                          lpValueName, lpcbValueName, _
```

```
                            lpReserved, lpType, lpData, _
                            lpcbData)

End Function

Public Function CloseKey(ByVal hKey As Long) As Long
    CloseKey = RegCloseKey(hKey)
End Function

Public Function QueryInfoKey(ByVal hKey As Long, lpcbMaxValueNameLen As Lon
g) As Long
    Dim lpftLastWriteTime As FILETIME

    QueryInfoKey = RegQueryInfoKey(hKey, 0&, 0&, 0&, 0&, 0&, 0&, 0&, _
                    lpcbMaxValueNameLen, 0&, 0&, lpftLastWriteTime)
End Function

Public Function RegSetValue2(ByVal hKey As Long, ByVal lpSubKey As String,
_
                            ByVal dwType As eHKEYDataType, lpData As String
, _
                            ByVal cbData As Long) As Long
    cbData = Len(lpData)
    RegSetValue2 = RegSetValueEx(hKey, lpSubKey, 0&, REG_SZ, ByVal lpData,
cbData)
End Function

Public Function DeleteKeyWin95(ByVal hKey As Long, ByVal szKey As String) A
s Long

    DeleteKeyWin95 = RegDeleteKey(hKey, szKey)

End Function

Public Function SetReg(ByVal sKey As String, ByVal sName As String, _
                       Optional ByVal vValue As Variant, _
                       Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER, _
                       Optional ByVal RegDataType As eHKEYDataType = REG_SZ
, _
                       Optional ByVal bCreateKey As Boolean = True) As eHKE
YError
    Dim lRet As Long
    Dim hKey2 As Long
    Dim cbData As Long
    Dim aData() As Byte
    Dim sData As String
    Dim lData As Long

    hKey = ParseKey(sKey, hKey)

    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)

    If lRet <> ERROR_SUCCESS Then
        If bCreateKey Then
            lRet = RegCreateKey(hKey, sKey, hKey2)
        End If
    End If
    If lRet <> ERROR_SUCCESS Then
        SetReg = lRet
        Exit Function
```

```
    End If

    Select Case RegDataType
    Case REG_BINARY
        aData = vValue
        cbData = UBound(aData)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, aData(0), cbDat
a)
    Case REG_DWORD
        cbData = 4
        lData = CLng(vValue)
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, lData, cbData)
    Case REG_SZ, REG_EXPAND_SZ
        sData = CStr(vValue)
        If Len(sData) = 0 Then
            sData = ""
        End If
        cbData = Len(sData) + 1
        lRet = RegSetValueEx(hKey2, sName, 0&, RegDataType, ByVal sData, cb
Data)
    Case Else
    End Select
    lRet = RegCloseKey(hKey2)

    SetReg = lRet
End Function

Public Function DeleteKey(ByVal sKey As String, _
                          Optional ByVal sValue As String = "", _
                          Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER
_
                          ) As eHKEYError
    Dim lRet As eHKEYError
    Dim hKey2 As Long

    Select Case hKey
    Case HKEY_FIRST To HKEY_LAST
    Case Else
        hKey = HKEY_CLASSES_ROOT
    End Select

    hKey = ParseKey(sKey)

    If Len(sValue) = 0 Then
        DeleteKey = DeleteKeyNT(hKey, sKey)
        Exit Function
    End If
    lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)
    If lRet = ERROR_SUCCESS Then
        lRet = RegDeleteValue(hKey2, sValue)
        Call RegCloseKey(hKey2)
    End If

    DeleteKey = lRet
End Function
'
Public Function DeleteValue(ByVal sKey As String, _
                            ByVal sValue As String, _
                            Optional ByVal hKey As eHKEY = HKEY_CURRENT_USE
R _
```

```
                            ) As eHKEYError
    Dim lRet As eHKEYError
    Dim hKey2 As Long

    Select Case hKey
    Case HKEY_FIRST To HKEY_LAST
    Case Else
        hKey = HKEY_CLASSES_ROOT
    End Select

    hKey = ParseKey(sKey)
    lRet = ERROR_NONE
    If Len(sValue) Then
        lRet = RegOpenKeyEx(hKey, sKey, 0&, KEY_WRITE, hKey2)
        If lRet = ERROR_SUCCESS Then
            lRet = RegDeleteValue(hKey2, sValue)
            Call RegCloseKey(hKey2)
        End If
    End If
    DeleteValue = lRet
End Function

Private Function RTrimZero(ByVal sString As String, _
                           Optional ByVal PorElFinal As Boolean = False) As
 String
    Dim i As Long

    If PorElFinal Then
        For i = Len(sString) To 1 Step -1
            If mid$(sString, i, 1) = Chr$(0) Then
                sString = left$(sString, i - 1)
                Exit For
            End If
        Next
        For i = 1 To Len(sString)
            If mid$(sString, i, 1) = Chr$(0) Then
                Mid$(sString, i, 1) = " "
            End If
        Next

    Else
        i = InStr(sString, Chr$(0))
        If i Then
            sString = left$(sString, i - 1)
        End If
    End If
    RTrimZero = sString
End Function

Public Function RegSaveKey(ByVal sKey As String, ByVal lpFile As String) As
 Long
    Const stmpFic As String = "\tmp.reg"
    Dim hKey As Long
    Dim hKey2 As Long
    Dim ret As eHKEYError

    hKey = ParseKey(sKey)
    ret = RegOpenKeyEx(hKey, sKey, 0&, 0&, hKey2)
    On Local Error Resume Next
    If Len(Dir$(stmpFic, vbHidden + vbReadOnly + vbSystem)) Then
```

```
        SetAttr stmpFic, vbNormal
        Kill stmpFic
    End If
    ret = RegSaveKeyA(hKey2, stmpFic, 0&)
    If ret = ERROR_SUCCESS Then
        SetAttr stmpFic, vbNormal
        FileCopy stmpFic, lpFile
        Kill stmpFic
    End If
    Err = 0
    RegCloseKey hKey2
End Function

Public Function GetRegString(ByVal sKey As String, Optional ByVal sValue As
 String = "", Optional ByVal hKey As eHKEY = HKEY_CURRENT_USER) As String
    Dim ret As Long
    Dim hKey2 As Long
    Dim rDT As eHKEYDataType
    Dim sData As String
    Dim lSize As Long

    hKey = ParseKey(sKey, hKey)

    ret = RegOpenKeyEx(hKey, sKey, 0&, KEY_READ, hKey2)

    If ret = ERROR_SUCCESS Then
        ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, 0&, lSize)
        Select Case rDT
        Case REG_SZ, REG_EXPAND_SZ
            If lSize Then
                sData = String$(lSize - 1, Chr$(0))
                ret = RegQueryValueEx(hKey2, sValue, 0&, rDT, ByVal sData,
lSize)
            End If
        End Select
        RegCloseKey hKey2
    End If
    GetRegString = sData
End Function
```

StreamOGG - 1

```
Option Explicit

'--------  Implementation of the OGG VORBIS MUSIC decoder for VB
'--------  Uses vrobisfile.dll vorbis.dll ogg.dll and msvcrt.dll
'--------  Uses clsDLL class

'adapted by davidgf from Monoton Audio Lib (on planet source code) by [rm_c
ode]

Private Type vorbis_info
    version                 As Long
    Channels                As Long
    rate                    As Long
    bitrate_upper           As Long
    bitrate_nominal         As Long
    bitrate_lower           As Long
    bitrate_window          As Long
    codec_setup             As Long
End Type

Private Enum byte_packaging
    bp_little_endian = 0
    bp_big_endian = 1
End Enum

Private Enum word_size
    word_size_8_bit = 1
    word_size_16_bit = 2
End Enum

Private Const BUFFER_SIZE   As Long = 4096

Private clsMSVCRT           As clsDLL
Private clsOgg              As clsDLL
Private clsVorbis           As clsDLL
Private clsVorbisFile       As clsDLL
Private udtInfo             As vorbis_info
Private vf(64& * 1024&)     As Byte
Private btBuffer()          As Byte
Private lngPosInBuffer      As Long
Private lngBufferData       As Long
Private lngDuration         As Long
Private lngCurPos           As Long
Private hFile               As Long
Private blnEOS              As Boolean
Private blnReady            As Boolean
Public BufferSizeBytes As Long

Private Function ogg_init_libs() As Boolean
    Dim bln As Boolean

    Set clsMSVCRT = New clsDLL
    Set clsOgg = New clsDLL
    Set clsVorbis = New clsDLL
    Set clsVorbisFile = New clsDLL

    bln = True

    bln = bln And clsMSVCRT.LoadDLL("msvcrt.dll")
    bln = bln And clsOgg.LoadDLL("ogg.dll")
```

```
    bln = bln And clsVorbis.LoadDLL("vorbis.dll")
    bln = bln And clsVorbisFile.LoadDLL("vorbisfile.dll")

    If Not bln Then Exit Function

    clsMSVCRT.IsCDECL = True
    clsOgg.IsCDECL = True
    clsVorbis.IsCDECL = True
    clsVorbisFile.IsCDECL = True

    ogg_init_libs = True
End Function

Private Sub ogg_free_libs()
    clsVorbisFile.UnloadDLL
    clsVorbis.UnloadDLL
    clsOgg.UnloadDLL
    clsMSVCRT.UnloadDLL
End Sub

Public Function FillBuffer() As Boolean
    Dim lngCurrentSection   As Long
    Dim lngRet              As Long

    lngRet = ov_read(VarPtr(vf(0)), VarPtr(btBuffer(0)), BUFFER_SIZE, bp_li
ttle_endian, word_size_16_bit, True, lngCurrentSection)

    If lngRet = 0 Then
        blnEOS = True
    ElseIf lngRet > 0 Then
        lngCurPos = lngCurPos + lngRet * 1000 / udtInfo.rate / 2 / udtInfo.
Channels
        lngBufferData = lngRet
        lngPosInBuffer = 0
        FillBuffer = True
    End If
End Function

Private Sub Class_Initialize()
    blnReady = ogg_init_libs
    ReDim btBuffer(BUFFER_SIZE - 1) As Byte
End Sub

Private Sub Class_Terminate()
    StreamClose
    ogg_free_libs
End Sub

Public Property Get BitsPerSample() As Integer
    BitsPerSample = 16
End Property

Public Property Get BitsPerSecond() As Long
    BitsPerSecond = udtInfo.bitrate_nominal
End Property

Public Property Get Channels() As Integer
    Channels = udtInfo.Channels
End Property
```

```
Public Property Get Duration() As Long
    Duration = lngDuration
End Property

Public Property Get Position() As Long
    Position = lngCurPos
End Property

Public Property Get SamplesPerSecond() As Long
    SamplesPerSecond = udtInfo.rate
End Property

Public Property Get EndOfStream() As Boolean
    EndOfStream = blnEOS
End Property

Public Function StreamClose() As SND_RESULT
    If hFile <> 0 Then
        ov_clear VarPtr(vf(0))
        clsMSVCRT.CallFunc "fclose", hFile
        hFile = 0
        blnEOS = False
        lngBufferData = 0
        lngPosInBuffer = 0

        StreamClose = SND_ERR_SUCCESS
    Else
        StreamClose = SND_ERR_INVALID_SOURCE
    End If
End Function

Public Function StreamOpen(ByVal Source As String) As SND_RESULT
    Dim btFile()    As Byte
    Dim btMode(2)   As Byte
    Dim lngRet      As Long

    If Not blnReady Then
        StreamOpen = SND_ERR_INTERNAL
        Exit Function
    End If

    StreamClose

    btFile = StrConv(Source & Chr$(0), vbFromUnicode)

                            ' file...
    btMode(0) = Asc("r")    ' read
    btMode(1) = Asc("b")    ' binary

    hFile = clsMSVCRT.CallFunc("fopen", VarPtr(btFile(0)), VarPtr(btMode(0)
))

    If hFile = 0 Then
        StreamOpen = SND_ERR_INVALID_SOURCE
        Exit Function
    End If

    If ov_open(hFile, VarPtr(vf(0)), 0, 0) < 0 Then
        StreamOpen = SND_ERR_INVALID_SOURCE
        clsMSVCRT.CallFunc "fclose", hFile
```

```
        Exit Function
    End If

    udtInfo = ov_info(VarPtr(vf(0)), -1)

    If udtInfo.rate < 1 Then
        ov_clear VarPtr(vf(0))
        clsMSVCRT.CallFunc "fclose", hFile
        StreamOpen = SND_ERR_INVALID_SOURCE
        Exit Function
    End If

    If udtInfo.Channels > 2 Then
        ov_clear VarPtr(vf(0))
        clsMSVCRT.CallFunc "fclose", hFile
        StreamOpen = SND_ERR_INVALID_SOURCE
        Exit Function
    End If

    lngRet = ov_pcm_total(VarPtr(vf(0)), -1)
    BufferSizeBytes = lngRet

    lngDuration = lngRet / udtInfo.rate * 1000
    lngCurPos = 0

    StreamOpen = SND_ERR_SUCCESS
End Function

Public Function StreamRead(ByVal buffer_ptr As Long, ByVal buffer_len As Lo
ng, buffer_read As Long) As SND_RESULT
    If Not blnReady Then
        StreamRead = SND_ERR_INTERNAL
        Exit Function
    End If

    StreamRead = SND_ERR_SUCCESS

    buffer_read = 0

    Do While buffer_read < buffer_len

         If lngBufferData = 0 Then
            If Not FillBuffer Then
                StreamRead = SND_ERR_END_OF_STREAM
                Exit Function
            End If

        ElseIf (lngBufferData - lngPosInBuffer) < (buffer_len - buffer_read
) Then
            If 0 < (lngBufferData - lngPosInBuffer) Then

                If 0 = IsBadReadPtr(ByVal VarPtr(btBuffer(0)) + lngPosInBuf
fer, _
                                    lngBufferData - lngPosInBuffer) Then

                    If 0 = IsBadWritePtr(ByVal buffer_ptr + buffer_read, _
                                         lngBufferData - lngPosInBuffer) Th
en

                        CpyMem ByVal buffer_ptr + buffer_read, _
```

```
                                ByVal VarPtr(btBuffer(0)) + lngPosInBuffer,
_
                                lngBufferData - lngPosInBuffer

                    End If
                End If

                buffer_read = buffer_read + (lngBufferData - lngPosInBuffer
)
            End If

            If Not FillBuffer Then
                StreamRead = SND_ERR_END_OF_STREAM
                Exit Function
            End If

        Else
            If 0 = IsBadReadPtr(ByVal VarPtr(btBuffer(0)) + lngPosInBuffer,
 _
                                buffer_len - buffer_read) Then

                If 0 = IsBadWritePtr(ByVal buffer_ptr + buffer_read, _
                                     buffer_len - buffer_read) Then

                    CpyMem ByVal buffer_ptr + buffer_read, _
                           ByVal VarPtr(btBuffer(0)) + lngPosInBuffer, _
                           buffer_len - buffer_read

                End If
            End If

            lngPosInBuffer = lngPosInBuffer + (buffer_len - buffer_read)
            buffer_read = buffer_read + (buffer_len - buffer_read)

        End If
    Loop
End Function

Public Function StreamSeek(ByVal Value As Long, ByVal seek_mode As SND_SEEK
_MODE) As SND_RESULT
    Dim dblTime     As Double

    If Not blnReady Then
        StreamSeek = SND_ERR_INTERNAL
        Exit Function
    End If

    Select Case seek_mode

        Case SND_SEEK_PERCENT
            If Value < 0 Or Value > 99 Then
                StreamSeek = SND_ERR_OUT_OF_RANGE
                Exit Function
            End If

            dblTime = (Duration / 1000) / 100 * Value

        Case SND_SEEK_SECONDS
            If Value < 0 Or Value > (Duration / 1000) Then
                StreamSeek = SND_ERR_OUT_OF_RANGE
```

```
                Exit Function
            End If

            dblTime = Value

    End Select

    lngCurPos = dblTime * 1000
    ov_time_seek VarPtr(vf(0)), dblTime

    lngBufferData = 0
    lngPosInBuffer = 0

    blnEOS = False

    StreamSeek = SND_ERR_SUCCESS
End Function

Private Function ov_time_seek(ByVal ovf_struct As Long, ByVal seconds As Do
uble) As Long
    Dim lngTimeLo   As Long
    Dim lngTimeHi   As Long

    CpyMem lngTimeLo, seconds, 4
    CpyMem lngTimeHi, ByVal VarPtr(seconds) + 4, 4

    ov_time_seek = clsVorbisFile.CallFunc("ov_time_seek", ovf_struct, lngTi
meLo, lngTimeHi)
End Function

Private Function ov_read(ByVal ovf_struct As Long, ByVal output_ptr As Long
, _
    ByVal Length As Long, ByVal bigendianp As byte_packaging, _
    ByVal word As word_size, ByVal signed As Boolean, _
    ByRef bitstream As Long) As Long

    ov_read = clsVorbisFile.CallFunc("ov_read", ovf_struct, output_ptr, Len
gth, bigendianp, word, Abs(signed), VarPtr(bitstream))
End Function

Private Function ov_pcm_total(ByVal ovf_struct As Long, ByVal link As Long)
 As Long
    ov_pcm_total = clsVorbisFile.CallFunc("ov_pcm_total", ovf_struct, link)
End Function

Private Function ov_info(ByVal ovf_struct As Long, ByVal link As Long) As v
orbis_info
    Dim ptr     As Long
    Dim udt     As vorbis_info

    ptr = clsVorbisFile.CallFunc("ov_info", ovf_struct, link)
    If ptr = 0 Then Exit Function

    CpyMem udt, ByVal ptr, Len(udt)

    ov_info = udt
End Function

Private Function ov_open(ByVal hFile As Long, ByVal ovf_struct As Long, ByV
al initial_ptr As Long, ByVal ibytes As Long) As Long
```

```
ov_open = clsVorbisFile.CallFunc("ov_open", hFile, ovf_struct, initial_ptr,
 ibytes)
End Function

Private Sub ov_clear(ByVal ovf_struct As Long)
clsVorbisFile.CallFunc "ov_clear", ovf_struct
End Sub
```

```
/*

  engine core Maths Library

  Created by David GF using code from the

    ColDet - C++ 3D Collision Detection Library
    Copyright (C) 2000   Amir Geva

  Under GNU LESSER GENERAL PUBLIC LICENSE (Version 2.1, February 1999)

          http://photoneffect.com/coldet/

*/

#include "stdafx.h"
#include "coldet.h"
#include <math.h>


#pragma warning(disable:4244)


typedef struct  D3DVECTOR
    {
    float x;
    float y;
    float z;
}   D3DVECTOR;


typedef struct  salida
    {
    int respuesta;
    D3DVECTOR puntocolision;
}   salida;


D3DVECTOR CrossProduct(const D3DVECTOR& v1, const D3DVECTOR& v2)
{
    D3DVECTOR resultado;
    resultado.x=v1.y*v2.z-v2.y*v1.z;
    resultado.y=v1.z*v2.x-v2.z*v1.x;
    resultado.z=v1.x*v2.y-v2.x*v1.y;
    return resultado;
}

float DotProduct(const D3DVECTOR& v1, const D3DVECTOR& v2)
{
    return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

D3DVECTOR resta(const D3DVECTOR &v1, const D3DVECTOR &v2)
{
    D3DVECTOR resultado;
    resultado.x=v1.x-v2.x;
    resultado.y=v1.y-v2.y;
    resultado.z=v1.z-v2.z;
    return resultado;
}

D3DVECTOR suma(const D3DVECTOR& v1, const D3DVECTOR& v2)
{
    D3DVECTOR resultado;
    resultado.x=v1.x+v2.x;
    resultado.y=v1.y+v2.y;
    resultado.z=v1.z+v2.z;
    return resultado;
}


D3DVECTOR multiplica(const D3DVECTOR& v1, long fact)
{
```

```
    D3DVECTOR resultado;
    resultado.x=v1.x*fact;
    resultado.y=v1.y*fact;
    resultado.z=v1.z*fact;
    return resultado;
}

D3DVECTOR Normalize(const D3DVECTOR& v1)
{
    D3DVECTOR resultado;
    float module;
    module=sqrt(v1.x*v1.x + v1.y*v1.y + v1.z*v1.z);
    resultado.x=v1.x/module;
    resultado.y=v1.y/module;
    resultado.z=v1.z/module;
    return resultado;
}

D3DVECTOR copyv (const D3DVECTOR& v1)
{
    D3DVECTOR resultado;
    resultado.x = v1.x;
    resultado.y = v1.y;
    resultado.z = v1.z;
    return resultado;
}

int vcompare (const D3DVECTOR& v1, const D3DVECTOR& v2)
{
    return ((v1.x == v2.x) && (v1.y == v2.y) && (v1.z == v2.z));
}

/*
     Processa el moviment horitzontal, les col·lisions horitzontals
     i la càmera (amb la seva corresponent col·lisió)
*/


  declspec( dllexport )  stdcall process(D3DVECTOR *cam, D3DVECTOR *tri, long num
tri, D3DVECTOR *pos, D3DVECTOR *pos2, D3DVECTOR *upv, double distance, double ang
le, double angleh, float disfromcol, float interpolationspeed, float midh, float
hih, float avrframe, float speed, float dimensions, long *tris)
{
    double CameraDistanceProj, CameraRelX, CameraRelZ, pi;
    float point1[3],point2[3];
    long i;
    double collx,colly,collz;
    pi = 3.14159265358979;

    pos->x = sin(angleh * pi / 180) * speed * avrframe / 1000 + pos->x ;
    pos->z = cos(angleh * pi / 180) * speed * avrframe / 1000 + pos->z;


    D3DVECTOR tripos[6];
    CollisionModel3D* model= newCollisionModel3D(TRUE);
    CollisionModel3D* model2= newCollisionModel3D(TRUE);

    model->setTriangleNumber (numtri);
    model2->setTriangleNumber (2);

    double var;
    var = cos(pi/6) * dimensions;

    tripos[0].x = pos->x + dimensions;
    tripos[0].y = pos->y + midh;
    tripos[0].z = pos->z;
    tripos[1].x = pos->x - dimensions;
    tripos[1].y = pos->y + midh;
    tripos[1].z = pos->z + var;
    tripos[2].x = pos->x - dimensions;
    tripos[2].y = pos->y + midh;
    tripos[2].z = pos->z - var;
    tripos[3].x = pos->x - dimensions;
```

```
    tripos[3].y = pos->y + midh;
    tripos[3].z = pos->z;
    tripos[4].x = pos->x + dimensions;
    tripos[4].y = pos->y + midh;
    tripos[4].z = pos->z + var;
    tripos[5].x = pos->x + dimensions;
    tripos[5].y = pos->y + midh;
    tripos[5].z = pos->z - var;

    D3DVECTOR vec, cam2,cam original;
    cam_original.x  = cam->x;cam_original.y  = cam->y;cam_original.z  = cam->z;

    CameraDistanceProj = (distance * cos(angle * pi / 180));
    CameraRelX = CameraDistanceProj * sin(angleh * pi / 180);
    CameraRelZ = CameraDistanceProj * cos(angleh * pi / 180);
    cam->x = CameraRelX + pos->x;
    cam->z = CameraRelZ + pos->z;
    cam->y = pos->y + (distance * sin(angle * pi / 180));
    cam2.x  = cam->x;cam2.y  = cam->y;cam2.z  = cam->z;

    BOOL chocan=FALSE;
    double varx1,varx2,varz1,varz2;
    var = pos->y + midh;
    varx1=pos->x+distance;
    varx2=pos->x-distance;
    varz1=pos->z+distance;
    varz2=pos->z-distance;
    double var2=cam->y;

    long tricounter;
    tricounter=0;

    /* Optimitació! Es descarten triangles utilitzant una bounding box */

    for (i=0;i<(numtri*3);i+=3) {
        if ( (tri[i].y > var) || (tri[i+1].y > var) || (tri[i+2].y > var) ) {
        if ( (tri[i].y < var2) || (tri[i+1].y < var2) || (tri[i+2].y < var2) ) {
        if ( (tri[i].x > varx2) || (tri[i+1].x > varx2) || (tri[i+2].x > varx2) )
 {
        if ( (tri[i].x < varx1) || (tri[i+1].x < varx1) || (tri[i+2].x < varx1) )
 {
        if ( (tri[i].z > varz2) || (tri[i+1].z > varz2) || (tri[i+2].z > varz2) )
 {
        if ( (tri[i].z < varz1) || (tri[i+1].z < varz1) || (tri[i+2].z < varz1) )
 {
            model->addTriangle (tri[i].x,tri[i].y,tri[i].z,tri[i+1].x,tri[i+1].y,
tri[i+1].z,tri[i+2].x,tri[i+2].y,tri[i+2].z);
            tris[tricounter++]=i;
        } } } } } }
    }
    for (i=0;i<6;i+=3) {
        model2->addTriangle (tripos[i].x,tripos[i].y,tripos[i].z,tripos[i+1].x,tr
ipos[i+1].y,tripos[i+1].z,tripos[i+2].x,tripos[i+2].y,tripos[i+2].z);
    }
    model->finalize ();
    model2->finalize ();

    int firstt, secondt;

    if (model2->collision (model)==TRUE) {
        pos->x = pos2->x;
        pos->z = pos2->z;

        var = cos(pi/6) * dimensions;

        tripos[0].x = pos->x + dimensions;
        tripos[0].y = pos->y + midh;
        tripos[0].z = pos->z;
        tripos[1].x = pos->x - dimensions;
        tripos[1].y = pos->y + midh;
        tripos[1].z = pos->z + var;
        tripos[2].x = pos->x - dimensions;
        tripos[2].y = pos->y + midh;
```

```
        tripos[2].z = pos->z - var;
        tripos[3].x = pos->x - dimensions;
        tripos[3].y = pos->y + midh;
        tripos[3].z = pos->z;
        tripos[4].x = pos->x + dimensions;
        tripos[4].y = pos->y + midh;
        tripos[4].z = pos->z + var;
        tripos[5].x = pos->x + dimensions;
        tripos[5].y = pos->y + midh;
        tripos[5].z = pos->z - var;

        CollisionModel3D* model4= newCollisionModel3D(TRUE);

        model4->setTriangleNumber (2);

        for (i=0;i<6;i+=3) {
            model4->addTriangle (tripos[i].x,tripos[i].y,tripos[i].z,tripos[i+1].
x,tripos[i+1].y,tripos[i+1].z,tripos[i+2].x,tripos[i+2].y,tripos[i+2].z);
        }
        model4->finalize ();

        if (model4->collision (model)==TRUE) {

            model4->getCollidingTriangles (firstt, secondt);

            D3DVECTOR a,b,c,h,i,n;
            a=tri[tris[secondt]];
            b=tri[tris[secondt]+1];
            c=tri[tris[secondt]+2];

            h.x = b.x - a.x;
            h.y = b.y - a.y;
            h.z = b.z - a.z;

            i.x = c.x - a.x;
            i.y = c.y - a.y;
            i.z = c.z - a.z;

            n=CrossProduct (h,i);
            n=Normalize(n);
            n.x = n.x * dimensions;
            n.y = n.y * dimensions;
            n.z = n.z * dimensions;

            float pn1[3],pn2[3];
            pn1[0]=pos->x+n.x;
            pn1[1]=pos->y+midh+n.y;
            pn1[2]=pos->z+n.z;
            pn2[0]=-n.x;
            pn2[1]=-n.y;
            pn2[2]=-n.z;

            float point5[3];
            model->getCollisionPoint (point5);
            pos->x = point5[0]+n.x;
            pos->z = point5[2]+n.z;
        }

        delete model4;
    }

    vec.x = pos->x - cam->x;
    vec.y = pos->y - cam->y;
    vec.z = pos->z - cam->z;

    vec.x = vec.x / distance;
    vec.y = vec.y / distance;
    vec.z = vec.z / distance;

    point1[0] = pos->x;
    point1[1] = pos->y+hih;
    point1[2] = pos->z;
```

```
    point2[0] = cam->x - point1[0];
    point2[1] = cam->y - point1[1];
    point2[2] = cam->z - point1[2];

    chocan=FALSE;

    chocan=model->rayCollision (point1,point2,true,0.0f);

    float point[3];
    BOOL camb = FALSE;
    double dis2;

    upv->x=0;upv->z=0;upv->y=1;

    if (chocan == TRUE) {
        model->getCollisionPoint (point);
        collx = point[0];colly = point[1];collz = point[2];

        dis2 = sqrt( (pos->x-collx) * (pos->x-collx) + (pos->y-colly) * (pos->y-c
olly) + (pos->z-collz) * (pos->z-collz));

        if ((disfromcol + distance) > dis2) {
            cam->x = collx + vec.x * disfromcol;
            cam->z = collz + vec.z * disfromcol;

            point1[0] = cam->x;
            point1[1] = pos->y+hih;
            point1[2] = cam->z;

            point2[0] = 0;
            point2[1] = 1;
            point2[2] = 0;

            camb=model->rayCollision(point1,point2,true);

            if (camb==TRUE) {
            model->getCollisionPoint (point);
            if (point[1]<cam->y) {
                cam->y = colly - vec.y * disfromcol;

                point1[0] = pos->x;
                point1[1] = pos->y+hih;
                point1[2] = pos->z;

                point2[0] = cam->x - point1[0];
                point2[1] = cam->y - point1[1];
                point2[2] = cam->z - point1[2];

                if (model->rayCollision (point1,point2,true,0.0f) == TRUE) {
                    model->getCollisionPoint (point);
                    cam->x = point[0] + vec.x * disfromcol;
                    cam->z = point[2] + vec.z * disfromcol;
                    cam->y = point[1] + vec.y * disfromcol;
                }
            }
            }
        }
    }



    double interpolation = interpolationspeed * avrframe / 1000;

    cam->x = (cam->x-cam original.x) * interpolation + cam original.x;
    cam->y = (cam->y-cam original.y) * interpolation + cam original.y;
    cam->z = (cam->z-cam_original.z) * interpolation + cam_original.z;

    double v1x,v1y,v2x,v2y;
    double angulo;

    v1x = cam->x - pos->x;
    v1y = cam->z - pos->z;
```

```
    v2x = sin(angleh * pi / 180);
    v2y = cos(angleh * pi / 180);

    angulo = acos( (v1x * v2x + v1y * v2y)/( sqrt(v1x*v1x+v1y*v1y) * sqrt(v2x*v2x
+v2y*v2y) ));

    if ((angulo / pi * 180) > 135) { upv->y = -1; }

    delete model;delete model2;

    return 0;
}


/*
     Processa el moviment vertical
*/


__declspec( dllexport ) _stdcall hprocess(D3DVECTOR *tri, D3DVECTOR *point, long
numtri, salida *collide)
{
    CollisionModel3D* model= newCollisionModel3D();

    double  pi = 3.14159265358979;

    model->setTriangleNumber (numtri);

    long i;
    BOOL chocan=FALSE;
    double var=point->y;


    /* Optimitació! Es descarten triangles */
    for (i=0;i<(numtri*3);i=i+3) {
        if ( (tri[i].y < var) || (tri[i+1].y < var) || (tri[i+2].y < var) ) {
        if ( !( (tri[i].x < point->x) && (tri[i+1].x < point->x) && (tri[i+2].x <
 point->x) ) ) {
        if ( !( (tri[i].x > point->x) && (tri[i+1].x > point->x) && (tri[i+2].x >
 point->x) ) ) {
        if ( !( (tri[i].z < point->z) && (tri[i+1].z < point->z) && (tri[i+2].z <
 point->z) ) ) {
        if ( !( (tri[i].z > point->z) && (tri[i+1].z > point->z) && (tri[i+2].z >
 point->z) ) ) {
            model->addTriangle (tri[i].x,tri[i].y,tri[i].z,tri[i+1].x,tri[i+1].y,
tri[i+1].z,tri[i+2].x,tri[i+2].y,tri[i+2].z);
        } } } } }
    }

    model->finalize ();

    float point1[3],point2[3];
    float outpoint[3];
    point2[0]=0;
    point2[1]=-1;
    point2[2]=0;

    point1[0]=point->x;
    point1[1]=point->y;
    point1[2]=point->z;

    chocan=model->rayCollision (point1,point2,true,0.0f);

    if (chocan == TRUE) {
        collide->respuesta=1;
        model->getCollisionPoint (outpoint);
        collide->puntocolision.x = point->x;
        collide->puntocolision.y = outpoint[1];
        collide->puntocolision.z = point->z;

    }else{ collide->respuesta =0; }

    delete model;
```

```
    return 0;
}

/*
     Col·lisió segment - triangles
*/

  declspec( dllexport )  stdcall segintersect(D3DVECTOR *origin, D3DVECTOR *direc
tion, D3DVECTOR *tri, long numtri, long numsegs, salida *collide)
{
    long i;
    BOOL chocan=FALSE;
    CollisionModel3D* model= newCollisionModel3D();

    model->setTriangleNumber (numtri);

    for (i=0;i<(numtri*3);i=i+3) {
        model->addTriangle (tri[i].x,tri[i].y,tri[i].z,tri[i+1].x,tri[i+1].y,tri[
i+1].z,tri[i+2].x,tri[i+2].y,tri[i+2].z);
    }

    model->finalize ();

    float point1[3],point2[3],outpoint[3];

    for (i=0;i<(numsegs);i=i+1) {
        point1[0]=origin[i].x;
        point1[1]=origin[i].y;
        point1[2]=origin[i].z;

        point2[0]=direction[i].x;
        point2[1]=direction[i].y;
        point2[2]=direction[i].z;

        chocan=model->rayCollision (point1,point2,true,0.0f);

        if (chocan == TRUE) {
            collide[i].respuesta=1;
            model->getCollisionPoint (outpoint);
            collide[i].puntocolision.x = outpoint[0];
            collide[i].puntocolision.y = outpoint[1];
            collide[i].puntocolision.z = outpoint[2];

        }else{ collide[i].respuesta =0; }
    }

    delete model;
}

/*
     Col·lisió segments - triangles
*/

  declspec( dllexport )  stdcall segintersectfast(D3DVECTOR *origin, D3DVECTOR *d
irection, D3DVECTOR *tri, long numtri, long numsegs, salida *collide)
{
    CollisionModel3D* model= newCollisionModel3D();

    model->setTriangleNumber (numtri);

    long i; long j;
    BOOL chocan=FALSE;

    for (i=0;i<(numtri*3);i=i+3) {
        for (j=0;j<numsegs;j=j+1) {
            /* Optimitació! Es descarten triangles */
            if ( (tri[i].y < origin[j].y) || (tri[i+1].y < origin[j].y) || (tri[i
+2].y < origin[j].y) ) {
            if ( !( (tri[i].x < origin[j].x) && (tri[i+1].x < origin[j].x) && (tr
i[i+2].x < origin[j].x) ) ) {
            if ( !( (tri[i].x > origin[j].x) && (tri[i+1].x > origin[j].x) && (tr
i[i+2].x > origin[j].x) ) ) {
```

```
            if ( !( (tri[i].z < origin[j].z) && (tri[i+1].z < origin[j].z) && (tr
i[i+2].z < origin[j].z) ) ) {
            if ( !( (tri[i].z > origin[j].z) && (tri[i+1].z > origin[j].z) && (tr
i[i+2].z > origin[j].z) ) ) {
                model->addTriangle (tri[i].x,tri[i].y,tri[i].z,tri[i+1].x,tri[i+1
].y,tri[i+1].z,tri[i+2].x,tri[i+2].y,tri[i+2].z);
            } } } } }
        }
    }

    model->finalize ();

    float point1[3],point2[3],outpoint[3];

    for (i=0;i<(numsegs);i=i+1) {
        point1[0]=origin[i].x;
        point1[1]=origin[i].y;
        point1[2]=origin[i].z;

        point2[0]=direction[i].x;
        point2[1]=direction[i].y;
        point2[2]=direction[i].z;

        chocan=model->rayCollision (point1,point2,true,0.0f);

        if (chocan == TRUE) {
            collide[i].respuesta=1;
            model->getCollisionPoint (outpoint);
            collide[i].puntocolision.x = outpoint[0];
            collide[i].puntocolision.y = outpoint[1];
            collide[i].puntocolision.z = outpoint[2];

        }else{ collide[i].respuesta =0; }
    }

    delete model;

}

  declspec( dllexport ) _stdcall computenormals(D3DVECTOR *verts, long numverts,
D3DVECTOR *normals)
{
    long i, tric = 0;
    D3DVECTOR vc1, vc2;

    for (i=0 ; i<numverts; i=i+3 )
    {
        vc1 = resta(verts[i+2],verts[i+1]);
        vc2 = resta(verts[i+1],verts[i]);

        normals[tric++] = Normalize(CrossProduct(vc1,vc2));
    }
}


VOID AddEdge( long* pEdges, long& dwNumEdges, long v0, long v1 )
{
    // Remove interior edges (which appear in the list twice)
    for( long i=0; i < dwNumEdges; i++ )
    {
        if( ( pEdges[2*i+0] == v0 && pEdges[2*i+1] == v1 ) ||
            ( pEdges[2*i+0] == v1 && pEdges[2*i+1] == v0 ) )
        {
            if( dwNumEdges > 1 )
            {
                pEdges[2*i+0] = pEdges[2*(dwNumEdges-1)+0];
                pEdges[2*i+1] = pEdges[2*(dwNumEdges-1)+1];
            }
            dwNumEdges--;
            return;
        }
    }
```

```
    pEdges[2*dwNumEdges+0] = v0;
    pEdges[2*dwNumEdges+1] = v1;
    dwNumEdges++;
}

  declspec( dllexport )  stdcall extrudeshadow(long proj, D3DVECTOR *tris, long n
umtri, D3DVECTOR *light, D3DVECTOR *trisout, long& numtrisout)
{
        long* pEdges = new long[numtri*6];
        long dwNumEdges = 0;

        D3DVECTOR lightbig;
        lightbig = Normalize(*light);
        lightbig.x = lightbig.x * proj;
        lightbig.y = lightbig.y * proj;
        lightbig.z = lightbig.z * proj;

        long i;
        numtrisout=0;

        D3DVECTOR vnormal, vc1, vc2;

        for (i=0; i<numtri; i=i+3)
        {
            vc1 = resta(tris[i+2],tris[i+1]);
            vc2 = resta(tris[i+1],tris[i]);

            vnormal = Normalize(CrossProduct(vc1,vc2));

            if (DotProduct(vnormal,lightbig)>=0) {
                AddEdge (pEdges, dwNumEdges, i , i+1 );
                AddEdge (pEdges, dwNumEdges, i+1 , i+2 );
                AddEdge (pEdges, dwNumEdges, i+2 , i );

                vnormal = multiplica(vnormal, -0.002);

                tris[i]=resta(tris[i],vnormal);
                tris[i+1]=resta(tris[i+1],vnormal);
                tris[i+2]=resta(tris[i+2],vnormal);

                trisout[numtrisout++]=resta(tris[i],lightbig);
                trisout[numtrisout++]=resta(tris[i+1],lightbig);
                trisout[numtrisout++]=resta(tris[i+2],lightbig);

                trisout[numtrisout++]=copyv(tris[i+2]);
                trisout[numtrisout++]=copyv(tris[i+1]);
                trisout[numtrisout++]=copyv(tris[i]);
            }
        }

        for (i=0; i<dwNumEdges; i++)
        {
            D3DVECTOR vec1,vec2,vec3,vec4;
            vec1 = copyv(tris[pEdges[2*i]]);
            vec2 = copyv(tris[pEdges[2*i+1]]);
            vec3 = resta(vec1,lightbig);
            vec4 = resta(vec2,lightbig);

            trisout[numtrisout++]=vec1;
            trisout[numtrisout++]=vec2;
            trisout[numtrisout++]=vec3;

            trisout[numtrisout++]=vec2;
            trisout[numtrisout++]=vec4;
            trisout[numtrisout++]=vec3;
        }

        delete pEdges;
}


  declspec( dllexport )  stdcall createindices(D3DVECTOR *vertices, long numverts
, D3DVECTOR *vertsout, long& numvertsout, long *indices, long& numindicesout)
```

```
{
    long x, y, goout, match;
    numvertsout=0;
    for (x=0; x<numverts; x++)
    {
        goout=0;
        y = x+1;
        while (y<numvertsout)
        {
            if ( vcompare(vertices[x],vertices[y]) )
            {
                y = numvertsout;
                goout=1;
                match=y;
            }
        y++;
        }

        if (goout!=1 || x==0) {
            vertsout[numvertsout] = vertices[x];
            indices[numindicesout++]=numvertsout;
            numvertsout++;
        }else{
            indices[numindicesout++]=match;
        }
    }
}
```