UNIVERSIDADE FEDERAL FLUMINENSE

RAFAEL MOREIRA SAVELLI

Visualização de Vegetação em Terrenos Utilizando Técnicas Dependentes da Posição do Observador

NITERÓI

2008

UNIVERSIDADE FEDERAL FLUMINENSE

RAFAEL MOREIRA SAVELLI

Visualização de Vegetação em Terrenos Utilizando Técnicas Dependentes da Posição do Observador

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Computação Visual e Interfaces.

Orientador:
Anselmo Antunes Montenegro

Co-orientador:
Roberto de Beauclair Seixas

NITERÓI

2008

Visualização de Vegetação em Terrenos Utilizando Técnicas Dependentes da Posição do Observador

Rafael Moreira Savelli

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

D.Sc. Anselmo Antunes Montenegro (Presidente) / IC-UFF - Universidade Federal Fluminense

---

D.Sc. Esteban Walter Gonzalez Clua / IC-UFF - Universidade Federal Fluminense

---

D.Sc. Aura Conci / IC-UFF - Universidade Federal Fluminense

D.Sc. Roberto de Beauclair Seixas / IMPA - Instituto de Matemática Pura e Aplicada

---

Ph.D. Paulo Cezar Pinto Carvalho / IMPA - Instituto de Matemática Pura e Aplicada

Niterói, 24 de novembro de 2008.

Dedico cordialmente este trabalho aos meus pais pois, por inúmeras razões, sem eles, eu não teria tido condições de realizá-lo.

Agradecimentos

Devo, em primeiro lugar, agradecer aos meus orientadores os Doutores Anselmo Antunes Montenegro e Roberto de Beauclair Seixas por toda acessoria prestada e pelo importante apoio nas principais decisões tomadas durante o trabalho.

Ao professor externo à Universidade Federal Fluminense, o Ph.D. Paulo Cezar Pinto Carvalho, por acreditar nesse trabalho e por se juntar prontamente à banca avaliadora tão logo quando solicitado.

Aos professores Doutores Esteban Walter Gonzalez Clua e Aura Conci, pelas contribuições que ajudaram na produção desse texto.

À Paula Frederick por toda ajuda no campo de conhecimento técnico durante toda a fase de implementação deste trabalho.

Ao amigo de longa data Gustavo Henrique Soares de Oliveira Lyrio por me ajudar em vários problemas e dificuldades encontradas durante a elaboração deste trabalho.

Ao amigo Sérgio Ricardo Carlos, suporte do laboratório Visgraf situado no Instituto de Matemática Pura e Aplicada, o qual me ajudou sempre pronta e diretamente todas as vezes em que lhe solicitei.

À minha namorada, Marcela Oliveira Imaginário pois, sem seu apoio, carinho, compreensão e paciência eu certamente não teria conseguido superar o momento mais contundente de toda a minha vida.

À minha amiga Juliana de Carvalho Ralha, não só pela prodigiosa e antiga amizade, mas também pela ajuda e apoio nos momentos mais difíceis.

À Angela Regina de Medeiros Correia Dias e Maria de Almeida Freitas da secretaria da pós-graduação, pela amizade e pelo exelente trabalho administrativo.

Aos amigos Marcelo Marzam, Mark Joselli, Simone Silva e todos os outros do Instituto de Computação, pela companheirismo e amizade vivida durante todo o período de estudo.

Resumo

Visualizadores de terrenos com objetos tem se mostrado uma importante ferramenta com muitas aplicações em diversas áreas do conhecimento. Entretanto, o uso desses visualizadores ainda é limitado, já que há muitos casos em que são utilizados dados complexos e reais. Nessas condições, tende-se a ter uma enorme quantidade de informação de modo que nem mesmo os atuais dispositivos gráficos disponíveis conseguem manipulá-las constantemente, consistindo portanto, em um verdadeiro desafio para a computação gráfica.

Esse trabalho mostra como construir uma complexa visualização de terreno em três dimensões, extraindo numerosos dados de vegetação a partir da própria imagem de satélite do terreno. É mostrado também, como armazenar e visualizar digitalmente esses objetos de vegetação, utilizando estruturas de dados complexas como a *quadtree*.

Assim, o objetivo principal desta dissertação é de apresentar soluções que vão preencher algumas dificuldades e faltas existentes no processo de renderização, investigando e analisando diversas formas de tornar possível ou ao menos viável a visualização de terrenos com grande quantidade de objetos.

Abstract

Terrain visualization with objects has become an important tool with a large number of applications in many different knowledgement fields. However, the usage of such tools are still limited, because there are many cases where complex and real informations must be handled. In such conditions, a large amount of data are expected and even modern graphical resources are not capable to deal with it constantly, consisting in a big challenge to graphical computing.

This work shows how to build a complex visualization of three dimensioned terrains by extracting a large number of vegetation data from its own digital satellite images. It shows also how can we store and visualize these data, according to complexes data structures like a *quadtree*.

Thus, the main objective of this dissertation is to present solutions to some of these problems by investigating and analysing different ways of making the whole visualization process viable, even if a large amount of objects is used. To be more specific, this work deals with the problem of terrain visualization with a large number of vegetation features extracted from real satellite images.

Palavras-chave

1. Featured Terrain Visualization
2. Digital Satellite Image
3. 3D Objects Modeling
4. Real Time Rendering
5. Image Based Rendering

Glossário

GPU	:	Unidade de processamento gráfico
FPS	:	Frames por segundo
CFK	:	Triangulação Coxeter-Freundenthal-Kuhn
FBO	:	Frame Buffer Object
LOD	:	Level of Detail
GLEW	:	OpenGL Extension Wrangler
pBuffer	:	Pixel Buffer
OBJ	:	Estrutura de arquivo contendo informações sobre a malha de um modelo 3D
CPU	:	Unidade de processamento central
Texel	:	Elemento de textura

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A visualização de terrenos digitais com objetos tem sido cada vez mais utilizada, tornando-se muito popular entre os usuários e desenvolvedores de todo o mundo. Diversas aplicações utilizam esquemas de visualização de terrenos, destacando-se principalmente as de âmbito científico, industrial e entretenimento. Com o recente avanço dos mais diversos recursos tecnológicos, bem como a prática de preços de mercado mais acessíveis, sabe-se que essa popularização só tende a aumentar ainda mais nos próximos anos.

O problema é que mesmo com os atuais e modernos recursos tecnológicos disponíveis, muitas vezes os dispositivos para geração de gráficos enfrentam problemas de desempenho em lidar com todo o volume de dados envolvidos na visualização, em especial se esses dados forem extremamente numerosos, como acontece em muitos casos, em que a visualização é feita sobre informações provenientes de ambientes reais.

Independentemente da evolução constante da indústria de *hardware*, entendemos que será muito difícil que esta, sozinha, consiga atender a crescente demanda atual vivenciada por todos nós. Por esse motivo, melhoramentos e otimizações no *software* são também de grande importância. Buscar soluções que otimizem o processamento ao máximo, e que produzam uma perda mínima de qualidade da visualização, são dilemas quase que constantes nas vidas desses estudiosos. Este é justamente um dos maiores desafios que os diversos pesquisadores da área da computação gráfica tem que enfrentar.

Durante o estudo realizado neste trabalho, foi possível constatar que, nos últimos anos, diversos trabalhos de visualização de terrenos com objetos têm sido propostos. Boa parte deles tem o intuito de apresentar melhorias do *software*, tais como otimizações no armazenamento de dados, aumento de desempenho nos algoritmos, dentre outros aspectos. Cada autor apresenta sua justificativa para a aplicabilidade do que fora proposto,

entretanto, o objetivo final quase sempre é o mesmo: exibir um grande número de objetos em tempo real, e com qualidade gráfica cada vez mais acurada. De todos os trabalhos analisados, destacam-se aqueles que lidam com níveis de representação de detalhes, não só do terreno [1], mas também dos objetos [2]. O descarte otimizado e inteligente de regiões e objetos temporariamente fora do *frustum* (ou campo de visão) [3] é outro aspecto muito comum na maioria dos trabalhos. Por fim, há ainda aqueles que realizam algum tipo de tratamento nas transições entre diferentes níveis de representação como é o caso dos trabalhos que fazem uso do geomorphing [4], técnica capaz de suavizar trocas de estruturas, deixando a visualização com um aspecto mais realista.

Quanto a natureza dos objetos presentes nas diversas visualizações, notamos que boa parte se trata de algum tipo de vegetação. Em geral, esse tipo de objeto sempre acaba trazendo algumas implicações e problemas com os quais é preciso lidar. Algumas dessas implicações são relativas a própria dificuldade em se retratar fielmente um ambiente natural real. Outra dificuldade diz respeito ao fato de que algumas áreas de vegetação podem conter somente um único tipo de elemento, mas que normalmente aparece em grupos numerosos e com densidades distintas. Problemas como esses são abordados em [5] e [6] e são bons exemplos de trabalhos que, como muitos outros, contribuiram de alguma forma para o estudo e análise de vegetação em terrenos.

Esta dissertação procura dar continuidade ao trabalho descrito em [7] onde o autor realiza, de forma semi-automática, a extração de dados de vegetação à partir da imagens de satélites de uma certa região, para que sejam visualizados posteriormente em um terreno tridimensional. Assim, nesse trabalho, o principal objetivo é investigar formas novas e diferentes de se representar grande quantidade de dados de vegetação, estudando mais a fundo os problemas de visualização envolvidos nessa tarefa. A principal contribuição deste trabalho consiste em apresentar uma estrutura adaptativa complexa que gerencia de forma eficiente e hierárquica as texturas utilizadas nos objetos da cena. Essa estrutura é capaz de selecionar de forma automática a textura mais adequada para que seja utilizada naquele momento por um dado objeto gráfico da cena.

O capítulo 2 deste trabalho apresenta os principais conceitos relacionados à visualização de cenas típicas de um ambiente repleto de vegetação. São discutidas também as principais características e propriedades de cada elemento apresentado. Ainda nesse capítulo, são apresentadas as principais estratégias de representação adotadas na bibliografia.

O capítulo 3, apresenta o processo de extração de dados de vegetação de imagens de satélites. Esse processo baseia-se em alguns conceitos da computação gráfica como o de

reconhecimento de padrões e de geração de assinaturas.

No capítulo 4, são apresentadas as contribuições feitas na visualização de terrenos com objetos, tanto no campo da aparência visual quanto no campo do desempenho. São mostrados nesse mesmo capítulo, os principais resultados obtidos através das implementações dessas técnicas.

Por fim, o capítulo 5 conclui o trabalho, fazendo uma revisao e análise do que foi apresentado, apontando as principais dificuldades observadas no processo de implementação dos conceitos e sugerindo rumos para futuros trabalhos.

Capítulo 2

Conceitos Básicos e Trabalhos Relacionados

2.1 Conceitos Básicos

Ao longo deste trabalho, diversos conceitos importantes serão abordados e discutidos. Entretanto, alguns deles são básicos e essenciais para o entendimento do texto. Por esse motivo, tais conceitos serão descritos à seguir, com o intuito de familiarizar o leitor ao tema da dissertação.

Serão discutidos aqui conceitos que envolvem tanto as formas de representação do terreno (seção 2.1.1) quanto dos objetos (seção 2.1.2), conceitos de renderização em texturas (seção 2.1.3) e por fim, será feito uma breve descricão dos visualizadores utilizados durante o trabalho (seção 2.2).

2.1.1 Representação do Terreno

Na natureza são encontrados diversos tipos de terrenos, onde cada um deles têm suas próprias características e propriedades. Assim, devido a essa enorme variedade e complexidade, a representação digital desses terrenos costumam ser simplificada através de um modelo. No modelo de terreno, apenas os aspectos mais relevantes do terreno são tratados. Os demais são simplesmente desconsiderados e por esse motivo não interferem naquele problema em particular de visualização de terrenos.

Por essa razão, sempre que há necessidade de se construir um terreno digital, precisamos também identificar os aspectos dele que são mais relevantes para o problema de visualização de terrenos. No caso desse trabalho, bem como em muitos outros encontra-

dos na literatura, serão utilizados somente terrenos com dimensões retangulares. Além disso, eles podem vir a conter acidentes naturais como montanhas, vales, lagoas e rios. Já cavernas, grutas e acidentes com inclinações negativas serão desconsiderados pois não têm relevância para esse caso de visualização em particular.

Assim, para terrenos como aquele descrito no parágrafo anterior, podemos encará-los como sendo uma superfície de alturas ou, matematicamente, como uma superfície paramétrica descrita por uma função matemática de duas variáveis g tal que $g(u, v) : \Re^2 \rightarrow \Re$. Como sabemos, esse modelo possui genus zero, o que equivale topologicamente a uma esfera, onde não é possível representar diversos aspectos previamente especificado.

Sabe-se ainda que, para efeito de representação do modelo descrito à cima, é necessário um processo de discretização. Essa discretização pode ser feita de vários modos, sendo o mais comum, através de amostras colhidas a partir dos dados originais do terreno e colocadas na forma de um mapa de alturas. Essas amostras podem ainda ser coletadas de forma regular ou irregular. Normalmente o uso de amostragem irregular é a melhor estratégia, já que tal amostragem leva a um aumento da eficiência e do desempenho, caso seja combinada com um esquema adaptativo, em que mais amostras são utilizadas para representar as regiões do terreno que sejam mais naturalmente acidentadas [8].

No entanto, por questões práticas, implementar um algoritmo de visualização de terrenos utilizando amostragem regular é mais fácil e por esse mesmo motivo, a maioria das atuais aplicações trabalham com esse tipo de amostragem. O terreno considerado aqui não é exceção. As amostras que o constituem encontram-se dispostas em uma grade regular contendo distâncias fixas entre colunas e linhas conforme pode ser observado na Figura 2.1. O próximo passo é definir uma malha poligonal que melhor represente a superfície do terreno. Poderíamos utilizar vários polígonos como forma de representação, entretanto, em geral, utiliza-se somente triângulos. A escolha pelo triângulo se dá em função de que este polígono utiliza a menor quantidade de pontos e arestas possível, além de ser o elemento de reconstrução mais simples possível, já que é um elemento plano. Logo, triângulos constumam ser o polígono predileto de desenvolvedores pela simplicidade de representação. Seguindo essas tendências, os terrenos aqui utilizados também são representados por malhas triângulares.

Um aspecto importante é o modo como esses triângulos são gerados à partir de uma grade de dimensões regulares. Existem inúmeras abordagens, sendo a mais simples a triangulação Coxeter-Freundenthal-Kuhn (ou simplesmente triangulação CFK), na qual um conjunto de células em um reticulado regular definido no domínio, resulta em um

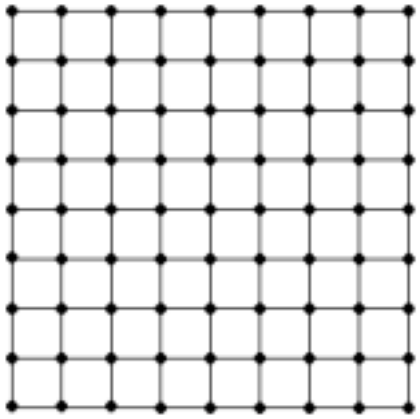


Figura 2.1: Amostragem formando uma grade regular.

conjunto de triângulos, simplesmente conectando-se os vértices não colineares de cada célula. Uma outra estratégia muito utilizada é a que se baseia em uma representação hierárquica. Nesse caso, o domínio é dividido em quatro triângulos que são formados ao traçar as diagonais principais do terreno conforme mostra Figura 2.2. A partir daí, divide-se recursivamente cada triângulo através do traçado da altura perpendicular ao maior lado, formando outros dois triangulos menores. O processo termina quando esgotadas as amostras e, nesse caso, a malha se encontra na resolução máxima, ou quando se tem uma boa aproximação do terreno avaliada por alguma avaliação de erro, que especifica quando interromper o processo de subdivisão para um certo triângulo específico. Algumas desses avaliadores utilizam o conceito de esferas envolventes como descrito em [9]. Ainda na Figura 2.2, o processo recursivo foi desenvolvido no triângulo inferior para exemplificar melhor como se deu o processo. A árvore binária resultante para esse mesmo terreno é criada até que se chegue ao triângulo marcado.

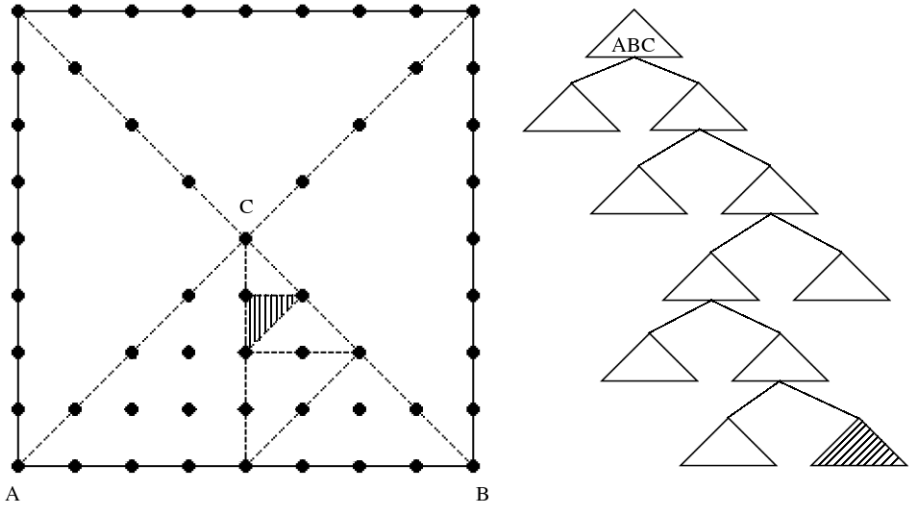
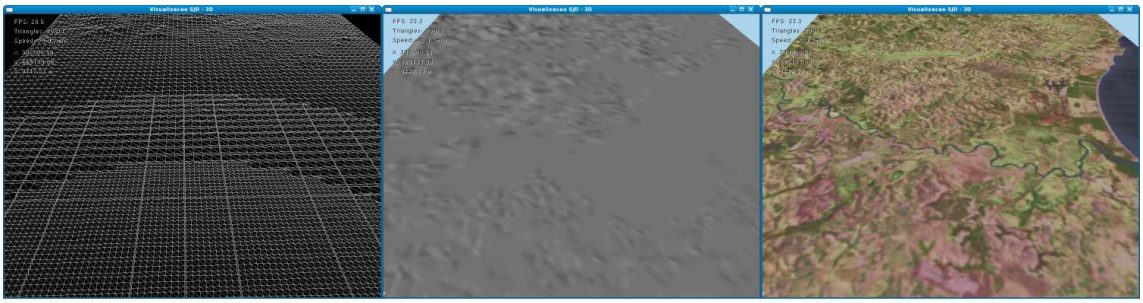


Figura 2.2: Algoritmo desenvolvido em apenas uma parte do terreno.

Uma outra consideração importante muito observada em terrenos digitais diz respeito a utilização de uma ou mais texturas. Essas texturas podem ser monocromáticas, entretanto, as mais comuns são as coloridas. Elas têm como principal função, representar a radiação luminosa dos materiais do terreno para uma dada iluminação [1].

A Figura 2.3 mostra como pode ser insuficiente a representação de terrenos sem a utilização de um mapa de texturas. Em (a), somente triângulos vazados são utilizados na formação da malha (ou do inglês, *wireframe*). Em (b), somente triângulos preenchidos (ou do inglês, *filled*) são utilizados. Por fim, em (c), uma textura é aplicada por cima da malha causando o efeito discutido.



(a) (b) (c)

Figura 2.3: Composição do terreno utilizado.

Note ainda que, nem em (a) e nem em (b) é possível identificar regiões com lagoas e oceanos. Somente quando foi adicionado a textura, pudemos nos beneficiar das cores e inferir a partir dela, outros elementos que estão presentes no terreno. Assim, entendemos que uma textura tem um papel fundamental no aumento do realismo e por essa razão ela não deve ser descartada.

2.1.2 Representação dos Objetos

Assim como um terreno digital, um objeto é simplesmente um elemento gráfico posto em cena para ser visualizado pelo usuário através de uma aplicação. Este elemento gráfico pode ser construído utilizando-se uma, ou mesmo combinando algumas das muitas formas de representações conhecidas e utilizadas por desenvolvedores de sistemas gráficos. A escolha por uma forma de representação ou outra depende de vários aspectos que incluem a experiência do programador, a própria natureza do objeto a ser modelado e a complexidade do sistema em que os objetos serão incluídos.

Nesse trabalho, os objetos utilizados desempenham a difícil função de representar elementos de grande complexidade tal como árvores e gramas, cujo tipo e distribuição são

obtidos à partir de imagens de satélite. Mesmo com os atuais recursos de hardware e *software*, bem como o desenvolvimento de técnicas e métodos para desempenhar tal função, construir esses elementos de forma genérica e com apreciáveis grau de realismo ainda consiste em um desafio para a computação gráfica [10]. O motivo para essa dificuldade advém, sobretudo, da grande diversidade das plantas e árvores encontradas na natureza. Por exemplo, olhando para a Figura 2.4 podemos verificar como pode ser difícil a representação com realismo de plantas e árvores tão diferentes entre si. Mais ainda, note que em alguns casos as folhas de árvores e plantas se misturam sendo difícil até mesmo para o olho humano interpretar de forma correta o objeto visualizado em questão.

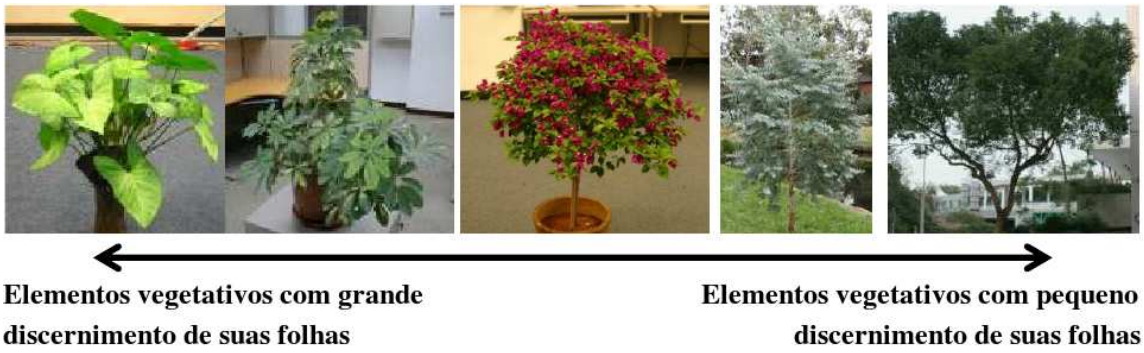


Figura 2.4: Alguns exemplos da vasta diversidade de vegetação.

Hoje em dia as formas mais utilizadas para representação de objetos complexos como, por exemplo, os de vegetação, são ou por alguma forma de modelagem gráfica ou por alguma estrutura baseada em imagem (termo que é proveniente do inglês *image-based rendering*).

Um modelo gráfico em 3D utiliza uma coleção de pontos no espaço 3D conectados por uma variedade de entidades geométricas como triângulos, linhas, curvas, etc. Como os dados reais obtidos diretamente da imagem de satélite são volumosos, esperamos também ter que lidar na visualização com um grande número de elementos. Assim, no intuito de aumentar a quantidade de elementos visíveis na visualização sem comprometer a taxa de quadros por segundo (que do inglês significa *frames-per-seconds*, ou simplesmente FPS), optamos por não utilizar essa forma de representação diretamente no terreno, evitando que esses modelos sejam redesenhados a cada quadro, tornando a visualização intratável.

Assim, para representação dos objetos na cena, optamos por usar uma visualização que realiza a sua renderização baseada em imagens (*image-based rendering*), visto que tais estruturas são mais simples, e costumam demandar menos processamento gráfico por parte da GPU. Com isso, naturalmente, é de se esperar que mais dados possam ser agregados à visualização, sem prejudicar de forma significativa as taxas interativas de

FPS. Nessa categoria de representação, podemos classificar a grande maioria dos trabalhos atuais em dois grupos distintos: **estático** e **dinâmico**. O primeiro deles diz respeito aos trabalhos que tratam seus elementos de forma estática, ou seja, a estrutura base dos elementos representados não sofre nenhum tipo de transformação, seja ela de translação, rotação ou de escala, durante todo o tempo de execução. Já o segundo, o dinâmico, possui algum tipo de movimento em tempo de execução. Sendo assim, diante das várias possibilidades de se modelar digitalmente uma vegetação, decidimos classificar os tipos mais comuns presentes na atual literatura. Estruturamos essa classificação sob a forma de um organograma conforme mostrado na Figura 2.5. Já na Figura 2.6, é possível ver um exemplo de vegetação gerada pelos métodos classificados na Figura 2.5.

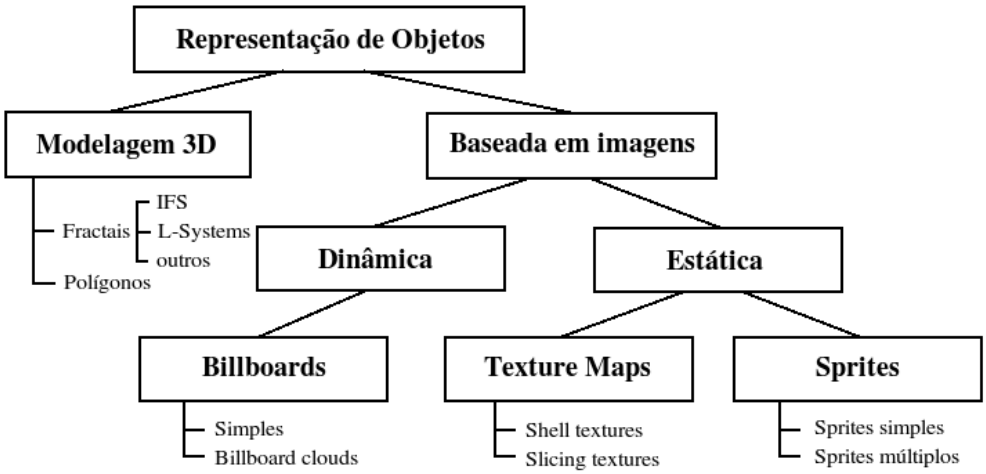


Figura 2.5: Formas de representação de vegetação mais comuns.

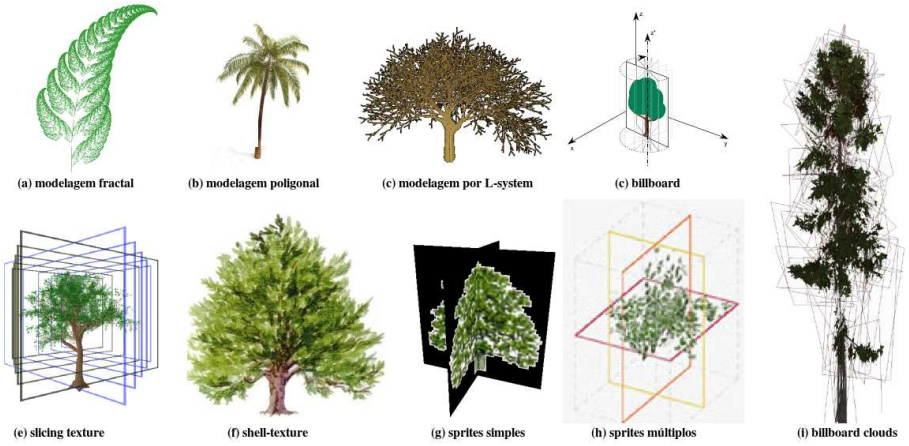


Figura 2.6: Exemplos de representação de vegetação mais comuns.

Talvez uma das técnicas mais antigas de se representar vegetação seja através de modelagens por fractais [11, 12, 13]. De forma sucinta, fractais são modelos matemáticos que apresentam padrões nos quais possuem uma certa auto-semelhança e complexidade

infinita [14]. Desde a criação de seu conceito básico, diversas variantes e tipos foram surgindo. O que se sabe é que hoje existem inúmeros tipos de fractais conforme podemos ver detalhadamente na dissertação de mestrado [15]. Um outro exemplo de trabalho que utiliza a modelagem fractal pode ser encontrado no artigo proposto em [16].

Um tipo de modelagem bem mais comum do que os fractais, é a chamada modelagem por polígonos. Nela são utilizados pontos e retas de modo a formar polígonos (usualmente, triângulos) para representar os objetos. Devido a complexidade de se modelar objetos de vegetação como árvores e plantas, muitos trabalhos fazem o uso dessa forma de representação. Como principal desvantagem, citamos o grande dispêndio de processamento gráfico, o que faz com que tal técnica não se adeque bem à visualizações com grandes volumes de dados. Podemos citar os trabalhos [17, 18] como exemplos de modelagem de vegetação utilizando polígonos.

Outra técnica presente nos principais trabalhos é a técnica de *Lindenmayer systems* (ou simplesmente *L-systems*) [19]. Como essa técnica consiste em um tipo de modelagem fractal, as suas vegetações e, sobretudo, as árvores acabam por terem propriedades e características semelhantes aos demais fractais. Assim, elas por sua vez são construídas através de execuções recursivas de uma ou mais regras de produção em um objeto inicial base [20]. O trabalho [5] faz um comparativo entre essa técnica e uma outra baseada em imagens. Fica claro que a técnica de modelagem por *L-systems* não é das mais econômicas em termos de processamento.

Pudemos observar ainda a existência de uma técnica de geração de vegetação que utiliza geração procedural em *geometry shaders*, baseada em *L-systems* . Entretanto, conforme descrito em [21], quando um grande número de objetos são exibidos por meio da implementação dessa técnica, pudemos observar que o desempenho final fica comprometido e que, por esta razão, essa técnica ainda não se aplica, devido às limitações de *hardware*, aos propósitos do trabalho aqui apresentado.

Partindo agora para técnicas alternativas às soluções dispendiosas envolvendo modelagem gráfica tridimensional, citamos as modelagens baseadas em imagens (*image-based modeling*). Dentro desse grupo, citamos as principais estruturas estáticas e dinâmicas utilizadas para modelar objetos de vegetação. Uma das estruturas estáticas, é a denominada *shell textures* que consiste basicamente no mapeamento de uma textura em um único polígono simples, normalmente um retângulo ou um quadrado. Essa estrutura não se move e devido a sua simplicidade, essa técnica pode ser encontrada em vários trabalhos que lidam com vegetação. Como exemplo do emprego dessa técnica, citamos os trabalhos

[2, 6]. Uma variante dessa técnica chamada *slicing texture* utiliza duas ou mais texturas para a representação do objeto. Essa técnica pode ser observada no trabalho [22].

Outra forma baseada em imagens muito freqüente na representação de vegetação são os *sprites*. *Sprites simples* são duas texturas colocadas sob a forma de cruz sob o terreno. Já os *sprites múltiplos* generalizam a idéia de *sprites simples* para o caso de três ou mais texturas justapostas para cada objeto da cena. Tanto *sprites* simples como omúltiplos podem ser observados no trabalho [5].

Entre as estruturas dinâmicas que utilizam estratégias baseadas em imagens, destacam-se os *billboards*. *Billboards* podem ser encarados como sendo uma *shell texture* a qual deve estar sempre orientada de modo a ficar perpendicular a visão do observador. Para isso, a medida em que a câmera muda de posição, deve-se aplicar rotações no eixo de simetria da figura (usualmente, o eixo Z' conforme mostrado na Figura 2.7) para que a estrutura atenda a esse requesito conceitual. Os trabalhos [2, 7] utilizam extensamente esses elementos para exibir seus objetos no terreno.

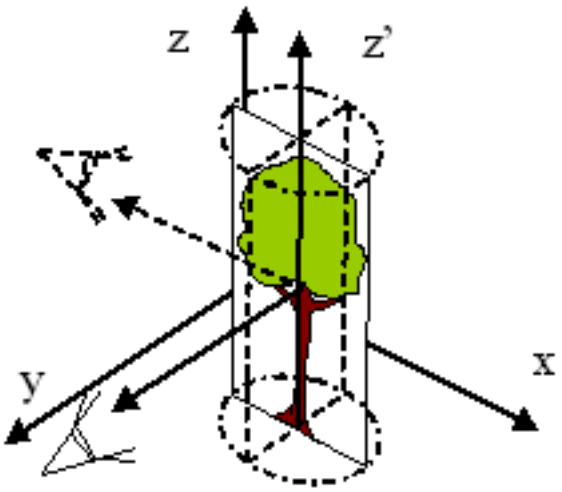


Figura 2.7: Esquema de um típico billboard.

Por fim, uma variação dos *billboards* chamada de *billboard clouds* também aparece com freqüência em vários trabalhos recentes, os quais lidam com o problema de representar a vegetação da melhor maneira possível. Assim, segundo [23], *billboard clouds* é um conjunto de planos que captura a geometria de um modelo o qual se deseja representar, através da projeção de seus triângulos nesses planos, gerando um conjunto de texturas que são utilizadas no lugar da geometria complexa. Alguns trabalhos que fazem uso dessa técnica para o caso de vegetação são [24, 5].

Dentre todas as formas de representação existentes, algumas se destacam tornando-

se muito utilizadas pelos programadores, enquanto outras já não são tão populares, e aparecem somente em uma minoria de trabalhos. O importante é entender que cada uma tem suas vantagens e desvantagens e que seu uso fica dependente da escolha do programador que leva em conta a natureza do problema a ser tratado.

Ainda baseando-se na Figura 2.5, a escolha de algum método para servir de representação de objetos deve levar em consideração o grande volume de dados e a manutenção de uma qualidade gráfica visual razoável. Por esses motivos, precisamos investigar dentre as alternativas baseadas em imagens quais atenderiam a esses requisitos.

No decorrer deste trabalho, abordaremos algumas dessas formas citando suas vantagens e desvantagens. Será apresentado também um teste de desempenho entre as duas estruturas mais promissoras. Vamos entender porque, às vezes, é mais útil e inteligente utilizar uma forma de representação híbrida para se ter os melhores resultados.

2.1.3 Renderização em Texturas

A Renderização em texturas é uma técnica que permite renderizar um objeto *off-screen* e utilizar essa renderização como textura para ser aplicada em uma outra estrutura, desta vez visível para o usuário. Assim, as texturas podem ser construídas em tempo de execução e não somente em tempo de pré-processamento mais comumente encontrado na literatura.

Uma forma de se implementar renderização em texturas é através da técnica denominada *frame buffer object* (FBO) na qual se utiliza uma extensão do OpenGL [25] para fazer renderizações em *buffers* especiais, normalmente invisíveis para o usuário da aplicação.

Uma outra solução plausível para o problema de geração de texturas *off-screen* é a utilização de *pixel buffers* (ou simplesmente, *pbuffer*). Entretanto, de acordo com [26], o processo de obtenção do *texel* em tempo de execução é mais ineficiente quando comparado com o método de FBO. A razão para isso se deve às constantes mudanças de contexto gráfico. Por esse motivo, optamos trabalhar somente com *frame buffer objects* para geração de texturas.

2.1.4 Revisão de Trabalhos Correlatos

Dedicamos essa seção para abordar os atuais trabalhos que, como este, tentam lidar com grande quantidade de objetos na cena de visualização. Serão discutidos os principais

conceitos e técnicas utilizados bem como suas vantagens e desvantagens.

O conceito de visualização de objetos através de representações hierárquicas pode ser observado no trabalho proposto em [2]. Apesar de também utilizar *billboards*, este trabalho realiza uma generalização do conceito de nível de detalhamento, onde toda a cena é armazenada numa hierarquia. Uma grande diferença notada é que este trabalho realiza o que chamamos de *walkthroughs* e não um sobrevôo no terreno como proposto. Isso simplifica o problema da visualização pois diversos aspectos não precisam ser levados em consideração. A exemplo disso citamos o fato de não haver motivação nem necessidade de se preocupar com diferentes vistas, em especial, a superior. Outra grande diferença crucial está no fato de que o trabalho utiliza uma variação do conceito de *octree* para armazenar a hierarquia de *bounding volumes*.

Outro trabalho que também trata a visualização através de hierarquia é o trabalho descrito em [17]. Nele, a hierarquia é espacial e não de objetos. Por esse motivo, os modelos em seus diferentes níveis de detalhamento representam regiões da cena e não objetos em si. Essa consiste, entretanto, na principal diferença com relação a este trabalho. Outra grande diferença diz respeito a reutilização de *frames* em *frames* subsequentes o que pode causar problemas de paralaxe. Paralaxe é um efeito que pode surgir quando ocorre algum tipo de deslocamento do observador, tornando objetos mais próximos aparentemente mais longe de outros que, de fato, estão mais distantes do observador. No trabalho aqui proposto, não são reutilizados *frames*.

Um trabalho mais atual foi apresentado em [6], onde o autor utiliza tanto modelos tridimensionais quanto o conceito de *billboard clouds* para representação de objetos na cena. Nesse trabalho, a hierarquia de nível de detalhe é utilizada para definir a quantidade de *billboards* necessária para cada objeto da cena. Conforme foi possível identificar, infelizmente esse trabalho não utiliza aglomerados de objetos nem texturas com multiresolução.

Existem ainda trabalhos que lidam diretamente com hierarquia de modelagem tridimensional como é o caso do trabalho [18]. Nesse caso, a vegetação 3D é renderizada separadamente por nível de detalhe em pontos e em arestas.

No trabalho descrito aqui, utilizamos a técnica de *render to texture* para gerar dinâmicamente as texturas de um ou mais objetos em suas principais vistas ortogonais de acordo com as disposições deles no terreno. Isso teve importância fundamental na hora de gerar as texturas dos agrupamentos da *quadtree*, conforme será discutido na seção 4.2.4.3. Apesar de também fazer uso de técnicas como o *culling* e LOD dos objetos,

este trabalho se difere dos demais justamente por tratar de dados reais obtidos de forma semi-automática através de imagens de satélite.

2.2 Descrição dos Visualizadores Utilizados

Ao longo desse trabalho foram utilizados três visualizadores de terrenos distintos. Eles tiveram papel importante neste trabalho pois ajudaram a validar conceitos, confirmar métricas e tirar muitas das conclusões mostradas neste documento.

Por esse motivo, dedicamos as próximas subseções para discutir um pouco cada um desses visualizadores de terreno. Procuraremos abordar o histórico, detalhes técnicos de construção, as tecnologias utilizadas, como obtê-los, como utilizá-los, dentre outros aspectos.

2.2.1 Terreno protótipo

Desde o início, quando foi proposto a idéia inicial desse trabalho, sabíamos que iríamos lidar muito mais com os objetos e suas disposições no terreno do que com o próprio terreno em si. Assim, com base nessa afirmação, foi desenvolvido um visualizador protótipo de um único terreno extremamente simples. A idéia era construir um aplicativo com poucos comandos, o suficiente para a realização de testes. Por isso, muito pouco ou quase nada de esforço foi direcionado para a estética de sua interface. O objetivo principal era manter o foco de termos uma aplicação simples, leve, rápida e confiável que pudesse atender bem aos nossos testes.

Com base nessa idéia, desenvolvemos então o aplicativo denominado *basicTerrain* capaz de exibir um terreno simples no qual consiste basicamente em um plano descrito pela função matemática $\boldsymbol{z = 0}$. Nesse ambiente, podemos deslocar a câmera através das teclas *z*, *s*, *x* e *c* que correspondem aos respectivos movimentos: esquerda, frente, trás e direita. Além disso, podemos utilizar o *mouse* para introduzir movimentos de transformação como a rotação da câmera. Informações como taxa de FPS, são lidas em *console* através da tecla *F10*.

A quantidade de tecnologias utilizadas no protótipo *basicTerrain* é pouca, entretanto, elas são bastante atuais e já provaram sua eficiência em muitas outras situações. As tecnologias são a linguagem de programação C [27] e a biblioteca gráfica OpenGL na versão 2.1 [25].

Para se ter uma idéia de sua simplicidade, todo o visualizador tem apenas 929 linhas de programação e seu executável apenas 32 KB. O *basicTerrain* pode ser visto através da Figura 2.8.

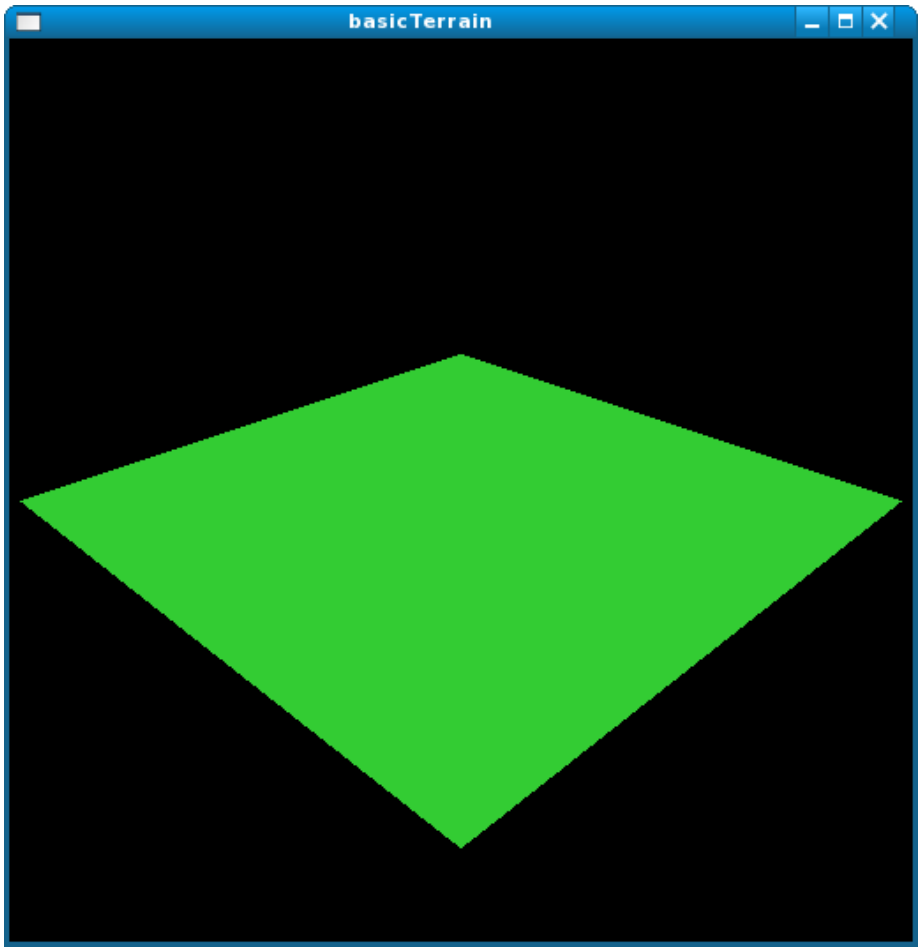


Figura 2.8: *Screenshot* da interface do visualizador do *basicTerrain*.

2.2.2 Enviro (projeto VTerrain)

Esse trabalho fez uso de parte da suíte de aplicações pertencente ao projeto denominado *Virtual Terrain Project* (VTP) [28], desenvolvido e distribuído de forma gratuita e de código aberto.

Os primeiros registros de contribuições recebidos por esse projeto está constado em seu *site* no ano de 1997 [29]. Desde então, o projeto VTP vem ganhando popularidade e robustez, tornando-se cada vez mais uma suíte poderosa de aplicativos gráficos. Além disso, graças às numerosas contribuições vinda de todo o mundo, foi possível acumular uma vasta coleção de dados para várias regiões distintas do mundo. Alguns desses dados incluem: elevação, textura e vegetação.

Todo o projeto do VTP utiliza C++ [30] como linguagem de programação. Além disso, várias bibliotecas opcionais e obrigatórias são utilizadas. Elas foram estruturadas na forma de um organograma, e podem ser vistas na Figura 2.9.

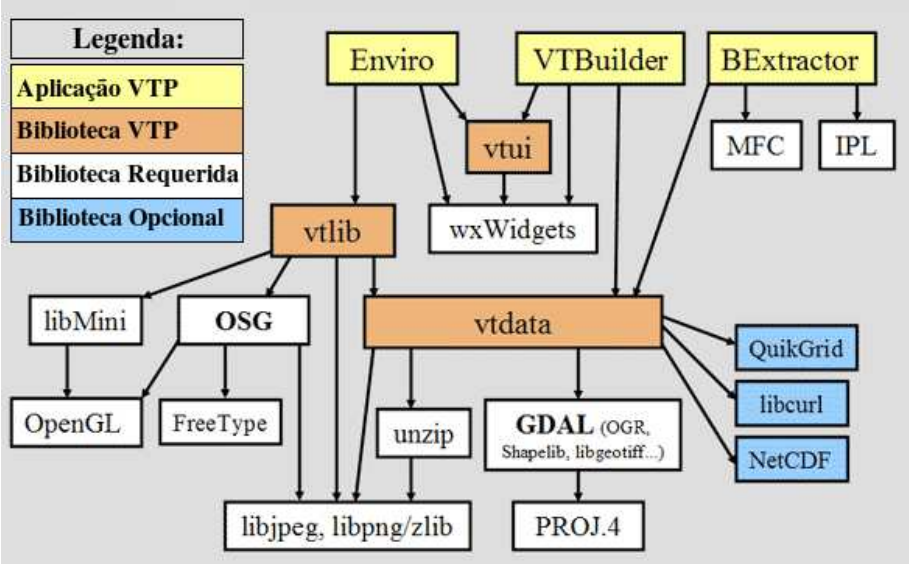


Figura 2.9: Bibliotecas utilizadas no VTP.

As bibliotecas opcionais não foram utilizadas na instalação do VTP para os testes realizados nesse trabalho. Entretanto, as demais bibliotecas precisaram ser instaladas para o funcionamento correto do *software*. Assim, segue abaixo a relação das bibliotecas e suas respectivas versões:

- Geospatial Data Abstraction Library (GDAL) - versão 1.3.1 [31];
- Cartographic Projections Library (PROJ) - versão 4.4.9 [32];
- Open Scene Graph Library (OSG) - versão 1.0 [33];
- wxWidgets Library - versão 2.6.3 [34];
- Mini Library - versão 7.1 [35];
- Libzip Library - versão 0.9 [36];
- Libjpeg Library - versão 36.2.1 [**?**];
- OpenGL Library - versão 2.1 [25];

De todos os aplicativos do projeto VTP, o **Enviro** foi o que mais contribuiu para o enriquecimento deste trabalho. Se tratando de um visualizador de terreno, esse aplicativo

nos permitiu tirar importante conclusões à cerca da forma de representação dos objetos no terreno. Veremos e discutiremos essa importância mais tarde, mas especificamente no capítulo 4.

Para fazer a navegação no terreno utilizando a interface do Enviro, o usuário deve realizar cliques com o *mouse* diretamente no *canvas*. As instruções mais precisas à cerca de todos os possíveis movimentos podem ser encontradas em um manual *online* [37] disponível através do site do projeto. As demais funções são ativadas através dos menus e de seus atalhos do teclado. Para obtermos em tempo real valores importantes e intrínsecos à navegação tal como a taxa de FPS, a posição do *mouse* sobre o terreno e a altura da câmera, podemos consultar a barra de *status* situada na parte inferior da aplicação.

A interface do Enviro pode ser vista na Figura 2.10. O terreno exibido é parte da área próxima à cratera *Kilauea*, no Havaí.

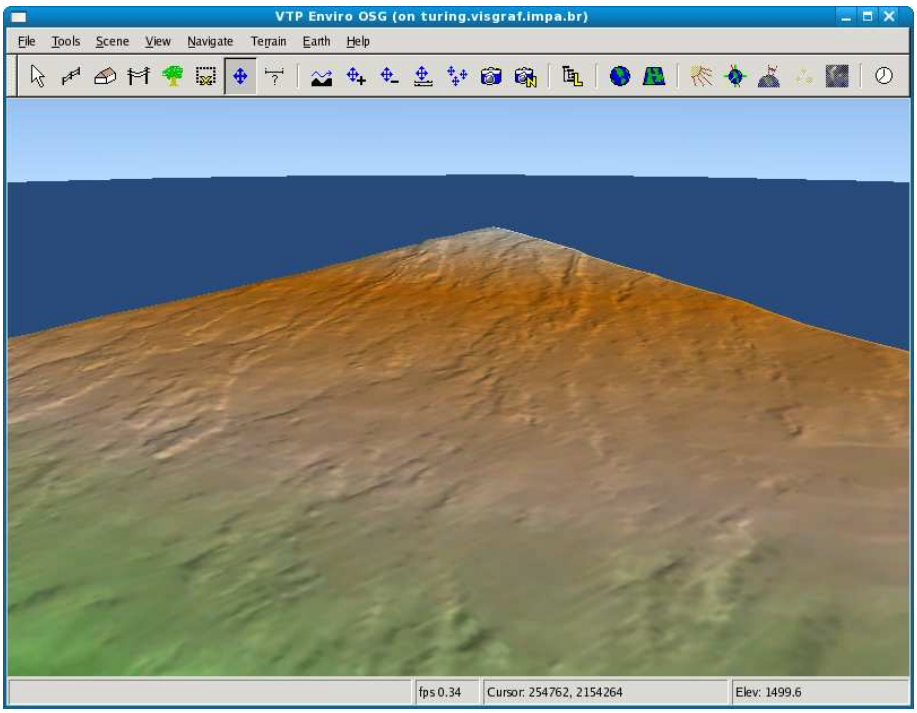


Figura 2.10: *Screenshot* da interface de visualização de terrenos do Enviro.

2.2.3 SJD-Vis3D

Históricamente, o visualizador SJD-Vis3D foi desenvolvido pelo laboratório TeCGraf/PUC-Rio [38] para o centro de Treinamento e Ensino da Marinha do Brasil [39] através dos Sistemas de Jogos Didáticos. O principal objetivo consistia em fazer com que essa ferramenta permitisse realizar vôos sobre terrenos diversos. O grande objetivo dela era de

auxiliar no treinamento de fuzileiros navais.

As tecnologias envolvidas na sua construção são: Liguagem de programação C++ [40]; Liguagem de programação Lua [41]; Biblioteca IUP versão 2.4.0 [42]; Biblioteca IM versão 3.1.0 [43]; Biblioteca CD versão 4.4.0 [44]; Biblioteca VIS versão 1.2 [45]; Biblioteca OpenGL versão 2.1 [25]; Biblioteca OpenGL Extension Wrangler Library [46];

No visualizador de terreno SJD-Vis3D a navegação através do terreno pode ser feita com o *mouse* ou com as setas do teclado. Conforme o *screenshot* mostrado na Figura 2.11, podemos reparar que o SJD-Vis3D imprime na tela constantemente informações essenciais como: o número de quadros por segundos (que do inglês abrevia-se para FPS que quer dizer *Frames per seconds*, o número corrente de triângulos desenhados na composição do terreno, a velocidade do vôo dada em kilômetros por hora e por fim as coordenadas de mundo da posição corrente da câmera no espaço do objeto.

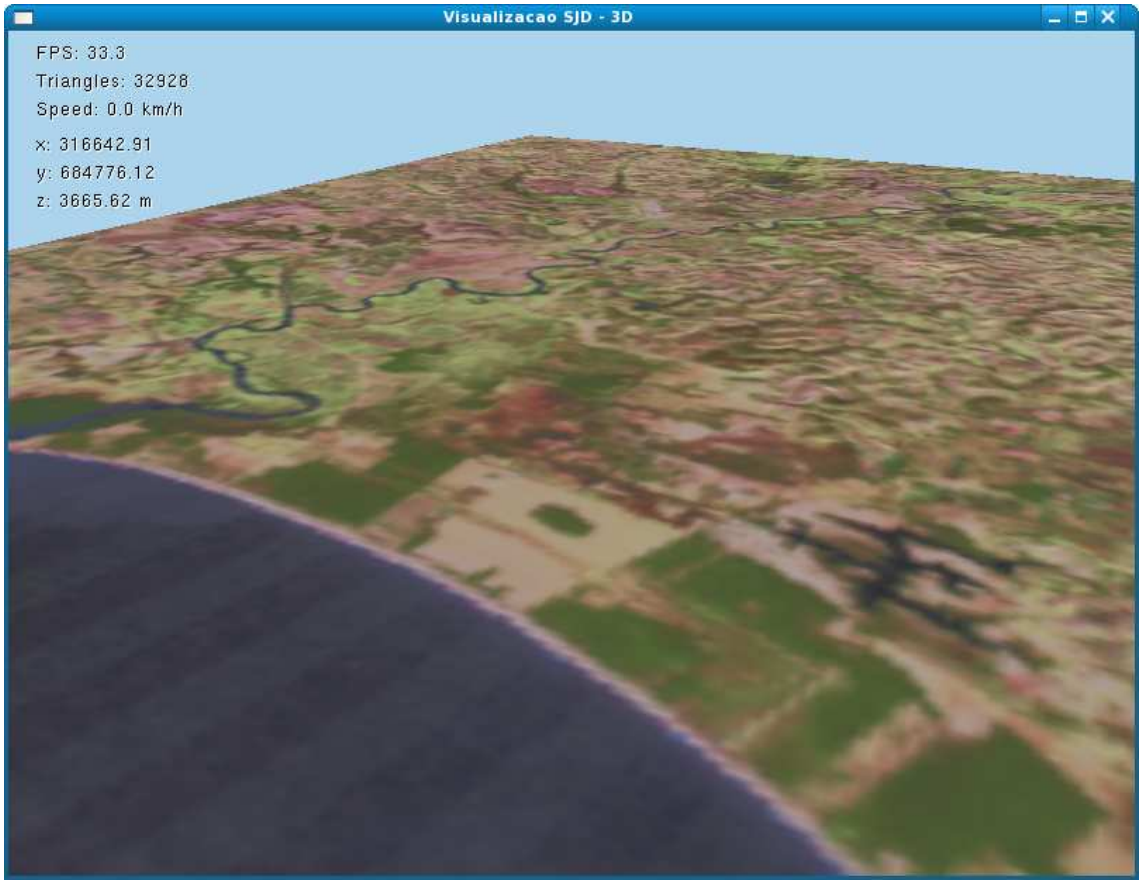


Figura 2.11: *Screenshot* da interface de visualização de terrenos da SDJ-Vis3D.

Não só através do *mouse*, este visualizador também possui algumas funcionalidades ativadas através do uso do teclado. São elas: as teclas *a* e *b* servem para realizar deslocamentos respectivamente para cima e para baixo, perpendicularmente ao terreno, a tecla *w* permite alternar entre malha de triângulo (ou *wireframe*), com e sem preenchimentos, a

tecla *t* permite exibir ou não textura para o terreno. A tecla F1 permite exibir ou não os objetos lidos da imagem de satélite, enquanto F12 permite chavear entre a exibição por *billboards* ou por *sprites*. A tecla *c* permite habilitar ou desabilitar o recurso do OpenGL de *alpha-channel* em todos os objetos presentes no terreno enquanto a tecla *b* faz o mesmo para o efeito *blur*.

Qualquer imagem pode ser utilizada como textura, desde que esta esteja em um formato aceito pela biblioteca IM [43]. Sendo assim, os formatos disponíveis e lidos pela SJD-Vis3D são: TIF, JPEG e BMP. É importante lembrar que essa imagem não precisa ser, necessariamente, georeferenciada e nesse caso, a aplicação considera que a imgem se encaixa perfeitamente em cima da malha do terreno. Entretanto, em muitos casos a imagem disponível não se encaixa perfeitamente no terreno, e por esse motivo, o georeferenciamento se faz necessário.

O georeferenciamento da imagem é feito através de um arquivo no formato chamado "ESRI wordfile", um simples e pequeno arquivo-texto com o mesmo nome da imagem mas com extensão **.tfw**. A estrutura desse arquivo deve obedecer o seguinte formato:

escala_x
rotação_x
rotação_y
escala_y
x0
x1

Os campos *escala_x* e *escala_y* são respectivamente as escalas horizontal e vertical da imagem. Os campos *x0* e *y0* correspondem às coordenadas do centro do *pixel* do canto superior esquerdo da imagem. Já os campos *rotação_x* e *rotação_y* são respectivamente ângulos de rotação em x e em y da imagem. Apesar da aplicação tratar rotações de imagens, nesse trabalho, nenhum tipo de rotação foi feito em nenhuma das imagens utilizadas como texturas. A razão disso se deve ao fato de que todas as imagens já foram rotacionadas quando retiradas de suas fontes.

Essa aplicação foi escolhida como a principal plataforma de teste utilizada nesse trabalho. Por esse motivo, ela foi a única visualização de terrenos utilizada em todos os testes apresentados nesse documento. Optamos por essa escolha na intenção de dar continuidade ao trabalho anterior que será discutido oportunamente no capítulo 3. Como esse trabalho anterior utilizava somente esse visualizador de terrenos, acreditamos que o mesmo deveria estar presente em todos os testes do trabalho atual.

2.3 Conclusão do Capítulo

Neste capítulo tratamos os principais conceitos de visualização de terrenos. Mostramos como um terreno é construído e de que maneira aplicamos uma textura à malha de triângulos. Apresentamos ainda, três visualizadores de terrenos que fizeram parte das análises e testes incluídas nesse trabalho. Assim, estando ciente dos principais conceitos envolvidos nesse trabalho, bem como apresentado os principais visualizadores de terrenos, acreditamos que o leitor possa prosseguir com a leitura do documento.

Capítulo 3

Extração dos Objetos a Partir de Imagens de Satélite

O presente trabalho se fundamenta nas idéias e resultados obtidos no trabalho anterior desenvolvido em [7]. Assim, o principal objetivo desse capítulo é discutir todo o processo de composição da visualização passando desde a extração de informação de vegetação, a partir de imagens de satélite, até a composição final da visualização. Os principais conceitos e técnicas envolvidos nesse processo são: imagens de satélite (seção 3.1), o reconhecimento de padrões (seção 3.2), armazenamento de informações (seção 3.3), carregamento de informações (seção 3.4) e, finalmente, a composição final da visualização (seção 3.5).

Tendo explicado todas as etapas, a seção 3.6 abre espaço para tratar e discutir os problemas que surgiram em decorrência da implementação prática desse processo, não deixando de comentar, inclusive, os próprios problemas relatados nas seções de "trabalhos futuros" do documento publicado.

Finalmente na seção 3.7 é apresentada uma breve conclusão do que foi abordado nesse capítulo.

3.1 Imagens de Satélite

Pela definição de [7], uma imagem digital de satélite é uma foto da terra tirada e enviada por um satélite que orbita esse mesmo planeta. Esse recurso agregado aos satélites, que permitem fotografar e enviar imagens cada vez melhores, consistiu em um enorme avanço tecnológico, já que essas imagens estão servindo cada vez mais como objeto de estudo nos mais variados campos da ciência.

No campo do estudo sobre terrenos, as imagens de satélites contribuem na medida em que, através de seus *pixels*, é possível identificar vários tipos de informações não só intrínsecas ao terreno como montanhas, lagos e vales, mas também informações adicionadas ao longo do tempo pelo homem como estradas, edificações, dentre outros.

Nesse trabalho, existe o interesse em identificar a partir de imagens de satélites acidentes como oceanos, lagos e elevações. Entretanto, os principais esforços estão focados em tentar indentificar e extrair regiões da imagem que contenham alguma concentração de vegetação. Inicialmente, foi buscado regiões com árvores e gramas.

Sabe-se que hoje em dia, existem várias empresas que fornecem imagens digitais de satélite. No Brasil, o governo federal mantém um projeto chamado "Brasil visto do espaço" o qual fica aos cuidados do Embrapa [47]. Assim como outras empresas e orgãos governamentais, sejam nacionais ou internacionais, o Embrapa cobra pelo envio das imagens que podem ser obtidas no site. Felizmente, os preços praticados por essas entidades estão cada vez mais acessíveis, tanto para empresas, quanto para pessoas comuns.

Uma outra boa constatação é o fato de que muitas vezes a obtenção dessas imagens não se dá somente pela compra e venda direta. Opções com acesso gratuito fornecidas por grandes empresas da indústria de *software* tais como a *Google Inc.* que disponibiliza para *download* a famosa ferramenta *Google Earth* [48] e a *Microsoft Inc.* que disponibiliza um *site* com a ferramenta *Microsoft Live Search Maps* [49], fornecem várias imagens de milhares de regiões espalhadas pelo mundo com apreciável grau de resolução, podendo ser consultadas por qualquer pessoa que tenha acesso a um computador conectado à Internet.

3.2 Reconhecimento de Padrões

O próximo passo importante foi desenvolver uma estratégia para realizar a extração da informação de vegetação do terreno utilizando alguma técnica de reconhecimento de padrões (do inglês, *pattern recognition*) [50]. Para isso, combinamos o algoritmo recursivo denominado *split-and-merge* [51] com o conceito matemático de *wavelets* [52].

Para realizarmos a extração de dados de uma imagem digital precisamos identificar nessa imagem, no mínimo, duas regiões: uma que contenha as informações relevantes (no caso, de vegetação) e outra que podemos descartar, pois não contém nenhuma informação útil. Logo, entendemos que algum algoritmo de segmentação deve ser aplicado com a tarefa de realizar essa divisão na imagem gerando as partes de interesse. A escolha foi feita pelo algoritmo *split-and-merge* conforme mostrado no livro [51].

A principal idéia do algoritmo *split-and-merge* consiste em dividir a imagem recursivamente em quatro partes iguais perguntando, para cada uma delas, se a região é homogênea. O critério de decisão é ditado por uma constante limite (ou *threshold*) ajustada pelo programador de acordo com a própria tolerância que ele define e espera para uma região homogênea. O resultado é uma árvore quaternária na qual todos os filhos são considerados regiões homogêneas.

Assim, tendo disponíveis todas as regiões homogêneas, precisamos selecionar aquelas que nos interessam, ou seja, aquelas que contenham informação relevante de vegetação. Para isso, recorremos à matemática e mais especificamente, às *wavelets* [52] implementando uma de suas variantes capaz de identificar uma assinatura comum nessas regiões de interesse.

Segundo a definição matemática encontrada em [52], *wavelets* são funções matemáticas que satisfazem certos requisitos. O próprio nome *"wavelets"* advém do efeito gráfico causado pelo requisisto original, o qual diz que a função deve integrar para zero dando justamente esse formato de "onda" (que do inglês é *wave*). Ainda nesse documento, os autores afirmam que existem diversas aplicações para o uso de wavelets. Em particular, nos interessamos por aquelas que envolvem aplicações de sinais e processamento de imagens [53].

A princípio, qualquer uma das variantes de *wavelets*, a saber, *Daubechies*, *Gabor*, *Laplaciano* e *Haar*, atenderia bem ao requisito de geração de assinaturas, entretanto, a última é a mais simples e por esse motivo foi escolhida para ser utilizada no trabalho.

O trabalho que descrevemos aqui, utiliza uma aplicação que auxilia no processo de identificação de assinaturas em regiões de interesse. Conforme mostrado, a aplicação *buildDB* desenvolvida para esse mesmo fim, permite o usuário carregar uma imagem de satélite conforme mostrada na Figura 3.1. Usando essa interface, o usuário marca algumas pequenas regiões que ele considera de interesse. Em seguida, o algoritmo de *Haar-wavelet* descrito no trabalho [54] é calculado para todas as regiões marcadas. Suas assinaturas digitais são armazenadas internamente por meio dos coeficientes da *wavelet* de *Haar*. O próximo momento consiste em aplicar o mesmo algoritmo de *Haar-wavelet* por toda a imagem buscando assinaturas compatíveis com as pré-armazenadas. Tendo identificado alguma assinatura compatível, a posição daquela região assim como seu tipo são salvos em um banco de dados.

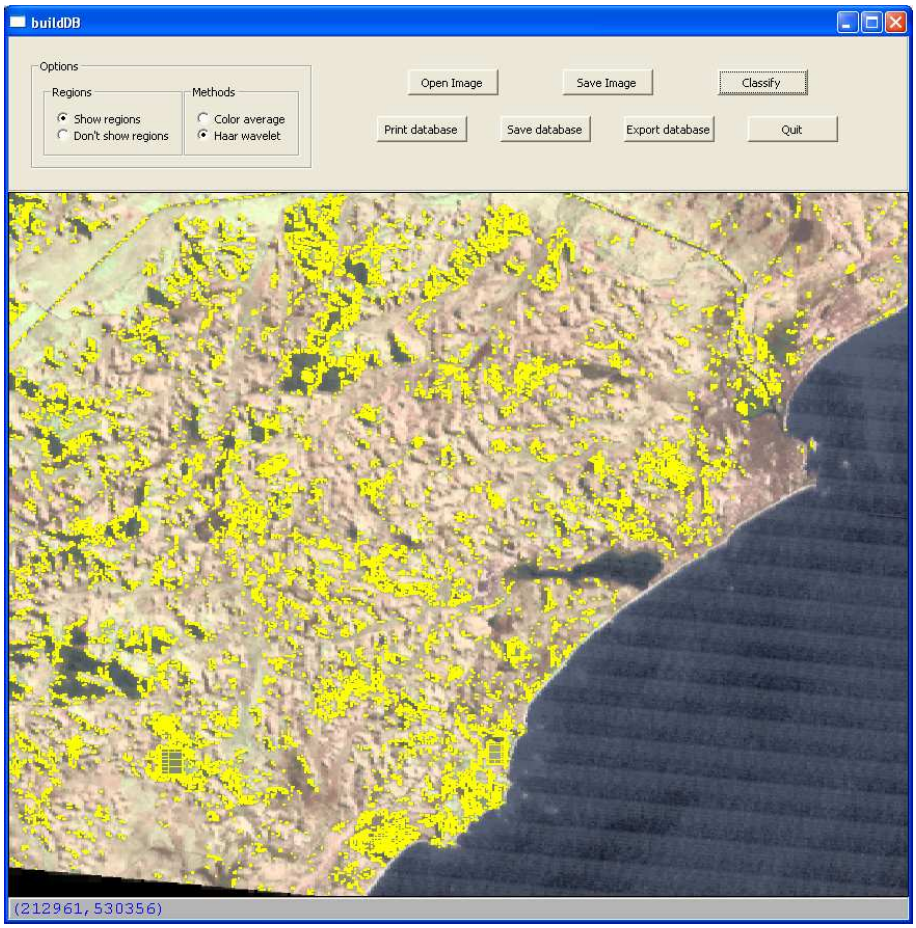


Figura 3.1: Aplicação *buildDB* para auxiliar no processo classificatório.

3.3 Armazenamento de Informações

O banco de dados envolvido nesse trabalho é simples, já que pouca informação como posição e tipo do elemento classificado precisam ser guardados. Na verdade, somente um arquivo texto no formato de separação por ponto-e-vírgula foi suficiente para guardar todas as informações citadas, evitando assim, os complexos relacionamentos e operações envolvidas em um típico banco de dados como *queries*, *store procedures*, *triggers* dentre outras.

3.4 Carregamento de Informações

A leitura do banco de dados é feita uma única vez e no início da aplicação da visualização, antes mesmo do usuário começar o sobrevôo. Esses valores lidos foram armazenados inicialmente em uma estrutura de dados simples do tipo lista, utilizando um vetor estático em memória principal. Como tal operação é realizada somente uma vez, podemos afirmar

seguramente que esse tipo de carregamento não influencia na taxa de interatividade (FPS) da visualização.

Entretanto, como conseqüência da escolha de um *threshold* refinado, definido na seção 3.2, muitas regiões acabam sendo geradas mesmo para terrenos menores. Com isso, mesmo tendo as posições dos elementos dispostos em memória, esse volume de informação causou um grande *overhead* no algoritmo de visualização do terreno comprometendo, inclusive, a taxa de FPS.

3.5 Composição Final da Visualização

Antes de se concretizar o trabalho descrito nesse capítulo, o que se tinha era basicamente uma estrutura de terreno bem como uma imagem de satélite que foi aplicada a esse terreno como textura. As informações hipsométricas só existiam para alguns dos terrenos do território nacional. Sendo assim, as regiões de *Macaé*, no Rio de Janeiro e *Itaoca*, no Espírito Santo se tornaram o estudo de caso. Nesse período, a visualização era composta de um simples processo no qual não se exibia nenhum elemento de vegetação. Para maior entendimento, um esquema foi montado e pode ser acompanhado pela Figura 3.2.

Figura 3.2: Processo de visualização antes de interpretar objetos.

Quando terminado o trabalho descrito nesse capítulo, um outro esquema foi adicionado ao processo mudando-o para aquele mostrado na Figura 3.3. Nesse novo esquema, os *billboards* aparecem para denotar uma região de vegetação.

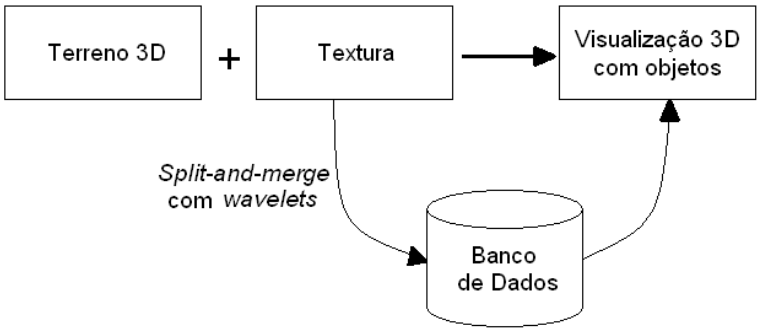


Figura 3.3: Processo de visualização depois de interpretar objetos.

Algumas imagens contendo o terreno vazio e o terreno com vegetação foram expostas novamente nesse capítulo com o intuito de discutir alguns problemas e desafios que surgi-

ram como efeito da implementação do trabalho [7]. Essas imagens podem ser visualizadas nas Figuras 3.4 e 3.5.

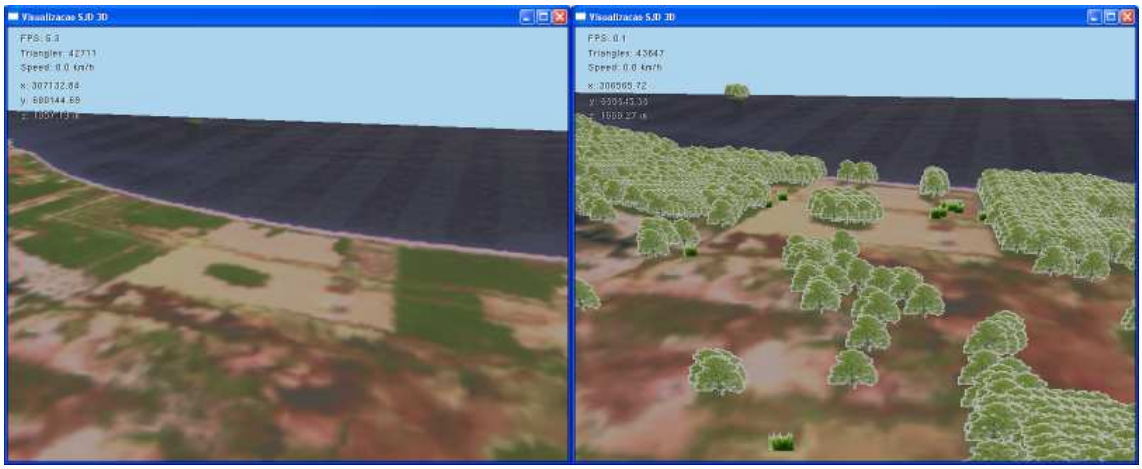


Figura 3.4: Visualização de Terreno para *Itaoca*, no Espírito Santo.

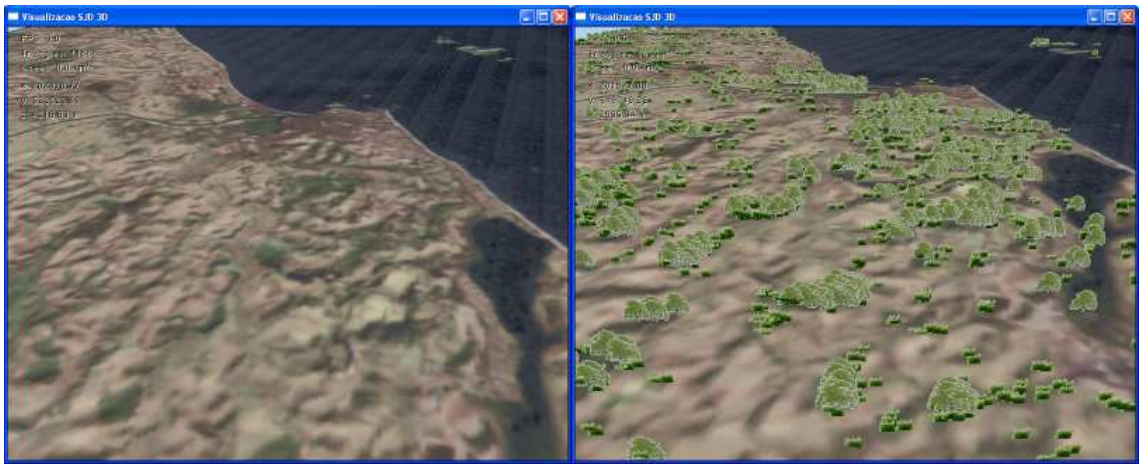


Figura 3.5: Visualização de Terreno para *Macaé*, no Rio de Janeiro.

3.6 Discussão dos Trabalhos Futuros

Apesar deste trabalho cumprir bem a tarefa de reconhecer padrões gráficos provenientes de imagens de satélite, alguns problemas graves surgiram quando seus métodos foram efetivamente aplicados. Parte desses problemas já foram, inclusive, previamente diagnosticados durante a construção do algoritmo. Eles foram notificados e abordados no próprio trabalho [7], mais especificamente na seção de "Trabalhos Futuros".

Um dos problemas apontado diz respeito ao próprio uso de *billboards*. Por construção, os *billboards* só podem realizar rotação em torno do eixo z normal ao terreno. Isso cria um problema no momento em que a câmera estiver direcionada de forma mais oblíqua com

relação ao terreno. Nesse momento, os *billboards* presentes naquela região irão causar um efeito indesejado pois não há como lidar com vistas superiores.

Outro problema com relação aos *billboards* se refere a qualidade da textura aplicada em suas estruturas. Não parece muito sensato utilizar a mesma qualidade de textura para elementos longe e perto da câmera. Para o desenvolvimento do trabalho apresentado nesse capítulo, foi estipulado que cada um dos *billboards* presentes na visualização utilizaria somente uma única textura estática, não importando o contexto no qual estão inseridos.

Além dos problemas relatados no trabalho [7], podemos chamar a atenção para mais alguns outros. Por exemplo, ainda com relação às texturas utilizadas nos *billboards*, observe com atenção a imagem à direita da Figura 3.4. Note a presença de uma indesejável borda branca nos *billboards*. Esse efeito foi observado como conseqüência da adição do *alpha-channel* através de um programa auxiliar escrito na linguagem *Python* [55]. Esse programa utiliza como entrada a imagem e seu negativo e produzia ao final uma imagem com *alpha-channel*. Entretanto, por construção, esse programa sempre gerava também essa borda branca. Isso acabou repercutindo de forma muito negativa, comprometendo a visualização.

Outro problema não relatado foi a precária taxa de FPS obtida com a estratégia desenvolvida. Novamente, olhando para a Figura 3.4, fica claro que algum tipo de otimização precisa ser considerado, uma vez que as taxas de FPS praticadas tornam muito difícil o sobrevôo virtual.

Esses e outros problemas são discutidos novamente nos capítulos subseqüentes, onde uma solução é apresentada para cada um deles na tentativa de melhorar os resultados obtidos no trabalho anterior, enriquecendo-o com técnicas e métodos existentes ou não na atual literatura.

3.7 Conclusão do Capítulo

Este capítulo mostrou de forma sucinta como foi possível extrair informação de vegetação a partir de uma imagem de satélite. Posteriormente, mostramos o modo em que as regiões foram armazenadas para que fossem utilizadas mais tarde pela visualização 3D. Ao final, discutimos alguns problemas não resolvidos que surgiram com o efeito natural da implementação das técnicas apresentadas. Esses problemas serviram como base de estudo para essa dissertação.

Capítulo 4

Contribuições e Resultados

Olhando para os problemas relatados na seção 3.6, podemos classificá-los basicamente em dois grupos distintos de problemas: no primeiro grupo estão os problemas relativos à estética e qualidade visual da aplicação. Já o segundo grupo envolve problemas relativos ao desempenho da aplicação.

Em cada problema abordado, seja do primeiro grupo ou do segundo, uma solução é investigada e seus resultados são apresentados e discutidos nesse mesmo capítulo. São verificados inclusive, os impactos causados por essas soluções na visualização como um todo e, sobretudo, na taxa de FPS.

Assim, o presente capítulo apresenta algumas das muitas formas de contribuição para os diversos problemas que envolvem visualização de terrenos com objetos, em especial, aqueles listados na seção 3.6. Primeiramente serão discutidos os problemas de âmbito estético incluindo a própria imagem de satélite (seção 4.1.1), a textura do terreno (seção 4.1.2), as texturas dos objetos (seção 4.1.3) e a mudança de vistas (seção 4.1.4). Posteriormente, as contribuições de desempenho que incluem a comparação entre *billboards* e *sprites* (seção 4.2.1), LOD dos *billboards* (seção 4.2.2), multi-resolução dos *billboards* (seção 4.2.3) e a estrutra de armazenamento para duas estratégias como *blobs* (seção 4.2.3.1) e *quadtree* (seção 4.2.3.2).

4.1 Contribuições na Qualidade Visual

Identificamos alguns problemas relativos à qualidade das representações visuais que afetam negativamente a visualização como um todo. Foram problemas gerados durante o próprio desenvolvimento do trabalho anterior, que tinha outros interesses e foco de estudo,

como o reconhecimento de padrões. Agora, um estudo um pouco mais detalhado sobre o aspecto de visualização se faz necessário sendo o principal tema abordado nas seguintes subseções.

4.1.1 Textura Utilizada no Terreno

Como vimos no capítulo 2, o terreno por si só consiste apenas em uma grande malha de polígonos (normalmente triângulos) disposta de modo a formar saliências, descrevendo uma região geográfica, a qual se deseja visualizar digitalmente em um computador. Entretanto, em alguns casos, só a malha de triângulos não é suficiente para descrever o terreno em sua forma digital e por isso devemos aplicar nessa malha uma textura para ajudar a realçar os acidentes naturais desse terreno.

Uma das maneiras mais comuns, e bastante eficiente de se fazer isso, consiste em usar como textura a própria imagem de satélite daquela região geográfica, pois quando tal imagem é aplicada georeferenciadamente, ocorre, dentre outras coisas, a sobreposição das regiões acidentadas, melhorando a visualização como um todo.

Fica claro que, na maioria dos casos, quanto melhor a resolução da imagem de satélite, melhor fica também a visualização no terreno, em especial, quando se tem a câmera bem próxima a ele.

Conforme mencionado no capítulo anterior 3, a imagem final utilizada para aplicação no terreno foi resultado da combinação, lado-a-lado, de algumas imagens de satélite retiradas do site do Embrapa [56] através do programa mantido pelo governo Brasileiro chamado "O Brasil visto do Espaço" [47].

Posto que as imagens dispostas anteriormente continham a definição de somente 1:25.000 metros, surgiu a oportunidade de buscarmos imagens melhores, já que esta definição não é das mais refinadas para os padrões nos tempos de hoje.

Assim, a idéia básica ainda consiste em compor imagens lado-a-lado. Porém, elas devem ser retiradas do sofisticado e moderno *software* da *Google Earth* [48] do grupo *Google Inc* [57]. Nesse *software* é possível conseguir imagens com definição superior àquelas do trabalho anterior. Sendo assim, foram emendadas mais de 100 imagens lado-a-lado cada uma contendo a resolução de 1280x890 *pixels*, resultando assim em uma única imagem final de resolução 7018x9889 *pixels*. Essa imagem contém cerca de 199 MB gravada em arquivo no formato **tiff**, atualmente sob proteção de direitos e cuidados da Adobe Systems Inc [58].

Para fazermos o cálculo aproximado da definição dessa imagem, utilizaremos o próprio software *Google Earth* para colher as coordenadas dos pontos inferior esquerdo e superior direito da imagem composta no sistema métrico *Universal Transverse Mercator* (ou UTM) [59]. Os valores lidos para os extremos da imagem composta são mostrados abaixo:

ponto superior direito (em metros): (318593, 7707411)
ponto inferior esquerdo (em metros): (290211, 7667968)

Uma vez obtido esses valores, calcula-se a largura e altura da imagem fazendo a diferença entre esses pontos. O resultado segue abaixo:

$$\boldsymbol{\Delta x = 318593 - 290211 = 28382m}$$
$$\boldsymbol{\Delta y = 7707411 - 7667968 = 39443m}$$

Por fim, divide-se a largura e altura medidas em metros pelas respectivas quantidades em *pixels*. O resultado segue a seguir:

$$\text{definição em } \boldsymbol{x} = \boldsymbol{\frac{28382}{7018} = 4.04}$$

$$\text{definição em } \boldsymbol{y} = \boldsymbol{\frac{39443}{9889} = 3.99}$$

Observe que era de se esperar o mesmo valor tanto para a definição vertical quanto horizontal, entretanto, ficou claro que esses valores embutem um pequeno erro devido às imprecisões na leitura dos dados proveniente do *software*. Não obstante, a definição aproximada da imagem dada pela relação 1:4 é bem superior aquela 1:25.000 utilizada anteriormente.

Note também que a aplicação da textura no terreno é dada pela função OpenGL *glTexGenfv()*, o que permite fácil administração dessa textura. Tendo configurado alguns poucos parâmetros da OpenGL, basta em seguida utilizar *glEnable()* para ativar a textura no terreno e *glDisable()* para desativar. Assim, entendemos que a aplicação da textura é um processo simples e de fácil manipulação.

O resultado da substituição dessa textura pode ser observado na Figura 4.1. Note que à esquerda temos imagens de duas regiões distintas de *Itaoca-ES*, utilizando no terreno a textura retirada do site do Embrapa. Já à direita, as mesmas regiões são mostradas porém agora utilizando as texturas tiradas do *Google Earth*.

Note também que como a aplicação da textura no terreno é dada pela função OpenGL *glTexGenfv()*, não há necessidade de incluí-la na rotina de redesenho de cada *frame*. Tendo configurado alguns poucos parâmetros da OpenGL, basta em seguida utilizar *glEnable()*

para ativar a textura no terreno e *glDisable()* para desativar. Isso gera uma enorme vantagem posto que a substituição da textura por uma outra melhor não causa impactos significativos na taxa de FPS, melhorando, significativamente, a qualidade da visualização conforme pode ser observada na Figura 4.1. Note que à esquerda temos imagens de duas regiões distintas de *Itaoca-ES* utilizando no terreno a textura retirada do site do Embrapa. Já à direita, as mesmas regiões são mostradas porém agora utilizando as texturas tiradas do *Google Earth*.

Figura 4.1: Texturas aplicadas ao terreno de *Itaoca-ES*.

4.1.2 Imagens Baseadas em Modelos 3D

Um dos graves problemas da visualização de objetos gráficos representado através de imagens está ligado à baixa qualidade das texturas à eles associadas. Conforme mostrado na seção 3.6, texturas com baixa qualidade (ou pouco *texels*) precisam dar lugar a outras melhores. Outro aspecto importante está ligado ao atual método pouco eficiente de se adicionar o *alpha-channel* às imagens. Um programa *Python* realiza essa tarefa, porém este causa um efeito final não muito bom nas texturas.

Assim, no âmbito de resolver esses dois problemas, ao mesmo tempo, propomos a geração de textura a partir de modelos sintéticos em três-dimensões. Para isso, adaptamos um tutorial que trata de transformações básicas de modelos 3D como translação e rotação

desenvolvido por Nate Robins [60]. Esse aplicativo nos permitiu carregar modelos tridimensionais lidos à partir de formatos de arquivo de extensão OBJ. A Figura 4.2 mostra esse aplicativo tendo carregado em sua interface um modelo simples de árvore.

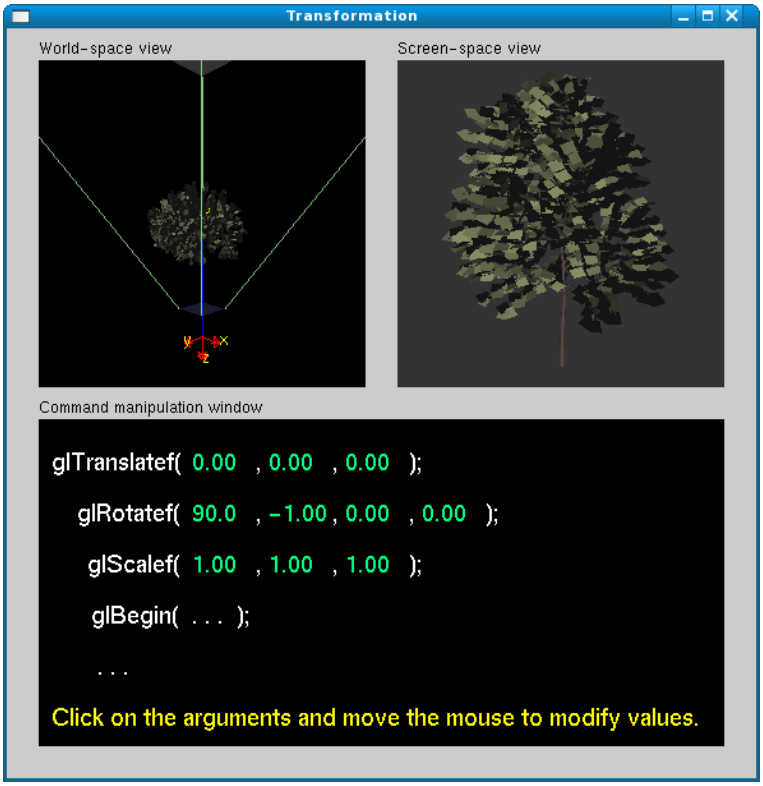


Figura 4.2: Aplicação adaptada para exibir modelos 3D.

O próximo passo consistiu em incorporar esse aplicativo à visualização 3D, para que fosse possível criar as novas texturas à partir desses modelos previamente renderizados. Para isso, o conceito de *frame buffer object* (ou simplesmente FBO) precisou ser utilizado.

O FBO é uma extensão do OpenGL [25] disponível na maioria das placas gráficas atuais que, ao invés de fazer com que o OpenGL renderize em um canvas comum, visto pelo usuário da aplicação, faz com que a renderização seja feita em *buffers* especiais, normalmente fora da visão do usuário [61]. Tais *buffers*, criados e gerenciados pela biblioteca *OpenGL Extension Wrangler* (ou simplesmente GLEW) [46], permitem realizar de forma mais fácil e eficiente o *render-to-texture* evitando assim alternativas que envolvem cópias de *pixels* ou *pixels buffer* (*pBuffer*).

Na tentativa de tirar o efeito de "bordas brancas" mencionado na seção 3.6, decidimos utilizar uma funcionalidade fornecida pelo próprio OpenGL a qual se chama *blur*. O *blur* serve, dentre outras coisas, para suavizar as mudanças abruptas entre as partes transparentes e opacas da textura. Entretanto, quando aplicamos o efeito na imagem

original, o comportamento observado não correspondeu ao esperado. Conforme consta na Figura 4.3, mesmo utilizando a mesma quantidade de *pixels* para as duas texturas, podemos observar que o *blur* ficou visualmente menos funcional na textura da esquerda. Este fenômeno se deve em função da adição do *alpha-channel*. No caso do FBO, o próprio OpenGL realiza essa tarefa de acrescentar o *alpha-channel*.

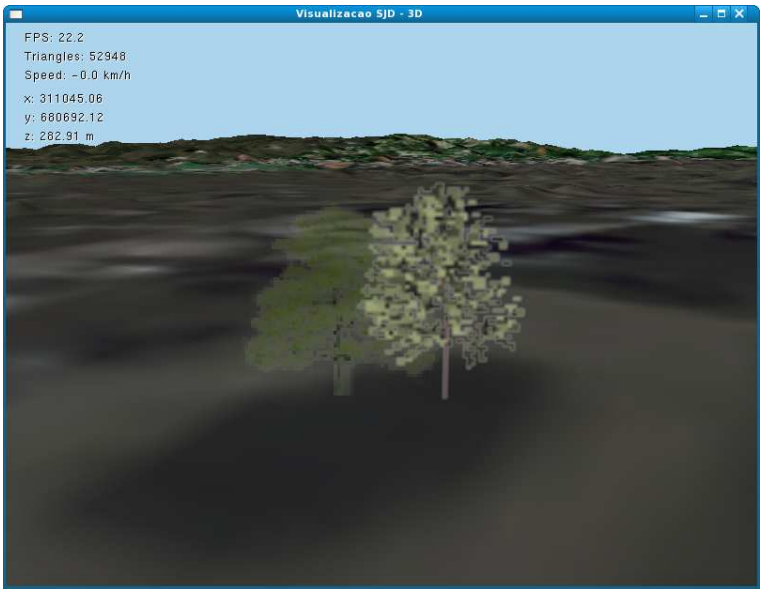


Figura 4.3: À esquerda, textura antiga; à direita textura gerada por FBO.

Assim como o recurso *alpha-channel* do OpenGL, o efeito *blur* tem pouco impacto na taxa de FPS, uma vez que não foram observadas mudanças mesmo para um grande número de elementos testados. Isso se deve ao fato de que tais recursos são realizados pela placa gráfica utilizada, livrando o processador da máquina dessa tarefa.

4.1.3 Mudanças de Vista

Note que a solução para gerar as texturas dos elementos a partir de modelos 3D apresentada na seção 4.1.2, também auxilia no problema de elementos vistos pela câmera em ângulos mais ortogonais ao terreno. Como no caso possuímos um modelo **.OBJ** lido e interpretado de forma automática pela visualização, podemos realizar operações de transformação como rotação, translação e de escala antes de realizarmos o *render to texture*. Assim, se desejamos ter uma vista superior de algum elemento, basta rotacionarmos esse modelo em 90 graus ao longo de um certo eixo e teremos a imagem vista de cima.

Claro que na vista superior uma outra estrutura deve ser utilizada no lugar de *billboards* e *sprites*. Isso porque, nesse caso em particular, essas estruturas não atendem mais às necessidades. Assim, como alternativa, propomos utilizar uma das estruturas mais

simples que existe baseada em imagens a qual denominamos *shell texture*. Essa estrutura costuma demandar menos processamento quando comparada com *billboards* e *sprites*. A razão para isso está no fato em que ela não requer transformações em sua estrutura (como acontece nos casos dos *billboards* em que uma rotação a cada *frame* se faz necessária). O outro aspecto está ligado à simplicidade, visto que um sprite pode ser encarado como duas estruturas *shell textures* disposta sob a forma de cruz.

Com a adição dessa nova funcionalidade, poderíamos estender a visualização fazendo-a considerar outros objetos diferentes de vegetação. Objetos que sofrem alterações significativas entre suas principais vistas poderiam agora ser visualizadas na aplicação. Carros, edifícios e grandes monumentos são apenas alguns dos exemplos que poderiam ser identificados em imagens de satélites e representados na visualização. Obviamente, dependendo do objeto que se deseja representar na visualização, poderíamos ter que recorrer a outros tipos de algoritmos de segmentação diferentes daquele abordado na seção 3.2. Esse novo algoritmo deverá ser capaz de identificar esses novos elementos.

Outro aspecto relevante está no momento da troca entre essas estuturas. Em outras palavras, precisamos decidir em qual momento utilizar *shell textures* ou, por exemplo, *billboards*. Seja qual for a política de chaveamento utilizada entre essas duas estruturas, o importante é manter o máximo de coerência possível fazendo com que essa troca respeite o senso comum, não comprometendo a qualidade da visualização como um todo.

Uma das estratégias de troca mais simples que existe, tem a ver com o ângulo formado entre o raio de visão da câmera e o objeto observado em questão. Podemos considerar que, a partir de um certo ângulo limite, a forma de representação deve mudar para *shell texture* naquele elemento observado, pois este é considerado visto por cima. Caso não seja atingido tal ângulo limite, então uma outra vista diferente da superior deve ser escolhida, uma vez que este elemento não é considerado visto por cima. Essa idéia é ilustrada na Figura esquemática 4.4 utilizando um ângulo $\boldsymbol{\theta}$, de $\mathbf{45}\,^{\circ}$. Observe que a imagem à esquerda, o observador está abaixo do ângulo limite, enquanto que a imagem à direita se encontra à cima do ângulo limite.

No intuito de descobrir qual é o melhor ângulo $\boldsymbol{\theta}$ que atende os requisitos de visualização, investigamos o comportamento desse ângulo limite através de um simples teste prático o qual consiste em colocar alguns *billboards* igualmente espaçados em uma certa região do terreno e definir em seguida uma única posição para a câmera. É importante lembrar que essa posição deve permanecer fixa durante todo o teste. Feito isso, fizemos o ângulo limite variar de 0 a 90 graus, observando o efeito visual da troca entre as vistas

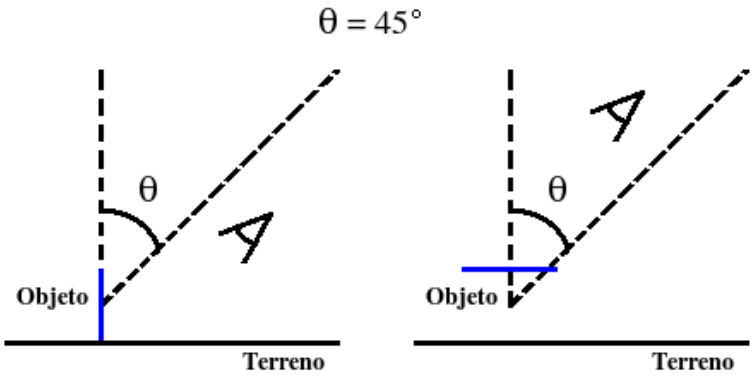


Figura 4.4: Estratégia para a seleção correta da estrutura utilizada.

frontal e superior. O resultado para alguns ângulos são mostrados na Figura 4.5. Os efeitos de *alpha-channel* e *blur* foram desligados para que possam ser observados as altas inclinações dos *billboards* mais próximos ao observador. Perceba que das três opções, a imagem do meio correspondente a $\boldsymbol{\theta = 45\,^{\circ}}$ parece ser a melhor opção de visualização para esse caso em particular estudado.

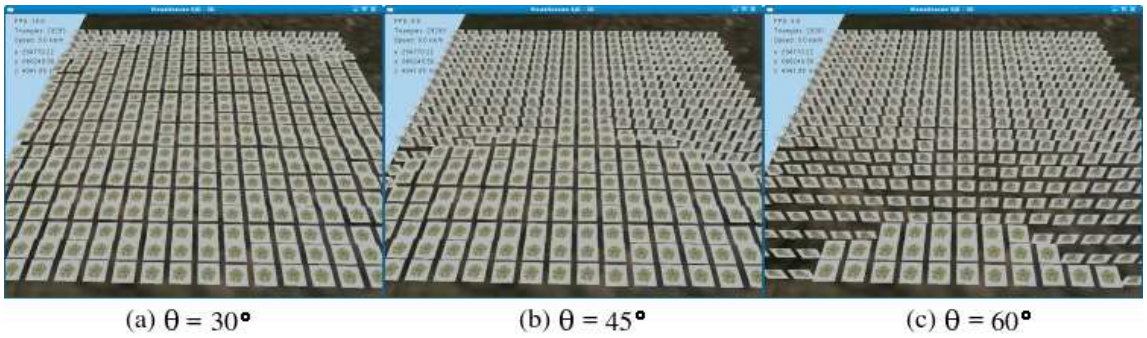


Figura 4.5: Seleção do ângulo limite ($\boldsymbol{\theta}$) utilizado na aplicação.

É importante lembrar que caso essa abordagem, por alguma razão não seja suficiente, poderíamos adaptar a estratégia atual fazendo com que esse ângulo limite seja calculado de forma automática tornando-o transparente ou não ao usuário da aplicação. Esse cálculo poderia ser descrito, por exemplo, por uma função matemática que levaria em consideração vários aspectos antes não tratados como a altura da câmera, o tipo do objeto visualizado, o ângulo de visada da câmera, dentre outros aspectos.

4.2 Contribuições no Desempenho

Dando continuidade às contribuições do trabalho, as próximas subseções propõem algumas técnicas e métodos desenvolvidos com a intenção de melhorar as atuais taxas de FPS

que não tiveram a mesma importância no trabalho anterior. Essas técnicas foram implementadas e testadas para que fossem investigados os reais impactos causados por esses métodos na taxa de FPS.

4.2.1 Comparação do Desempenho entre *Billboards* e *Sprites*

Considerando-se vistas não-ortogonais ao terreno, pode-se afirmar que as estruturas mais simples de representação baseadas em *image-based rendering* são *billboards* e *sprites*. A principal diferença entre essas duas estruturas se baseia no fato de que *billboards* precisam sofrer uma transformação de rotação a cada *frame* enquanto os *sprites* não. Em compensação, cada *sprite* precisa exibir duas imagens estáticas em forma de "cruz", aplicadas em sua estrutura base, enquanto os billboards exibem somente uma.

Considerando essa diferença conceitual entre as duas estruturas mencionadas, surgiu a necessidade de se descobrir qual delas gera o menor esforço computacional. Em outras palavras, se fez necessário verificar se rotacionar cada *billboard* envolve mais ou menos esforço computacional do que exibir duas imagens estáticas, como é o caso de *sprites*. Para poder determinar o desempenho dessas estruturas, um teste precisou ser elaborado.

O presente teste abordado nessa seção, faz variar diversas quantidades de *billboards* em três terrenos distintos. Para cada quantidade pré-estabelecida, é anotada a taxa de FPS obtida para aquele terreno visualizado. Posteriormente, esse procedimento é repetido para as mesmas quantidades de *sprites* posicionados nos mesmos lugares dos billboards em cada terreno. Novamente, foram anotados todos os valores das taxas de FPS, agora gerados por *sprites*. Para evitar o tratamento de muitas variáveis, nenhum efeito como *culling*, *alpha-channel* ou *blur* foram utilizados. O resultado do teste pode ser encontrado na Tabela 4.1 ou sob a forma de gráficos conforme a Figura 4.6. É importante resaltar que, todos os valores envolvidos no teste foram obtidos utilizando uma máquina simples e sem muitos recursos. Sua configuração inclui: processador intel pentium IV 1.60 GHz, com 512 MB de memória RAM e placa de vídeo modelo GeForce FX5200 de 256 MB.

Observe que, para a máquina descrita, as taxas máximas obtidas foram aquelas em que o terreno protótipo foi utilizado, uma vez que uma parcela muito insignificante do processamento fora destinada para a renderização do terreno simplificado. Com isso, quase a totalidade do poder computacional proveniente dos recursos da máquina pôde ser dedicado a tratar os elementos da cena envolvidos no teste. Por esse motivo, têm-se justamente, as melhores taxas de FPS dos três exemplos analisados.

Tabela 4.1: Resultados experimentais dos desempenhos de *billboards* e *sprites*.

Quantidade	Terreno protótipo		Enviro (vterrain)		SJD-Vis3D	
	billboard	sprite	billboard	sprite	billboard	sprite
100	145,0	110,4	70,3	50,3	50,0	33,3
200	108,3	74,7	46,8	32,6	39,8	25,0
300	85,4	56,3	35,6	24,1	28,6	20,7
400	66,3	42,6	29,5	18,7	25,0	14,3
500	59,5	38,9	23,7	14,8	20,0	11,8
600	52,4	32,2	21,2	13,0	18,7	10,5
700	47,5	28,4	17,1	11,4	16,7	9,5
800	42,8	25,6	16,5	9,5	14,3	8,0
900	37,5	23,1	14,0	8,5	13,3	7,4
1000	36,2	21,5	13,6	7,4	12,1	6,7

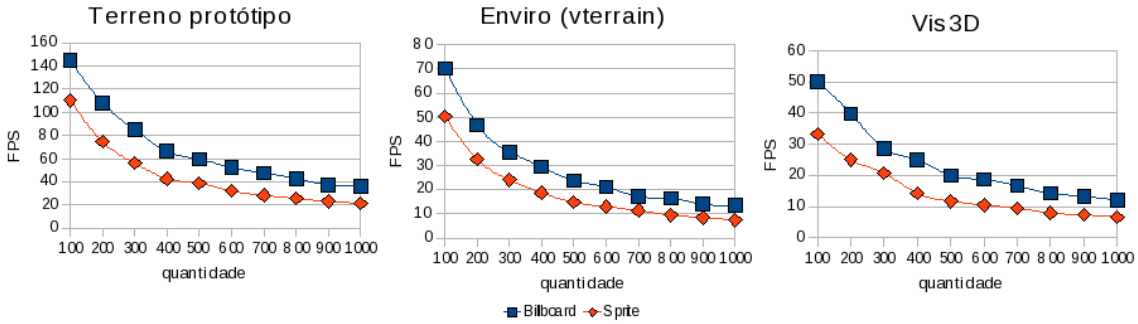


Figura 4.6: Desempenho de *billboards* e *sprites* em três visualizadores distintos.

A conclusão mais relevante que podemos tirar desses dados é que, em todos os casos amostrados, *billboards* se mostraram mais econômicos em termos computacionais do que *sprites* visto que para um mesmo recurso de hardware disponível, foi possível praticar taxas superiores de FPS em várias quantidades distintas de elementos gráficos.

4.2.2 *Culling* dos Objetos

A idéia envolvida nessa seção se baseia no fato de que, durante um vôo sobre um terreno grande, somente uma fração desse terreno é visível ao observador (usuário da aplicação de visualização). Logo, a idéia de *culling* anteriormente desenvolvida na aplicação SJD-Vis3D para o terreno foi estendida aqui para os objetos da cena. Pelas mesmas razões que motivaram o uso de *culling* no caso do terreno, entendemos que não se faz necessário enviar todos os objetos da visualização para a fila de renderização, até porque se o fizermos, muito esforço computacional estará sendo desperdiçado por conta dessa parcela não vista pelo observador. Assim, se conseguirmos descobrir uma forma rápida de identificar e descartar tais elementos, que estão fora do volume ou *frustum* de visão, acreditamos que

o desempenho da aplicação como um todo pode ser consideravelmente melhorado.

Para aplicar a técnica de *culling* precisamos entender o conceito de *frustum*. Conforme descrito em [1], o *frustum* de visão pode ser aproximado por um cubo normalizado em um sistema de coordenadas de *clipping* conforme a Figura 4.7. Note a presença de seis planos (*near, far, left, right, top* e *bottom*) que formam justamente os limites do campo de visão. Assim, para saber se um dado objeto se encontra fora do *frustum* basta comparar seus limites com cada plano do *frustum*.

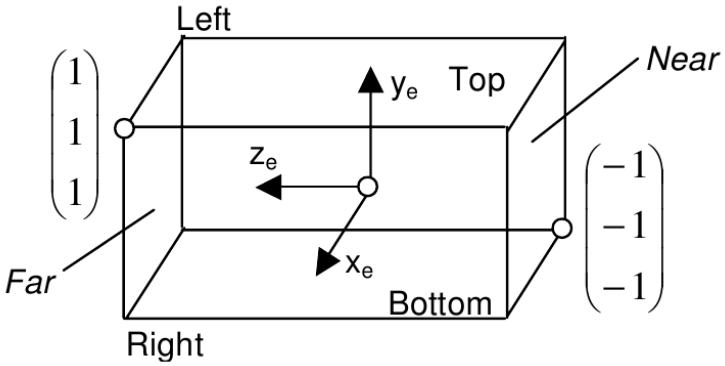


Figura 4.7: Sistema de coordenadas de *clipping* para mapear *frustums*.

O *culling* do terreno já havia sido desenvolvido previamente no trabalho anterior para os planos *left* e *right*. Aqui, a função do cálculo do *frustum* foi estendida para garantir também o eventual descarte de objetos nessas mesmas fronteiras. Além disso, o plano *near* foi colocado logo atrás da câmera para descartar todos os objetos que estão atrás dela. O plano *far* ficou suficientemente longe e por isso não foi necessário calcular os limites para esse plano. Outros cálculos foram incluídos para os testes dos planos *top* e *bottom* que até então eram inexistentes. O modo utilizado para o cálculo é similar ao utilizado no caso dos planos *left* e *right*, bastando mudar o ângulo de visada da câmera de acordo com a razão de aspecto utilizado no canvas da aplicação. Sendo assim, demostraremos somente os cálculos utilizados nos planos *top* e *bottom*.

Os cálculos envolvidos nessa operação são simples e utilizam apenas conceitos básicos de trigonometria. Mas especificamente, esses cálculos envolvem basicamente algum tipo de relação entre os principais ângulos formados pelo traçado de raios mostrado na Figura 4.8. Nessa figura, a câmera se encontra no ponto $\boldsymbol{C}$ e tem direção apontada pelo vetor $\vec{\boldsymbol{d}}$. Logo, a região sombreada denota a região efetivamente vista pela câmera naquele exato momento em particular. O ângulo $\boldsymbol{\theta}$ consiste no ângulo de visada e para demostrarmos o cálculo, um objeto foi colocado no terreno no ponto $\boldsymbol{O}$ formando um ângulo $\boldsymbol{\alpha}$ com a câmera.

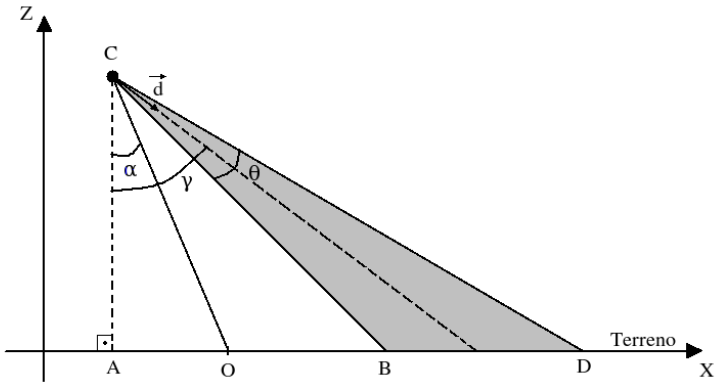


Figura 4.8: Esquema do *frustrum* baseado em ângulos para o caso de bottom.

Note que, para testar o limite inferior representado pelo ponto $\boldsymbol{B}$ devemos colocar um objeto antes desse ponto conforme indicado na Figura 4.8. Note que, ao fazer isso, teremos o valor absoluto do ângulo $\boldsymbol{\alpha}$ menor do que o do $\boldsymbol{\gamma}$. Sendo assim, resta apenas verificar a seguinte inequação 4.1:

$$\boldsymbol{\gamma - \alpha > \frac{\theta}{2}} \tag{4.1}$$

Caso ela seja verdadeira, podemos descartar o objeto com segurança já que este se encontra abaixo do limite inferior do plano *bottom* mostrado na Figura 4.7. Se for falsa, temos que continuar com os testes nos demais planos do *frustum*, já que não podemos inferir categoricamente que este elemento estará totalmente dentro do *frustum* de visão.

De maneira análoga, podemos testar o limite superior colcando um objeto depois do ponto $\boldsymbol{D}$. Assim, o valor absoluto do ângulo $\boldsymbol{\alpha}$ ficará agora maior do que o do $\boldsymbol{\gamma}$. Nessas condições, basta verificar a inequação 4.2:

$$\boldsymbol{\alpha - \gamma > \frac{\theta}{2}} \tag{4.2}$$

Da mesma forma, caso a inequação seja verdadeira, podemos descartar aquele objeto analisado. Caso contrário, não se pode chegar a nenhum tipo de conclusão até que sejam analisados os demais planos do *frustum*.

Para validar o modelo de *culling* discutido nessa seção, um percurso no terreno que sobrevoa exatamente mil objetos foi montado. Durante esse percurso, a direção de visualização não foi alterada. Além disso, foram coletadas amostras de alguns trechos ao longo desse percurso anotando, para cada uma dessas amostras, a quantidade de elementos que

foram descartados e a taxa de FPS praticada naquele instante. Um esquema do percurso utilizado pode ser visto na Figura 4.9 e os resultados colhidos nos pontos de amostragem para o terreno da SJD-Vis3D pode ser visto na Tabela 4.2. É importante resaltar que, todos os valores envolvidos nesse teste foram obtidos utilizando uma máquina com os seguintes recursos: processador AMD 64 bits 3500+, com 2.0 GB de memória RAM e placa de vídeo modelo GeForce 8400GS. O sistema operacional instalado nesse computador é o Linux pertencente à distribuição Fedora 8 de 32 bits.

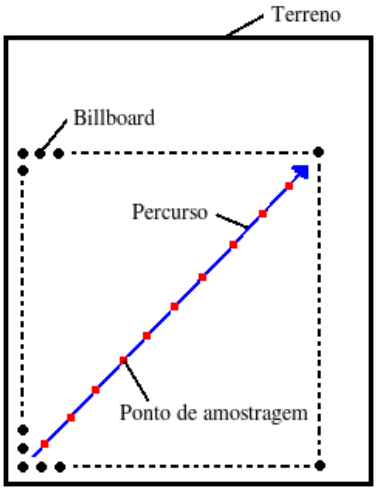


Figura 4.9: Esquema para descrever o percurso utilizado no teste.

Tabela 4.2: Resultados experimentais do culling.

Ponto	Sem *culling*	Com *culling*	
	FPS	FPS	Objetos descartados
(292292;670834)	26,9	27,5	15
(295269;673519)	26,8	31,9	232
(296627;674608)	27,3	34,7	314
(298142;676111)	27,5	38,7	415
(299883;677816)	28,1	44,9	521
(301933;679527)	28,9	53,5	621
(303541;681418)	29,2	61,7	708
(303541;683565)	30,3	81,5	806
(309065;686858)	31,0	117,3	900
(311445;688109)	32,0	156,7	943

Observe que, no caso em que não foi aplicado o *culling* nos objetos durante o percurso, era de se esperar, teoricamente, que a taxa de FPS permanecesse sempre constante, visto que nenhum objeto é descartado da fila de renderização. Entretanto, conforme os resultados práticos da Tabela 4.2, observamos claramente um ligeiro aumento dessa taxa à medida em que nos aproximamos do ponto final do percurso. O surgimento desse acréscimo pode ser explicado se levarmos em consideração o fato de que, no final do percurso,

uma menor quantidade de primitivas do terreno deve ser desenhada, pois afinal, o *culling* do terreno é mantido ligado durante todo o teste. Como o *culling* do terreno está presente nos dois casos analisados no teste (com e sem *culling* de objetos), acreditamos que essa interferência não invalida a conclusão de que a implementação do *culling* de objetos é benéfica para a aplicação de visualização.

4.2.3 Multi-Resolução nos Objetos

Sabemos que não há coerência em se utilizar texturas com mesma resolução para todos os objetos da cena. Considerando o fato de que nem todos os elementos estão a uma mesma distância do observador, podemos tirar vantagem desse fato atribuindo texturas com uma maior quantidade de *pixels* para elementos mais próximos do observador e texturas com menos *pixels* para elementos mais distantes.

Assim, desde que a troca dessa textura se dê de forma rápida, automática e dinâmicamente durante a visualização, entendemos que o uso de multi-resolução de textura poderia beneficiar muito a visualização como um todo. Assim, foi preparado um teste para averiguar o comportamento da visualização na presença de diversas texturas em diversas resoluções diferentes.

O teste consiste em selecionar quantidades fixas de *billboards* para serem colocados no terreno fazendo variar as resoluções das texturas aplicadas nas estruturas de cada um desses *billboards*. Para cada resolução examinada, foram anotados a taxa de FPS obtida. O resultado do teste pode ser observado na Tabela 4.3. A máquina utilizada possui os seguintes recursos: processador AMD 64 bits 3500+, com 2.0 GB de memória RAM e placa de vídeo modelo GeForce 8400GS. O sistema operacional instalado nesse computador é o Linux pertencente à distribuição Fedora 8 de 32 bits.

Note que as taxas de FPS caem à medida em que texturas maiores são utilizadas em cada quantidade observada correspondendo ao comportamento esperado.

4.2.4 Estrutura de Armazenamento dos Objetos

Sabemos que dois possíveis algoritmos para resolver o clássico problema de ordenação de um conjunto de números são os chamados *insertion sort* e *heap sort* [62]. Sabemos também que o segundo algoritmo realiza, no pior caso, a tarefa em menos tempo do que o primeiro. Parte do sucesso do *heap sort* se deve em função da forma com o qual o algoritmo armazena os números, facilitando a tarefa de ordenação. Logo sabemos que,

Tabela 4.3: Resultados experimentais da multi-resolução.

Quantidade	FPS			
	32x32	64x64	128x128	256x256
100	80,8	66,5	45,0	11,2
200	71,3	57,5	31,2	5,8
300	62,3	48,6	23,3	4,3
400	55,6	40,2	19,2	3,1
500	52,3	37,0	16,3	2,5
600	48,2	33,5	13,2	2,1
700	43,7	29,4	11,7	1,8
800	40,9	27,5	10,7	1,6
900	38,7	25,5	9,6	1,4
1000	36,2	23,2	8,9	1,2

dependendo da estrutura de dados utilizada, é possível melhorar o desempenho final de algumas tarefas.

Com base nessa abordagem, imaginamos duas formas de representação alternativas à lista simples, utilizada no trabalho anterior e descrita na seção 3.4. A primeira definição faz uso de um conceito de agrupamento simples no qual denominamos *blobs* (que do inglês significa "gota"). A outra forma, bem mais complexa, recebe o nome de *quadtree*, onde os objetos são agrupados respeitando uma certa hierarquia.

Nas próximas subseções, mostraremos como funciona cada técnica citando suas vantagens e desvantagens. Posteriormente, faremos um teste para verificar qual das estruturas (lista simples, *blobs* ou *quadtree*) funciona de forma mais adequada.

4.2.4.1 Estrutura Utilizando *Lista Simples*

O conceito de armazenamento digital dos dados dos objetos introduzido no trabalho anterior e apresentado aqui durante o capítulo 3, faz uso de uma estrutura de dados extremamente simples: uma lista. Assim, para cada objeto gerado pelo *split-and-merge*, este é colocado na lista.

4.2.4.2 Estrutura Utilizando *Blobs*

Nessa abordagem não descartamos de todo o conceito de lista simples utilizado inicialmente no trabalho anterior. Entretanto, algumas mudanças foram exploradas de modo a tentar percorrê-la mais rapidamente.

Tendo em vista esse objetivo, propomos a utilização de uma técnica na qual deno-

minamos de *blobs*. Dado uma constante K, um *blob* é um conjunto de objetos situados em uma mesma região geográfica do terreno delimitada por um quadrado subjetivo e imaginário de dimensão $K * K$. Assim, a tradicional lista de objetos individuais agora é substituída por uma lista de *blobs*. Como cada *blob* representa um conjunto desses objetos, espera-se que essa lista de objetos seja reduzida. Com isso, os algoritmos realizam tarefas não mais em cima de cada objeto individualmente e sim, em cima de um único *blob*.

A construção da lista de *blobs* é feita da seguinte forma: inicialmente, pega-se o primeiro objeto da lista simples. Em seguida, varre-se todo o restante da lista em busca de novos objetos que estejam situados na mesma região geográfica compreendida no espaço delimitado por um quadrado de dimensão $K * K$, o qual deve conter a posição do primeiro objeto lido no centro dessa região. Note que K precisa ser um valor inteiro e constante previamente definido antes de se começar o algoritmo. Assim, todos os objetos que estão contidos nessa região (inclusive o objeto do centro) são retirados da lista simples e o *blob* é adicionado em uma lista de *blobs*. Cada *blob* guarda informações do tipo: quantidade de objetos, delimitações geográficas e uma lista de todos os objetos que pertencem àquele *blob*. Depois desse primeiro passo, todo o processo é repetido para os objetos remanescentes da lista simples. O processo termina quando a lista de objetos se encontra vazia.

Tendo entendido o conceito de *blobs* desenvolvido e implementado especialmente para esse trabalho, fica claro que, quanto maior o valor atribuído à constante K, menos *blobs* tendemos a ter. Nessas situações, a quantidade de objetos associado a cada *blob* é grande assumindo, no máximo, a quantidade total de elementos no terreno. Por um outro lado, caso tenhamos K um valor demasiadamente pequeno, teremos a lista de *blob* do mesmo tamanho que a lista simples de objetos já que cada *blob* tenderá a ter somente um único objeto associado. Isso pode ser observado com mais clareza através do exemplo mostrado na Figura 4.10. Na imagem (a) temos um terreno com vista superior o qual exibe algumas posições de objetos denotados por pontos. Nessa imagem o algoritmo não foi aplicado e por esse motivo não há *blobs*. Na imagem (b), o algoritmo de construção de *blob* gerou uma lista utilizando um valor de K pequeno. Em (c) o mesmo procedimento foi aplicado porém agora para um K maior do que aquele utilizado em (b). Note que o número total de *blobs* em (b) é maior do que em (c) pois correspondem, respectivamente, às quantidades 19 e 14. Por outro lado, o número médio de objetos/*blobs* em (b) é menor do que em (c) pois correspondem respectivamente às taxas 1,58 e 2,07.

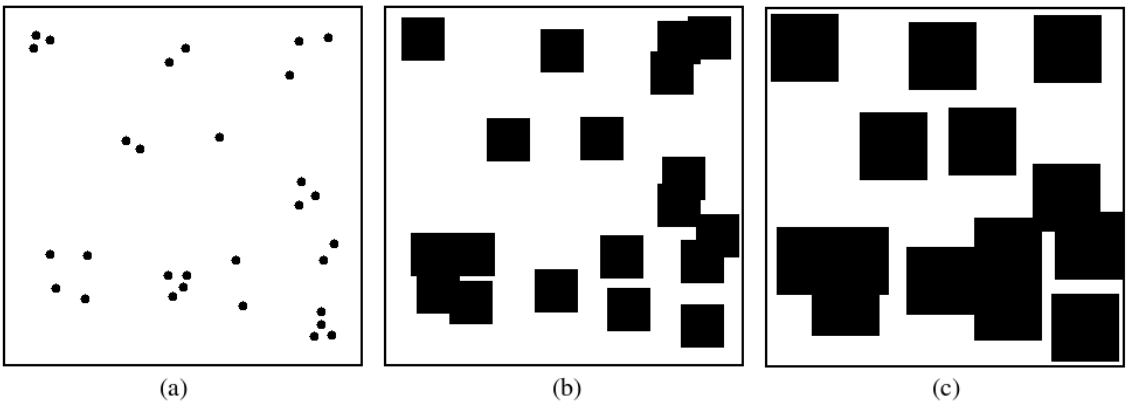


Figura 4.10: Geração dos *blobs* com diferentes constantes $\boldsymbol{K}$.

No momento em que o processo de renderização é realizado, somente alguns *blobs* serão considerados visíveis pelo observador e por esse motivo, nem todos os objetos serão desenhados no canvas OpenGL. O critério utilizado leva em consideração o *culling* dos objetos que agora foi adaptado para o caso dos *blobs*. Nessa abordagem, não fazemos o teste do *culling* para cada objeto da lista simples, mas sim, para cada *blob* presente na lista de *blobs*. Em outras palavras, caso todo o domínio de um certo *blob* esteja fora do *frustum* de visão, então este *blob* é descartado do processo de renderização e nenhum de seus objetos são desenhados. Caso contrário, mesmo que apenas uma pequena parcela do domínio desse *blob* se encontre dentro do *frustum* de visão, então todos os objetos daquele *blob* são desenhados. Outro melhoramento importante obtido se deve o fato de que, somente *blobs* considerados próximos ao observador serão visíveis. Logo, ainda que exista um *blob* totalmente dentro do *frustum* de visão, mas que esteja suficientemente distante do observador, este será desconsiderado e seus objetos não serão desenhados.

Uma das vantagens do uso dessa técnica está no fato de que quando um *blob* é considerado descartado, todos os objetos pertencentes àquele *blob* são descartados. Logo, a busca pelos *blobs* se dá mais rapidamente quando comparado com a busca por objetos individuais no caso da lista simples. Outra vantagem está no fato de termos o tamanho do *blob* ajustável. Esse ajuste se dá por intermédio da constante K definida antes da aplicação se iniciar. Assim, temos mais controle sobre a visualização, o que nos garante uma maior flexibilidade e adequação à real situação visualizada.

Como principal desvantagem, podemos citar o fato de que, deixar de desenhar simultâneamente todos os objetos pertencentes a um *blob*, pode causar um efeito indesejado de descontinuidade, em especial quando utilizado valores relativamente grandes para a constante K. Como consequência, mais oscilante tende a ficar a taxa de FPS, uma vez que, a cada *blob* desenhado ou descartado, um grande número de objetos passam a ser

desenhados ou descartados. Logo, um pequeno deslocamento na câmera poderia incluir um novo *blob* repleto de objetos, baixando, de repente, a taxa de FPS.

4.2.4.3 Estrutura Utilizando *Quadtree* com Agrupamentos

A abordagem utilizada aqui muda radicalmente a estrutura de representação e armazenamento dos objetos da cena. Até o presente momento, cada objeto da cena era representado individualmente utilizando uma única forma de representação, seja ela dada por um *billboard* ou *sprite*. Entretanto, acreditamos ser possível representar mais de um objeto em uma só textura se estes elementos estiverem suficientemente longe da câmera e relativamente perto um do outro. Se essa representação for mesmo possível e viável, existe a chance de economizar tempo de processamento já que menos *billboards* ou *sprites* seriam utilizados, não comprometendo ainda, a qualidade da visualização. Para decidirmos que agrupamento dar a esses objetos, precisamos explorar uma estrutura que embutisse o conceito de hierarquia, onde poderíamos descer ou subir ajustando a tolerância dada a esses agrupamentos.

Com esse objetivo, propomos a substituição da lista simples de objetos por uma árvore de divisão quaternária (ou *quadtree*). Essa estrutura permite utilizar uma hierarquia de representação dos objetos da cena onde as folhas dessa árvore armazenariam os objetos individualmente enquanto os demais nós armazenariam algum agrupamento desses mesmos objetos.

Como não é possível determinar de antemão os agrupamentos, foi necessário a inclusão da técnica chamada *Frame Buffer Object* (ou FBO). Essa técnica faz uso de *buffers* especiais do OpenGL, os quais permitem capturar imagens de agrupamentos de objetos em tempo de execução. Essa operação é transparente ao usuário que nem se quer toma conhecimento da aplicação dessa técnica. Logo, as imagens dos agrupamentos são geradas no momento em que elas se fazem necessárias durante o próprio processo de construção da *quadtree*.

A construção da *quadtree* transcorre da seguinte maneira: Inicialmente, todo o domínio do terreno é considerado, e informações como nível zero e quantidade total de objetos são armazenadas no nó raíz da *quadtree*. A partir daí, esse domínio é subdividido recursivamente em quatro partes iguais até que se tenha a quantidade de um ou zero objetos em alguma subdivisão subseqüente. Cada subdivisão recebe o nome de *nó* ou simplesmente *quad*. Cada *quad* é definida como mostrado na listagem da Figura 4.11. Como exemplo, ilustramos esse processo para um pequeno terreno hipotético que pode ser observado na

Figura 4.12. Em (a), temos um terreno e as posições dos objetos. Em (b) temos o desenvolvimento do algoritmo de construção da *quadtree*. Por fim, em (c) temos a *quadtree* gerada para o caso em (b). Durante a geração dessas *quad*, utiliza-se o FBO para capturar as principais vistas ortogonais daquele agrupamento. Essas capturas geram texturas que são armazenadas na própria *quad*.

```
struct QuadElements
{
    int texIds[5];                /* guarda as texturas top, front, back, ... */
    int totalElements;            /* total de elementos na quad. */
    int depth;                    /* profundidade corrente (root depth = 0). */
    float xmin, xmax, ymin, ymax; /* coordenadas da quad. */
    struct QuadElements * pai;    /* sub-divisao UpperLeft da quad. */
    struct QuadElements * ul;     /* sub-divisao UpperLeft da quad. */
    struct QuadElements * ur;     /* sub-divisao UpperRight da quad. */
    struct QuadElements * ll;     /* sub-divisao LowerLeft da quad. */
    struct QuadElements * lr;     /* sub-divisao LowerRight da quad. */
    struct B_list * inicio;       /* ponteiro para inicio da lista de elementos.*/
    struct B_list * fim;          /* ponteiro para fim da lista de elementos.*/
};
```

Figura 4.11: Listagem contendo os atributos de cada *quad*.

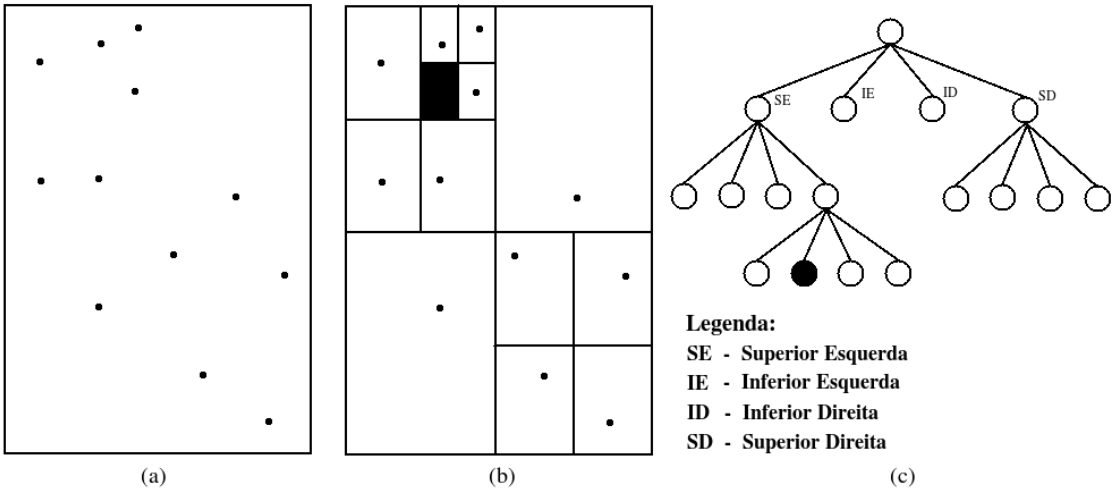


Figura 4.12: Exemplo da construção de uma *quadtree* para um terreno hipotético.

No momento da renderização, algumas regiões são representadas com nível de agrupamento maior do que outras. Para isso, um critério de escolha precisou ser criado para tomar as decisões corretas à cerca do nível mais apropriado para cada *quad* visualizada. Uma das abordagens mais simples consistiu em utilizar a distância para fazer essa escolha. Assim como mostrado na Figura 4.13, quanto mais distante do observador uma *quad* está, menor o nível de agrupamento tolerado pelo algoritmo. Análogamente, quanto mais próximo do observador, maior deve ser o nível de representação. Uma outra possibilidade para a seleção do nível de detalhe apropriado, consistiria em levar em conta não só a

distância, mas também o ângulo como exemplo na métrica proposta em [63] aplicada em impostores.

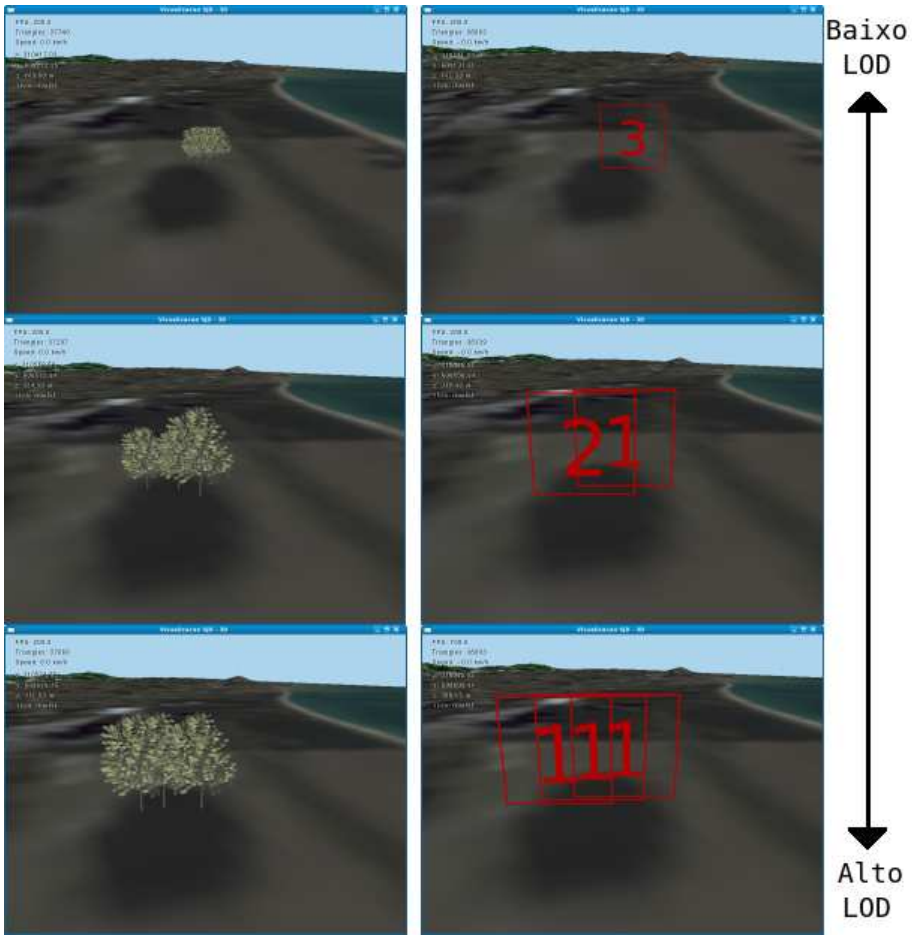


Figura 4.13: Seleção do LOD baseado na distância entre uma *quad* e o observador.

Outro aspecto importante inclui ainda o conceito de *culling*. Quando está se utilizando *quadtree*, o *culling* precisa ser adaptado de modo que, uma *quad* situada fora do *frustum* de visão não são desenhadas independentemente de ter ou não aglomerações.

Através de sucessivas observações feitas em diversos sobrevôos que utilizaram o conceito de agrupamentos, percebemos que, para os casos em que a região correspondente à *quad* selecionada para renderização é bem maior do que o tamanho físico da região padrão utilizada na criação do *billboard*, a utilização da textura aproximada não condiz com exatidão o espalhamento real dos objetos naquela *quad*. Tal efeito acontece pois, nessas situações, a largura do *billboard* é inferior a largura da *quad*, conforme se pode observar através da Figura esquemática 4.14, mas específicamente na situação da *quad* descrita em (a). Como conseqüência, se esse fenômeno ocorrer simultâneamente em várias *quads* vizinhas, poderemos ter uma impressão errônea a cerca da densidade dos objetos naquela região.

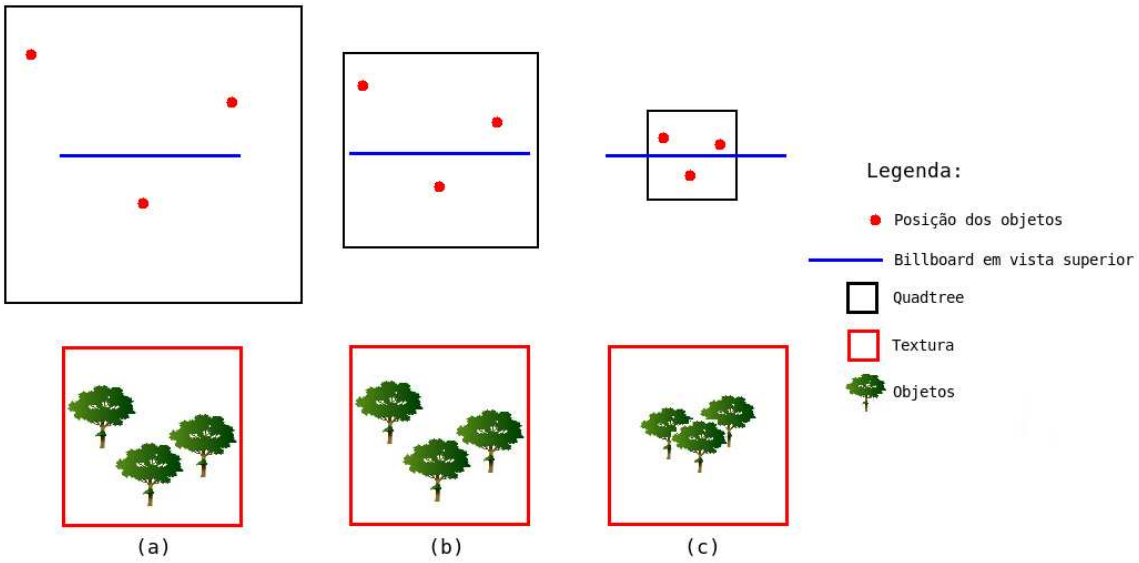


Figura 4.14: O espalhamento dos objetos na textura dos *billboards*.

Uma solução simples para resolver esse problema seria calcular todas as larguras e alturas de cada *quad* no momento da construção da *quadtree*. Assim, no momento da renderização, caso a *quad* selecionada contenha dimensões que sejam muito superiores aquela padrão do *billboard*, esta deverá ser subdividida novamente até que se tenha um tamanho compatível. Uma outra solução possível, seria generalizar o tamanho dos *billboards* (estrutura e textura) de modo que eles possam ser desenhados de acordo com o tamanho de sua *quad* correspondente. Essa solução parece mais promissora do que a anterior visto que evitará descer níveis da árvore quaternária, diminuindo assim, a quantidade efetiva de objetos a serem renderizados. Por um outro lado, esta última solução demandará um esforço para a elaboração de uma métrica capaz de ajustar automaticamente os parâmetros do *canvas* do FBO, para que este se ajuste adequadamente aos mais variados tamanhos, auxiliando no processo de geração de texturas.

Assim, diante dessas e outras possibilidades de se contornar esse problema, entendemos que é preciso investigar detalhadamente cada uma delas para que possamos averiguar seus desempenhos para fins de comparação.

4.2.4.4 Comparação entre as Estruturas

Como a estrutura de *quadtree* é construída de modo a formar uma hierarquia de aglomeração de objetos, é de se esperar que menos estruturas baseadas em imagens como *billboards* e *sprites* sejam de fato desenhadas à cada *frame*, já que agora não temos somente a correspondência de 1:1 entre o número de objetos da cena e a estrutura baseada

em imagem. Pelo fato de podermos agrupar dois ou mais objetos em uma única estrutura de representação, proporções como 2:1, 3:1, 4:1, dentre outras, passam a ser aceitas e tratáveis. Como efeito, há uma tendência na diminuição da quantidade efetiva de estrutras baseadas em imagem que devem ser renderizadas. Com essa diminuição, espera-se ainda que a taxa de FPS aumente significativamente, melhorando o desempenho da aplicação como um todo.

Por um outro lado, a estrutura de *quadtree* é bem mais complexa do que aquelas propostas anteriormente. Por esse motivo, sabemos que essa estrutura pode vir a acarretar em um *overhead* maior do que aqueles gerados pelas outras estruturas, causando um impacto negativo no desempenho da aplicação. Assim, precisamos elaborar um teste que nos comprove experimentalmente se a estrutura de *quadtree* é mais vantajosa do que as demais estruturas discutidas anteriormente.

Como desejamos investigar as estruturas citadas, precisamos manter o máximo de coerência no teste aplicado a elas. Isso implica considerar as mesmas características e configurações da aplicação em todos os casos de teste analisados. Tais configurações incluem:

- Utilização do mesmo terreno e textura.
- Não utilização de técnica de *culling* nos objetos;
- Utilização somente de *billboards* com texturas de tamanho 64x64 *pixels*;
- Não utilização de nenhum efeito adicional do openGL como *fog*, *alpha-channel* e *blur*;
- A câmera deverá ter os mesmos parâmetros em todos os casos, incluindo as mesmas localizações geográficas no terreno e vetor direção;

Além disso, utilizaremos *blobs* do mesmo tamanho e equivalente a 3.000 metros em coordenadas do mundo real para o terreno de *Itaoca-ES*. Outra consideração importante é a forma escolhida para as aglomerações feitas no caso em que a *quadtree* é utilizada. Nesse caso, testamos mais de uma possível configuração de agrupamento. Entretanto, em todas as configurações, utilizamos sempre três níveis de agrupamentos aceitáveis. Eles são construídos em função da distância $\boldsymbol{d_1}$, $\boldsymbol{d_2}$ e $\boldsymbol{d_3}$ do observador gerando as regiões denotadas por I, II e III na Figura 4.15. Como objetos que distam mais do que $\boldsymbol{d_3}$ do observador são automaticamente descartados pelo processo do *culling*, tivemos que

atribuir a distância $\boldsymbol{d_3}$ um valor demasiadamente grande (maior do que o comprimento da diagonal do terreno), pois assim garantimos que nenhum objeto será descartado no processo de teste.

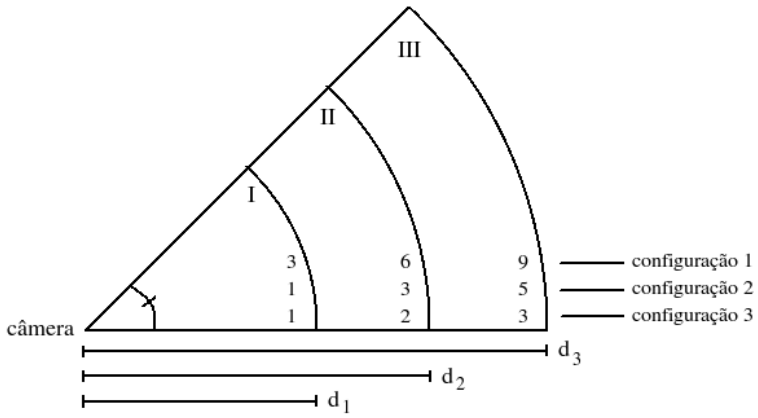


Figura 4.15: As três configurações testadas na estrutura de *quadtree*.

Assim, para efeito de exemplificação, observe que a Figura 4.15 nos informa que quando utilizada a configuração 3, somente seis ou menos *billboards* poderão ser agrupados na região intermediária (região compreendida entre as distâncias $\boldsymbol{d_1}$ e $\boldsymbol{d_2}$). Já na configuração 1, essa mesma região suportará somente agrupamentos de dois objetos ou menos.

Os testes realizados aqui foram implementados em uma máquina com os seguintes recursos: processador AMD 64 bits 3500+, com 2.0 GB de memória RAM e placa de vídeo modelo GeForce 8400GS. O sistema operacional instalado nesse computador é o Linux da distribuição Fedora 8 de 32 bits. Os valores de FPS presentes na Tabela 4.4 são médios, já que diferentes pontos do terreno foram levados em consideração no teste. Esses pontos consistem em um subconjunto daqueles que compõem o percurso visto na seção 4.2.2, mas especificamente através do esquema ilustrado na Figura 4.9. Pelos mesmos motivos de trabalharmos com FPS médios, a quantidade real de *billboards* obtidos no caso da *quadtree* também são médias.

Note ainda que nessa tabela a taxa FPS1 corresponde à taxa de FPS média obtida utilizando a configuração 1 de agrupamento mostrada na Figura 4.15. Analogamente, as taxas FPS2 e FPS3 são as taxas obtidas nas configurações 2 e 3 respectivamente. Outra observação é o fato de que os valores mostrados para a *quadtree*, representam a quantidade de *billboards* que são redesenhados a cada quadro e não o total de objetos que eles representam.

Para os números de *billboards* examinados no teste, pudemos observar na Tabela 4.4

Tabela 4.4: Resultados experimentais das estruturas de representação utilizadas.

Objetos	Lista	Blob	Quadtree					
	FPS	FPS	Objetos	FPS1	Objetos	FPS2	Objetos	FPS3
100	85,9	84,6	25	98,8	60	90,3	78	90,9
200	69,3	68,9	49	92,7	125	75,9	157	71,8
300	58,5	58,5	71	87,3	166	67,6	219	62,9
400	50,7	50,5	120	77,1	219	60,9	300	54,8
500	45,0	44,6	143	71,3	266	55,7	417	45,1
600	39,9	40,0	163	68,9	327	49,4	486	40,2
700	36,2	36,2	171	67,0	394	45,0	556	37,2
800	33,0	33,0	197	63,1	479	40,0	608	35,0
900	30,3	30,4	197	62,9	531	36,9	660	32,2
1000	27,9	28,3	201	62,8	641	33,0	729	30,8

que a estrutura que apresentou as melhores taxas de FPS foi a estrutura de *quadtree*. Em todas as configurações de agrupamento testados os resultados superaram as estruturas lista simples e *blobs*. Isso mostra que mesmo sendo uma estrutura bem mais complexa, a *quadtree* diminui o volume de *billboards* efetivamente representados na cena através da aglomerações de objetos. As vantagens obtidas pelo uso e implementação de aglomerados, por sua vez, superam a complexidade da estrutura utilizada. Pode-se afirmar que utilizando-se uma maior taxa de aglomeração, é esperado que a taxa de FPS aumente ainda mais. Neste caso, entretanto, é necessário uma averiguação mais cuidadosa dos impactos da representações na qualidade visual.

4.3 Resultados Finais

Devido ao superior desempenho obtido pela estrutura *quadtree*, reservamos esta seção para mostrar alguns dos resultados visuais obtidos através da implementação dessa técnica.

Através das Figuras 4.16–4.23, podemos ver o comportamento da estrutura *quadtree* aplicada em situações envolvendo dados reais. Para isso, foram utilizados ambos os terrenos de Itaoca-ES e Macaé-RJ. As configurações utilizadas nos principais parâmetros da *quadtree* estão descritas logo abaixo das respectivas imagens. Essas configurações seguem o estilo daquelas mostradas na seção 4.2.4.4. Já os resultados finais obtidos podem ser encontrados na Tabela 4.5

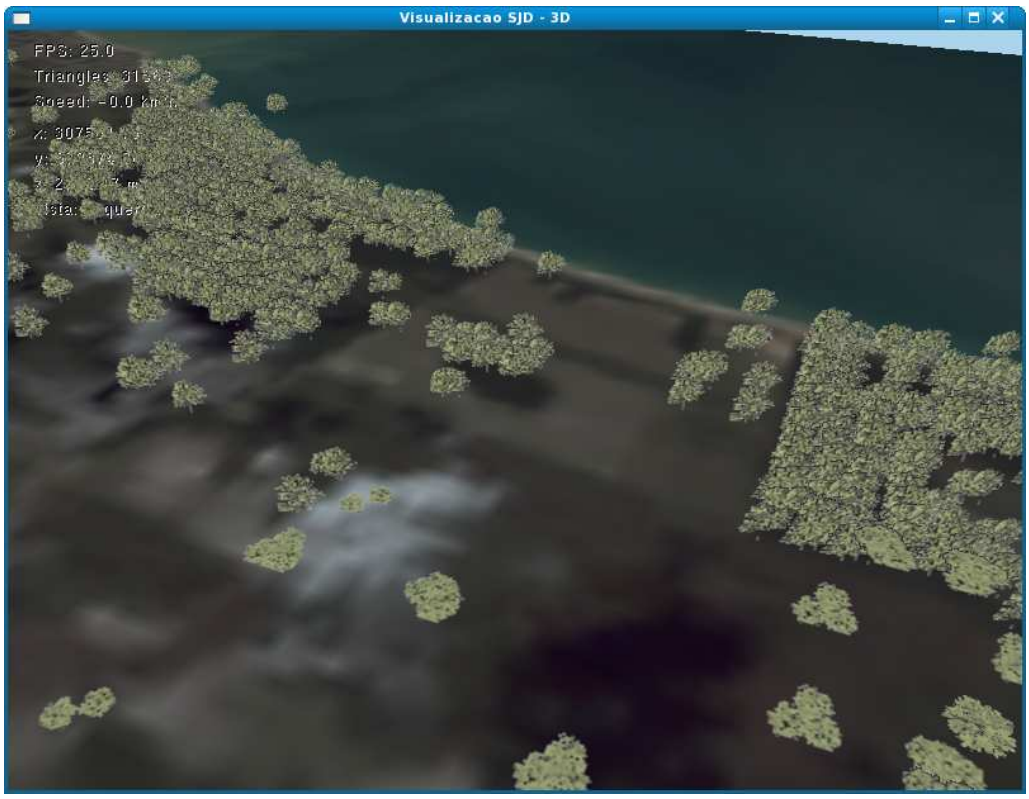


Figura 4.16: Visualização de 2050 objetos no terreno de *Itaoca-ES*

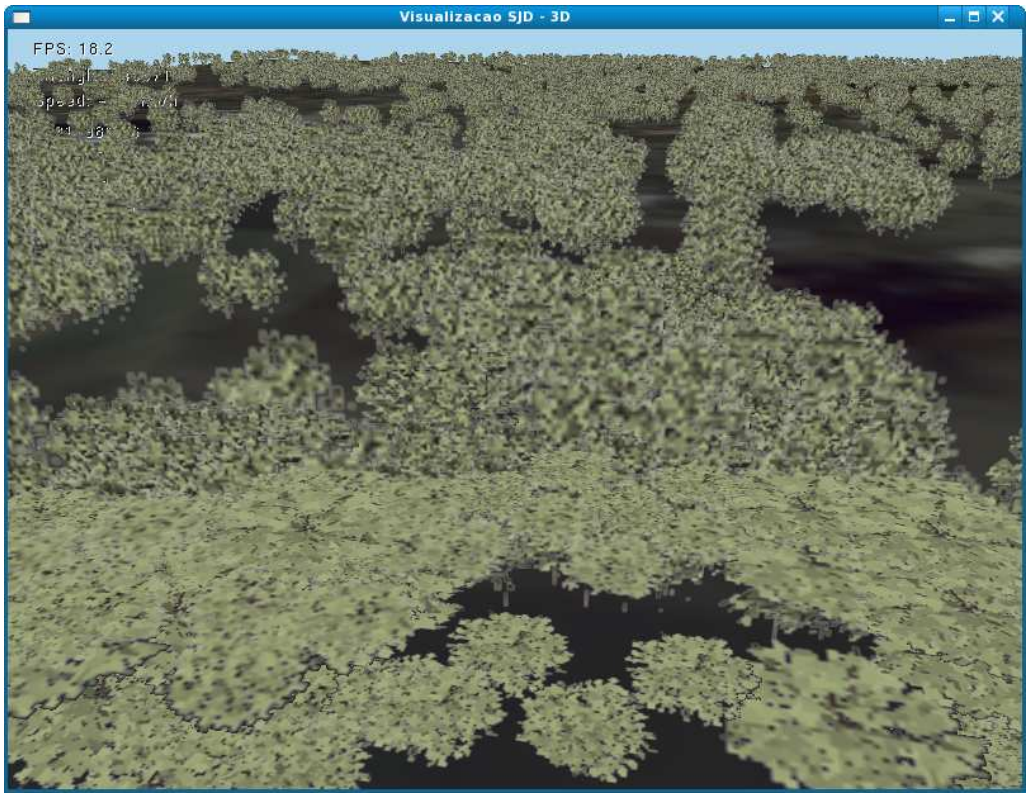


Figura 4.17: Visualização de 6762 objetos no tererno de *Itaoca-ES*

Figura 4.18: Visualização de 2010 objetos no terreno de *Itaoca-ES*

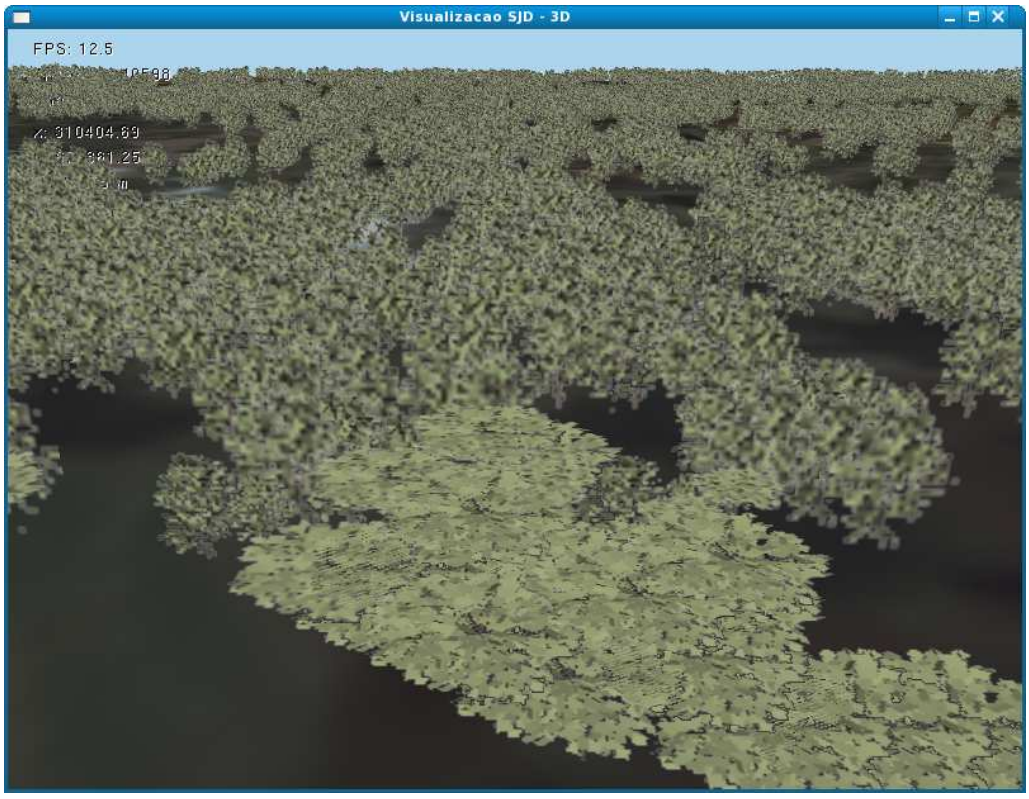


Figura 4.19: Visualização de 9277 objetos no terreno de *Itaoca-ES*

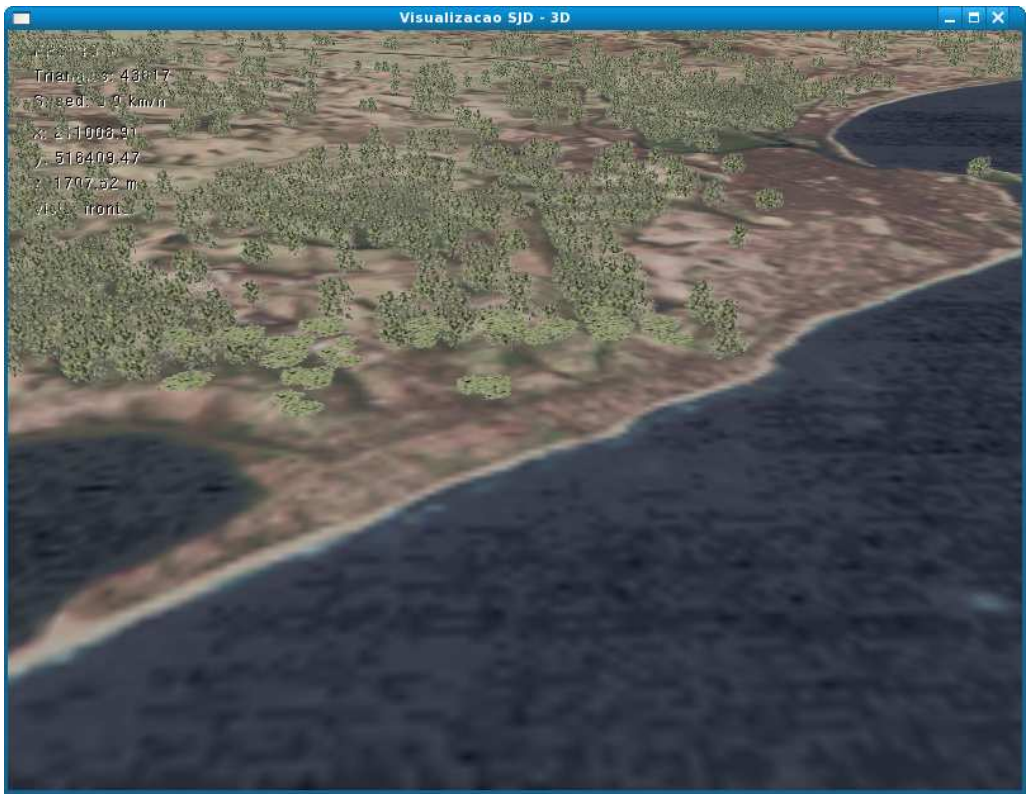


Figura 4.20: Visualização de 14219 objetos no terreno de *Macae-RJ*

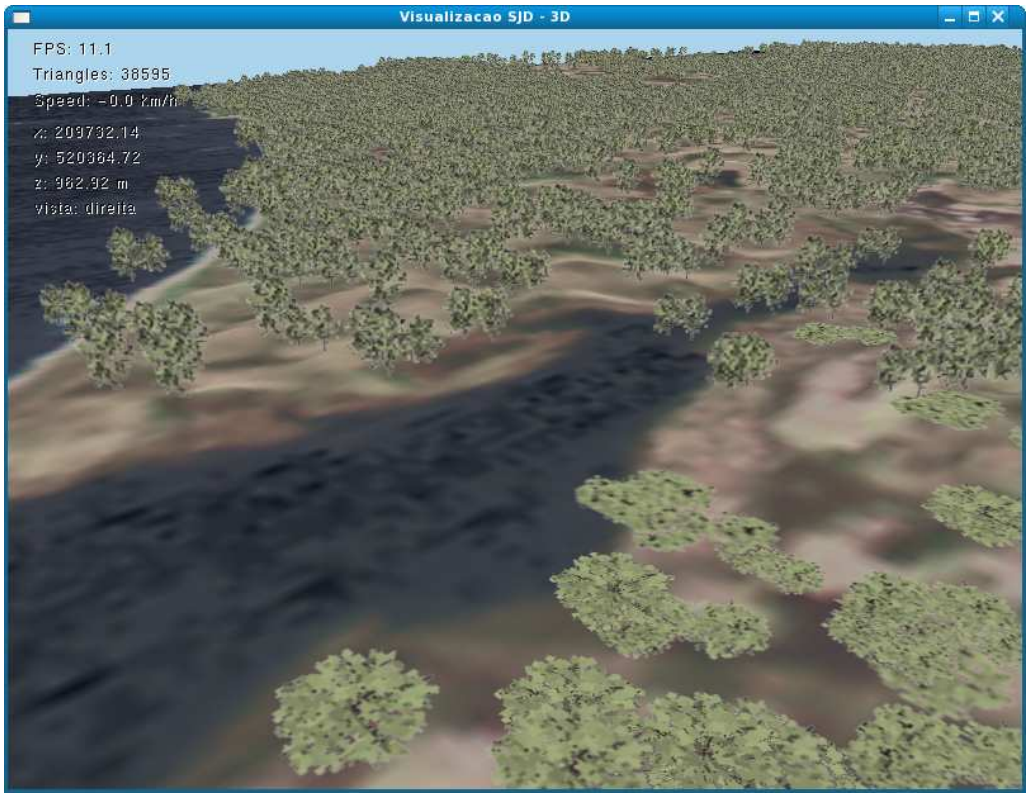


Figura 4.21: Visualização de 10385 objetos no terreno de *Macae-RJ*

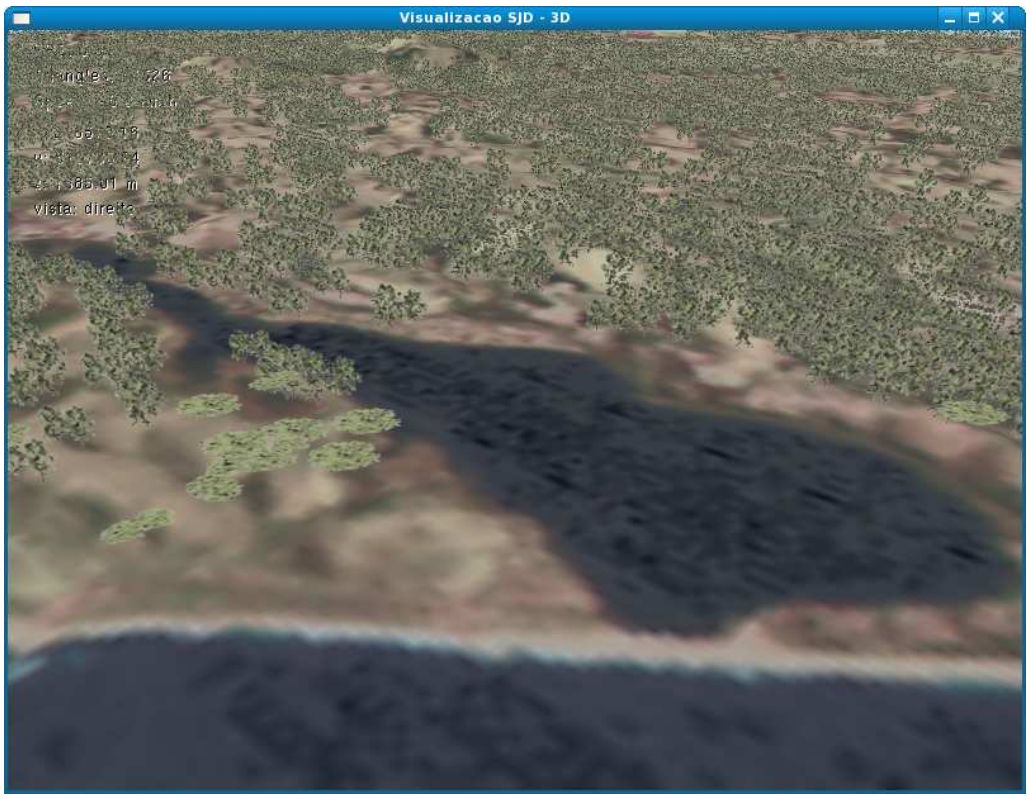


Figura 4.22: Visualização de 17012 objetos no terreno de *Macae-RJ*

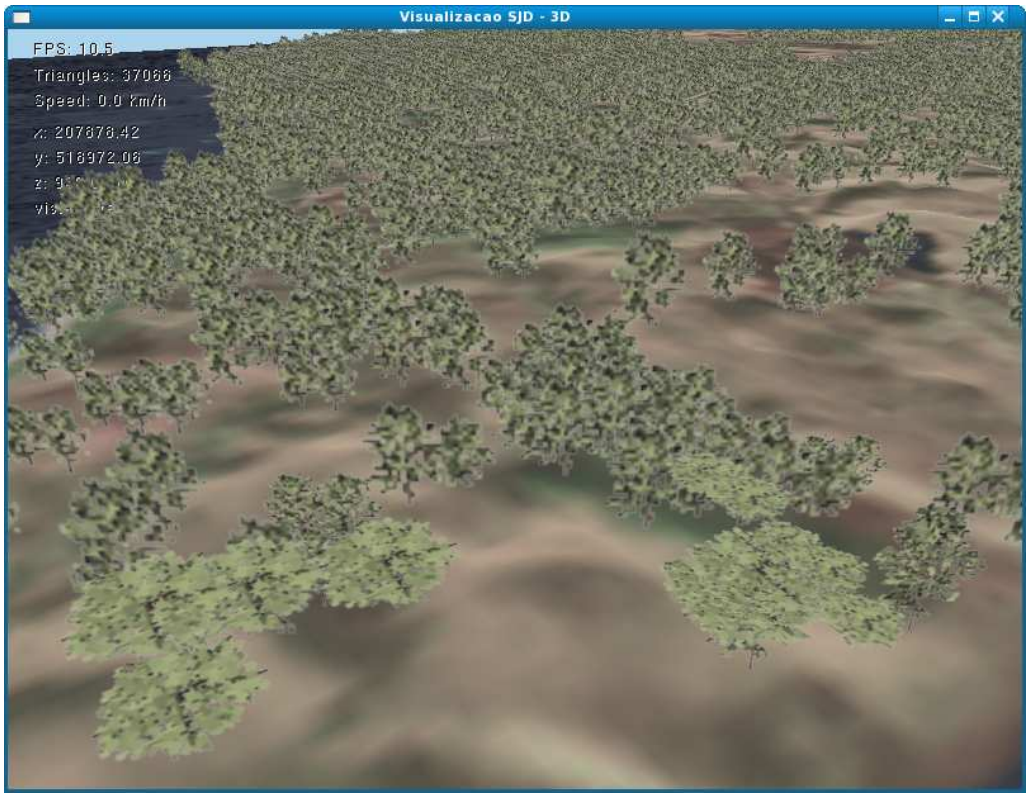


Figura 4.23: Visualização de 7715 objetos no terreno de *Macae-RJ*

Tabela 4.5: Resultados experimentais obtidos nas Figuras 4.16 – 4.23.

Figura	Propriedade da *quadtree*	FPS	objetos reais	objetos simbólicos
Figura 4.16	configuração 1	25,0	486	2050
Figura 4.17	configuração 1	18,2	1660	6762
Figura 4.18	configuração 2	22,2	817	2010
Figura 4.19	configuração 2	12,5	3721	9277
Figura 4.20	configuração 2	10,5	5726	14219
Figura 4.21	configuração 2	11,1	4139	10385
Figura 4.22	configuração 3	5,9	10056	17012
Figura 4.23	configuração 3	10,5	4690	7715

4.4 Conclusão do Capítulo

Este capítulo abordou diversas técnicas e métodos para trazer melhorias no processo de visualização de terrenos, com uma grande quantidade de objetos espalhados por sua superfície. Algumas das contribuições foram feitas no campo da qualidade visual, enquanto outras foram feitas no campo do desempenho da aplicação. Ambos careciam de um estudo mais aprofundado sobre essas formas de representação, o que acabou consistindo no principal objetivo desse capítulo.

No campo da qualidade visual, abordamos os conceitos de imagem de satélite, a textura do terreno, texturas baseadas em imagens e modificação das vistas. Já no campo do desempenho, fizemos testes para verificar técnicas como *culling* dos objetos, multitexturas além de comparativos entre duas importantes formas de representação de objetos no terreno que se deram através do uso das estruturas de *sprites* e *billboards*. Ainda nesse capítulo, vimos como ficou o comportamento de três importantes estruturas de dados abordadas nesse trabalho como listas, *blobs* e *quadtree* com aglomeração de objetos em *billboards*, esta última, proposta deste trabalho.

Capítulo 5

Conclusões

Dedicamos este espaço para descrever, de forma sucinta, as principais contribuições que esse trabalho acabou por gerar. Além disso, discutiremos algumas das possíveis aplicações para a visualização de terrenos com grande quantidade de objetos.

No início da dissertação, fizemos uma revisão dos principais elementos que usualmente compõem visualizações de terrenos. Foi explicado como um terreno é comumente representado na forma digital e de que maneira podemos criar boas imagens para servirem como textura para a malha poligonal desse terreno. Apresentamos ainda, as diversas formas de representação de objetos conhecidas, e largamente utilizadas pela computação gráfica, fazendo inclusive, a comparação de desempenho entre as duas formas identificadas como as mais promissoras no que diz respeito aos objetivos desse trabalho.

Mostramos também como combinar importantes métodos e técnicas dependentes da visão (conhecidas ou não na literatura) para tirar o máximo proveito da visualização de terrenos com objetos.

Vimos ainda como a escolha correta da estrutura de dados pode impactar profundamente no processo de visualização. Para isso, propomos três estratégias diferentes de armazenamento de dados em memória principal e mostramos, por experimentação, qual dela se mostrou mais adequada para o problema abordado nesse trabalho.

Foram utilizadas também técnicas de programação em computação gráfica, tirando vantagens dos recursos disponíveis nas atuais placas de vídeo. A exemplo disso, podemos citar a utilização do conceito de *render to texture* e de FBO como ferramenta fundamental para geração das texturas utilizadas no processo de criação dos agrupamentos de billboards a partir de *quadtrees*.

São inúmeras as aplicações que podemos tirar quando vários objetos podem ser vistos

simultâneamente em uma visualização. Um exemplo simples de aplicação voltada para o caso de vegetação, poderia tratar a visualização como uma ferramenta para auxiliar o Governo no controle e combate ao desmatamento, prática que infelizmente ainda ocorre no Brasil com razoável freqüência. Assim, portando uma imagem de satélite atual da região na qual se deseja observar, poderíamos sobrevoar essa área digitalmente para fins de produção de relatórios e documentos. Outra aplicação das técnicas apresentadas, poderia ser perfeitamente agregadas em *engines* de jogos ou em simuladores de treinamentos militares.

5.1 Dificuldades

É fato que existe hoje na literatura uma razoável quantidade de trabalhos isolados que utilizam uma ou outra técnica dependente da visão. Entretanto, quando foi procurado por trabalhos que envolviam um conjunto de três ou mais técnicas juntas, esse número caiu drasticamente. Sendo assim, a primeira barreira que precisou ser vencida foi encontrar na literatura trabalhos com elevado grau de semelhança ao desenvolvido aqui.

Outra grande dificuldade enfrentada, refere-se ao desenvolvimento de uma estrutura hierárquica implementada através de uma *quadtree*. Essa complexa estrutura gerou uma série de problemas que não haviam sido previstos antecipadamente. A exemplo disso, podemos destacar o enorme volume de texturas geradas pela rotina de FBO o que tornou inviável mantê-las todas simultaneamente em memória principal. Esse problema foi contornado reutilizando-se alguns índices de texturas principalmente entre os objetos presentes nos mais altos níveis de profundidade da árvore (aqueles mais próximos aos nós-folhas).

Outra tarefa que demandou razoável tempo e grau de dificuldade foi a confecção da textura utilizada no terreno conforme mostrada na seção 4.1.1. Para se ter um nível aceitável de detalhamento, muitas imagens precisaram ser compostas lado-a-lado e isso foi feito de forma totalmente artesanal. Seria de grande ajuda se os atuais *softwares* de visualização de imagens de satélite permitissem de forma mais acessível e prática a obtenção de grandes regiões com alto nível de detalhamento. Pelo que foi observado na maioria dessas aplicações, essa tarefa ainda não é possível sem o longo e demorado procedimento descrito e utilizado nesse trabalho.

Outro fato que causou uma certa dificuldade de utilização foi devido a criação de uma grande quantidade de constantes definidas no início de vários módulos da aplicação. Essas

constantes foram criadas para controlar as mais variadas funcionalidades da visualização, tornando-as mais flexíveis. Entretanto, estipular os valores adequados para cada uma dessas constantes pode não ser uma tarefa fácil. O que se nota na prática é que, em geral, quanto mais familiaridade com a aplicação o usuário tiver, maior facilidade em calibrá-la ele será capaz.

5.2 Trabalhos futuros

Durante a elaboração desse trabalho, percebemos que ainda há muito por ser estudado no campo da visualização de terrenos com objetos. As complexidades envolvidas nas diversas formas de representação dos objetos abrem um leque muito extenso de possibilidades de estudo. No presente momento, muito desses conceitos já foram investigados, entretanto ainda há muitos outros que merecem igual atenção.

No intuito de destacar esses problemas que merecem atenção, dedicamos essa seção para discutir as possíveis continuidades que esse trabalho poderia ter. Foi levado principalmente em consideração os problemas e efeitos que surgiram em detrimento das implementações mostradas no capítulo 4.

A primeira observação remete à contribuição apresentada na seção 4.1.3 introduzida com o intuito de resolver o problema da visão de objetos em ângulos quase ortogonais ao terreno. Note que a solução de *shell textures* apresentada de fato resolve esse problema da ortogonalidade. Entretanto, em algumas ocasiões, é possível observar a transição entre as estruturas com vista frontal (seja ela de *billboard* ou *sprites*) e aquela com vista superior (utilizando *shell texture*). Isso causa um efeito indesejado já que essa mudança abrupta de estrutura compromete a qualidade visual da aplicação. Logo, uma razoável e possível correção consiste em suavizar essa troca através de alguma técnica já conhecida na literatura. Uma das opções poderia ser a técnica de *morphing* [64] que faria essa mudança de uma forma mais contínua e suave. Outra solução a ser investigada seria o uso de *fading*. Poderíamos fazer com que a estrutura utilizando *shell texture* fosse aparecendo aos poucos em um processo chamado *fade-in*. Quando tivéssemos seu brilho máximo ou seja, *fade-in máximo*, a estrutura com vista frontal poderia ser retirada.

Outro ponto observado se refere ao que foi abordado na seção 4.2.4.4. Note que a utilização das distâncias $\boldsymbol{d_1}$, $\boldsymbol{d_2}$ e $\boldsymbol{d_3}$ serviram, sobretudo, para estipular três tipos distintos de agrupamentos aceitáveis na visualização. Para o teste mostrado nessa mesma seção, essa estratégia se mostrou suficiente para contrastar o método da *quadtree* com os

demais abordados. Entretanto, uma extensão razoável seria generalizar esse processo de modo a permitir $\boldsymbol{n}$ tipos distintos de agrupamentos. Assim, uma vez que $\boldsymbol{n}$ seja um valor inteiro positivo bem maior do que três, é de se esperar que a seleção do agrupamento ocorra de forma mais acurada permitindo trocas suaves entre os tipos de agrupamentos, melhorando assim a visualização como um todo.

Outra questão interessante que merece ser estudada de forma mais aprofundada, é a questão da exibição das texturas com grande quantidades de objetos. A medida em que se permite agrupamentos maiores de objetos, foi notado o surgimento de alguns problemas. Um deles se refere ao próprio formato quadrado utilizados nas estrutura dos *billboards* e *sprites* proposto inicialmente. Pudemos notar que esta se torna ineficiente, visto que grandes quantidades de objetos tendem a se espalhar mais horizontalmente e pouco verticalmente. Assim, talvez seja mais interessante e viável substituir o formato atualmente utilizado por um outro, podendo ser, por exemplo, um retângulo. Ao trocar de formato, deve-se tomar o cuidado de observar a questão da simetria em cima do eixo $\boldsymbol{z}$ em especial quando se trata de visualização de *billboards* os quais necessitam de rotação.

Pelo que pudemos observar na prática, em alguns casos, ainda é possível encontrar uma grande concentração de objetos mesmo quando aplicado as técnicas de aglomerados introduzidas pelo método da *quadtree* visto anteriormente. Uma das formas de se contornar esse problema e diminuir essa densidade de concentração de objetos é aumentando o número máximo de objetos permitidos por aglomerado, atribuindo quinze, vinte, trinta ou mais elementos para cada aglomerado. Entretanto, devido ao problema reportado no parágrafo anterior, entendemos que talvez seja necessária a investigação de outros métodos como o de se criar mapas de oclusão para essas estruturas tomando a idéia de *occlusion culling* proposta em [65].

Acreditamos que poderíamos ter inúmeras aplicações caso a idéia aplicada no caso de vegetação fosse expandida para outros objetos como veículos, edificações, estradas, dentre outros. Entendemos que, desde que seja possível gerar e armazenar as assinaturas proveniente da aplicação das técnicas de *wavelets*, poderíamos ser capazes de extrair praticamente qualquer elemento da imagem de satélite. Logo, a principal dificuldade se resumiria em obter o melhor modelo tridimensional para que sejam geradas as principais vistas do novo objeto a ser considerado.

Com a intenção de melhorar ainda mais a qualidade visual dos objetos, poderíamos também investigar o uso de *relief texture mapping* [66]. Através da implementação dessa técnica, poderíamos introduzir uma noção de profundidade entre os elementos gráficos

mostrados na cena, efeito não muito bem representado quando utilizado texturas simples.

Antes de finalizar a seção de trabalhos futuros, ressaltamos uma última observação: durante a elaboração desse trabalho, por razões de limitação física de memória principal, tivemos que reutilizar índices de textura para os agrupamentos próximos às folhas da *quadtree*. Porém, acreditamos que em um futuro não distante, esse problema pode ser amenizado ou mesmo resolvido devido aos avanços tecnológicos e sobretudo da eletrônica e da informática.

Por fim, tendo citado os principais problemas que surgiram em decorrência da implementação das técnicas nesse trabalho, entendemos claramente que, apesar de eficientes e funcionais, elas podem ser perfeitamente aperfeiçoadas e estendidas, agregando ainda mais funcionalidades e melhoramentos para todo o processo de visualização de terrenos com objetos.

Referências

[1] SOUZA, E. M. de. *Um Estudo sobre um Algoritmo para Visualização de Terrenos.* Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 2003.

[2] MACIEL, P. W. C.; SHIRLEY, P. Visual navigation of large environments using textured clusters. In: *In 1995 Symposium on Interactive 3D Graphics.* New York, EUA: ACM, 1995. p. 95–102.

[3] ASSARSSON, U.; MOLLER, T. *Optimized View Frustum Culling Algorithms.* Gothenburg, Sweden, 1999. Techinical Report.

[4] ZACH, C. Integration of geomorphing into level of detail management for realtime rendering. In: *SCCG '02: Proceedings of the 18th Spring Conference on Computer Graphics.* New York, NY, USA: ACM, 2002. p. 115–122. ISBN 1-58113-608-0.

[5] LLUCH, J.; CAMAHORT, E.; VIVÓ, R. An image-based multiresolution model for interactive foliage rendering. In: *Winter School of Computer Graphics.* Valencia, Spain: ACM WSCG, 2004. p. 507–514.

[6] BEHRENDT, S. et al. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum*, v. 24, n. 3, p. 507–516, 2005.

[7] SAVELLI, R. M.; SEIXAS, R. B. Semi-automatic detection of vegetations in digital satellite images for building 3d terrains. *The Sixth Iasted Internacional Conference On Visualization Imaging And Image Processing*, p. 494–498, 2006.

[8] MONTENEGRO, A. A. *Investigação de Novos Critérios para Inserção de Pontos em Métodos para Simulação de Modelos de Terreno Através de Refinamento.* Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 1997.

[9] LINDSTROM, P.; PASCUCCI, V. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *Proceedings of IEEE Visualization 2001*, IEEE, San Diego, California, p. 363–370, 2002.

[10] TAN, P. et al. Image-based tree modeling. In: *SIGGRAPH '07: ACM SIGGRAPH 2007 papers.* New York, NY, USA: ACM, 2007. p. 87.

[11] CONCI, A. *Capítulo 1 do curso de Fractais.* Disponível em: `http://www.ic.uff.br/~aconci/curso/Cap1.pdf`.

[12] CONCI, A. *Capítulo 2 do curso de Fractais.* Disponível em: `http://www.ic.uff.br/~aconci/curso/Cap2.pdf`.

[13] CONCI, A. *Capítulo 3 do curso de Fractais.* Disponível em: `http://www.ic.uff.br/~aconci/curso/Cap3.pdf`.

[14] CONCI, A.; AZEVEDO, E.; LETA, F. R. *Computação Gráfica: Teoria e Prática, volume 2*. Brasil: Elsevier, 2007.

[15] MELO, R. H. C. de. *Using Fractal Characteristics such as Fractal Dimension, Lacunarity and Succolarity to Characterize Texture patterns on Images*. Dissertação (Mestrado) — Universidade Federal Fluminense, 2007. Disponível em: <http://www.ic.uff.br/PosGraduacao/lista_dissertacao.php?ano=2007>.

[16] SMITH, A. R. Plants, Fractals, and Formal Languages. v. 18, n. 3, p. 1–10, jul. 1984. Disponível em: <http://alvyray.com/Papers/PapersCG.htm#Graftals>.

[17] SHADE, J. et al. Hierarchical image caching for accelerated walkthroughs of complex environments. In: *SIGGRAPH 96*. Los Angeles, USA: ACM SIGGRAPH, 1996. p. 75–82.

[18] DEUSSEN, O. et al. Interactive visualization of complex plant ecosystems. In: *Proceedings of the IEEE Visualization Conference*. IEEE, 2002. Disponível em: <http://www-sop.inria.fr/reves/Basilic/2002/DCSD02>.

[19] CHOMSKY, N. Three models for the description of language. In: *IRE Transations on Information Theory*. USA: IEEE Press, 1956. p. 113–124.

[20] PRUSINKIEWICZ, P.; LINDENMAYER, A. *The Algorithmic Beauty of Plants (The Virtual Laboratory)*. First. EUA: Springer, 1991.

[21] MELO, A.; CLUA, E.; MONTENEGRO, A. *Real-time procedural generation of 3D trees in GPU using Geometry Shaders*. Rio de Janeiro, Brasil, 2008.

[22] JAKULIN, A. *EUROGRAPHICS 2000 / A. de Sousa, J.C. Torres Short Presentations Interactive Vegetation Rendering with Slicing and Blending*. 2000. 120-129 p.

[23] DéCORET, X. et al. Billboard clouds for extreme model simplification. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 22, n. 3, p. 689–696, 2003. ISSN 0730-0301.

[24] COLDITZ, C. et al. Realtime rendering of complex photorealistic landscapes using hybrid level-of-detail approaches. In: *Trends in Real-Time Landscape Visualization and Participation*. Berlin-Dahlem, Germany: IEEE, 2005. p. 97–106.

[25] OPENGL. *OpenGL The Industry's Foundation for High Performance Graphics*. Disponível em: `http://www.opengl.org/`. 30/06/2007.

[26] MCREYNOLDS, T.; BLYTHE, D. *Advanced Graphics Programming Using OpenGL*. Second. EUA: Elsevier Science & Technology Books, 2005.

[27] KERNIGHAN, B. W.; RITCHIE, D. M. *C Programming Language*. Second. EUA: Prentice Hall Software, 1988.

[28] DISCOE, B. *Virtual Terrain Project*. Disponível em: `http://www.vterrain.org`. 20/08/2008.

[29] DISCOE, B. *Histórico do Virtual Terrain Project*. Disponível em: `http://www.vterrain.org/Site/changes_old.html`. 20/08/2008.

[30] STROUSTRUP, B. *C++ Programming Language*. Third. EUA: Addison-Wesley Professional, 1997.

[31] GDAL. *Geospatial Data Abstraction Library*. Disponível em: `http://www.gdal.org/`. 20/08/2008.

[32] PROJ. *Cartographic Projections Library*. Disponível em: `http://trac.osgeo.org/proj/`. 20/08/2008.

[33] OSG. *Open Scene Graph Library*. Disponível em: `http://www.openscenegraph.org/projects/osg`. 20/08/2008.

[34] WXWIDGETS. *wxWidgets Library*. Disponível em: `http://www.wxwidgets.org/`. 20/08/2008.

[35] MINI. *Mini Library*. Disponível em: `http://www.stereofx.org/terrain.html`. 20/08/2008.

[36] LIBZIP. *Libzip Library - A C library for reading, creating, and modifying zip archives*. Disponível em: `http://www.nih.at/libzip/`. 20/08/2008.

[37] DISCOE, B. *Manual de navegação da interface do Enviro*. Disponível em: `http://www.vterrain.org/Navigation/simple.html`. 20/08/2008.

[38] GATTASS, M. *Laboratório de Computação Gráfica*. Disponível em: `http://www.tecgraf.puc-rio.br/`. TeCGraf/PUC-Rio.

[39] BRASIL, M. do. *Centro de Jogos Didáticos*. Disponível em: `http://www.mar.mil.br/`. 19/08/2008.

[40] DEITEL, H.; DEITEL, P. *C++ How to Program*. Sixth. EUA: Prentice Hall, 2007.

[41] FIGUEIREDO, L. H.; CELES, W.; IERUSALIMSCHY, R. *Lua Programming Language*. Disponível em: `http://www.lua.org/`. 19/08/2008.

[42] SCURI, A. *Portable User Interface – IUP*. Disponível em: `http://luaforge.net/frs/?group_id=89`. 19/08/2008.

[43] SCURI, A. *Image Manipulation – IM*. Disponível em: `http://luaforge.net/frs/?group_id=86`. 19/08/2008.

[44] SCURI, A. *Canvas Draw – CD*. Disponível em: `http://luaforge.net/frs/?group_id=88`. 19/08/2008.

[45] ABRAHAM, F. *VIS*. 19/08/2008.

[46] GLEW. *The OpenGL Extension Wrangler Library*. Disponível em: `http://glew.sourceforge.net/`. 30/06/2007.

[47] EMBRAPA. *O Brasil visto do espaço*. Disponível em: `http://www.cdbrasil.cnpm.embrapa.br/`. 17/07/2008.

[48] GOOGLE. *Google Earth*. Disponível em: `http://earth.google.com.br/`. 17/07/2008.

[49] MICROSOFT. *Live Search Maps*. Disponível em: `http://maps.live.com/`. 17/07/2008.

[50] BOW, S. T. *Pattern Recognition and Image Preprocessing*. Second. EUA: CRC, 2002.

[51] PITAS, I. *Digital Image Processing: Algorithms and Applications*. Primeira. EUA: Wiley-Interscience, 2000.

[52] VIDAKOVIC, B.; MUELLER, P. *Wavelets for Kids*. USA, 1991.

[53] FONSECA, M. S. *Um Estudo sobre a Influência das Famílias Wavelets na Compressão de Imagem*. Dissertação (Mestrado) — Universidade Federal Fluminense, 2004.

[54] BEYLKIN, G.; COIFMAN, R.; ROKHLIN, V. Fast wavelet transforms and numerical algorithms i. *Communication on Pure and Applied Mathematics*, ACM, New Haven, Connecticut, p. 141–183, 1991.

[55] PYTHON. *Python Programming Language*. Disponível em: `http://www.python.org/`. 17/07/2008.

[56] EMBRAPA. *Ministério da Agricultura, Pecuária e Abastecimento*. Disponível em: `http://www.embrapa.br/`. 17/07/2008.

[57] GOOGLE. *Google Inc*. Disponível em: `http://earth.google.com.br/`. 17/07/2008.

[58] ADOBE. *Adobe Systems Inc*. Disponível em: `http://www.adobe.com/`. 17/07/2008.

[59] SNYDER, J. P. *Map Projections - A Working Manual*. First. Los Angeles, USA: US Government Printing Office, 1987.

[60] ROBINS, N. *Tutorials - Transformations*. Disponível em: `http://www.xmission.com/~nate/tutors.htm`. 17/07/2008.

[61] JONES, R. *OpenGL Frame Buffer Object*. Disponível em: `http://www.gamedev.net/reference/articles/article2331.asp`. 17/07/2008.

[62] CORMEN, T. H. et al. *Introduction to Algorithms*. Second. EUA: The MIT Press, 2001.

[63] GERNOT, S. Image-based object representation by layered impostors. In: *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*. New York, NY, USA: ACM, 1998. p. 99–104. ISBN 1-58113-019-8.

[64] GOMES, J. et al. *Warping & Morphing of Graphical Objects*. First. EUA: Morgan Kaufmann, 1999.

[65] MICIKEVICIUS, P.; HUGHES, C. E.; MOSHELL, J. M. *Interactive Forest Walkthrough*. 2004.

[66] OLIVEIRA, M. M.; BISHOP, G.; MCALLISTER, D. Relief texture mapping. In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. p. 359–368. ISBN 1-58113-208-5.