Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Fábio Markus Nunes Miranda

MONOGRAFIA DE PROJETO ORIENTADO EM COMPUTAÇÃO I

Desenvolvimento de um Arcabouço para a Geração Procedural e Visualização de Terrenos em Tempo-Real

Belo Horizonte – MG
2008 / 2º semestre

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Desenvolvimento de um Arcabouço para a Geração Procedural e Visualização de Terrenos em Tempo-Real

por

Fábio Markus Nunes Miranda

Monografia de Projeto Orientado em Computação I

Apresentado como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFMG

Prof. Dr. Luiz Chaimowicz
Orientador

Carlúcio Cordeiro
Co-Orientador

Assinatura do aluno:

Assinatura do orientador:

Assinatura do co-orientador:

Belo Horizonte – MG
2008 / 2º semestre

À Deus,
aos professores,
aos colegas de curso e
aos meus familiares,
dedico este trabalho.

Agradecimentos

Agradeço aos meus pais, pelo apoio durante o curso.
Aos meus professores e orientadores, pelos conhecimentos adquiridos.
E finalmente aos colegas de curso pela convivência e trocas de experiências.

"'Fractal geometry will make you see everything differently.
You risk the loss of your childhood vision of clouds, forests,
flowers, galaxies, leaves, feathers, rocks, mountains,
torrents of water, carpets, bricks, and much else besides.
Never again will your interpretation of these things be quite the same."

Michael F. Barnsley

Resumo

Com o aumento da demanda por modelos 3D em áreas como jogos eletrônicos, a geração procedural se faz cada vez mais necessária, para reduzir custos e tempo de desenvolvimento. Através de técnicas procedurais, é possível criar um grande número de modelos com a variação de alguns poucos parâmetros dos algoritmos responsáveis pela geração. Porém, técnicas procedurais se caracterizam por uma baixa interação do usuário, gerando assim modelos nem sempre expressivos. Este trabalho é focado na geração de terrenos proceduralmente e aqui será proposto um arcabouço que insira no processo uma maior participação do usuário, com o objetivo de criar modelos que traduzam melhor a sua espectativa. Técnicas procedurais já difundidas são exploradas e mescladas com inserção de terrenos e modelos por parte do usuário.

Palavras-chave: Geração procedural de conteúdo, terreno.

Abstract

With the increasing demand for 3D models in areas such as games, the procedural content generation is becoming increasingly popular, in order to reduce costs and development time. With procedural techniques, it is possible to create a large number of models modifying a few parameters on the algorithms responsable for the generation. However, procedural techniques have a low user interaction, creating models that are not always expressive. This work focus is on procedural generation of terrains and it proposes a framework that increases the user participation level, in order to create models that better translate his expectations. Well know procedural techniques are explored and mixed with terrain and models inserted by the user.

Keywords: Procedural content generation, terrain.

Lista de Figuras

Figura 2.1 Exemplo de uma cena com um carro e um ponto de luz. 15

Figura 2.2 Exemplo de um grafo de cena. 15

Figura 2.3 Exemplo de um mapa de altura. 16

Figura 2.4 Exemplo de um ruído de Perlin. 16

Figura 2.5 Retângulo e seus quatro pontos. 17

Figura 4.1 Classes representando os nodos. 20

Figura 4.2 Grafo de cena e terrenos, com a câmera no Terreno 0. 21

Figura 4.3 Grafo de cena e terrenos, com a câmera no Terreno 5. 21

Figura 4.4 Visualização e geração dos terrenos. 21

Figura 4.5 Tela com o terreno gerado (exibição em *wireframes*). 22

Figura 4.6 Tela com o terreno gerado. 23

Figura 4.7 Tela com o terreno gerado (exibição em *wireframes*). 23

Figura 4.8 Tela com o terreno gerado. 24

Figura 4.9 Mapa de altura inserido pelo usuário. 24

Figura 4.10 Tela com o terreno gerado e um mapa de altura inserido. 25

Figura 4.11 Tela com o terreno gerado e um mapa de altura inserido (exibição em *wire-frames*). 25

Figura 4.12 Tela com a *interface* do usuário (exibição em *wireframes*). 26

Figura 4.13 Tela com a *interface* do usuário. 26

Figura 4.14 Gráfico com a porcentagem dos tempos médios de cada chamada em relação ao tempo total médio de geração 28

Figura 4.15 Teste variando o número de *octaves*, e o número de terrenos vizinhos fixo em 2. 29

Figura 4.16 Teste variando o número de terrenos vizinhos, e o número de octaves fixo. . . 30

Lista de Tabelas

Tabela 4.1 Tempos da geração dos terrenos (em segundos) 27

Tabela 4.2 Tempos da geração dos terrenos (em segundos) 28

Tabela 4.3 Porcentagem dos tempos médios de cada chamada em relação ao tempo total médio de geração 28

Tabela 4.4 FPS das execuções variando o número de *octaves*, e o número de terrenos vizinhos fixo em 2. 29

Tabela 4.5 FPS das execuções variando o número de terrenos vizinhos, e o número de octaves fixo. 30

Lista de Siglas

API	Application Programming Interface
VBO	Vertex Buffer Object
GPU	Graphics Processing Unit
POO	Programação Orientada a Objetos
FPS	Frames por segundo
XML	Extensible Markup Language

Sumário

1 INTRODUÇÃO 12

1.1 Visão geral 12

1.2 Objetivo, justificativa e motivação 12

2 REFERENCIAL TEÓRICO 14

2.1 Grafo de cena 14

2.2 Mapa de altura 15

2.3 mapaaltura 15

2.4 Terrenos fractais 16

2.4.1 Ruído de Perlin 16

2.4.2 Fractal plasma 17

3 METODOLOGIA 18

3.1 Tipo de Pesquisa 18

3.2 Procedimentos metodológicos 18

4 RESULTADOS E DISCUSSÃO 20

4.1 Arcabouço 20

4.2 Grafo de cena implementado 21

4.3 Terrenos gerados 22

4.4 Testes 27

4.4.1 Teste 1 27

4.4.2 Teste 2 29

4.4.3 Teste 3 30

5 CONCLUSÕES E TRABALHOS FUTUROS 31

Referências Bibliográficas 32

1 INTRODUÇÃO

1.1 Visão geral

A geração procedural de modelos é uma área da Ciência da Computação que propõe que modelos gráficos tridimensionais (representação em polígonos de algum objeto) possam ser gerados através de rotinas e algoritmos. Tal técnica vem se tornando bastante popular nos últimos tempos, tendo em vista que, com o crescimento da indústria do entretenimento, há uma necessidade de se construir modelos cada vez maiores e com um grande nível de detalhe. A técnica de geração procedural vem então como uma alternativa à utilização do trabalho de artistas e modeladores na criação de modelos tridimensionais.

1.2 Objetivo, justificativa e motivação

O objetivo deste trabalho é construir um arcabouço para a criação de terrenos proceduralmente em tempo real e que permita a inserção de modelos pelo usuário. O trabalho pode ser divido em duas vertentes: criação de terrenos e a sua respectiva visualização.

Na primeira parte, criação de terrenos, foi realizado um estudo de diversos algoritmos e técnicas procedurais para a criação de terrenos.

O segundo aspecto (a visualização) também é outro problema muito estudado no campo da computação. O modelo de um terreno é algo que pode demandar um número extremamente alto de triângulos, inviabilizando a sua renderização em tempo real nos computadores atuais. Faz-se então necessária a utilização de técnicas que limitam e minimizam o número de triângulos a serem desenhados na tela. Entra aí o uso de *culling* e níveis de detalhe dos modelos.

Um problema recorrente em todos os tipos de geração procedural é a falta de controle do usuário com o resultado gerado. Quanto mais procedural é um sistema, menos expressivo ele se torna [1]. Como exemplo, podemos citar programas de modelagem 3D como *Maya* ou *3dMax*: eles oferecem centenas de entradas possíveis (geometrias representando cones, esferas,

etc.), e dão ao usuário o poder de criar praticamente tudo que é imaginável [2], resultando em uma alta expressividade; porém, demandam mais tempo e, como possuem um grande número de entradas, são pouco procedurais. Já o jogo *Elite* [3], de 1984, cria universos a partir de um único número (*seed*); é um exemplo de algo pouco expressivo, mas altamente procedural.

Por isso, o objetivo deste trabalho é também dar uma maior expressividade à geração procedural de terrenos, sem que, no entanto, essa geração se torne tão trabalhosa quanto criar um modelo tridimensional no *Maya* ou *3dMax*.

Em resumo, o trabalho desenvolvido ao longo das matérias de POC I e POC II, é um arcabouço que una a geração procedural automática com dados inseridos pelo usuário, seja na forma de modelos tridimensionais ou então de mapas de altura (imagens em preto-e-branco que representam algum terreno). Estes dados inseridos pelo usuário podem representar áreas de maior interesse para ele, e que necessitam de uma representação mais fiel.

2 REFERENCIAL TEÓRICO

Vários trabalhos publicados abordam a geração procedural de modelos. Alguns destes trabalhos abordam a geração de cidades ([4] e [5]), outros abordam a geração de terrenos realistas (em tempo-real, como os trabalhos [6] e [7], ou não, como o *MojoWorld*[8]), ou então a geração de árvores [9]. A principal referência na área é o livro *Texturing and Modeling: A Procedural Approach* [10], em que é explicada a geração procedural de diversos tipos de modelos.

Algumas técnicas largamente utilizadas na geração procedural são: Sistemas de Lindenmayer (*l-System*)[11], que, através de uma gramática, modela o crescimento de plantas; geometrias fractais [12]; e também ruído de Perlin (*perlin noise*[13]), que será abordado na próxima seção. Todos possuem o mesmo objetivo: tentar sintetizar, através de um algoritmo, um fenômeno natural. Algumas técnicas pertinentes a este trabalho serão abordadas na Seção 2.4.

Nas Seções 2.1 e **??** será apresentado alguns conceitos relevantes neste trabalho, como Grafo de Cena e Mapa de Altura.

2.1 Grafo de cena

Um grafo de cena é uma estrutura de dados largamente utilizada em aplicações gráficas e jogos eletrônicos. O seu objetivo é organizar os objetos presentes em um ambiente virtual (uma cena). Todo os grafos de cena são acíclicos e dirigidos.

No exemplo da Figura 2.2 [**??**], o grafo representa uma cena com um carro (*car*) e uma luz (*light*). O nodo carro possui, como nodos filhos, quatro rodas (*wheel*). A Figura 2.1 mostra a cena com o carro e um ponto de luz.

Figura 2.1: Exemplo de uma cena com um carro e um ponto de luz.

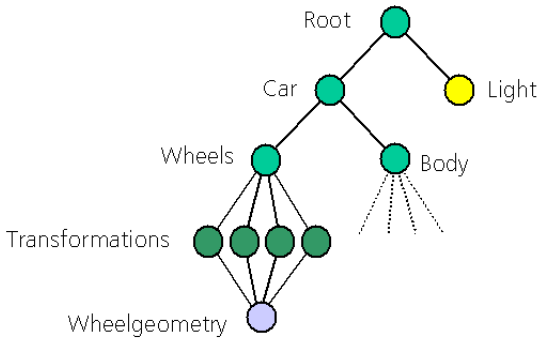

Figura 2.2: Exemplo de um grafo de cena.

A maior vantagem de um grafo de cena é a possibilidade de aplicar transformações (como rotação, translação, escala) a um nodo, e os filhos desse nodo também sofrerão tais transformações. Técnicas mais avançadas, como desenhar na tela apenas parte de uma cena (ou *culling*), também podem ser implementadas através de grafos de cena.

2.2 Mapa de altura

2.3 mapaaltura

Um mapa de altura (ou *heightmap*) é uma imagem bidimensional que armazena dados referentes à altura de um terreno. Geralmente, tons mais claros representam pontos mais altos, enquanto tons mais escuros são pontos mais baixos do mapa. A Figura **??** [**??**] é um exemplo de mapa de altura.

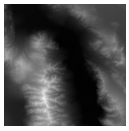

Figura 2.3: Exemplo de um mapa de altura.

2.4 Terrenos fractais

Existem uma série de técnicas para criação de terrenos fractais, como *perlin noise* e o algoritmo *fractal plasma*, detalhados a seguir.

2.4.1 Ruído de Perlin

O ruído de Perlin foi criado pelo Professor Ken Perlin, da *New York University*. O ruído é usado para simular estruturas naturais, como nuvens, texturas de árvores, e terrenos.

Para criar um ruído de Perlin, precisamos de uma função que retorne, para um dado domínio, números entre 0 e 1. Para gerarmos esses números, é preciso uma semente (*seed*); dessa forma, em uma segunda execução, com uma mesma semente, teremos os mesmos números entre 0 e 1.

A Figura 2.4 [14] mostra três funções de ruído, criadas com diferentes valores de amplitude e frequência. A primeira função poderia ser uma representação de montanhas, a segunda de morros, a terceira de blocos de pedras. Cada função de ruído é chamada de um *octave*.

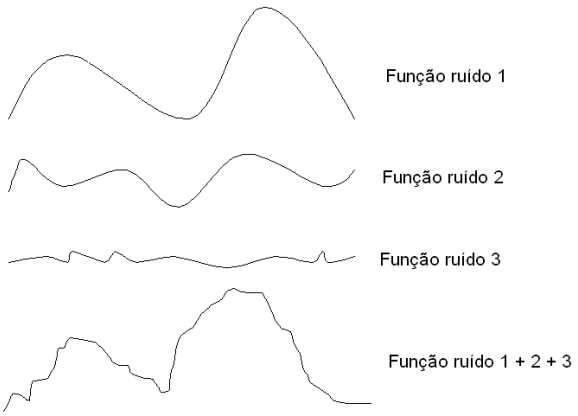

Figura 2.4: Exemplo de um ruído de Perlin.

2.4.2 Fractal plasma

O fractal plasma é um outro algoritmo utilizado na geração de terrenos. Basicamente, ele inicia com um retângulo onde, na primeira iteração, cada aresta recebe um peso aleatório, como mostra a Figura 2.5. Na segunda iteração, o retângulo é particionado, resultando em quatro retângulos, cujos novos pesos serão a média dos pesos das arestas adjacentes, que pode ser acrescido de um pequeno erro (que irá determinar o númerode "montanhas" no terreno). Ao final de um número de iterações (determinado pelo usuário), cada média dos pesos irá corresponder a uma determinada altura do terreno.

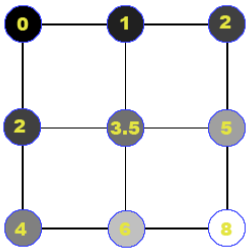

Figura 2.5: Retângulo e seus quatro pontos.

3 METODOLOGIA

3.1 Tipo de Pesquisa

O trabalho proposto é uma pesquisa de natureza aplicada, pois busca aplicar um conhecimento teórico (técnicas de geração procedural) e obter um resultado prático na forma de um arcabouço e tem um objetivo exploratório. A pesquisa se dá em laboratório, pois se trata de um ambiente controlado.

3.2 Procedimentos metodológicos

O primeiro passo do trabalho foi a escolha de uma *Application Programming Interface* (API). Por se tratar de uma plataforma aberta e já estudada em matérias durante o curso, o *OpenGL* foi escolhido, juntamente com a linguagem C++. Além disso, essa *API* oferece algumas extensões que permitem maximizar a performance do sistema, como o a estrutura *Vertex Buffer Object* (VBO), que armazena os vértices do modelo 3D diretamente na memória da *Graphics Processing Unit*(GPU), diminuindo o número de chamadas para a renderização. O *OpenGL* também possui algumas bibliotecas, como o *GLFW* [15], para tratamento de eventos do mouse e teclado, o *AntTweakBar* [16], para a construção de interfaces gráficas e o *DevIL* [17] para a leitura de imagens.

O desenvolvimento do arcabouço seguiu técnicas de *Programação Orientada a Objetos* (POO), sempre com a intenção de deixar o sistema o mais flexível possível. Um estudo do funcionamento de *frameworks* também foi necessário, envolvendo questões como *hot spots*, *frozen spots*, caixa-preta, caixa-branca [18].

Foi pesquisado também diversas técnicas procedurais envolvidas na geração de terrenos, como, por exemplo, ruído de Perlin, algoritmo fractal plasma, bem como algumas sistemas que oferecem soluções ligadas à geração procedural, como o *CityEngine* [19] e o *MojoWorld* [8].

Por fim, foi construído um arcabouço que servirá como base para a implementação de algoritmos de geração procedural de terrenos, e uma *interface* que permita uma maior interação com o usuário. Além disso, também foi construído um grafo de cena para a organização dos terrenos. Os detalhes serão apresentados na próxima seção.

4 RESULTADOS E DISCUSSÃO

Os conhecimentos adquiridos ao longo desse trabalho permitiram a criação de um arcabouço capaz de gerar terrenos procedurais e também de levar em consideração algumas entradas do usuário. Na Seção 4.1 será apresentado tal arcabouço. Na Seção 4.2 será mostrado o grafo de cena implementado e, finalmente, na Seção 4.3 alguns exemplos de terrenos gerados proceduralmente com o arcabouço.

4.1 Arcabouço

O arcabouço implementado aqui teve como um dos principais objetivos a possibilidade de futuras expansões. A Figura 4.1 mostra as classes já implementadas, como *PerlinNoise* e *Plasma* (para geração procedural através dos algoritmos de ruído de Perlin e Plasma), e também a *FileHeightmap*, que é a responsável por ler mapas de altura de arquivos de imagem. Quaisquer novos algoritmos de geração procedural de terrenos devem ser filhos da classe *Terrain*, que já encapsula uma série de funções pertinentes à visualização de terrenos.

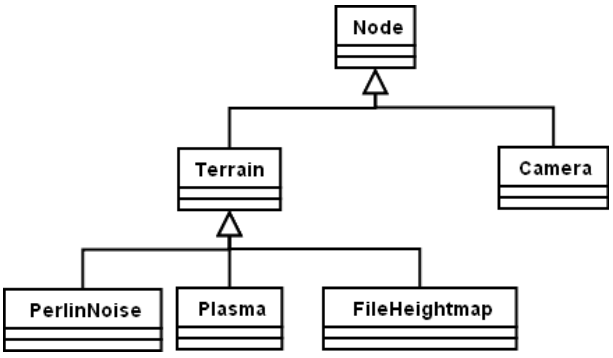

Figura 4.1: Classes representando os nodos.

A câmera implementada no arcabouço permite que o usuário navegue pelo terreno com o controle do mouse e do teclado.

4.2 Grafo de cena implementado

Para a visualização dos terrenos, foi criada uma estrutura de grafo de cena. Na estrutura, cada nodo representa um terreno gerado. A Figura 4.2 considera que a câmera está situada acima do *Terreno 0*, e assim ele possui oito nodos filhos ao seu redor.

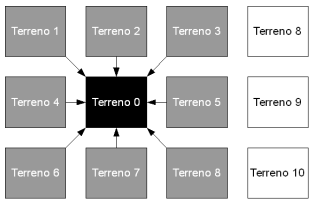

Figura 4.2: Grafo de cena e terrenos, com a câmera no Terreno 0.

Na Figura 4.3, a câmera passa a estar em cima do Terreno 5. Logo, serão gerados os terrenos 8, 9 e 10. Os terrenos 1, 4 e 6 (em branco) poderão ser excluídos, enquanto que os terrenos 0, 2, 3, 7 e 8 passarão a apontar para um novo nodo pai: o nodo que representa o terreno 5.

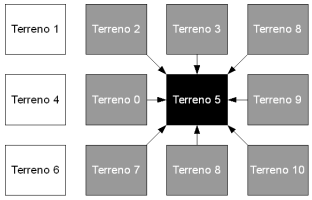

Figura 4.3: Grafo de cena e terrenos, com a câmera no Terreno 5.

Outro aspecto levado em consideração, foi o número de terrenos gerados e o número de terrenos que estão sendo visualizados. Na Figura 4.4, os terrenos em branco foram gerados, mas não serão renderizados na tela, para economizar recursos da placa de vídeo. O número e as distâncias entre os terrenos gerados e visualizados são parâmetros do arcabouço.

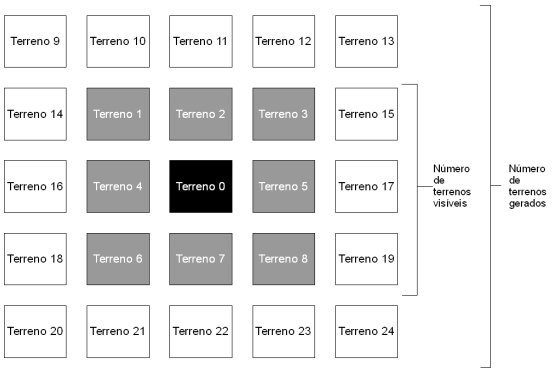

Figura 4.4: Visualização e geração dos terrenos.

4.3 Terrenos gerados

As Figuras 4.5 e 4.6, e 4.7 e 4.8 mostram os terrenos gerados variando o número de iterações do ruído de Perlin e o número de terrenos vizinhos exibidos.

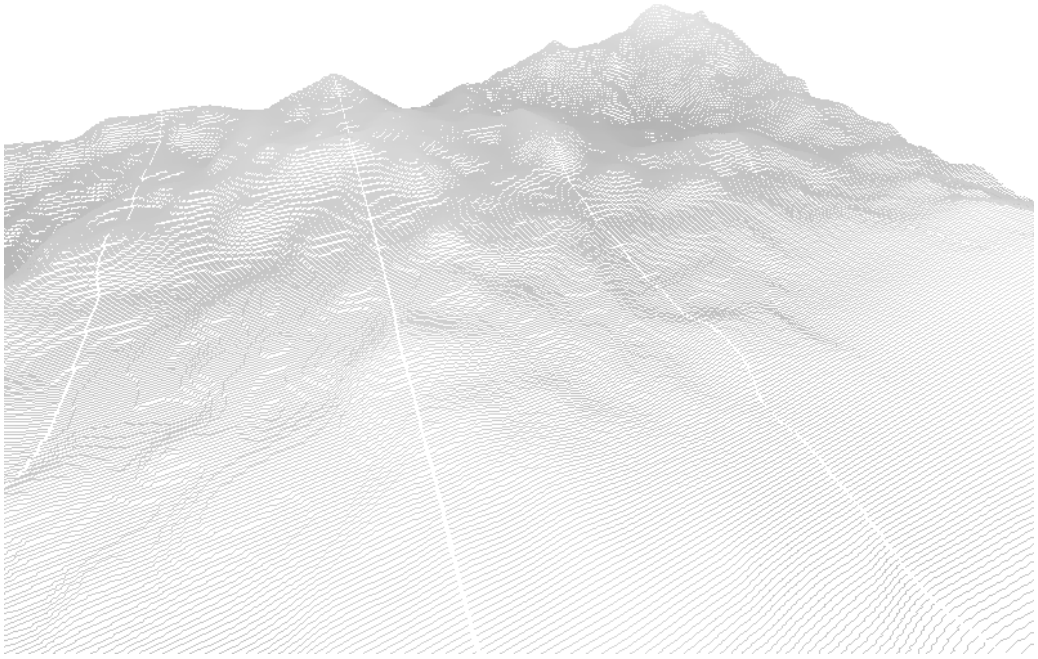

Figura 4.5: Tela com o terreno gerado (exibição em *wireframes*).

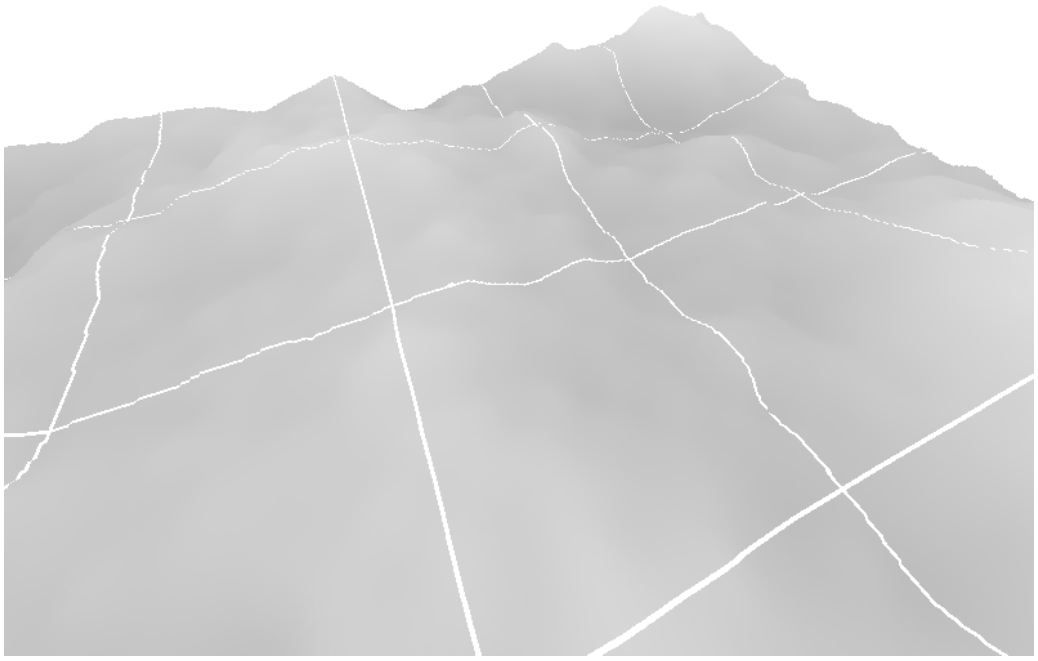

Figura 4.6: Tela com o terreno gerado.

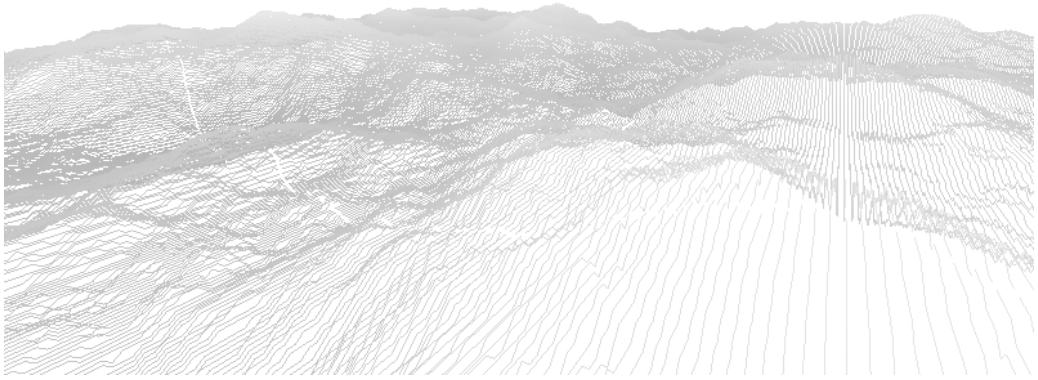

Figura 4.7: Tela com o terreno gerado (exibição em *wireframes*).

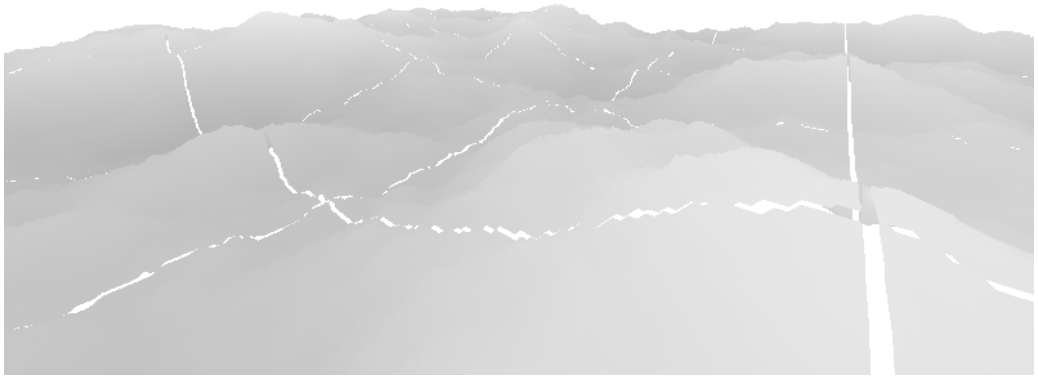

Figura 4.8: Tela com o terreno gerado.

As Figuras 4.10 e 4.11 mostram um terreno gerado proceduralmente e o mapa de altura exibido na Figura 4.9 inserido no arcabouço (mostrado no arcabouço em um tom cinza mais escuro).

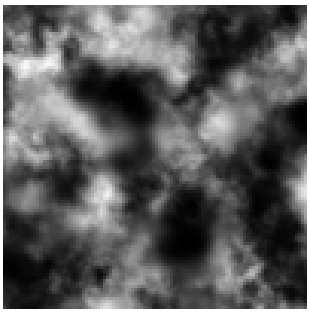

Figura 4.9: Mapa de altura inserido pelo usuário.

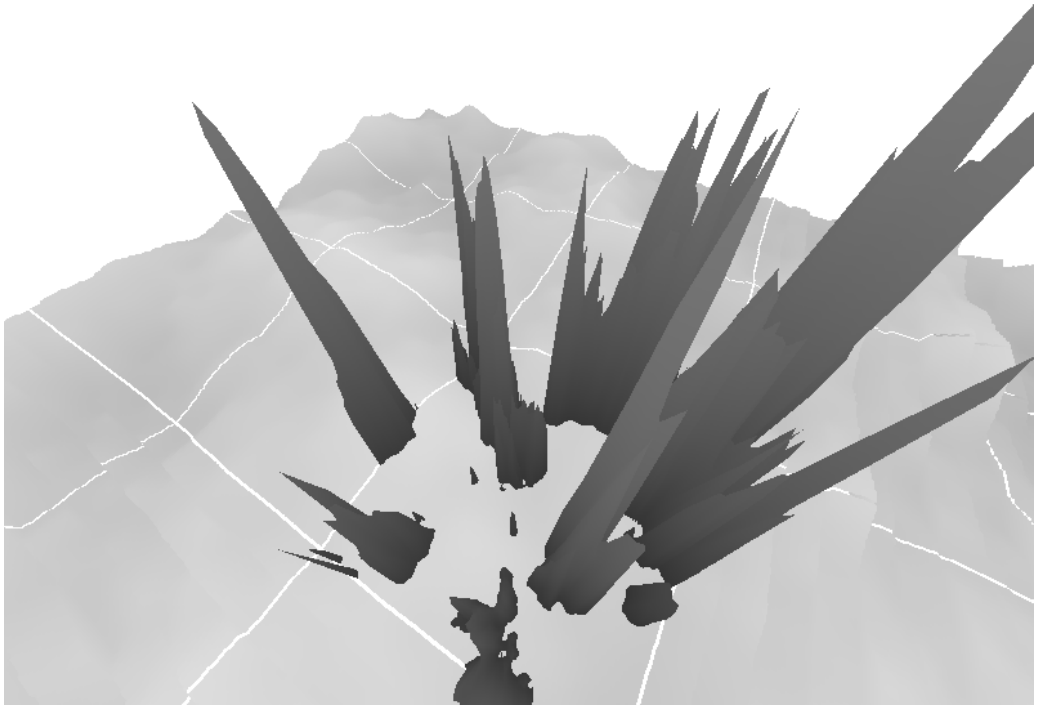

Figura 4.10: Tela com o terreno gerado e um mapa de altura inserido.

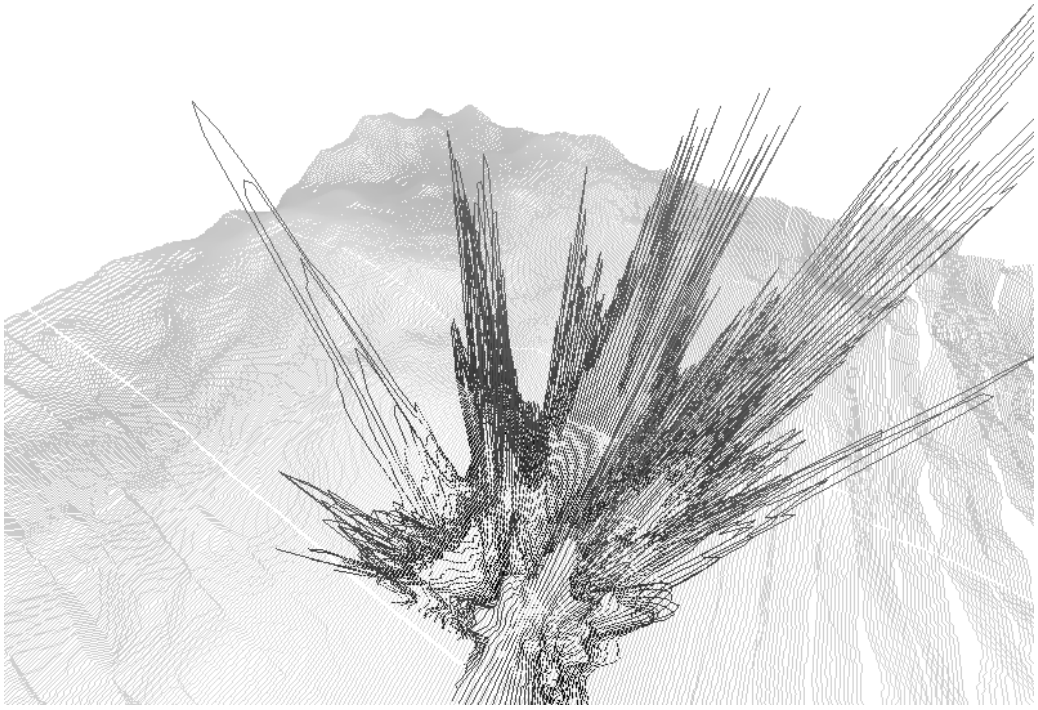

Figura 4.11: Tela com o terreno gerado e um mapa de altura inserido (exibição em *wireframes*).

As Figuras 4.12 e 4.13 mostram os terrenos com uma *interface* provisória, onde é possível escolher os parâmetros dos terrenos a serem gerados.

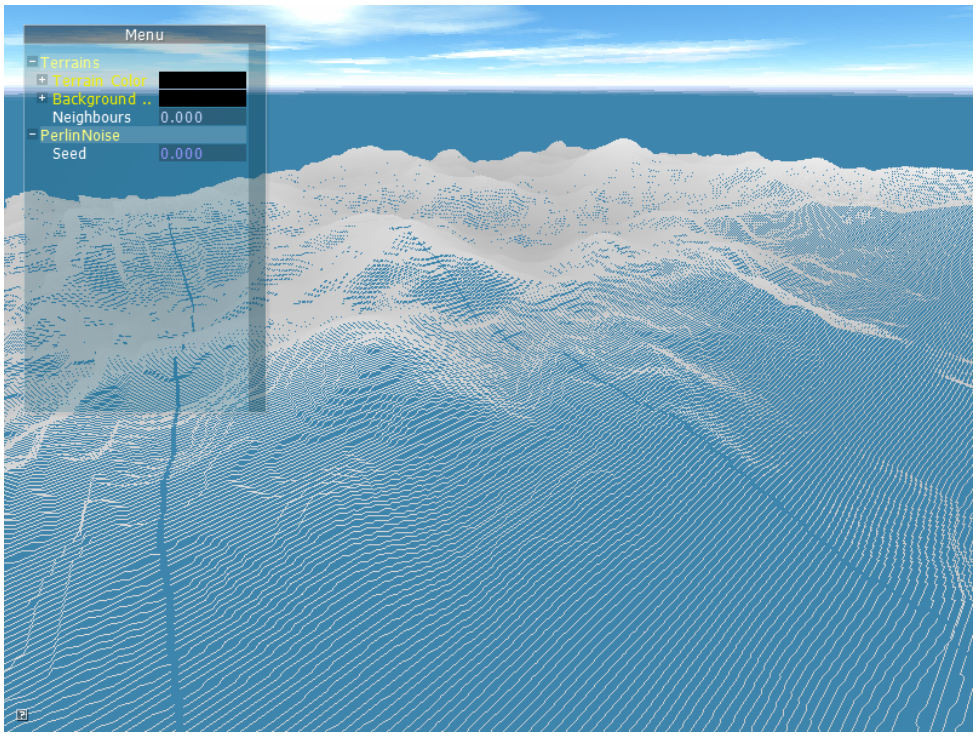

Figura 4.12: Tela com a *interface* do usuário (exibição em *wireframes*).

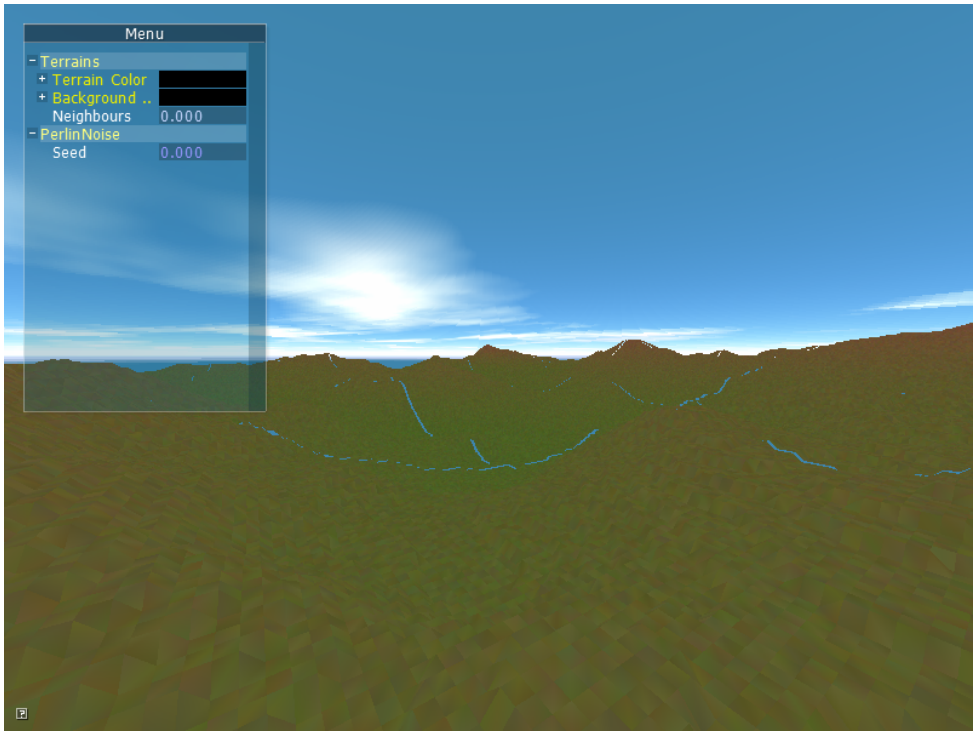

Figura 4.13: Tela com a *interface* do usuário.

4.4 Testes

Alguns testes foram feitos, para avaliar a variação de frames por segundo (FPS) com a alteração de alguns parâmetros. Eles foram executados em um *Athlon64 3500*, com 2GB de memória *RAM* e placa de vídeo GeForce6600 com 64MB de memória.

A Seção 4.4.1 mostra um teste com a geração de 100 terrenos proceduralmente.

As Seções 4.4.2 e 4.4.3 apresentam testes que consistiam em um vôo da câmera pelo terreno durante 60 segundos, em um trajeto constante para todos os testes.

4.4.1 Teste 1

No primeiro teste, foi medido o tempo gasto com a geração de terrenos. Para cada número de octaves (4, 16 e 32), foi gerado 100 terrenos, e medido o tempo gasto.

Nas Tabelas 4.1 e 4.2 é apresentado os tempos das chamadas às funções responsáveis pela geração dos terrenos, considerando o algoritmo de ruído de Perlin. Abaixo uma explicação sobre cada função:

- FillHeightmap: Constrói o terreno, determinando a altura e posição de cada vértice e armazena em um vetor de *float*. Sua complexidade é *O(n*m*c)*, onde *n* é a largura do terreno, *m* o comprimento e *c* é o número de octaves.
- CopyHeightmap: Copia o vetor, construído na função *FillHeightmap* para um novo vetor VBO. Sua complexidade é *O(n*m)*, onde *n* é a largura do terreno e *m* o comprimento.
- BuildVBOs: Modifica os ponteiros para o desenho dos vetores VBO. É independente do tamanho do vetor (*O(1)*).

-	FillHeightMap			CopyHeightMap		
Octaves	Mín.	Máx.	Média	Mín.	Máx.	Média
4	0,0410248	0,0536568	0,0443777	0,0737747	0,0866761	0,0782942
16	0,1644840	0,2593410	0,1698030	0,0710867	0,0865546	0,0734247
32	0,3315010	0,4503400	0,3405685	0,0708154	0,0778093	0,0730120

Tabela 4.1: Tempos da geração dos terrenos (em segundos)

-	BuildVBOs			Total		
Octaves	Mín.	Máx.	Média	Mín.	Máx.	Média
4	0,0078359	0,0161562	0,0107964	0,1264320	0,1498800	0,1336476
16	0,0081773	0,0407795	0,0114127	0,2469740	0,3459500	0,2550180
32	0,0075448	0,0129804	0,0093645	0,4121210	0,5310980	0,4231183

Tabela 4.2: Tempos da geração dos terrenos (em segundos)

A Tabela 4.3 apresenta os percentuais de tempo de cada chamada da função em relação ao tempo médio total gasto com a geração de um terreno. A Figura 4.14 mostra um gráfico com esses dados.

Octaves	FillHeightMap	CopyHeightMap	BuildVBOs	Total
4	32,2%	58,6%	9,2%	100%
16	66,6%	28,8%	4,6%	100%
32	80,5%	17,3%	2,2%	100%

Tabela 4.3: Porcentagem dos tempos médios de cada chamada em relação ao tempo total médio de geração

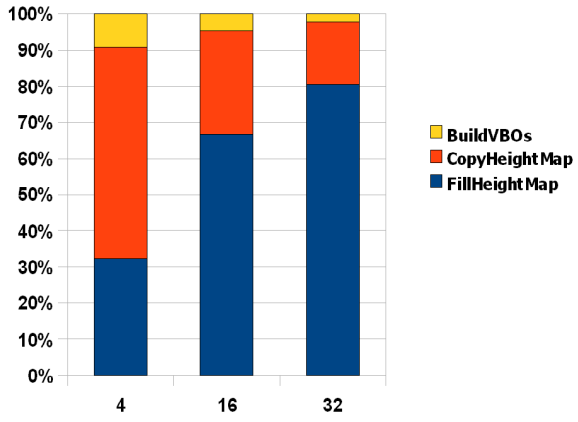

Figura 4.14: Gráfico com a porcentagem dos tempos médios de cada chamada em relação ao tempo total médio de geração

Como podemos observar, o tempo gasto com a cópia dos dados para a nova estrutura de dados (*CopyHeightMap*) gasta um tempo considerável nas três medições (com 4, 16 e 32 octaves), algo a ser considerado em trabalhos futuros.

4.4.2 Teste 2

O próximo teste (Tabela 4.4 e Figura 4.15) mostra o impacto na mudança do número de *octaves*, considerando o número de terrenos vizinhos fixo em 2. Quanto maior o número de *octaves*, maior o número de vértices da malha do terreno; explicando assim o FPS menor.

Por causa dessa relação entre número de *octaves* e número de vértices da malha, é possível fazer, em trabalhos futuros, uma geração dos terrenos com um número de *octaves* variando de acordo com a distância da câmera.

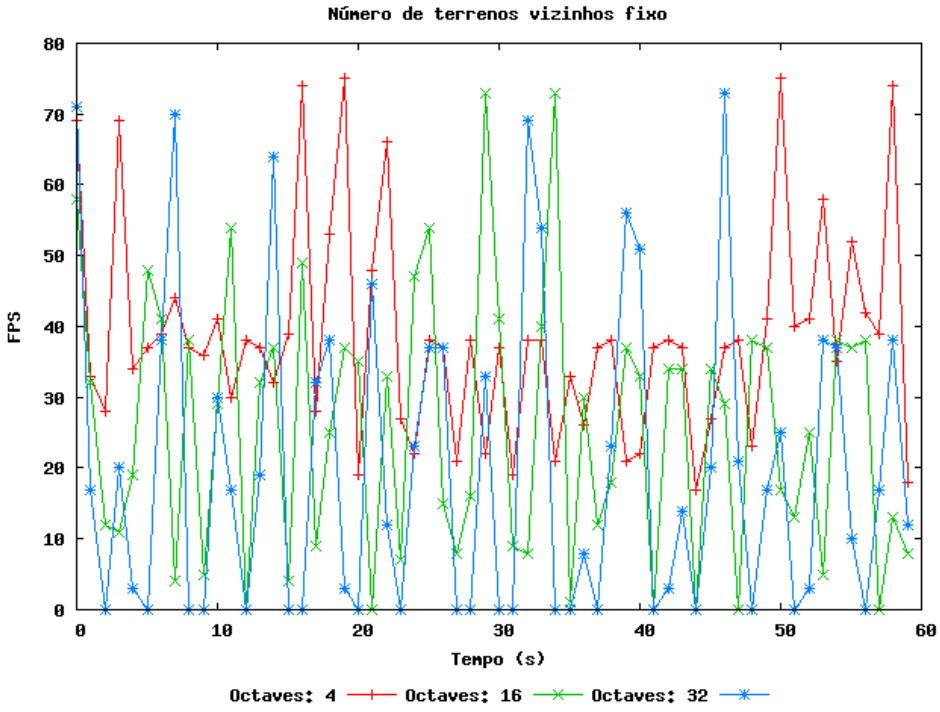

Figura 4.15: Teste variando o número de *octaves*, e o número de terrenos vizinhos fixo em 2.

-	*Frames* por segundo		
Octaves	Mín.	Máx.	Média
4	17,0	75,0	38,5
16	0	73,0	25,6
32	0	73,0	19,9

Tabela 4.4: FPS das execuções variando o número de *octaves*, e o número de terrenos vizinhos fixo em 2.

4.4.3 Teste 3

O próximo teste (Tabela 4.5 e Figura 4.16) mostra o impacto variando o número de terrenos vizinhos. Como era de se esperar, quanto maior o número de vizinhos, menor o FPS. Para melhorar isso, é possível ajustar o número de vizinhos desenhados de acordo com o computador.

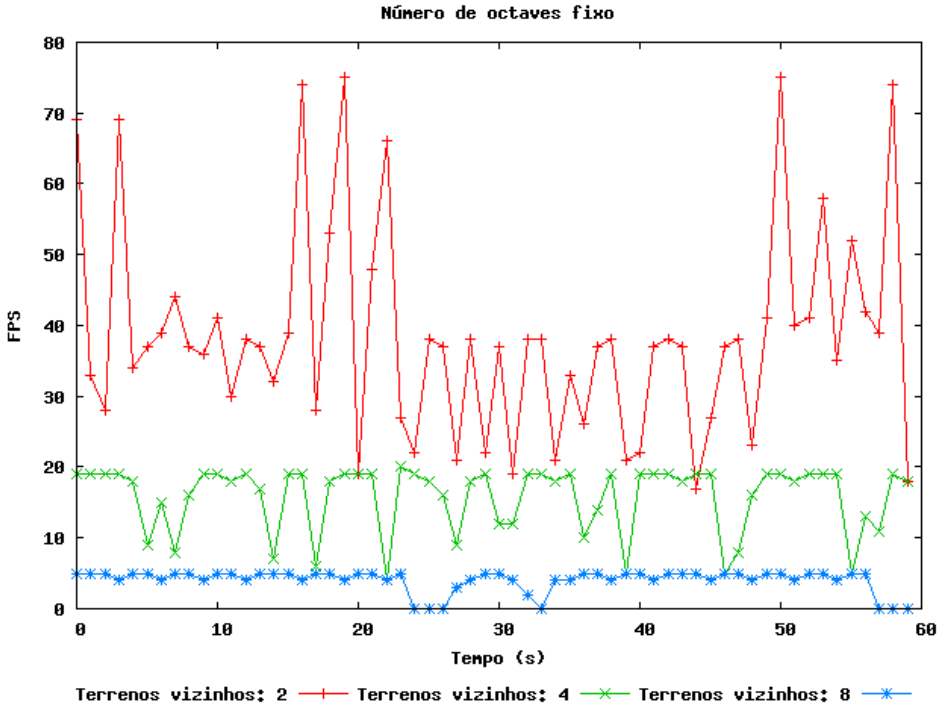

Figura 4.16: Teste variando o número de terrenos vizinhos, e o número de octaves fixo.

-	*Frames* por segundo		
Vizinhos	Mín.	Máx.	Média
2	17,0	75,0	38,5
4	4,0	20,0	15,8
8	0	5,0	4,1

Tabela 4.5: FPS das execuções variando o número de terrenos vizinhos, e o número de octaves fixo.

5 CONCLUSÕES E TRABALHOS FUTUROS

O trabalho desenvolvido em POC I teve como objetivo principal estabelecer uma base para ser desenvolvida posteriormente em POC II, e também como uma forma de aprendizado de técnicas procedurais.

Um ponto a ser otimizado é a cópia dos vértices para as estruturas VBO. Dessa forma, será possível diminuir o tempo total gasto com a geração procedural dos terrenos, e suavizar as transições entre terrenos. Outro ponto que pode ser abordado é a visualização dos terrenos na medida em que eles são gerados; no caso do ruído de Perlin, a visualização poderia ser a cada *octave* gerado. Terrenos mais longe da câmera poderiam também ser gerados com um número menor de *octaves*.

Outro ponto que também pode ser explorado é processamento em paralelo, um dos temas abordados no paradigma apresentado em [20]. O trabalho também poderá ser migrado para a *engine PsyGen*, utilizado no artigo anterior.

Um ponto que só foi iniciado em POC I é a geração de terrenos esféricos e deve ser abordado em POC II. Além disso, o uso de texturas também será aprofundado, possivelmente com o uso de sombras pré-calculadas [21], pois atualmente, os terrenos podem ser facilmente confundidos com nuvens, água parada, ou qualquer outro fenômeno natural. Outra questão a ser desenvolvida em POC II é uma *interface* gráfica mais atrativa, e uma forma eficiente de se armazenar as informações inseridas pelo usuário (possivelmente em arquivos XML).

Referências Bibliográficas

1 PROCEDURAL content generation. Disponível em: <http://lukehalliwell.wordpress.com-/2008/08/05/procedural-content-generation/>. Acesso em: 23 nov. 2008.

2 PIXAR Animation Studios. Disponível em: <http://www.pixar.com/>. Acesso em: 23 nov. 2008.

3 IAN Bell's Elite pages. Disponível em: <http://www.iancgbell.clara.net/elite/>. Acesso em: 23 nov. 2008.

4 PARISH, Y. I. H.; MüLLER, P. Procedural modelling of cities. In: *in Proc. ACM SIGGRAPH, (Los Angeles, 2001) ACM Press*. [S.l.: s.n.], 2001. p. 301–308.

5 GREUTER, S. et al. Real-time procedural generation of 'pseudo infinite' cities. In: *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA: ACM, 2003. p. 87–ff. ISBN 1-58113-578-5.

6 OLSEN, J. Realtime procedural terrain generation. In: *Department of Mathematics And Computer Science (IMADA).* [S.l.: s.n.], 2004.

7 ZIMMERLI, L.; VERSCHURE, P. Delivering environmental presence through procedural virtual environments. In: *PRESENCE 2007, The 10th Annual International Workshop on Presence*. [S.l.: s.n.], 2007.

8 MOJOWORLD Generator. Disponível em: <http://www.mojoworld.org/>. Acesso em: 23 nov. 2008.

9 SPEEDTREE | IDV, Inc. Disponível em: <http://www.speedtree.com/>. Acesso em: 23 nov. 2008.

10 EBERT, D. S. et al. *Texturing and Modeling: A Procedural Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 1558608486.

11 PRUSINKIEWICZ, P.; LINDENMAYER, A. *The algorithmic beauty of plants*. New York, NY, USA: Springer-Verlag New York, Inc., 1996. ISBN 0-387-94676-4.

12 MANDELBROT, B. B. *The Fractal Geometry of Nature*. W. H. Freeman, 1982. Hardcover. ISBN 0716711869. Disponível em: <http://www.amazon.ca/exec/obidos-/redirect?tag=citeulike09-20\&path=ASIN/0716711869>.

13 KEN Perlin's homepage. Disponível em: <http://mrl.nyu.edu/~perlin/>. Acesso em: 23 nov. 2008.

14 ACMC Projects, CG Rendering of Coral at The University of Queensland. Disponível em: <http://www.acmc.uq.edu.au/Projects/CG_Rendering.html>. Acesso em: 23 nov. 2008.

15 GLFW - An OpenGL Framework. Disponível em: <http://glfw.sourceforge.net/>. Acesso em: 23 nov. 2008.

16 ANTTWEAKBAR GUI library to tweak parameters of OpenGL and DirectX applications. Disponível em: <http://www.antisphere.com/Wiki/tools:anttweakbar>. Acesso em: 23 nov. 2008.

17 DEVIL - A full featured cross-platform Image Library. Disponível em: <http://openil-.sourceforge.net/>. Acesso em: 23 nov. 2008.

18 OBJECT Oriented Framework Development. Disponível em: <http://www.acm.org-/crossroads/xrds7-4/frameworks.html>. Acesso em: 23 nov. 2008.

19 KELLY, G.; MCCABE, H. Citygen: An interactive system for procedural city generation. In: *Game Design & Technology Workshop*. [S.l.: s.n.], 2006.

20 CORDEIRO, C. S.; CHAIMOWICZ, L. Parallel lazy amplification. *VII Brazilian Symposium on Computer Games and Digital Entertainment, 2008, Belo Horizonte. SBGames 2008*, 2008.

21 GAMEDEV.NET - 'Slope Lighting' Terrain. Disponível em: <http://www.gamedev.net-/reference/articles% -/article1436.asp>. Acesso em: 23 nov. 2008.