Um sistema integrado de modelagem 3D com motor de jogo - MIRAGEM3D

Aluno: Antonio Prates
Orientador: Bruno Feijó

Introdução

Nesta pesquisa, foi feito um estudo transversal entre diversos motores de jogos 3D (3D *game engines*) [1] para visualização de ambientes 3D de projetos de engenharia, propondo-se o 3dsmax [2] como ambiente de modelagem. As tecnologias de jogos sem fins meramente lúdicos, mas também usados em educação e simulação, caracterizam o novo paradigma de *serious games* (jogos sérios ou empresariais).

Os motores de jogos existentes no mercado (*e.g.*: RenderWare, Unreal, 3D GameStudio, Crystal Space, etc.) são muito caros, inacessíveis ou, então, limitados no que diz respeito à importação de modelos 3D e à visualização de cenários em céu aberto. No presente trabalho, optou-se pela utilização de um *framework* mais genérico, porém mais adequado para superar as limitações acima citadas: Ogre3D [3]. Este *framework* é reconhecidamente um motor gráfico 3D de alto nível, com a vantagem de ser em código aberto e possuir licença GPL, além de ter sido o projeto do mês no SourceForge.net em março de 2005. O Ogre3D não é um motor de jogos propriamente dito, pois não implementa física nem outras funcionalidades utilizadas na confecção de jogos, porém é muito utilizado pela comunidade desenvolvedora de jogos, porque possui uma arquitetura organizada, bem documentada e totalmente orientada a objetos - o que facilita a sua integração e adaptação.

O Ogre3D possui um gerenciamento flexível para diversos tipos de cenário, inclusive para cenários grandes em céu aberto, e dispõe de uma série de projetos desenvolvidos pela comunidade de usuários (de código aberto e licença GPL), tais como: exportação de malhas em diversos ambientes de modelagem (3dsmax, maya, blender, etc.) e integração com bibliotecas de colisão e física (ODE, OPCODE, etc.).

Baseado neste motor de jogos, foi implementado um simulador, denominado Miragem3D, que obtém imagens de alta qualidade de cenários modelados no 3dsmax e possibilita a interação do usuário em tempo real, com vistas a aplicações em engenharia.

Objetivos

Este trabalho objetiva estudar e programar um sistema para visualização de cenários do 3dsmax, com interação em primeira pessoa, baseado em um motor de jogo de código aberto e licença GPL.

Metodologia

Para exportar as malhas no 3dsmax e carregá-las no ambiente do Ogre3D, destacam-se dois exportadores (*plugins* para 3dsmax escritos em MaxScript):

- Ogre3DSExporter – que, embora seja a solução oficial para o Ogre3D, não atende os objetivos, pois exporta apenas uma malha individualmente e não um cenário completo.

- Octopus Exporter – capaz de exportar múltiplas malhas e gerar um arquivo XML de referência (DotScene) para o cenário. Este exportador apresentava uma série de pequenas falhas que foram parcialmente resolvidas ao longo do desenvolvimento deste projeto.

Uma das possibilidades para o carregamento do cenário é a geração de um arquivo binário de Octree a partir do arquivo DotScene exportado do ambiente de modelagem. Entretanto, a implementação da biblioteca de Octree está com código desatualizado (desde 2002), o que a torna incompatível com as versões atualizadas do Ogre3D; não podendo, portanto, ser corrigida nem adequada para o uso neste projeto, por falta de documentação.

O Miragem3D implementa um *parser* para o arquivo XML (DotScene) gerado pelo Octopus, e um *loader* que utiliza o *framework* do Ogre3D para apresentar a cena ao usuário.

A aplicação-exemplo (Fig.1) utiliza um cenário do Cais do Porto do Rio de Janeiro (cujos arquivos 3dsmax foram cedidos pela empresa Nigraph) e um *Cubemap* (6 imagens JPEG que formam um cubo 360 graus com horizonte e nuvens) gerado no software *Terragen*.

Fig.1 Cais do Porto do Rio de Janeiro no Miragem3D

O posicionamento da câmera e a otimização para o carregamento de cenários grandes acontecem através de regras de nomeação de objetos no 3dsmax que servem de *trigers* para o *parser* de DotScene do Miragem3D. Um destes *trigers* é o nome de objeto "`__player`", que é um objeto criado na cena do arquivo 3dsmax para indicar onde a câmera deve "nascer". Outro *triger* permite o uso de instâncias de uma mesma malha: "`__instance|NOME_DA_MATRIZ|__|SEQUENCIAL`". Esse *triger* permite que o *loader* reutilize uma malha (matriz) no lugar de uma outra que seja semelhante, em tempo de execução.

Para que a navegação simule a movimentação de um veículo, a aplicação-exemplo implementa gravidade e trata colisão usando a biblioteca OPCODE. A integração desta biblioteca de colisão com Ogre3D é denominada OgreOpcode. O *loader* instancia as malhas do cenário na biblioteca OgreOpcode e a câmera colide com o cenário usando uma esfera envolvente (*bounding sphere*). A componente normal da colisão é praticamente anulada, fazendo com que a câmera "escorregue" no sentido tangencial à superfície do objeto colidido.

Arquitetura do Ogre3d

Este trabalho consiste de uma extensão da aplicação do Ogre3D, e como tal está inserido dentro do seu contexto de classes. A arquitetura adotada pelo Ogre utiliza vários conceitos de "Padrões de Projeto" (Factories, Singletons, Observers etc.), que a tornam mais enxuta e fácil de usar. Vários conceitos e abstrações são introduzidos pela arquitetura, sendo os principais: Root, SceneMananger, RenderSystem, Entity, Mesh, SceneNode e Material. O relacionamento entre estes componentes pode ser observado na Fig.2.

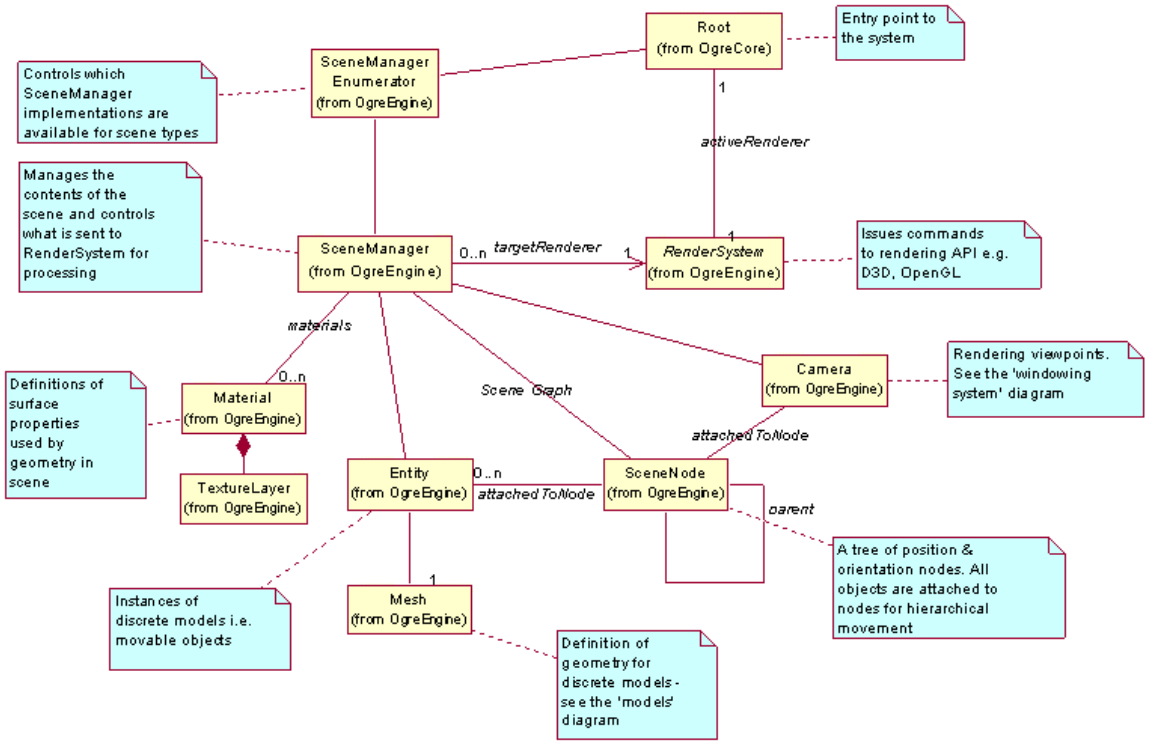


Fig.2 Visão UML do Ogre3D

O SceneManager é o objeto que gerencia os objetos que serão renderizados pelo motor. É responsável por aplicar algoritmos para otimizar a exibição da cena, por criar as Entidades (objetos, luzes, malhas), movê-las e transformá-las.

A maior parte da interação com o SceneMananger se dá ao longo da criação da cena. Durante este processo uma grande quantidade de objetos é criada, havendo a possibilidade de modificação dinâmica posteriormente durante o ciclo de renderização através de um FrameListener.

Histórico do desenvolvimento

➔ Desde o inicio até setembro de 2005 foram pesquisadas diversas possibilidades para solucionar o problema de exportar as malhas no 3dsmax e carregá-las no Ogre3D, testando para verificar qual apresentaria o melhor resultado:

- Ogre3DSExporter – não atendia, pois exportava apenas uma malha/objeto individualmente, precisávamos de uma solução capaz de lidar com cenas complexas.

- Octopus Exporter – atendia, mas apresentava uma série de pequenas falhas que foram parcialmente resolvidas.

OBS: Ambos consistem de plugins para 3dsmax escritos em MaxScript.

➔ Até final de outubro de 2005 o tempo foi investido sobre uma implementação de Dot_Scene_Octree, a qual estava com código (desde 2002) incompatível com a versão

atualizada do Ogre3D e não pode ser corrigida nem adequada para o uso, por falta de documentação.

➔ A partir de novembro de 2005 o projeto tomou um novo rumo e começou a implementação de um *parser* para o arquivo XML gerado pelo Octopus, tarefa que tomou quase um mês, mas que valeu à pena, pois em janeiro já era possível exportar uma cena no 3dsmax e visualiza-la através desta aplicação composta por uma parser e um loader executando dentro do framework do Ogre3d. Acredito que a maior parte dos resultados foi fruto do desenvolvimento que aconteceu neste período.

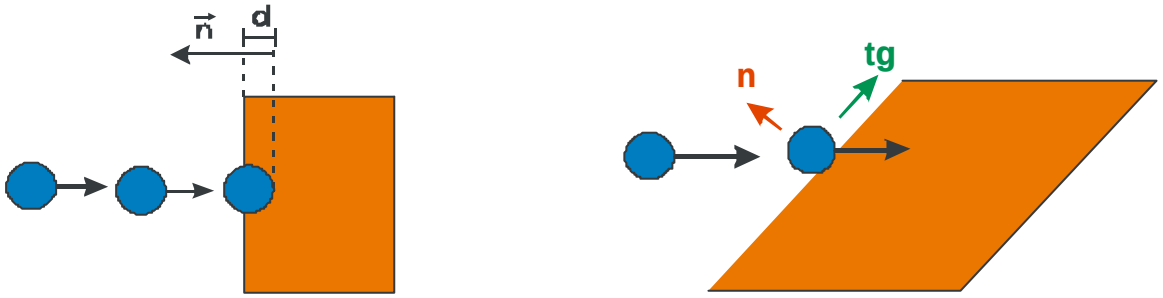
➔ Em janeiro de 2006 foram produzidos algumas aplicações exemplo com cenários do porto do Rio de Janeiro. Os originais de arquivos MAX foram produzidos e gentilmente cedidos pela empresa Nigraph. Nesta etapa também foi estudado e criado um *Cubemap* (6 imagens JPEG que formam um cubo 360 graus com horizonte e nuvens) gerado no software *Terragen* para acompanhar os exemplos.

➔ A partir de fevereiro de 2006 iniciou se a terceira etapa do projeto, que buscava otimizações para o load de cenários grandes e soluções para a sua navegabilidade como se um personagem humano caminhasse, ou veículo passeasse. Para isso, foram criadas regras de nomeação de objetos no 3dsmax que servem de trigers para o DotScene parser do Miragem3d.

➔ Deste março de 2006 até a presente data, vem se buscando aperfeiçoar o tratamento de colisão que é baseado na biblioteca OPCODE (também em código aberto e regida pela licença GPL).

Sistema de Colisão

Fig.3 e Fig.4 Vetores de movimento de uma colisão



Na implementação do Miragem3d existe a necessidade de realizar a correção da posição do objeto no sentido normal da colisão (a disância indicada pela letra ‘d’ na Fig.3), quando há penetração no objeto colidido. Após a correção desta distância penetrada, o restante da componente normal é anulada, o que faz com que a câmera “escorregue” no sentido tangencial à superfície do objeto colidido (vetor ‘tg’ da Fig.4).

Exportação de mapas à partir do 3dsmax

Utilizando o Octopus (http://www.ogre3d.org/wiki/index.php/ArtisticallyDisabled) para exportar os arquivos DotScene à partir do 3dsmax:

O Octopus instala um tipo de material denominado “OGRE (single/pass)”. Recomenda-se utilizar este tipo de material, apesar dos materiais do tipo Standard do 3dsmax funcionarem também. Recomenda-se também não utilizar os nomes default do 3dsmax, pois o exportador tem problemas com espaços em branco, sempre renomear, por exemplo: material_carro, material_arvore etc. Observação: Ao construir a modelagem do cenário, recomenda-se evitar triângulos muito grandes, pois reduzem o desempenho e qualidade da imagem em tempo real.

O Ogre3D lê texturas de arquivos PNG, JPEG, TGA, BMP ou DDS, incluindo formatos pouco usuais de textura como textura 1D, textura volumétrica, “cubemaps” e texturas comprimidas (DXT/S3TC). O Octopus Exporter consegue aproveitar as seguintes propriedades dos materiais:

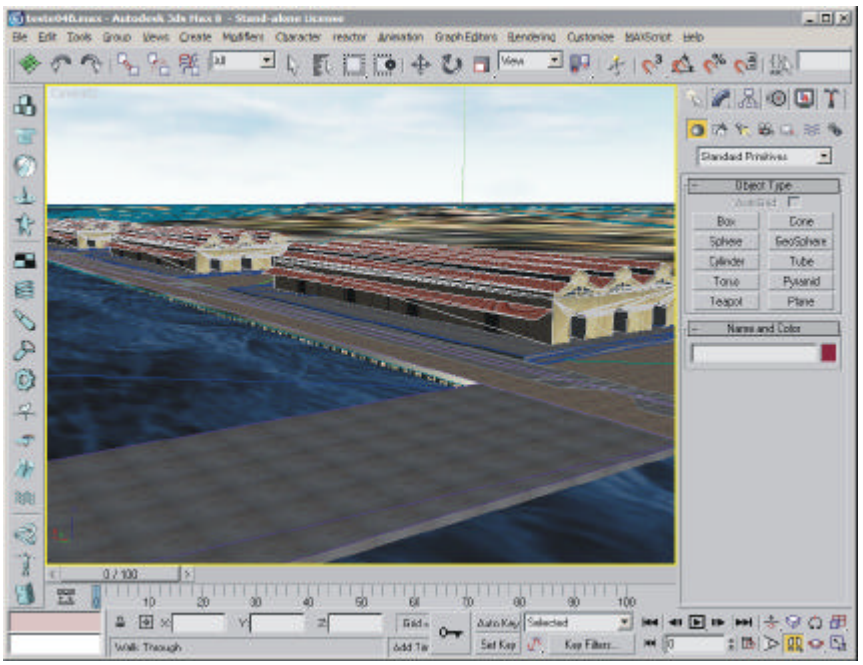
- Texture;
- Ambient;
- Diffuse;
- Specular;
- Emissive (Self Illumination).

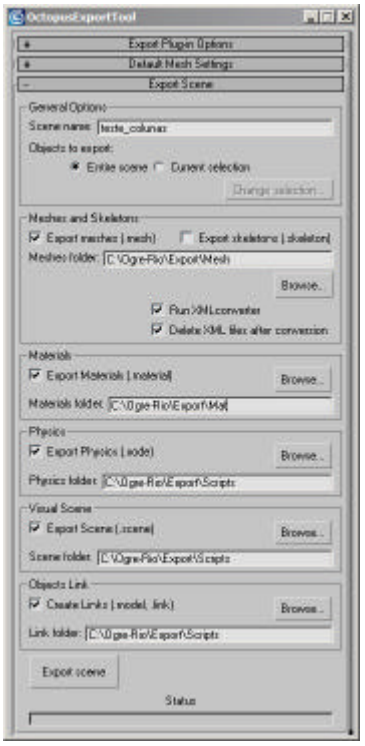
Problemas com o Octopus

Um problema não mencionando em nenhum local na Internet, mas que foi encontrado com certa freqüência nos testes realizados nesta pesquisa é o fato que, em algumas instalações do 3dsmax a cena exportada já tinha as transformações de posição, escala e rotação aplicados, sendo, portanto, a informação descrita no XML da cena redundante, e em outras era preciso utilizar esta informação do XML para obter a cena corretamente apresentada.

Para solucionar este problema foi criada uma opção no arquivo de configurações da aplicação Miragem3d (chave USE_SCENE_TRANSFORMATION mencionando mais à frente).

Fig.5 – Cais do Porto sendo editado no 3dsmax e Fig.6 – Octopus Exporter





Algoritmo do Loader da cena

Este é o algoritmo que utiliza o Parser para o formato .scene (xml) que carrega o Cenário direto no SceneMananger, permitindo a visualização instantânea da cena:

```
	void onTheFlyDotSceneRun(char * filepath)
	{
		multiParser * dotSceneParser = new multiParser(filepath);
		multiParser * NodeParser;
		dotSceneParser->writeLog("MIRAGEM3D está iniciando em: ");
		dotSceneParser->writeLog(filepath);
		dotSceneParser->writeLog(" \n");
		if(dotSceneParser->getType() == TFAIL)
		{
			dotSceneParser->writeLog("Falhou em abrir o arquivo! \n");
			return; // error
		}
		if(!dotSceneParser->parseDotScene()) return; // error
		if(!dotSceneParser->parseNodes()) return; // error

		// para cada elemento do arquivo XML
		while(dotSceneParser->getType() == TNODES)
		{
			NodeParser = dotSceneParser->getNextNode();
			if(dotSceneParser->getType() == TNODES)
			{
				// crie a Entidade!
				doNode(NodeParser);
			}
		}
		dotSceneParser->writeLog("onTheFlyDotSceneRun terminou. \n\n");
	}

	void doNode(multiParser * NodeParser)
	{
		Entity *e;
		SceneNode *node;
		char * subString;
		multiParser * StrParser;

		char name[MI3D_BUFSIZE], filename[MI3D_BUFSIZE];
		char pos1[MI3D_BUFSIZE], pos2[MI3D_BUFSIZE], pos3[MI3D_BUFSIZE];
		char rot1[MI3D_BUFSIZE], rot2[MI3D_BUFSIZE], rot3[MI3D_BUFSIZE],
rot4[MI3D_BUFSIZE];
		char sca1[MI3D_BUFSIZE], sca2[MI3D_BUFSIZE], sca3[MI3D_BUFSIZE];

		// le o nome
		strcpy(name, NodeParser->getNodeInfo(NULL, "name"));
		strcpy(filename, name);
		strcat(filename, ".mesh");

		// le a posicao
		strcpy(pos1, NodeParser->getNodeInfo("position", "x"));
		strcpy(pos2, NodeParser->getNodeInfo(NULL, "y"));
		strcpy(pos3, NodeParser->getNodeInfo(NULL, "z"));

		// le a rotacao
		strcpy(rot1, NodeParser->getNodeInfo("rotation", "qx"));
		strcpy(rot2, NodeParser->getNodeInfo(NULL, "qy"));
		strcpy(rot3, NodeParser->getNodeInfo(NULL, "qz"));
		strcpy(rot4, NodeParser->getNodeInfo(NULL, "qw"));

		// le a escala
		strcpy(sca1, NodeParser->getNodeInfo("scale", "x"));
		strcpy(sca2, NodeParser->getNodeInfo(NULL, "y"));
```

```
            strcpy(sca3, NodeParser->getNodeInfo(NULL, "z"));

            if(strcmp(name, "__player") == 0)
            {
                  // posiciona o player no cenario
                  mCamera->setPosition((Real)atof(pos1), (Real)atof(pos2),
(Real)atof(pos3));
            }
            else
            {
                  StrParser = new multiParser(name, 0, sizeof("__instance")-1);
                  subString = StrParser->getSource();
                  // verifica se eh instancia
                  if(strcmp(subString, "__instance") == 0)
                  {
                        delete StrParser;
                        StrParser = new multiParser(name, sizeof("__instance")-1,
sizeof(name)-1);
                        strcpy(filename, StrParser->getUntilUscores()); // usa o
nome do arquivo matriz
                        strcat(filename, ".mesh");
                  }
                  delete StrParser;
                  e = mSceneMgr->createEntity(name, filename);
                  node = mSceneMgr->getRootSceneNode()->createChildSceneNode();
                  node->attachObject(e);

                  CollisionShape *collideShape1 =
CollisionManager::getSingletonPtr()->NewShape(name);
                  collideShape1->Load(e);
                  collideShape1->SetName(name);
                  collideObject1 = collideContext->NewObject();
                  collideObject1->SetCollClass("level");
                  collideObject1->SetShape(collideShape1);
                  collideContext->AddObject(collideObject1);

                  if(strcmp(MI3D_USE_SCENE_TRANSFORMATION, "true") == 0)
                  {
                        // configura transformações do objeto - as vezes funciona
corretamente sem isso
                        node->setPosition((Real)atof(pos1), (Real)atof(pos2),
(Real)atof(pos3));
                        node->setOrientation(atof(rot4), atof(rot1), atof(rot2),
atof(rot3));
                        node->setScale(atof(sca1), atof(sca2), atof(sca3));
                  }
            }
      }
```

Configurações possíveis no Miragem3d (miragem3d.cfg)

Dot-scene-file define qual o arquivo que será carregado como cena inicial pelo Miragem3d.
Ex: DOT_SCENE_FILE="scenes/teste.scene"

Use-scene-transformation, dependendo da versão do Octopus ou 3dsmax é necessário setar *true* ou *false* esta opção para que os objetos sejam carregados apropriadamente na sua respectiva posição, rotação e escala.
Ex: USE_SCENE_TRANSFORMATION="false"

Use-collision define se o player irá colidir com os objetos ou se irá passar através deles.
Ex: USE_COLLISION="true"

Gravity define a aceleração da gravidade para o player.
Ex: GRAVITY="10"

Show-bounding-boxes define se a visualização irá apresentar os B-boxes.
Ex: SHOW_BOUNDING_BOXES="false"

Player-speed-max define a velocidade maxima que o player anda.
Ex: PLAYER_SPEED_MAX="500"

Player-speed-up define a velocidade o player acelera.
Ex: PLAYER_SPEED_UP="10"

Player-speed-down define o fator de redução da velocidade (em 0 funciona como uma espacionave – não tem desaceleração).
Ex: PLAYER_SPEED_DOWN="1"

Execução do Miragem3d

Para carregar mapas, malhas e cenários mais facilmente, sem ter que escrever um loader específico da cena inteiramente em C++ toda vez que se fosse criar algo novo, criei um pequeno parser que sabe carregar a cena tal qual ela é exportada pelo Octopus e logo em seguida carregar os objetos. Importante: Uma restrição que deve ser observada ao se exportar o cenário é que antes de exportar deve-se selecionar todos os *meshes* da cena e aplicar o transformador "XForm" do 3dsmax. Isto é necessário pois a implementação do Octopus não realiza a exportação corretamente das matrizes de transformação. Além disso, se mesmo seguindo esta orientação, você encontrar problemas ao visualizar o seu mapa, tente alterar o valor da chave USE_SCENE_TRANSFORMATION. Para incluir os arquivos necessários à visualização do cenário usando o Miragem3d, é necessário copiar os arquivos exportados pelo Octopus, descrito à seguir:

Matérias de: Mat*.material para: miragem3d\media\materials\scripts\
Imagens de: (pasta de texturas da cena) para: miragem3d\media\materials\textures\
Modelos de: Mesh*.mesh para: miragem3d\media\models\
Script de: Scripts\arquivo.scene para: miragem3d\scenes\arquivo.scene

Altere a chave DOT_SCENE_FILE para que o valor corresponda ao nome do seu arquivo de script ("arquivo.scene"). Depois basta executar: miragem3d\miragem3d.exe

Conclusões

A solução gerada neste trabalho viabiliza uma transposição bastante eficiente de cenários produzidos no 3dsmax para um ambiente de tempo real com visão de primeira pessoa.

O desempenho computacional do MIRAGEM3D é bastante razoável. Em um cenário com aproximadamente 2500 triângulos, executado em um processador Celeron 2.8Ghz, com placa gráfica 3D integrada de 32MB, o MIRAGEM3D apresenta uma taxa média de 50 fps (*frames per second*) e uma taxa mínima de 20 fps.

Para o futuro, pode-se melhorar o sistema, revendo o uso de Octree, aperfeiçoando o tratamento de colisão e implementando o uso das opções de sombra (*shadow casting*) do Ogre3D. O tratamento de sombras traria um maior realismo para a visualização da cena.

Referências

1 - ZERBST, S. and DUVEL, O. **3D Game Engine Programming**, Premier Press, 2004.

2 – AUTODESK, **3ds Max**, www.autodesk.com/3dsmax [acessado em 7/07/06].

3 – OGRE, **Ogre v.1.2**, http://www.ogre3d.org [acessado em 7/07/2006].

4 – Terragen, **Photorealistic scenery rendering software**, www.planetside.co.uk/terragen [acessado em 7/07/2006].