UNIVERSIDADE DO CONTESTADO – UNC
CURSO DE CIÊNCIA DA COMPUTAÇÃO

EDIRLEI EVERSON SOARES DE LIMA

3D GAME BUILDER: UMA GAME ENGINE PARA A CRIAÇÃO DE JOGOS 3D

PORTO UNIÃO
2008

EDIRLEI EVERSON SOARES DE LIMA

3D GAME BUILDER: UMA GAME ENGINE PARA A CRIAÇÃO DE JOGOS 3D

Trabalho de conclusão de curso apresentado como exigência para obtenção do titulo de bacharel em Ciência da Computação pela Universidade do Contestado - UnC Núcleo de Porto União, sob orientação do professor Ms. Pedro Luiz de Paula Filho.

PORTO UNIÃO
2008

3D GAME BUILDER: UMA GAME ENGINE PARA A CRIAÇÃO DE JOGOS 3D

EDIRLEI EVERSON SOARES DE LIMA

Este trabalho de conclusão de curso foi submetido ao processo de avaliação pela Banca examinadora para obtenção do Titulo (Grau) de:

Bacharel em Ciência da Computação

E aprovada na sua versão final em _____ (data), atendendo às normas da legislação vigente da Universidade do Contestado e Coordenação do Curso de Ciência da Computação.

_____
Nome do coordenador do curso

BANCA EXAMINADORA:

_____
Nome do presidente

_____
Membro

_____
Membro

_____
Membro Suplente (se houver)

Resumo

Os jogos eletrônicos movimentam o bilionário mundo das produtoras e tem causado uma revolução nas formas de entretenimento, superando até mesmo o mercado cinematográfico. Porém o desenvolvimento de um jogo não é uma tarefa simples e pode demorar anos, envolver centenas de pessoas e custar milhões de dólares. Para facilitar e agilizar este processo existem softwares específicos para criação de jogos, eles são conhecidos como *game engines*. A *game engine* é o núcleo básico para a criação de um jogo e tem o objetivo de simplificar o processo de codificação além de fornecer uma série de funcionalidades agregadas, é utilizada tanto por desenvolvedores iniciantes como por profissionais da área. O objetivo do presente trabalho é desenvolver uma *game engine* que permita a criação rápida e fácil de jogos e aplicativos tridimensionais. A metodologia aplicada no desenvolvimento deste trabalho foi a divisão da *game engine* em diversas *sub-engines*, como por exemplo, a *engine* gráfica, *engine* física, *script engine*, entre outras.

Palavras-Chave: Jogos, Game Engine, Computação Gráfica, OpenGL.

Abstract

The electronic games move the billionaire world of game production companies, and have caused a revolution on the entertainment forms, surpassing even the movie industry. However the game developing is not a so simple task, it could take years, involve hundreds of people and cost millions of dollars. There are specific softwares that turn easier and speed up the creation process; these softwares are known as game engines. The game engine is the basic centre of the game development, and it has the objective of simplify the encoding process and besides it add a series of functionalities. A game engine is used by both beginners and professional game developers. The objective of this work is to develop a game engine that allows a fast and easy creation of games and tridimensional applications. The applied methodology on the development of this work was the division of the game engine in many sub-engines, for example, the graphic engine, the physical engine, script engine, and other sub-engines.

Word-Key: Games, Game Engine, Computer Graphics, OpenGL

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos o mercado de jogos eletrônicos vem se expandindo notoriamente, com isso o desenvolvimento desta forma de entretenimento também tem se tornando cada vez mais popular, seja pelo fato de ser um negócio lucrativo ou pelo simples fato de inúmeros jovens sonharem em criar os seus próprios jogos por diversão.

Porém a criação de um jogo não é uma tarefa simples, pois envolve diversas áreas da computação, como inteligência artificial, computação gráfica, banco de dados, redes de computadores, entre outras. Para facilitar este desenvolvimento existem *softwares* conhecidos como *game engines,* o seu principal objetivo é dar ao usuário uma maior abstração da programação envolvida no processo de criação de jogos.

As *game engines* são tão importantes que estão em praticamente todos os jogos eletrônicos da atualidade, seja em jogos de computador, vídeo games ou jogos para celular.

Atualmente existem no mercado inúmeras *game engines*, algumas delas gratuitas, já outras podem custar mais de 700 mil dólares. Muitas das empresas desenvolvedoras de jogos possuem as suas próprias *engines*, já outras utilizam produtos adquiridos de empresas especializas na criação deste tipo de *software*.

O objetivo deste trabalho é desenvolver uma *game engine* completa, onde todos os módulos necessários para o desenvolvimento de jogos estejam reunidos em uma única ferramenta, e que está possibilite a criação de jogos 3D de maneira rápida e simplificada.

O capitulo 2 apresenta uma introdução sobre jogos eletrônicos e também a analise e comparação do mercado mundial de jogos com o mercado de jogos no Brasil.

No capitulo 3 são abordados os conceitos de *game engine* e *sub-engines*, bem como o seu detalhamento na estruturação de uma ferramenta completa. Também são abordadas algumas técnicas utilizadas no desenvolvimento deste tipo de *software* e as principais *game engines* existentes atualmente.

No capitulo 4 é apresentada a *game engine* proposta neste trabalho, descrevendo sua arquitetura, requisitos e principais bibliotecas e *sub-engines*

utilizadas. Também são apresentadas algumas aplicações desenvolvidas com este trabalho para demonstrar as suas capacidades.

O capitulo 5 apresenta as conclusões deste trabalho bem como algumas sugestões de trabalhos futuros.

2 JOGOS ELETRÔNICOS

Segundo PESSOA (2001) *apud* Battaiola “Um jogo de computador pode ser definido como um sistema composto de três partes básicas: enredo, motor e interface interativa. O sucesso de um jogo esta associado à combinação perfeita destes componentes.”. O enredo trata dos temas, tramas e objetivos. A interface interativa é o meio de comunicação entre o jogador e o jogo, e também o meio pelo qual o jogo mostra o seu estado atual. O motor ou *engine* é o ponto central, é ele que controla o jogo internamente, sendo responsável pela renderização, controle de eventos, execução de ações, entre outras funções.

Os jogos digitais vêm apresentando cada vez um maior grau de evolução tecnológica, empresas das áreas de computação e entretenimento estão investindo no desenvolvimento de técnicas computacionais sofisticadas que podem ser empregadas em múltiplos programas interativos, principalmente naqueles que envolvam interfaces gráficas complexas, o que significa o uso conjunto de várias mídias, incluindo animações com gráficos 2D e 3D, vídeos, som 3D, etc., bem como ambientes multiusuário baseados em Internet (SOFTEX, 2005).

2.2 TIPOS DE MÍDIA

Segundo SOFTEX (2005), existem atualmente 5 tipos de mídias para quais os jogos eletrônicos são direcionados, estas mídias são:

- **Console** - Equipamento digital dotado de um conector para o aparelho de TV e um encaixe para algum tipo de unidade de armazenamento contendo programas e/ou dados de um determinado jogo. Os consoles mais avançados disponíveis atualmente são o PlayStation 3, da Sony; o Xbox360, da Microsoft e o Nintendo Wii, da Nintendo.
- **PC** - Tipo de jogo destinado a usuários de computador. Podendo ser usado localmente ou pela Internet.
- **Celular** - Jogos para telefones celulares.

- **Consoles Portáteis** - Estas plataformas possuem telas de maior tamanho e melhor qualidade que celulares, possibilitando o acesso a jogos de melhor resolução gráfica. Atualmente os principais consoles portáteis são o Game Boy, da Nintendo e o PlayStation portátil (PSP), da Sony.
- **TV Digital** - Jogos que podem ser rodados em aparelhos de TV digital como forma de conteúdo interativo.

Outra possibilidade é que o jogo seja destinado a mais de uma plataforma. Assim, pode atingir diferentes nichos tecnológicos, tais como computadores pessoais, celulares, portáteis, Internet, videogames ou qualquer outro dispositivo.

2.3 TIPOS DE JOGOS

Existem diversos tipos jogos, sendo que muitos deles constituem-se de junções de dois ou mais gêneros, além disto, também é comum que um jogo seja constituído de mais de um gênero.

Existem diversas classificações para os gêneros de jogos, a mais completa é apresentada por BAKIE (2005) que divide os gêneros em: aventura, ação, plataforma, luta, tiro em primeira pessoa (FPS), estratégia em tempo real (RTS), estratégia por turnos, *role playing game* (RPG), *massive multiplayer on-line role playing game* (MMORPG), horror, simulação, corrida, esporte, rítmico, *puzzle*, mini-game, tradicional, educacional, e *advergames*. No anexo A é apresentado uma tabela detalhando os gêneros propostos por BAKIE (2005).

Além da classificação por gênero os jogos também podem ser divididos em dois grupos baseados na sua forma gráfica de renderização, podendo ser 2D (duas dimensões) ou 3D (três dimensões). O termo três dimensões refere-se a objetos que podem ser descritos ou exibidos usando-se medidas de altura, largura e profundidade. Já objetos com duas dimensões possui apenas altura e largura, um exemplo de objeto 2D é uma folha em uma mesa em que nela não há nenhuma percepção de profundidade. A figura 1 mostra um comparação entre um objeto 2D e um objeto 3D (WRIGHT; LIPCHAK; HAEMEL, 2007).

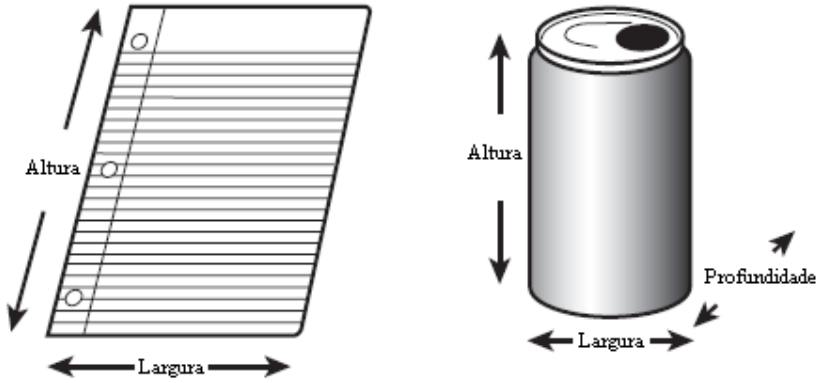


Figura 1 - Comparação entre objetos 2D e 3D
Fonte: (WRIGHT; LIPCHAK; HAEMEL, 2007).

2.4 O MERCADO DE JOGOS MUNDIAL

Segundo ASSIS (2003), o mercado mundial de jogos eletrônicos movimentou cerca de US$ 22,3 bilhões em 2003, com a comercialização de jogos, se forem incluídos também os gastos com *hardware* e acessórios este valor chega a US$ 55,5 bilhões, superando até mesmo a indústria cinematográfica que no mesmo ano obteve um lucro de US$ 19 bilhões.

A indústria de jogos eletrônicos é uma das que mais se expande no mundo, tendo crescido 12,5% em 2002 e 6,2% em 2003. A previsão era de que chega-se a uma taxa anual de crescimento de 20,1% no período 2004-2008 e atingi-se a marca de US$ 55,6 bilhões em 2008 (SOFTEX, 2005). Não existe nenhuma pesquisa recente que comprove se está meta foi atingida ou não, ou se até mesmo foi superada.

Segundo SOFTEX (2005), a receita no mercado de jogos está segmentado basicamente entre quatro categorias, jogos para consoles (73% do gasto mundial (US$ 16,2 bilhões)), PCs (17%), dispositivos *wireless* (3%) e jogos *online* (7%). O gráfico 1 mostra esta divisão.

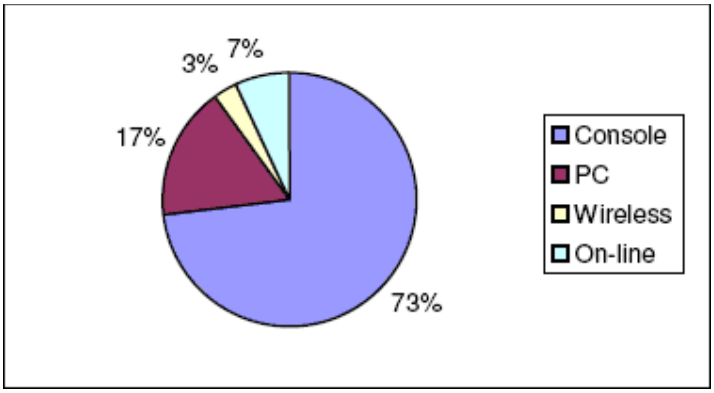


Gráfico 1 - Mercado Mundial de Jogos por Segmento
Fonte: SOFTEX (2005)

Nota-se que o mercado está concentrado principalmente em jogos que utilizam consoles como plataforma, SOFTEX (2005) aponta que a grande diferença do segmento para console em relação à plataforma PC está na maior capacidade gráfica dos consoles, além de que os PCs são mais vulneráveis à pirataria, fato que desestimula os desenvolvedores.

O mercado é complementado por dois segmentos emergentes com altas taxas de crescimento: os *games online* e os dispositivos sem fio ou *wireless* (celulares e consoles portáteis). Embora estes segmentos representem somente 10% do mercado (US$ 2,2 bilhões), a previsão era que em 2008 atingisse US$ 36,5 bilhões (66% do mercado esperado), superando a fatia dos jogos para console. Em 2003, a indústria de *games online* cresceu 69,5% e a de *wireless* 106,7% (SOFTEX, 2005).

2.5 O MERCADO DE JOGOS NO BRASIL

No Brasil esta área de entretenimento digital ainda é pouco explorada, mas já existem algumas iniciativas neste segmento, como mostra a pesquisa realizada em 2005 pela Abragames (Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos), segundo ela, há no Brasil o registro de 55 empresas desenvolvedoras de jogos em atividade, 33% delas no Paraná, 30% em São Paulo e 12% no Rio de Janeiro. De acordo com Marcelo Carvalho, na época presidente da Abragames e

sócio-diretor da *Devworks Game Technology*, a concentração de empresas no Paraná deve-se a criação da Gamenet-PR, rede de empresas local e ao investimento do governo do Estado. Segundo a pesquisa, o faturamento do mercado de jogos no Brasil fica em torno de 18 milhões de reais somente para as desenvolvedoras, se incluirmos também outras áreas relacionadas este valor chega 100 milhões. Um dos fatores que impedem o crescimento deste mercado no Brasil são os altos graus de pirataria, que chega a 94%, prejuízo calculado em 210 milhões de dólares (ABRAGAMES, 2005). O gráfico 2 representa a distribuição geográfica das empresas desenvolvedoras de jogos situadas no Brasil.

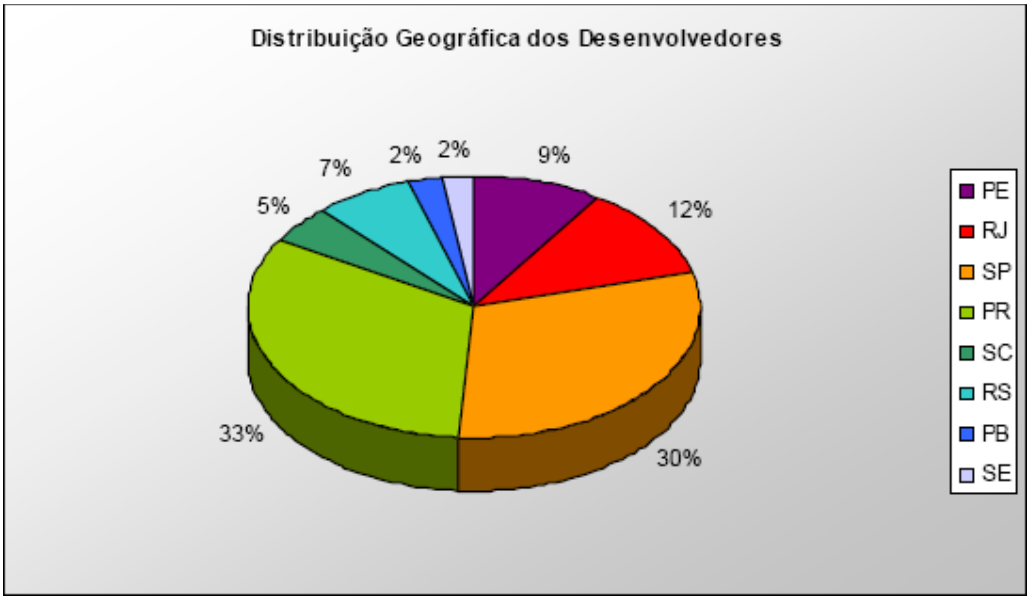


Gráfico 2 - Distribuição Geográfica dos Desenvolvedores de jogos no Brasil
Fonte: ABRAGAMES (2005)

Segundo KISHIMOTO (2006) *apud* Laramée, “A indústria de jogos possui um forte mercado em constante crescimento, rendendo bilhões de dólares por ano e com novas empresas surgindo pelo mundo todo. O Brasil está dentro desse contexto, sendo considerado um país com uma indústria emergente, embora já seja reconhecido internacionalmente pelas pesquisas tecnológicas que tem realizado ao longo dos anos”.

No Brasil as desenvolvedoras ficam focadas principalmente no desenvolvimento para a plataforma PC, diferente do mercado mundial onde os consoles são as plataformas dominantes. Segundo ABRAGAMES (2005), as

desenvolvedoras brasileiras focam no mercado de PC (63%), em segundo lugar, vem o de celulares (22%). Esta divisão é devido à dificuldade em se obter as licenças dos *kits* de desenvolvimento para os grandes consoles, o que visivelmente impede uma penetração das desenvolvedoras brasileiras nesse nicho de mercado. O gráfico 3 mostra a divisão das plataformas de desenvolvimento utilizadas no Brasil.

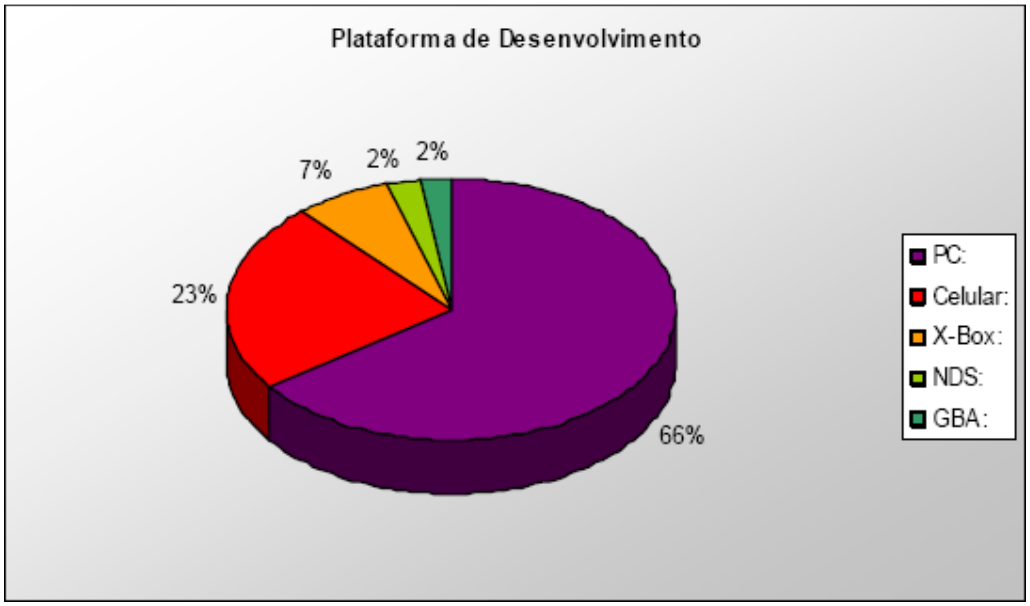


Gráfico 3 - Divisão das plataformas de desenvolvimento de jogos no Brasil
Fonte: ABRAGAMES (2005).

Segundo ABRAGAMES (2008), a indústria brasileira é hoje responsável por 0,16% do faturamento mundial com jogos eletrônicos. Na indústria de software (e não apenas de jogos), o que é desenvolvido no Brasil representa aproximadamente 1,8% da produção mundial, entretanto, é principalmente focado no mercado interno.

Comparado com o mercado mundial os lucros no Brasil nesta área ainda são pequenos, mas existem sinais que demonstram que o país ainda pode evoluir muito neste segmento, um deles é a existência de vários portais brasileiros na internet voltados especialmente para esta área, como é o caso da Unidev (www.unidev.com.br), que atualmente conta com 47281 usuários cadastrados, muitos destes, tendo como principal *hobbie* a criação de jogos ou alguma outra área relacionada (UNIDEV, 2008).

2.6 CONCLUSÃO

O desenvolvimento de jogos é um dos segmentos mais promissores do mercado internacional de *software* e o Brasil também está entrando para este segmento. Porém, criar jogos não é uma tarefa simples, pois envolve várias áreas da computação, como banco de dados, redes, computação gráfica, estrutura de dados, inteligência artificial (IA), entre outras. Mas existem *softwares* que tem como objetivo facilitar este desenvolvimento, estas ferramentas são chamadas de *Game Engines* o seu principal objetivo é dar ao usuário uma maior abstração da programação envolvida no processo de desenvolvimento de um jogo.

3 GAME ENGINE

O termo *Game Engine* surgiu durante a década de 90, juntamente com os primeiros jogos 3D (Doom, Quake) e a necessidade de se criar jogos mais complexos, além disto, surgiram equipes de desenvolvimento, e para manter um certo padrão de programação entre os desenvolvedores, foram criadas as primeiras *Engines* (WIKIPEDIA, 2008). Atualmente existem no mercado inúmeras *engines*, algumas delas grátis, já outras podem chegar a custar mais de U$ 700.000.

Segundo MIGUEL, MONTEIRO e CAVALHIERI (2006) uma *game engine* é o núcleo básico para a criação de um jogo e tem o objetivo de simplificar o processo de codificação, além de fornecer uma série de funcionalidades agregadas.

Desde seu surgimento as *game engines* se tornaram muito populares e usadas por programadores de jogos no mundo inteiro, porém muitos destes nem sempre entendem completamente o que é uma *game engine*. Uma maneira simplificada de entender é pensar nela como o motor de um carro, quando o motorista coloca e gira a chave na ignição, o motor é ligado e quando o motorista pisa no acelerador o carro começa a se mover. Internamente o motor do carro gera energia cinética e transfere esta energia para o eixo das rodas e faz o carro movimentar-se, o motorista não necessita saber exatamente o que está acontecendo dentro do motor, para ele o importante é que o carro está andando. O mesmo conceito é aplicado em uma *Game Engine*, quando o programador executa a função Rotacionar_Objeto() internamente a *Game Engine* irá fazer todos os cálculos necessários e executar a rotação, e assim como o motorista do carro o programador não necessariamente precisa saber como o objeto é rotacionado internamente, tudo o que ele precisa saber é chamar a função Rotacionar_Objeto (ZERBST; DÜVEL, 2004).

3.1 ARQUITETURA DE UMA GAME ENGINE

A arquitetura de uma *game engine* corresponde exatamente à definição dos seus módulos e a maneira como é realizada a interação entre eles. Tais módulos

são responsáveis por identificar e gerenciar todos os eventos que ocorrem ao longo do jogo, desde eventos gerados pelo usuário até aqueles causados por interação entre os objetos do jogo. Por isso, existem normalmente módulos para manipulação de objetos, renderização, sonorização, IA, rede e outros (PESSOA; RAMALHO; BATTAIOLA, 2002).

Segundo BATTAIOLA (2001) a arquitetura de uma *game engine* é composta pelos seguintes componentes:

- **Gerenciador de Entrada:** Módulo responsável por identificar eventos em dispositivos de entrada, por exemplo, o teclado, e encaminhar o ocorrido para o módulo principal que irá processar o dado e realizar a ação correspondente;
- **Gerenciador Gráfico:** Realizar todo o processamento necessário para transformar a cena criada pelos objetos do jogo em dados que sejam suportados pelas rotinas do sistema para desenho na tela;
- **Gerenciador de Som:** Responsável pela execução e processamento de todo o áudio do jogo;
- **Gerenciador de Inteligência Artificial:** Gerencia o comportamento de objetos controlados pelo computador. Para isso, realiza certas ações de acordo com o estado atual do jogo e algumas regras.
- **Gerenciador de Múltiplos Jogadores:** Permitir que jogadores do mesmo jogo se comuniquem entre si, gerencia a conexão e a troca de informações entre os diversos computadores que estão conectados via Internet ou Intranet;
- **Gerenciador de Objetos:** Módulo responsável por gerenciar os grupos de objetos do jogo, carregando os mesmos e controlando o seu ciclo de vida;
- **Objeto do Jogo:** Representa uma entidade que faz parte do jogo e as informações necessárias para que ela seja gerenciada, como por exemplo, posição, ângulo e dimensão, além disto, também é responsável pelo controle de colisões;
- **Gerenciador do Mundo:** Responsável por armazenar o estado atual do jogo, conta com os diversos gerenciadores de objetos para realizar tal tarefa;
- **Editor de Cenários:** Ferramenta para a criação de mundos que posteriormente serão carregados pelo gerenciador de mundos;

- **Gerenciador Principal:** Parte principal do jogo, servindo como ligação para a troca de informações entre os outros módulos do jogo.

A figura 2 mostra, segundo BATTAIOLA (2001), como os módulos de uma *game engine* estão interligados.

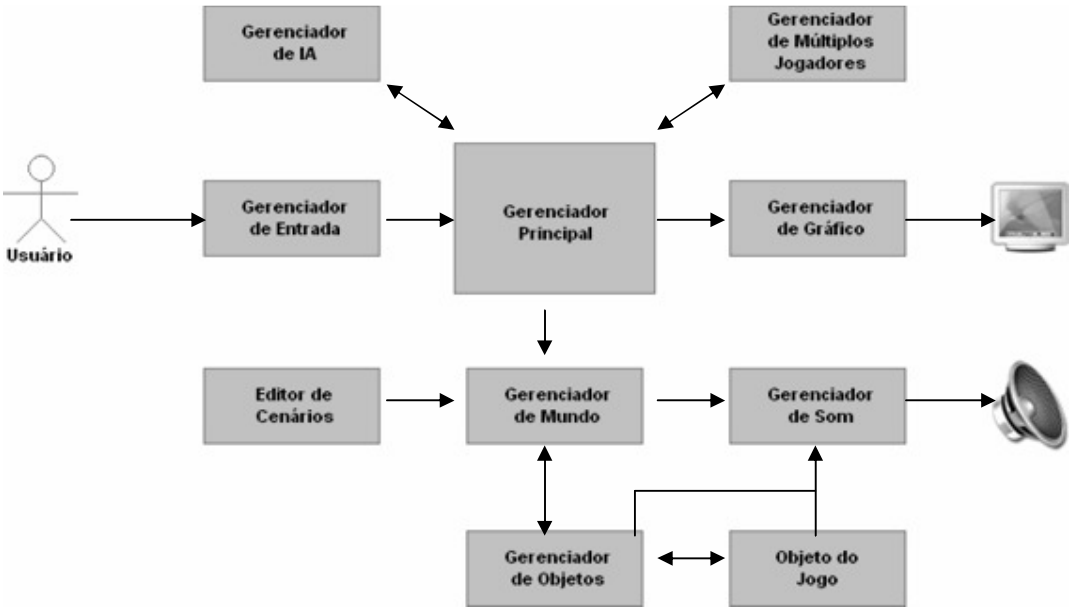


Figura 2 - Módulos de uma Game Engine
Fonte: (Battaiola, et al. ,2001, adaptado)

Segundo PESSOA (2001), uma *game engine* completa deve atender aos seguintes requisitos:

- Arquitetura modular, para que possa ser utilizada em cada novo jogo;
- Bom nível de abstração, ao prover objetos com funcionalidades já implementadas;
- Definição de objetos básicos, como objetos do jogo e o mapa (cenário) do jogo.
- Detecção e gerenciamento de eventos gerados por teclado, *mouse*, *joysticks*, etc;
- Algoritmos para desenho dos objetos do jogo, podendo ser 2D e 3D;
- Funcionalidades úteis a jogos 3D, como iluminação e transformações geométricas;

- Execução dos sons do jogo em resposta a eventos ocorridos;
- Implementação de algoritmos de IA para os objetos inteligentes dos jogos;
- Implementação de algoritmos para troca remota de dados, gerenciamento de sessões, sincronização das informações compartilhadas do jogo, etc;
- Implementação de algoritmos para modelagem física de objetos.

3.2 SUB-ENGINES

Uma *Game Engine* pode ser dividida em diversas *sub-engines*, como por exemplo, *engine* gráfica, *engine* física, *engine* sonora, *engine* de inteligência artificial, etc. (MIGUEL; MONTEIRO; CAVALHIERI, 2006). A figura 3 ilustra os diversos componentes que formam uma *game engine* e a figura 4 mostra as possíveis *sub-engines* que poderiam ser utilizadas para estruturar estes componentes.

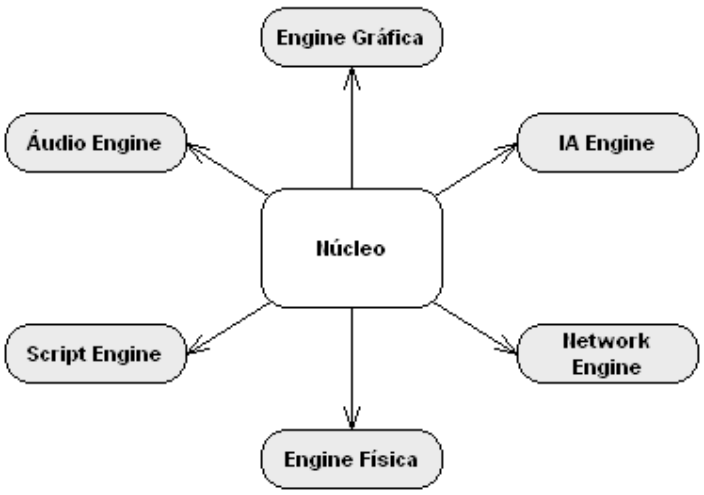


Figura 3 - Componentes de uma Game Engine
Fonte: (MIGUEL; MONTEIRO; CAVALHIERI, 2006, adaptado)

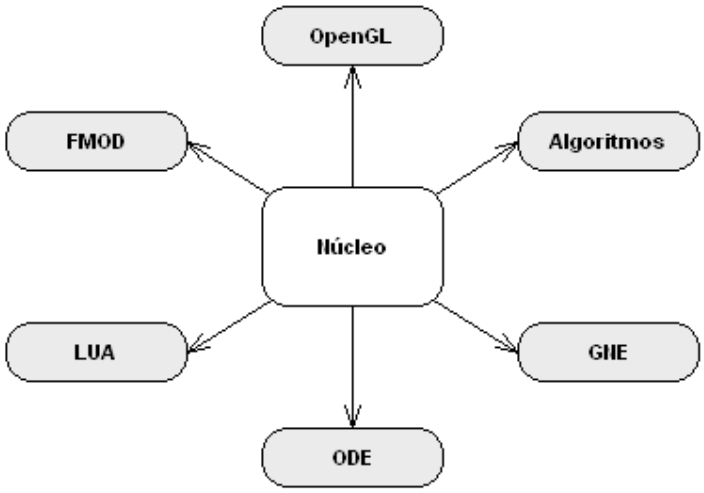


Figura 4 - Principais Sub-Engines Utilizadas
Fonte: (MIGUEL; MONTEIRO; CAVALHIERI, 2006, adaptado)

3.2.1 Engine Gráfica

A *engine* gráfica é um dos principais e mais importantes componentes de uma *game engine*, pois é através dela que todas as cenas e objeto serão renderizados e uma das principais características de um jogo é a sua qualidade gráfica (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

O OpenGL e o DirectX são as principais *engines* gráficas utilizadas. Estas duas APIs (*Application Program Interface*) provem uma interface entre o aplicativo e o *hardware* de processamento gráfico. Elas usam conceitos diferentes para cumprir as suas funções de APIs, o OpenGL utiliza o conceito de programação estruturada, já o DirectX é completamente orientado a objeto. (ZERBST; DÜVEL, 2004).

3.2.1.1 OpenGL

Segundo AZEVEDO e CONCI (2003), o OpenGL (*Open Graphical Library*) pode ser definido como uma interface de *software* (API) para aceleração da

programação de dispositivos gráficos, com aproximadamente 120 comandos para especificações de objetos e operações necessárias para a produção de aplicações gráficas interativas 3D. Pode-se classificá-la como uma biblioteca de rotinas gráficas para modelagem 2D ou 3D, extremamente portável e rápida possibilitando a criação de gráficos 3D com excelente qualidade visual e rapidez, uma vez que usa algoritmos bem desenvolvidos e otimizados pela Silicon Graphics.

A biblioteca OpenGL vai além do desenho de primitivas gráficas, tais como linhas e polígonos, dando suporte a iluminação, sombreamento, mapeamento de texturas e transparência. Além disto, a biblioteca OpenGL executa transformações de translação, escala e rotação, através da multiplicação de matrizes com transformações cumulativas, ou seja umas sobre as outras (AZEVEDO; CONCI, 2003).

O OpenGL foi planejado para o uso direto com o hardware projetado e aperfeiçoado para a exibição e manipulação de gráficos 3D. Porém implementações baseadas em software também são possíveis, como por exemplo, a biblioteca Mesa3D (www.mesa3d.org), que permite o uso de aplicativos OpenGL em computadores que não possuem placas aceleradoras gráficas. A Apple também desenvolveu uma destas implementações que está disponível para o OS X. As implementações baseadas em software podem não ser tão rápidas e com menos recursos avançados que só são possíveis com o uso de placas de vídeo com aceleração gráfica, mas permite que OpenGL possa ser executado em uma grande variedade de sistemas operacionais, mesmo que estes não possuam dispositivos de aceleração gráfica (WRIGHT; LIPCHAK; HAEMEL, 2007).

Segundo AZEVEDO e CONCI (2003), as principais características do OpenGL são:

- **Aceleração de hardware**: Aceleração do processamento geométrico, luzes, transformações e render.
- **Efeitos 3D em tempo real**: *Real-time fog, anti-aliasing*, volume *shadows*, *bump mapping, motion blur*, transparências, reflexões, texturas 3D, volume *rendering* e outras.
- **Suporte a inovações futuras de *software* e *hardware***: Um mecanismo de extensão permite a configuração para novos *hardwares* e *softwares*.

- **Estabilidade**: Estações avançadas de supercomputadores vêm utilizando essas bibliotecas desde 1992.
- **Escalável**: Suas aplicações API podem ser executadas de pequenos aparelhos eletrônicos até supercomputadores com a utilização máxima dos recursos disponíveis.

3.2.1.2 DirectX

O DirectX é uma API para o sistema operacional Microsoft Windows que foi desenvolvida para prover uma interface para o controle eficiente de *hardware* multimídia e permitir aos programadores trabalharem com comandos e estruturas de dados de alto nível. Ele é construído sobre uma camada abstrata de *hardware*, chamada de HAL (*Hardware Abstraction Layer*) que oculta às dependências especificas de determinados dispositivos, o que torna os programas desenvolvidos independentes de dispositivo, ou seja, funcionarão em qualquer *hardware*, desde que sobre a plataforma Windows (KOVACH, 2000).

Segundo JONES (2004), o DirectX é formado por diversos componentes, cada um representando um diferente aspecto do sistema, cada um destes pode ser utilizado independentemente, permitindo ao desenvolvedor utilizar apenas os necessários para a sua aplicação, os componentes são os seguintes:

- **DirectX Graphics:** Componente que manipula todas as saídas gráficas, provê funções para a manipulação de objeto gráficos 2D e 3D.
- **DirectInput:** Toda a interação entre o usuário e o aplicativo é feita através deste componente. Inclui suporte para diversos dispositivos de entrada, como teclado, *mouse*, *gamepad* e *joysticks*.
- **DirectPlay:** Componente que dá suporte a criação de aplicativos para comunicação em rede, permitindo a criação de jogos *multiplayer*. O DirectPlay fornece uma interface de alto nível, não sendo necessário a implementação de todos os aspectos da comunicação.

- **DirectSound:** Componente para a manipulação de efeitos sonoros, permite a execução simultânea de um ou mais arquivos no formato WAV, provendo o controle total sobre a mídia em execução.
- **DirectMusic:** Permite a criação de trilhas sonoras dinâmicas, baseadas em determinadas regras determinadas pelo desenvolvedor.
- **DirectShow:** Provê acesso a execução de vídeos e arquivos de áudio, o principais formatos suportados são: AVI, MPEG, MP3, ASF, entre outros.
- **DirectSetup:** Componente para a distribuição dos aplicativos desenvolvidos, permitindo que antes da execução seja verificado se o sistema possui o DirectX instalado e se a versão é compatível, caso não seja a nova versão é instalada.

Nas versões antigas do DirectX o componente DirectX Graphics era dividido em dois componentes, o Direct3D e o DirectDraw, separando as funcionalidades 2D das 3D. A versão 7 foi a ultima a separar estes componentes.

A arquitetura do DirectX é baseada em duas camadas, a camada da API e a camada de hardware abstrato. A camada de API se comunica com hardware da máquina através da camada HAL, ela fornece uma interface padronizada para o DirectX, e também comunica-se diretamente com o *hardware* através de um driver especifico. O desenvolvedor nunca interage diretamente com esta camada, mas a acessa indiretamente através das funções fornecidas pelo DirectX (JONES, 2004). A figura 5 mostra como são estruturadas as camadas de um aplicativo utilizando o DirectX.

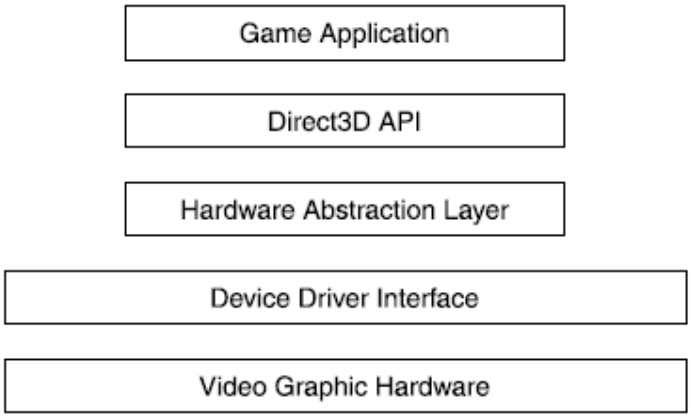


Figura 5 - Camadas de uma aplicativo DirectX
Fonte: (JONES, 2004).

As versões anteriores do DirectX possuíam mais uma camada conhecida como HEL (*Hardware Emulation Layer*). Esta camada simulava algumas funcionalidades não encontradas em alguns dispositivos.

3.2.1.3 Outras engines gráficas

Além do OpenGL e do DirectX também existem outras *engines* gráficas que são derivadas destas duas, com por exemplo, o OGRE3D (*Open-source Graphics Rendering Engine*) que foi escrita em C++ com o objetivo de tornar fácil e intuitivo o desenvolvimento de aplicativos 3D, a biblioteca abstrai os detalhes complexos da utilização do DirectX e do OpenGL provendo uma interface baseada em objetos e classes de alto nível (DEVMASTER, 2008).

3.2.2 Engine Física

Uma *engine* física consiste em um programa ou parte de um programa que simula o modelo físico de Newton, usando variáveis como massa, velocidade, fricção e resistência ao vento. A *engine* física possui dois componentes principais, o sistema de detecção de colisões que é responsável por detectar qualquer tipo de colisão entre objetos. O outro é o sistema de simulação, que é responsável por aplicar e resolver qualquer tipo de força que interaja com os objetos. É possível simular e prever efeitos sobre diferentes condições que se aproximam muito ao que ocorre no mundo real ou em um mundo fantasiado. A sua principal aplicação está na simulação cientifica e em jogos (MILLINGTON, 2007).

Existem dois tipos de *engines* físicas, as *engines* de tempo real e as de grande precisão. *Engines* de grande precisão requerem maior poder de processamento para os cálculos extremamente precisos, são mais utilizadas em simulações por cientistas e na animação cinematográfica. Um dos principais propósitos do computador ENIAC foi utilizá-lo como uma simples *engine* física. Ele foi usado para criar tabelas de balística para ajudar as forças militares dos Estados

Unidos a estimar a trajetória dos projéteis utilizados pela sua artilharia quando disparados sobre diversos ângulos e com diferentes armas e sofrendo a ação do vento. Em jogos é mais comum a utilização de *engines* de tempo real, que simplificam os cálculos, e tem uma menor precisão, porém fornecem uma melhor performance (WIKIPEDIA, 2008).

Na maioria dos jogos de computador a velocidade de uma simulação é mais importante do que a precisão desta simulação. Tipicamente a maioria dos objetos 3D em um jogo são representados por duas malhas ou formas separadas, umas destas malhas é o objeto, composto de diversas faces e vértices, extremamente complexo e detalhado, este objeto é o que o jogador vê. Entretanto, com o propósito de melhorar a performance do jogo, o outro objeto é uma forma extremamente simplificada, com poucos vértices e faces, este objeto não é visível ao jogador e é utilizado para representar o outro objeto para a *engine* física. Por exemplo um vaso com uma planta em um ambiente 3D, a planta é um objeto complexo, com diversas curvas, diversas faces e vértices, detectar uma colisão com este objeto pode custar um certo tempo de processamento, para diminuir este tempo pode se utilizar um cilindro, dentro deste ficará o vaso com a planta, para o jogador o cilindro não será visível, mas será utilizado pela *engine* física para determinar uma possível colisão de outros objetos com o vaso. Porém esta técnica pode gerar certa limitação, pois, por exemplo, a simular um tiro no vaso não seria realista, isto porque a *engine* física não conhece as faces do vaso e sim as do cilindro que o envolve (MILLINGTON, 2007).

Uma das principais *engines* físicas é o ODE (*Open Dynamics Engine*). Desenvolvido em 2001 por Russel Smith, a *engine* trabalha com as leis de física clássica atuando sobre corpos rígidos, articulados ou não. A *engine* foi desenvolvida em C/C++ e também é bastante utilizada fora da área de jogos, em aplicativos e simulações onde é necessário um ambiente virtual que simule as principais características do ambiente real (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

3.2.3 Script Engine

Scripts são linguagens de programação desenvolvidas para um aplicativo específico, seu código fonte é interpretado pelo próprio programa em tempo de

execução e não em tempo de compilação. Em jogos os scripts são utilizados principalmente para a implementação da lógica do jogo.

Dentre as diversas linguagens de script disponíveis, destacam-se o Lua, Python e Ruby, sendo que a Lua é uma das principais usadas na implementação de jogos. Esta foi desenvolvida em 1994 pela PUC-Rio, em um projeto do Tecgraf (Grupo de Computação Gráfica da PUC-Rio) com a Petrobrás, onde o objetivo era a visualização de perfis geológicos e simulação de plataformas marítimas de extração de petróleo. A linguagem foi usada em jogos pela primeira vez em 1997 após a publicação de um trabalho relatando o projeto brasileiro, que logo em seguida chamou a atenção de desenvolvedores do mundo inteiro (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

3.2.4 Áudio Engine

A questão do áudio em um jogo é tão importante quanto sua capacidade gráfica, diferentes gêneros exigem um tratamento específico do áudio. Uma *áudio engine* é responsável por todo tratamento do áudio 2D ou 3D de um jogo.

O OpenAL (*Open áudio Library*) é um exemplo de *áudio engine*, ela consiste em uma API exclusiva para o tratamento do áudio. A Creative Labs, principal fornecedora de *hardware* para áudio em PCs, é atualmente a coordenadora do projeto OpenAL (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

A biblioteca OpenAL trata uma coleção de fontes de áudio que podem se mover por um espaço 3D e que são percebidas por um único receptor no mesmo espaço, no caso o jogador. Os objetos básicos do OpenAL são um Ouvinte (*listener*), uma Fonte (*source*) e um Buffer (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

Outro exemplo de *áudio engine* é o FMOD desenvolvido pela Firelight Technologies, e pode ser utilizada para o tratamento de áudio 2D ou 3D, o seu funcionamento também é baseado em fontes e ouvintes.

3.2.5 IA Engine

As IA Engines (engines de inteligência artificial) servem para o desenvolvimento de comportamento autônomo em NPCs (*Non-Player Characters*), ou seja, personagens que não são controlados pelo jogador (MIGUEL; MONTEIRO; CAVALHIERI, 2006).

Existem diversas definições para o termo inteligência artificial (IA), segundo MIFFLIN (2006) a inteligência artificial consiste na habilidade do computador ou outra máquina de executar atividades que normalmente necessitam de inteligência. Outras fontes definem a inteligência artificial como o processo ou ciência da criação de máquinas inteligentes.

A maioria dos jogos eletrônicos incorpora algum tipo de inteligência artificial, desde o clássico *Pac Man* com os fantasmas que tentam capturar o jogador, até jogos 3D complexos como a série *Unreal Tournament* (BOURG; SEEMAN, 2004).

As técnicas de inteligência artificial para jogos normalmente são divididas em duas categorias, determinísticas e não determinísticas. As determinísticas possuem um comportamento previsível, um exemplo é o algoritmo *Chasing*, nele o ator sempre irá seguir o seu alvo, independente de caminho ou distância, por isso se torna previsível. As técnicas não determinísticas são o oposto das determinísticas, tornando o comportamento imprevisível, por exemplo, um ator que aprende e se adapta as táticas do jogador, para criar este tipo de inteligência artificial pode ser utilizado algoritmos genéticos, redes neurais, ou a técnica Bayesianas. Os algoritmos determinísticos são rápidos, fáceis de implementar e testar, porém o desenvolvedor necessita programar todos as possíveis situações que podem ocorrer durante o jogo, estes algoritmos também diminuem consideravelmente o tempo de vida de um jogo, pois após jogar algumas vezes os comportamentos determinísticos se tornam previsíveis para o jogador (BOURG; SEEMAN, 2004).

Segundo MIGUEL, MONTEIRO e CAVALHIERI (2006) não existe nenhuma *engine* especializada na implementação de inteligência para um jogo, geralmente esse processo é feito por cada desenvolvedor.

3.2.6 Network Engine

As *network engines* ou *engines* de rede tem como objetivo prover funcionalidades para a criação jogos *multiplayer online*, fornecendo funções para conexões entre computadores.

Segundo ARMITAGE, CLAYPOOL e BRANCH (2006) existem quatro tipos básicos de arquiteturas para a criação de jogos *multiplayers, single screen/split, peer-to-peer, client–server* e *peer-to-peer client–server hybrid*.

A arquitetura *single screen/split* é a única que não utiliza uma rede física para o jogo *multiplayer*, todos os jogadores interagem em uma máquina, sendo cada um em seu turno ou com a divisão da tela (*split*). Este modelo é bastante utilizado em consoles.

Na arquitetura *peer-to-peer* (ponto a ponto), cada cliente (jogador) é considerado um ponto e nenhum deles tem mais controle sobre o jogo do que o outro. Não existe nenhum mediador entre os jogadores para controlar o estado do jogo ou gerenciar as mensagens entre os jogadores. Este modelo é comumente utilizado para jogos em redes locais (LAN – *Local Área Network*), pois esta permite o uso de mensagens em *broadcast* (ARMITAGE; CLAYPOOL; BRANCH, 2006). A figura 6 exemplifica esta arquitetura.

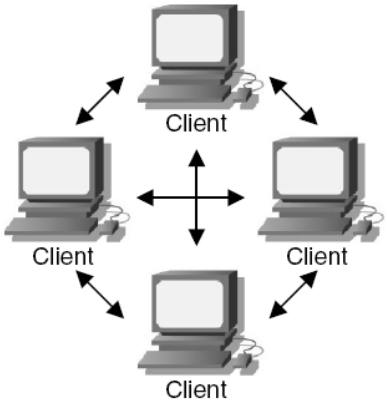


Figura 6 - Arquitetura *Peer-to-Peer*
Fonte: (ARMITAGE; CLAYPOOL; BRANCH, 2006).

Na arquitetura *client–server* (cliente-servidor) um servidor central é utilizado para gerenciar o jogo e toda comunicação entre os jogadores (clientes). Os clientes

não se comunicam diretamente, todas as mensagens são enviadas primeiramente para o servidor que trata de interpretá-las e distribuí-las para os outros clientes. Esta arquitetura é bastante popular em jogos que envolvem diversos jogadores espalhados pela internet (ARMITAGE; CLAYPOOL; BRANCH, 2006). A figura 7 exemplifica esta arquitetura.

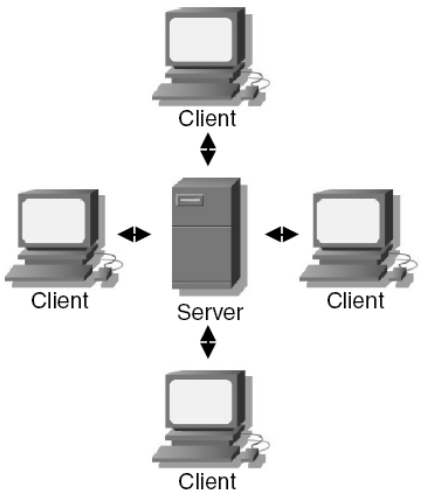


Figura 7 - Arquitetura *client–server*
Fonte: (ARMITAGE; CLAYPOOL; BRANCH, 2006).

A arquitetura *peer-to-peer client–server hybrid* é basicamente uma união da arquitetura *peer-to-peer* com a *client–server*, permitindo que o jogo seja gerenciado por um servidor central e também que os clientes se comuniquem diretamente, isso diminui o processamento do servidor pois somente o necessário é enviado para o servidor e o que for específico para um cliente é enviado diretamente para este (ARMITAGE; CLAYPOOL; BRANCH, 2006).

Um exemplo de *network engine* é a GNE (*Game Networking Engine*) que consiste de uma biblioteca multiplataforma desenvolvida em C++ com todas as funcionalidades necessárias para a criação de jogos *multiplayer* em rede (GILLIUS, 2008).

Assim como as IA *engines* é comum que o desenvolver de um jogo *multiplayer* desenvolva as suas próprias rotinas para o controle *multiplayer*, visto que cada jogo possui um padrão especifico para a comunicação.

3.3 TÉCNICAS E ALGORITMOS

Existem diversas técnicas e algoritmos utilizados para a criação e estruturação de uma *game engine*, a seguir serão detalhadas algumas destas técnicas, levando em consideração as utilizadas no desenvolvimento da *game engine* proposta neste projeto.

3.3.1 Transformações Geométricas

Transformações geométricas são operações que podem ser utilizadas visando a alteração de características como posição, orientação, forma ou tamanho de um determinado objeto a ser representado.

Segundo WANGENHEIM (2003) existem em computação gráfica 3 transformações geométricas básicas, que podem ser combinadas para se obter o comportamento de um objeto no mundo em que está modelado: translação, escalonamento e rotação. A translação consiste em mover o objeto para um determinado ponto (figura 8), a rotação é o ato de rotacionar o objeto sobre um eixo (figura 9) e o escalonamento consiste em aumentar ou diminuir o tamanho de um objeto, figura 10 (ASTLE; HAWKINS, 2004).

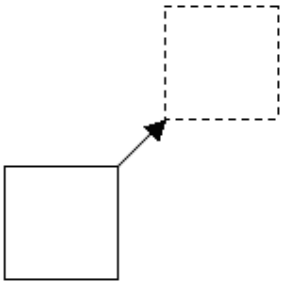


Figura 8 - Translação

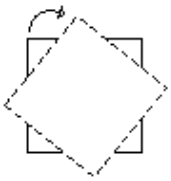


Figura 9 - Rotação

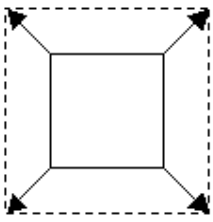


Figura 10 - Escalonamento

3.3.2 Fontes Emissoras de Luz

Para a criação de ambientes mais realistas é importante a utilização de fontes emissoras de luz, estas luzes são responsáveis por interagir com o material dos objetos, assim as tonalidades de cor do objeto variam conforme a posição da luz emitida sobre ele.

Basicamente existem 3 tipos de luzes:

- *Directional Light*: São raios de luz emitidos para uma determinada posição, estes são infinitos e paralelos como mostrado na figura 11.
- *Point Light*: É uma fonte emissora posicionada em um determinado ponto do ambiente e emite raios de luz para todas as direções como mostrado na figura 12.
- *Spot Light*: A fonte de luz fica posicionada no ambiente, mas emite raios de luz em forma de cone para uma terminada área como mostrado na figura 13.

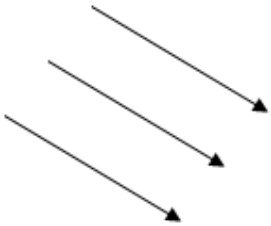


Figura 11 - Directional Light

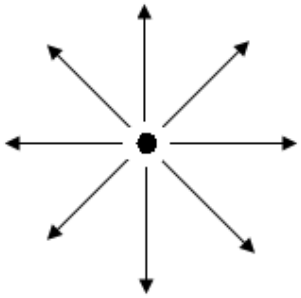


Figura 12 - Point Light

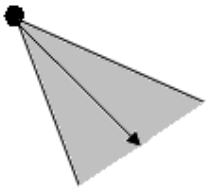


Figura 13 - Spot Light

3.3.3 Modelos Hierárquicos

O uso de modelos hierárquicos é uma técnica usada para representar estruturas articuladas, como robôs, animais, seres humanos, entre outros objetos que se adaptem ao modelo. Na estrutura é utilizada a filosofia de árvore, onde cada parte móvel ou articulada do objeto é considerado um nodo. A vantagem da utilização desta técnica é a facilidade em aplicar-se transformações nos objetos,

onde esta se reflete também sobre todos os objetos filhos ao nodo à qual esta foi aplicada (WANGENHEIM, 2003). Na figura 14 é possível observar um exemplo de modelo hierárquico, formando uma perna rudimentar, nela todos os nodos estão interligados, na figura 15 é aplicado sobre a esfera central uma rotação, o que faz com que todos os nodos seguintes se rotacionem, e os objetos anteriores não sofram nenhuma transformação, já na figura 16 é aplicada uma rotação no nodo principal do objeto hierárquico, consequentemente todos os outros objetos filhos sofrerão transformações equivalentes.

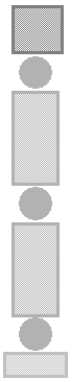


Figura 14 - Modelo Hierárquico 1
Fonte: (WANGENHEIM, 2003)

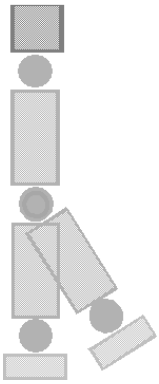


Figura 15 - Modelo Hierárquico 2
Fonte: (WANGENHEIM, 2003)

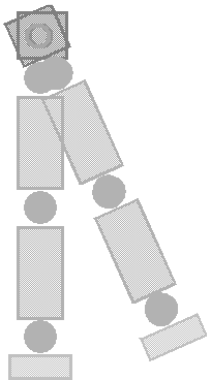


Figura 16 - Modelo Hierárquico 3
Fonte: (WANGENHEIM, 2003)

3.3.4 Heightmap

Um *heightmap* consiste em uma superfície 3D formada a partir de uma imagem 2D, esta imagem é constituída por tons de cinza, como pode ser visto na figura 17, a variação de cor é usada para a formação da superfície, onde tons claros representam os locais de maior elevação e os escuros os de menor elevação (ZERBST; DÜVEL, 2004). A figura 18 mostra uma superfície criada a partir da figura 17. Os *Heightmaps* normalmente são utilizados para a criação de terrenos, pois possibilitam a formação de ambientes externos, como montanhas, de maneira rápida e fácil.

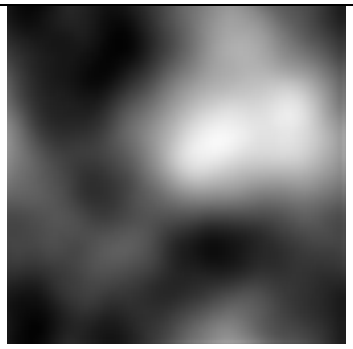


Figura 17 - Imagem 2D em tons de Cinza

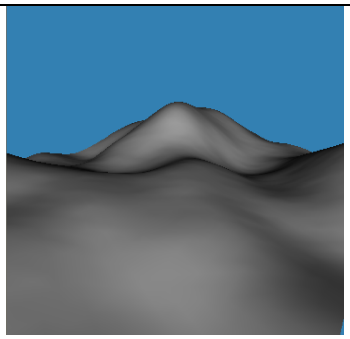


Figura 18 - Heightmap 3D

3.3.5 Projected Shadows

Projected Shadows ou sombras projetadas, consiste em projetar a sombra de um objeto sobre a textura de outro, a sombra é gerada a partir de um ponto emissor de luz do ambiente e é projetada nos objetos que estejam oclusos ao objeto que esteja emitindo a sombra, como mostrado na figura 19.

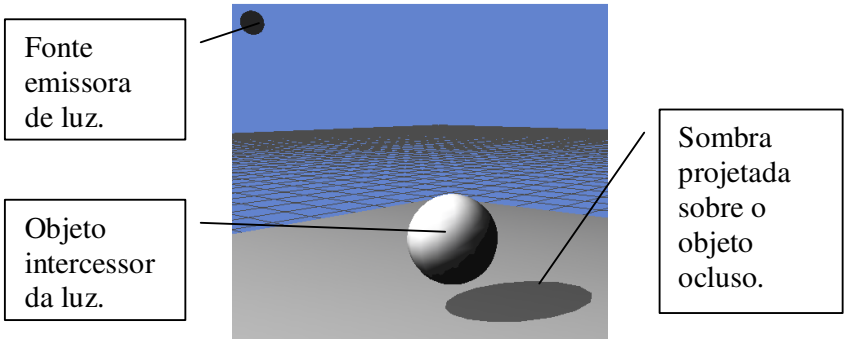


Figura 19 - Sombra projetada

3.3.6 Clipping e Culling

O *Clipping* e *Culling* são duas técnicas utilizadas para otimização do desempenho de aplicações gráficas.

Segundo EBERLY (2004), o *culling* consiste em uma técnica usada para determinar as partes visíveis dos objetos ao usuário, visto que não há a necessidade

de se redesenhar áreas não visíveis. Como por exemplo, quando o usuário está visualizando um dos lados de um cubo, os outros lados não estão visíveis, neste caso, com o *culling* as outras faces não seriam redesenhadas.

A outra técnica é o clipping, que consiste em determinar os objetos que estão na área de visão do usuário, os que estiverem completamente visíveis serão todos redesenhados, caso apenas uma parte de algum deles esteja visível somente esta será redesenhada e os que não estiverem no campo de visão não serão redesenhados, esta técnica também é usada para evitar que objetos que estejam muito distantes da visão do usuário precisem ser redesenhados (EBERLY, 2004).

O clipping é comum na maioria dos aplicativos gráficos, já o *culling* é mais usado em aplicações 3D.

3.3.6 Sistema a de Partículas

Para a criação de efeitos especiais, que são muito usados em jogos, é possível utilizar o sistema de partículas, este termo refere-se à técnica de simular determinados fenômenos, dificilmente produzido com a utilização de técnicas convencionais, como por exemplo, fogo, fumaça, neve, neblina, entre outros. O sistema consiste em primeiramente determinar um emissor, normalmente um objeto ou ponto no ambiente 3D, também o número máximo de partículas a serem geradas simultaneamente e alguns parâmetros para elas, como o vetor de velocidade inicial, o tempo de vida, sua cor, seu tamanho, entre outros. O próximo passo é iniciar a simulação do efeito, onde são criadas as partículas e sobre elas é aplicado o vetor de velocidade inicial para que entrem em movimento, então cada uma delas passa para um estado de *loop* até o fim do seu tempo de vida, onde são destruídas. Este processo é executado até o fim da simulação. Na figura 20 é possível observar um efeito de fogo criado na ferramenta e na figura 21 um efeito de neve utilizando o mesmo sistema.

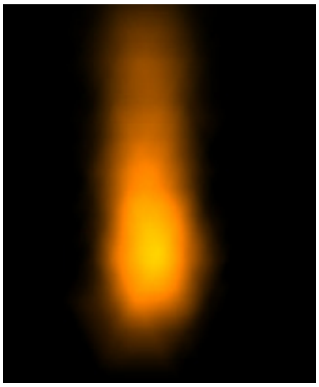


Figura 20 - Simulação de fogo

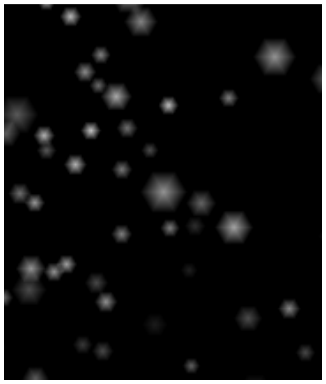


Figura 21 - Simulação de neve

3.3.7 Shaders

Segundo DALMAU (2003), em um ambiente 3D os objetos e cenários não são definidos apenas pela sua geometria, forma ou estrutura, o olho humano também é muito sensível a textura e a luz, pode-se citar como exemplo uma praia sem uma textura de areia, ou um mar ondulado onde a luz do sol não é refletida corretamente. Por isso para se obter ambientes mais realistas são utilizados *shaders*.

Shaders são pequenos programas que podem ser executados diretamente na GPU (Núcleo de processamento das placas de vídeo). Inicialmente, a maioria dos programas para GPUs eram escritos em linguagem *Assembly*, porém atualmente já existem linguagens de alto nível para esta programação. Entre elas, pode-se citar o CG, HLSL e GLSL. A base sintática e semântica destas linguagens é bem próxima a linguagem C (CODEPLEX, 2008).

O CG, desenvolvido através de uma parceria entre NVIDIA e Microsoft, é uma linguagem de *shaders* que permite ao desenvolvedor controlar através de programação a forma, aparência e movimentação dos objetos renderizados. O HLSL e o CG são basicamente a mesma linguagem, porém com nomes diferentes para distinguir entre a tecnologia base utilizada nas linguagens, enquanto HLSL é destinado para programas escritos em DirectX o CG pode atuar também em aplicações OpenGL. Já o GLSL, cujo nome oficial é OpenGL Shading Language, é uma extensão da linguagem OpenGL para criação de *shaders* (CODEPLEX, 2008).

As linguagens de programação de *shaders* apresentam recursos característicos para a implementação de algoritmos gráficos. Estes recursos visam

facilitar a codificação e um maior aproveitamento do paralelismo presente nas placas gráficas.

3.3.8 Animação

Animações são indispensáveis para a maioria dos jogos, são elas que dão "vida" aos objetos e personagens, tornando os movimentos destes o mais realista possível.

Os algoritmos de animação podem ser divididos em duas categorias: métodos explícitos e métodos implícitos. Métodos explícitos armazenam a seqüência de vértices animados da geometria de todos os *frames* da animação. Uma vez armazenado, pode ser usada uma variedade de técnicas para re-converter os dados originais, e gerar as animações. Como característica, métodos de animação explícitos são fáceis de codificar e entender, e freqüentemente envolve matemática muito simples. Por outro lado, necessitam um uso intensivo de memória para armazenar todos os vértices animados. Muitos quadros são necessários para criar o verdadeiro senso de movimento de um objeto, e isso significa armazenar muitos dados. Um arquivo MD3 (formato de animação explícito popular usado pelo jogo Quake 3) possui um tamanho de aproximadamente 10MB, e isso somente para um personagem (DALMAU, 2003). A Figura 22 mostra um exemplo de um *loop* de animação criado usando um método explícito.

Figura 22 - Animação explícita.
Fonte: (DALMAU, 2003)

A segunda categoria é a de animação implícita, estes métodos não armazenam os dados de animação, mas ao invés disto, eles armazenam uma descrição de alto nível do movimento. Por exemplo, sistemas de animação de esqueleto armazenam a configuração (em termos de seus ângulos de rotação) para

cada junta, como o cotovelo, joelho, e assim por diante. Então, em tempo real, esta descrição é traçada a uma malha de um personagem não animado, e a animação é computada. Esta computação normalmente envolve matemática complexa com trigonometria e matrizes. Assim, estes métodos produzem cálculos intensivos para a CPU (DALMAU, 2003). A figura 23 mostra um exemplo de animação implícita.

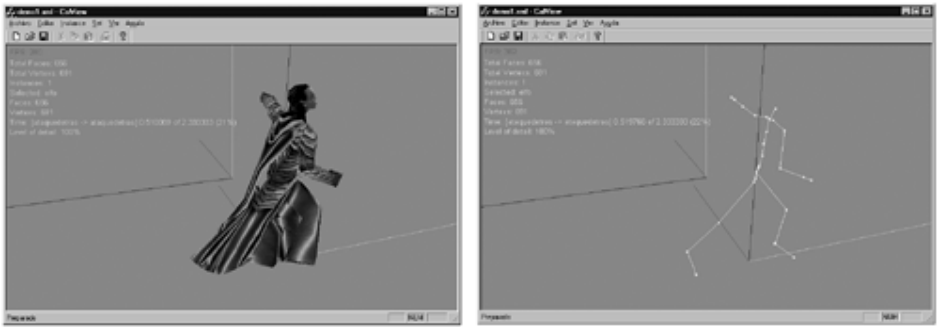


Figura 23 - Animação implícita.
Fonte: (DALMAU, 2003)

Segundo DALMAU (2003), métodos explícitos eram muito populares em jogos 3D mais antigos porque o *hardware* possuía pouco poder de CPU e estes jogos não usavam muitos ciclos de animação, assim a quantidade de memória usada era aceitável, mais recentemente, métodos explícitos também foram bastante usados em jogos que envolvem vários jogadores e personagens, pois usando métodos implícitos os cálculos envolvendo a animação para os diversos personagens seria inviável. Por outro lado, animação implícita está ficando mais popular, pois torna a movimentação mais detalhada, alem disso os jogos atuais tentam prover uma variedade larga de interações, o que em métodos explícitos causaria um uso desnecessário de memória.

- **Frame Animation** (Método explícito): O modo mais simples de animar um personagem é derivado de técnicas de animação tradicionais, onde cada pequeno movimento de um personagem é desenhado em uma pagina de um caderno, em seguida estas paginas são folheadas para apresentar a ilusão de movimento. Nas animações por *frame* o conceito é basicamente o mesmo, são gravadas diversas vezes a mesma malha com pequenas alterações para gerar os movimentos, e estas serão posteriormente exibidos em uma

determinada velocidade, esta velocidade é chamada de *frames* por segundo (normalmente 25 *frames* por segundo ou mais).

- **Keyframe Animation** (Método explícito): *Keyframe animation* é uma técnica que pode ajudar a reduzir o uso de memória enquanto mantêm a mesma simplicidade e elegância da animação por *frames* convencional. Para uma seqüência de 50 *frames*, que poderia envolver *keyframes* nos instantes 0, 15, 30, e 45. O personagem fará o *loop* pela seqüência usando os *keyframes* e interpolação para derivar todos os *frames* restantes. Isto torna o processo de animação mais rápido, pois o artista pode criar animações complexas usando poucos frames e não é necessário armazenar todos os frames em memória.
- **Forward Kinematics** (Método implícito): *Forward Kinematics* é o modo mais fácil de implementar animação implícita. Em sua concepção mais simples, ela é iniciada em algum nodo de base (normalmente a pélvis) e as transformações se propagam aos descendentes. Cada junta tem seu próprio sistema de coordenada local que é usada para propagar transformações de uma maneira hierárquica.

3.4 GAME ENGINES

Atualmente existem diversas *game engines* completas, algumas delas são grátis e open source, já outras são comercializadas. A seguir serão apresentadas algumas game destas *engines*.

3.4.1 3D Game Studio

O 3D Game Studio, desenvolvido pela Conitec, é uma ferramenta para a criação de jogos e aplicativos 3D/2D. Ela combina a linguagem de programação C-Script com uma interface para o desenvolvimento 3D, juntamente com a sua *engine* física, os editores de níveis, terrenos e modelos. Possui também uma grande

biblioteca de objetos 3D e scripts pré-desenvolvidos, permitindo a criação de jogos simples sem a necessidade de programar (DEVMASTER, 2008).

Algumas características:

- Utiliza a API gráfica DirectX.
- Áudio 3D, dando suporte a arquivos wav, mid, ogg, CD, mp3, avi, e mpg.
- A *engine* física suporta primitivas básicas (gravidade, massa, elasticidade, frisão, etc.) e detecção de colisões.
- Fontes emissoras de luzes estáticas, dinâmicas e direcionais.
- Permite o uso de sombras projetadas e volume para sobras.
- *Keyframe Animation, Skeletal Animation, Morphing* e *Animation Blending.*
- *Mesh Loading, Skinning, Deformation.*
- Permite a criação de jogos *multiplayer online*.

A compatibilidade do 3D Game Studio restringe-se a plataforma Windows, e por se tratar de uma ferramenta comercial é necessário adquirir uma licença para utilizá-la, as licenças variam de R$ 114,00 a R$ 1.500,00 (CONITEC, 2008).

Algumas Imagens:

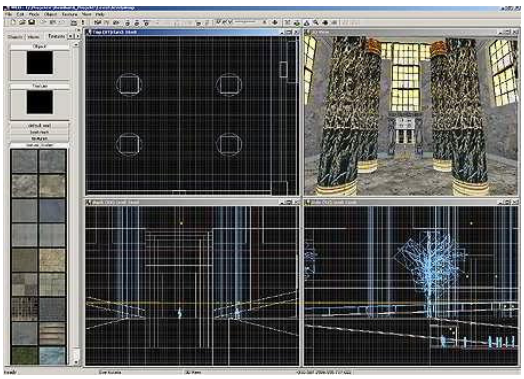


Figura 24 - Ambiente de desenvolvimento do 3D Game Studio.
Fonte: (CONITEC, 2008)

Figura 25 - Jogo desenvolvido com o 3D Game Studio.
Fonte: (CONITEC, 2008)

3.4.2 Unity3D

A Unity3D, desenvolvida pela Unity Technologies, é uma *game engine* 3D voltada para o mercado comercial, possui gráficos de ultima geração e permite tanto o desenvolvimento de jogo para computadores como para o Nintendo Wii. Uma característica interessante é o UnityPlayer, que permite a execução dos jogos desenvolvidos através de um simples navegador *web*, com a qualidade gráfica comparável a de um jogo 3D instalado normalmente no computador (DEVMASTER, 2008).

Algumas características:

- Utiliza a API gráfica DirectX ou OpenGL.
- Áudio 2D e Áudio 3D com *streaming* de áudio e vídeo.
- A engine física é baseada na *Ageia's physX Engine*.
- *Keyframe Animation*.
- Suporte a *shaders.*
- *Mesh Loading*, oferece suporte aos formatos Collada, FBX, 3DS e OBJ.
- Diversos efeitos especiais.
- Permite a criação de jogos *multiplayer*.

O Unity3D é compatível com as plataformas Windows, MacOS e Nintendo Wii. Somente a versão *trial* para MacOS está disponível para *download*, o que torna a *engine* inacessível para o publico que não utiliza este sistema operacional. O preço das licenças varia entre R$ 354,00 e R$ 3.500.00 (UNITY3D, 2008).

Algumas Imagens:

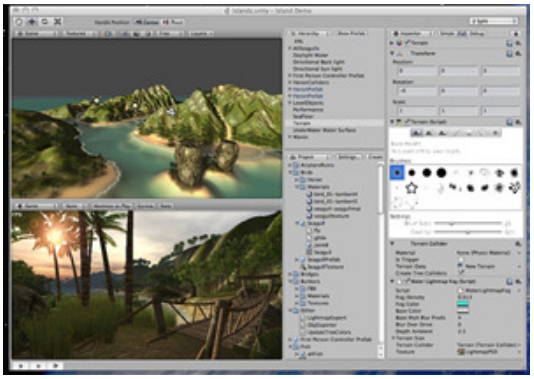


Figura 26 - Ambiente de desenvolvimento do Unity3D.
Fonte: (UNITY3D, 2008)

Figura 27 - Jogo desenvolvido com o Unity3D.
Fonte: (UNITY3D, 2008)

3.4.3 C4 Engine

A C4 Engine desenvolvida pela Terathon Software, possui ótimos gráficos, excelente suporte, e está frequentemente em atualização, com novas versões sendo lançadas aproximadamente uma vez por mês. A sua arquitetura consiste de uma coleção de camadas de *software* nos quais as mais baixas camadas interagem com o hardware e com o sistema operacional, e as camadas mais altas provêem serviços de plataforma-independente ao código do jogo. Enquanto uma parte considerável da *engine* é dedicada a renderização 3D, também há componentes dedicados a funcionalidade como áudio, rede e dispositivos de entrada (DEVMASTER, 2008).

Algumas características:

- Utiliza a API gráfica OpenGL.
- Editor de scripts gráfico.
- Suporte a *shaders.*
- *Skeletal Animation* e *Animation Blending.*
- *Mesh Loading*, oferece suporte aos formatos Collada.
- Diversos efeitos especiais (*Environment Mapping, Lens Flares, Billboarding, Particle System, Motion Blur, Sky, Water, Fire, Decals, Fog, Mirror*).
- Permite a criação de jogos *multiplayer*.

- Áudio 2D e 3D.

A C4 Engine é compatível com as plataformas Windows, MacOS e Playstation 3. Ao adquirir uma licença o comprador recebe também todo o código fonte da C4 Engine, os preços das licenças disponíveis variam de R$ 350,00 e R$ 35.600,00 (TERATHON, 2008).

Algumas Imagens:

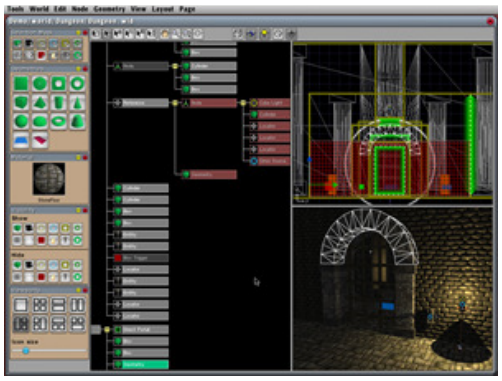


Figura 28 - Ambiente de desenvolvimento do C4 Engine.
Fonte: (TERATHON, 2008)

Figura 29 - Ambiente desenvolvido com o C4 Engine.
Fonte: (TERATHON, 2008)

3.4.4 Irrlicht

A Irrlicht é uma *game engine open source* e multiplataforma desenvolvida em C++. Consiste em uma API de alto nível para a criação de jogos e simulações 3D ou 2D. É compatível com os sistemas operacionais Windows, Linux e MacOS.

Algumas características:

- Utiliza a API gráfica OpenGL ou DirectX.
- Suporte a *dynamic lighting* e *light maps.*
- *Dynamic shadows*.
- *Hierarchical scene*.

- Suporte a diversos formatos de modelos 3D.

Algumas Imagens:

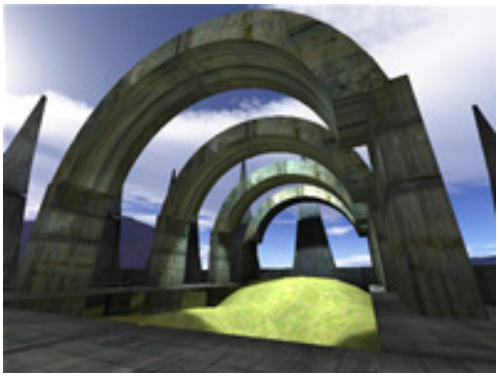


Figura 30 - Ambiente desenvolvido com a engine Irrlicht.
Fonte: (IRRLICHT, 2008)

Figura 31 - Ambiente desenvolvido com a engine Irrlicht.
Fonte: (IRRLICHT, 2008)

3.4.5 Unreal Engine

A Unreal Engine foi desenvolvida pela Epic Games, escrita em C++, oferecendo um alto grau de portabilidade. Grande parte da *engine* é escrita usando o UnrealScript, uma linguagem de script proprietária.

Uma das principais características visuais da Unreal Engine é seu *pipeline* gráfico de 64 *bits* de cores para tratamento de iluminação, conhecido *High Dynamic Range Rendering* ou *High Dynamic Range Lighting* (UNREAL ENGINE, 2008).

Algumas características:

- Utiliza a API gráfica DirectX.
- Utiliza a *engine* física *Ageia's physX Engine*.
- HDR, *per-pixel lighting, dynamic shadows.*
- Suporte a áudio 3D multiplataforma.
- *Skeletal animation system.*
- Permite a criação de jogos *multiplayer.*

A Unreal Engine está atualmente na 3° versão, é compatível com as mais recentes plataformas de consoles, Playstation 3 e XBox 360, e também compatível com a plataforma Windows. O preço da licença da versão mais recente não é revelado, mas segundo DEVMASTER (2008) ele é estimando em mais de 700 mil dólares.

Algumas Imagens:

Figura 32 - Ambiente desenvolvido com a Unreal Engine.
Fonte: (UNREAL ENGINE, 2008)

Figura 33 - Ambiente desenvolvido com a Unreal Engine.
Fonte: (UNREAL ENGINE, 2008)

3.5 CONCLUSÃO

Atualmente as *game engines* são tão importantes que estão em praticamente todos os jogos eletrônicos, seja em jogos de computador, vídeo games ou jogos para celular. Internamente uma *game engine* pode ser considerado um software complexo, mas que deve prover ao seu usuário final funções de alto nível para facilitar e agilizar o processo de criação dos jogos.

Existem hoje no mercado inúmeras *game engines*, algumas delas gratuitas, já outras podem custar mais de 700 mil dólares. Muitas empresas desenvolvedoras de jogos possuem as suas próprias *engines*, já outras utilizam produtos adquiridos de empresas especializas na criação deste tipo de software.

4 DESENVOLVIMENTO

A *game engine* desenvolvida, chamada de “3D Game Builder”, consiste em um *framework* completo, onde todos os módulos necessários para o desenvolvimento de jogos, como editor de personagens, editor de cenários, linguagem script, entre outros, estão reunidos em uma única ferramenta.

O 3D Game Builder não fica restrito somente a criação de jogos, podendo ser utilizado para a criação de qualquer tipo de ambiente virtual. O seu principal objetivo é facilitar a criação de aplicativos 3D, tanto por pessoas experientes na área de programação, como por leigos.

Este capítulo descreve a sua arquitetura, a metodologia usada para o desenvolvimento, as principais *sub-engines* e bibliotecas utilizadas e também os resultados alcançados.

4.1 PRINCIPAIS REQUISITOS

Basicamente o 3D Game Builder foi projetado para atender aos seguintes requisitos:

- Ser uma ferramenta completa para a criação de jogos, não necessitando da prévia instalação de nenhuma biblioteca externa;
- Gerenciar e gerar automaticamente a estrutura inicial dos jogos nela desenvolvidos, restando ao desenvolvedor apenas criar os mapas, editar personagens, criar eventos e definir as demais configurações;
- Permitir a criação de vários gêneros de jogos;
- Possuir uma arquitetura modular para que novos recursos sejam facilmente inseridos;
- Possuir uma interface simples e intuitiva.

4.2 PRINCIPAIS SUB-ENGINES E BIBLIOTECAS UTILIZADAS

Para o desenvolvimento do 3D Game Builder foram utilizadas diversas bibliotecas externas, a tabela 1 lista as principais bibliotecas utilizadas.

Nome	**Site Oficial**
OpenGL	http://www.opengl.org
ODE	http://www.ode.org
JEDI VCL	http://jvcl.sourceforge.net
FMOD	http://www.fmod.org

Tabela 1 - Principais bibliotecas e *sub-engines* utilizadas.

A API OpenGL foi utilizada para o processamento gráfico 3D, ou seja como *engine* gráfica. Optou-se pelo uso do OpenGL pela sua simplicidade e portabilidade. A *engine* física ODE foi usada para a simulação da física e detecção de colisões entre objetos. Para o módulo de áudio foi utilizada a biblioteca FMOD, permitindo a utilização de áudio 3D integrado diretamente com o ambiente. A linguagem de programação C++ foi utilizada para o desenvolvimento do 3D Game Builder. Em algumas interfaces foram utilizados alguns componentes da biblioteca JEDI VCL.

4.3 ARQUITETURA DO 3D GAME BUILDER

A arquitetura do 3D Game Builder é formada por diversos módulos que fornecem todos recursos para a criação de jogos, atendendo assim ao requisito de ser uma ferramenta completa e não necessitar da instalação de nenhuma biblioteca ou software externo. O módulo principal (Editor de Cenários) é responsável por gerenciar e utilizar os recursos para a geração dos mapas, do projeto e dos eventos que formaram a estrutura do jogo criado. A arquitetura completa é ilustrada na figura 34.

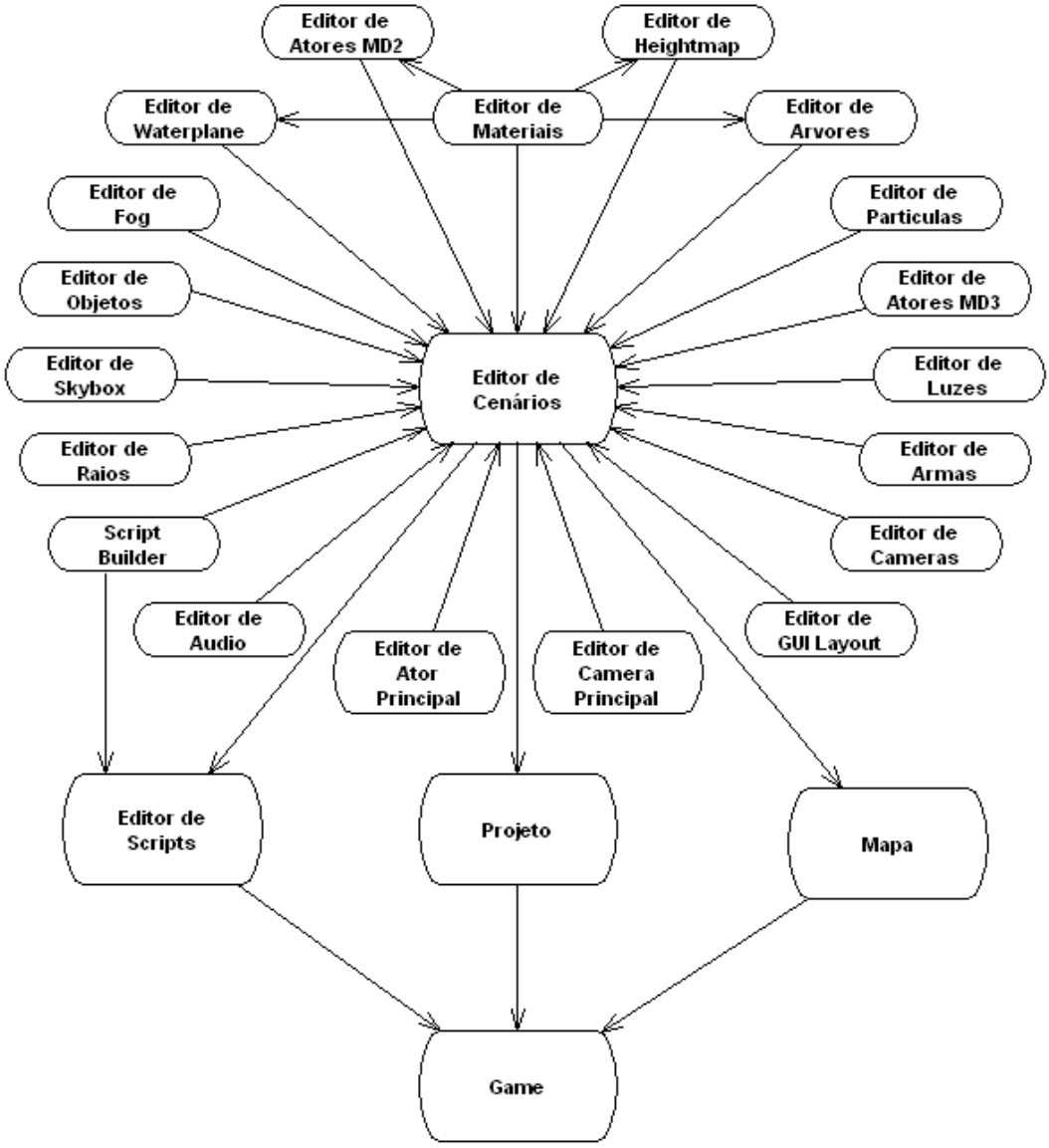


Figura 34 - Arquitetura do 3D Game Builder

Os outros módulos que fornecem recursos ao editor de cenários são descritos a seguir.

- **Editor de Materiais** - Módulo responsável pela criação e gerenciamento dos materiais que definem as características visuais dos objetos utilizados na criação dos cenários dos jogos. Vários outros módulos do 3D Game Builder fazem acesso a este módulo para a associação de matérias aos objetos neles editados. Um material é composto por diversas propriedades como, por exemplo, cores, textura, formato, mapeamento, entre outras.

- **Editor de Luzes** - Módulo para a criação de luzes personalizadas. Uma fonte emissora de luz é responsável por iluminar os objetos do ambiente e dar um grau de realismo a cena. Na criação de uma luz é possível definir as suas cores, sua forma (*Spot, Omni, Parallel*), grau de atenuação, entre outras propriedades.

- **Editor de Objetos** - Para a criação de cenários o 3D Game Builder possui diversos objetos básicos pré-definidos, como por exemplo, cubo, esfera, cilindro, entre outros. Porém para a criação de ambientes realistas são necessárias formas geométricas mais complexas, para isto é utilizado o Editor de Objetos que permite a importação de objetos criados em softwares específicos para modelagem 3D. Diversos formatos de modelos são suportados, entre eles o 3DS (3D Studio Max File), MD2 (Quake 2 Model File), LWO (Lightwave File), OBJ (Wavefront File), VRML (Virtual Reality Model File), entre outros.

- **Editor de Heightmap** - Módulo responsável pela criação e edição de terrenos. Os terrenos são criados utilizando a técnica de *heightmap*, ou seja, a superfície 3D é formada a partir de um imagem 2D em tons de cinza. Além da definição de imagem para a formação do terreno também é possível definir diversos parâmetros, como por exemplo, a precisão usada na geração do terreno, o tamanho do terreno e da textura aplicada, entre outras.

- **Editor de Atores MD2** - Módulo responsável pela criação e gerenciamento de personagens secundários com animações. O editor de atores MD2 permite a importação de modelos 3D no formato MD2 com animações criadas em software específicos para esta tarefa. Não necessariamente o objeto criado tem que ser um personagem, podendo ser somente um objeto animado para o ambiente.

- **Editor de Atores MD3** - Possui a mesma função do editor de atores MD2, porém voltado para modelos no formato MD3, este formato é uma evolução do MD2 e possibilita a criação de modelo de qualidade superior além de permitir que o personagem seja dividido em 3 partes distintas (penas, torso e cabeça), permitindo a aplicação de conceitos de modelos hierárquicos na movimentação do personagem.

- **Editor de Arvores** - Módulo que permite a criação e edição de arvores *procedurais*. A estrutura das arvores é criada em tempo real com definição das propriedades estruturantes, como por exemplo, numero de folhas, ângulo do tronco, largura do tronco, numero de ramificações, entre outras.

- **Editor de Waterplane** - Módulo para criação de superfícies que simulam água. Diversos parâmetros podem ser definidos para a sua criação, como por exemplo, a mascara que definira a estrutura do plano, a viscosidade da água, a força e a quantidade de ondas que serão geradas na superfície, entre outros.

- **Editor de Fog** - Módulo que permite a adição e edição de neblina para os ambientes. A neblina pode ser usada tanto para tornar o cenário mais realista como também para ocultar os cortes causados pelo clipping a certa distancia no ambiente. Diversos parâmetros podem ser definidos para a formação da neblina, como por exemplo, a densidade da neblina, cor, posição inicial e final, entre outros.

- **Editor de Partículas** - Módulo para a criação e edição de efeitos especiais utilizando o sistema de partículas e é possível criar efeitos realistas de fogo, fumaça, neve e vários outros. A simulação é feita a partir da definição de propriedades como densidade, quantidade e tempo de vida das partículas, fator de evaporação, cores, direção, entre outras propriedades.

- **Editor de Raios** - Módulo que permite a criação e simulação de efeitos de raios. O raio pode ser proveniente de um ponto no cenário e estender-se a outro ponto, pode ser utilizado tanto para a simulação de campos de força como para criação de grades elétricas futuristas.

- **Editor de Skybox** - Módulo responsável pela criação e edição de skybox. Um *skybox* consiste de um cubo em que 6 imagens são posicionadas em suas faces internas, e todo o resto do cenário é criado no interior deste cubo, as imagens posicionadas nas faces superiores normalmente formam o céu do ambiente e as faces inferiores o chão ou montanhas. É normalmente utilizado em ambientes externos para dar ao usuário a sensação de que o ambiente é infinito ou simplesmente para a simulação do céu ou atmosfera.

- **Editor de Câmeras** - Módulo que permite a criação de câmeras personalizadas, definido o estilo da câmera, o zoom aplicado à cena, entre outras definições. O 3D Game Builder permite que o usuário crie mais de uma câmera no mesmo cenário e em tempo de execução altere a câmera atual.

- **Editor de Áudio** - Módulo para a importação de arquivos de áudio para serem utilizados no jogo. O 3D Game Builder utiliza áudio 3D baseado em fontes emissoras e receptoras, por isso quando o áudio é adicionado no ambiente ele é associado a um objeto (emissor) e será reproduzido em relação à posição do jogador (receptor). Os formatos de áudio aceitos são o WAV e MP3.

- **Editor de GUI Layout** - Módulo que permite a edição da aparência dos objetos do tipo GUI Layout, estes objetos são utilizados para a criação de

interfaces e menus, constituem-se de botões, *labels*, painéis, formulários, entre outros.

- **Editor de Ator Principal** - Módulo responsável pela edição do personagem principal utilizado no jogo. O personagem e suas animações devem ser criadas em qualquer software especifico para modelagem 3D e exportados para o formato MD2 ou MD3 para que este possa ser importado para o 3D Game Builder. Além do modelo do personagem, diversas outras propriedades podem ser definidas, como por exemplo, a força aplicada para a movimentação, a escala de colisão do ator, a velocidade das animações, entre outras.

- **Editor de Câmera Principal** - Módulo para a edição das propriedades da câmera principal utilizada para a exibição das cenas.

- **Editor de Armas** - Módulo para criação e edição de armas que podem ser utilizadas pelo ator principal em jogos de tiro. O modelo 3D das armas devem ser modelados juntamente com o personagem e posteriormente exportados em arquivos separados para que ao serem importados para o 3D Game Builder as animações dos modelos estejam sincronizadas. Além das propriedades da arma o editor de armas também permite a edição da forma, velocidade e outras definições dos disparos feitos pelas armas.

- **Editor de Scripts** - Módulo responsável pela edição e criação de scripts associados a eventos que podem ocorrer durante a execução do jogo, tais como a colisão de objetos, interação do jogador, entre outros. A linguagem de programação utilizada nos scripts é baseada na linguagem Object Pascal e permite dês da manipulação de strings até o controle de objetos 3D, acesso a banco de dados, manipulação de arquivos, entre outras funcionalidades. Cada objeto criado no mapa pode possuir scripts de eventos a ele associados, além dos eventos globais referentes ao mapa.

- **Script Builder** - Módulo que facilita a criação de scripts sem a necessidade que estes sejam programados. Constitui-se de uma biblioteca com scripts comumente utilizados onde o usuário define algumas constantes e opções, e o Script Builder cria e insere automaticamente o script aos eventos correspondentes.

O projeto, mapas e demais recursos criados no 3D Game Builder são utilizados para gerar uma nova aplicação, ou seja o jogo nele criado. A arquitetura desta aplicação é ilustrada na figura 35.

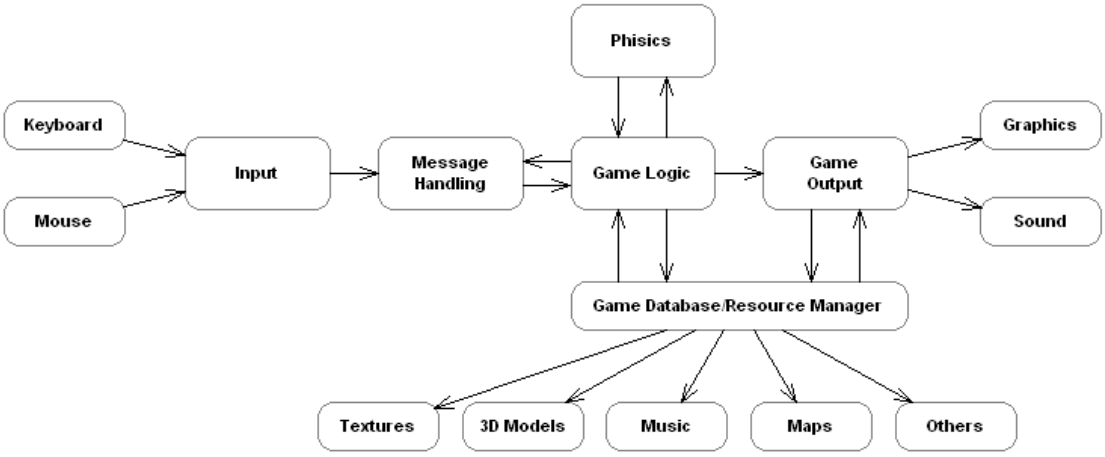


Figura 35 - Arquitetura de um jogo criado no 3D Game Builder

Os módulos que compõe a arquitetura desta aplicação são:

- **Input** - Módulo responsável por receber as entradas de dados provenientes do teclado ou *mouse*, estas informações são enviadas para o módulo *Message Handling*.

- **Message Handling** - Módulo que interpreta e gerencia todos os comandos enviados pelo usuário através do *Input*.

- **Game Logic** - Módulo principal, responsável por toda a lógica envolvida no jogo. É também encarregado de fazer a comunicação e gerenciamento de todos os outros módulos da aplicação.

- **Phisics** - Módulo que executa todos os cálculos envolvendo a física atuante nos objetos do ambiente, tanto para aplicar determinadas forças ou para verificar colisões entre objetos.

- **Game Database/Resource Manager** - Módulo responsável por gerenciar e prover acesso a todos os recursos utilizados no jogo, texturas, modelos, musicas, mapas, entre outros.

- **Game Output** - Módulo responsável por exibir o estado atual da aplicação ao usuário através de gráficos e áudio.

4.4 APLICAÇÕES

Para demonstrar as funcionalidades do 3D Game Builder diversos jogos e protótipos foram desenvolvidos, a seguir serão detalhadas algumas destas aplicações.

1.4.1 Guerra do Contestado

Para demonstrar a capacidade do 3D Game Builder em criar jogos longos, complexos e educativos foi desenvolvido um do jogo do gênero RPG abordando o tema da Guerra do Contestado, um conflito armado que ocorreu no estado de Santa Catarina durante os anos de 1912 à 1915. O jogo inicia-se em 1912 com a aparição do terceiro monge e vai até a batalha de Irani, onde ele morre. Durante a aventura o jogador vive o papel de um caboclo e pode conhecer a história dos monges do contestado, entender o porquê de tanta revolta da população da região e participar de todo o desenrolar dos fatos.

A história é toda contada através de diálogos entre os personagens como pode ser visto na figura 38, para poder avançar no jogo o usuário deve acompanhar

e entender a história para saber os próximos passos a serem seguidos. Existem diversos desafios a serem vencidos pelo jogador, como por exemplo, reunir alguns seguidores para o grupo do monge José Maria.

Algumas cenas do jogo podem ser vistas nas figuras 36, 38 e 39, e a figura 37 mostra um dos ambientes sendo do jogo sendo editado.

Figura 36 - Cena do jogo da Guerra do Contestado

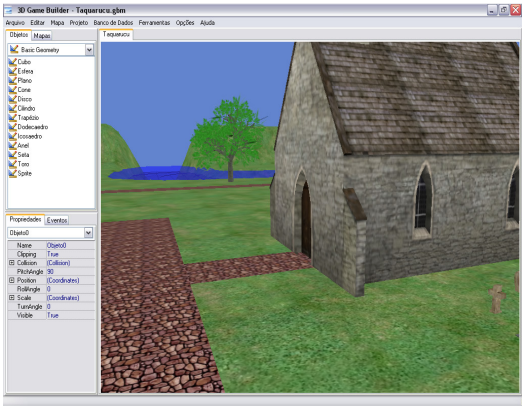


Figura 37 - Edição de um cenário da Guerra do Contestado

Figura 38 - Diálogo entre personagens da Guerra do Contestado

Figura 39 - Cena de uma animação do jogo da Guerra do Contestado

1.4.2 Coelhinho Saltitante

O Coelhinho Saltitante foi um jogo simples desenvolvido no 3D Game Builder para demonstrar a sua capacidade de criação de jogos infantis ou casuais. O objetivo do jogo é saltar com o coelho pelo caminho formado pelas plataformas sem cair e antes do fim do tempo definido para cada nível.

O jogo é formado por diversas fases com incremento de dificuldade em cada nível avançado. Nos níveis mais avançados varias dificuldades surgem para o jogador, como por exemplo, plataformas moveis ou falsas. Nas fases o jogador também pode encontrar objetos para facilitar o seu avanço, como por exemplo, para aumentar o numero de vidas, aumentar o tempo restante, entre outros itens.

Algumas cenas do jogo podem ser vistas nas figuras 40, 42 e 43, e a figura 41 mostra um dos ambientes sendo do jogo sendo editado.

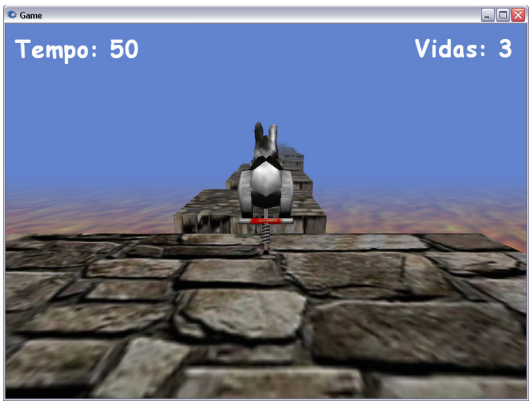


Figura 40 - Cena do jogo Coelhinho Saltitante

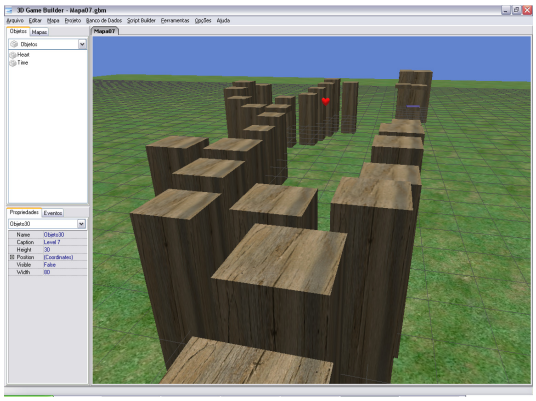


Figura 41 - Edição de um dos cenários do Coelhinho Saltitante

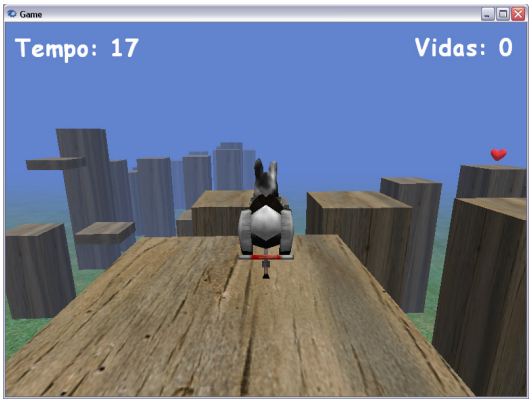


Figura 42 - Cenário de um nível avançado do Coelhinho Saltitante

Figura 43 - Cenário de um nível avançado do Coelhinho Saltitante

1.4.3 UnC Virtual

O conceito de *walkthrough* consiste na navegação por um ambiente virtual de maneira interativa, o walkthrough vem difundindo-se muito rapidamente no ramo

arquitetônico, pois permite que determinada construção seja vista de todos os ângulos antes mesmo de ser construída, possibilitando assim possíveis alterações no projeto de construção e também dando uma visão bastante próxima da real ao cliente.

Para demonstrar que o 3D Game Builder também pode ser utilizado como uma ferramenta para a criação *walkthrough* foi desenvolvido a UnC Virtual, uma representação gráfica e interativa do ambiente real da Universidade do Contestado campus de Porto União. O ambiente foi criado a partir da planta baixa e também da analise visual dos prédios, dando ênfase aos laboratórios de informática que tiveram até o seu interior detalhado.

Algumas cenas da simulação podem ser vistas nas figuras 44, 46 e 47, e a figura 45 mostra um dos ambientes sendo do jogo sendo editado.

Figura 44 - Simulação do laboratório 1 da UnC

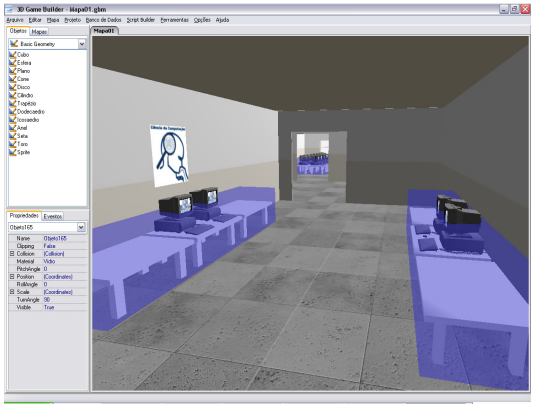


Figura 45 - Edição de um dos ambientes da UnC

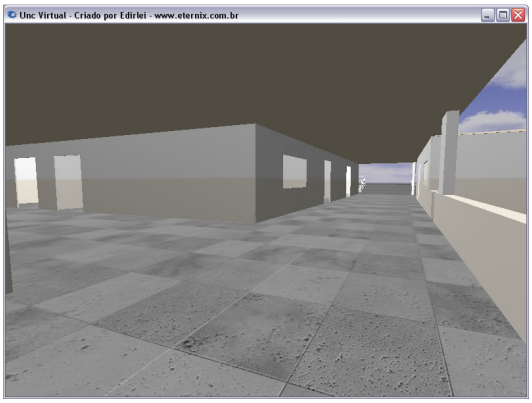


Figura 46 - Simulação dos corredores da UnC

Figura 47 - Simulação da fachada da UnC

1.4.4 Interação com ambientes 3D através de movimentos corporais

Nas últimas décadas as inovações presentes nos computadores vieram evoluindo gradativamente, porém até hoje é normal que toda a interação entre o usuário e a maquina seja normalmente feita através de teclado, *mouse* ou *joystick*. Nos jogos esta realidade esta mudando, novas tecnologias e métodos de interação estão surgindo, como por exemplo, o controle através movimentos corporais, comandos de voz, entre outros.

Buscando atingir estes novos métodos de interação do usuário com os jogos, foi incluído no 3D Game Builder um módulo de reconhecimento de movimentos, que permite ao usuário navegar e interagir com ambientes 3D utilizando uma *webcam* e efetuando movimentações com a sua face, por exemplo, ao olhar para o lado o personagem no jogo irá ratacionar-se para a direção correspondente.

As figuras 48 e 49 mostram cenas de um jogo de tiro onde toda a movimentação do personagem pode ser feita através de movimentos corporais do jogador.

Figura 48 - Jogo com movimentação através de reconhecimento de movimentos

Figura 49 - Interação com o jogo através de movimentos corporais.

1.4.5 Interação com uma maquete através de um ambiente 3D

Durante o desenvolvimento do 3D Game Builder foi inserido na sua linguagem script a possibilidade de enviar dados para a porta paralela do computador, com isto e com o uso de circuitos eletrônicos é possível que diversos dispositivos possam ser acionados e controlados pelas aplicações criadas no 3D Game Builder.

Para demonstrar esta funcionalidade foi criado no 3D Game Builder um ambiente 3D para simular uma maquete já existente de uma casa, com luzes que pode ser acesas e portas que podem ser abertas com o uso de pequenos motores. Com a ajuda de um circuito conectado a porta paralela foi possível executar todos os acionamentos dos componentes eletrônicos que compõem a casa, permitindo ao usuário navegar e interagir com o ambiente 3D tendo esta interação refletida na maquete, por exemplo, quando o usuário passa por uma porta no ambiente virtual a mesma porta é aberta na maquete.

Esta funcionalidade pode ser útil para dar ao usuário uma maior sensação de imersão virtual, onde as ações executadas no ambiente virtual são refletidas no mundo real.

As figuras 50 e 53 mostram cenas do ambiente virtual da casa, a figura 52 mostra uma imagem real da maquete e a figura 51 mostra a edição da do ambiente virtual da maquete.

Figura 50 - Simulação virtual da maquete

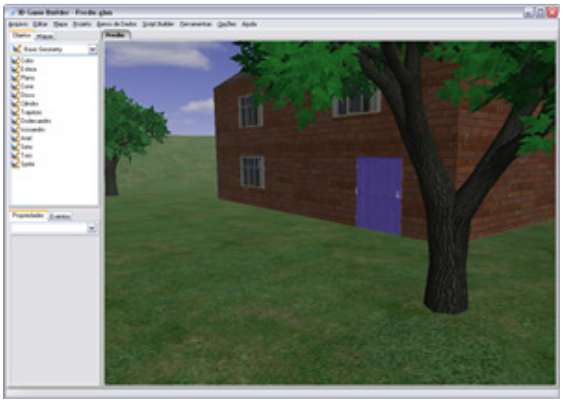


Figura 51 - Edição da maquete virtual.

Figura 52 - Simulação virtual da maquete

Figura 53 - Edição da maquete virtual.

5 CONCLUSÃO

Logo no surgimento dos primeiros jogos eletrônicos eles chamaram a atenção das pessoas devido à diversão que tinham a oferecer. Porém, atualmente para muitas pessoas os jogos já deixaram de ser apenas uma diversão e se tornaram a sua profissão, inúmeras empresas especializadas na criação de jogos surgiram em todo o mundo, inclusive no Brasil. O principal motivo do surgimento destas empresas está ligado aos lucros que este mercado tem a oferecer.

Internacionalmente a lucratividade com a comercialização de jogos é enorme, e já superou até mesmo a indústria cinematográfica. Porém estes lucros poderiam ser ainda maiores, se não existisse o principal vilão deste mercado, a pirataria. O Brasil também está entrando para este promissor segmento, porém ainda a “passos lentos”, o motivo de tal lentidão também é a pirataria, que devido à baixa renda da população se torna ainda maior. Muitas empresas que surgiram há alguns anos atrás como as primeiras iniciativas no ramo no país hoje já não existem mais, mas mesmo assim novas empresas vêm surgindo e outras mesmo com a pirataria se mantêm no mercado, a principal tática utilizada por tais empresas é a exportação dos jogos aqui criados para outros países, conseguindo desta maneira uma maior renda com a comercialização internacional de seus jogos.

Além do quesito financeiro que os jogos têm a oferecer, existe também o lado dos “sonhadores”, pessoas que tem como sonho criar um jogo, sem se importar se iram ter algum lucro com ele.

A criação de jogos é uma área atraente, mas não é uma tarefa simples, pois envolve diversas áreas da computação, como banco de dados, redes, computação gráfica, estrutura de dados, inteligência artificial, entre outras. Justamente para facilitar este desenvolvimento existem as *game engines*, que ao proverem funcionalidades de alto nível facilitam e agilizam a codificação de jogos.

Praticamente todos os jogos existentes na atualidade possuem internamente uma *game engine* provendo as suas funcionalidades de baixo nível. Seja ela desenvolvida especificamente para aquele jogo ou uma genérica utilizada em vários jogos.

Levando isto em consideração este trabalho teve como objetivo a criação de uma *game engine* que permitisse a criação de jogos de maneira rápida e de modo simplificado.

A *game engine* desenvolvida, chamada de 3D Game Builder, atendeu a todas as expectativas e requisitos definidos inicialmente, além de oferecer novas funcionalidades que foram agregadas durante o desenvolvimento, fornecendo uma interface simples e intuitiva para pessoas com pouca experiência e oferecendo recursos avançados para desenvolvedores experientes.

O 3D Game Builder foi desenvolvido com o objetivo de ser uma ferramenta completa para criação de jogos, porém não ficou restrito a jogos, podendo ser utilizado para diversas tarefas, como por exemplo, para a simulação de ambientes 3D, *walkthroughs*, entre outras aplicações. Ou seja, se tornou uma ferramenta genérica para criação de qualquer tipo de aplicação 3D.

A ferramenta não se tornou apenas mais uma entre as existentes, se tornou bastante popular entre os desenvolvedores da área, principalmente por ser a primeira do gênero desenvolvida no Brasil. Durante os 6 primeiros meses em que foi disponibilizada para *download* teve mais de 25 mil *downloads*, além de se tornar notícia em diversos sites de desenvolvimento de jogos.

O desenvolvimento deste trabalho serviu para a aquisição de muito conhecimento, justamente pelo motivo dos jogos envolverem diversas áreas da computação.

5.1 TRABALHOS FUTUROS

A Game Engine desenvolvida constitui-se de uma ferramenta completa para a criação de jogos, porém ainda é possível que novas funcionalidades e tecnologias sejam adicionadas a ela. As sugestões para trabalhos futuros são:

- **VRML** - O VRML (*Virtual Reality Modeling Language*), é uma tecnologia que permite a visualização e interação com objetos e ambientes 3D através de equipamentos de realidade virtual, como óculos estereoscópicos, luvas digitais e capacetes de imersão. Como o 3D Game Builder se tornou uma

ferramenta que pode ser usada tanto para a criação de jogos como para o desenvolvimento de ambientes e simulações 3D, a implementação de um exportador para o formato VRML seria algo interessante para ser estudado. Através de um breve estudo realizado sobre o formato VRML constatou-se que ele é simples e pode ser facilmente integrado junto a arquitetura do 3D Game Builder. Outro ponto interessante sobre o VRML é que os ambientes neste formato podem ser visualizados através da internet, utilizando um navegador web, como por exemplo, o Internet Explorer, onde com a instalação de um *plugin* é possível fazer navegação e interação direta com o ambiente 3D.

- **Módulo para Modelagem** - Atualmente o 3D Game Builder permite que os ambientes 3D sejam criados no editor de cenários, para isso o usuário dispõem de diversos objetos pré-definidos (cubos, esferas, cilindros, entre outros), porém para a criação de ambientes realistas é necessário que objetos mais complexos sejam utilizados, para isso existe o importador de objetos externos que permite a importação de objetos modelados em *softwares* específicos para esta tarefa. Isto acarreta na dependência do 3D Game Builder com estas ferramentas. Justamente para suprir esta dependência, sugere-se o desenvolvimento de um módulo que permita a modelagem de objetos complexo dentro do próprio 3D Game Builder.

- **Criação de Animação** - A mesma dependência citada acima para a criação de objetos complexos ocorre na criação de personagens para os jogos, estes devem ser primeiramente modelados e animados em softwares específicos para tal tarefa. Sugere-se como trabalho futuro a implementação de um modulo que permita a seleção de personagem pré-definido ou até mesmo a modelagem destes no próprio 3D Game Builder e também a criação e edição das animações destes personagens. Uma possível solução seria a utilização de técnicas de animação implícitas, visto que permitem maior liberdade de movimentos, além de que a implementação de um editor para este tipo de animação se adaptaria melhor a arquitetura do 3D Game Builder.

REFERÊNCIAS

ABRAGAMES. **A Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil.** Pesquisa realizada em 01 mai. 2005. Disponível em: http://www.abragames.org/docs/PesquisaAbragames.pdf Acessado em: nov. 2008.

ABRAGAMES. **A indústria brasileira de jogos eletrônicos: Um mapeamento do crescimento do setor nos últimos 4 anos.** Pesquisa realizada em 17 jul. 2008. Disponível em: http://www.abragames.org/docs/Abragames-Pesquisa2008.pdf Acessado em: nov. 2008.

ARMITAGE, Grenville; CLAYPOOL, Mark; BRANCH, Philip. **Networking And Online Games: Understanding And Engineering Multiplayer Internet Games.** England: John Wiley & Sons Ltd. 2006.

ASSIS, D., MATIAS, A**. Game supera cinema como opção de entretenimento em 2003**. Artigo da Folha de São Paulo, 31 dez. 2003. Disponível em: www1.folha.uol.com.br/folha/ilustrada/ult90u40114.shtml. Acessado em: nov. 2008.

ASTLE, Dave; HAWKINS, Kevin. **Beginning OpenGL Game Programming.** Boston: Thomson, 2004.

AZEVEDO, Eduardo; CONCI, Aura. **Computação Gráfica - Teoria e pratica.** Editora Campus, 2003

BAKIE, R.T.. **A Brief History of Video Games.** Introduction to Game Development. Hingham, USA, 2005.

BATTAIOLA, André L. et al. **Desenvolvimento de jogos em computadores e celulares.** Revista de Informática Teórica e Aplicada, 2001.

BOURG, M. David; SEEMAN, Glenn. **AI for Game Developers.** O'Reilly, 2004.

CARVALHO, P. H. Gustavo. **Um Modelo Preditivo para Desenvolvimento de Jogos de Computador.** Trabalho de Graduação, Universidade Federal de Pernambuco, 2006.

CODEPLEX. **GLSL Debugger**. Disponível em: http://www.codeplex.com/hl2glsl/Wiki/View.aspx?title=GLSL%20Debugger&referringTitle=Home. Acessado em: nov. 2008.

CONITEC. **3D Game Studio.** Disponível em: http://www.conitec.net/english/gstudio/ Acessado em: nov. 2008.

DALMAU, Daniel S. **Core Techniques and Algorithms in Game Programming**. New Riders Publishing. 2003.

DEVMASTER. **DevMaster.net - 3D Game and Graphics Engines Database.** Disponível em: http://www.devmaster.net/engines/. Acessado em: nov. 2008.

EBERLY, H., David. **3D Game Engine Design - A Practical Approach To Real-Time Computer Graphics.** San Francisco: Morgan Kaufmann Publishers, 2004.

GILLIUS. **GNE - Game Networking Engine.** Disponível em: http://www.gillius.org/gne/. Acessado em: nov. 2008.

IRRLICHT. **Irrlicht Engine.** Disponível em: http://irrlicht.sourceforge.net/. Acessado em: nov. 2008.

JONES, Wendy. **Beginning DirectX 9.** Boston: Thomson, 2004.

KISHIMOTO, André. **Brasil: uma Indústria em Crescimento.** PDJzine – Revista Eletrônica de Distribuição Livre da Programadores e Desenvolvedores de Jogos. 1° Edição, mai. 2006. Disponível em: http://www.programadoresdejogos.com/pdjzine/pdjzine01.zip. Acessado em: nov. 2008.

KOVACH, Peter J. **Inside Direct3D.** Redmond, Washington: Microsoft Press. 2000.

MIGUEL, G. Cesar; MONTEIRO, L., R., Júlio; CAVALHIERI, A., Marcos. **Componentes de uma Game Engine.** 2006. Disponível em: http://cognitio.incubadora.fapesp.br/portal/atividades/cursos/posgrad/jogos_eletronicos/2006/trabalhos/tf2/TF2-M_GameEngine_Cesar-Marcos-Julio.pdf. Acessado em: nov. 2008.

MILLINGTON, Ian. **Game Physics Engine Development.** San Francisco: Morgan Kaufmann, 2007.

MIFFLIN, H. **The American Heritage Dictionary of the English Language.** Fourth Edition, 2006.

PESSOA, C. **wGEM: Um Framework de Desenvolvimento de Jogos para Dispositivos Móveis.** Dissertação de Mestrado, Pernambuco, Novembro, 2001.

PESSOA, A. C. Carlos; RAMALHO, L. Geber; BATTAIOLA, L. André. **wGEM: um Framework de Desenvolvimento de Jogos para Dispositivos Móveis.** 2002.

SOFTEX. **Tecnologias de Visualização na Indústria de Jogos Digitais.** Relatório Softex, 2005. Disponível em: http://www.softex.br/observatorio/_publicacoes/default.asp. Acessado em: nov. 2008.

TERATHON. **C4 Engine**. Disponível em: http://www.terathon.com/c4engine/index.php. Acessado em: nov. 2008.

UNIDEV. **Portal de Programação de Jogos.** Disponível em: http://www.unidev.com.br. Acessado em: set. 2008.

UNITY3D. **UNITY: Game Development Tool.** Disponível em: http://unity3d.com/. Acessado em: nov. 2008.

UNREAL ENGINE. **Unreal Technology.** Disponível em: http://www.unrealtechnology.com/. Acessado em: nov. 2008.

WANGENHEIM, V., Aldo. **Modelos Hierárquicos.** Anotações de aula, Computação Gráfica, UFSC, 2003.

WIKIPEDIA. **Game Engine**. Disponivel em: http://en.wikipedia.org/wiki/Game_engine. Acessado em: jan. 2008.

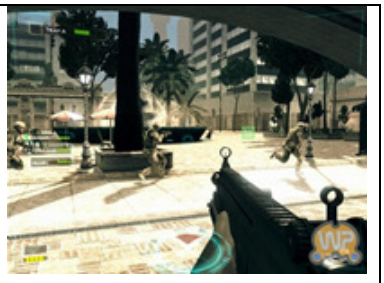
WIKIPEDIA. **Physics Engine.** Disponível em: http://en.wikipedia.org/wiki/ Physics_engine Acessado em: set. 2008.

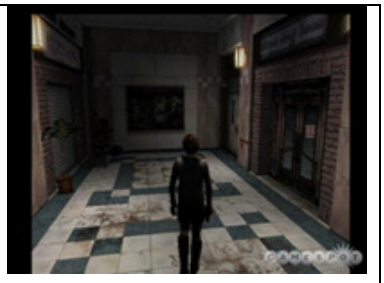
WRIGHT, Richard; LIPCHAK, Benjamin; HAEMEL, Nicholas. **OpenGL SUPERBIBLE: Comprehensive Tutorial and Reference.** 4° Edição. United States: Pearson Education Inc. 2007.

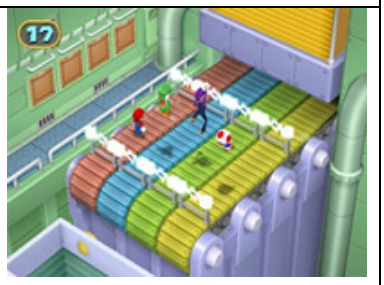
ZERBST, Stefan; DÜVEL, Oliver. **3D Game Engine Programming.** Boston: Thomson, 2004.

ANEXO A – Tipos de Jogos

Gênero	Descrição	Imagem
Aventura	Caracteriza-se pela exploração de ambientes e interação com outros personagens. Os jogos são mais focados na narrativa e não em desafios baseados em reflexos.	 Rayman 3
Ação	É bastante utilizado na combinação com outros gêneros. A principal característica é a utilização de elementos rápidos de combate e movimentação.	 Starcraft Ghost
Plataforma	Envolve jogos bidimensionais com personagens andando e pulando sobre plataformas.	 Super Mario 3
Luta	Nos jogos deste gênero o jogador luta contra outros jogadores ou contra o computador, normalmente utilizando artes marciais ou armas.	 Tekken 5

FPS	Variação do gênero ação onde o jogador é colocando em uma perspectiva de visão de primeira pessoa.	 Ghost Recon 3
RTS	Em jogos típicos deste gênero o objetivo é coletar recursos, construir um determinado exercito e controlar as suas unidade para atacar o inimigo.	 Warcraft 3
Estratégia por Turnos	Gênero similar ao RTS, porém neste as ações dos jogadores são realizadas em turnos alternados.	 Civilization IV
RPG	Versão digital dos jogos de RPG clássicos.	 Final Fantasy XIII
MMORPG	RPG digital presente em um mundo on-line e povoado simultaneamente por centenas de jogadores.	 World of Warcraft

Horror	Gênero que envolve a exploração de ambientes que tendem a provocar medo psicológico nos jogadores.	 Silent Hill 3
Simulação	Este gênero busca simular com alto grau de realismo ambientes ou situações do mundo real.	 Flight Simulator 2004
Corrida	Gênero onde é objetivo é a disputa entre jogadores em corridas, podendo ser através de carros, motos, bicicletas, etc.	 Project Gotham Race 3
Esporte	Gênero que retrata esportes reais como futebol, tênis, basquete, etc.	 FIFA 2006
Rítmico	Este gênero explora as habilidades de sincronização do jogador com uma determinada musica.	

		Dance Dance Revolution
Puzzle	Gênero que combina elementos identificação de padrões, lógica e estratégia.	 Bejeweled
Mini-game	São jogos simples e rápidos. Também podem ser inseridos em jogos maiores como um pequeno desafio.	 Mario Party 7
Tradicional	Classe que envolve todos os jogos tradicionais de cartas e tabuleiros.	 Chessmaster 10000
Educativo	Gênero que tem como objeto ensinar de maneira lúdica o jogador sobre determinado tema.	 Carmen Sandiego

Advergames	Gênero cujo objetivo é fazer a divulgação de um produto, empresa ou ponto de vista.	 Clio Xtrem Racer

Tabela A1 - Gêneros de jogos.
Fonte: (CARVALHO, 2006, adaptado).