UNIVERSIDADE FEDERAL DE PELOTAS

Instituto de Física e Matemática
Departamento de Informática

Trabalho Acadêmico

UTILIZAÇÃO DE REALIDADE VIRTUAL NÃO IMERSIVA PARA A DEMONSTRAÇÃO DE TÉCNICAS DE COMPUTAÇÃO GRÁFICA

Pablo Duarte Pereira

Pelotas, 2008

PABLO DUARTE PEREIRA

UTILIZAÇÃO DE REALIDADE VIRTUAL NÃO IMERSIVA PARA A DEMONSTRAÇÃO DE TÉCNICAS DE COMPUTAÇÃO GRÁFICA

Trabalho acadêmico apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Lucas Ferrari de Oliveira

PELOTAS, 2008

Dados de catalogação na fonte:

Maria Beatriz Vaghetti Vieira – CRB-10/1032

Biblioteca de Ciência & Tecnologia - UFPel

P436u Pereira, Pablo Duarte

Utilização de realidade virtual não imersiva para a demonstração de técnicas de computação gráfica / Pablo Duarte Pereira ; orientador Lucas Ferrari de Oliveira. – Pelotas, 2008. – 71f. : il. - Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2008.

1.Informática. 2.Computação gráfica. 3. Ambiente virtual. 4.Técnicas de computação gráfica. 5.Comparação de engines. I.Oliveira, Lucas Ferrari de. II.Título.

CDD: 006.6

Banca Examinadora:

.....
Prof. Lucas Ferrari de Oliveira, Dr. (Orientador)

.....
Prof. Gerson Geraldo Homrich Cavalheiro, Dr.

.....
Prof. Leomar Soares da Rosa Junior, Dr.

Agradecimentos

Primeiramente, agradeço a Deus por me permitir chegar ao final desta importante etapa da vida.

Agradeço aos meus pais Eraldo e Angela, que me deram todo o suporte necessário, tanto emocional quanto financeiro para lutar até o fim dessa jornada. Vocês são mais do que especiais, pois sei que fizeram o impossível para conquistarmos esta vitória juntos.

Agradeço a minha irmã Samanta, que sempre tinha as palavras de incentivo certas nos momentos mais complicados.

Agradeço ao professor Lucas, meu competente orientador, pelos ensinamentos, incentivo e pela sabedoria compartilhada neste período.

Agradeço também a Letícia, minha grande companheira, pela força em momentos bem difíceis e pelo carinho com que me tratou nesses cinco anos juntos.

Agradeço a minha filha Brenda, que veio ao mundo durante o curso e me proporcionou momentos de extrema felicidade. Tu és o orgulho do papai.

Agradeço aos meus colegas de faculdade, que sempre estiveram dispostos a ajudar, compartilhar conhecimento e alegrias. Espero manter essas amizades por toda a vida.

E por fim, mas não menos importante, agradeço a todos os professores do curso Bacharelado em Ciência da Computação da UFPel, pois o esforço de vocês foi imprescindível para a minha formação.

Muitas pessoas devem a grandeza de suas vidas aos problemas e obstáculos que tiveram que vencer.

Autor Desconhecido.

Resumo

PEREIRA, Pablo Duarte. **Utilização de Realidade Virtual Não Imersiva Para a Demonstração de Técnicas de Computação Gráfica**. 2008. 71f. Trabalho acadêmico (Graduação) – Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

Sistemas de computação gráfica são desenvolvidos com diversos propósitos. Alguns destes ambientes possuem um aspecto voltado para a demonstração e simulação de técnicas dos mais variados campos. Além disso, estudantes de computação gráfica, muitas vezes, apresentam dificuldade em visualizar os ensinamentos ministrados nas disciplinas da área. Dessa forma, este trabalho busca desenvolver um ambiente tridimensional que demonstre algumas destas técnicas lecionadas em sala de aula. As funcionalidades selecionadas para serem inseridas nesta aplicação são modificações que afetam a renderização do cenário. Estas técnicas são divididas em dois grupos de interação: tonalização e iluminação. Para a criação deste sistema, um conjunto de *engines* livres, normalmente utilizadas no auxilio a desenvolvimento de jogos eletrônicos, foram comparadas. Sendo assim, foi selecionado o motor de jogo mais adequado para o desenvolvimento deste software. Os resultados obtidos foram animadores, pois o ambiente desenvolvido atingiu seu propósito de mostrar as diferentes técnicas de computação gráfica escolhida, porém uma série de novas funcionalidades podem ser incluídas no ambiente.

Palavras-chave: Computação gráfica. Ambiente virtual. Técnicas de computação gráfica. Comparação de *engines*.

Abstract

PEREIRA, Pablo Duarte. **Utilização de Realidade Virtual Não Imersiva Para a Demonstração de Técnicas de Computação Gráfica**. 2008. 71f. Trabalho acadêmico (Graduação) – Bacharelado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

Computer graphics systems are designed with different purposes. Some of these environments aim the demonstration and simulation of techniques of the most different areas. In addition, students in computer graphics often have difficulty viewing the lessons taught in the disciplines of the area. Thus, this study attempts to develop a three-dimensional environment that demonstrates some of these techniques taught in the classroom. The features selected to be included in this application are those that affect the rendering of the scene. These techniques are divided into two groups of interaction: shading and lighting. To set up this system, a set of free engines, normally used to aid the development of electronic games, were compared. So, it was selected the most suitable game engine for the development of this software. The results were encouraging, because the environment has reached its designed purpose to show the different techniques of computer graphics chosen, and a series of new features may be included in the environment.

Keywords: Computer graphics. Virtual environment. Techniques of computer graphics. Comparison engines.

Lista de Figuras

Figura 1 - Fluxograma das subáreas da Computação Gráfica. 19
Figura 2 - Figura de um dinossauro modelado no 3DS Max. 20
Figura 3 - Ilustração do modelo de renderização. 21
Figura 4 - Ilustração do processo físico de visualização dos objetos. 21
Figura 5 - Ilustração da técnica de *Scan Line*. 23
Figura 6 - Imagens de exemplos de RV imersiva e não imersiva..... 25
Figura 7 - Ilustração da estrutura modular de um motor de jogo. 27
Figura 8 - Imagens de telas de projetos desenvolvidos com Irrlicht. 31
Figura 9 - Diagrama da arquitetura do Ogre..... 34
Figura 10 - Imagens de telas de alguns projetos desenvolvidos com Ogre. 34
Figura 11 - Ilustração do cubo RGB. 36
Figura 12 - Ilustração do cone hexagonal HSV. 36
Figura 13 - Ilustração da reflexão das superfícies. 38
Figura 14 - Ângulo de incidência da luz..... 40
Figura 15 - Ângulo de reflexão especular é igual ao ângulo de incidência..... 41
Figura 16 - Reflexão da luz em superfícies especular e difusa. 42
Figura 17 - Processo de Tonalização *Gouraud*. 44
Figura 18 - Interpolação das normais ao longo das extremidades do polígono. 45
Figura 19 - Esferas tonalizadas com os métodos *Flat*, *Gouraud* e *Phong*..... 45
Figura 20 - Ambientes testes..... 47
Figura 21 - Ilustração da técnica de skybox. 48
Figura 22 - Tela que demonstra o cenário do ambiente CG House. 50
Figura 23 - Objetos utilizados no ambiente CG House..... 51
Figura 24 - Tela que demonstra o sistema interativo do ambiente CG House. 52
Figura 25 - Ambiente de interação. 53
Figura 26 - Menus de interação com o ambiente CG House..... 53
Figura 27 - Polygon Mode: Solid. 57
Figura 28 - Polygon Mode: Wireframe. 57

Figura 29 - Polygon Mode: Points. 58
Figura 30 - *Shading: Flat*. 58
Figura 31 - Shading: Gouraud. 59
Figura 32 - Shading: Phong. 59
Figura 33 - Reflection: Normal. 60
Figura 34 - Reflection: Diffuse. 60
Figura 35 - Reflection: Specular. 61
Figura 36 - Light Type: Point. 62
Figura 37 - Light Type: Spotlight. 62
Figura 38 - Light Type: Directional. 63
Figura 39 - Shadow Type: Texture Modulative. 63
Figura 40 - Shadow Type: Stencil Modulative. 64
Figura 41 - Shadow Type: Stencil Additive. 64
Figura 42 - Ambient Light: On. 65
Figura 43 - Ambient Light: Off. 65

Lista de Tabelas

Tabela 1 - Engines *open source* mais comentadas..... 46

Lista de Abreviaturas e Siglas

2D	Bidimensional
3D	Tridimensional
API	Application Programming Interface - Interface de Programação de Aplicativos
CAD	*Computer-Aided Design* - Desenho Auxiliado por Computador
CEGUI	*Crazy Eddie's Gui System* - Sistema de Interface do Eddie Louco
CG	Computação Gráfica
CSS	Cascading Style Sheets - Folhas de Estilos em Cascata
FPS	*First Person Shoter* - Atirador em Primeira Pessoa
GUI	*Graphical User Interface* - Interface Gráfica com o Usuário
HMD	*Head Mounted Display* - Dispositivo de visualização utilizado em realidade virtual.
HSV	*Hue, Saturation, Value* - Matiz, Saturação e Brilho
HTML	*HyperText Markup Language* - Linguagem de Marcação de Hipertexto
LGPL	*Lesser General Public License* - Licença GNU
ODE	*Open Dynamics Engine* - Motor de Física Aberto
OGRE	*Object-Oriented Graphics Rendering Engine* - Motor de Renderização Gráfica Orientada a Objetos
RGB	*Red, Green, Blue* - Vermelho, Verde e Azul
RV	Realidade Virtual
SDK	*Software Development Kit* - Kit de Desenvolvimento de Software

Sumário

1 INTRODUÇÃO 15
1.1 Motivação 16
1.2 Objetivos 17
1.3 Organização do trabalho 17

2 COMPUTAÇÃO GRÁFICA E REALIDADE VIRTUAL 18
2.1 Computação Gráfica 18
2.1.1 Modelagem 19
2.1.2 Renderização 20
2.2 Realidade Virtual 23
2.2.1 RV imersiva e não imersiva 24

3 ENGINES 26
3. 1 Engines proprietárias 27
3.2 Engines livres 28
3.2.1 Panda 3D 28
3.2.2 Crystal Space 28
3.2.3 Irrlicht 29
3.2.4 Ogre 3D 31

4 CORES, ILUMINAÇÃO E TONALIZAÇÃO 35
4.1 Cores 35
4.2 Iluminação 37
4.2.1 Fontes de Luz 37
4.2.2 Modelos de Iluminação 38
4.3 Tonalização 42
4.3.1 *Flat Shading* 43

4.3.2 *Gouraud Shading* **44**
4.3.3 *Phong Shading* **44**

5 METODOLOGIA **46**
5.1 Irrlicht X Ogre **47**
5.2 Ambiente Final **49**
5.2.1 Construção do cenário **50**
5.2.2 Objetos **51**
5.2.3 Modelo de interação **51**

6 RESULTADOS **56**
6.1 Tonalização **56**
6.1.1 *polygon_mode* **56**
6.1.2 *shading* **58**
6.1.3 *reflection* **59**
6.2 Iluminação **61**
6.2.1 *light_type* **61**
6.2.2 *shadow_type* **63**
6.2.3 *ambient light* **64**

7 CONCLUSÕES E DISCUSSÃO **66**
7.1 Funcionalidades bem sucedidas **66**
7.2 Problemas encontrados **67**
7.3 Trabalhos futuros **68**

Referências **70**

1 INTRODUÇÃO

Em busca de modelos tridimensionais que representem os ambientes reais de forma mais fiel, a construção de ambientes virtuais vem desenvolvendo-se rapidamente, utilizando técnicas da Computação Gráfica (CG) cada vez mais sofisticadas e algoritmos mais aprimorados. Desta forma, comunidades de desenvolvedores buscam implementar e refinar seus códigos, de forma que seja possível simular objetos do mundo real. Para tornar isso possível e agilizar o processo de criação, empresas e grupos de desenvolvedores constroem as chamadas *engines* 3D.

Uma *engine* 3D é um conjunto de funções que são especializadas no desenvolvimento de aplicações gráficas, muito utilizada na indústria de jogos, por facilitar a manipulação de uma série de requisitos na concepção deles, entre os quais, destaca-se: renderização, cálculos físicos, detecção de colisão, utilização de som, algoritmos de IA, redes, etc.

Este trabalho realiza, em um primeiro momento, um estudo que visa escolher a melhor *engine* livre e sem custo para ser utilizada no desenvolvimento de um ambiente tridimensional de simulação, que possibilite a demonstração de técnicas de Computação Gráfica.

Para a escolha da *engine* apropriada, quatro de grande utilização foram pré-selecionadas, após uma pesquisa em fóruns de discussão do assunto e sites relacionados.

Sendo que destas, duas *engines* se destacaram para o desenvolvimento de jogos e ambientes virtuais como o desenvolvido no trabalho: Irrlicht e Ogre3D. Elas obtiveram uma comparação mais detalhada, com o desenvolvimento de ambientes 3D básicos em ambas. Proporcionando assim, naturalmente, uma comparação mais realista, já que além da análise teórica, foi realizada uma análise prática.

Em decorrência da análise das plataformas de desenvolvimento e dos sistemas de teste em ambas as *engines*, há também a implementação de um ambiente final, que cria alguns objetos tridimensionais clássicos da computação gráfica.

Utilizando a *engine* escolhida, o software busca demonstrar na prática o que ocorre com os objetos, quando submetidos a técnicas estudadas em CG. O sistema aplica estas modificações nos objetos através de uma Interface Gráfica do Usuário (GUI). Permitindo assim, diversas interações com o ambiente.

1.1 Motivação

A necessidade de uma apresentação visual das técnicas dos mais diversos conteúdos discutidos em sala de aula na área de computação gráfica é bastante evidente, pois a era dos computadores e do vídeo-game faz dos alunos usuários práticos, porém com pouco conhecimento da teoria envolvida.

Levando-se em consideração os conteúdos relacionados à computação gráfica, esta necessidade torna-se ainda mais acentuada. Sendo que, na maioria das vezes, usuários iniciantes no estudo da computação gráfica carregam um ensinamento proeminente dos jogos eletrônicos. Onde se tem contato com os mais avançados sistemas tridimensionais e incríveis de realidade virtual não imersiva.

Cientes da importância, alguns ambientes tridimensionais de demonstração podem ser desenvolvidos para trazer para a tela do computador a visualização da aplicação das técnicas de computação gráfica estudadas em sala de aula.

Com a utilização destes ambientes, possibilitando a interação do usuário com o sistema para a geração dos resultados, maximiza-se a possibilidade de interesse e aprendizado deste usuário. Com esta certeza, este trabalho busca construir este ambiente que permita a interação dos usuários e visualização imediata das modificações nas propriedades de iluminação e tonalização.

É importante ressaltar que o sistema desenvolvido será disponibilizado como software livre. Portanto, pode ser utilizado, modificado e redistribuído sem maiores restrições.

1.2 Objetivos

O objetivo do trabalho foi realizar a comparação entre as *engines Irrlicht* e a *Ogre3D*, selecionando a mais adequada para o desenvolvimento de um ambiente de realidade virtual não imersiva, que permitiu a demonstração de conceitos da computação gráfica referentes a tonalização e a iluminação.

O sistema deve propiciar ao usuário:

- Interagir com o ambiente tridimensional utilizando as entradas típicas dos computadores pessoais como mouse e teclado, além de receber a resposta visual através do monitor;
- Visualizar o ambiente com o realismo necessário, representando os objetos que compõe a cena, através de modelagem 3D ou imagens 2D na criação de texturas dos objetos;
- Receber do sistema uma resposta rápida o bastante para tornar a experiência do usuário perante o sistema o mais agradável possível, sempre considerando as limitações de hardware e software.

1.3 Organização do trabalho

Este trabalho é dividido em seis capítulos onde apresenta alguns conceitos inicias necessários para o bom entendimento, partindo para análise e comparação entre as *engines* e acaba demonstrando como foi implementado o ambiente.

No segundo capítulo, apresentam-se os principais conceitos associados à computação gráfica a realidade virtual.

No terceiro capítulo, realiza-se a conceituação e analise do que diz respeito às *engines* e renderização. Explicando suas principais características.

No quarto capítulo, descrevem-se os fundamentos mais específicos presentes na renderização, tratando de aspectos de cores, iluminação e tonalização.

No quinto capítulo, demonstra-se toda a metodologia adotada no desenvolvimento do ambiente, desde o sistema de testes para escolha da *engine*, até a descrição das etapas de desenvolvimento do ambiente final.

No sexto capítulo, apresentam-se os resultados encontrados pelo sistema, demonstrando todas as suas funcionalidades.

No sétimo capítulo, há uma discussão a respeito dos resultados encontrados, a conclusão de alguns aspectos e os possíveis trabalhos futuros.

2 COMPUTAÇÃO GRÁFICA E REALIDADE VIRTUAL

2.1 Computação Gráfica

Existem diversos conceitos um pouco diferenciados sobre Computação Gráfica (CG): um deles refere-se a ela como a área da ciência da computação que estuda a geração, manipulação e interpretação de modelos e imagens de objetos utilizando o computador. Tais modelos vêm de uma variedade de disciplinas, como física, matemática, engenharia, arquitetura, etc. (TRAINA; OLIVEIRA, 2006).

Pode-se dizer que CG também é o conjunto de algoritmos, técnicas e metodologias para o tratamento e a representação gráfica de informações através da criação, armazenamento e manipulação de desenhos, utilizando-se computadores e periféricos gráficos (BATTAIOLA, 1999; FOLEY, 1993).

Ela é vista também, como a área da computação destinada à geração de imagens, em forma de representação de dados e informação, ou em forma de recriação do mundo real. Porém, pode possuir uma infinidade de aplicações para as mais diversas áreas. Desde a própria informática ao produzir interfaces gráficas para softwares, sistemas operacionais e sites na internet, quanto para produzir animações e jogos (Wikipedia CG, 2008).

No entanto, esta variedade de conceitos, gira em torno de uma mesma idéia: de que depende da aplicação e do contexto ao qual estamos inseridos. Profissionais que utilizam Computação Gráfica na publicidade referem-se a ela de forma diferente dos desenvolvedores de jogos, por exemplo.

Foley et al. (1993) acrescenta que a Computação Gráfica é utilizada em muitas áreas da indústria, negócios, governo, educação e entretenimento. A lista de aplicações é enorme, abrange itens como: interface do usuário, cartografia, medicina, sistemas multimídia, ferramentas CAD, entretenimento e etc.

A computação gráfica pode ser classificada em três importantes áreas interligadas, como demonstra a Fig. 1 (TRAINA; OLIVEIRA, 2006). Estas áreas seguem um ciclo, onde a divisão dá-se da seguinte forma:

- Síntese de imagens: representação visual através de representações geométricas;
- Processamento de imagens: Transformações de imagens através de representação visual (matricial);
- Análise de imagens: especificações dos componentes de uma imagem a partir de sua representação visual.

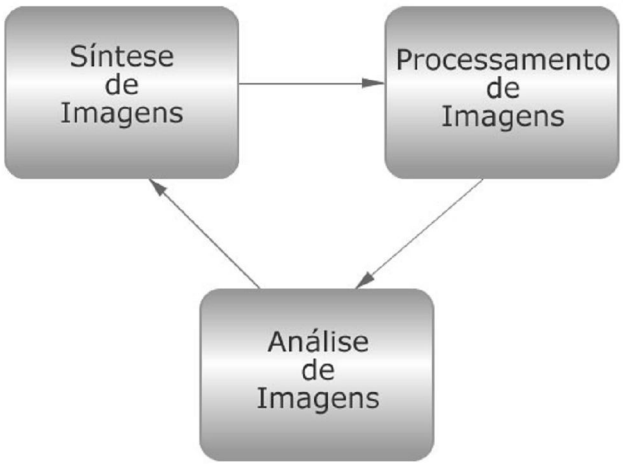


Figura 1 - Fluxograma das subáreas da Computação Gráfica.

Logo, a Computação Gráfica pode tratar da síntese de imagens no computador, o que envolve as etapas de modelagem e de renderização (TRAINA; OLIVEIRA, 2006).

2.1.1 Modelagem

A coleção de métodos usados para descrever a forma e outras características geométricas de um objeto ficou conhecida por modelagem geométrica. Tais métodos são uma síntese de outros campos: geometria analítica e descritiva, topologia, cálculo vetorial, análise numérica, entre outros (FILHO, 1998).

Há diversas formas de modelagem 3D. Entre elas, a vasta área da modelagem de superfícies. As superfícies de malhas de polígonos estão entre as três representações mais comuns de superfícies tridimensionais (FOLEY et al., 1993).

Com a utilização de softwares de modelagem 3D, os objetos são criados através de grandes malhas de polígonos e podem ser anexados a sistemas computacionais de CG.

Foley (1993) explica que uma malha de polígonos é uma coleção de arestas, vértices e polígonos conectados de forma que cada aresta é compartilhada por no máximo dois polígonos. Uma aresta conecta dois vértices e um polígono é uma seqüência fechada de arestas.

A Fig. 2 demonstra um objeto em *wireframe* modelado no software 3D Studio Max.

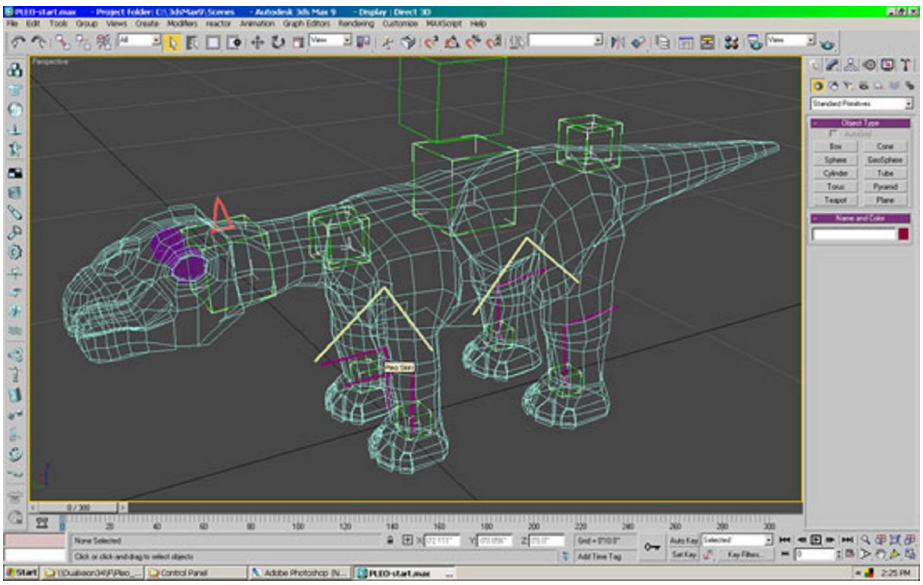


Figura 2 - Figura de um dinossauro modelado no 3DS Max.
Fonte: ZDNET, 2008.

2.1.2 Renderização

Renderização é a utilização de técnicas para geração de imagens matriciais de cenas tridimensionais. Para sintetizar essas imagens é necessário definir os objetos que compõe a cena (TRAINA; OLIVEIRA, 2006).

Há dois grandes ramos de estudo na Computação Gráfica: gráficos bidimensionais e tridimensionais. Porém, eles aparecem relacionados quando imagens 3D são renderizadas para serem visualizadas em uma saída bidimensional, tornando-se assim imagens 2D (SANTOS, 2004), como mostra a Fig. 3.

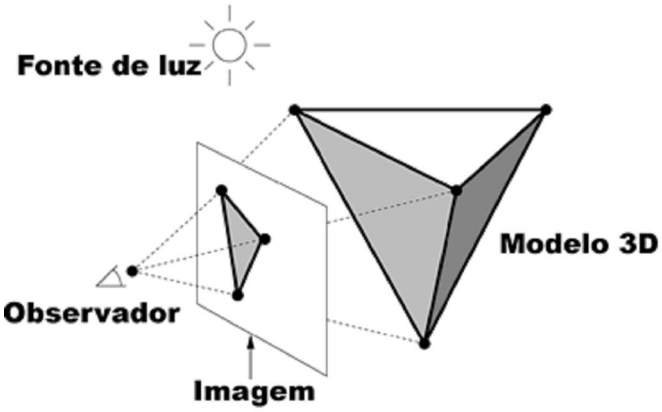


Figura 3 - Ilustração do modelo de renderização.

Uma série de fatores deve ser analisada pelos algoritmos de renderização para concepção de uma cena. Além da própria geometria dos objetos desta cena, é necessário levar em consideração informações como iluminação, texturas, cores e câmera.

A Fig. 4 mostra o processo físico do olho humano na visualização dos objetos. Funciona da seguinte forma: há uma fonte que emite raios de luz em todas as direções. Alguns destes raios atingem o objeto que estamos observando. A superfície deste objeto absorve alguns raios e reflete outros. O objeto é visto quando parte da luz refletida atingir nossos olhos.

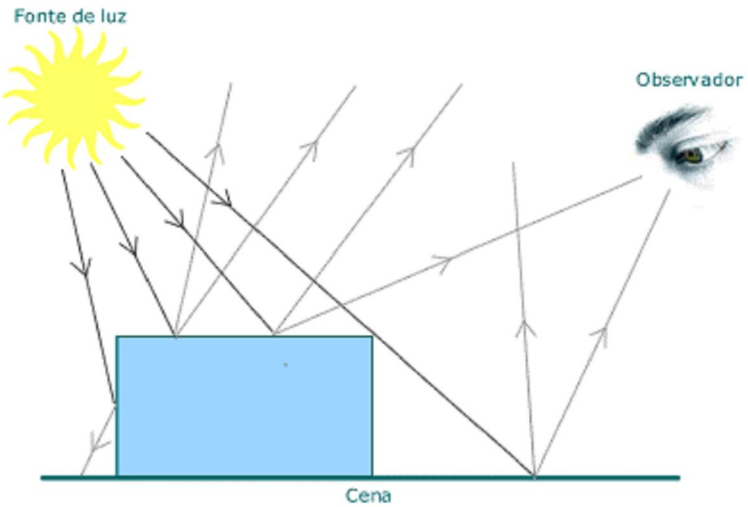


Figura 4 - Ilustração do processo físico de visualização dos objetos.

Uma técnica comumente utilizada em Computação Gráfica é o *ray-tracing* (emissão de raios). Ele simula a interação dos raios de luz com o ambiente, seguindo o caminho percorrido por cada raio. Mas, simular todos os raios que partem da fonte de luz torna-se totalmente inviável. Por isso se faz o processo contrário, simulando apenas os raios que chegam até olho do observador,

determinando quais objetos o raio atingiu desde que partiu da fonte de luz (TRAINA; OLIVEIRA, 2006).

O objetivo é lançar pelo menos um raio para cada pixel que compõe a imagem e traçá-lo de volta até que ele intercepte uma fonte de luz. O *ray-tracing* gera imagens foto-realistas, mas para isso necessita de um alto custo computacional.

O *ray-tracing* é um método do tipo *image-order*, que funciona tentando determinar o que acontece para cada raio de luz, um de cada vez. Um processo do tipo *object-order* funciona renderizando os objetos que compõem a cena. No exemplo acima, um processo deste tipo tentaria renderizar o chão, e depois o cubo. Este tipo de processo pode ser agilizado com sua implementação em um hardware dedicado (FOLEY, 1993).

Observe que na renderização dos objetos, apenas é considerado a interação de suas superfícies com os raios de luz, sendo que o interior dos objetos não é visualizado. Esse tipo de renderização é denominado *surface rendering*. Entretanto, esta estratégia não é adequada para renderizar objetos como nuvens, água e neblina, que são translúcidos e deixam a luz passar por eles. Nesse caso, utiliza-se outra estratégia denominada *volume rendering*, que considera a interação da luz com o interior dos objetos.

Scan Line

De acordo com Hearn e Baker (1997), o algoritmo *scan line* é utilizado para preenchimento e tonalização dos objetos. Há uma técnica utilizada para preenchimento de polígonos de forma arbitrária, mesmo que eles tenham auto-intersecção e buracos em seu interior.

O funcionamento deste algoritmo é mostrado na Fig. 5. Ele opera computando blocos de pixels que estão entre os lados esquerdo e direito do polígono. O extremo do bloco é calculado por um algoritmo incremental que calcula a linha de varredura a partir da intersecção com a linha de varredura anterior.

É preciso determinar quais dos pixels da linha de varredura estão dentro do polígono, e estabelecer os pixels correspondentes (no exemplo da Fig. 5, blocos para x=2 até 4 e 9 até 13) com seu valor apropriado. Repetindo este processo para cada linha de varredura, o polígono inteiro é convertido.

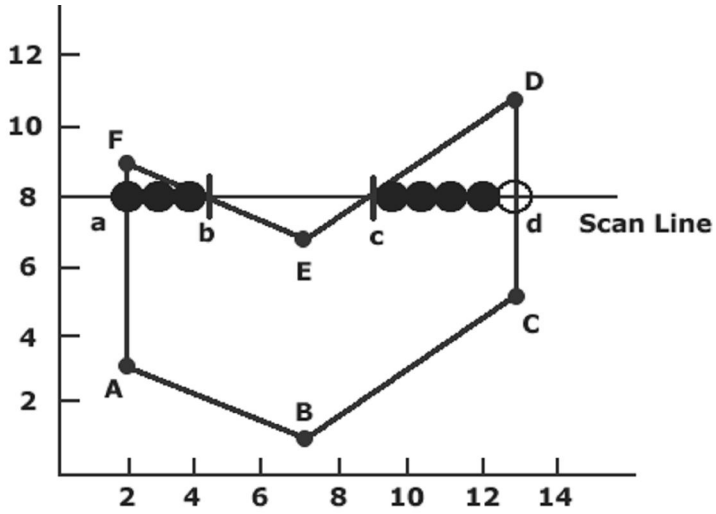


Figura 5 - Ilustração da técnica de *Scan Line*.

Outras etapas do processo de renderização serão apresentadas mais detalhadamente no capítulo 4.

2.2 Realidade Virtual

O termo Realidade Virtual (RV) é utilizado por várias pessoas com muitos significados. Para alguns, RV é uma coleção de tecnologias específicas, ou seja, um dispositivo de visualização chamado HMD (*Head Mounted Display*), um dispositivo de entrada (Luvas de RV) e de áudio, por exemplo. Porém, outras pessoas estendem o termo para incluir livros convencionais, filmes ou pura fantasia e imaginação (ISDALE, 1998).

Um dos conceitos mais aceito é o de que Realidade Virtual é uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos (AUKSTAKALNIS; BLATNER, 1992).

Ela é tratada também, como uma “interface avançada do usuário” para acessar aplicações executadas no computador, propiciando a visualização, movimentação e interação do usuário, em tempo real, em ambientes tridimensionais gerados por computador (KIRNER; SISCOUTTO, 2007).

Esta técnica computacional é utilizada para desenvolver ambientes artificiais que permitam aos usuários ter a sensação de estar dentro dos ambientes, isto é, sentir que esta realidade artificial realmente existe. Sistemas baseados nesta técnica utilizam recursos gráficos em três dimensões, facilitando em tempo real a interatividade entre um ou mais usuários e entre um ou mais sistemas computacionais (KIRNER; TORI, 2004).

Para Isdale (1998), a visualização refere-se ao computador gerando saídas visual, auditiva ou outras para o usuário de um mundo dentro do computador. Este mundo pode ser um modelo CAD, uma simulação científica, ou uma visão em um banco de dados. O usuário pode interagir diretamente com o mundo e manipular objetos nele. Alguns mundos são animados por outros processos, talvez simulações físicas, ou animações simples. O autor considera uma característica crucial para verificar se um ambiente é “realidade virtual”, a interação com o mundo virtual em tempo quase real do controle do ponto de vista.

Alguns conceitos, já haviam sido introduzidos em 1965 por Sutherland (SUTHERLAND, 1965). Conhecido posteriormente como o precursor da Realidade Virtual, ele dizia que “a pessoa deve olhar a tela (do computador) como sendo uma janela através da qual se vê o mundo virtual. O desafio da computação gráfica é fazer a imagem na janela parecer real, soar real e os objetos agirem de forma real”.

O sentido da visão costuma ser preponderante em aplicações de realidade virtual, mas os outros sentidos, como tato, audição e etc. também podem ser usados para enriquecer a experiência do usuário (KIRNER; SISCOUTTO, 2007). De acordo com o nível de imersão, a realidade virtual se divide em dois segmentos.

2.2.1 RV imersiva e não imersiva

Existem duas formas de utilizar RV, a imersiva e a não imersiva. Modelos imersivos são construídos para utilizarem interfaces especiais. Geralmente, essas interfaces são relacionadas à visão, à audição, ao olfato e ao tato, além de possibilitarem a detecção do movimento do usuário. Neste tipo de interface os componentes são acoplados diretamente ao corpo do usuário.

Para a visualização de ambientes imersivos têm-se óculos 3D e capacetes de RV (HMD). Sensores em luvas fazem a captação dos movimentos da mão do usuário e, dependendo da ação a ser realizada, simulam o impacto ou o contato da mão com um objeto existente no mundo virtual. Por fim, a detecção de movimentos do corpo é feita por sensores instalados no equipamento de imersão usado pelo usuário, como, por exemplo, uma CAVE, ou através de rastreadores presos na cabeça do usuário, para captação dos movimentos da cabeça (KELNER et al., 2002).

Evidente que com estes equipamentos o sentimento de realidade do usuário maximiza, porém o custo ainda é alto e torna-se inviável a sua utilização.

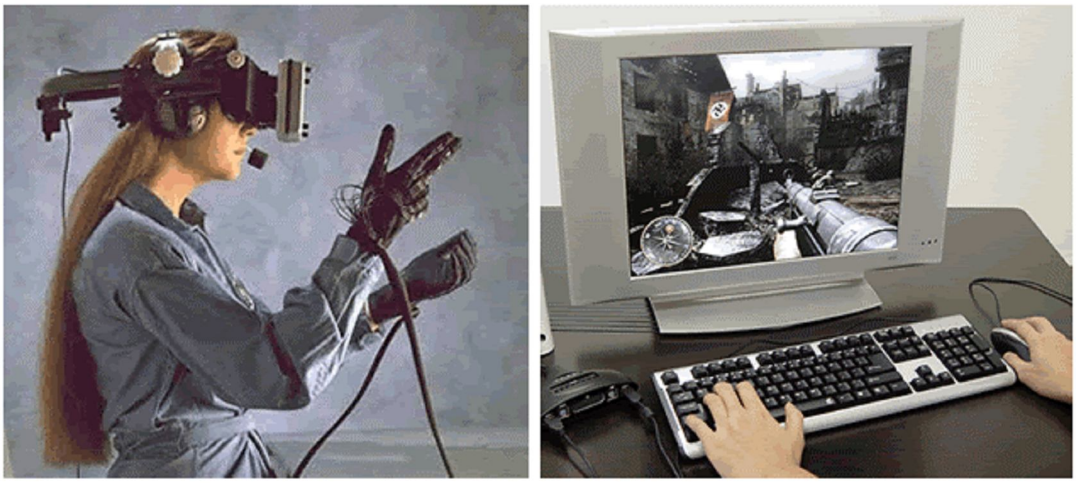
Já a RV não imersiva (RVNI), utiliza apenas o monitor de um computador *desktop* como janela para o mundo virtual. Pelo fato do usuário continuar mantendo contato com o mundo real que o cerca enquanto interage com um mundo virtual não imersivo, diz-se que RVNI oferece um nível de imersão baixo. A grande vantagem deste tipo de RV se dá pela forma como é implementada: ambientes virtuais não imersivos podem ser facilmente utilizados por qualquer usuário doméstico que possua um computador pessoal com desempenho razoável e com um software (gratuito) adequado instalado (KELNER et. al., 2002).

Os jogos de computador de hoje, para citar um exemplo, se utilizam largamente de tecnologia de RV não imersiva, mas alguns autores acreditam que já não são conhecidos como tal (KIRNER; TORI, 2004).

De acordo com Netto (1998), são características necessárias a Realidade Virtual:

- Foco no usuário;
- Renderização das imagens em tempo real;
- Imersão: de algum nível;
- Interação: usuário pode influenciar o comportamento dos objetos;
- Interface intuitiva: pouca ou nenhuma dificuldade em manipular as interfaces computacionais entre o usuário e a máquina;
- Funcionalidade: descrição funcional dos objetos, não apenas geometria, topologia e aparência.

A Fig. 6 compara imagens de RV imersiva (a) e não imersiva (b).



(a) RV Imersiva (b) RV Não Imersiva

Figura 6 - Imagens de exemplos de RV imersiva e não imersiva.
Fonte: WIKIPEDIA VR, 2008 (adaptada).

3 ENGINES

Conhecido popularmente pelo termo em inglês *game engine*, motor de jogo, *framework* para jogos ou simplesmente *engine*, o motor de um jogo é o seu sistema de controle, o mecanismo que controla a reação do jogo em função das ações dos usuários. A implementação de um motor envolve diversos aspectos computacionais, tais como, a escolha apropriada da linguagem de programação em função da sua facilidade de uso, eficiência e portabilidade; o desenvolvimento de algoritmos específicos, serviços para a gerência e renderização de cenários, controle de personagens e de mundos, gerência de janelas das aplicações, gerência de sons, suporte a rotinas matemáticas, estruturação e classificação de dados, mecanismos de comunicação, sincronização, modelo de interface com o usuário, dentre outros (MADEIRA, 2001).

Em geral, porém, o conceito de um motor de jogo é bastante simples: ele existe para abstrair os (por vezes dependentes de plataforma) detalhes de fazer tarefas comuns relacionadas com o jogo, como renderização, física, e de entrada, de modo que desenvolvedores (artistas, *designers*, *scripters* e até mesmo outros programadores) possam dar maior atenção aos detalhes que tornam os seus jogos exclusivos (WARD, 2008).

Por se utilizar largamente nos jogos eletrônicos, os conceitos de *engine* confundem-se com algumas características dos jogos. Sendo assim, para melhor visualização, utiliza-se da analogia com os *games*.

Nos jogos de hoje, construídos modularmente, o motor de jogo se refere à coleção de módulos de código de simulação, que não especificam diretamente o comportamento do jogo (lógica do jogo) ou características do jogo (nível de dados) (LEWIS; JACOBSON, 2002).

A Fig. 7 mostra a estrutura modular de um jogo. Observa-se que a *engine* localiza-se logo acima da camada dos *drivers* gráficos e abaixo do código do jogo propriamente dito.

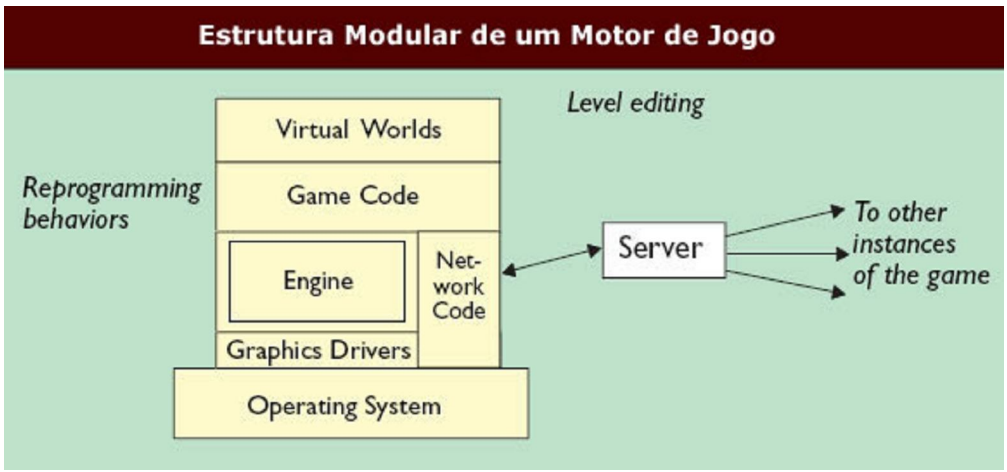


Figura 7 - Ilustração da estrutura modular de um motor de jogo.
Fonte: LEWIS; JACOBSON, 2002.

Com este conjunto de bibliotecas é possível desenvolver jogos, visualizações de arquitetura, simulações ou qualquer aplicação que necessite de uma solução para renderização (NETTO; MACHADO; MORAES; 2006).

Existem um número muito grande de *engines*, algumas proprietárias e outras livres. As mais utilizadas e conhecidas geralmente estão entre as *engines* proprietárias. Porém, há também *engines* livres de qualidade e bastante utilizadas.

3. 1 Engines proprietárias

A grande maioria dos famosos jogos eletrônicos para computador são produzidos utilizando motores desenvolvidos por grandes empresas. As quais, naturalmente, visam o lucro e vendem a licença para utilização em jogos ou qualquer outra finalidade.

Entre algumas das *engines* proprietárias muito utilizadas estão: *3D GameStudio*, *Torque* e *Game Maker*. Além de algumas *engines* desenvolvidas para jogos famosos que são comercializadas para o desenvolvimento de outros jogos, como as *engines* dos jogos *Quake 3*, *Doom 3* e *Unreal*.

3.2 Engines livres

Muitas das *engines* livres são desenvolvidas em meio acadêmico. Grande parte delas, com alguma finalidade científica. Onde não é raro encontrar bons sistemas sub-utilizados em pouquíssimos projetos, ou até mesmo nunca utilizados.

Por outro lado, algumas *engines* destacam-se e conquistam a escolha de muitos desenvolvedores. Entre elas, foram selecionadas quatro boas e importantes *engines* livres: *Panda 3D*, *Crystal Space*, *Irrlicht* e *Ogre 3D*.

3.2.1 Panda 3D

O Panda3D é um conjunto de bibliotecas C++ que utiliza a linguagem script *Python* para tratar estas bibliotecas, elevando o nível de abstração durante a programação do jogo. Tal característica permite obter uma curva de aprendizado alta em pouco tempo (NETTO; MACHADO; MORAES, 2006).

Além do fato de ser apontado por muitos como de fácil aprendizado, talvez a principal característica da *engine*. Ela ainda conta com um bom material para suporte ao aprendizado, incluindo um bom manual e um fórum de discussão disponíveis em seu site oficial (PANDA 3D, 2008).

Pode ser utilizado sob plataforma Windows, Linux e SunOS. Além de suportar OpenGL e DirectX.

O Panda3D utiliza o conceito de objeto e nodos, otimizando o processo de renderização da cena, já que, dado um nodo com vários filhos, estes filhos terão os mesmo atributos que o nodo pai como cor, iluminação, transparência e movimento, dentre outras características (NETTO; MACHADO; MORAES, 2006).

Como destaque principal, o Panda3D tem a sua concepção e desenvolvimento voltados para aplicações envolvendo RV e computação gráfica. Desse modo, oferece suporte a dispositivos específicos de RV e tem uma adaptação simples para as várias formas de projeção estereoscópica (NETTO; MACHADO; MORAES, 2006).

3.2.2 Crystal Space

Crystal Space é um software livre que consiste de um pacote de componentes e bibliotecas que pode ser utilizado para a construção de jogos de computador 2D e 3D. Ele disponibiliza ferramentas para a construção de cenários,

exemplos de código para demonstração das suas capacidades básicas, e uma boa documentação da sua estruturação e utilização (MADEIRA, 2001).

É considerado um kit de desenvolvimento de jogos 3D livre e portável escrito em C++ (MOTORES DE JUEGOS, 2008). É distribuída gratuitamente sob a licença LGPL, além de ser compatível com as plataformas Windows, Linux e MacOS. Também suporta DirectX no Windows e OpenGL em todos os sistemas (CLUA; BITTENCOURT, 2005).

Uma das principais características do *Crystal Space* é que seus componentes são projetados para serem funcionais com o mínimo de outros componentes em conjunto (MADEIRA, 2001).

O *Crystal Space* é considerado por Madeira (2001) como a mais bem desenvolvida entre as *engines* livres, pois utiliza os princípios de projetos orientados a objetos para a construção de softwares modulares, reusáveis e portáveis, e apresenta boa documentação de código.

Além do mais, muitos apontam como o principal problema o de não funcionar para ambientes abertos. Ele foi projetado inicialmente para renderizar cenários de ambientes fechados com polígonos que apresentem quantidade moderada de vértices (MADEIRA, 2001).

Há uma grande comunidade discutindo e aprimorando o software. Possui uma boa documentação on-line e um fórum de discussão no site do projeto (CRYSTAL SPACE, 2008).

Não há disponibilização de nenhum arquivo binário, logo o *Crystal Space* necessita ser compilado pelo próprio usuário. O processo de instalação é trivial, pois se trata de descompactar um arquivo. Nesse arquivo estão disponibilizados o manual do programador, o guia da API do *Crystal Space* e todo seu código-fonte (CLUA; BITTENCOURT, 2005).

O processo de instalação pode ser feito nas plataformas Microsoft Windows, GNU/Linux e MacOS (CLUA; BITTENCOURT, 2005). Há muitos jogos e aplicações que utilizam o *Crystal Space*, entre eles um bom exemplo de aplicação é o jogo *Plane Shift* (PLANE SHIFT, 2008).

3.2.3 Irrlicht

Irrlicht é um motor de alto desempenho. É uma API de alto nível para criar aplicações científicas em 2D e 3D. Possui uma boa documentação e integra

tecnologias como sombras dinâmicas, sistemas de partículas, animação de personagens, tecnologia de espaços interiores e exteriores e detecção de colisão. Tudo isto é acessível através de uma bem desenhada interface C++, muito fácil de utilizar (QUIROZ, 2005).

Ele é multiplataforma, rodando atualmente sobre Microsoft Windows, Mac OS X e Linux, e pode-se encontrar vários aprimoramentos para Irrlicht por toda a web, como exportadores, tutoriais, editores, *bindings* para as linguagens java, perl, rubi, basic, python, lua, e assim por diante. (IRRLICHT, 2008).

O Irrlicht é conhecido por sua velocidade e funcionalidades de motor 3D, além da pequena curva de aprendizado. Outras funcionalidades que são requeridas para se desenvolver jogos devem ser fornecidas por outros pacotes, tais quais, motor de áudio como o *Audiere* ou motor de física como o ODE. Possui seu código aberto e disponível sob a licença flexível ZLIB, que permite seu uso para software livre ou comercial (WIKIPEDIA IRRLICHT, 2008).

Ela é vista por alguns como a engine ideal para desenvolvedores iniciantes. Ela utiliza uma *scenegraph* hierárquico para gerenciamento de cenas, e tem funcionalidades básicas GUI 2D. É consideravelmente menor que outras *engines*, com apenas um desenvolvedor principal, porém há um grande suporte da comunidade de entusiastas. Provê algumas funções físicas básicas, mas carece de funções de rede (KOT, 2005).

Para Gowen (2004), o melhor do Irrlicht é sua curva de aprendizado. Ele cita que estava desenvolvendo aplicações 3D depois de ter examinado um exemplo de “hello, world”. Ela utiliza um sistema baseado em classes que é bastante intuitivo e fácil de lembrar.

Ele suporta muitos formatos de arquivos. Pode carregar e mostrar arquivos *3ds max*, modelos *Quake 2 .md2*, objetos *Maya .obj*, mapas *Quake 3 .bsp*, objetos *Milkshape3D* e arquivos DirectX *.x*. Objetos *3ds Max* e *Maya* são largamente suportados por muitos modeladores 3D, então a criação destes arquivos é muito fácil (GOWEN, 2004).

Sistema de partículas também é executado no motor, possibilitando a criação de efeitos complexos, tais como incêndios e mananciais de água facilmente. Ela tem até um rudimentar efeito de gravidade que você pode anexar ao sistema de partículas que provocam a queda de partículas após alguns segundos (GOWEN, 2004).

O irrlicht tem uma grande e ativa comunidade que pode responder perguntas sobre a engine (GOWEN, 2004). Algumas telas de projetos desenvolvidos com Irrlicht são demonstrados na Fig. 8.

Figura 8 - Imagens de telas de projetos desenvolvidos com Irrlicht.
Fonte: IRRLICHT, 2008.

A instalação da biblioteca Irrlicht ocorre muito facilmente. Primeiramente, o download é disponibilizado no site oficial do Irrlicht (2008), o arquivo é descompactado e os caminhos de configuração adicionados a plataforma de desenvolvimento utilizada, neste caso, o *Microsoft Visual C++ 2005 Express Edition*.

Como ocorre normalmente para o desenvolvimento em Windows, deve ser feita a instalação da plataforma SDK disponível no site oficial da Microsoft (2008).

3.2.4 Ogre 3D

Ogre 3D (*Object-Oriented Graphics Rendering Engine*), ou simplesmente Ogre, é uma madura, estável, flexível, multiplataforma e completa biblioteca para se utilizar no desenvolvimento de aplicações gráficas 3D em tempo real (JUNKER, 2006). Escrita em C++, orientada a objetos, distribuída gratuita e livremente sob a licença LGPL. Ela foi desenvolvida para tornar mais fácil e mais intuitiva para desenvolvedores produzirem aplicações utilizando hardware acelerador 3D (Ogre3D, 2008). Mas é puramente uma *engine* gráfica, não tem editor de fases, sistema de som e outros sistemas utilizados em *game engines*. Entretanto, é uma *engine* gráfica muito popular e robusta (ELIAS, 2005).

Atualmente conta com um grande e respeitado time de desenvolvedores. Além de ser largamente utilizado pela comunidade desenvolvedora de jogos, que contribuem para sua melhoria e participam ativamente de fóruns e listas de discussão. Provavelmente, é a engine livre mais utilizada no mundo. Somado ao bom suporte disponibilizado pela classe, a plataforma conta com uma bem completa documentação on-line disponível em seu site oficial (Ogre3D, 2008).

A engine fornece ao desenvolvedor a possibilidade de personalização e extensão, potencializando desta forma a reusabilidade de código, abstraindo primitivas de baixo nível e permitindo o desenvolvimento de novas técnicas que poderão ser acopladas na arquitetura dos motores.

O Ogre 3D oferece suporte a renderização com DirectX e OpenGL, *shaders*, suporte a texturas, além de uma série de outras funcionalidade de suporte para criação de jogos 3D (CLUA, BITTENCOURT, 2005). A biblioteca de classes permite abstrair os detalhes associados às bibliotecas de baixo nível, proporcionando uma interface baseada em objetos (MOTORES DE JUEGOS, 2008).

Possui ainda, características para facilitar o desenvolvimento de uma aplicação gráfica, já que oferece recursos que aumentam o desempenho durante a renderização de cenas e objetos tridimensionais.

Em contrapartida a tantos benefícios, o Ogre cuida apenas da parte gráfica com relação ao ambiente 3D. Física, rede, interface (GUI), som e outros componentes do jogo devem ser adicionados através das várias bibliotecas que possuem compatibilidade com a Ogre. Desta forma, o Ogre trabalha em conjunto com um motor próprio para cálculo e execução de propriedades físicas para objetos e cenas - ODE - o que facilita ainda mais a produção de um jogo eletrônico (NETTO; MACHADO; MORAES, 2006).

Também possui recentes técnicas e tecnologias na área de computação gráfica como sombreamento por vértice e por fragmentos de programas; multi-textura; animação independente de grama (ou pêlos); suporte a arquivos comprimidos (arquivos *zip* e *PK3*); sistema de partículas; efeitos especiais com transparência (NETTO; MACHADO; MORAES, 2006).

Pode-se utilizar Ogre tanto em plataforma Windows, como também em Linux ou Mac OS. Possui formato de importação próprio (*mesh*). Porém, é possível criar objetos em softwares de modelagem, como no *3D Studio Max* ou *Blender 3D* por

exemplo, que contam com algum *plugin* para exportação em *mesh*, que posteriormente podem ser importados pelo Ogre.

Quanto à RV, o Ogre3D dá suporte às mais variadas formas de visualização estereoscópica (anaglifo, luz polarizada), Head Mounted Displays (HMDs), Caves e muros de visualização (NETTO; MACHADO; MORAES, 2006).

Existem algumas características presentes no Ogre que podem ser consideradas as principais razões para o seu sucesso. Uma delas é a abordagem orientada a objetos, o que contribui para um desenvolvimento mais estruturado e dinâmico.

O fato de Ogre ser composta por classes com interfaces bem definidas entre si favorece a execução e abstração de diferentes partes da engine. Não existe uma dependência em relação à forma como os módulos responsáveis para a realização de operações nativas e renderização em 3D são codificados. Devido a isso, Ogre torna-se uma plataforma independente, sendo possível portá-lo para uma ampla gama de ambientes.

Outra vantagem é o uso de scripts escritos em formato de texto para carregar as configurações do aplicativo. O desenvolvedor então é capaz de alterar configurações como texturas dos objetos e materiais sem ter de recompilar o programa.

A arquitetura do Ogre é demonstrada na Fig. 9. A classe principal do Ogre é a classe Root, responsável pela instanciação e acesso a todos os outros ramos do motor. Ogre tem classes de gerenciamento para cada elemento do sistema. Um *ResourceManager* controla todos os recursos disponíveis para a aplicação. Existem muitos tipos de *ResourceManager*, como *TextureManager*, *FontManager* e *MeshManager*. A classe responsável pela organização dos elementos da cena é chamada *SceneManager*. O *SceneManager* tem uma referência a um objeto *RenderSystem*, que é responsável por renderizar a cena usando uma biblioteca gráfica. Outro item importante é a *PlatformManager*, que encapsula todas as funcionalidades que são nativas do sistema operacional destinadas pela aplicação, como a movimentação de entrada, calendário e gerenciamento de GUI (LIMA, 2006).

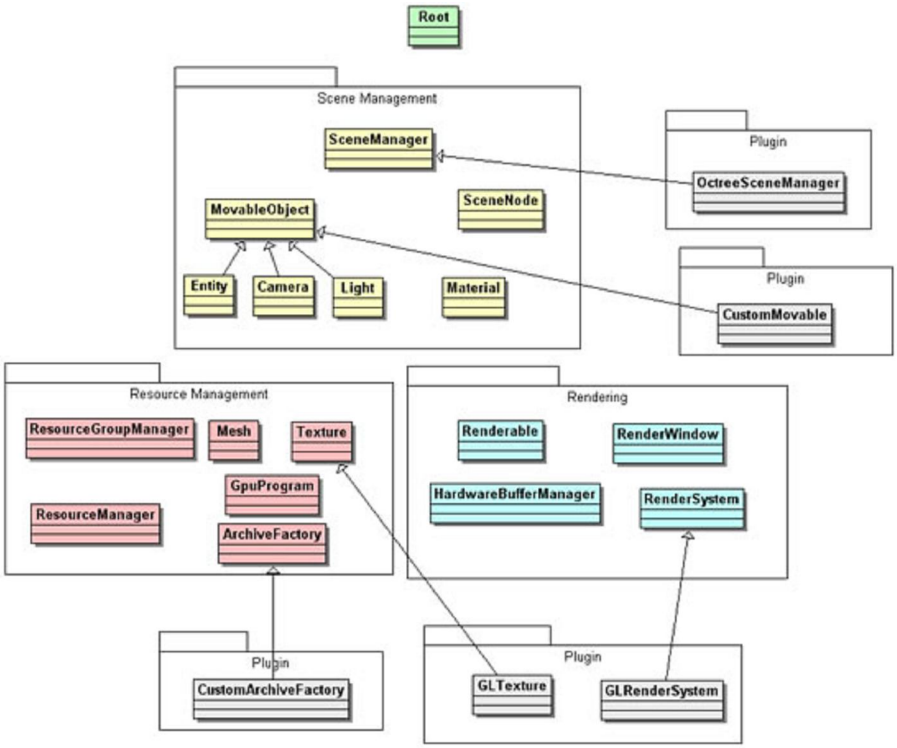


Figura 9 - Diagrama da arquitetura do Ogre.

Fonte: OGRE, 2008.

A instalação em plataforma Windows é muito simples, dá-se como qualquer instalação típica de Windows. Necessitando apenas, incluir as bibliotecas no ambiente de desenvolvimento utilizado. Também há a necessidade da instalação da plataforma SDK. Alguns projetos desenvolvidos com Ogre são mostrados na Fig. 10.

Figura 10 - Imagens de telas de alguns projetos desenvolvidos com Ogre.

Fonte: OGRE, 2008.

4 CORES, ILUMINAÇÃO E TONALIZAÇÃO

4.1 Cores

Na renderização de objetos coloridos deve-se levar em consideração algum sistema computacional de representação de cores. O modelo de cores mais utilizado e conhecido é o que combina as cores vermelho, verde e azul para gerar as outras cores. Mas outros modelos de cores são utilizados em aplicações gráficas.

Hearn e Baker (1997) fundamentam que um modelo de cores é um método para explicação das propriedades ou comportamento das cores em algum contexto particular. Um único modelo de cores não pode explicar todos os aspectos das cores, então fazemos uso de diferentes modelos para ajudar a descrever as diferentes características das cores percebidas.

Existem vários sistemas de representação de cores, como o CMY e o HIQ. Além dos dois sistemas que se destacam nesta representação, o RGB e o HSV. Sendo que se pode dizer que o sistema RGB é normalmente utilizado em computação gráfica.

Modelo RGB

O sistema RGB (*Red*, *Green*, *Blue*) representa cada cor com base na sua intensidade de vermelho, verde e azul. Este modelo é utilizado em monitores CRT coloridos, por exemplo.

As cores primárias RGB são aditivas, isto é, as contribuições individuais de cada primária aditiva são adicionadas para produzir o resultado. O cubo de cores RGB é mostrado na Fig. 11. A diagonal principal do cubo, com quantidades iguais de cada primária, representa os níveis de cinza: preto (0, 0, 0) e branco (1, 1, 1) (FOLEY et al., 1993).

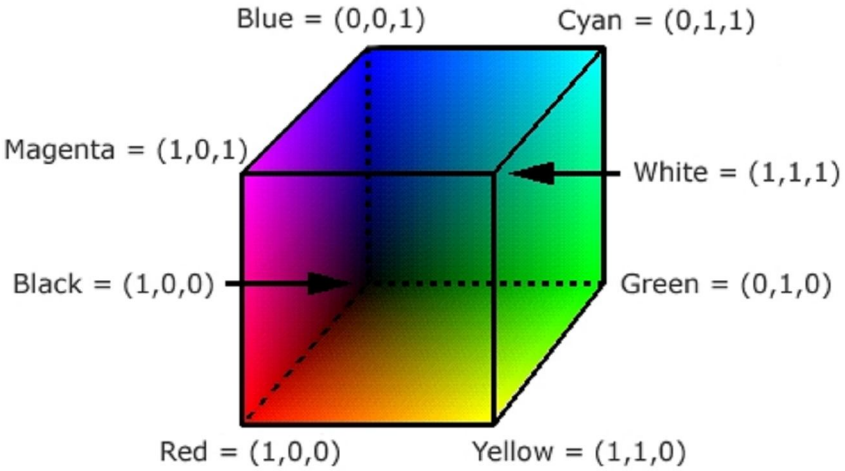


Figura 11 - Ilustração do cubo RGB.

Fonte: PROSJEKT, 2008 (adaptada).

Modelo HSV

Diferentemente dos outros modelos que são orientados a hardware, pois são utilizados em sistemas físicos de cores, como impressoras e monitores, o sistema HSV (*Hue*, *Saturation*, *Value*) é orientado ao usuário, sendo baseado na atração intuitiva do modelo artístico de tinta, sombra e tom (FOLEY et al., 1993).

Este modelo representa a cor com base em sua matiz, saturação e brilho. Onde o sistema de coordenadas é cilíndrico e define o modelo de cores através do cone hexagonal conforme mostra a Fig. 12.

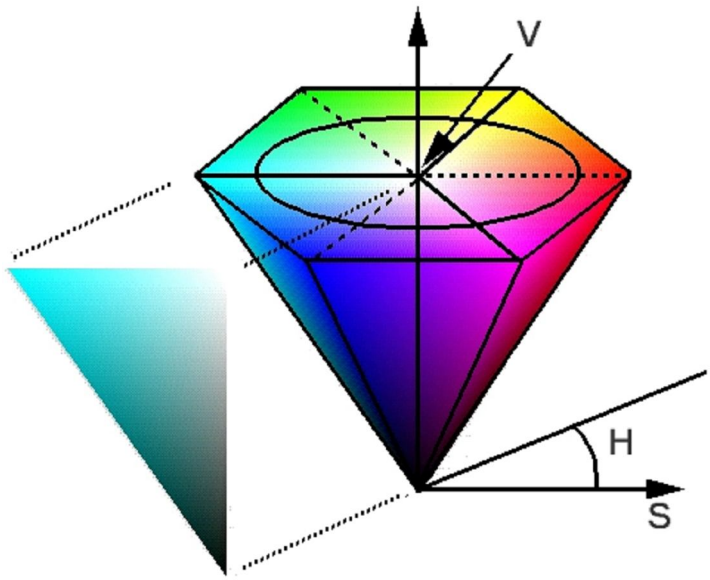


Figura 12 - Ilustração do cone hexagonal HSV.

Fonte: PROSJEKT, 2008 (adaptada).

Foley et al. (1993) demonstra com a figura, que o topo do cone corresponde a V=1, que representa a relatividade do brilho das cores. O matiz H é mensurado por um ângulo ao redor do eixo vertical, com vermelho em 0° e verde em 120°. Cores complementares no cone HSV são opostas 180° uma da outra. O valor de S é um valor entre 0, na linha do centro, e 1 nas extremidades do cone.

4.2 Iluminação

Segundo Hearn e Baker (1997), exibições realísticas de uma cena são obtidas pela geração de projeções perspectivas dos objetos e pela aplicação do efeito de iluminação natural das superfícies visíveis. Um modelo de iluminação é usado para calcular a intensidade de luz que podemos ver em um dado ponto na superfície do objeto.

A renderização de uma superfície pode ser produzida pela aplicação do modelo de iluminação para todos os pontos da superfície visível, ou a renderização pode ser produzida pela interpolação de intensidades, resultante do cálculo do modelo de iluminação de um pequeno grupo de pontos (HEARN; BAKER, 1997).

Além do modelo de iluminação empregado na renderização, as características das fontes de luz também devem ser analisadas quando se calcula a cor definida para cada pixel da cena.

4.2.1 Fontes de Luz

Quando visualizamos um objeto não luminoso e opaco, nós vemos a luz refletida na superfície do objeto. O total de luz refletida em um ambiente é a soma das contribuições das fontes de luz e da reflexão de outras superfícies na cena como mostra a Fig. 13 (HEARN; BAKER, 1997).

As fontes de luz de uma cena podem ser consideradas um objeto como os outros e se distinguem apenas pelo fato de emitirem luz. É a interação dos raios de luz com os objetos que definem o que será visto na cena. Não havendo nenhum ponto de luz, nenhum objeto da cena seria visualizado (TRAINA; OLIVEIRA, 2006).

Algumas grandezas são atribuídas às fontes de luz, são elas:

- Geometria: formato físico da fonte;
- Intensidade: associa uma intensidade de iluminação a cada ponto;
- Distribuição espectral: distribuição da fonte em cada comprimento de onda (cor) do espectro visível.

As fontes de luz podem ser de três tipos quanto ao seu aspecto geométrico:

- Fontes direcionais: considerada no infinito e pode ser determinada por um vetor unitário que define sua direção. Em geral são utilizadas para aproximar fontes de luz pontuais a uma distância infinita, como o sol;
- Fontes pontuais: definidas por um vetor posição (x, y, z) no espaço da cena e um vetor unitário para definir sua direção. Suas dimensões são desprezíveis se comparadas aos objetos da cena;
- Fontes de área: possuem uma superfície não pontual emissiva e um sistema de coordenadas locais associado que é utilizado para especificar a sua posição no espaço e direção da iluminação.

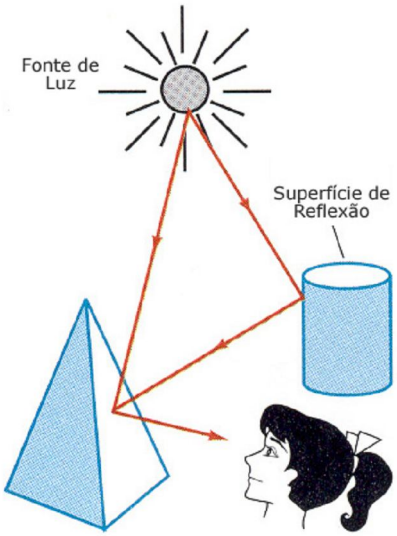


Figura 13 - Ilustração da reflexão das superfícies.
Luz visualizada de uma superfície não luminosa opaca é uma combinação da luz refletida de uma fonte de luz e a reflexão da luz refletida de outros objetos.
Fonte: HEARN e BAKER, p. 496, 1997 (adaptada).

4.2.2 Modelos de Iluminação

Luz Ambiente

Um modelo mais simples de iluminação é utilizado implicitamente nos projetos de ambientes virtuais. Pois sem ao menos a luz ambiente, nenhum objeto é visualizado.

De acordo com Foley et al. (1993), este modelo pode ser expresso por uma equação de iluminação com as variáveis associadas a cada ponto do objeto que

está sendo tonalizado. A equação de iluminação que expressa este modelo simples é:

$$I = I_a k_a$$

Onde k_a é o nível de luz ambiente na cena e, portanto, iluminada com um valor constante para toda a superfície. Resultado que independe da direção da visualização e da orientação espacial da superfície (HEARN; BAKER, 1997).

Levando-se em consideração que a intensidade da luz I_a é constante para todos os objetos, a quantidade de luz ambiente refletida pela superfície do objeto é determinada por k_a.

Porém, Hearn e Baker (1997) dizem que essa intensidade da luz refletida depende das propriedades ópticas da superfície, ou seja, o quanto de energia incidente está sendo refletida e o quanto está sendo absorvida.

Quando uma luz atinge um objeto, uma porção dela é refletida, assim como é absorvida e transmitida. Objetos têm uma tendência para seletivamente absorver, refletir ou transmitir luzes de certas cores. Dessa forma, um objeto poderia refletir a luz verde enquanto absorve todas as outras freqüências de luz visível, fazendo com que possamos enxergar este objeto verde (GOVIL-PAI, 2004).

Um modelo que represente fielmente o fenômeno real é muito custoso computacionalmente. Logo, o que se utiliza é uma simplificação deste modelo, onde a cor dos objetos é definida por valores da reflexão difusa ou especular.

Reflexão Difusa

Considerando um objeto iluminado por uma luz pontual, que emite raios uniformemente em todas as direções de um mesmo ponto. O brilho do objeto varia de uma parte para outra, dependendo da direção e da distancia da fonte de luz (FOLEY et al., 1993).

Objetos foscos possuem superfícies difusas. Estas superfícies possuem o mesmo brilho de todos os ângulos de visão porque elas refletem a luz em todas as direções em igual intensidade. Para cada superfície, o brilho depende apenas do ângulo θ entre a direção da fonte de luz $\overline{L}$ e a normal da superfície $\overline{N}$.

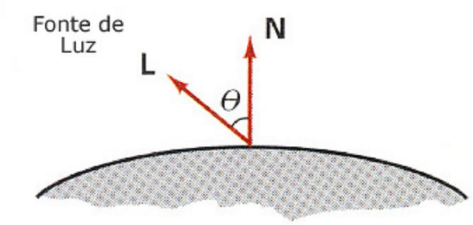


Figura 14 - Ângulo de incidência da luz.
Ângulo de incidência θ entre a direção da luz refletida e a normal da superfície.
Fonte: HEARN e BAKER, p.499, 1997 (adaptada).

A equação para cálculo da iluminação difusa é dada por:

$$I = I_p k_d \cos\theta$$

Onde a iluminação é afetada pela intensidade do ponto de luz I_p, pelo coeficiente de iluminação difusa do material k_d e pelo ângulo θ, que precisa estar entre $0°$ e $90°$ se a fonte de luz tiver algum efeito direto no ponto que está sendo tonalizado.

Dessa forma, a luz refletida difusamente é emitida pela superfície com igual intensidade em todas as direções, sendo de menor intensidade que a luz incidente. Este tipo de reflexão é característico de superfícies ásperas.

Reflexão Especular

A reflexão especular é o resultado de uma reflexão perfeita, ou seja, o ângulo em que a luz deixa o objeto é determinado pelo ângulo segundo o qual a luz atinge o objeto. Ao contrário da reflexão difusa, onde não se pode determinar em um primeiro momento, a direção em que o raio de luz irá tomar (TRAINA; OLIVEIRA, 2006).

Segundo Hearn e Baker (1997), a reflexão especular é fenômeno que percebemos quando olhamos uma superfície lisa, como um metal polido ou uma maçã. Onde visualizamos uma parte mais iluminada que o restante. Isto é o resultado de uma total, ou quase total, reflexão da luz incidente em uma região concentrada ao redor do ângulo de reflexão especular.

A Fig. 15 mostra a direção da reflexão especular em um ponto da superfície iluminada.

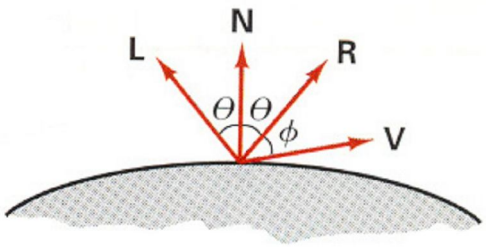


Figura 15 - Ângulo de reflexão especular é igual ao ângulo de incidência.
Fonte: HEARN e BAKER, p.501, 1997.

Foley et al. (1993) demonstra o modelo de iluminação Phong, desenvolvido para o cálculo de refletores não perfeitos, como a maçã. O modelo assume o valor máximo de reflexão especular quando o ângulo ϕ é zero e diminui à medida que este ângulo aumenta.

O ângulo pode assumir valores entre $0°$ e $90°$, logo o $\cos\phi$ varia entre 0 e 1. A intensidade de reflexão especular depende das propriedades materiais da superfície e do ângulo de incidência, além de outros fatores como a polarização e a cor da luz incidente. Podemos obter, aproximadamente, a variação da intensidade de um modelo monocromático especular, utilizando um coeficiente de reflexão especular $W(\theta)$ para cada superfície (HEARN; BAKER,1997).

Em geral, o coeficiente $W(\theta)$ tende a aumentar à medida que o ângulo aumenta. Utilizando a função de reflexão espectral $W(\theta)$, nós podemos escrever o modelo de reflexão especular Phong da seguinte forma.

$$I = W(\theta) I_l \cos\phi$$

Onde I_l é a intensidade da fonte de luz, e ϕ é o ângulo de visualização relativo à direção da reflexão especular *R*.

A Fig. 16 demonstra duas superfícies com características diferentes. A imagem *a* de uma superfície lisa especular com os raios sendo refletidos paralelamente. E a imagem *b* de uma superfície rugosa difusa, onde os raios refletem em varias direções.

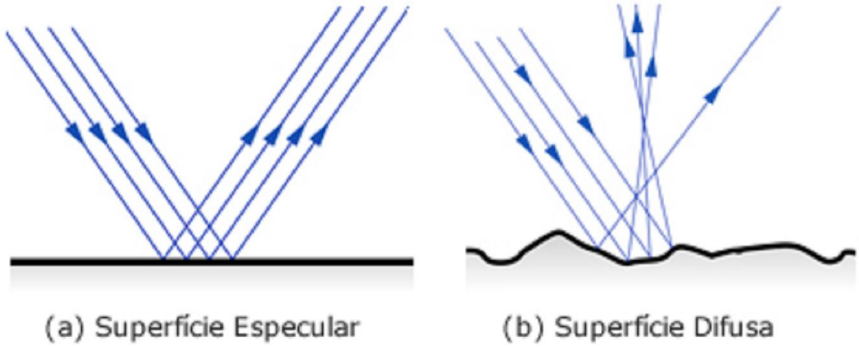


Figura 16 - Reflexão da luz em superfícies especular e difusa.

Há dois tipos extremos de superfícies quando se considera o fenômeno da reflexão: as refletoras idealmente especulares, que atuam como espelhos, e as refletoras idealmente difusas, que são opacas ou foscas. Porém, a grande maioria dos objetos possui características intermediárias entre estes extremos (TRAINA; OLIVEIRA, 2006).

4.3 Tonalização

A renderização da cena pode ser feita aplicando a equação de iluminação em cada pixel da superfície de cada objeto. Mas este modelo de tonalização por força bruta requer muita capacidade de processamento é torna-se inviável (FOLEY et al., 1993).

Além disso, em uma cena existem muitas superfícies que não são visualizadas e, para otimizar o processo, não precisam ser renderizadas. Desta forma, um algoritmo de remoção de superfícies ocultas deve ser utilizado (TRAINA; OLIVEIRA, 2006).

No caso de uma cena modelada por um conjunto de polígonos, o algoritmo *scanline* pode ser aplicado, e o modelo de iluminação é usado para calcular a cor de cada pixel.

Este processo de iluminação é baseado no modelo local de iluminação, ou seja, são considerados no cálculo da cor apenas a luz ambiente e os raios de luz que incidem diretamente sobre cada ponto da superfície. A interação dos raios refletidos pelos outros objetos é ignorada. Diferentemente do *ray-tracing*, um algoritmo baseado em um modelo global de iluminação, que considera estas interações.

As técnicas de tonalização (*Shading*) utilizam algum modelo de iluminação para determinar as cores associadas aos pixels de cada polígono que compõe a cena. Estas técnicas dividem-se em dois tipos: onde cada polígono pode ser renderizado com uma única intensidade, ou quando a intensidade é obtida para cada ponto da superfície utilizando um esquema de interpolação (HEARN; BAKER, 1997).

O mais simples deles é o de Tonalização Constante (*Flat Shading*), porém costuma-se utilizar modelos que geram objetos mais próximos da realidade, através da interpolação. São eles: Tonalização por *Gouraud* (*Gouraud Shading*) e a Tonalização por *Phong* (*Phong Shading*).

4.3.1 *Flat Shading*

Na tonalização constante é considerada a iluminação ambiente e a reflexão difusa, com apenas uma luz posicionada no infinito e na direção do observador para evitar sombras. O processo é bastante simples e eficiente, porém não considera o fato de que, a representação poliedral de um modelo representa uma aproximação linear por partes do modelo real (TRAINA; OLIVEIRA, 2006).

Este modelo assume que a representação poligonal é o modelo real em estudo. Fazendo com que cada polígono represente uma parte do modelo onde o vetor normal é constante e, portanto a intensidade de luz também seja constante, gerando uma aparência facetada da imagem. Sendo assim, aplicamos a equação de iluminação apenas uma vez para cada polígono (FOLEY et al., 1993).

Podemos perceber claramente as diferenças de resultado na figura 4.9, utilizando a tonalização constante ou alguma das tonalizações que utiliza interpolação.

Tonalização Interpolada

Como uma alternativa para a aplicação da equação de iluminação para cada ponto do polígono, surgiu a tonalização interpolada, que realiza uma interpolação linear entre valores previamente calculados. Estes valores podem ser definidos nos vértices (*Gouraud*) ou nas normais (*Phong*) da malha de polígonos.

4.3.2. *Gouraud Shading*

A técnica *Gouraud* estende o conceito de tonalização interpolada aplicado a polígonos individuais, consiste em aplicar modelos de iluminação para calcular as intensidades nos vértices do polígono, e interpolar os valores obtidos para obter a iluminação ao longo de cada aresta e nos pontos interiores.

De acordo com Foley et al. (1993), cada superfície de polígono é renderizada com *Gouraud* da seguinte forma:

Um primeiro passo é encontrar a normal para cada vértice da malha de polígonos. *Gouraud* está apto a calcular essas normais dos vértices diretamente de uma descrição analítica da superfície.

Posteriormente, encontra-se a intensidade dos vértices utilizando as normais destes vértices com algum modelo de iluminação designado. Logo após, cada polígono é tonalizado pela interpolação linear das intensidades dos vértices ao longo das extremidades do polígono.

Finalmente, o interior do polígono é tonalizado realizando outra interpolação linear e utilizando um algoritmo *scanline* como mostra a Fig. 17.

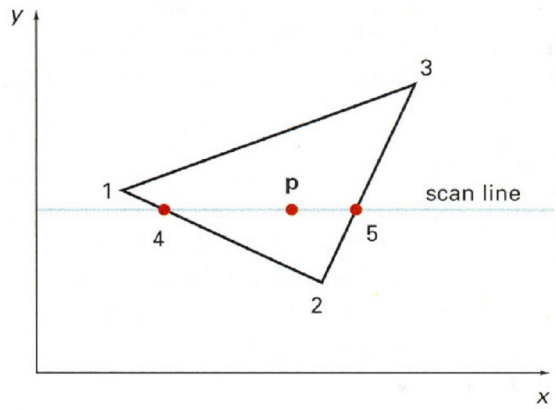


Figura 17 - Processo de Tonalização *Gouraud*.

Pela tonalização *Gouraud*, a intensidade no ponto 4 é linearmente interpolada das intensidades dos vértices 1 e 2. A intensidade no ponto 5 é linearmente interpolada das intensidades dos vértices 2 e 3. A intensidade do ponto interno **p** é encontrada pela interpolação linear das intensidades das posições 4 e 5.

Fonte: HEARN e BAKER, 1997.

4.3.3 *Phong Shading*

Um método mais exato para renderizar a superfície do polígono é interpolar os vetores normais, e aplicar o modelo de iluminação para cada ponto da superfície.

Este método, que encontra melhores resultados, é chamado *Phong shading*. Ele, na maioria das vezes, demonstra mais realismo nos objetos.

O cálculo dá-se da seguinte forma: primeiramente, encontram-se os vetores normais dos vértices de cada polígono. Após, realiza-se a interpolação linear dessas normais sobre a superfície do polígono. E por fim, aplica-se o modelo de iluminação ao longo de cada linha do polígono, para calcular a intensidade do pixel para os pontos da superfície. Como demonstra a Fig. 18 (HEARN; BAKER, 1997).

A tonalização por *Phong* apresenta os melhores resultados para superfícies especulares, mas tem um custo computacional mais alto.

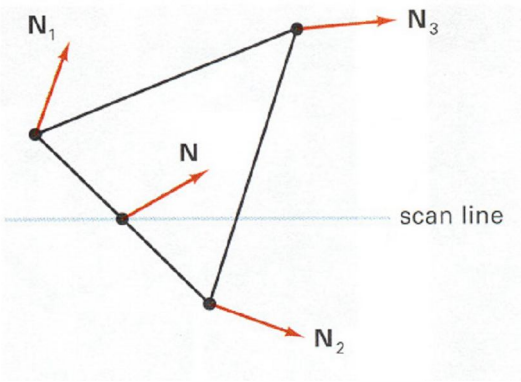


Figura 18 - Interpolação das normais ao longo das extremidades do polígono.
Fonte: HEARN e BAKER, 1997.

As técnicas desenvolvidas por *Gouraud* e por *Phong*, permitem a obtenção de uma aparência mais suave. Sendo, dessa forma, mais utilizada em superfícies que representam uma aproximação linear da superfície realmente desejada. Ambas consideram as componentes de iluminação ambiente, difusa e especular (TRAINA; OLIVEIRA, 2006). A Fig. 19 exemplifica a diferença das três técnicas acima mencionadas.

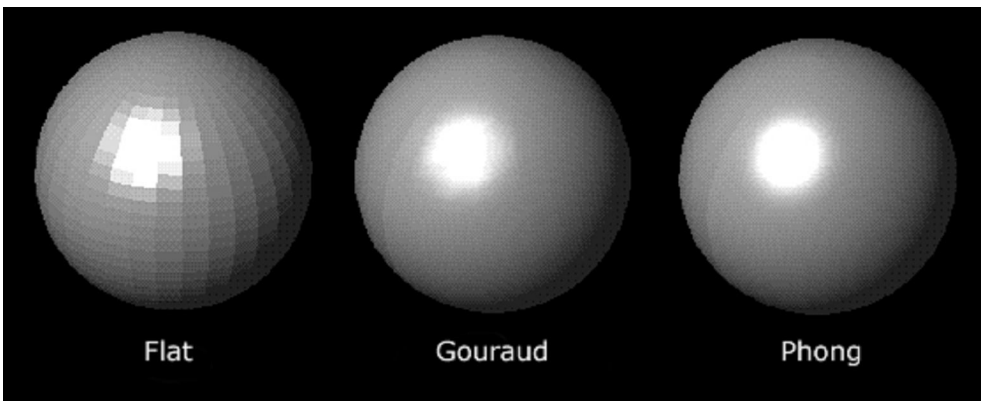


Figura 19 - Esferas tonalizadas com os métodos *Flat*, *Gouraud* e *Phong*.
Fonte: PROSJEKT, 2008.

5 METODOLOGIA

Um dos propósitos deste trabalho foi comparar as principais *engines* livres existentes, realizando uma análise teórica e prática de ambas, para que dessa forma pudéssemos selecionar a que mais se adaptaria a construção de um ambiente de demonstração de técnicas da computação gráfica.

Inicialmente, foi realizada uma comparação apenas teórica entre algumas *engines*, levando-se em consideração a opinião de desenvolvedores em fóruns sobre desenvolvimento de jogos e listas de discussão da área.

Decorrente desta primeira etapa, foram selecionadas quatro engines livres de grande utilização. São elas: Irrlicht 3D, Ogre 3D, Crystal Space e Panda 3D. Que se encontram, respectivamente, nas primeiras colocações de uma relação das mais comentadas *engines* de código aberto em um importante site de desenvolvedores de games, como mostra a tabela 1.

Tabela 1 - Engines *open source* mais comentadas.

1.	Irrlicht
2.	Ogre
3.	Crystal Space
4.	Panda3D
5.	jME
6.	Reality Factory
7.	The Nebula Device 2
8.	Blender Game Engine
9.	RealmForge
10.	OpenSceneGraph

Fonte: DEVMASTER, 2008.

Em uma etapa posterior, foram analisadas algumas características dos quatro motores e selecionamos os dois melhores para o desenvolvimento do ambiente. As *engines* Ogre e Irrlicht destacaram-se nesta análise, tornando-se aptas

para uma análise mais completa, agora não apenas na teoria, como também na prática.

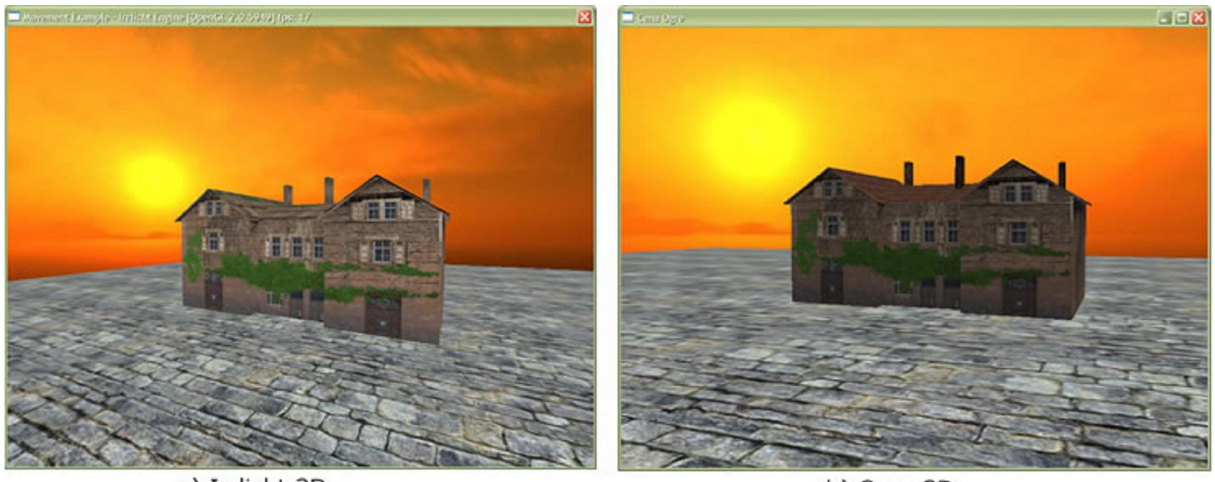
Levou-se em consideração para seleção das *engines*, principalmente, o fato de serem as mais populares entre as opções *open source*. Além disso, pesou na escolha uma melhor estruturação de seus códigos perante as demais e um grau de dificuldade de desenvolvimento mais baixo.

5.1 Irrlicht X Ogre

Houve uma comparação das características dos dois motores, onde se levou em consideração as vantagens de um sobre o outro. Porém, dando uma maior ênfase no que diz respeito às características gráficas da renderização do cenário.

Para realizar a comparação prática entre as *engines* Ogre e Irrlicht, foram desenvolvidos dois ambientes tridimensionais básicos. Com o propósito de comparar de forma “justa” a plataforma que produz melhores resultados, ambos os sistemas foram desenvolvidos da forma mais parecida possível, utilizando os mesmos objetos.

Imagens dos ambientes 3D criados podem ser visualizadas na Fig. 20, elas procuram demonstrar os sistemas testes desenvolvidos em ambas as plataformas.



a) Irrlicht 3D b) Ogre 3D

Figura 20 - Ambientes testes.
Imagens de telas dos ambientes comparativos desenvolvidos em Irrlicht (a) e em Ogre (b).

Tanto no ambiente desenvolvido com a *engine* Ogre, quanto no desenvolvido com a Irrlicht, foram utilizados o mesmo modelo tridimensional de uma

casa, disponível gratuitamente na internet (THE3DSTUDIO, 2008) e exportado no formato adequado para cada uma.

Para a biblioteca irrlicht, utilizou-se a casa modelada em formato (*3ds*), formato nativo e proprietário do *software 3D Studio Max*. A Ogre, por outro lado, possui um único formato próprio para importação das malhas (*mesh*). Dessa forma, houve necessidade de uma conversão de formatos, passível de ser atingida através de um conversor disponível para o software de modelagem *Blender*.

Utilizou-se também nos ambientes, um plano horizontal com uma mesma textura para simular o chão. Além de uma estrutura idêntica de *skybox*, inclusive com as mesmas imagens, para simulação do céu.

A técnica de *skybox*, demonstrada na Fig. 21 é uma das melhores formas de simular a visualização do céu em um sistema tridimensional. Ela é produzida mantendo-se seis planos com texturas ao redor da câmera que representa a visão do usuário, com as normais destes planos voltadas pra dentro, formando assim um cubo (SANTOS, 2004).

O *skybox* é tonalizado sem qualquer tipo de iluminação, fazendo com que a junção das imagens não seja visível, o que impossibilita ao usuário a consciência de estar dentro deste grande cubo (SANTOS, 2004).

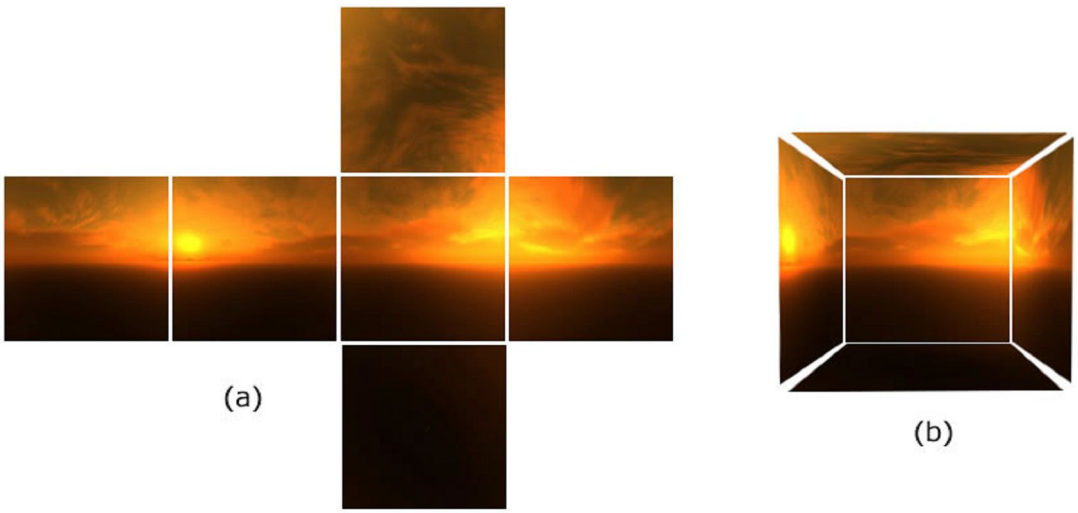


Figura 21 - Ilustração da técnica de skybox.

Além disso, as estruturas de iluminação foram compatíveis, de forma a não prejudicar o desempenho de uma plataforma em relação à outra. Onde, por

problemas de velocidade de renderização na plataforma adotada, foi utilizada apenas uma luz ambiente branca para permitir a visualização dos objetos.

Os dois ambientes foram criados e executados na seguinte plataforma computacional: Laptop Acer com processador AMD Turion 64bits, 2.0Ghz, 1Gb de memória DDR2, placa de vídeo ATI Radeon Xpress 1100 e HD 80GB PATA.

O irrlicht atualmente esta na versão 1.4.2, enquanto que o Ogre está na versão 1.6. Ambos ambientes foram codificados na IDE Microsoft Visual C++ 2005, mas em versões diferentes das últimas, que disponibilizaram versões estáveis após o desenvolvimento do ambiente.

O código desenvolvido utilizando irrlicht foi baseado nos exemplos disponíveis em seu site oficial (IRRLICHT, 2008), e o desenvolvimento em Ogre foi baseado em um exemplo de código disponível no website Irados (2008).

Pelo fato de uma das *engines* possuir recursos diferenciados e mais completos sobre a outra, estipulou-se compará-las apenas no que diz respeito aos seus recursos gráficos, principalmente considerando suas características e a qualidade de renderização.

Estipulou-se a escolha da Ogre para o desenvolvimento do ambiente final. Levaram a esta decisão, além de motivos pessoais de simpatia com a *engine*, as seguintes características:

- Na comparação teórica das características descritas no capítulo 3, a Ogre mostrou-se um pouco mais completa, no que diz respeito as funcionalidades gráficas;
- Encontra-se referência sobre a *engine* na internet mais facilmente, sendo que o site oficial, o fórum e o wiki concentram boa parte das informações necessárias;
- Na comparação prática entre Ogre e Irrlicht, a primeira apresenta uma renderização de objetos mais bem acabada.

5.2 Ambiente Final

Uma cena 3D foi criada para exemplificar alguns conceitos e técnicas empregados pela Computação Gráfica. Dessa forma, uma sala foi montada para que possa ser visualizado e modificado alguns objetos modelados. Preocupando-se com características da tonalização dos objetos e de iluminação da cena.

5.2.1 Construção do cenário

Estipulou-se desenvolver este ambiente a partir do sistema de teste criado com a plataforma da Ogre apresentada anteriormente, buscando o reaproveitamento de alguns trechos de programação necessários em ambos os ambientes.

Sendo assim, utilizou-se um sistema de câmera em FPS (*First Person Shoter*), Este método permite ao usuário a navegação livre por toda a sala, movendo a câmera com as teclas típicas dos jogos de computador e rotacionando a câmera sobre o próprio eixo com a utilização do mouse.

Com a navegação por FPS neste sistema, o usuário pode escolher o melhor local para o posicionamento da câmera e o ângulo de visão mais apropriado para interação com os objetos.

A sala foi montada com uma mesa no centro, disponível no site *the3dstudio.com* (2008), que serve de suporte para os objetos aptos a sofrer as modificações propostas pelo sistema. Estes objetos são rodeados por seis planos: quatro paredes, o teto e o piso, todos texturizados como mostra a Fig. 22.

Figura 22 - Tela que demonstra o cenário do ambiente CG House.

5.2.2 Objetos

Para receber as modificações disponíveis nos menus, dois objetos foram escolhidos. Um deles é o modelo tridimensional clássico de uma chaleira (Fig. 23-a), escolhido pela importância que representa na Computação Gráfica. Este objeto foi apontado por muitos como o mais adequado para a percepção dos efeitos resultantes da aplicação das técnicas de Computação Gráfica.

Para uma maior percepção de todas as modificações do sistema, constatou-se a necessidade da existência de outro objeto. Logo, foi inserido no ambiente e recebe as mesmas modificações impostas a chaleira, uma cabeça tridimensional de um ogro (Fig. 23-b). Optou-se por este objeto por ser o objeto clássico da *engine* utilizada no ambiente (OGRE, 2008).

(a) chaleira (b) cabeça do ogro

Figura 23 - Objetos utilizados no ambiente CG House.

5.2.3 Modelo de interação

Neste cenário apresentado, há dois menus de interação com o ambiente.

No primeiro menu o usuário tem a possibilidade de modificar as técnicas de tonalização clássicas que podem ser aplicadas aos objetos. Permitindo ainda, que os usuários selecionem os modos de visualização deste modelo e modifique os tipos de sombras disponíveis para o cenário.

De forma semelhante, o segundo menu leva em consideração os aspectos de iluminação, permitindo que o usuário interaja com o ambiente e modifique as propriedades dos pontos de iluminação.

Os dois menus, mostrados na Fig. 24, foram desenvolvidos com a utilização da biblioteca *Navi Library*.

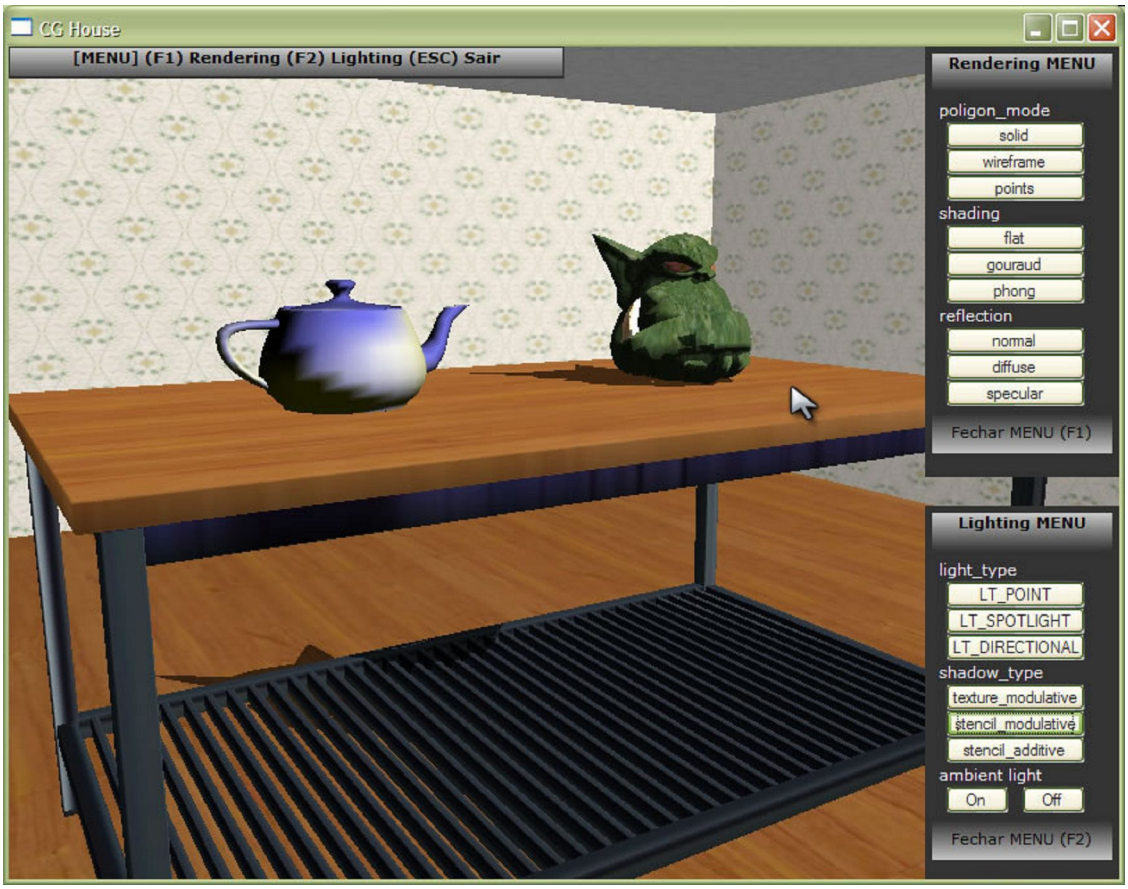


Figura 24 - Tela que demonstra o sistema interativo do ambiente CG House.

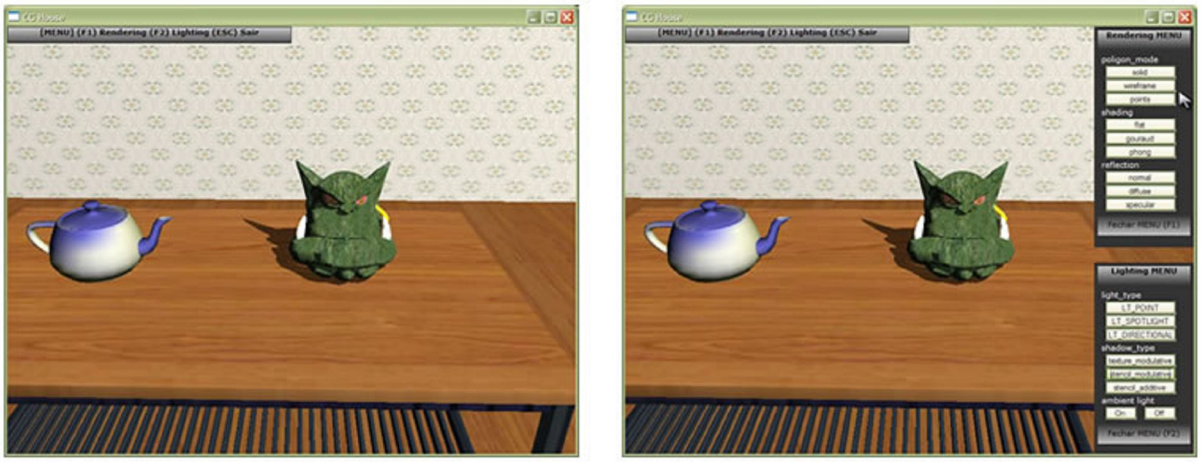
Navi Library

A alternativa encontrada para GUI (*Graphical User Interface*) do sistema, ou em português, Interface Gráfica do Usuário, foi a *Navi Library*, que se destacou entre as opções existentes.

Esta biblioteca possui um caráter diferenciado das demais, como a famosa CEGUI por exemplo. Ela difere, principalmente, por utilizar-se da estrutura da web para gerar os seus componentes. Dessa forma, permite uma grande personalização da aparência dos seus componentes por tratar-se de páginas HTML e folhas de estilos CSS. Característica que traz muitos benefícios gráficos se comparada às tradicionais Interfaces Gráficas que apresentam uma aparência fixa e padrão.

Utilizando a biblioteca *Navi* foram criados dois menus de interação com os objetos: um menu modifica os aspectos de tonalização e outro menu modifica as características da iluminação.

Os menus de interação podem ser ativados em qualquer momento no sistema pressionando as teclas *F1* e *F2*, como é visto na Fig. 25. Para sair do sistema a tecla *ESC* deve ser pressionada.



(a) menus fechados (b) menus abertos

Figura 25 - Ambiente de interação.

Tela que demonstra o ambiente com o sistema de interação inativo e ativo, respectivamente, nas figuras a e b.

A partir do momento que os menus são abertos, é desabilitada a rotação do eixo da câmera através do mouse, permitindo assim que o cursor possa ser movimentado para efetuar o clique em cada botão dos menus sem movimentar a câmera.

A Fig. 26 mostra quando o botão esquerdo do mouse é utilizado para fazer as interações com a cena. Enquanto que com o botão direito permite-se a mudança de posicionamento dos menus.

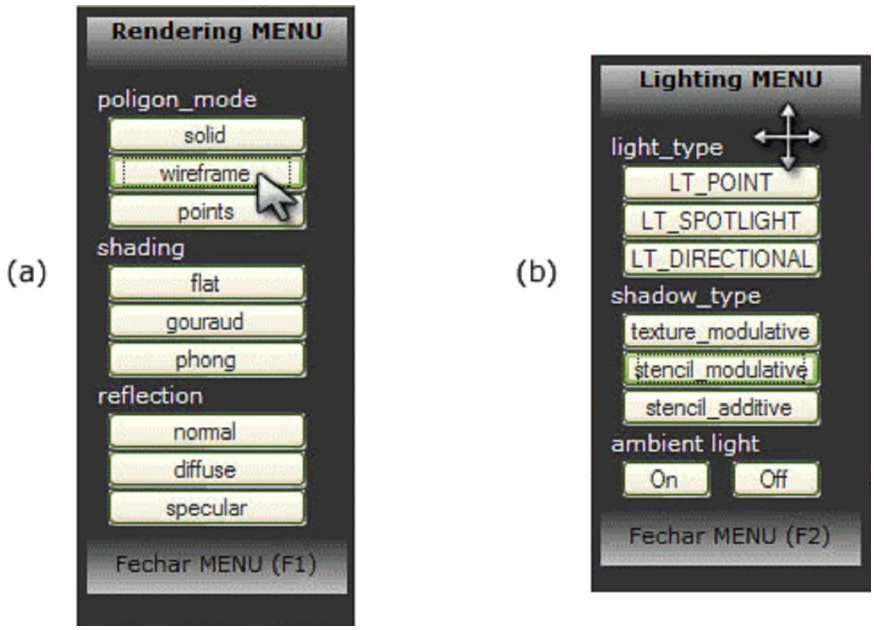


Figura 26 - Menus de interação com o ambiente CG House.

Menu Tonalização

O menu de tonalização permite uma série de modificações nas propriedades dos objetos.

A primeira delas é o modo com que os polígonos são mostrados (*poligon mode*). Classificam-se nos seguintes três tipos:

- *Solid*: apresenta os polígonos do objeto preenchidos com a coloração por toda a superfície. Deste modo, o objeto é demonstrado como ele é realmente;
- *Wireframe*: preenche com cores apenas o contorno dos polígonos (arestas). Ele permite visualizar os polígonos que modelam o objeto.
- *Points*: colore apenas os vértices da malha de polígonos. Ou seja, mostra os pontos de junção de vários polígonos da malha do objeto.

Outra funcionalidade que o menu de tonalização proporciona são as técnicas de tonalização (*shading*).

Estas técnicas de tonalização foram explicadas na seção 4.3 do capítulo 4. Elas possibilitam a visualização nas seguintes formas:

- *Flat*: a tonalização constante, onde se aplica a equação de iluminação apenas uma vez para cada polígono, fazendo com que o objeto possua uma aparência facetada;
- *Gouraud*: um tipo de tonalização interpolada, onde se calcula a intensidade dos vértices e interpolam-se estes para calcular a iluminação ao longo de cada aresta e nos pontos interiores;
- *Phong*: outro tipo de tonalização interpolada. Com esta técnica interpolam-se as normais dos vértices e aplica-se o modelo de iluminação para cada uma dessas normais geradas. Técnica mais custosa computacionalmente;

A última funcionalidade do menu de tonalização é a modificação nas propriedades de reflexão dos objetos. São elas:

- *Normal*: reflexão mais moderada, onde os valores de reflexão difusa e especular são equilibrados;
- *Diffuse*: reflexão difusa, a qual representa as superfícies rugosas e foscas;

- *Specular*: reflexão especular, que aplica aos objetos a simulação de superfícies lisas e polidas;

Menu Iluminação

O menu de iluminação preocupa-se com modificações nas luzes presentes no ambiente.

Uma mudança possível no menu de iluminação faz referencia ao tipo de luz. A luz que recebe estas modificações esta localizada no cenário em frente os objetos direcionada para eles. As fontes de luz do ambiente podem ser dos tipos descritos a seguir:

- *Point*: cria uma fonte de luz pontual que emite luz em todas as direções;
- *Spotlight*: cria uma luz similar a de uma lanterna;
- *Directional*: cria uma luz em uma direção que ilumina uma grande área, como se fosse a luz do sol ou da lua.

Outra característica referente à iluminação, são os tipos de sombras. Estão disponíveis no Ogre os seguintes tipos:

- *Texture_modulative*: o tipo de sombra mais simples, elas não possuem muita precisão e tornam-se um pouco borradas;
- *Stencil_modulative*: possui sombras bem mais definidas. Esta é indicada para ambientes que necessitam de qualidade nas sombras e de uma resposta mais rápida do sistema;
- *Stencil_additive*; o melhor tipo de sombra na engine. Porém, necessita de um maior processamento, pois a criação de sombras é verificada para cada luz da cena separadamente;

Para que os objetos possam ser visualizados em qualquer cena, este deve receber ao menos a luz ambiente. Neste sistema foram atribuídas duas possibilidades de luz ambiente, são elas:

- On: Aplica valores altos a luz ambiente, aparentando que a luz esteja ligada;
- Off: Aparenta a luz ambiente desligada, aplicando valores baixos a luz ambiente.

6 RESULTADOS

Algumas técnicas de Computação Gráfica foram selecionadas e disponibilizadas a interação dos usuários do sistema. Essas técnicas foram classificadas em dois menus: tonalização e iluminação.

6.1 Tonalização

O primeiro menu demonstra a seqüência de funcionalidades que se referem a características de tonalização. Foi selecionado um conjunto de três itens que modificam atributos dos objetos.

As modificações disponíveis neste menu afetam as características dos objetos através dos arquivos *material*, que são responsáveis por guardar estas informações de renderização de cada objeto, tais como textura, cor, etc.

A aplicação das modificações nos objetos dá-se em toda a entidade representada pela chaleira e em uma parte do objeto ogro, representado pela sub-entidade que forma a pele.

6.1.1 *polygon_mode*

Uma das técnicas disponíveis neste menu possibilita a modificação do modo de visualização dos objetos. Este item foi denominado *polygon_mode* por ser o nome do parâmetro dado no arquivo de configuração.

A Fig. 27 apresenta os objetos renderizados com a opção ***solid*** selecionada. Os objetos por padrão são renderizados desta forma. Onde, diferentemente das outras opções, toda a superfície do objeto é tonalizada.

Figura 27 - Polygon Mode: Solid.

Outra opção disponível para o mesmo item é a visualização do objeto em ***wireframe***. Nesta representação os objetos são apresentados com a tonalização apenas das arestas das malhas de polígonos, como visto na Fig. 28.

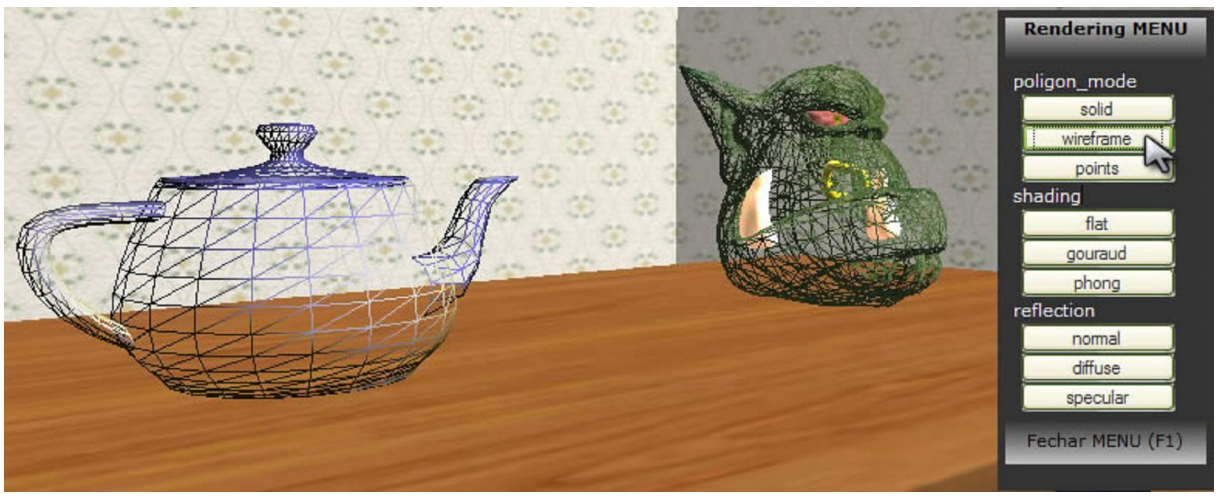


Figura 28 - Polygon Mode: Wireframe.

A última possibilidade de escolha para o *polygon_mode* é a opção ***points***. Os objetos que recebem esta transformação são tonalizados apenas nos vértices das malhas de polígonos. Fazendo com que possuam a aparência pontilhada mostrada na Fig. 29.

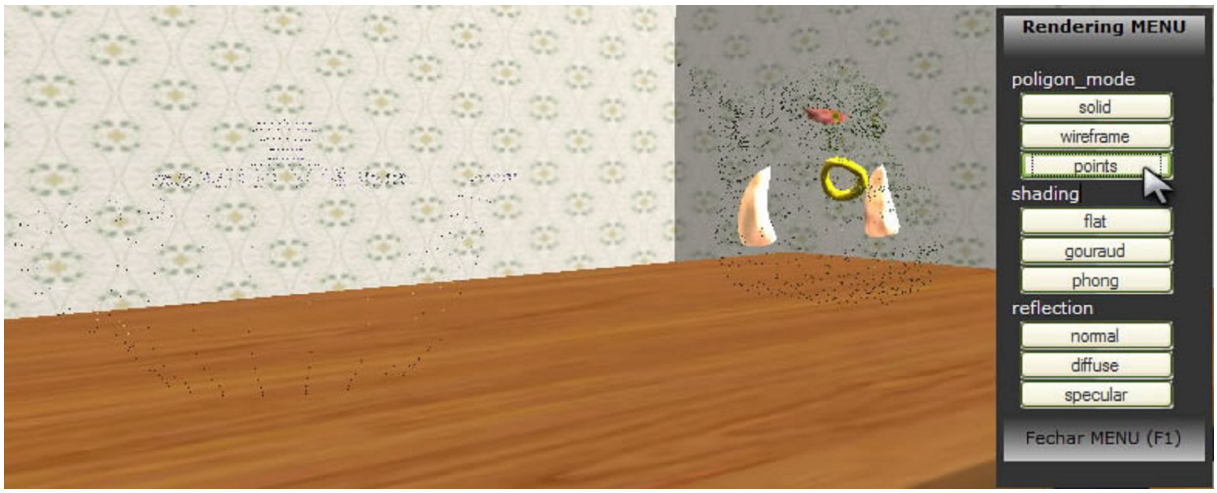


Figura 29 - Polygon Mode: Points.

6.1.2 *shading*

Outra característica dos objetos que pode ser modificada é o tipo de tonalização. Conhecido como *shading*, pode receber três valores: *flat*, *gouraud* e *phong*.

A Fig. 30 mostra a tonalização por ***flat***, conhecida como tonalização constante. Ela apresenta uma aparência facetada, onde toda a superfície de cada polígono que forma a malha recebe apenas uma cor.

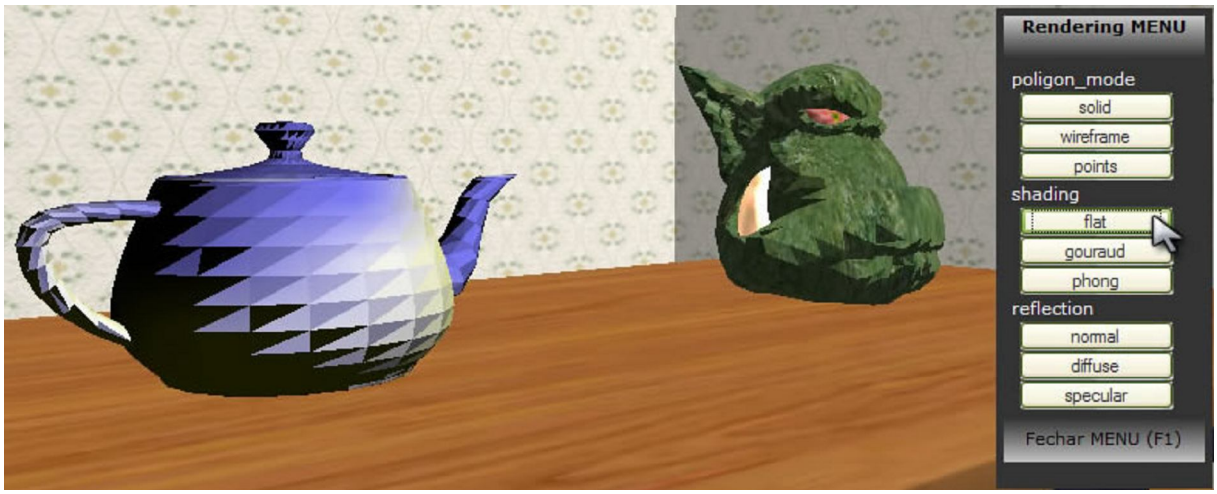


Figura 30 - *Shading: Flat*.

Além da tonalização *flat*, os objetos podem receber mais dois tipos de tonalização. Ambos aplicam técnicas que selecionam a cor de cada pixel baseados em um tipo de interpolação.

A Fig. 31 mostra a tela do ambiente aplicando-se a tonalização por ***gouraud***, esta aparenta uma superfície mais suave. Sendo que as cores dos pontos

ao longo da cada polígono é interpolada a partir do cálculo da cor dos vértices da malha.

Figura 31 - Shading: Gouraud.

A tonalização que apresenta os melhores resultados na maioria das vezes é com a utilização de ***phong***. Porém, este necessita de mais poder computacional para a realização os cálculos.

Para encontrar os resultados demonstrados na Fig. 32, o algoritmo de *phong* calcula as normais dos vértices e interpola essas normais. Logo após, calcula-se a intensidade do pixel para os pontos de superfície.

Figura 32 - Shading: Phong.

6.1.3 *reflection*

Assim como os demais itens do menu de tonalização, as opções atribuídas ao tipo de reflexão dos objetos afetam as características dos próprios objetos. Cada

um dos três botões a disposição no item *reflection*, atribuem valores pré-definidos de reflexão ambiente, difusa e especular aos objetos.

No tipo de reflexão ***normal***, visualizado na Fig. 33, os objetos recebem valores moderados de reflexão especular e difusa. Isto torna o objeto mais próximo da realidade, onde eles não possuem superfícies idealmente difusas e idealmente especulares.

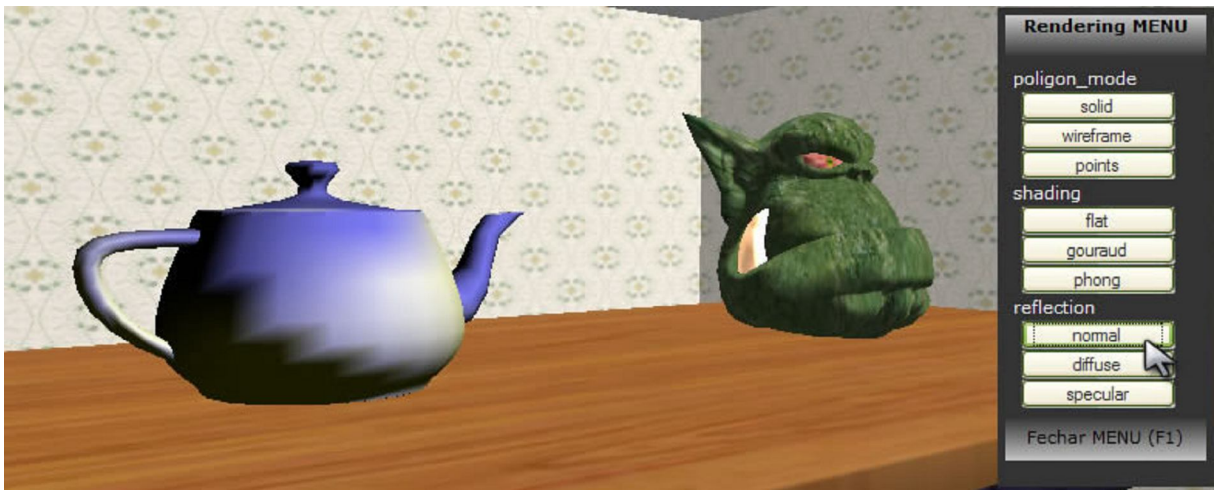


Figura 33 - Reflection: Normal.

Já no tipo de reflexão ***diffuse***, os objetos recebem valores altos de reflexão difusa e baixos de reflexão especular. Superfícies, como a da Fig. 34, que não recebem reflexão especular são foscas, fazendo com que o usuário perceba os objetos como sendo ásperos.

Figura 34 - Reflection: Diffuse.

Diferentemente da reflexão do tipo *diffuse*, objetos que possuem um alto índice de reflexão especular e baixo índice de reflexão difusa, possuem o tipo de reflexão ***specular*** como os da Fig. 35.

Figura 35 - Reflection: Specular.

6.2 Iluminação

O segundo menu criado neste sistema preocupa-se em demonstrar características de iluminação do ambiente. As opções deste menu não modificam as características dos objetos. Essas modificam as características das fontes de luz, que interferem na renderização de praticamente todo o ambiente.

Para possibilitar as interações com o sistema de iluminação, foi inserida na cena uma fonte de luz com seu posicionamento em frente, a cima e a esquerda dos objetos.

6.2.1 *light_type*

O tipo de luz é um dos itens do menu de iluminação. Ele é responsável por permitir a escolha entre três alternativas para o tipo da fonte de luz. Este item modifica diretamente as características da fonte de luz presente no ambiente.

A luz pontual apresentada na Fig. 36, ou simplesmente ***point***, é o tipo de luz mais comum. Este tipo simula algo como uma lâmpada incandescente, onde os raios de luz são emitidos a partir de um ponto em todas as direções.

Figura 36 - Light Type: Point.

A iluminação que simula uma lanterna é chamada de ***spotlight***. Como pode ser visualizada na Fig. 37, ela emite raios em uma direção obedecendo ao ângulo configurado para o feixe de luz.

Figura 37 - Light Type: Spotlight.

O ultimo tipo de luz é o ***directional***, demonstrado na Fig. 38. Este cria uma luz em uma direção que ilumina uma grande área, similar ao que ocorre com a luz do sol ou da lua.

Figura 38 - Light Type: Directional.

6.2.2 *shadow_type*

Outra funcionalidade importante é o tipo de sombra. As sombras dividem-se em três tipos, demonstrados a seguir:

Existe um tipo de sombra, chamado ***texture_modulative***, que pode ser visualizado na Fig. 39.

Figura 39 - Shadow Type: Texture Modulative.

O ***stencil_modulative*** (Fig. 40) apresenta sombras de melhor qualidade que o anterior. Aconselha-se bastante sua utilização pelo desempenho e qualidade satisfatórios.

Figura 40 - Shadow Type: Stencil Modulative.

O terceiro tipo é o mais custoso computacionalmente, pois trata cada fonte de luz da cena separadamente. Mostrado na Fig. 41, ele é chamado de ***stencil_additive***.

Figura 41 - Shadow Type: Stencil Additive.

6.2.3 *ambient light*

Por fim, o mais simples dos modelos de iluminação é a luz ambiente. De acordo com os valores atribuídos a ela, a cena inteira torna-se mais clara ou mais escura.

Há duas possibilidades de valores predefinidos para a luz ambiente. Na primeira delas ela é colocada em ***On***, onde recebe valores de luz ambiente altos (Fig. 42).

Figura 42 - Ambient Light: On.

Ou ainda, pode receber valores baixos de luz ambiente e ficar no estado ***Off***. Desta forma, o cenário apresenta um ambiente sombrio como na Fig. 43.

Figura 43 - Ambient Light: Off.

7 CONCLUSÕES E DISCUSSÃO

Os capítulos 5 e 6 procuraram explicar o ambiente criado e cada uma das suas funcionalidades. Neste capítulo serão discutidos os resultados encontrados e algumas modificações para trabalhos futuros serão propostas.

Realizou-se uma rápida busca por ambientes similares na internet e não foram encontrados sistemas com a mesma finalidade que possam ser comparados com o desenvolvido neste trabalho.

7.1 Funcionalidades bem sucedidas

A maioria das funcionalidades e técnicas desenvolvidas apresentaram resultados satisfatórios. Entre as bem sucedidas está o item 6.1.1 (*polygon_mode*). Esta modificação aplicada aos objetos é bem visualizada e percebem-se as modificações em seus três estados: *solid*, *wireframe* e *points*.

Exceto pela grande semelhança dos tipos de tonalização *gouraud* e *phong*, o item 6.1.2 (*shading*) também apresentou diferenças bastante evidentes do tipo *flat* com os demais tipos.

Já no tipo de reflexão (item 6.1.3 - *reflection*), a chaleira não permitiu resultados melhores. Porém, o foco importante desta modificação, que era permitir a percepção das diferenças de reflexão, apresentou resultados bem interessantes nas comparações dos tipos: *normal*, *diffuse* e *specular*.

Nas outras transformações, do menu de iluminação, também foi possível entender as diferenças entre os tipos de iluminação possíveis. O light_type (item 6.2.1) demonstra claramente as diferentes fontes de luz entre *point*, *spotlight* e *directional*.

A funcionalidade tipo de sombra, ou *shadow_type* (item 6.2.2), pode ser bem diferenciada apenas no objeto *Ogre*, já que esta funcionalidade não surtiu efeito no

objeto chaleira. Os três tipos de sombra (*texture_modulative*, *stencil_modulative* e *stencil_additive*) mostraram-se bem definidos.

E por fim, a luz ambiente (item 6.2.3 - *ambient light*) mostra perfeitamente as modificações entre *On* e *Off*.

7.2 Problemas encontrados

Por outro lado, Algumas funcionalidades não obtiveram os resultados esperados ou realizaram modificações em desacordo com o previsto pelas funcionalidades.

Chaleira

Uma das técnicas que apresentaram uma visualização inesperada foi a mudança de cor na chaleira quando aplica-se o tipo de sombra *stencil_additive*. Este problema pode ser percebido na Fig. 41. Essa irregularidade torna-se ainda mais estranha, quando se percebe que neste caso a interação não modifica as propriedades dos objetos, apenas o tipo de sombra do cenário.

A mudança de coloração inesperada deste objeto ocorre também quando se aplica o tipo de reflexão difusa (Fig. 34). Assim como ocorre na sombra do tipo *stencil_additive*, neste tipo de reflexão é possível observar uma aparência amarelada do objeto. Sendo que o objeto foi tonalizado com uma cor azul.

Especulou-se que estes problemas devem ocorrer pela presença do objeto ogro próximo a chaleira. Esta presença faria com que a chaleira recebesse algum tipo de reflexão que modificasse sua tonalização. Visto que a chaleira sozinha no cenário não apresentou estas anomalias.

Além dos problemas de tonalização apresentados, a chaleira não recebe nenhuma das modificações de sombra passiveis de serem aplicadas ao cenário. Estes são visualizados nas figuras referentes ao tópico 6.2.2 do capítulo anterior.

Acredita-se que o problema da aplicação de sombras advindas da chaleira ocorre pela forma com que o objeto foi exportado. Sendo que, como dito anteriormente, este objeto foi disponibilizado na internet por alguém que pode não ter se preocupado com as características de importação do mesmo.

Este problema foi minimizado pela presença do objeto ogro. Nele podem-se visualizar claramente todas as modificações do tipo sombra que não estão visíveis na chaleira.

Tonalização

Outra modificação que rendeu resultados quase imperceptíveis foi a renderização dos objetos com tonalização *gouraud* e *phong*. A interação com estes objetos fez com que a tonalização por *phong*, que em tese deveria apresentar uma melhoria considerável, não surtisse o efeito esperado.

7.3 Trabalhos futuros

Solução dos problemas

Uma série de trabalhos futuros podem ser realizados a partir do ambiente criado. Entre eles, um estudo importante e necessário seria a respeito dos problemas ocorridos com a chaleira.

Este estudo procuraria solucionar os problemas mencionados anteriormente, como mudanças inesperadas na tonalização com a aplicação de algumas funcionalidades e a inexistência de sombras naquele objeto.

Outro problema que não foi tratado neste trabalho, por não ser julgado tão importante, é a detecção de colisão com o cenário. Da forma como está implementado o sistema, é possível atravessar paredes com a movimentação da câmera.

Aprimoramento da cena

Outro trabalho que pode ser desenvolvido é a melhoria na navegação e no cenário do sistema. Mudanças do cenário seriam interessantes para tornar o ambiente mais atrativo para os usuários.

Uma das formas de aperfeiçoar a cena seria com a inserção de novos objetos para montar um contexto ao ambiente. E ainda, uma expansão dos limites da cena, de forma que o usuário não se restrinja apenas a movimentar-se em uma sala, mas em uma casa inteira.

Novas funcionalidades

Aliado a expansão do cenário, seria interessante um aumento do número de funcionalidades. Onde se pode selecionar qualquer das funcionalidades que representam as técnicas aprendidas em disciplinas de computação gráfica.

Com a casa toda criada, ficaria muito interessante se cada cômodo dela permitisse a aplicação de funcionalidades diferentes.

Algumas funcionalidades que se mostram necessárias para a melhoria do ambiente como um todo são:

- A criação de um painel que mostre os valores que foram recebidos em cada modificação. Isso faria com que o usuário estivesse sempre a par das modificações que estão em vigor nos objetos;
- Um botão no menu de interação que permita o reset do sistema, aplicando os valores padrão em todas as funcionalidades. Dessa forma, o usuário poderia voltar ao estado inicial do sistema e desfazer todas as interações sem ter q sair do ambiente;
- Permitir a utilização de áudio no ambiente, onde a interação se tornaria mais interessante com a produção de algum efeito sonoro;
- Outra possibilidade seria disponibilizar outros objetos para receber as transformações. O usuário poderia assim, selecionar o objeto que irá receber cada interação.

Referências

AUKSTAKALNIS, Steve; BLATNER David. **Silicon Mirage; The Art and Science of Virtual Reality.** Peachpit Press Berkeley, CA, USA, 1992.

BATTAIOLA, André Luiz. **Apostila do Curso de Computação Gráfica**. (Adaptado por: Prof. Dr. José Hiroki Saito). Departamento de Computação. UFSCar, 1999.

CLUA, E. W. G.; BITTENCOURT, J. R. **Desenvolvimento de Jogos 3D: Concepção, Design e Programação**. XV Congresso da Sociedade Brasileira da Computação. A Universidade da Computação: Um Agente de inovação e Conhecimento. Capítulo 3, pg 1350, 2005.

CRYSTAL SPACE. **Crystal Space 3D**. Disponível em: <http://www.crystalspace3d.org>. Acesso em: 20 jun. 2008.

DEVMASTER, 2008. **DevMaster.net - Your source for game development**. Disponível em: <http://www.devmaster.net/>. Acesso em: 24 jul. 2008.

ELIAS Project. **Open Source & Low Cost Game Engines**. 2005. Disponível em: <http://ludocraft.oulu.fi/elias/>. Acesso em: 19 mai. 2008.

FILHO, Antonio Castelo. **Modelagem Geométrica: Representação e Manipulação de Objetos Geométricos Utilizando o Computador**. ICMC/USP. São Carlos. Brasil, 1998.

FOLEY, James et al. **Introduction to Computer Graphics**. USA. Addison-Wesley. 1993. 557p.

GOVIL-PAI, Shalini. *Principles of Computer Graphics – Theory and Practice Using OpenGL and Maya*. Sunnyvale, CA, USA Capítulo 6, Springer, 2004.

GOWEN, Lan. **Irrlicht**. 2004. Disponível em: <http://freshmeat.net/articles/view/1182/irrlichht>. Acesso em: 01 jun. 2008.

HEARN, Donald; BAKER M. Pauline. **Computer Graphics**, C Version. 2.Ed. New Jersey, USA: Prentice Hall, 1997. 652p.

IRADOS. **Irados Ogre**. Disponível em: <http://ogre.irados.org>. Acesso em: 16 out. 2008.

IRRLICHT. **A Free Open Source 3D Engine**. Disponível em: <http://irrlicht.sourceforge.net/>. Acesso em: 19 mai. 2008.

IRRLICHT DOC. **Irrlicht Engine 1.4 API Documentation**. Disponível em: <http://irrlicht.sourceforge.net/docu/index.html>. Acesso em: 19 mai. 2008.

ISDALE, Jerry. **What Is Virtual Reality? A Web-Based Introduction.** Version 4, Draft 1, 1998.

JUNKER Gregory. **Pro OGRE 3D Programming**. Apress. 2006. 288 p.

KELNER J. Et Al. - **Modelagem Virtual Urbana: Perspectiva Histórica e Estudo de Caso,** Universidade Federal de Pernambuco, Recife, 2002.

KIRNER, Claudio; SISCOUTTO, Robson. **Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações**. Livro do Pré-Simpósio IX Symposium on Virtual and Augmented Reality, Petrópolis - RJ, 2007.

KIRNER Claudio, TORI Romero. **Realidade Virtual Conceitos e Tendências**. Livro do Pré-Simpósio VII Symposium on Virtual Reality. São Paulo, 2004.

KOT, Blazej. **Information Visualisation utilizing 3D Computer Game Engines Case Study: A source code comprehension tool**. 2005, Department of Computer Science, University of Auckland, pg 7.

LEWIS, Michael; JACOBSON, Jeffrey. **Game Engines in Scientific Research Communications of the ACM**. Vol. 45. No. 1, 2002.

LIMA, João Paulo Silva do Monte et. al.. **Port of the OGRE 3D Engine to the Pocket PC Platform**. 2006.

MADEIRA, Charles Andryê Galvão. Dissertação de Mestrado. **FORGE V8: Um framework para o desenvolvimento de jogos de computador e aplicações multimídia**. Universidade Federal de Pernambuco. Centro de Informática. Pós-graduação em Ciência da Computação. Recife/PE, 2001.

MICROSOFT. **Microsoft Corporation**. Disponível em: <http://www.microsoft.com/>. Acesso em: 31 out. 2008.

MOTORES DE JUEGOS. **Modelado Multirresolución em Juegos por Ordenador**. Grupo de Informática Gráfica. Universitat Jaume. Disponível em: <http://iiia.udg.edu/>. Acesso em: 26 mai. 2008.

NETTO, Antonio Valério. **Prototipação de um torno CNC utilizando realidade virtual**. USP, Dissertação de Mestrado, São Carlos, 1998.

NETTO J., MACHADO L., MORAES R. **Um Estudo Comparativo de Ferramentas para a Criação de Jogos Educacionais Baseados em Realidade Virtual**. Universidade Federal da Paraíba, João Pessoa, 2006.

OGRE 3D. **Open Source Graphics Engine**. Disponível em: <http://www.ogre3d.org/>. Acesso em: 19 mai. 2008.

OGRE MANUAL. **Manual v1.4.7 ('Eihort')**. Disponível em: <http://www.ogre3d.org/docs/manual/>. Acesso em: 19 mai. 2008.

PANDA 3D. **Panda3D - Free 3D Engine** - Disponível em: <http://www.panda3d.org/>. Acesso em: 23 jun. 2008.

PLANE SHIFT. **Plane Shift - A 3D Fantasy MMORPG**. Disponível em: <http://www.planeshift.it/>. Acesso em: 20 jun. 2008.

PROSJEKT. **FFI Prosjecter**. Disponível em: <http://prosjekt.ffi.no/unik-4660/>. Acesso em: 29 out. 2008.

QUIROZ Andrés et al.**Reconstrucción virtual - del mito emberá de la creación del água utilizando agentes autônomos**. REVISTA Universidad EAFIT Vol. 41, No. 140, 2005, pg. 62.

SANTOS, Juliano Soares, Dissertação de Mestrado. **Nuvens virtuais como exemplo de técnicas de jogos para gráficos tridimensionais em tempo real** – UFSC. Mestrado em Engenharia de Produção. Florianópolis/SC, 2004

SUTHERLAND, I. E. **The Ultimate Display**. Proceedings of IFIPS Congress 1965, New York, 1965, Vol. 2, pp. 506-508.

THE3DSTUDIO. **3D models**. Disponível em: <http://www.the3dstudio.com>. Acesso em: 16 out. 2008.

TRAINA, Agma Juci Machado; OLIVEIRA Maria Cristina Ferreira. **Apostila de Computação Gráfica**. ICMC/USP, 2006.

ZDNET. **Tech News, Blogs and White Papers for IT Professionals** . Disponível em: < http://www.zdnet.com/>. Acesso em: 22 nov. 2008.

WARD Jeff. **What Is a Game Engine.** Disponível em: <http://www.gamecareerguide.com/features/529/what_is_a_game_engine.php>. Acesso em: 3 mai. 2008.

WIKIPEDIA CG. **Computação Gráfica**. Disponível em: <http://pt.wikipedia.org/> - Acesso em: 11 mai. 2008.

WIKIPEDIA IRRLICHT - **IRRLICHT** - Disponível em: <http://pt.wikipedia.org/wiki/Irrlicht>. Acesso em: 31 mai. 2008.