FACULDADE DO ESPÍRITO SANTO - UNES

NATALI BARBOZA CARALO

INSERÇÃO DE OBJETOS EM TERRENOS VIRTUAIS UTILIZANDO BILLBOARDS

CACHOEIRO DE ITAPEMIRIM – ES
2010

NATALI BARBOZA CARALO

INSERÇÃO DE OBJETOS EM TERRENOS VIRTUAIS UTILIZANDO BILLBOARDS

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Faculdade do Espírito Santo - UNES, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Me. Ricardo Maroquio Bernardo

CACHOEIRO DE ITAPEMIRIM – ES
2010

NATALI BARBOZA CARALO

INSERÇÃO DE OBJETOS EM TERRENOS VIRTUAIS UTILIZANDO BILLBOARDS

Monografia apresentada ao curso de Bacharelado em Sistemas de Informação da Faculdade do Espírito Santo - UNES, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em 13 de dezembro de 2010.

COMISSÃO EXAMINADORA

_____

Prof. Me. Ricardo Maroquio Bernardo
Orientador

_____

Prof. Me. Jocimar Fernandes

_____

Prof. Esp. Bernardo Casotti Vidaurre Avilla Paldês

Dedico à minha mãe Ilda,
por me ensinar a valorizar
o que é realmente importante.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, pela minha vida e de todas as pessoas que amo. Obrigada Senhor por me conceder discernimento nos momentos que necessitei e força para superar os obstáculos encontrados durante a jornada desta graduação.

Aos meus pais, Ilda e Natalino, pela educação, dedicação, carinho e amor que sempre recebi. Aos meus irmãos, Marcelo e Jorge, que mesmo longe, se fazem tão presentes. E, principalmente, à minha mãe, verdadeira guerreira que me ensinou os reais valores da vida. Mãe obrigada por toda a paciência, apoio, compreensão e ajuda constante para tudo.

Ao meu namorado Anderson, que nos momentos difíceis sempre me fortaleceu e me motivou com os seus sábios conselhos e seu infindável otimismo. Obrigada pelo amor, amizade, companheirismo, carinho e incentivo. Agradeço por preencher a minha vida de alegrias, por entender e suportar as minhas mudanças de humor e pelo apoio incondicional durante todo esse caminho (neoqeav).

Aos professores que lecionaram cada disciplina deste curso, em especial ao meu orientador, professor Ricardo Maroquio Bernardo, sou grata por acreditar em mim, pela paciência, incentivo e apoio no desenvolvimento deste trabalho. Obrigada por ser além de professor, um grande amigo. Que Deus o ilumine.

A todos que estudaram comigo, entre eles os amigos Marcelo Costalonga, obrigada pela paciência de irmão mais velho e pelos conselhos que guardarei comigo para sempre, à Carla e Keilla, por compartilharmos tantos bons momentos juntas, inclusive os estudos intensivos antes de cada prova.

Enfim, um agradecimento a todos que, direta ou indiretamente, auxiliaram na minha graduação.

RESUMO

A inserção de objetos em ambientes virtuais é essencial para aumentar o realismo das cenas. Atualmente, aplicações interativas, como realidade virtual e jogos, tornaram-se frequentes e tais cenas realistas são de grande importância para estas aplicações. O presente trabalho introduz o conceito de *billboard*, que é uma técnica eficiente para representar objetos complexos utilizando geometria simples, assim como, apresenta conceitos básicos da Computação Gráfica e a aplicação destes conceitos utilizando a biblioteca gráfica OpenGL. Ao final, é apresentado um estudo de caso, que mostra os algoritmos utilizados para a aplicação da técnica *billboard*.

Palavras-chave: Computação Gráfica. OpenGL. *Billboard*. Renderização de objetos. Simulação. Jogos 3D.

ABSTRACT

The insertion of objects in virtual environments is essential to increase the realism of the scenes. Nowadays, interactive applications as virtual reality and games became frequent and such realistic scenes are of great importance for these applications. This paper introduces the concept of billboard: an efficient technique to represent complex objects using simple geometry. We also present basic concepts of computer graphics and the application of these concepts using the OpenGL graphics library. Finally, a case study is presented, that shows the algorithms used for the application of the technique billboard.

Keywords: Computer Graphics. OpenGL. Billboard. Rendering objects. Simulation. 3D games.

LISTA DE ILUSTRAÇÕES

LISTA DE EQUAÇÕES

LISTA DE ALGORITMOS

LISTA DE SÍMBOLOS

β	Ângulo para rotacionar o objeto em função do eixo x
δ	Ângulo para rotacionar o objeto em função do eixo y
α	Ângulo para rotacionar o objeto em função do eixo z
θ	Ângulo da matriz de rotação

LISTA DE SIGLAS

API	*Application Programming Interface[*
BMP	Extensão do arquivo *bitmap*
CAD	*Computer Aided Design*
MIT	*Massachusetts Institute of Technology*
OPENGL	*Open Graphics Library*
PIXEL	*Picture Element*
RGB	*Red, Green e Blue*
RGBA	*Red, Green, Blue e Alpha*
SAGE	*Semi-Automatic Ground Environment*
2D	Bidimensional
3D	Tridimensional

SUMÁRIO

1. INTRODUÇÃO

A Computação Gráfica é a área que estuda a transformação de dados em imagens, cuja principal intenção é a recriação do mundo real através de fórmulas matemáticas e complexos algoritmos.

No mercado atual a Computação Gráfica está presente em jogos eletrônicos, efeitos especiais em filmes, simulação de viagens espaciais, projetos de equipamentos modernos, publicidade, e até mesmo, na medicina com a recriação de órgãos virtuais, exames e nas terapias de fobia. (AZEVEDO; CONCI, 2003).

Se você puder imaginar algo, isso pode ser gerado com a Computação Gráfica. A Computação Gráfica está a um passo de um mundo novo repleto de aplicações, ainda desconhecidas, e muitas oportunidades de trabalho. Toda essa realidade impulsiona a Computação Gráfica para o desenvolvimento de diversas aplicações tridimensionais (3D).

Os cenários tridimensionais virtuais estão cada vez mais realistas, devido à evolução do poder de processamento e das diversas técnicas de aplicação de efeitos visuais com alto desempenho como cores, textura, mapeamento de texturas, transparência e iluminação.

Cenários naturais que fazem uso de vegetação, tais como: árvores, plantas, arbustos são triviais e muito importantes, no entanto, a representação de vegetação, mais especificamente árvores, apresenta alguns problemas. Sendo assim, para alcançar uma representação realista torna-se necessário a utilização de uma grande quantidade de polígonos para modelar uma árvore com todos os detalhes como, caule, galhos e folhas. Um cenário natural geralmente contém diversas árvores, plantas e arbustos. Com isso, a renderização do grande número de polígonos utilizados acarreta um alto custo computacional exigindo um poder de processamento maior do que existe atualmente, prejudicando assim o realismo do cenário. Por esse motivo, existem várias técnicas que reduzem a complexidade

geométrica do modelo apresentado, utilizando geometria simples para representar objetos de uma forma geral.

Este trabalho propõe um estudo e execução de uma técnica chamada *billboard,* que é utilizada para inserir objetos em ambientes virtuais, com o objetivo de aumentar o realismo do cenário e diminuir o custo de processamento computacional.

1.1. Histórico da Computação Gráfica

A arte rupestre foi a primeira manifestação gráfica do homem na luta, para se comunicar e revelar sua evolução. Com o passar dos anos, surgiram os computadores, que, a princípio, não geravam imagens gráficas e os resultados eram emitidos através de caracteres alfanuméricos.

Azevedo e Conci (2003) afirmam que existe um consenso entre os pesquisadores, de que o primeiro computador a possuir recursos gráficos de visualização de dados numéricos foi o *Whirlwind I*, desenvolvido pelo MIT (*Massachussets Institute of Technology*), em 1950, com finalidades acadêmicas e militares.

O primeiro sistema de aplicação gráfica, o SAGE (*Semi-Automatic Ground Environment*), surgiu em 1955 e sua função era “[...] converter informações do radar em imagens para monitoramento e controle de vôos [...]” (MANSSOUR; COHEN, 2006, p. 2).

Em 1959, surgiu o termo *Computer Graphics,* criado por Verne Hudson. De acordo com Azevedo e Conci (2003), em 1962, surgiu uma das mais importantes publicações da computação gráfica de todos os tempos, através da tese de Ivan Sutherland (*Sketchpad – A Man-Machine Graphical Communication System*), o famoso pesquisador apresentou ao mundo o primeiro sistema gráfico interativo.

Essa publicação chamou a atenção das indústrias automobilísticas e aeroespaciais americanas. Assim, os conceitos de estruturação de dados, bem como o núcleo da noção de computação gráfica interativa, levaram a *General Motors* a desenvolver em 1965 o precursor dos programas de CAD (*Computer Aided Design*).

Segundo Manssour e Cohen (2006), na década de 1970 foram desenvolvidas novas técnicas e algoritmos que são utilizados até hoje, assim como o algoritmo de *z-buffer*. Além disso, o surgimento da tecnologia de circuitos integrados permitiu a popularização dos computadores pessoais, disseminando os aplicativos prontos e integrados, como os editores gráficos. Também foi nesta década o lançamento do primeiro computador com interface visual, predecessor do *Macintosh*.

Com a evolução das imagens geradas pela Computação Gráfica, baseado em Azevedo e Conci (2003), na década de 1980 surgiram diversas técnicas novas de iluminação global, como o *ray-tracing* (em 1980) e a radiosidade (em 1984), que aproximam as imagens geradas pelo computador ao fotorrealismo.

Figura 1 - O *ray-tracing* pode alcançar um alto grau de realismo visual
Fonte: SANTOS, M. P. (2005).

Azevedo e Conci (2003) afirmam, ainda, que a década de 1990 marcou o amadurecimento da Computação Gráfica com imagens impressionantes, como no filme *Jurassic Park*, em 1993. O filme marca a perfeição do fotorrealismo, nas cenas de movimentos dos dinossauros. Como também o filme *Terminator 2,* com a utilização de um personagem computadorizado, e *Toy Story*, o primeiro longa

metragem 3D, em 1995. Em 1992, na área de Sistemas, surge a biblioteca gráfica *OpenGL* e as primeiras placas gráficas para PC da NVIDIA, em 1999.

Assim, a evolução dos dispositivos gráficos contribuiu para o desenvolvimento da Computação Gráfica, permitindo a geração de imagens com grande realismo em tempo real.

1.2. Objetivos

O presente trabalho tem como objetivo geral apresentar a inserção de objetos em ambientes virtuais, utilizando a técnica *billboard*, para aumentar o realismo do ambiente.

Como objetivos específicos têm-se:

a) Compreender o algoritmo de criação de terrenos a partir de mapas de altura;
b) Compreender os algoritmos utilizados para aplicação de textura em terrenos;
c) Analisar as técnicas de inserção de objetos em ambientes virtuais;
d) Compreender e implementar a inserção de objetos em ambientes virtuais utilizando a técnica *billboard*.

1.3. Justificativa

O interesse pelo assunto surgiu durante a execução de um trabalho, da disciplina Computação Gráfica estudada no 7º período do curso de Sistemas de Informação.

Surgiu então o desafio de dar continuidade a um projeto fascinante e promissor, que é uma grande oportunidade de adquirir conhecimento na área que, atualmente, é supervalorizada e carente de mão-de-obra. Sendo assim, é uma contribuição para trabalhos futuros, já que existem poucas publicações sobre o assunto na Língua

Portuguesa, o que valoriza e justifica ainda mais a elaboração de um trabalho abordando esse assunto.

1.4. Metodologia

Para o desenvolvimento deste trabalho foram utilizadas diversas fontes de pesquisa, como livros, artigos, apostilas, tutoriais, trabalhos acadêmicos, monografias, dissertações e teses de mestrado, publicados na internet.

Como ferramenta para desenvolvimento foi utilizado o Delphi 7, pois esta linguagem foi estudada durante as disciplinas de programação do Curso Sistemas de Informação, bem como na disciplina de Computação Gráfica, e a biblioteca *OpenGL*. Segundo Jacobs (1999), a biblioteca *OpenGL* consiste de uma API (Interface de Programação de Aplicação) para criação de aplicações gráficas.

1.5. Estrutura do trabalho

Todo o desenvolvimento deste estudo, inserção de objetos em terrenos virtuais utilizando *billboards,* está distribuído em seis capítulos, sendo organizados da seguinte forma:

O Capítulo 1 apresenta a introdução ao tema proposto por este estudo, um breve histórico da Computação Gráfica e os objetivos deste trabalho. Descreve-se ainda a justificativa para a realização deste estudo, a metodologia utilizada para o seu desenvolvimento e os detalhes da organização do mesmo.

O Capítulo 2 mostra um breve relato dos trabalhos anteriores, que fazem parte do mesmo projeto que o atual estudo, ilustrando o resultado das respectivas

implementações. Sendo realizada neste capítulo uma análise dos trabalhos pesquisados na literatura, que foram selecionados pela proximidade com esse trabalho.

O Capítulo 3 descreve os assuntos específicos, que contribuíram direta ou indiretamente para a realização deste estudo. São citados os principais conceitos da Computação Gráfica e da biblioteca *OpenGL*, bem como os aspectos relevantes e a sintaxe dos comandos desta API.

O Capítulo 4 aborda o tema estudado, sendo, portanto, o ápice deste trabalho. Neste capítulo são apresentados os conceitos da técnica *billboard*, que permite aumentar o realismo dos ambientes virtuais através da inserção de objetos. Em seguida, é apresentado um estudo de caso com a implementação da técnica *billboard* e a descrição dos algoritmos utilizados.

O Capítulo 5 descreve as conclusões deste trabalho, finalizando o estudo sobre inserção de objetos em terrenos virtuais utilizando *billboards*.

O Capítulo 6 cita as referências utilizadas como fundamentação teórica, para o desenvolvimento deste trabalho.

2. REVISÃO DE LITERATURA

Este trabalho é continuidade de um grande projeto, composto diretamente pelas monografias de Fabiano Jussim Marchezi (2007) e Marcelo José dos Santos (2008), e indiretamente pelas monografias de Rafael Ribeiro Rocha (2008) e Horácio Luiz da Silva (2008). Este capítulo realiza uma breve apresentação dos trabalhos supracitados, mostrando o resultado de cada implementação. Apresenta-se, também, uma análise dos materiais pesquisados que têm relação com o atual estudo.

2.1. Trabalhos anteriores

A monografia de Fabiano Jussim Marchezi (2007), cujo tema é “Estudo Sobre Renderização de Terreno Para Jogos e Simuladores”, foi o alicerce para que outros trabalhos fossem realizados, com o objetivo de programar técnicas para tornar esse ambiente virtual mais realista.

Para construir o terreno, foram utilizados mapas de relevo ou *heightmap,* onde cada pixel da imagem monocromática representa um valor de altura, e o algoritmo de força bruta.

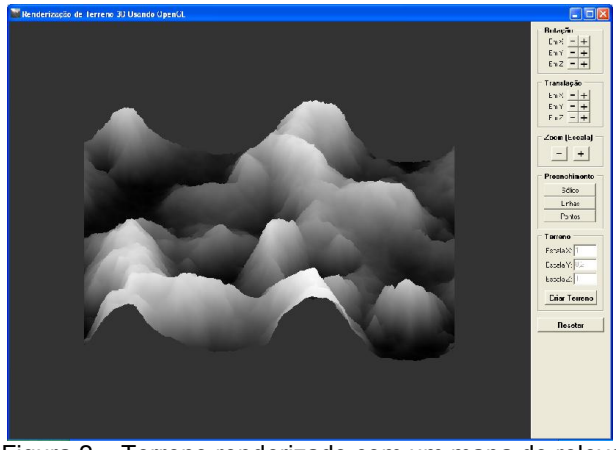

Figura 2 – Terreno renderizado com um mapa de relevo ou *heighmap*
Fonte: MARCHEZI (2007).

Marcelo José dos Santos (2008) desenvolveu a monografia, “Aplicação de Texturas em Terrenos Tridimensionais Virtuais Usando *OpenGL*”, que demonstrou como gerar uma textura e aplicá-la ao terreno desenvolvido por Marchezi (2007), de forma a aprimorar o aspecto visual do mesmo. A criação da textura que cobriu todo o terreno foi resultado da mistura de quatros texturas representadas abaixo.

Figura 3 – Texturas utilizadas para aplicação no terreno.
Fonte: SANTOS, M. J. (2008).

A textura aplicada ao terreno foi obtida por meio da mesclagem das quatro texturas representadas acima, utilizando a altura do respectivo ponto do mapa de relevo para determinar a participação de cada textura naquele ponto, respeitando alguns limites definidos para cada textura.

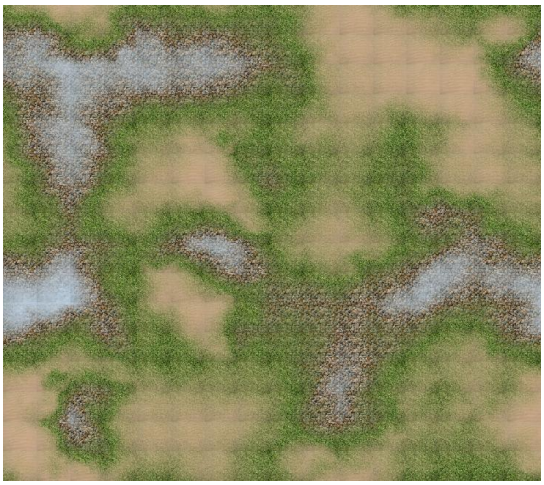

Figura 4 – Textura 2D gerada.
Fonte: SANTOS, M. J. (2008).

Depois da criação da textura heterogênea, a mesma foi aplicada ao terreno tridimensional, cuja variação depende da altura do ponto de aplicação da textura no relevo.

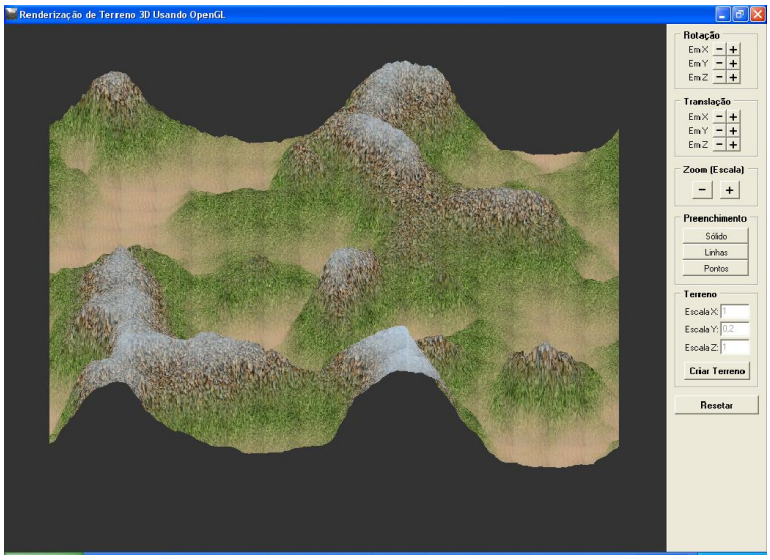

Figura 5 – Textura aplicada ao terreno tridimensional.
Fonte: SANTOS, M. J. (2008).

Além dos trabalhos supracitados, foram realizados outros separadamente, dos quais as técnicas ainda não foram inseridas no projeto, proporcionando oportunidades para o surgimento de trabalhos futuros.

Rafael Ribeiro Rocha (2008) realizou, em sua monografia, um estudo sobre a iluminação de ambientes tridimensionais virtuais, por meio da *OpenGL*, explicando os conceitos dos três tipos de iluminação disponíveis: iluminação ambiente, iluminação difusa e iluminação especular. Abaixo, seguem os resultados das implementações feitas por Rocha (2008).

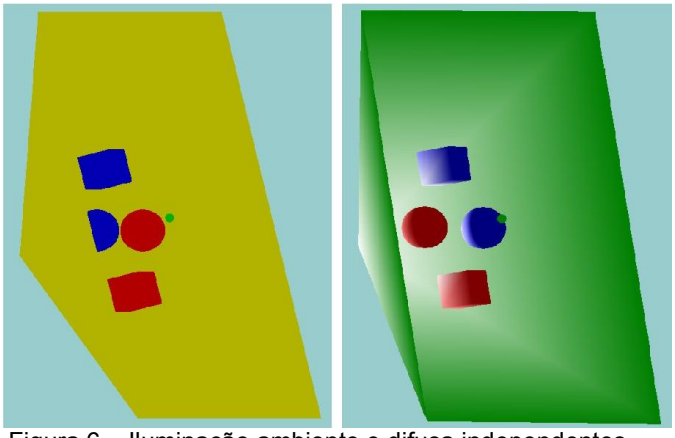

Figura 6 – Iluminação ambiente e difusa independentes.
Fonte: ROCHA (2008).

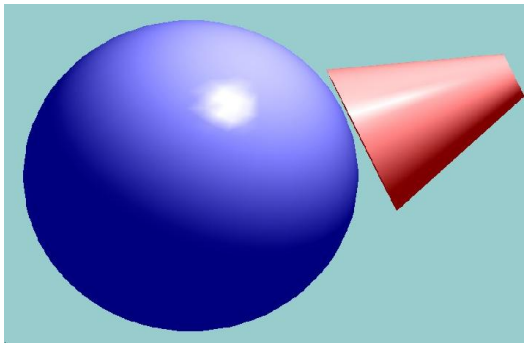

Figura 7 – Iluminação especular.
Fonte: ROCHA (2008).

A monografia de Horácio Luiz da Silva (2008), “Detecção de Colisão de Objetos Virtuais em Jogos Tridimensionais”, utilizou a técnica de volumes envolventes para melhorar o desempenho e encontrar um meio de utilizar detecção de colisão em grande escala e com rapidez. A idéia é envolver cada objeto da cena com uma primitiva geométrica, e ao se constatar que dois volumes envolventes não estão colidindo, pode-se afirmar que todas as primitivas (triângulos) também não estão.

O resultado da implementação do estudo de caso de Silva (2008) será demonstrado abaixo. Pode-se notar que a esfera muda de cor ao detectar a colisão.

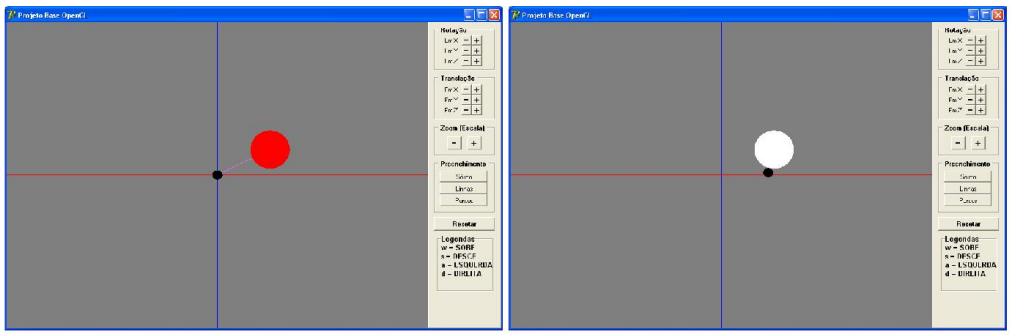

Figura 8 – Antes e depois da detecção da colisão.
Fonte: SILVA (2008)

Todas as técnicas apresentadas acima têm como objetivo principal proporcionar mais veracidade ao ambiente virtual, deixando-o mais próximo possível da realidade.

2.2. Trabalhos relacionados

Existem diversos trabalhos envolvendo inserção de objetos em ambientes virtuais, no entanto, esses trabalhos se restringem a inserir apenas vegetação no cenário e acabam delimitando, de maneira errônea, a utilização e aplicação das técnicas estudadas, como, por exemplo, a técnica *billboard*. Dentre os estudos pesquisados três deles foram selecionados e serão descritos a seguir.

No artigo, “Visualização de Terrenos Digitais Utilizando Técnicas Dependentes da Visão”, são apresentadas técnicas para melhorar a visualização interativa de terrenos digitais tridimensionais. Essas técnicas são aplicadas numa estrutura chamada *billboard,* com o objetivo de representar os diferentes tipos de vegetações. A leitura desse artigo foi proveitosa, pois os autores Savelli, Seixas e Montenegro (2007) apresentaram além do conceito de *billboard*, uma análise comparativa entre a utilização das técnicas *billboard* e *sprite*, relatando, assim, qual delas é mais custosa computacionalmente.

Este material demonstra a importância da utilização das técnicas dependentes da visão (ou *view-dependent*) para aumentar o realismo da cena, contribuindo para que o atual estudo tenha trabalhos futuros que abordem tal assunto. Em contrapartida, como esse artigo mostrou no final do trabalho os resultados das implementações das técnicas citadas, seria interessante se os autores tivessem ido além da parte teórica, especificando os passos utilizados para chegar ao resultado demonstrado.

O exame de qualificação de mestrado de Edgar Vilela Gadbem (2009), “Representação de Plantas para Renderização em Ambientes Interativos”, apresentou o conceito de *billboard*, os tipos de *billboards* existentes e suas aplicações, bem como algumas técnicas, que quando aplicadas aos *billboards* aumentam o realismo do ambiente. Este material foi importante para a elaboração deste estudo, pois demonstrou que renderizar uma árvore com polígonos pode ter um custo computacional altíssimo. Uma cena aberta, por exemplo, pode normalmente conter várias árvores, arbustos e outras plantas. Assim, o número de

polígonos da cena iria superar com facilidade o poder de processamento gráfico atual, de tal forma que essa simulação não poderia ser executada com uma taxa de quadros considerada interativa para uma aplicação. Gadbem (2009) explicou que a técnica *billboard* é usada atualmente por ser extremamente eficiente, pois com uma ou poucas imagens é possível apresentar uma grande quantidade de detalhes de uma planta utilizando pouquíssima geometria.

O artigo, "*A Survey of Computer Representations of Trees for Realistic and Efficient Rendering*", trouxe uma visão geral da técnica billboard e suas aplicações. Segundo Boudon, Meyer e Godin (2006), a reprodução dos cenários naturais, o mais realista possível, resulta em um dos principais objetivos da Computação Gráfica. Atualmente, *billboard* é a ferramenta mais comum para renderização de florestas em tempo real, graças ao seu baixo custo computacional. Ao analisar este artigo conclui-se que, a maior dificuldade na construção deste trabalho foi a superficialidade técnica dos materiais pesquisados, já que estes focam apenas o estudo estrutural das árvores, com grande ênfase na botânica, deixando de explicar dados técnicos importantes, como, por exemplo, mencionar em alto nível como a técnica deve ser implementada.

Apesar dos trabalhos relacionados terem como foco a inserção de vegetação em ambientes virtuais através de *billboard*, é importante ressaltar que essa técnica pode ser utilizada para inserir objetos complexos em geral, tendo assim uma ampla utilidade.

2.3. Considerações sobre o capítulo

Este capítulo apresentou um breve relato dos trabalhos anteriores, que compõem o projeto, no qual este estudo está inserido e ressaltou os pontos positivos e negativos dos principais materiais pesquisados, que auxiliaram na realização deste trabalho.

O capítulo seguinte descreverá os principais conceitos da Computação Gráfica e os aspectos relevantes da biblioteca gráfica *OpenGL*, descrevendo os comandos mais utilizados desta API.

3. A COMPUTAÇÃO GRÁFICA E A OPENGL

Para o desenvolvimento deste trabalho foi necessário adquirir conhecimentos mais específicos sobre Computação Gráfica e a biblioteca *OpenGL*. Este capítulo faz uma abordagem geral sobre tais assuntos e apresenta as rotinas da biblioteca *OpenGL*, que foram utilizadas na implementação do estudo de caso.

3.1. Conceitos

A Computação Gráfica preocupa-se com a manipulação e visualização de objetos reais ou imaginários. Para representação computacional desses objetos são utilizadas estruturas de dados, algoritmos e modelos matemáticos e físicos. De uma maneira simplista, o termo Computação Gráfica significa criar ou manipular imagens utilizando um computador.

De acordo com Cohen e Manssour (2006, p. 17), “A Computação Gráfica é uma área da Ciência da Computação que se dedica ao estudo e ao desenvolvimento de técnicas e algoritmos para a geração, manipulação e análise de imagens [...]”.

A Computação Gráfica, atualmente, é composta por três grandes subáreas: a Síntese de Imagens, o Processamento de Imagens e a Análise de Imagens (AZEVEDO; CONCI, 2003).

A Síntese de Imagens considera as representações visuais de objetos criados pelo computador, a partir das especificações geométricas e visuais de seus componentes. O Processamento de Imagens envolve o processamento das imagens na forma digital e suas transformações, para melhorar ou realçar suas características visuais. A Análise de Imagens considera as imagens digitais e as analisa para obtenção de características desejadas, como, por exemplo, a especificação dos componentes de uma imagem, a partir de sua representação visual.

Outro importante conceito utilizado em Computação Gráfica é o de sistemas de coordenadas, cuja função é nos dar uma referência em termos de medidas do tamanho e posição dos objetos dentro de nossa área de trabalho.

De acordo com Govil-Pai (2004) e Salomon (2006), o mais comum sistema de coordenadas para definir os dados bidimensionais e tridimensionais é o sistema de coordenadas cartesianas. O sistema de coordenadas cartesianas em duas dimensões é baseado em um conjunto de duas retas, chamado eixos. Os eixos são perpendiculares entre si e se encontram na origem, representada por (0,0). O eixo horizontal é chamado de eixo x e o eixo vertical é chamado eixo y. Entretanto, o sistema de coordenadas cartesianas em três dimensões é composto por três eixos perpendiculares, chamados de x, y e z.

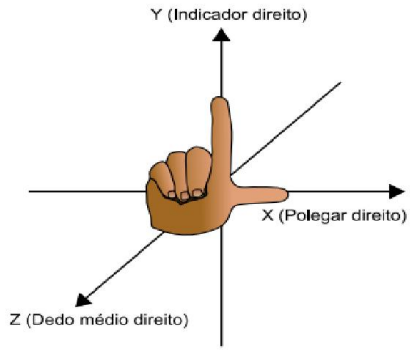

Figura 9 - Regra da mão direita para orientação dos eixos 3D.
Fonte: MANSSOUR e COHEN (2006, p. 9)

Para identificar, facilmente, como o eixo z é posicionado em relação à x e y, normalmente, utiliza-se a regra da “mão direita” ou a regra da “mão esquerda”. Manssour e Cohen (2006) explicam que, na regra da “mão direita” deve-se posicionar a mão direita de maneira que o indicador aponte para direção positiva de y (para cima), o polegar aponte para a direção positiva de x (para o lado) e o dedo do meio aponte para a direção positiva de z (para frente).

3.1.1. Transformações geométricas

Transformações geométricas são operações, que podem ser utilizadas visando à alteração de algumas características como posição, orientação, forma ou tamanho do objeto a ser desenhado.

Ao trabalhar em projetos de Computação Gráfica, descobre-se, rapidamente, que transformações são uma parte importante do processo de construção de uma imagem. Salomon (2006) explica que, se uma imagem tem duas peças idênticas, como rodas, apenas uma parte precisa ser construída a partir do zero. As outras partes podem ser obtidas através da cópia da primeira e, em seguida, modeladas para a forma correta, tamanho e posição, através das transformações geométricas como a translação, escala e rotação, que são analisadas a seguir.

3.1.1.1. Translação

A translação é a transformação geométrica que movimenta o objeto de um lugar para o outro no sistema de coordenadas, deslocando todos os pontos do objeto de uma mesma distância em relação à posição anterior.

Manssour e Cohen (2006) relatam que esta operação consiste em adicionar constantes de deslocamento a todos os vértices, visando trocar o objeto ou cena de lugar. Assim, explicam Azevedo e Conci (2003) que cada ponto em (x,y) pode ser movido por Tx unidades em relação ao eixo x, e por Ty unidades em relação ao eixo y. Logo, a nova posição do ponto (x,y) passa a ser (x',y'), que pode ser escrito como:

$$x' = x + Tx \qquad \text{Equação 1}$$
$$y' = y + Ty$$

Para aplicar translação num sistema cartesiano tridimensional o processo é o mesmo, sendo que, as coordenadas x, y e z, quando adicionadas ou subtraídas, passam a ser x', y' e z', que são obtidas da seguinte equação:

$$x' = x + Tx \qquad \text{Equação 2}$$
$$y' = y + Ty$$
$$z' = z + Tz$$

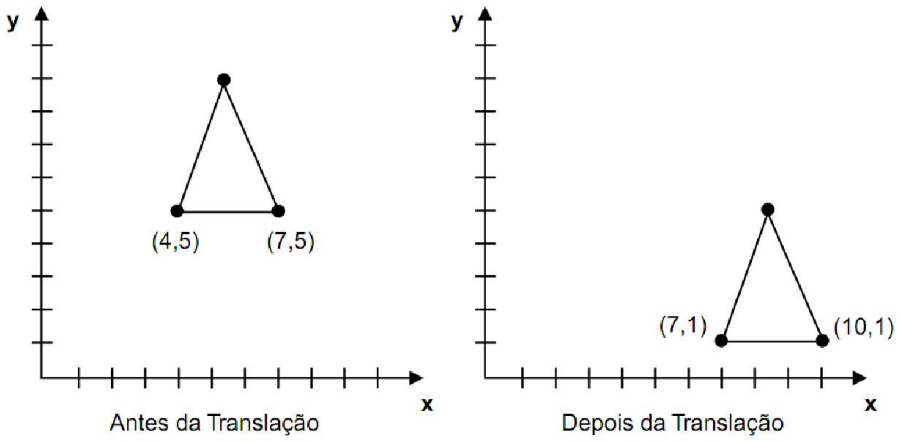

Figura 10 - Translação de um triângulo de 3 unidades na horizontal e –4 na vertical.
Fonte: AZEVEDO e CONCI (2003, p. 39).

3.1.1.2. Escala

A escala é a transformação geométrica utilizada para alterar o tamanho de um objeto, diminuindo-o ou aumentando-o em relação ao tamanho original, definindo, assim, as dimensões de exibição do objeto ou cena.

Manssour e Cohen (2006) explicam que esta operação consiste em multiplicar um valor de escala por todos os vértices do objeto (ou conjunto de objetos), que terá seu tamanho aumentado ou diminuído. Como se trata de uma multiplicação, para aumentar o tamanho deve ser aplicado um fator de escala maior que 1, enquanto que, para diminuir o tamanho basta aplicar um valor de escala entre 0 e 1. Quando aplicamos um fator de escala negativo o objeto terá os sinais de suas coordenadas

invertidos, gerando como resultado o seu “espelhamento” no eixo que teve o fator de escala negativo.

Segundo Azevedo e Conci (2003), o fator de escala em 2D é dado por Sx e Sy e em 3D acrescenta-se Sz, aplicando o escalonamento nos eixos x, y e z, respectivamente. A seguinte equação demonstra as novas coordenadas do objeto escalonado.

$$x' = x \cdot Sx \qquad \text{Equação 3}$$
$$y' = y \cdot Sy$$
$$z' = z \cdot Sz$$

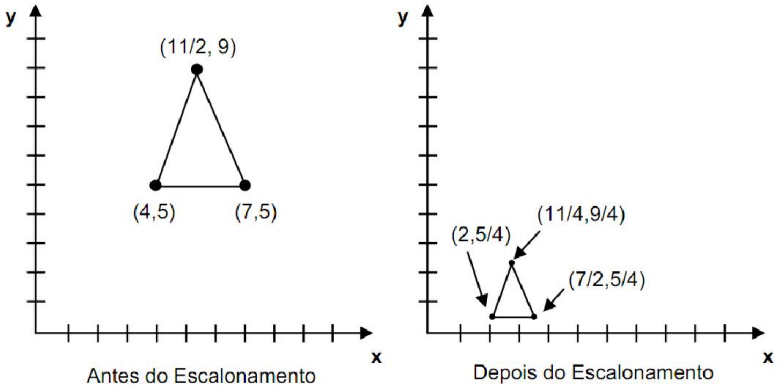

Figura 11 - A mesma figura antes e depois de uma mudança de escala genérica, de 1/2 na horizontal e 1/4 na vertical.
Fonte: AZEVEDO e CONCI (2003, p. 40).

3.1.1.3. Rotação

A rotação é a transformação geométrica utilizada para girar um objeto ou cena em torno de um eixo, de acordo com um determinado ângulo, passando pela origem.

Baseado em Manssour e Cohen (2006), esta operação consiste em aplicar uma composição de cálculos, utilizando o seno e cosseno do ângulo de rotação a todas as coordenadas dos vértices que compõem o objeto ou cena.

Na transformação bidimensional, os pontos no plano *xy* podem ser rotacionados ao redor da origem por um ângulo *θ*, gerando as novas coordenadas x' e y'. Essa operação é representada através da fórmula:

$$x' = x \cdot \cos(\theta) - y \cdot \operatorname{sen}(\theta) \qquad \text{Equação 4}$$
$$y' = x \cdot \operatorname{sen}(\theta) + y \cdot \cos(\theta)$$

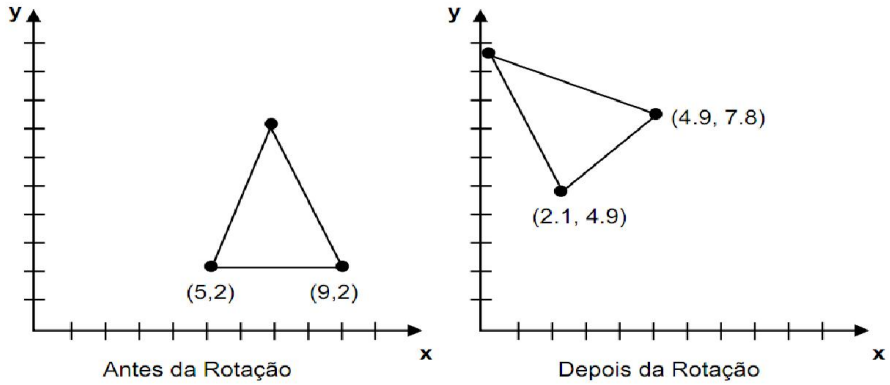

Figura 12 - A mesma figura antes e depois da rotação.
Fonte: AZEVEDO e CONCI (2003, p. 43)

Quando trabalha-se com três dimensões, deve-se definir em torno de qual eixo se aplicará a rotação. Azevedo e Conci (2003) afirmam que a rotação tridimensional torna-se mais simples se for realizada individualmente sobre cada um dos eixos x, y e z, onde são definidos ângulos em relação a cada eixo, denominados ângulos de Euler.

Para rotacionar o objeto em relação ao eixo x deve-se modificar as coordenadas do plano yz. Para rotacionar o objeto em torno do eixo y deve-se alterar as coordenadas do plano xz e, por último, para rotacionar um objeto em relação ao eixo z deve-se modificar as coordenadas do plano xy. Portanto, três matrizes diferentes são definidas na rotação tridimensional, uma para cada eixo. Lembrando que, as rotações em x, y e z são aplicadas de acordo com os ângulos β, δ e α, respectivamente.

A matriz de rotação em relação ao eixo x é:

$$[x'\ y'\ z'] = [x\ y\ z] * \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\beta) & \text{sen}(\beta) \\ 0 & -\text{sen}(\beta) & \cos(\beta) \end{bmatrix} \quad \text{Equação 5}$$

A matriz de rotação em relação ao eixo y é:

$$[x'\ y'\ z'] = [x\ y\ z] * \begin{bmatrix} \cos(\delta) & 0 & -\text{sen}(\delta) \\ 0 & 1 & 0 \\ \text{sen}(\delta) & 0 & \cos(\delta) \end{bmatrix} \quad \text{Equação 6}$$

A matriz de rotação em relação ao eixo z é:

$$[x'\ y'\ z'] = [x\ y\ z] * \begin{bmatrix} \cos(\alpha) & \text{sen}(\alpha) & 0 \\ -\text{sen}(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Equação 7}$$

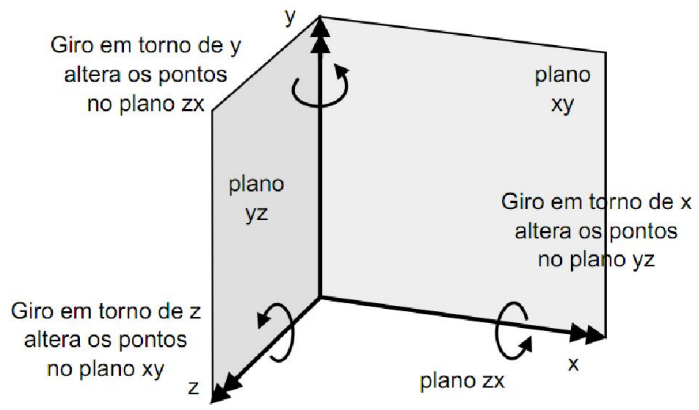

Figura 13 - Definição dos três ângulos de Euler em relação aos eixos x, y e z.
Fonte: Azevedo e Conci (2003, p. 45)

Seguindo a convenção da regra de mão direita, já descrita na subseção 3.1, quando o valor do ângulo de rotação é positivo, a rotação é feita no sentido anti-horário (MANSSOUR; COHEN, 2006).

3.1.2. Técnicas de aplicação de efeitos visuais

O objetivo final da Computação Gráfica é criar imagens sintéticas realistas. Os autores Azevedo e Conci (2003) afirmam que o realismo visual é alcançado através da utilização de técnicas de tratamento computacional aplicadas aos objetos sintéticos. Algumas técnicas de aplicação de efeitos visuais importantes são: cores, textura, mapeamento de texturas, transparência e iluminação, que são detalhadas a seguir.

3.1.2.1. Cores

O uso de cores em Computação Gráfica deixa o processo de comunicação mais eficiente, pois possibilita a geração de imagens realistas, melhora a legibilidade da informação e permite focar a atenção do observador.

Para que exista cor a presença de três componentes são essenciais: um observador, um objeto e a luz.

> A descrição das cores é muito importante, pois existe um padrão internacional que deve ser conhecido. Além disso, a quantidade de cores disponíveis no hardware gráfico atual chega a superar os 16 milhões. A cor é um fenômeno “psicofísico”, que depende da presença de luz no ambiente: se não há luz, não é possível enxergar cores. A percepção de cor depende tanto da física da luz – considerada uma energia eletromagnética – e sua interação com os materiais físicos, quanto da interpretação do fenômeno resultante pelo sistema visual humano (COHEN; MANSSOUR, 2006, p. 213).

Segundo Cohen e Manssour (2006), a cor de um objeto é determinada pela distribuição das ondas da fonte de luz e pelas características físicas do objeto, uma vez que a luz refletida é que determina a cor. Por exemplo, quando a luz branca incide em um objeto algumas freqüências são refletidas e outras são absorvidas pelo objeto. A combinação de freqüências presentes na luz refletida determina o que o olho humano percebe como cor do objeto.

LENGYEL (2004) afirma que a maioria dos monitores exibe informações de cor, usando uma combinação de apenas três comprimentos de onda da luz: vermelho, verde e azul. Este sistema é comumente, chamado de modelo RGB (*Red, Green, Blue*).

No modelo RGB os valores das cores variam entre 0 e 1, determinando, respectivamente, a intensidade dos componentes vermelho, verde e azul. De acordo com Cohen e Manssour (2006), a formação de cores se dá por um processo aditivo: a soma dos componentes em intensidade máxima de vermelho, verde e azul resulta na cor branca. No caso da ausência total de raios de luz, obtém-se a cor preta. E quando a intensidade dos três componentes é igual, obtêm-se os tons de cinza.

3.1.2.2. Textura

Observa-se que praticamente tudo que está a nossa volta possui certo padrão em sua superfície. Pele, tecidos, madeira, metal, folhagens, padrões mais ou menos detalhados, tudo isso pode-se resumir em uma palavra: textura (CORRÊA; GOMES, 2004).

Em Computação Gráfica, texturização é um processo que modifica a aparência de uma superfície, utilizando alguma imagem, função, ou outra fonte de dados. Como exemplo, em vez de representar precisamente a geometria de uma parede de tijolo, uma imagem colorida de um muro de tijolos é aplicada a um único

polígono. Quando o polígono é visto, a imagem colorida aparece onde o polígono está situado. A menos que, o observador se aproxime da parede, a falta de detalhes geométricos, por exemplo, o fato de que a imagem de tijolos e argamassa está em uma superfície lisa, não será perceptível (AKENINE-MOLLER; HAINES; HOFFMAN, 2008).

Em meados de 1970, a incorporação de “realismo” em imagens sintetizadas em computador foi fortemente auxiliada pela introdução da textura de superfície. Técnicas de texturização de superfície permitiram criar imagens visualmente interessantes e ricas sem a necessidade de produzir descrições complexas das superfícies dos objetos. Desde então, muitas técnicas de texturização foram criadas e exploradas e a textura passou a ser uma característica essencial na qualidade de uma imagem sintética (SANTOS, C.; SANTOS, M., 2004).

3.1.2.3. Mapeamento de texturas

A técnica de mapeamento de texturas é uma ferramenta poderosa para obter maior realismo nas cenas geradas. O termo “textura”, em geral, refere-se a uma imagem bidimensional que é “colada” à superfície de um objeto durante o processo de desenho (COHEN; MANSSOUR, 2006).

Para Corrêa e Gomes (2004) as texturas, juntamente com a luz, auxiliam até mesmo na percepção do movimento. Rotacionando-se, continuamente, duas esferas, uma “lisa” e outra, devidamente texturizada, nota-se muito facilmente o movimento desta última.

De acordo com os autores Wright Junior, Lipchak e Haemel (2007), mapeamento de textura é o ato de aplicar uma imagem a uma primitiva geométrica.

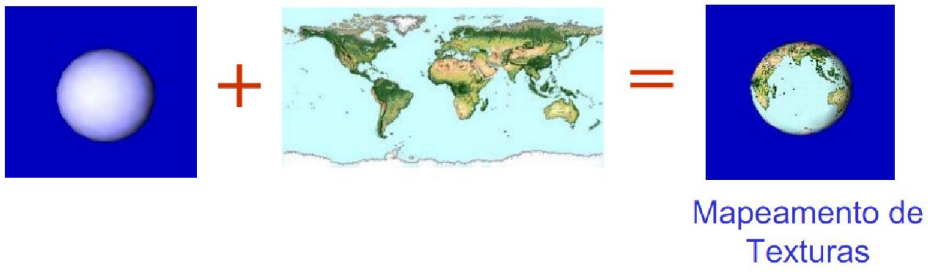

Figura 14 – Mapeamento de textura.
Fonte: SANTOS, M. P. (2005).

3.1.2.4. Transparência

Além dos componentes de cor vermelha, verde e azul (RGB), a biblioteca *OpenGL* utiliza um quarto componente chamado alfa (RGBA). Para McREYNOLDS e BLYTHE (2005), o *Alpha* é usado principalmente para realizar operações de mistura entre duas cores diferentes ou para obter objetos e superfícies transparentes, proporcionando um efeito visual avançado ao cenário.

Para definir o nível de transparência de um objeto este quarto parâmetro alfa pode variar entre 0 e 1, sendo que 0 identifica um objeto totalmente transparente e 1 um objeto totalmente opaco (HENRIQUES; CABRAL; BACCON, 2010).

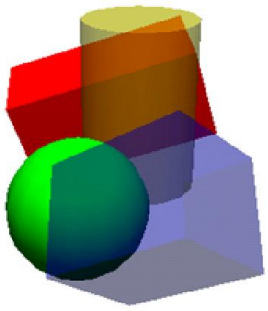

Figura 15 – Transparência.
Fonte: CELES (2001).

3.1.2.5. Iluminação

Na Computação Gráfica, a manipulação da luz assume um papel fundamental no aspecto realístico da apresentação. Sendo que, os efeitos da luz sobre as superfícies e seus materiais, o obscurecimento de superfícies em função de sua posição, orientação e características da luz são peças-chave.
Existem vários modelos de iluminação diferentes, que expressam e controlam os fatores que determinam a cor de uma superfície em função de um determinado conjunto de luzes. (AZEVEDO; CONCI, 2003).

De acordo com COHEN e MANSSOUR (2006), no mundo real para que seja possível enxergar objetos em um ambiente, é fundamental que exista pelo menos uma fonte de luz. De forma simplificada, os objetos são visíveis porque refletem e absorvem raios de luz. Portanto, a noção de uma cena 3D com realismo está intimamente ligada à ideia de iluminação, pois os pontos na superfície de um objeto iluminado possuem diferentes tonalidades de acordo com a luz recebida. Como na Computação Gráfica procura-se simular o que acontece na realidade, é necessário incluir fontes de luz no ambiente virtual 3D e descrever como será a interação dessas fontes com os demais objetos da cena.

A iluminação é muito importante no ambiente 3D porque cria a sensação de profundidade, pois sem iluminação os objetos parecem planos, além de adicionar realismo à cena, deixando-a mais bonita.

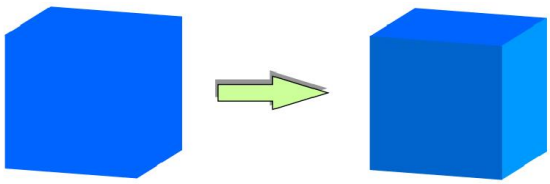

Figura 16 – Iluminação.
Fonte: SANTOS, M. P. (2005).

3.2. Aplicações da Computação Gráfica

Antigamente os computadores não geravam imagens gráficas, os resultados eram emitidos através de caracteres alfanuméricos, de maneira que as imagens eram obtidas pela composição de símbolos. Manssour e Cohen (2006) explicam que durante os anos 50 e 60 foram projetadas as primeiras configurações de sistemas gráficos, que possuíam um novo conceito em visualização: no lugar de caracteres, tornou-se necessário administrar os pontos individuais da tela ou *pixels* (*picture elements*). Assim, os programas passaram a contar com a possibilidade de apresentar saídas na forma gráfica.

Com a evolução do *hardware* gráfico, atualmente, a Computação Gráfica está presente em quase todas as áreas do conhecimento humano. De acordo com SHIRLEY et al. (2005), os maiores consumidores de tecnologia de Computação Gráfica são as seguintes indústrias:

a) **Entretenimento:** Este segmento de mercado engloba os jogos modernos, que consomem uma enorme quantidade de animações, modelos 3D e ilustrações, assim como os desenhos animados e filmes, que utilizam a Computação Gráfica para criação dos efeitos especiais e produção de filmes 3D, tornando-os convincentes e realistas.

b) **Arquitetura e Engenharia:** Estes segmentos utilizam as tradicionais ferramentas CAD (*Computer-Aided Design*), para a visualização de projetos arquitetônicos e criação de peças mecânicas virtuais para futura fabricação.

c) **Simulação:** Graças à evolução dos processadores e das placas de vídeo, as simulações de ambientes estão cada vez mais realistas, auxiliando nos treinamentos de pilotos, nas terapias de fobia, na reconstituição de fatos e acidentes, dentre outras aplicações.

d) **Medicina:** Avançados exames como a Ressonância Magnética criam, através da Computação Gráfica, imagens significativas de dados digitalizados dos pacientes que auxiliam no diagnóstico.

e) **Visualização da Informação:** Neste segmento a Computação Gráfica tem a tarefa de traduzir números e dados em representações gráficas de fácil leitura.

3.3. Introdução à OpenGL

A evolução do *hardware* propiciou o desenvolvimento da Computação Gráfica, pois antigamente a implementação de aplicações gráficas de alto desempenho costumava ser exclusividade de instituições de pesquisa ou empresas que dispunham de *hardware* gráfico veloz, além de programadores experientes. Há poucos anos, essa situação modificou-se com o surgimento do mercado de placas gráficas para computadores pessoais e o desenvolvimento de bibliotecas gráficas, que não exigiam conhecimentos extensivos de programação ou de *hardware*. Uma dessas bibliotecas, hoje o padrão da indústria para aplicações profissionais, é a *OpenGL.*

OpenGL (*Open Graphics Library)* significa biblioteca gráfica aberta. Para Cohen e Manssour (2006), *OpenGL* é uma poderosa e sofisticada API (*Application Programming Interface*) e pode-se classificá-la como uma biblioteca, que possui rotinas gráficas e de modelagem, bidimensional (2D) e tridimensional (3D), extremamente portável e rápida, utilizada em diversas plataformas.

A partir da criação da *OpenGL* em 1992 pela *Silicon Graphics*, tornou-se possível o desenvolvimento de aplicações interativas, com criação de personagens de animações e geração imagens de cenas 3D com um alto grau de realismo e em tempo real.

Segundo Azevedo e Conci (2003), Cohen e Manssour (2006) e Shreiner (2010), por ser portável, a *OpenGL* não possui funções para gerenciamento de janelas, interação com o usuário ou manipulação de arquivos. Cada plataforma, como, por exemplo, o *Microsoft Windows*, possui suas próprias funções para esses propósitos.

Os autores ainda afirmam que a *OpenGL* não é uma linguagem de programação como C, C++ ou Java, mas uma poderosa e sofisticada biblioteca de códigos, para desenvolvimento de aplicações gráficas 3D em tempo real. Normalmente quando se diz que um programa é baseado em *OpenGL* ou é uma aplicação *OpenGL*, significa ser escrito em alguma linguagem de programacão, como *Delphi, C++, Fortran, Python, Perl, Java, Visual Basic*, dentre outras, que fazem chamadas a essa biblioteca.

3.3.1. Comandos da OpenGL

De acordo com Cohen e Manssour (2006), com o objetivo de padronizar e facilitar a sua utilização, todos os nomes das funções das bibliotecas *OpenGL* seguem uma convenção, que divide os nomes das funções em quatro partes:

<PrefixoBiblioteca> <Comando> <ContadorArgumento> < TipoArgumento>.

Tomando como exemplo o comando *glColor3f,* podemos observar que o prefixo gl representa a biblioteca gl, o comando *Color* define o objetivo da função e o sufixo 3f significa que a função possui três valores (contador) de ponto flutuante (tipo argumento) como parâmetro.

3.3.1.1. Inicialização da OpenGL

Como a *OpenGL* é independente de plataforma, precisa-se criar um link entre esta API e o sistema operacional utilizado, no caso da implementação deste estudo, foi utilizado o sistema operacional *Windows*. No *Delphi*, o evento *OnCreate,* do formulário principal, é um bom lugar para fazer essa ligação.

Jacobs (1999) explica que ao iniciar um projeto deve-se criar uma variável, com a finalidade de obter o formato de *pixel* suportado pelo sistema operacional com as configurações necessárias da *OpenGL*. Em seguida, configura-se o descritor de tipo de *pixel*, obtém-se o identificador da tela de pintura e o índice do formato do tipo de *pixel*, fornecido pelo sistema operacional. Depois de definir um tratador de exceções, com o objetivo de manipular as exceções geradas dentro das funções *OpenGL*, configura-se o formato do tipo de *pixel* para o formato fornecido pelo sistema operacional, criando o contexto de pintura e tornando o contexto criado como contexto principal de pintura. Se não ocorrer nenhuma falha, os próximos passos serão indicar que a *OpenGL* foi inicializada corretamente, habilitar o teste de profundidade, definir os limites da tela de pintura da *OpenGL* e setar a cor de fundo da tela de pintura.

3.3.1.2. Construção de primitivas

Em *OpenGL*, as primitivas são comandos que desenham as formas básicas e classificam-se em três categorias: pontos, linhas e polígonos. Essas formas servem como peças, para montar qualquer desenho desejado.

Cohen e Manssour (2006) relatam que as primitivas são definidas por meio de um ou mais vértices, dentro de um comando para especificação de primitivas, descrito a seguir:

```
glBegin(<argumento>);
        glvertex3f (x,y,z);
        ...
glEnd();
```

A função *glvertex3f* é usada para definir um trio de coordenadas x, y, z, sendo passados três argumentos de ponto flutuante.

A função *glBegin* aceita como *a*rgumento uma constante que define qual o tipo de objeto será desenhado. Podem ser passadas como argumento para a função *glBegin* uma das constantes abaixo:

GL_POINTS: Desenha Pontos.

GL_LINES: Desenha segmentos de linhas, considerando os pares de vértices passados entre *glBegin e glEnd*.

GL_LINE_STRIP: Desenha segmentos de linhas conectados na seqüência em que os vértices são declarados.

GL_LINE_LOOP: Desenha segmentos de linhas conectados na seqüência em que os vértices são passados, unindo o primeiro vértice ao último.

GL_TRIANGLES: Para cada trio de vértices declarados, desenha-se um triângulo independente.

GL_TRIANGLE_STRIP: Para o primeiro trio de vértices declarados, desenha-se um triângulo, depois a cada ponto definido pela chamada à função *glvertex,* forma-se um novo triângulo com os dois vértices anteriores, formando uma tira de triângulos conectados.

GL_TRIANGLE_FAN: O primeiro vértice definido pela chamada à função *glvertex* é o vértice central que, junto dos dois vértices definidos em seguida formam o primeiro triângulo. A partir daí, cada vértice declarado é conectado ao vértice anterior e ao

primeiro, formando triângulos conectados em torno de um ponto central, como um leque.

GL_QUADS: Desenha quadriláteros independentes a cada quatro vértices declarados.

GL_QUAD_STRIP: Desenha um quadrilátero a partir da declaração dos quatro primeiros vértices. Depois, a cada dois vértices declarados, forma-se um novo quadrilátero conectado ao quadrilátero anterior, formando assim uma tira de quadriláteros conectados.

GL_POLYGON: Desenha um polígono convexo com o número de lados igual à quantidade de vértices declarados, unindo os vértices na ordem que foram passados e o último na sequência.

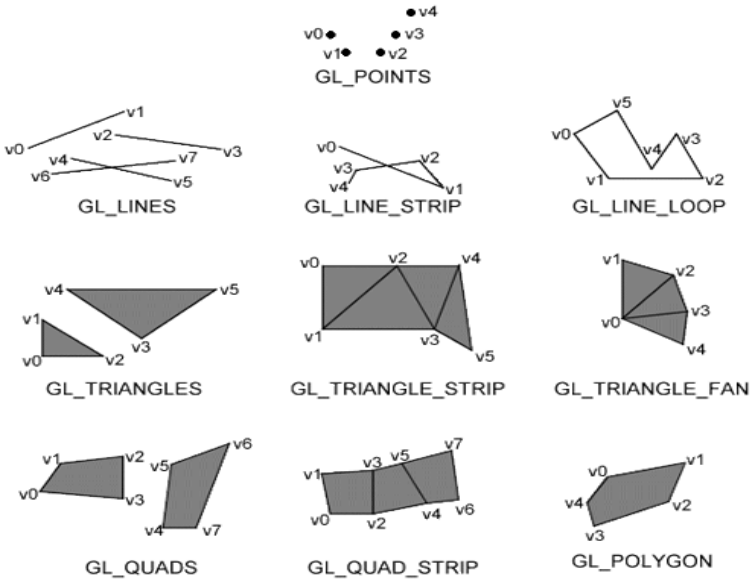

Figura 17 – Primitivas da OpenGL.
Fonte: CELES (2001).

3.3.1.3. Transformações geométricas em OpenGL

A subseção 3.1.1. apresentou os conceitos matemáticos das transformações geométricas, como translação, escala e rotação. Em seguida, será analisado como utilizar essas operações no contexto da *OpenGL*, com o objetivo de alterar a posição, o tamanho e a orientação dos objetos.

De acordo com Cohen e Manssour (2006), para efetuar uma translação usa-se o comando glTranslatef (Tx, Ty, Tz), que move todas as coordenadas dos objetos ao longo dos eixos. Os parâmetros, Tx, Ty e Tz são três valores do tipo *float,* que correspondem, respectivamente, aos valores de translação que devem ser aplicados nos eixos x, y e z.

Para efetuar uma escala utilizamos o comando glScalef (Ex, Ey, Ez)**,** que altera a escala do objeto ao logo dos eixos. Os parâmetros Ex, Ey e Ez são três valores do tipo *float* que correspondem, respectivamente, aos valores de escala que devem ser aplicados nos eixos x, y e z.

Para efetuar uma rotação usamos o comando glRotatef (Angulo, x, y, z) que gira um ou mais objetos de uma cena. Os parâmetros são do tipo *float* que indicam, respectivamente, o ângulo de rotação e o eixo ao redor do qual será aplicada a rotação. Lembrando que, deve-se informar quantos graus deseja-se rotacionar o objeto e adicionar 1, para escolher o eixo em torno do qual o objeto será rotacionado.

Pode-se isolar as transformações geométricas utilizando as funções *glPushMatrix* e *glPopMatrix*.

> Cada vez que *glPushMatrix* é chamada, faz-se uma cópia do conteúdo da matriz corrente que é armazenada no topo da pilha. Analogamente, cada vez que *glPopMatrix* é chamada, a matriz armazenada no topo da pilha é retirada dali e atribuída para a matriz de transformação corrente, restaurando o estado anterior. (COHEN; MANSSOUR, 2006, p. 129).

Pode-se afirmar que, a partir do *glPushMatrix* os comandos de transformação tem sua execução “empilhada”, para evitar que a transformação ocorra no cenário como um todo, e quando a função *glPopMatrix* é chamada, executa-se realmente a transformação.

3.3.1.4. Textura em OpenGL

A OpenGL permite o uso de texturas de uma, duas ou três dimensões, sendo que o tipo mais utilizado em aplicações gráficas, incluindo *OpenGL*, é a textura bidimensional. Cohen e Manssour (2006) explicam que para desenhar um objeto utilizando textura bidimensional, deve-se ativá-la usando o comando *glEnable() e* a constante *GL_TEXTURE_2D*, utilizando a seguinte sintaxe: *glEnable (GL_TEXTURE_2D).* Para desativar a textura utiliza o comando *glDisable (GL_TEXTURE_2D).*

Para aplicar uma textura em duas dimensões é preciso criar uma relação entre os vértices da textura e os vértices do polígono, sobre os quais se deseja inserir a figura escolhida, utilizando o comando *glTexCoord2f.*

Figura 18 – Exemplo de textura em OpenGL.

O sistema de coordenadas da textura tem como (0,0) o ponto inferior esquerdo da imagem e como (1,1) o ponto superior direito. Ou seja, na imagem acima temos as seguintes coordenadas para os pontos A, B, C e D.

Vértice da Textura	Coordenada
A	(0, 1)
B	(1, 1)
C	(1, 0)
D	(0, 0)

Quadro 1 - Relação entre as coordenadas do polígono e da textura.

3.3.1.5. Bitmaps

De acordo com Azevedo e Conci (2003), um *bitmap* é uma imagem produzida por uma matriz fixa de *pixels* coloridos. Cada arquivo BMP é composto por um cabeçalho (*header*), por uma área de informação do cabeçalho (*bitmap-information header*), uma tabela de cores (*color table*) e uma seqüência de *bits* (*bitmap bits*) que definem o conteúdo da imagem. Assim sendo, o arquivo possui a seguinte forma genérica:

```
BITMAPFILEHEADER bmfh;
BITMAPINFOHEADER bmih;
RGBQUAD ColorTable[];
BYTE BitmapBits[];
```

Quadro 2 - Estrutura de um arquivo bitmap.
Fonte: OLIVEIRA (2000).

3.4. Considerações sobre o capítulo

Este capítulo apresentou uma visão geral da Computação Gráfica, seus principais conceitos, técnicas para obtenção de efeitos visuais e aplicações. Como também os conceitos, características e vantagens da biblioteca *OpenGL*, bem como a especificação dos seus comandos. O capítulo 4, dando continuidade ao trabalho, abordará a técnica *billboard*, juntamente com seus conceitos e a aplicação dos mesmos através da implementação de um estudo de caso.

4. BILLBOARDS

A inserção de objetos em ambientes virtuais adiciona realismo ao cenário. No entanto, a representação de objetos apresenta desafios particulares. Para uma representação realista, é necessário um número elevado de polígonos para modelar um objeto com todos os detalhes do mundo real. Assim, um cenário pode normalmente conter vários objetos, logo o número de polígonos da cena iria superar com facilidade o poder de processamento gráfico atual, de tal forma que essa simulação não poderia ser executada com uma taxa de quadros considerada interativa para uma aplicação.

Gadbem 2009 explica que as diversas aplicações, que utilizam objetos em suas cenas, precisam visualizar tais cenas em tempo real, como simulações em realidade virtual e jogos, o que torna proibitivo o alto detalhamento dos objetos. Para atingir boa taxa de eficiência, representações globais mais simples, focadas na percepção geral do objeto e geralmente baseadas em imagens, têm sido utilizadas. Com o objetivo de reduzir a complexidade geométrica da renderização dos objetos, utiliza-se uma técnica chamada *billboard*, que é o assunto da seção seguinte.

4.1. Conceitos

Objetos de natureza mais complexa ou que requerem uma visualização cujo *hardware* gráfico seja incapaz de calcular, são modelados por imagens e representados por *billboards*. Modelar um objeto através de imagens pode resumir-se a projetar uma textura com a figura do elemento que se deseja criar sobre um plano inserido no espaço.

De acordo com McReynolds e Blythe (2005), *billboards* são estruturas geométricas, as quais mapea-se texturas simples. Ao se criar um objeto através do mapeamento de uma imagem (foto ou imagem sintetizada) sobre um plano, surge um inconveniente: o objeto sempre será retangular. Isto ocorre devido à natureza

retangular da imagem a ele aplicada, fazendo com que os elementos não possuam uma silhueta adequada. Para contornar este problema, costuma-se utilizar texturas com transparência, sendo que o valor da transparência de cada pixel será descrito por um byte extra, chamado de canal alfa (*alpha channel*).

Para resolver a aparência plana do *billboard*, pode-se realizar uma rotação do mesmo sempre que o observador se mover, garantindo que a frente do *billboard* esteja sempre apontada para a câmera. Este processo, também, é conhecido como *billboarding* e o polígono correspondente é denominado de *billboard*.

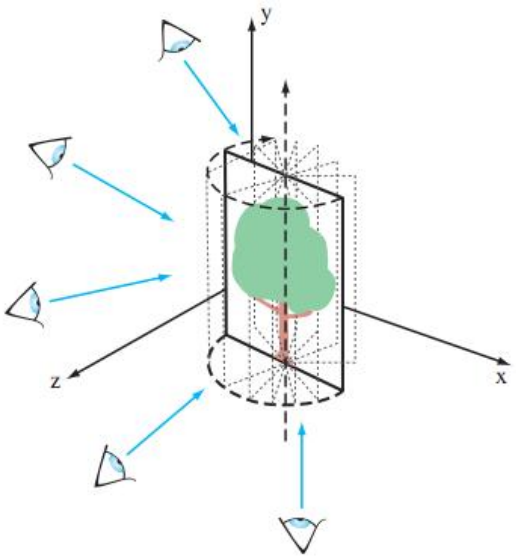

Figura 19- Orientação do billboard.
Fonte: MCREYNOLDS e BLYTHE (2005, p. 260)

Segundo Watt (2000), o *billboard* é de fato uma imagem bidimensional que é girada em torno do eixo y através de um ângulo, previamente calculado, que faz com que o *billboard* fique sempre de frente para o observador, o que dá o aspecto de tridimensionalidade à cena. Lengyel (2004), também, menciona que a técnica *billboarding* é uma maneira eficaz para criar a ilusão de que um objeto plano tem volume.

A técnica de *billboards* é utilizada para simular a geometria de alguma figura tridimensional em duas dimensões, sendo bastante aplicada em jogos, porque

simula uma geometria complexa utilizando apenas um polígono (quadrilátero). As árvores, que possuem uma geometria complexa, são um dos objetos mais simulados com essa técnica. Em aplicações como jogos, normalmente, o usuário não está interessado em apreciar todos os detalhes da árvore. Então, utiliza-se um quadrilátero e uma textura, que contém uma projeção bidimensional da árvore, para simular a geometria da árvore. Assim, a textura é aplicada ao quadrilátero, que é exibido de forma a sempre estar de frente para o observador. Desta forma, o observador é iludido a acreditar que está visualizando uma figura complexa.

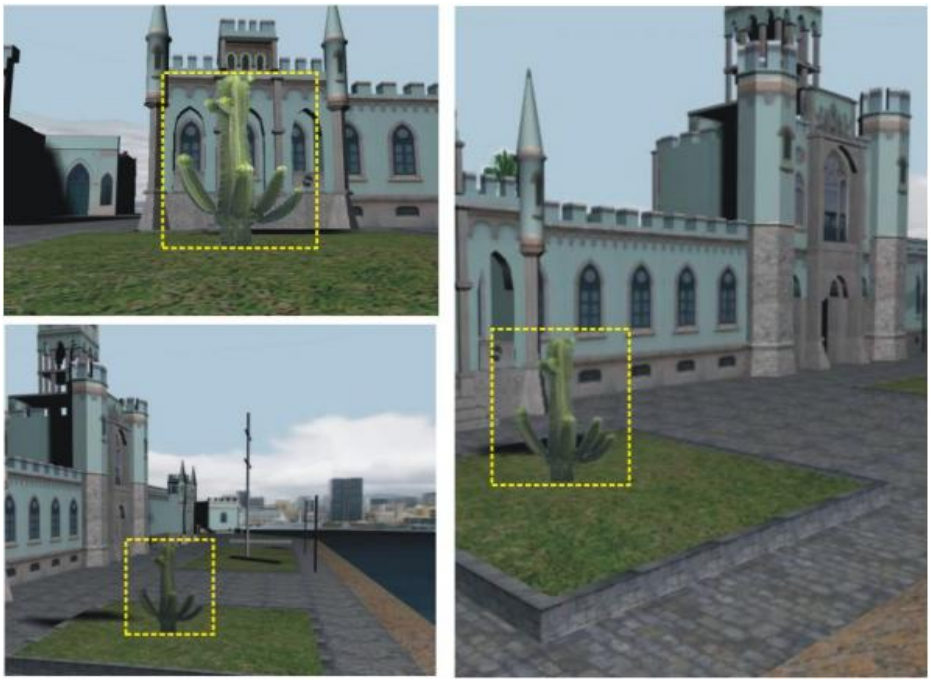

Figura 20 - Representação de um billboard em vários ângulos.
Fonte: CLUA (2004, p. 47).

Clua (2004) afirma que o *billboard* é implementado por um polígono com uma textura com transparência, cuja normal aponta sempre para o observador. Na figura acima pode-se observar que, independente do ponto de vista, o cacto é sempre igual.

Os autores Akenine-Möller, Haines e Hoffman (2008) explicam que as técnicas de *billboarding,* combinadas com alfa e animações, podem ser usadas para representar além de objetos, fenômenos não sólidos como fumaça, névoa, fogo e nuvens.

Outra técnica utilizada para representar objetos em um ambiente virtual chama-se *sprite*. O *sprite* é parecido com *billboard*, entretanto não existe rotação, pois os *sprites,* sempre aparecem aos pares, com duas estruturas colocadas perpendicularmente entre si na forma de uma cruz.

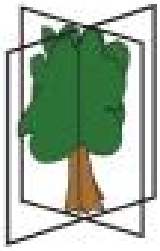

Figura 21 - Exemplo de *sprite*.
Fonte: GADBEM (2009).

Savelli, Seixas e Montenegro (2007) realizaram experiências utilizando *sprites,* para representar objetos da cena. Essas experiências visavam economizar processamento com a rotação do *billboard*, mas os resultados práticos mostraram exatamente o oposto, contrariando as expectativas. Pelo que foi observado, redesenhar um *sprite* a cada *frame* é mais computacionalmente custoso do que redesenhar um *billboard*. Acredita-se que para o computador é mais fácil calcular a matriz de rotação a cada *frame* do que redesenhar duas texturas estáticas.

É possível observar na tabela abaixo, que o resultado foi quase o mesmo para ambas técnicas quando o número de vegetação no terreno era relativamente baixo. Entretanto, à medida que mais árvores foram sendo adicionadas ao terreno, a quantidade de quadros por segundos (ou *frame-per-second*) para o *sprite* cai de forma acentuada.

Número de objetos:	Quadros por segundo:	
	Billboard:	*Sprite*:
50	40.0	40.0
100	40.0	40.0
200	33.3	25.0
300	28.6	20.0
400	25.0	16.7

Quadro 3 - Desempenho das técnicas billboard e sprite.
Fonte: SAVELLI; SEIXAS e MONTENEGRO(2007).

4.2. Estudo de caso

Nesta subseção é apresentado o estudo de caso desenvolvido neste trabalho, que implementa um algoritmo, visando renderizar objetos em um ambiente virtual utilizando a técnica *billboard*.

Para a implementação deste estudo de caso foi utilizada a linguagem de programação *Delphi* 7, juntamente com a biblioteca gráfica *OpenGL*. A escolha desta linguagem ocorreu devido o fato de ter sido a linguagem estudada no curso de Sistemas de Informação e, também, pela pré-existência de uma aplicação apresentada na disciplina de Computação Gráfica, contendo todas as inicializações da *OpenGL* codificadas em *Delphi* 7.

Inicialmente, tem-se um problema: a árvore sempre será retangular, pois a árvore foi criada através do mapeamento de uma imagem sobre um plano, não tendo, portanto, suas formas originais, conforme a figura apresentada abaixo.

Figura 22 - Mapeamento da imagem de uma árvore.

Para contornar este problema, implementa-se a função *ReadBitmapTransp*, que utiliza texturas com transparência, cujo valor da transparência de cada pixel é descrito por um byte extra, chamado de canal alfa (*alpha channel*).

São passados para função *ReadBitmapTransp* três parâmetros, que especificam, respectivamente, o caminho do arquivo bmp (*FilePath*), bem como a largura e a altura do *bitmap*, através das variáveis *tHeight* e *sWidth*.

O procedimento *AssignFile* vincula o arquivo encontrado em disco à variável *bmpfile*. Logo após, o procedimento *Reset* abre efetivamente o arquivo *bmpfile* para leitura, sendo lido um *pixel* de cada vez.

Em seguida, a função *ReadBitmap* obtém o tamanho real do arquivo, que está armazenado na variável *Size*. O procedimento *BlockRead* faz a leitura do cabeçalho do *bitmap* e guarda na variável bfh, para depois verificar se o arquivo é realmente um *bitmap.*

Após a execução do primeiro *BlockRead,* há uma segunda execução desse procedimento para fazer a leitura das informações do arquivo *bitmap* e guardar na variável bmi, para, em seguida, extrair a largura e a altura em *pixels* do arquivo bmp, que são passados como parâmetros para a função *ReadBitmapTransp.*

O procedimento *GetMem* reserva memória para o resultado da função do mesmo tamanho do arquivo lido (*size*). Os *pixels* do *bitmap* são lidos e armazenados na variável *result*^, e logo após são convertidos para *TWrap*, que armazena dados RGB.

Como o *Windows* contém o formato de pixel RGB diferente da OpenGL, foi necessário implementar um *loop* para percorrer todo o *bitmap*, invertendo o “R” com o “B”, convertendo o formato de pixel do *Windows* para o formato de pixels da OpenGL, que é o formato BGR.

Em seguida, é reservada memória para uma variável auxiliar (Data1), que armazenará os *bytes* lidos com um componente de transparência. Logo, foi implementado um *loop* com o objetivo de percorrer todo o *bitmap*, comparando os *pixels* lidos. Se os valores de R, G e B, ou seja, os valores dos *pixels* lidos, forem igual a 0, significa que o *pixel* é preto e que deve, portanto, ser transparente.

A variável *result* armazena o resultado da função *ReadBitmapTransp* que está na variável Data1, e, em seguida, o arquivo é fechado e liberado.

```
function ReadBitmapTransp(const FilePath:string; var
sWidth,tHeight:GLsizei):pointer;
const
  szh=SizeOf(TBitmapFileHeader);
  szi=SizeOf(TBitmapInfoHeader);
type
  PPixelArray = ^TByteArray;
var
  bmpfile: file;
  bfh: TBitmapFileHeader;
  bmi: TBitmapInfoHeader;
  t: byte;
  x, i, size: integer;
  Data, Data1: PPixelArray;
begin
  AssignFile(bmpfile,FilePath);
  Reset(bmpfile,1);
  size := FileSize(bmpfile)-szh-szi;
  BlockRead(bmpfile,bfh,szh);
```

```
  if Bfh.bfType<>$4D42 then
    raise EInvalidGraphic.Create('Invalid Bitmap');
  BlockRead(bmpfile,bmi,szi);
  with bmi do
  begin
    sWidth := biWidth;
    tHeight := biHeight;
  end;
  GetMem(result,size);
  BlockRead(bmpfile,result^,size);
  for x := 0 to sWidth*tHeight-1 do
  begin
    with TWrap(result^)[x] do
    begin
      t := r;
      r := b;
      b := t;
    end;
  end;
  //acrescentado
  Data := result; //variável auxiliar para receber os bytes lidos
  GetMem(Data1,bmi.biWidth*bmi.biHeight*4); //reserva memória para uma
variável auxiliar que vai conter os bytes lidos com um componente de
transparência
  for i:=0 to (bmi.biWidth*bmi.biHeight)-1 do
  begin
    Data1[i*4] := Data[i*3];
    Data1[i*4+1] := Data[i*3+1];
    Data1[i*4+2] := Data[i*3+2];
    if ((Data1[i*4] = 0) and (Data1[i*4+1] = 0) and (Data1[i*4+2] = 0))then
//se R, G e B são 0, significa que o pixel é preto e que deve ser
transparente
      Data1[i*4+3] := 0
    else
      Data1[i*4+3] := 255;
  end;
  result := Data1;
  //fim do acrescentado
  CloseFile(bmpfile);
end;
```

Algoritmo 1 – Função ReadBitmapTransp.

Para desenhar o polígono transparente deve-se fazer uma chamada ao procedimento DesenharCenarioTransp, no *FormPaint* do formulário principal. Este procedimento faz uma chamada à função *ReadBitmapTransp,* tendo como resultado uma imagem preparada para ser gerada com transparência. Ativa-se, então, a imagem carregada anteriormente, passando os parâmetros adicionais para a transparência.

Com os comandos *glEnable* (GL_TEXTURE_2D), *glEnable* (GL_BLEND) e *glBlendFunc* habilita-se, respectivamente, a texturização e a translucência na

OpenGL. Desenha-se finalmente o polígono, que receberá a textura ativada acima, através do procedimento DesenharPlano, que é responsável por passar as coordenadas para a criação do quadrilátero, por meio do comando *glVertex3f*, e também de declarar as coordenadas da textura, para que a mesma seja renderizada corretamente.

```
procedure DesenharCenarioTransp;
var
  imgTextura: Pointer;
  sWidth, tHeight: GLsizei;
begin

 //carregamento da imagem de textura
  imgTextura := ReadBitmapTransp('arvore2.bmp', sWidth, tHeight);

  //ativa a imagem carregada anteriormente
  glTexImage2D(GL_TEXTURE_2D, 0, 4, sWidth, tHeight,
    0, GL_RGBA, GL_UNSIGNED_BYTE, imgTextura);

  //parâmetros adicionais para a transparência
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
  glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

  //habilita a texturização no OpenGL
  glEnable(GL_TEXTURE_2D);

  //habilita a translucência
  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

  //desenha um plano com a textura
  //Desenha árvore da esquerda
  DesenharPlano(ppX2, 0, ppZ2, 5, 5, 0, a2, 0);

  //libera a memória alocada para a imagem
  Freemem(imgTextura);

  //desabilita a texturização no OpenGL para que outros objetos
  //sejam renderizados com suas cores normalmente
  glDisable(GL_TEXTURE_2D);

  //desabilita a translucência
  glDisable(GL_BLEND);

  //Desenha esfera
  glColor3f(0,1,0);
  glPushMatrix();
  DesenharEsfera(0.5,36,12,peX,peY,peZ);
  glPopMatrix();
end;
```

Algoritmo 2 – Procedimento DesenharCenarioTransp.

```
procedure DesenharPlano(CentroX, CentroY, CentroZ, Altura, Largura: Double;
  RotX, RotY, RotZ: Double);
begin
  glPushMatrix;
  glTranslatef(CentroX, CentroY, CentroZ);
  glRotate(RotX, 1, 0, 0);
  glRotate(RotY, 0, 1, 0);
  glRotate(RotZ, 0, 0, 1);
  glBegin(GL_POLYGON);
    glTexCoord2f(0,0);// coordenada da textura
    glVertex3f(+(Largura/2), -(Altura/2), 0);
    glTexCoord2f(0,1);
    glVertex3f(+(Largura/2), +(Altura/2), 0);
    glTexCoord2f(1,1);
    glVertex3f(-(Largura/2), +(Altura/2), 0);
    glTexCoord2f(1,0);
    glVertex3f(-(Largura/2), -(Altura/2), 0);
  glEnd;
  glPopMatrix;
end;
```

Algoritmo 3 – Procedimento DesenharPlano.

Figura 23 - Billboard com canal alfa ativado.

Após obter o objeto com a sua forma original, surge outro problema: como fazer com que um objeto bidimensional aparente seja tridimensional?

Para resolver a aparência plana do billboard implementou-se o procedimento *CalculaAnguloBillBoard.* O observador, representado por uma esfera, movimenta-se, portanto, suas posições não são fixas. Cria-se, então, as variáveis peX e peZ, para armazenar a posição da esfera nos eixos X e Z. O *billboard* para estar de frente para

o observador, sempre, precisa formar com o mesmo um ângulo de 90º. Portanto, suas posições também não são fixas, sendo armazenadas pelas variáveis ppX e ppZ.

Para movimentar a esfera, simulando a movimentação do observador, deve-se pressionar as teclas “w” e “s”, cuja variável peZ é incrementada ou decrementada, movendo a esfera para frente e para trás, e pressionando as teclas “d” e “a” incrementa-se e decrementa-se a variável peX, movendo a esfera para direita e para a esquerda, respectivamente.

```
procedure TfrmPrincipal.FormKeyPress(Sender: TObject; var Key: Char);
begin
  case Key of
    'x': rcX := rcX + 1;
    'y': rcY := rcY + 1;
    'z': rcZ := rcZ + 1;
    'd': peX := peX + 0.2; //para direita
    'a': peX := peX - 0.2; //para esquerda
    'w': peZ := peZ + 0.2; //para frente
    's': peZ := peZ - 0.2; //para trás
  end;
  //Atualiza os valores das variáveis
  CalculaAnguloBillBoard;
end;
```

Algoritmo 4 – Procedimento FormKeyPress.

A cada movimento do observador, ou seja, a cada incrementação ou decrementação da posição da esfera, tem-se que obter o ângulo que a nova posição do observador forma com o plano, para, então, aplicar a rotação no *billboard* de acordo com o ângulo calculado, posicionando assim, o *billboard* sempre de frente para o observador.

O ângulo é obtido através de um cálculo relativamente simples, que pode-se chamar de triangulação. Subtraindo a posição atual da esfera e a posição do *billboard* no eixo X (peX - ppX), obtem-se o cateto adjacente, e subtraindo a posição atual da esfera e a posição do *billboard* no eixo Z (peZ - ppZ), obtem-se o cateto oposto. Com os valores dos catetos, calcula-se, então, a tangente do ângulo.

Calcula-se a tangente, porque ela sempre vai estar de frente para a origem e, considerando que a posição do observador seja (0,0,0), o *billboard* sempre estará de frente para o mesmo.

Com a função ArcTan, obtem-se o valor em radianos do arco, cuja tangente é (peX-ppX) / (peZ-ppZ). Em seguida, converte-se o valor do ângulo que está em radianos em graus, através da função RadToDeg. O valor do ângulo em graus é armazenado na variável *a*, que é passada como parâmetro para o procedimento DesenharPlano, que por sua vez, rotaciona o *billboard* em torno do eixo Y, de acordo com o ângulo obtido.

Como o *billboard* ficará sempre de frente para o observador, o mesmo é iludido a acreditar que está visualizando uma figura complexa e tridimensional.

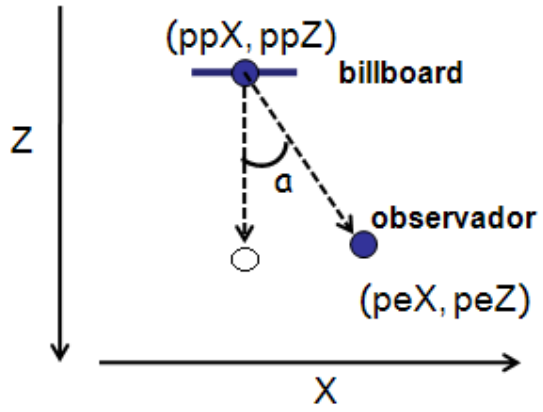

Figura 24 - Movimento do observador.

```
Procedure TfrmPrincipal.CalculaAnguloBillBoard;
begin
  //Cálculo do ângulo do billboard
  tangente := (peX-ppX) / (peZ-ppZ);
  arcotg := ArcTan(tangente);
  a := RadToDeg(arcotg);

  ...

  //atualização da tela
  InvalidateRect(Handle, nil, False);
end;
```

Algoritmo 5 – Procedimento CalculaAnguloBillBoard.

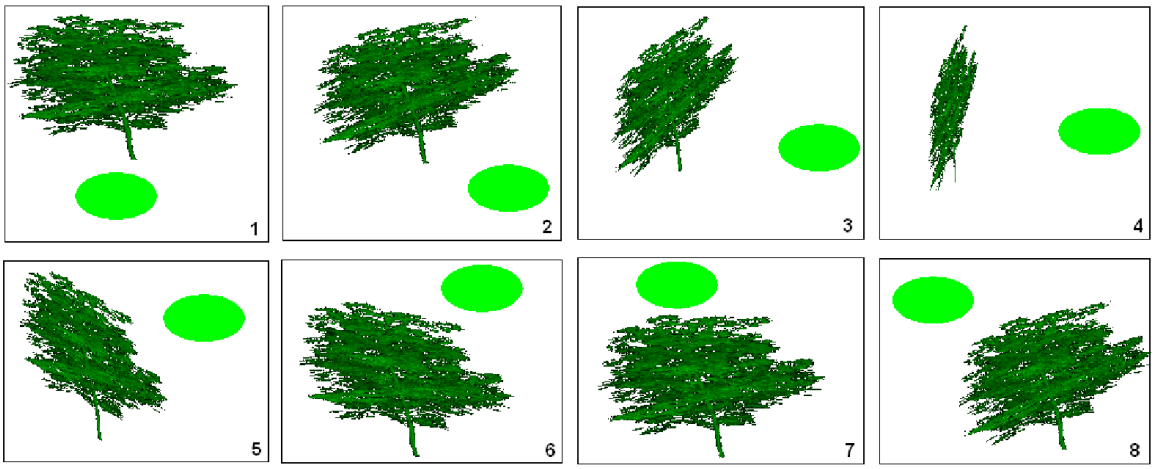

Figura 25 - *Billboard* rotacionado no sentido anti-horário para melhor percepção da sua orientação.

4.3. Considerações sobre o capítulo

Sabe-se que, a inserção de objetos em ambientes virtuais é essencial para aumentar o realismo da cena. Portanto, este capítulo abordou os conceitos e aplicações de *billboards*, técnica utilizada atualmente por ser extremamente eficiente e ter um baixo custo computacional, pois faz uso de geometria simples para representar objetos complexos.

Neste capítulo foi apresentado o estudo de caso realizado durante este trabalho, cujo objetivo foi demonstrar a aplicação de toda parte teórica estudada, mostrando os algoritmos utilizados durante a implementação da técnica *billboard*.

5. CONCLUSÃO

Ao final deste trabalho conclui-se que a inserção de objetos em ambientes virtuais é essencial para aumentar o realismo do cenário. Apesar da representação de objetos complexos ser um desafio, percebe-se que ele pode ser superado, utilizando uma técnica chamada *billboard,* para inserir objetos nos cenários virtuais.

Segundo a literatura consultada, os *billboards* são mais eficientes que os *sprites* e, atualmente, o *billboard* é a técnica mais utilizada por representar objetos complexos, com grande quantidade de detalhes, através de geometria simples, reduzindo drasticamente o custo computacional.

Nota-se que a evolução da Computação Gráfica, bem como, o surgimento de novas áreas de aplicação, decorreu da evolução do próprio *hardware*. Portanto, foi o aprimoramento dos equipamentos gráficos, em termos de memória e velocidade, que tornou possível o desenvolvimento e a execução de técnicas mais elaboradas de Computação Gráfica, para a geração de imagens com alto grau de realismo.

Os estudos e a capacitação profissional, na área de Computação Gráfica, são extremamente atrativos, pois esta é uma área promissora, em franca evolução, que está presente em diversos segmentos do mercado atual e carente de mão-de-obra especializada.

Uma das dificuldades encontradas durante o desenvolvimento deste trabalho foi encontrar fontes de pesquisas específicas para programação de *billboards* em português. Portanto, o presente trabalho é uma contribuição para futuros estudos, principalmente, por ser uma fonte de referência do assunto na Língua Portuguesa.

Como sugestão para trabalhos futuros, espera-se que os algoritmos implementados neste estudo sejam inseridos no projeto ao qual pertence, juntamente com outras técnicas para aumentar o realismo dos objetos na cena. Podem ser implementadas, futuramente, as técnicas *Billboards* com multi-resolução,

Alturas aleatórias e Clusterização de *billboards*, que são chamadas de Técnicas Dependentes da Visão, cujo principal objetivo é reduzir o custo de processamento sem prejudicar a visualização.

6. REFERÊNCIAS

AKENINE-MÖLLER, T.; HAINES, E.; HOFFMAN, N. **Real-Time Rendering.** 3. ed. Massachusetts: A K Peters, Ltd., 2008.

AZEVEDO, E.; CONCI, A. **Computação Gráfica - Teoria e Prática.** 2. ed. Rio de Janeiro: Campus, 2003.

BOUDON, F.; MEYER, A.; GODIN, C. **A Survey of Computer Representations of Trees for Realistic and Efficient Rendering.** 2006. Artigo.
Disponível em: <http://liris.cnrs.fr/Documents/Liris-2301.pdf>. Acesso em: 15 Out. 2010.

CELES, W. **OpenGL Conceitos Básicos.** 2001. Trabalho Acadêmico (Computação Gráfica). PUC-Rio. Disponível em:
<http://www.tecgraf.puc-rio.br/ftp_pub/lfm/CIV2801Aula08OpenGL.pdf>. Acesso em: 10 Nov. 2010.

CLUA, E. W. G. **Impostores com Relevo.** 2004. Tese (Doutorado em Informática) - Programa de Pós-graduação em Informática da PUC-Rio, Pontifícia Universidade Católica do Rio de Janeiro.
Disponível em: <www.ic.uff.br/~esteban/files/tese%20esteban%20gonzalez.pdf>.
Acesso em: 10 Out. 2010.

COHEN, M.; MANSSOUR, I. H. **OpenGL Uma Abordagem Prática e Objetiva.** São Paulo: Novatec, 2006.

CORRÊA, R.; S. P.; GOMES. **Mapa de Textura**: Mip-Mapping. 2004. Monografia (Graduação em Engenharia da Computação e Automação Industrial) **-** Faculdade de Engenharia Elétrica e da Computação da Universidade de Campinas.
Disponível em:
<http://www.dca.fee.unicamp.br/courses/IA725/1s2004/monografias/MipMap.pdf>.
Acesso em: 15 Nov. 2010.

GADBEM, E. V. **Representação de Plantas para Renderização em Ambientes Interativos.** 2009. Exame de Qualificação de Mestrado. Universidade Estadual de Campinas. São Paulo.
Disponível em: <http://www.students.ic.unicamp.br/~ra023578/eqm.pdf>. Acesso em: 02 Ago. 2010.

GOVIL-PAI, SHALINI. **Principles of Computer Graphics:** Theory and Practice Using OpenGL and Maya. The United States of America: Springer, 2004.

HENRIQUES, J.; CABRAL, B.; BACCON D. **Computação Gráfica.** 2010. Trabalho Acadêmico (Graduação em Ciências e Tecnologia) - Universidade de Coimbra. Disponível em: <https://woc.uc.pt/dei/getFile.do?tipo=4&id=1892> . Acesso em: 21 Nov. 2010.

JACOBS, J. Q. **Delphi Developer's Guide To OpenGL.** Texas: Wordware Publishing, 1999.

LENGYEL, E. **Mathematics for 3D Game Programming and Computer Graphics.** 2. ed. Massachusetts: Charles River Media, Inc, 2004.

MANSSOUR, I. H.; COHEN, M. **Introdução à Computação Gráfica**. Porto Alegre, 2006. Disponível em: <http://www.inf.pucrs.br/~manssour/publicacoes.html>. Acesso em: 25 Nov. 2010.

MARCHEZI, F. J. **Um Estudo Sobre Renderização de Terrenos Tridimensionais Para Jogos e Simuladores.** 2007. Monografia – Graduação em Sistemas de Informação, Faculdade do Espírito Santo – UNES.

MCREYNOLDS, T.; BLYTHE, D. **Advanced Graphics Programming Using OpenGL.** Morgan Kaufmann Publishers, 2005.

OLIVEIRA , M. V. de. **Formato de arquivo bmp**. 2000. Trabalho acadêmico – Universidade Federal Fluminense.
Disponível em: <http://www.ic.uff.br/~aconci/curso/bmp.pdf>. Acesso em: 20 Nov. 2010.

ROCHA, R. R. **Um Estudo Sobre Iluminação de Ambientes Tridimensionais Virtuais Usando OpenGL.** 2008. Monografia – Graduação em Sistemas de Informação, Faculdade do Espírito Santo – UNES.

SALOMON, D. **Transformations and Projections in Computer Graphics.** The United States of America: Springer, 2006.

SANTOS ,C. H. da S.; SANTOS, M. A. dos. **Mapeamento de Ambiente.** 2004. Trabalho Acadêmico (Trabalho de aproveitamento do curso Computação Gráfica I da Faculdade de Engenharia Elétrica e de Computação) – Universidade Estadual de Campinas. Disponível em:
<http://www.dca.fee.unicamp.br/courses/IA725/1s2004/monografias/map_ambiente.pdf >. Acesso em: 15 Nov. 2010.

SANTOS, M. J. dos. **Aplicação de Texturas em Terrenos Tridimensionais VIrtuais Usando OpenGL.** 2008. Monografia – Graduação em Sistemas de Informação, Faculdade do Espírito Santo – UNES.

SANTOS, M. P. dos. **Computação Gráfica.** 2005. Trabalho acadêmico - Departamento de Informática da FCT / UNL.
Disponível em: <http://ctp.di.fct.unl.pt/~ps/documentos/essencial-CG.pdf>. Acesso em: 10 Ago. 2010.

SAVELLI, R. M.; SEIXAS, R. B.; MONTENEGRO, A. A. **Visualização de Terrenos Digitais Utilizando Técnicas Dependentes da Visão.** 2007. Artigo - Universidade Federal Fluminense. Disponível em: <www.impa.br/~rbs/pdf/savelli_02_2007.pdf>. Acesso em: 15 Jul. 2010.

SHIRLEY, P. et al. **Fundamentals of Computer Graphics.** 2. ed. Massachusetts: A K Peters, 2005.

SHREINER, D. **OpenGL Programming Guide:** The Official Guide To Learning OpenGL. 7. ed. The United States of America: Addison-Wesley, 2010.

SILVA, H. L. da. **Detecção de Colisão de Objetos Virtuais em Jogos Tridimensionais.** 2008. Monografia – Graduação em Sistemas de Informação, Faculdade do Espírito Santo – UNES.

WATT, A. **3D Computer Graphics.** 3. ed. The United States of America: Addison-Wesley, 2000.

WRIGHT JUNIOR, R. S.; LIPCHAK, B.; HAEMEL, N. **OpenGL Superbible:** Comprehensive Tutorial and Reference. 4. ed. The United States of America: Addison-Wesley, 2007.