3D GAME BUILDER: UMA GAME ENGINE PARA CRIAÇÃO DE AMBIENTES TRIDIMENSIONAIS [1]

Edirlei E. Soares de Lima [2], Pedro Luiz de Paula Filho [3]

Resumo

Este artigo apresenta o *3D Game Builder*, uma ferramenta para a criação de jogos e ambientes 3D. Com este software é possível criar, com relativa facilidade, qualquer tipo de aplicação tridimensional. Será apresentada a sua arquitetura, suas principais funcionalidades e aplicações.

Abstract

This article presents the 3D Game Builder, a tool to the creation of 3D games and environments. With this software it is possible create, with relative easiness, any type of three-dimensional application. In this paper will be show its architecture, main functionalities and applications.

Introdução

Atualmente os jogos eletrônicos movimentam o bilionário mundo das produtoras e vêm causando uma revolução no entretenimento digital. Em 2003, o mercado mundial de jogos eletrônicos movimentou cerca de US$ 22,3 bilhões somente com a comercialização de jogos, se forem incluídos também os gastos com hardware e acessório 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 (ASSIS, 2003).

No Brasil, a 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 a pesquisa, há no Brasil o registro de 55 empresas desenvolvedoras de jogos em atividade, 33% delas no Paraná, 30% em São Paulo, 12% no Rio de Janeiro. De acordo com Marcelo Carvalho, 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).

Comparado com o mercado de jogos 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 40880 usuários cadastrados, muitos destes, tendo como principal *hobby* a criação de jogos ou alguma outra área relacionada (UNIDEV, 2007).

Porém, criar jogos não é uma tarefa simples, ela envolve várias áreas da computação, como banco de dados, redes, computação gráfica, estrutura de dados, inteligência artificial, entre outras. Justamente por este motivo, existem softwares que tem como objetivo facilitar este desenvolvimento, estas ferramentas são chamadas de g*ame engines*, o seu principal objetivo é dar ao usuário uma maior abstração da programação envolvida no processo de desenvolvimento de um jogo.

O termo g*ame engine* surgiu durante a década de 90, juntamente com os primeiros jogos 3D (Doom, Quake) e a necessidade de criar-se jogos mais complexos, além disto, surgiram equipes de desenvolvimento, e para manter um certo padrão de programação entre os

desenvolvedores, foram criadas as primeiras *game engines*. Atualmente existem no mercado inúmeras *game engines*, algumas delas grátis, já outras podem chegar a custar U$ 10.000.

O objetivo deste trabalho é desenvolver uma g*ame engine 3D*, que possibilite a qualquer pessoa criar seus próprios jogos 3D com relativa facilidade.

Metodologia

A *Game Engine* desenvolvida, chamada de "*3D Game Builder*", é uma ferramenta destinada a criação de jogos, mas não fica restrita somente a estes, podendo ser utilizada para a criação de qualquer tipo de ambiente virtual. O seu principal objetivo é facilitar a criação de aplicativos 3D (três dimensões), tanto por pessoas experientes na área de programação, como por leigos.

A ferramenta foi desenvolvida utilizando a API *OpenGL* (*Open Graphics Library*), que é usada para a escrita de aplicações gráficas 3D e que consiste em cerca de 250 funções que podem ser usadas para a criação de cenas tridimensionais, porém possui a desvantagem de ser totalmente orientada a código, não possuindo nenhum tipo de interface gráfica que facilite a criação destas cenas. Justamente para facilitar este processo é que foi criado o *3D Game Builder*. O *3D Game Builder* possui diversas funcionalidades que abstraem toda a programação envolvida no processo de criação de um jogo, evitando assim que os usuários necessitem de um prévio estudo de uma determinada linguagem de programação.

No *3D Game Builder* todo o ambiente 3D é criado utilizando o conceito RAD (*Rapid Application Development*). Todos os objetos 3D são criados com apenas um clique do *mouse*. Existem ainda diversos objetos pré-definidos, como cubos, esferas, cones, entre outras formas geométricas.

Além das formas geométricas básicas presentes, também é possível importar modelos 3D criados em outros aplicativos específicos para a modelagem de objetos tridimensionais. A modelagem 3D é uma área da computação gráfica que tem como objetivo a geração de entidades em três dimensões, como objetos ou conjuntos de objetos. Para elaboração dos objetos são utilizadas ferramentas computacionais avançadas e direcionadas para este tipo de tarefa como o *3D Studio Max*, que é muito utilizado na produção de filmes de animação, comerciais para TV e jogos. A modelagem em 3D conta com uma enorme variedade de ferramentas genéricas, permitindo uma comunicação mais fácil entre dois programas diferentes, pois a maioria delas exportam e importam os mesmos formatos de arquivo. As técnicas de modelagem mais conhecidas são: técnica por polígonos, técnica por vértices e técnica por bordas. Todas elas são realizadas através da criação de uma malha complexa de segmentos que dão forma ao objeto.

O *3D Game Builder* suporta vários formatos de objetos externos, abaixo são listados alguns deles:

- 3DS: Formato padrão do *3D Studio Max*, contém informações do objeto, seus atributos, referências a texturas, informações de luz, entre outras.

- MD2: Formato usado inicialmente no jogo *Quake 2*, além das informações sobre o modelo contém também as suas animações baseadas em *frames*.

- MD3: Formato usado pelo jogo *Quake 3*, possui informações sobre o modelo e também sobre as suas animações que baseadas em vértices, é uma evolução do formato MD2.

- MD5: Formato usado inicialmente no jogo *Doom 3*, alem das informações sobre o modelo, contém também as suas animações baseadas em esqueletos.

- OBJ: Formato usado por diversos programas de modelagem 3D, possui informações sobre o objeto e suas texturas.

- MS3D: Formato usado pelo programa de modelagem *MilkShape 3D*, possui informações sobre vértices, triângulos, texturas, *joints* entre outras.

- LWO: Formato padrão do programa *Lightwave*, possui informações sobre o objeto e seus materiais.

- VRML: Formato usado para guardar objetos usados em realidade virtual.

- SMD: Formato usado em jogos como o *Half Life*, possui informações sobre o objeto, seus materiais e animações.

- BSP: Formato usado por diversos jogos, é normalmente usado para guardar mapas.

Nos objetos criados ou importados de outros *softwares* para o *3D Game Builder* é possível aplicar um material a eles, este material é uma espécie de película que ficará recobrindo toda a superfície do mesmo, esta película é composta por diversas características, entre elas algumas cores e texturas. Ao aplicar-se um material a um objeto, este ganha todas as características associadas ao material, como *ambient color, diffuse color, emisive color, specular color, shininess e alpha.* O *emissive color* representa a cor emitida pelo objeto, já o *ambient color, diffuse color* e o *specular color* representam as cores assumidas pelo objeto perante a interação com as luzes do ambiente. Na figura 1 é possível ver a combinação de cores provenientes de uma luz emitida sobre o material do objeto. O *shininess* é utilizado para controlar o quanto brilhante o material será, e o *alpha* é usado para torná-lo parcialmente ou totalmente transparente. Ao utilizar uma textura no material este pode também utilizar o *alpha channel*, usado para controlar a transparência e opacidade da textura aplicada (EBERLY, 2004). Na figura 2 é possível ver um objeto utilizando uma textura, esta é basicamente uma imagem aplicada ao material.

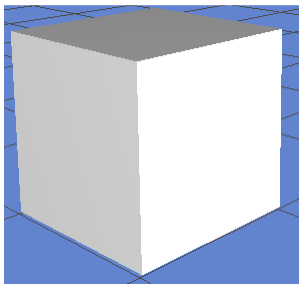

Figura 1: Cubo sem textura
Fonte: O autor

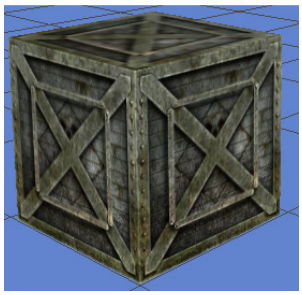

Figura 2: Cubo com textura
Fonte: O autor

Para a criação de ambientes mais realistas o *3D Game Builder* possibilita trabalhar com 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. É possível utilizar três tipos diferentes de luzes:

- *Directional Light*: São raios de luz emitidos para uma determinada posição, estes são infinitos e paralelos como mostrado na figura 3.
- *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 4.
- *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 5.

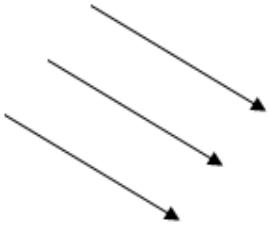

Figura 3: Directional Light
Fonte: O autor

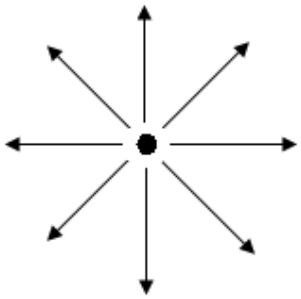

Figura 4: Point Light
Fonte: O autor

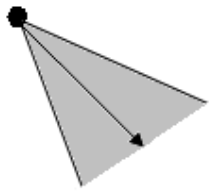

Figura 5: Spot Light
Fonte: O autor

Na representação de um objeto 3D, 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 (WANGENHEIM, 2003). A translação consiste em mover o objeto para um determinado ponto, como mostrado na figura 6. A rotação é o ato de rotacionar o objeto sobre um eixo, como na figura 7. O escalonamento consiste em aumentar ou diminuir o tamanho de um objeto, figura 8 (ASTLE; HAWKINS, 2004). No *3D Game Builder* estas transformações podem ser feitas em tempo de criação do ambiente, através da alteração das propriedades do objeto, e também durante a execução do aplicativo através de *scripts*.

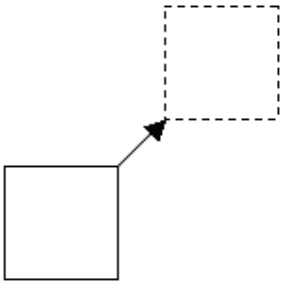

Figura 6: Translação
Fonte: O autor

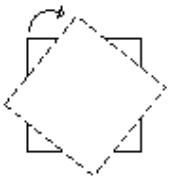

Figura 7: Rotação
Fonte: O autor

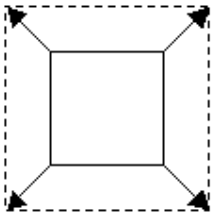

Figura 8: Escalonamento
Fonte: O autor

Na criação de um ambiente é possível adotar o uso de objetos hierárquicos para a formação do cenário. 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 9 é possível observar um exemplo de modelo hierárquico, formando uma espécie de perna "rudimentar", nela todos os nodos estão interligados, na figura 10 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 11 é aplicada uma rotação no nodo principal do objeto hierárquico, consequentemente todos os outros objetos filhos sofrerão transformações equivalentes.

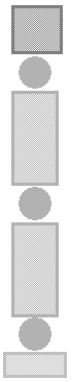

Figura 9
Fonte: WANGENHEIM, 2003

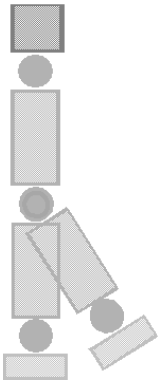

Figura 10
Fonte: WANGENHEIM, 2003

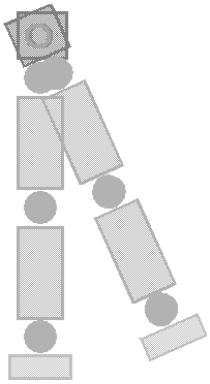

Figura 11
Fonte: WANGENHEIM, 2003

Em jogos ou ambientes mais complexos é comum ver em um mesmo cenário um número grande de objetos, para que estes sejam utilizados é necessário primeiramente que sejam carregados em memória, este processo consome muita memória do sistema. Para evitar este tipo de problema no *3D Game Builder* é possível utilizar objetos por referência. Este tipo de objeto consiste em uma representação de outro já existente na cena, este possui todas as características do objeto referenciado, porém pode ocupar uma posição diferente no ambiente, não necessitando da criação de uma nova instancia do objeto em memória. Caso este conceito não seja utilizado todo novo objeto tem que ser carregado em memória, mesmo que já exista outro igual no ambiente.

Em quesito de desempenho, o *3D Game Builder* utiliza também outras duas técnicas para melhores resultados na exibição dos ambientes 3D. O *Culling* que 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 esta 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. Porém o *culling* nem sempre obtém o resultado esperado, algumas vezes, dependendo do objeto a ser redesenhado, o processo de analise das faces visíveis ao usuário acaba sendo mais lento do que o próprio processamento do objeto por completo, por este motivo, o *culling* foi deixado como opcional para o usuário escolher em utilizá-lo ou não.

Ao aplicar o *clipping* em ambientes abertos para evitar que objetos distantes sejam desenhados, é comum que o usuário acabe percebendo o recorte que ocorre no cenário, para evitar isto, pode ser utilizada uma técnica chamada de *fog*. O *fog* é um efeito de neblina muito utilizado em jogos, pois além de dar um efeito mais realista ao cenário ele também permite esconder do usuário o *clipping*. Com o aumento da densidade do *fog* também é possível diminuir esta distância em que ocorre o *clipping*, assim é possível aumentar consideravelmente a performance do jogo.

Para a criação de ambientes externos o *3D Game Builder* conta com o uso de *Heightmap,* que 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 12. 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 13 mostra uma superfície criada a partir da figura 12. Os *Heightmaps* normalmente são utilizados para a criação de

terrenos, pois possibilitam a formação de ambientes externos, como montanhas, de maneira rápida e fácil.

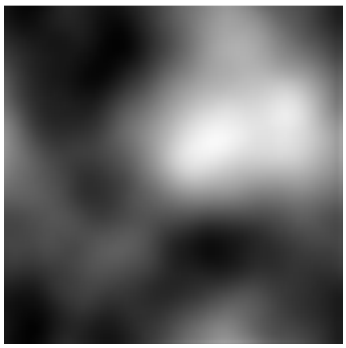

Figura 12: Imagem 2D
Fonte: O autor

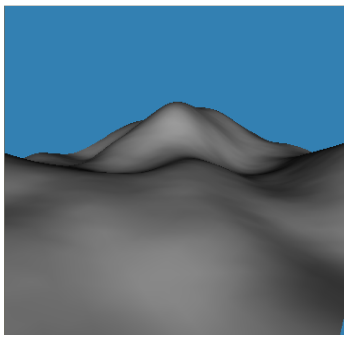

Figura 13: Heightmap 3D
Fonte: O autor

Outra técnica que pode ser usada para a criação de ambientes externos, onde é importante que este possua certa profundidade, é o *Skybox*, que consiste em um cubo, com 6 texturas diferentes posicionada e alinhadas em cada uma de suas faces internas. Todo o resto do cenário é criado no seu interior, assim o jogador tem a ilusão de que o ambiente é infinito.

O *3D Game Builder* oferece a possibilidade de geração de sombras de objetos, este é um efeito muito interessante para tornar o ambiente mais realista, pois a sombra é baseada nas fontes emissoras de luz do local. Para a criação destas sombras foi utilizada uma técnica conhecida como *projected shadows*, que 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 14.

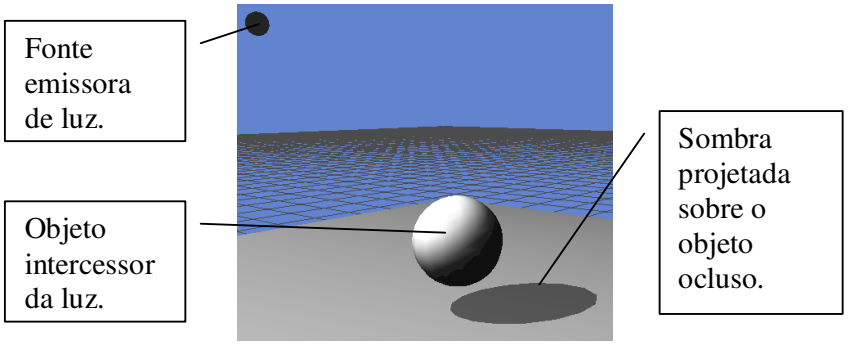

Figura 14: Sombra projetada
Fonte: O autor

Para a criação de efeitos especiais, muito usados em jogos, é possível utilizar o sistema de partículas, este termo refere se a 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 elas 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 15 é possível observar um efeito de fogo criado na ferramenta e na figura 16 um efeito de neve utilizando o mesmo sistema.

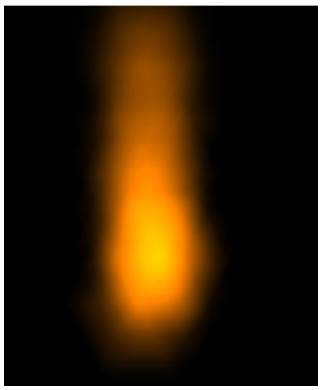

Figura 15: Simulação de fogo
Fonte: O autor

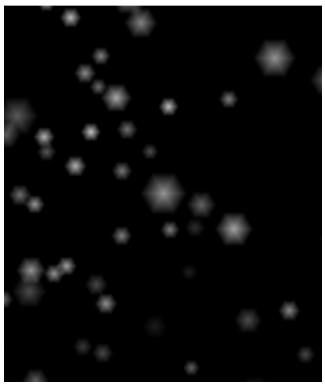

Figura 16: Simulação de neve
Fonte: O autor

Em um jogo além das funções gráficas também existem processos não visíveis como, por exemplo, a detecção de colisões, que é uma área muito importante para os jogos, pois sem ela um objeto poderia passar livremente sobre os outros. O processo consiste em determinar se, quando e onde dois objetos se interceptam. Segundo LIMA (2007) a implementação da detecção de colisão, normalmente é dividida em duas etapas, a primeira é verificar se a colisão ocorreu e a outra é qual a reação do objeto, como por exemplo, quando o jogador colide com uma parede, ele não poderá continuar se movimentando na sua direção, porém pode deslizar ao seu redor, outro tipo de reação poderia ser a movimentação do objeto colidido, tudo depende do sistema de colisão utilizado. No *3D Game Builder* todo este processo ocorre automaticamente, basta definir que o objeto será sólido e definir o tipo de física será aplicada a ele, caso seja um objeto estático ele não sofrerá nenhuma alteração quando um outro objeto colidir com ele, porém o outro objeto não passara por dentro dele, caso seja definido como um objeto dinâmico ele poderá ser movido por outros objetos, dependendo da força e da massa do objeto que estiver colidindo sobre ele.

A Inteligência Artificial também é considerada uma área muito importante em um jogo, é ela que possibilita aos inimigos pensar e agir o mais próximo possível de um ser humano. Existem diversas técnicas de inteligência artificial e a maioria delas baseia-se em analisar todas as possibilidades de eventos que possam ocorrer e determinar que ação será tomada perante determinada possibilidade. A criação de inteligência artificial no *3D Game Builder* é baseada em *scripts*, o próprio desenvolvedor pode criar o *script* que será utilizado durante o jogo.

Para a criação de eventos o *3D Game Builder* utiliza uma linguagem *script* interpretada baseada na linguagem *Object Pascal.* A linguagem *script* conta com estruturas de controle, estruturas de repetição, funções, procedimentos, classes, controle de variáveis, funções de manipulação de dados, operações matemáticas, entre outras funcionalidades. A manipulação das entidades 3D é feita através de funções, na figura 17 é mostrado um exemplo de *script* para controle de uma câmera secundária, onde inicialmente é declarada uma variável de nome x do tipo inteiro, no próximo passo o comando *setCamera* altera a câmera principal utilizada para exibir o ambiente para a *Camera2*, que havia sido criada anteriormente na construção do cenário, após isso é iniciado um *loop* indo de 0 à 100 e em cada interação a *Camera2* é transladada em mais 0.1 no seu eixo Z, após isto é chamada a função *DrawScene* que atualiza a tela, assim surge o efeito de uma câmera movimentando-se lentamente para frente. Os scripts são baseados em eventos, como por exemplo, a interação do usuário com um objeto do ambiente, aciona o evento de *OnInteract* associado ao objeto. Para facilitar a criação de eventos é possível utilizar a inserção de *scripts* automáticos.

```
var x:Integer;
setCamera('Camera2');
for x:=0 to 100 do
begin
  TranslateObject('Camera2', 0,0,0.1);
  DrawScene();
end;
```

Figura 17: Exemplo de script para controle uma câmera secundaria
Fonte: O autor

A união das características citadas acima e também de outras funcionalidades formam o *3D Game Builder*, nele todos os módulos necessários para a criação de um jogo, como a edição de mapas, criação de eventos, criação de atores, controle de colisões estão integrados em uma única ferramenta. Possibilitando ao desenvolvedor criar um cenário e ao mesmo tempo definir eventos aos objetos e podendo também visualizar como será o resultado final. A figura 18 mostra a tela principal do *3D Game Builder*, nela são criados os cenários, eventos, são adicionados objetos e efeitos especiais aos mapas. Porém, antes de um objeto externo, material, ou outro tipo de entidade ser utilizada em um mapa ela primeiramente deve ser importada para o projeto, para esta tarefa existem diversos cadastros, um para cada tipo de entidade, a figura 19 mostra o cadastro de materiais, ele engloba todas as características do material, sua textura e cores.

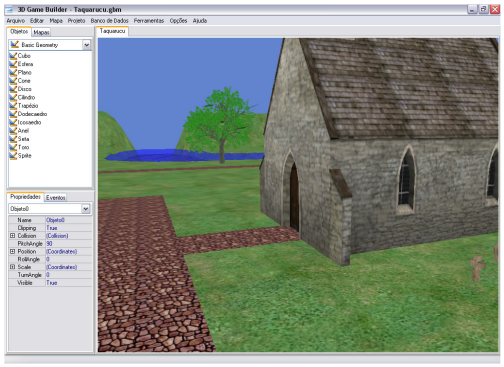

Figura 18: Edição de cenário
Fonte: O autor

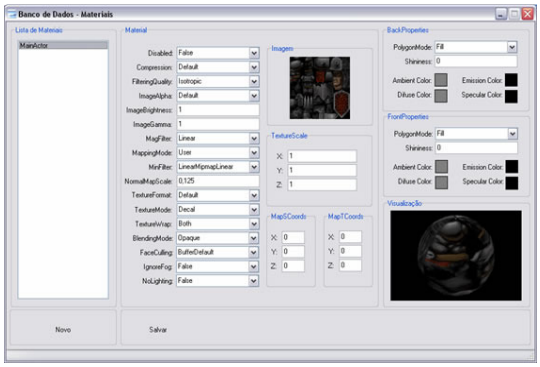

Figura 19: Edição de Materiais
Fonte: O autor

Para comprovar o funcionamento do *3D Game Builder* foi desenvolvido um jogo abordando o tema da Guerra do Contestado, um conflito armado que ocorreu em Santa Catarina durante os anos de 1912 a 1915. No jogo o jogador pode vivenciar e entender com mais clareza os fatos ocorridos na época, a figura 20 e 21 mostram cenas deste jogo. Também foi desenvolvido o controle da maquete de uma casa controlada pelo computador através da interação com os objetos 3D simulados no ambiente como portas, luzes e elevador, todas as interações feitas nestes objetos são simuladas igualmente na maquete, a figura 22 mostra o ambiente 3D que controla a maquete. Outros pequenos jogos e simulações também foram desenvolvidos, todos tiveram

ótimos resultados, entre eles está à simulação do planeta terra como pode ser visto na figura 23.

Figura 20: Jogo da Guerra do Contestado
Fonte: O autor

Figura 21: Jogo da Guerra do Contestado
Fonte: O autor

Figura 22: Casa Interativa
Fonte: O autor

Figura 23: Simulação do Planeta Terra
Fonte: O autor

Conclusão

O uso de uma g*ame engine* no desenvolvimento de jogos agiliza o processo e permite que pessoas com pouca experiência em programação possam criar os seus próprios jogos, este foi o objetivo do desenvolvimento do *3D Game Builder*, porém ele não ficou restrito somente a criação de jogos, pode ser utilizado para diversas tarefas, como para a simulação de ambientes 3D e também para a simulação de ambientes usando o conceito de "*walkthrough*" que consiste em um programa de computador que permite uma viagem virtual por uma simulação 3D de um ambiente do mundo real.

Como trabalhos futuros indicam-se a possível aplicação em escolas do jogo sobre a Guerra do Contestado desenvolvido com o *3D Game Builder*, o jogo pode servir como uma ferramenta de auxilio ao ensino da História do Contestado, pois conta de maneira interativa os principais acontecimentos da época. E também existem possibilidade de criação de outros jogos educativos ou simulações 3D utilizando o *3D Game Builder*.

Referências

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

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

UNIDEV. **Portal de Programação de Jogos.** Disponivel em: http://www.unidev.com.br. Acessado em: dez. 2007.

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

ASTLE, D.; HAWKINS, K. **Beginning OpenGL Game Programming.** Boston: Thomson, 2004.

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

ZERBST, S.; DÜVEL, O. **3D Game Engine Programming.** Boston: Thomson, 2004.

LIMA, S., E. **RPG Builder: Uma Ferramenta para o Desenvolvimento de Jogos.** Porto União, 2007.

---

[1] Artigo Científico financiado pelo Fundo de Apoio à Pesquisa – FAP, apoiado pela Universidade do Contestado – UnC, Campus de Canoinhas/Porto União – SC

[2] Acadêmico da 7ª fase do curso de Ciência da Computação da Universidade do Contestado, Campus de Canoinhas/Porto União – SC
[3] Professor orientador, Mestre em Ciência da Computação