4
Testes

Neste capítulo apresentamos uma aplicação que permite realizar vôos sobre terrenos utilizando o algoritmo discutido ao longo deste trabalho. Em seguida, adicionamos e comentamos alguns testes realizados utilizando esta aplicação.

4.1.
Breve Descrição da Aplicação

O programa SJD-Vis3D[11] simula um vôo sobre um terreno e está sendo desenvolvido pelo Tecgraf para a Marinha do Brasil. O algoritmo discutido neste trabalho é utilizado para obter, a cada quadro, a malha do terreno sendo sobrevoado. Vale destacar que o programa SJD-Vis3D é implementado em linguagem C++, utiliza a biblioteca gráfica OpenGL[12] e arquivos de descrição feitos na linguagem de *scripts* Lua[13]. Para maiores detalhes sobre a descrição do programa SJD-Vis3D, veja o apêndice A.

4.2.
Descrição dos Testes com a Aplicação SJD-Vis3D

Todos os nossos testes foram realizados nas seguintes plataformas:

1) Computador Intel® Pentium® 4 CPU 1700 MHZ AT/AT Compatible com 512 MB de RAM, equipado com uma placa gráfica NVIDIA GeForce4 Ti 4200.

2) Computador Intel® Pentium® 4 CPU 2.53 GHz com 3,00 GB de RAM, com uma placa gráfica NVIDIA Quadro FX1000.

Durante os testes, a plataforma 1 executava o sistema operacional Microsoft Windows 2000 e a plataforma 2 o Microsoft Windows XP. Além disso, em todos os testes o tamanho da janela utilizada foi sempre de 800x600 *pixels*.

A Tabela 2 apresenta os principais dados dos quatro terrenos utilizados em nossos testes. Os dois primeiros pertencem à Marinha do Brasil e representam a região de Itaóca, localizada no litoral do estado brasileiro do Espírito Santo. Os

outros dois foram obtidos a partir de imagens disponibilizadas em [14] e representam uma região de Washington, nos Estados Unidos.

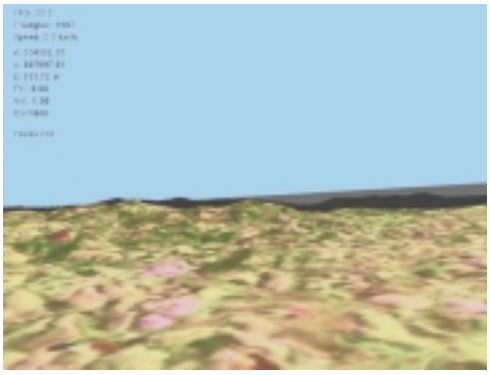
Dados dos terrenos	Dimensões do terreno (km^2)	Número de amostras	Dimensões de uma célula de amostra (m^3)	Dimensões da matriz utilizada
Itaoca1	26,2 x 34,8	525 colunas e 697 linhas = 365.925	50x50x1	1025 x 1025
Itaoca2	26,2 x 34,8	1312 colunas e 1741 linhas = 2.284.192 amostras	20x20x1	2049x2049
Washington1	163,84x163,84	1025 colunas e 1025 linhas = 1.050.625	160x160x0,1	1025 x 1025
Washington2	163,84x163,84	4097 colunas e 4097 linhas = 16.785.409	40x40x0,1	4097x4097

Tabela 2 - Dados dos Terrenos.

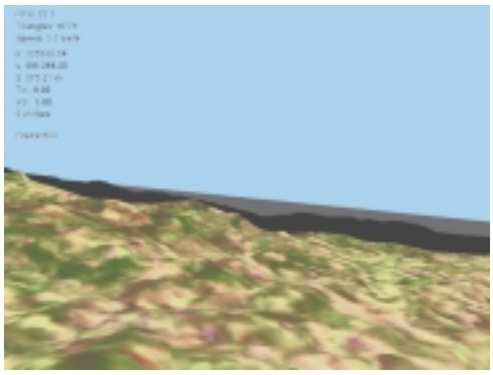
Em algoritmos dependentes da visão, como o tratado neste trabalho, o número de triângulos gerados depende sobretudo da distância da câmera, do erro tolerável especificado e se a região vista é mais ou menos plana. Assim, por exemplo, regiões vistas de longe e mais planas geram menos triângulos.

Todavia, com o intuito de analisar o algoritmo em condições variadas, construímos quatro caminhos (um sobre cada um dos terrenos mostrados na Tabela 2) e depois realizamos algumas medidas. As figuras 19, 20, 21 e 22 apresentam graficamente cada um dos caminhos de teste. Cada caminho é composto por 1500 quadros e leva o mesmo nome do terreno sobre o qual foi executado. Em geral, procuramos incluir nos caminhos as regiões mais montanhosas de cada terreno. Além disso, repare que todos iniciam com a câmera posicionada no centro do respectivo terreno. (Observação: as informações escritas sobre as imagens (no canto superior esquerdo) em (b), (c), (d) e (e) de cada figura podem ser desprezadas neste ponto).

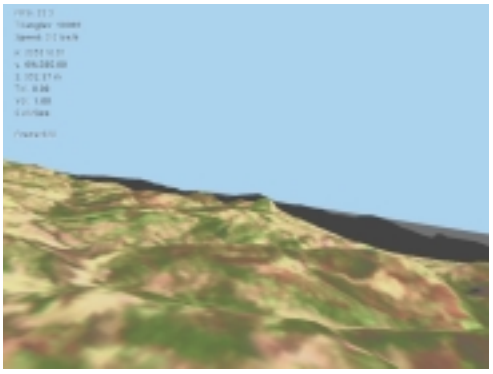
a) Caminho sobre o terreno



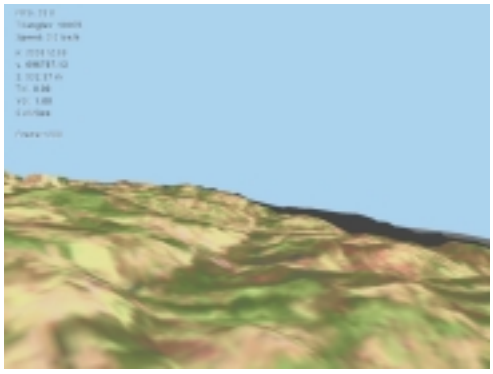
b) Vista em (***a***), quadro 200.



c) Vista em (***b***), quadro 600.

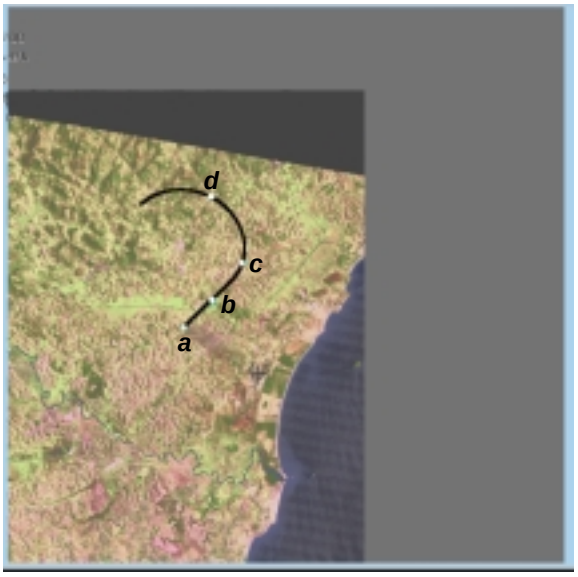


d) Vista em (***c***), quadro 920.

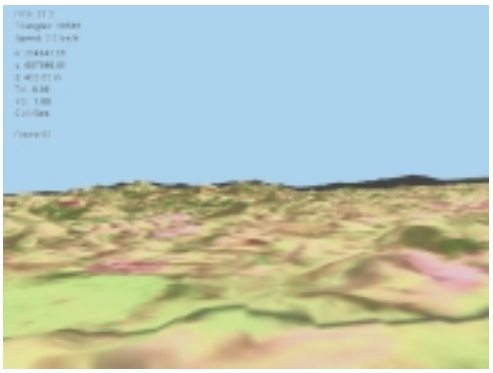


e) Vista em (***d***), quadro 1200.

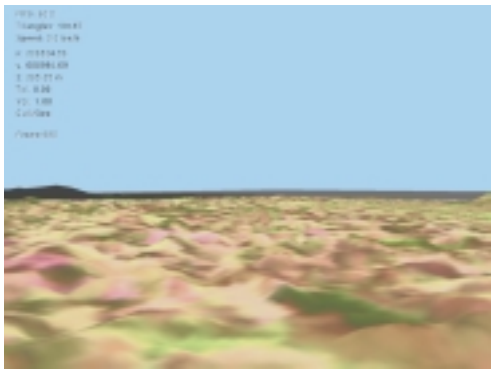
Figura 19 - Caminho sobre o terreno Itaoca1



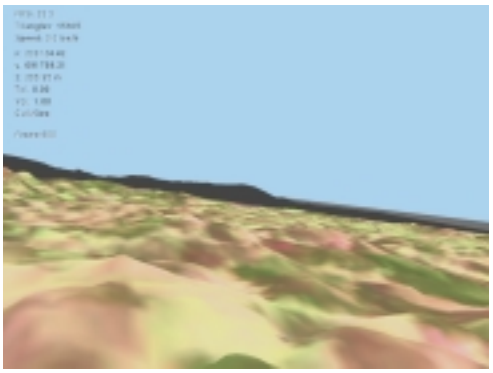
a) Caminho sobre o terreno



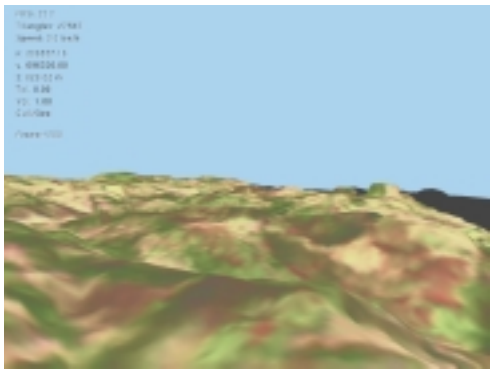
b) Vista em (***a***), quadro 90.



c) Vista em (***b***), quadro 580.

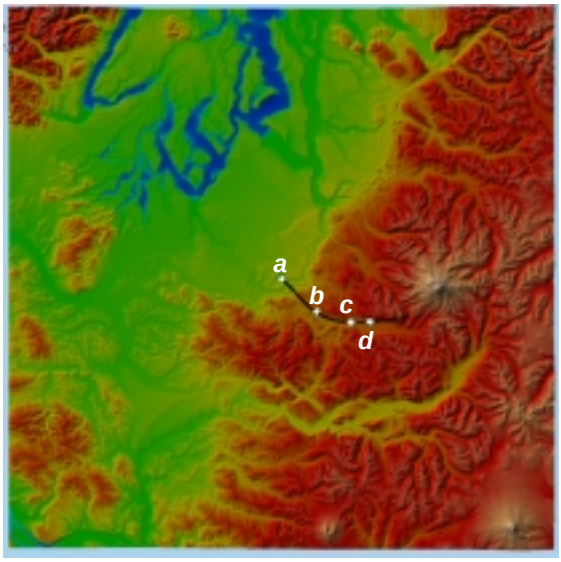


d) Vista em (***c***), quadro 900.

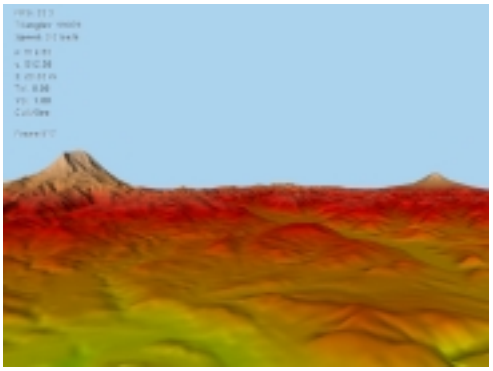


e) Vista em (***d***), quadro 1200.

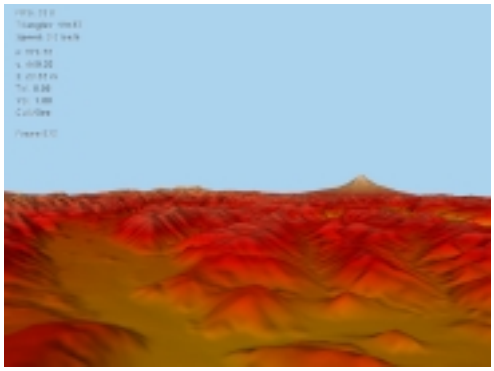
Figura 20 - Caminho sobre o terreno Itaoca2.



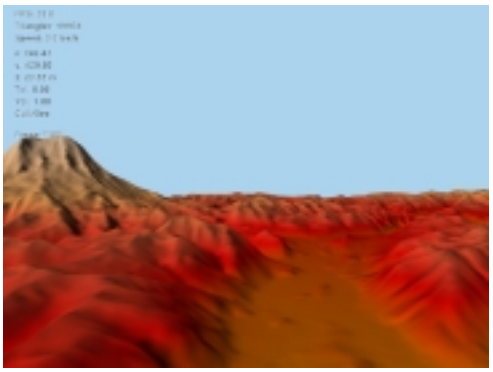
a) Caminho sobre o terreno



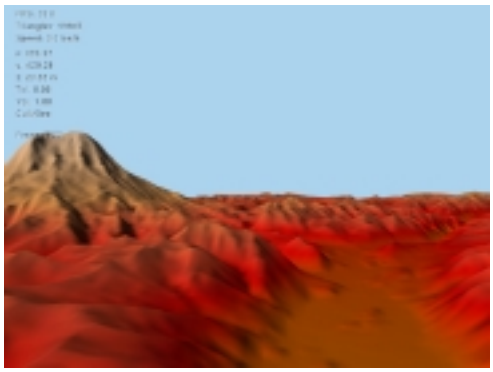
b) Vista em (***a***), quadro 512.



c) Vista em (***b***), quadro 970.

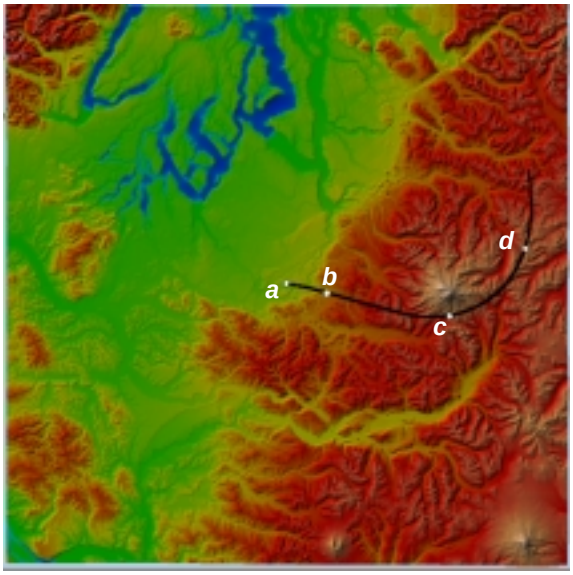


d) Vista em (***c***), quadro 1160.

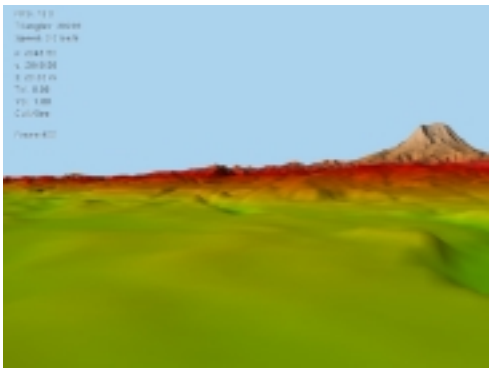


e) Vista em (***d***), quadro 1200.

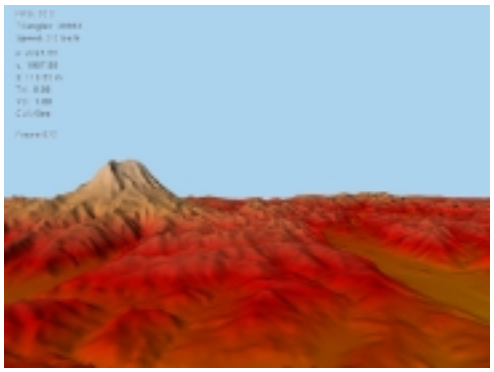
Figura 21 - Caminho sobre o terreno Washington1



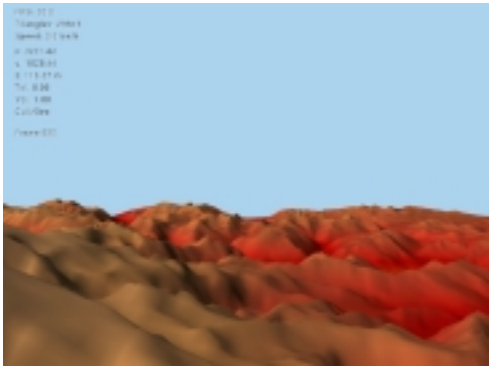
a) Caminho sobre o terreno



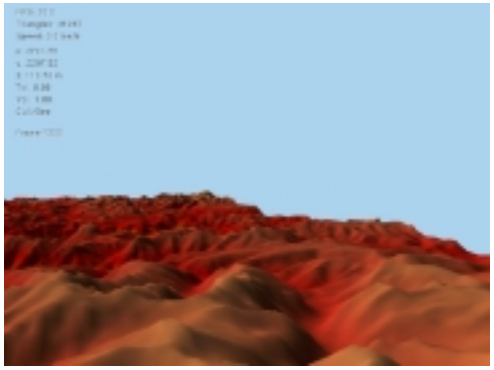
b) Vista em (***a***), quadro 400.



c) Vista em (***b***), quadro 670.



d) Vista em (***c***), quadro 980.



e) Vista em (***d***), quadro 1300.

Figura 22 - Caminho sobre o terreno Washington2

4.2.1.
Verificando o Culling da Malha fora da Visão

Para analisar a operação de *culling* da malha fora da visão, executamos cada um dos caminhos de teste com o *culling* e depois repetimos o mesmo caminho sem o *culling*. Realizamos este procedimento nas duas plataformas usadas para teste e então, com os resultados coletados, construímos os gráficos mostrados na Figura 23 e nas tabelas 3 e 4. As tabelas apresentam o valor médio em FPS (*frames per second*) obtido ao longo de cada caminho nas plataformas 1 e 2, respectivamente. Já os gráficos realçam o contraste entre a quantidade de triângulos desenhados quando a operação de *culling* está sendo utilizada e quando não está. Repare que, em todos os caminhos e em todos os momentos, esta operação sempre permite descartar um número significativo de triângulos e aumentar o desempenho da aplicação.

FPS Médio	Itaoca1	Itaoca2	Washington1	Washington2
Com *Culling*	48,6	31,6	33,0	20,7
Sem *Culling*	17,7	9,2	10,4	4,8

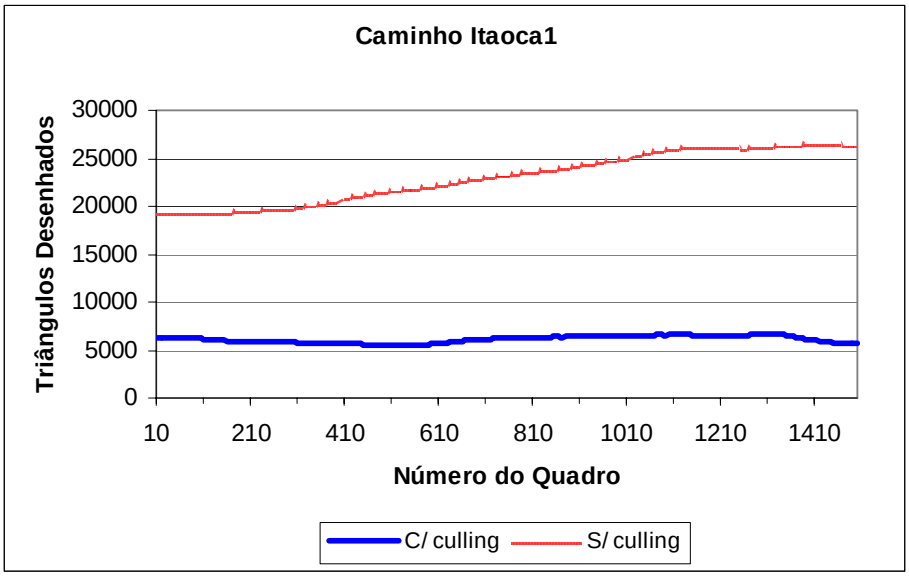
Tabela 3 - FPS Médio de cada caminho feito com e sem *culling* na plataforma 1.

FPS Médio	Itaoca1	Itaoca2	Washington1	Washington2
Com *Culling*	79,0	50,9	52,0	35,3
Sem *Culling*	31,4	15,9	17,1	8,3

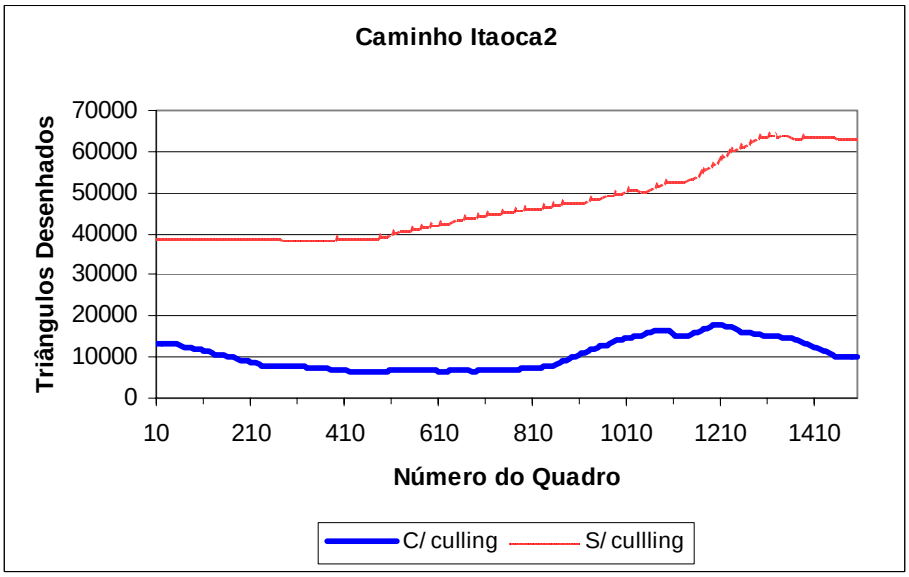
Tabela 4 - FPS Médio de cada caminho feito com e sem *culling* na plataforma 2.

Podemos observar, através destes gráficos, que, quanto maior o terreno, mais importante se torna esta operação. Tomemos como primeiro exemplo o caminho Itaoca1 executado na plataforma 2. Neste caso, como o terreno é pequeno e a plataforma é relativamente potente, então mesmo sem o *culling* de visão ainda conseguimos obter uma taxa de 31,4 quadros por segundo.

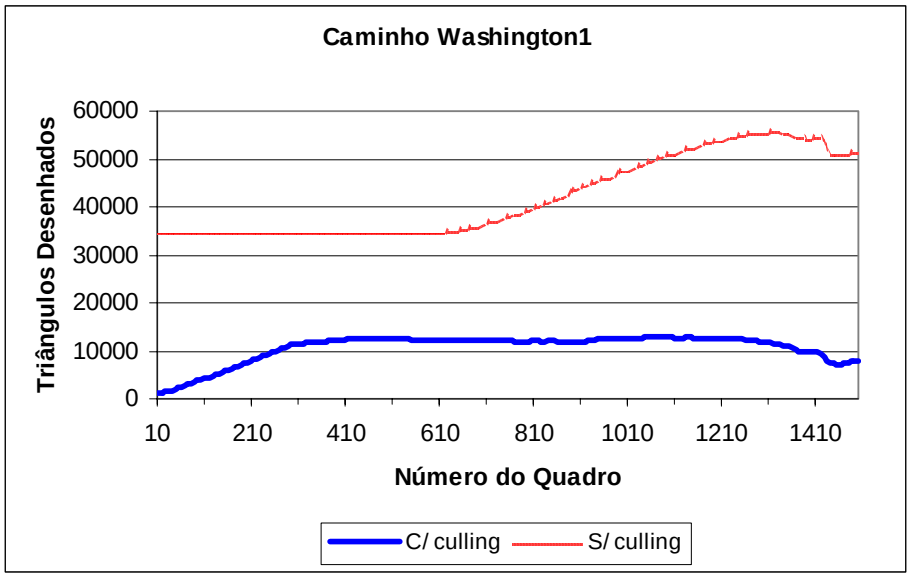
Agora tomemos como segundo exemplo o caminho Washington2 também executado na plataforma 2. Repare que obtemos uma boa taxa média de 35,3 quadros por segundo quando utilizando o *culling* da malha fora da visão. Eliminando esta operação a taxa média cai para 8,3 quadros por segundo. Neste exemplo, fica evidenciado que a operação de *culling* da malha fora da visão tornou-se essencial para assegurar taxas interativas (normalmente acima de 20 FPS) ao longo do caminho.



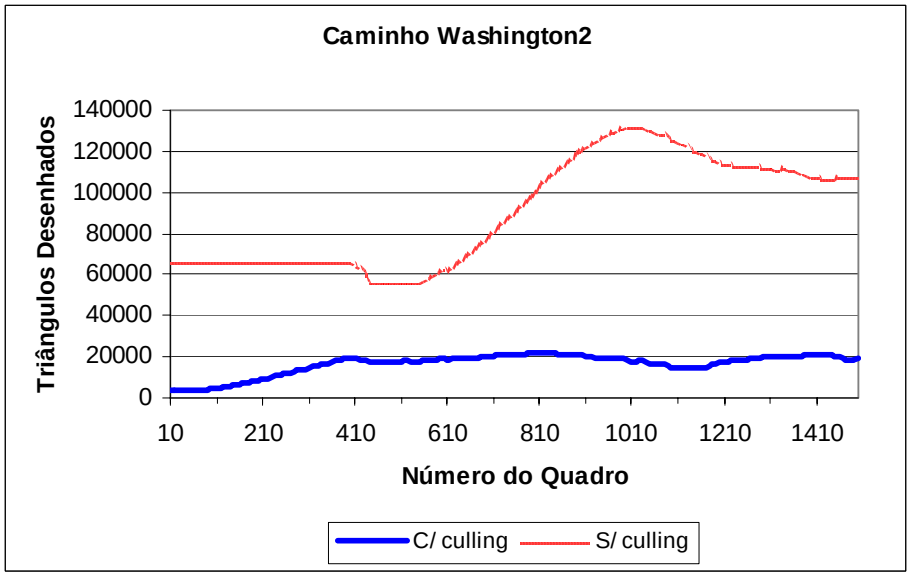
a)



b)



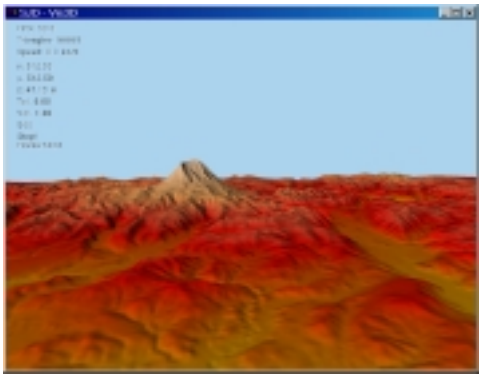
c)



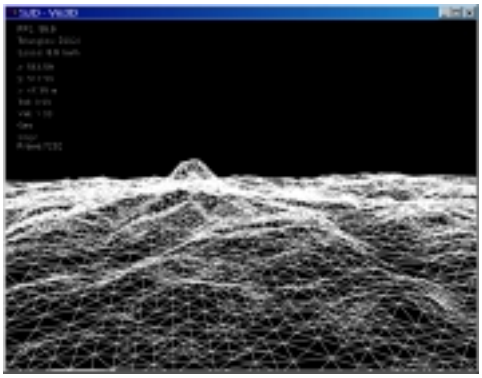
d)

Figura 23 - Comparação do número de triângulos gerados com e sem a operação de *culling* fora da visão.

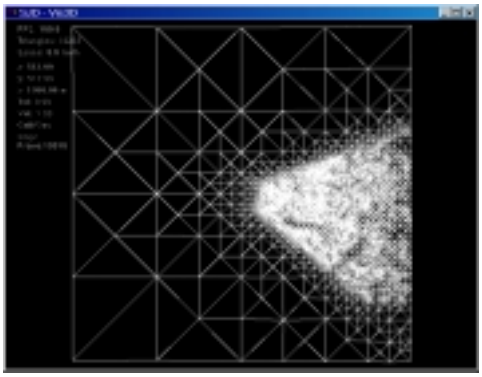
Além disso, através da seqüência de imagens geradas a partir de um quadro do terreno Washington1 e apresentadas na Figura 24 é possível verificar visualmente a diferença entre uma malha produzida quando a operação de *culling* da malha fora da visão está sendo utilizada e quando não está.



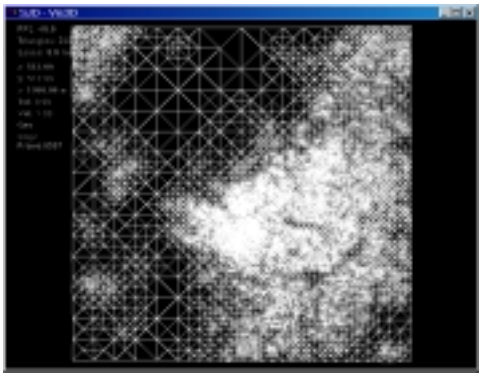
a) Quadro com textura.



b) Quadro em *wireframe* e sem textura.



c) Malha do quadro em *a,* com *culling.*



d) Malha do quadro em ***a*****,** sem *culling*.

Figura 24 - Quadro do terreno Washington1

4.2.2. Verificando o *Geomorphing*

Analogamente ao método usado para a operação anterior, para analisar a operação de *geomorphing* executamos o algoritmo com o *geomorphing* e, depois sem o *geomorphing*, mas sem modificar o erro tolerável e, finalmente, sem o *geomorphing*, porém usando um erro tolerável bem pequeno. A idéia aqui é que para se obter um resultado visual equivalente ao *geomorphing*, o erro tolerável deve ser pequeno. O propósito do *geomorphing* é permitir especificar um erro tolerável relativamente grande, buscando reduzir o número de triângulos gerados, portanto aumentando o desempenho da aplicação e, ao mesmo tempo, sem produzir artefatos visuais desagradáveis (como montanhas aparecendo e desaparecendo bruscamente) à medida que a câmera se movimenta.

Além disso, conforme já explicado no capítulo anterior, para utilizar o *geomorphing* devemos especificar uma faixa de erro tolerável (diferença entre as tolerâncias mínima e máxima especificadas). Quanto maior esta faixa mais suave

serão as transições durante o vôo, porém mais triângulos serão gerados. Neste caso, estamos utilizando uma tolerância máxima igual 3/2 * tolerância mínima. Esta fórmula foi sugerida por Lindstrom & Pascucci[3] e confirmada em nossos testes como fornecedora de bons resultados.

Desta forma, eliminando o *geomorphing*, temos duas opções. A primeira é continuar usando um erro tolerável grande, ou seja, igual à tolerância máxima do caso anterior. Neste caso, conforme pode ser confirmado pelas tabelas 5 e 6 e nos gráficos da Figura 25, o número de triângulos gerados diminui e a taxa de quadros por segundo aumenta. O inconveniente é que também perdemos em qualidade de apresentação, pois grandes montanhas começam a aparecer e desaparecer instantaneamente quando a câmera se move.

Como exemplo, vamos considerar o caminho Itaoca2 realizado na plataforma 1. Neste caso, utilizando o *geomorphing* alcançamos uma taxa média de 32,0 quadros por segundo. Retirando o *geomorphing* a média obtida sobe para 45,6 quadros por segundo. Contudo, claramente, as montanhas do terreno aparecem e desaparecem bruscamente quando a câmera se movimenta.

Por outro lado, usando a segunda opção, ou seja, um erro tolerável pequeno o bastante para que estes artefatos visuais não sejam percebidos, a taxa de quadros ou o FPS médio cai para apenas 10,4.

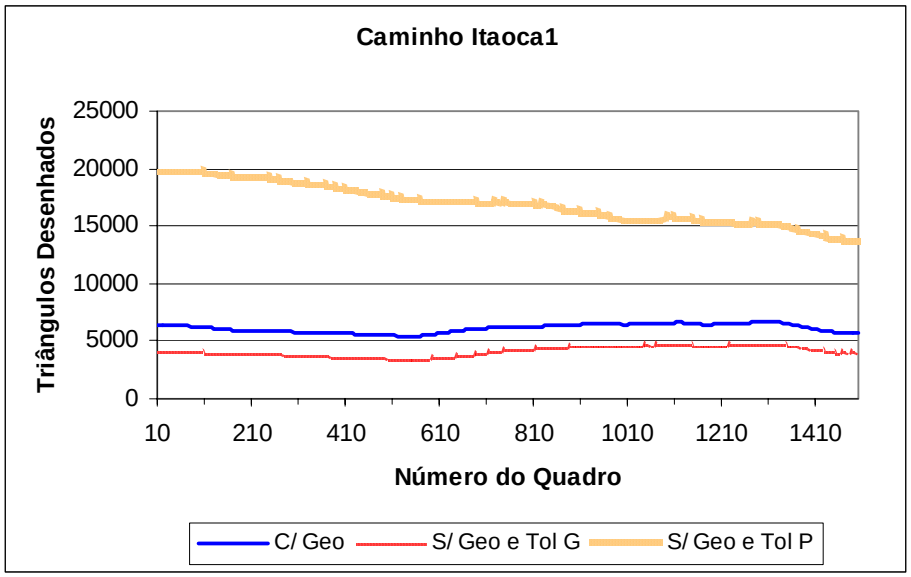
Repare que, analogamente ao *culling* da malha fora da visão, a operação de *geomorphing* também torna-se mais importante à medida que aumentamos o tamanho do terreno.

FPS Médio	Itaoca1	Itaoca2	Washington1	Washington2
Com Geo	48,4	32,0	33,1	21,7
Sem Geo e Tol grande	61,5	45,6	48,2	28,5
Sem Geo e Tol pequena	21,7	10,4	10,9	5,6

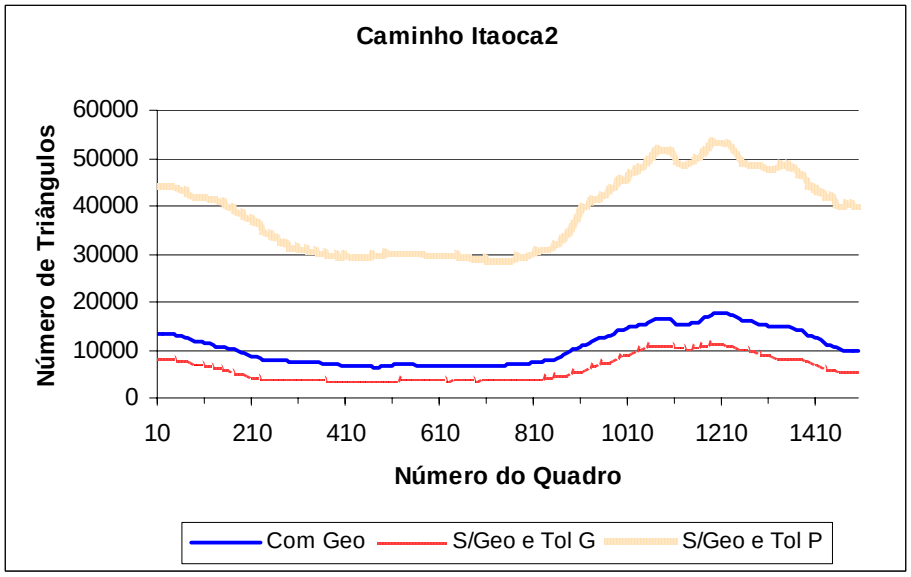
Tabela 5 - FPS Médio de cada caminho feito com e sem *geomorphing* na plataforma 1.

FPS Médio	Itaoca1	Itaoca2	Washington1	Washington2
Com Geo	79,0	50,9	52,0	35,3
Sem Geo e Tol grande	85,0	66,3	69,6	51,4
Sem Geo e Tol pequena	34,2	16,4	17,2	9,2

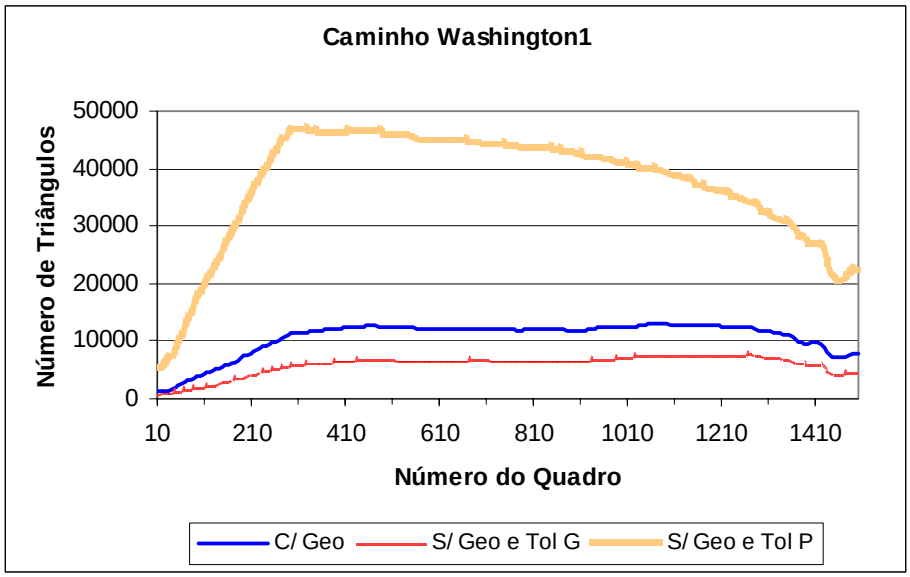
Tabela 6 - FPS Médio de cada caminho feito com e sem *geomorphing* na plataforma 2.



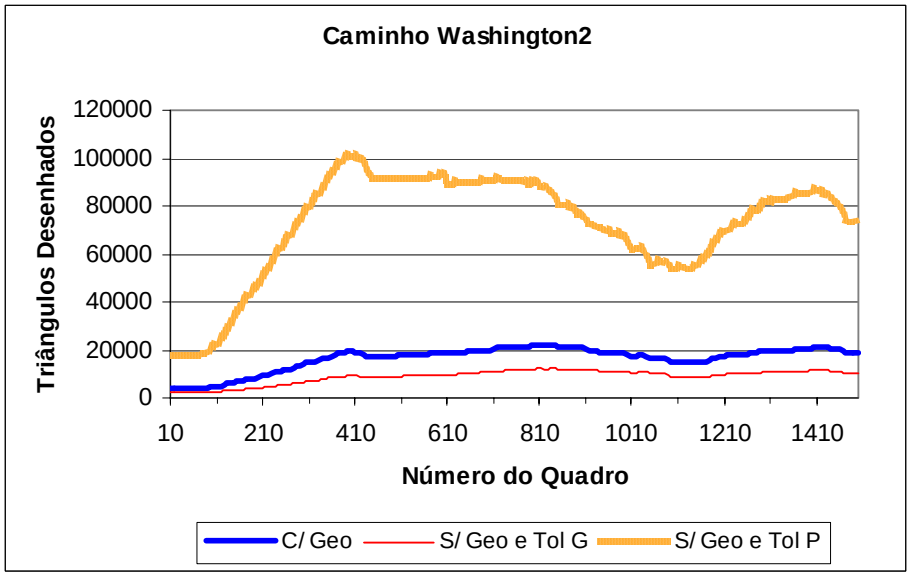
a)



b)



c)



d)

Figura 25 - Gráficos comparativos entre o número de triângulos desenhados.

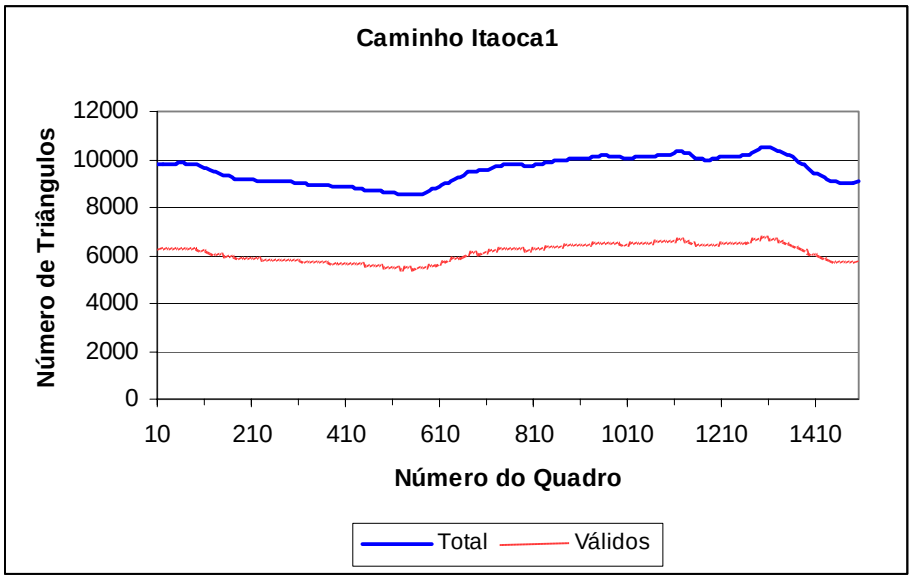
1) (C/Geo): a operação de *geomorphing* é usada. 2) (S/ Geo e Tol G): o *geomorphing* não é usado e o erro tolerável é tão grande quando no primeiro caso. 3) (S/Geo e Tol P): o *geomorphing* não é usado, mas o erro tolerável é pequeno a ponto de oferecer uma sensação visual similar ao caso em que o *geomorphing* está sendo usado.

4.2.3.
Verificando a Strip de Triângulos

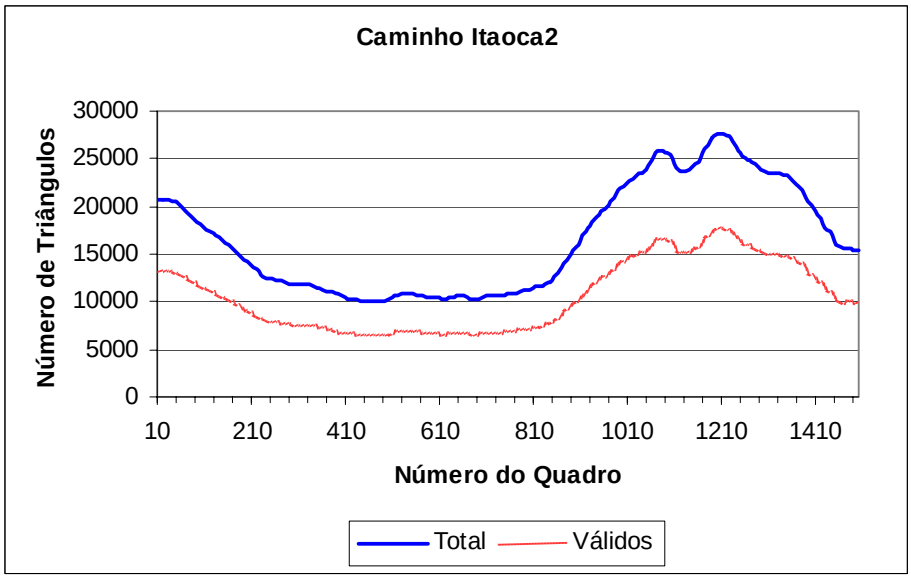
Para analisar a eficiência da *strip* ou lista de vértices que é construída pelo algoritmo de refinamento, vamos considerar todos os nossos quatro caminhos de teste. Em cada intervalo de dez quadros ao longo de cada caminho, coletamos o número total de triângulos enviados para o *pipeline* gráfico e o número de triângulos válidos, ou seja, triângulos com área não nula de um quadro de amostra. Apresentamos os resultados obtidos nos gráficos da figura 26.

Além disso, considerando os caminhos Itaoca1 e Itaoca2 inteiros, obtivemos que o número total de triângulos ao longo do caminho dividido pelo número de triângulos válidos resultou em 1,56. Em outras palavras, neste caso, para desenhar cada triângulo da malha gerada, estamos enviando uma média de 1,56 vértices para o *pipeline* gráfico. No caso dos caminhos Washington1 e Washington2, o número total de triângulos dividido pelo número de triângulos válidos resultou em 1,57, ou seja, estamos enviando em média 1,57 vértices para desenhar cada triângulo da malha. Repare pelos gráficos que esta razão permanece constante ao longo de todos os caminhos.

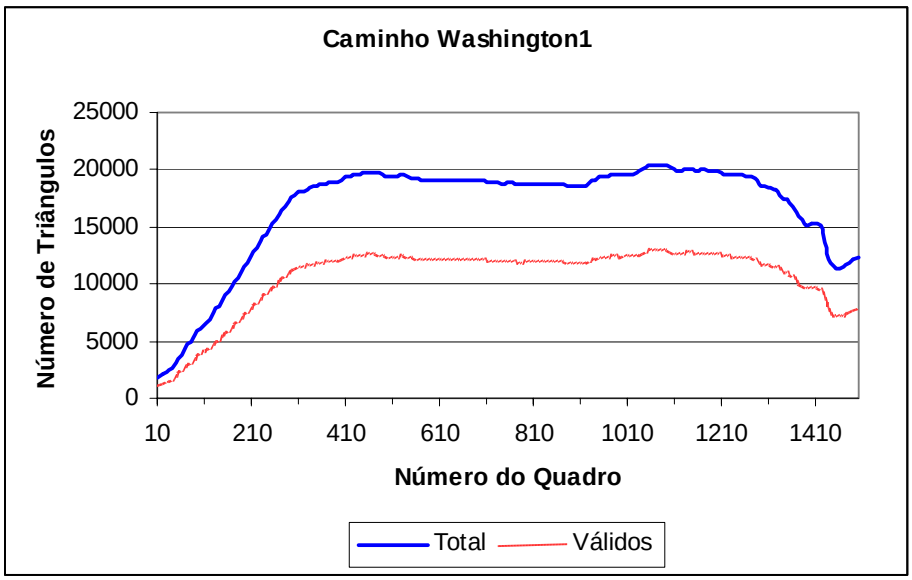
Portanto, podemos considerar esta *strip* ou lista de vértices eficiente, pois sem ela teríamos que informar sempre 3 vértices para cada triângulo a ser desenhado.



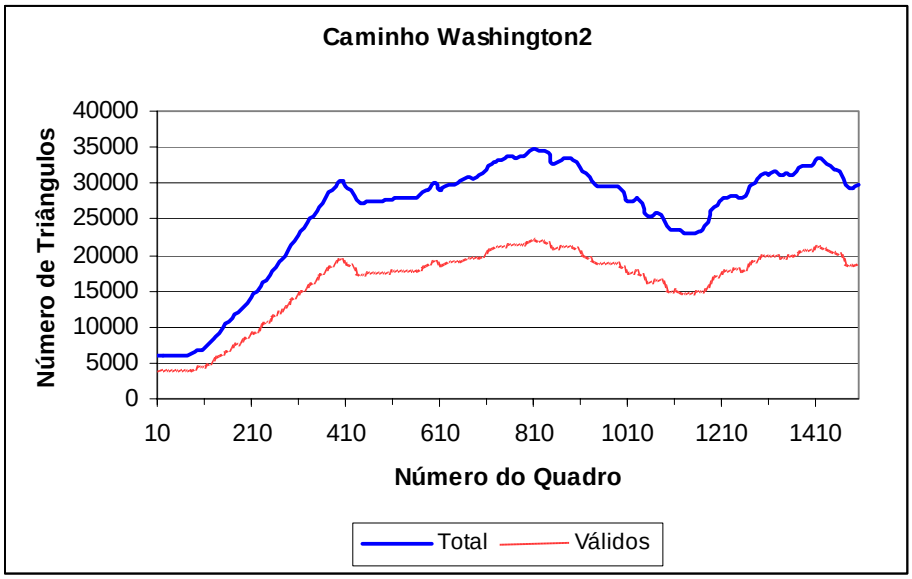
a)



b)



c)



d)

Figura 26 - Gráficos comparativos entre o número total de triângulos ou vértices enviados para o pipeline gráfico e o número de triângulos válidos.

4.2.4. Verificando o Pré-Processamento

Muitos algoritmos para visualização de terrenos mencionam estratégias complexas e tempos consideráveis para esta etapa. A Tabela 7 mostra o tempo em segundos gasto na etapa de pré-processamento de cada um dos quatro terrenos utilizados em nossos testes nas plataformas 1 e 2. Os tempos fornecidos nesta tabela englobam a leitura dos arquivos contendo o campo de alturas e a textura, além, é claro, do procedimento de preparação dos dados de entrada para o refinamento em tempo real. O tempo consumido nesta etapa é proporcional ao tamanho do terreno e dependente do poder de processamento da plataforma. Porém, note que mesmo para o terreno maior, o Washington2, composto por mais de 16 milhões de amostras, o pré-processamento não atinge 2 minutos.

Além disso, pode-se executar o pré-processamento para cada terreno uma única vez e depois mantê-los armazenados em disco. A partir daí quando desejarmos sobrevoar um terreno basta ler o respectivo arquivo pré-processado e a partir dele executar o refinamento em tempo real.

Repare na Tabela 7 que para todos os terrenos, exceto Washington2, os valores de tempo são bem próximos nas duas plataformas. No caso do terreno Washington2 na plataforma 1 o que acontece é que, com nossas estruturas de dados, ele ocupa algo em torno de 900MB em memória e a plataforma 1 só dispõe 512 MB de memória RAM. Desta forma, o sistema precisa utilizar memória virtual e isto justifica um consumo de tempo maior.

Tempo (em s) do Pré-Processamento	Plataforma 1	Plataforma 2
Itaoca1	0,9	0,7
Itaoca2	2,6	2,0
Washington1	1,1	0,8
Washington2	111,0	13,8

Tabela 7 - Tempo médio do pré-processamento de cada terreno.

4.2.5. Comparando os Algoritmos de Lindstrom & Pascucci e de De Boer

Esta seção dedica-se a uma comparação da nossa implementação do algoritmo analisado ao longo deste trabalho e proposto por Lindstrom & Pascucci[2] com uma implementação do algoritmo proposto por De Boer[8] feita por Eduardo Ribeiro Poyart. Este último também trabalha com malhas regulares e é ainda mais simples que o primeiro. A idéia principal do trabalho de De Boer é aproveitar o máximo do potencial oferecido pelos mais recentes hardwares gráficos, enviando o máximo de triângulos que o processador for capaz de gerar e, ao mesmo tempo, que o hardware gráfico suportar. Vale ressaltar, porém, que De Boer não supõe a necessidade de nenhum esquema de *geomorphing*. Assim, para não obter o efeito desagradável de montanhas aparecendo e desaparecendo bruscamente, é preciso usar sempre uma resolução e, conseqüentemente uma tolerância, bem pequenas. Desta forma, procuramos escolher uma tolerância para o algoritmo de De Boer que fornecesse resultados visuais mais ou menos equivalentes aos resultados fornecidos pelo algoritmo de Lindstrom & Pascucci.

Assim, para realizar esta comparação utilizamos todos os caminhos de teste citados anteriormente. Executamos cada caminho uma vez usando o algoritmo de Lindstrom & Pascucci e outra usando o algoritmo de De Boer. Além disso, repetimos isto nas duas plataformas utilizadas para teste.

Os resultados obtidos são mostrados nos gráficos da Figura 27 e nas tabelas 8, 9, 10 e 11. Nestas figuras e tabelas, P1 denota o respectivo algoritmo executado

na plataforma 1 e analogamente P2 na plataforma 2. Assim, por exemplo, Lindstrom-P1 refere-se aos resultados obtidos com o algoritmo de Lindstrom & Pascucci executado na plataforma 1 e assim por diante.

Para desenhar estes gráficos, coletamos o número de triângulos desenhados em um quadro de amostra a cada dez quadros produzidos. Portanto, ao longo de cada caminho, que é composto por 1.500 quadros ou *frames*, coletamos 150 valores.

As tabelas 8, 9, 10 e 11 apresentam o FPS médio e o tempo consumido na etapa de inicialização para os caminhos Itaoca1, Itaoca2, Washington1 e Washington2, respectivamente.

Caminho Itaoca1	Lindstrom-P1	DeBoer-P1	Lindstrom-P2	DeBoer-P2
FPS Médio do Caminho	48,4	42,8	79,0	52,4
Tempo de Inicialização	2,6	0,7	0,7	0,5

Tabela 8 - Comparação do caminho Itaoca1 com ambos os algoritmos e ambas as plataformas.

Caminho Itaoca2	Lindstrom-P1	DeBoer-P1	Lindstrom-P2	DeBoer-P2
FPS Médio do Caminho	31,9	21,1	50,9	23,8
Tempo de Inicialização	2,6	1,5	2,1	1,1

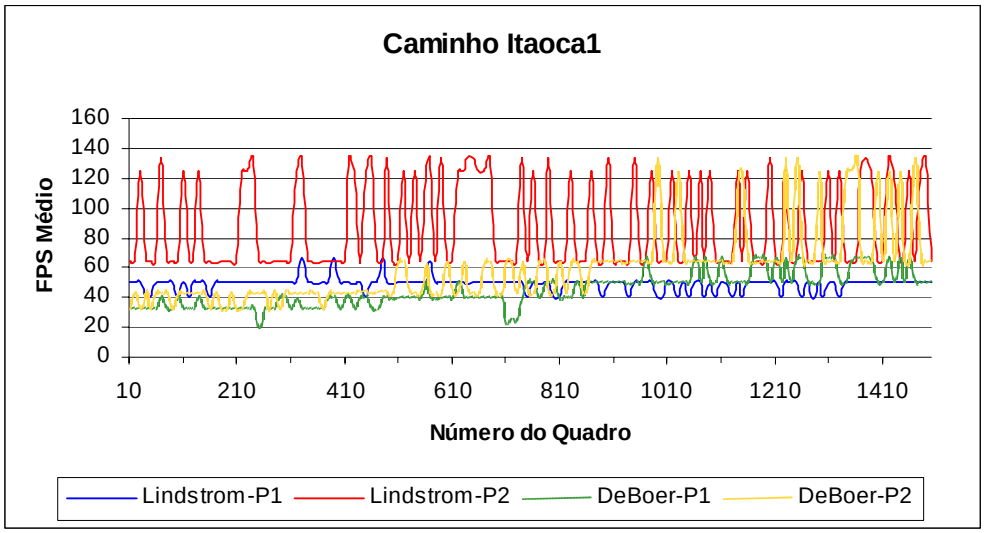
Tabela 9 - Comparação do caminho Itaoca2 com ambos os algoritmos e ambas as plataformas.

Caminho Washington1	Lindstrom-P1	DeBoer-P1	Lindstrom-P2	DeBoer-P2
FPS Médio do Caminho	33,0	25,8	52,0	29,1
Tempo de Inicialização	1,2	1,1	0,9	0,6

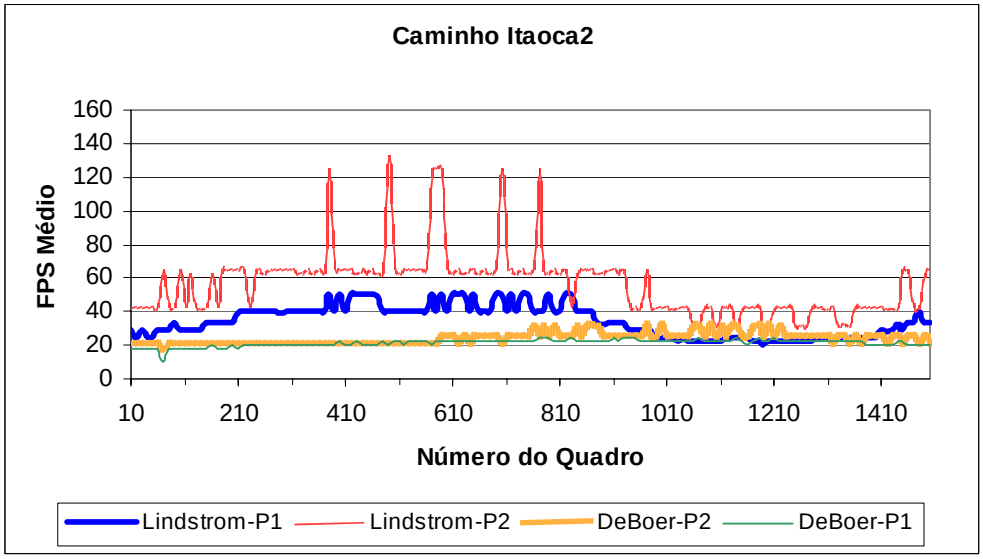
Tabela 10 - Comparação do caminho Washington1 com ambos os algoritmos e ambas as plataformas.

Caminho Washington2	Lindstrom-P1	DeBoer-P1	Lindstrom-P2	DeBoer-P2
FPS Médio do Caminho	20,7	7,4	35,3	8,0
Tempo de Inicialização	98,6	11,8	13,8	5,6

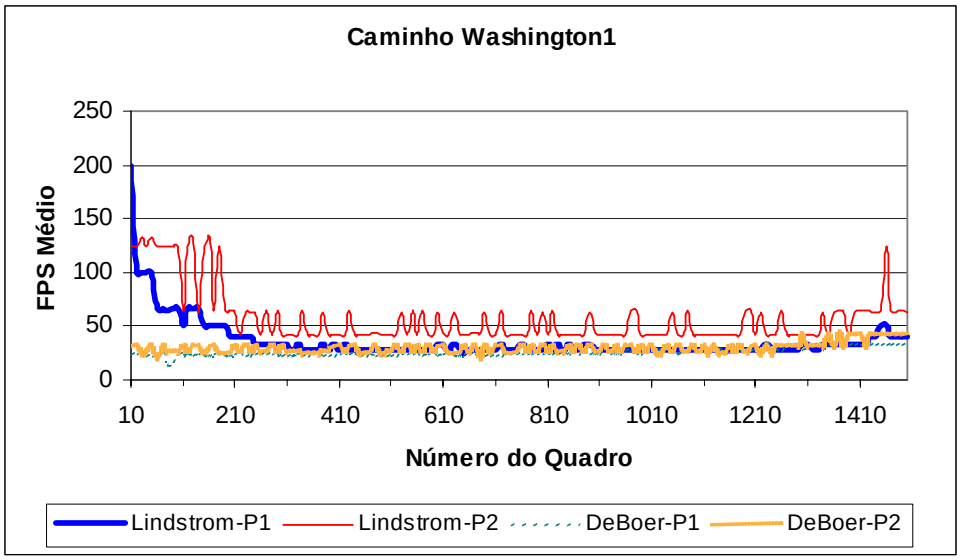
Tabela 11 - Comparação do caminho Washington2 com ambos os algoritmos e ambas as plataformas.



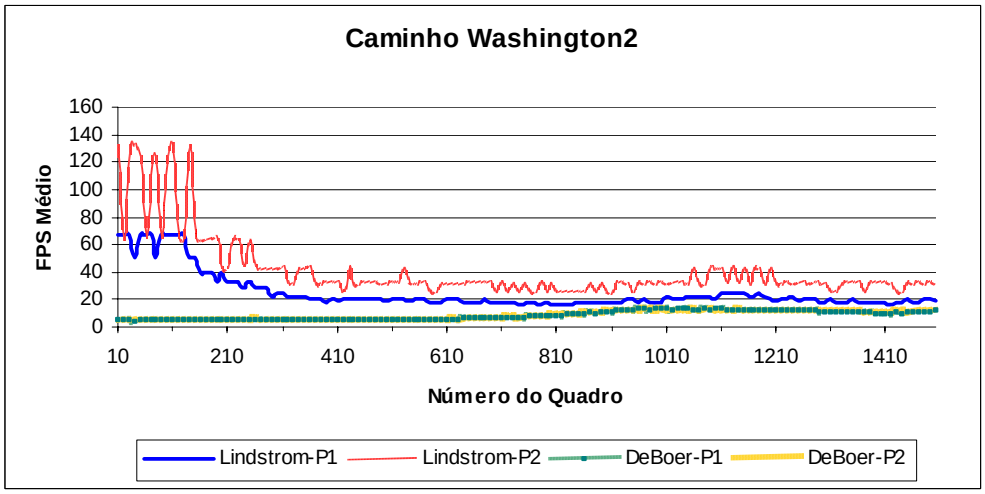
a)



b)



c)



d)

Figura 27 - Comparação dos algoritmos de Lindstrom & Pascucci e de De Boer nas plataformas P1 e P2.

Note que no caminho Itaoca1 não obtemos grandes diferenças. Como trata-se de um terreno pequeno, então podemos dizer que conseguimos obter taxas interativas com ambos os algoritmos e em ambas as plataformas.

Porém, o mesmo não acontece com os outros caminhos, principalmente com o caminho Washington2. Neste caso, por se tratar de um terreno bem maior, podemos dizer que o algoritmo de De Boer enviou mais triângulos para o hardware do que ele podia suportar. Já com o algoritmo de Lindstrom & Pascucci ainda conseguimos obter taxas interativas, mesmo com este terreno maior. Repare

no gráfico que mostra os resultados do caminho Washington2 que nos primeiros quadros o algoritmo de Lindstrom & Pascucci obtém taxas (FPS) bem elevadas, mas depois estas taxas decrescem um pouco. Isto acontece porque o terreno dentro do *frustum* de visão é mais plano nesta parte inicial do caminho. Já o algoritmo de De Boer, por não levar em conta a geometria do terreno, não apresenta esta mesma variação.

Pelo gráfico, podemos ver que, no caminho Washington2 o algoritmo de De Boer nunca ultrapassa o de Lindstrom & Pascucci. No caminho Itaoca1, entretanto, na fase final, mais ou menos a partir do quadro 900, o algoritmo de De Boer consegue obter taxas iguais ou até superiores ao algoritmo de Lindstrom & Pascucci. O que acontece é que, neste trecho, somente uma parte bem pequena do terreno (que já é pequeno) está sendo vista. Assim, como o *culling* da malha fora da visão de De Boer é mais simples e rápido, o algoritmo de De Boer começa a ganhar uma certa vantagem.

Note que o algoritmo de Lindstrom & Pascucci invariavelmente apresenta melhor desempenho na plataforma 2, que é equipada com um processador mais potente. Já o algoritmo de De Boer em geral apresenta o mesmo desempenho nas duas plataformas. Isto acontece, provavelmente, porque o hardware gráfico de cada uma delas tem apresentado desempenhos mais ou menos equivalentes.

Quanto ao pré-processamento, devemos destacar que se no algoritmo de Lindstrom & Pascucci ele é relativamente simples, no algoritmo de De Boer ele praticamente inexiste. Os valores de tempo medidos na fase de inicialização, no caso do algoritmo de De Boer, certamente referem-se ao tempo gasto com a leitura do mapa de alturas e da textura. No caso dos caminhos Itaoca1, Itaoca2 e Washington1, podemos dizer que ambos foram bem rápidos nas duas plataformas, conforme pode ser observado nas tabelas 8, 9 e 10. Já para o caminho Washington2, obtivemos uma diferença relativamente significativa na plataforma 1. Neste caso, o algoritmo de Lindstrom & Pascucci consumiu mais de um minuto e meio na inicialização, enquanto o de De Boer consumiu 11,8 segundos. Contudo, na plataforma 2, que possui um processador mais potente, a diferença não foi tão considerável, conforme pode ser visto na Tabela 11.

4.2.6.
Verificando o Número de Triângulos da Malha Gerada

Para investigar o número de triângulos da malha gerada pelo algoritmo descrito neste trabalho, foi feita uma comparação com um método de simplificação por inserção gulosa baseada no erro vertical local descrito em[15] e que promete gerar uma boa malha aproximada para um dado conjunto de amostras, não necessariamente dispostas em uma grade regular. O algoritmo descrito em [15] não leva em conta nenhum critério perceptual. Em vez disso, ele apenas gera uma malha de triângulos aproximada a partir de um conjunto de amostras e sempre comparando esta malha aproximada com a malha original. Na verdade, é usado o algoritmo de triangulação de Delaunay incremental, usando um método de inserção gulosa como critério de seleção baseado no erro vertical local. Assim, o algoritmo começa com uma triangulação mínima (2 triângulos grandes envolvendo o terreno inteiro), mantendo uma estrutura tipo *heap* para guardar o ponto com maior erro em cada triângulo existente na triangulação corrente. Então, a cada iteração, escolhe-se o ponto com maior erro dentre estes pontos do *heap* para entrar na triangulação de Delaunay. Este procedimento é repetido enquanto o maior erro for maior que uma tolerância especificada.

Assim, para efetuar a comparação do algoritmo em estudo com o algoritmo descrito brevemente acima, desabilitamos o *culling* da malha fora da visão, o *geomorphing* e modificamos o erro projetado na tela para também não levar em conta a posição do observador. Em outras palavras, estamos usando o refinamento básico e fazendo nosso erro no espaço da tela como sendo igual ao erro no espaço do objeto.

Executamos nosso algoritmo modificado para dois terrenos usados no trabalho descrito em [15]. A Tabela 12 abaixo, extraída de [15], mostra os dados destes dois terrenos.

Terreno	Dim X	Dim Y	Total de Triângulos	Alt. mínima	Alt. máxima	Variação máxima	Desvio padrão
cvzbuffalo	120	120	28832	15,80	64,2	48,40	13,22
Ilha	256	256	130050	0,070	17,08	17,01	3,39

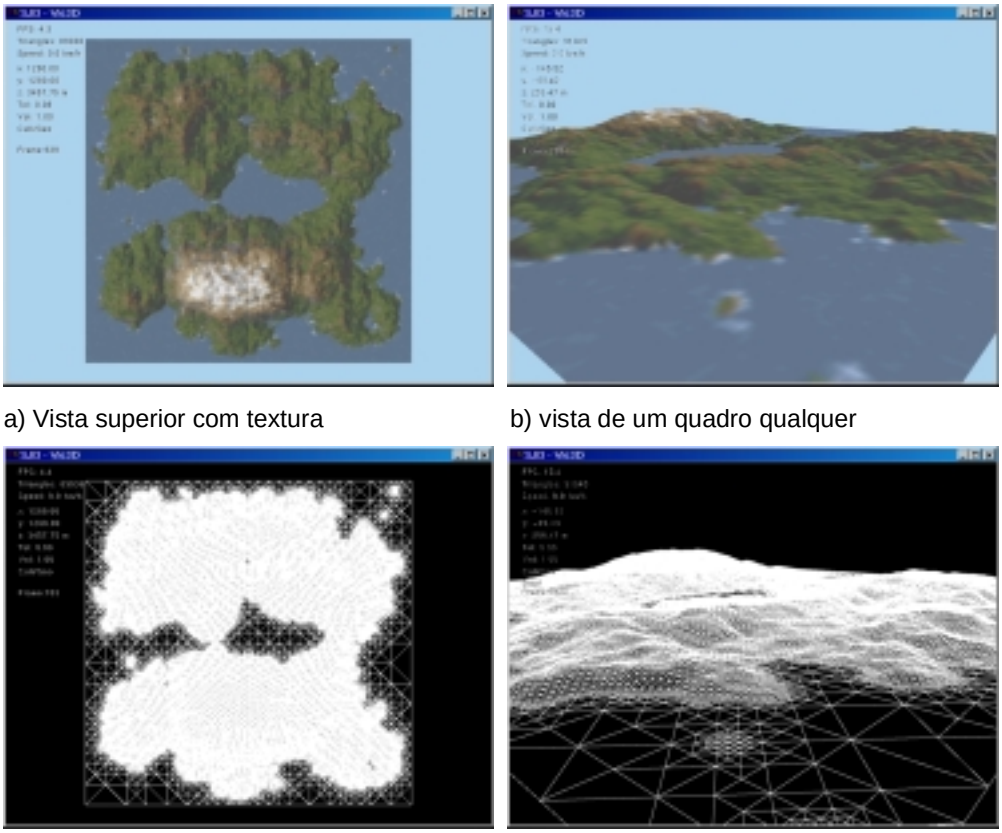
Tabela 12 - Características dos terrenos testados.

Na Tabela 13 apresentamos os resultados da comparação usando os dois algoritmos para o terreno ilha.

Delaunay com inserção gulosa	Lindstrom & Pascucci modificado	Erro
2.498	1.297	0,97
4.998	3.086	0,56
7.498	5.585	0,39
9.998	8.060	0,32
12.498	9.932	0,28
14.998	12.153	0,25

Tabela 13 - Número de triângulos gerados para o terreno ilha.

Na Figura 28 mostramos algumas imagens do terreno ilha.



a) Vista superior com textura

b) vista de um quadro qualquer

c) quadro (a) em wireframe e sem textura

d) quadro (b) em wireframe e sem textura.

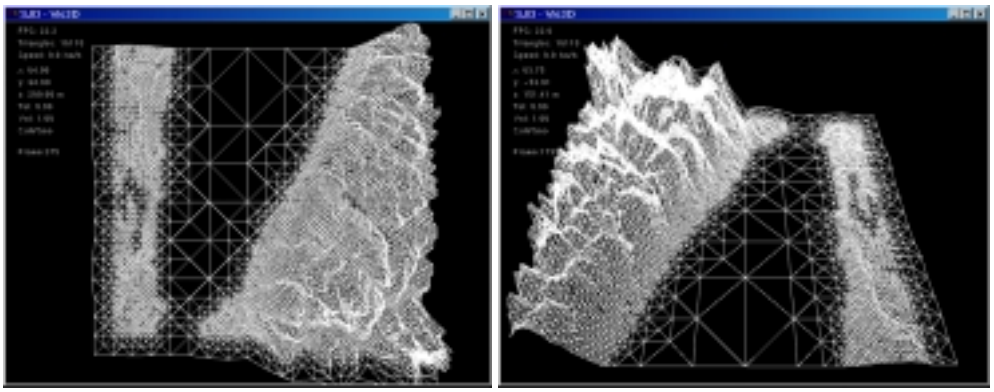
Figura 28 - Imagens do terreno ilha

Na Tabela 14 apresentamos os resultados da comparação usando os dois algoritmos para o terreno cvzbuffalo.

Delaunay com inserção gulosa	Lindstrom & Pascucci modificado	Erro
2.497	4.098	2,10
4.998	7.380	1,00
7.498	9.823	0,50
9.947	11.451	0,33
12.497	14.535	0,08
14.998	16.110	0,00

Tabela 14 - Número de triângulos gerados para o terreno cvzbuffalo

Na Figura 29 mostramos algumas imagens do terreno cvzbuffalo. Repare que não temos uma textura específica para este terreno.



a) Vista superior b) vista de um quadro qualquer

Figura 29 - Imagens do Terreno cvzbuffalo

Pelas tabelas 13 e 14, podemos verificar que para o terreno ilha o algoritmo de Lindstrom & Pascucci modificado consegue obter uma malha aproximada com menor número de triângulos que a malha produzida pelo algoritmo de Delaunay com inserção gulosa. Já para o terreno cvzbuffalo acontece o contrário. Neste caso, a malha gerada pelo algoritmo de Delaunay com inserção gulosa contém menos triângulos que a malha gerada pelo algoritmo de Lindstrom & Pascucci modificado.

O que acontece é que estes dois terrenos não encaixam-se no formato $(2^{n/2}+1)$ x $(2^{n/2}+1)$ exigido pelo algoritmo de Lindstrom & Pascucci e então eles precisaram ser estendidos. Porém para o terreno ilha esta estensão consiste de apenas uma faixa de um vértice em cada direção.

Enfim, podemos concluir que o algoritmo de Lindstrom & Pascucci, quando trabalha com terrenos que possuem dimensões $(2^{n/2}+1)$ x $(2^{n/2}+1)$ ou valores bem

próximos disto (como o terreno ilha que possui dimensões $2^{n/2}$ x $2^{n/2}$), funciona muito bem, gerando uma malha aproximada provavelmente bem próxima a uma malha ótima. Porém, quando o terreno não encaixa-se neste formato e é necessário estendê-lo, não se pode continuar afirmando o mesmo.