

**UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA**

**FERRAMENTA PARA GERAÇÃO DE
MUNDOS VIRTUAIS
PSEUDO-INFINITOS PARA JOGOS 3D
MMO**

DISSERTAÇÃO DE MESTRADO

Fernando Bevilacqua

Santa Maria, RS, Brasil

2008

**FERRAMENTA PARA GERAÇÃO DE MUNDOS
VIRTUAIS PSEUDO-INFINITOS PARA JOGOS 3D
MMO**

por

Fernando Bevilacqua

Dissertação apresentada ao Programa de Pós-Graduação em Informática
da Universidade Federal de Santa Maria (UFSM, RS), como requisito
parcial para a obtenção do grau de

Mestre em Informática

Orientador: Prof. Dr. Cesar Tadeu Pozzer (UFSM)

**Dissertação de Mestrado Nº 2
Santa Maria, RS, Brasil**

2008

**Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática**

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

**FERRAMENTA PARA GERAÇÃO DE MUNDOS VIRTUAIS
PSEUDO-INFINTOS PARA JOGOS 3D MMO**

elaborada por
Fernando Bevilacqua

como requisito parcial para obtenção do grau de
Mestre em Informática

COMISSÃO EXAMINADORA:

(Presidente/Co-orientador)

Prof. Dr. Gerson Geraldo Homrich Cavalheiro (UFPel)

Prof^a Dr^a Iara Augustin (UFSM)

Santa Maria, 22 de Agosto de 2008.

“Não sabendo que era impossível, ele foi lá e fez.” — JEAN COCTEAU

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

FERRAMENTA PARA GERAÇÃO DE MUNDOS VIRTUAIS PSEUDO-INFINITOS PARA JOGOS 3D MMO

Autor: Fernando Bevilacqua
Orientador: Prof. Dr. Cesar Tadeu Pozzer (UFSM)
Local e data da defesa: Santa Maria, 22 de Agosto de 2008.

Resumo em português [aqui](#).

Palavras-chave: MMO, mundos virtuais, geração de terreno, jogos 3D.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

VXDL: A LANGUAGE FOR INTERCONNECTION AND RESOURCES SPECIFICATION IN VIRTUAL GRIDS

Author: Fernando Bevilacqua
Advisor: Prof. Dr. Cesar Tadeu Pozzer (UFSM)

Grid computing has been defined as an infrastructure integrator of distributed resources. Although it is already used on a large scale in many areas, this type of computational infrastructure is still an area of active research, with many open questions. Today, new research works investigate the application of resources virtualization techniques to perform the composition of virtual grids. These grids can be defined as a high level abstraction of resources (computing and network), through which users have a view of a wide range of interconnected computers, that can be selected and virtually organized. In a virtual grid, as well in a real grid, users and middleware must have tools that allow the composition and management of the infrastructure. Among these tools, there are languages for resource description that allow the specification of components that will be used in the infrastructure. In a virtualized environment, the resources descriptions languages should offer attributes that interact with some peculiarities, such as the possibility of allocate multiple virtual resources (computing and network) on the same physical resource. In this context, this work presents VXDL, a language developed for the interconnections and resources description in virtual grids. The innovations proposed in VXDL allow the description, classification and parameter specification of all desirable components, including network topology and virtual routers. VXDL also allow the specification of a execution timeline, which can assist grid middleware in the tasks of resources sharing and scheduling. To evaluate the proposed language, this work presents I) a comparative study between VXDL and other resources description languages and II) an analysis of results obtained with the benchmarks execution in virtual infrastructures composed using different VXDL descriptions.

Keywords: virtualization, virtual grids, virtual clusters, resources description language.

LISTA DE FIGURAS

Figura 2.1 – Quatro primeiras iterações do fractal de floco-de-neve de Koch [1] ...	13
Figura 2.2 – À direita, valores aleatórios gerados a partir de pontos discretos. À esquerda, interpolação suave dos pontos [6]	13
Figura 2.3 – À direita, valores aleatórios gerados a partir de pontos discretos. À esquerda, interpolação suave dos pontos [6]	14
Figura 2.4 – À esquerda, <i>heightmap</i> gerado a partir de divisões estocásticas; à direita, o mesmo <i>heightmap</i> renderizado em 3D [1]	16
Figura 2.5 – Geração de relevo pela disposição do ponto médio (perspectiva 2D) [1]	17
Figura 2.6 – À direita, <i>heightmap</i> gerado a partir de ruído de Perlin. À esquerda, <i>heightmap</i> gerado a partir da combinação de camadas de ruído de Perlin (turbulência). [2]	18
Figura 2.7 – À esquerda, <i>heightmap</i> gerado a partir da técnica de ruído de Perlin <i>ridged</i> ; à direita, o mesmo <i>heightmap</i> renderizado em 3D [2]	18
Figura 2.8 – Mundo virtual dividido em células. Cada uma delas possui uma semente calculada com base em sua posição e com base uma semente global referente à cidade [14]	19
Figura 2.9 – Geração de construções: cada andar é gerado pela mescla de polígonos escolhidos e centralizados aleatoriamente [14]	20
Figura 2.10 – Campo de visão do usuário: apenas as células visíveis tem o seu conteúdo gerado [14].....	20
Figura 2.11 – Exemplo de cidade virtual gerada [14]	21
Figura 2.12 – Terreno texturizado com duas camadas: bloco escuro e um conjunto de texturas em formato de borda com transparência [2]	22
Figura 2.13 – Diferentes relevos gerados pela utilização de fractais e suas combinações. [3]	23
Figura 2.14 – Planetas gerados com os diversos algorítimos. (a) Falhas aleatórias. (b) Disposição do ponto médio. (c) Disposição do ponto médio multifractal. (d) Ruído de Perlin. (e) Ruído de Persil Multifractal. (f) Ruído de Perlin <i>Ridged</i> . (g) Ruído de Perlin <i>Ridged Multifractal</i> . [3]..	25
Figura 2.15 –(a) Mapa de altura gerado a partir de um diagrama de Voronoi. (b) Aplicações de funções de ruído em (a). (c) Mapa resultante da aplicação de um filtro de perturbação em (b). [4]	26
Figura 2.16 –À direita, renderização de um mapa de altura sem a aplicação de erosão; à esquerda, o mesmo mapa com a aplicação de erosão. [4] ...	27
Figura 2.17 – <i>Quad-tree</i> utilizada para a geração do terreno [1]	28
Figura 2.18 –Subdivisão de células e níveis de detalhe representados por uma árvore binária em 2D. [1]	29

Figura 3.1 – Problema da geração de conteúdo sob demanda.	30
Figura 3.2 – Organização do sistema de coordenadas do mundo virtual	32
Figura 3.3 – Campo de visão do usuário: a área visualizada corresponde a uma fatia do mundo virtual existente.	33
Figura 3.4 – Mapeamento das coordenadas do mundo virtual para as coordenadas de desenho da tela.....	34
Figura 3.5 – Geração de conteúdo com base apenas as informações do campo de visão	36
Figura 3.6 – Função de geração de relevo parametrizável e o resultado gerado	37

LISTA DE TABELAS

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Contexto e Motivação	11
1.2	Objetivos e Contribuição	11
1.3	Organização do Texto	11
2	REVISÃO DE LITERATURA	12
2.1	Funções de ruído e fractais	12
2.1.1	Fractais	12
2.1.2	Ruído de Perlin	13
2.2	Geração de relevo	15
2.2.1	Divisões estocásticas	15
2.2.2	Falhas geológicas	16
2.2.3	Deposição de sedimentos	17
2.2.4	Disposição do ponto médio	17
2.2.5	Ruído de Perlin	17
2.3	Mundos pseudo-infinitos	19
3	IMPLEMENTAÇÃO	30
3.1	Terreno infinito	31
3.1.1	Renderização de conteúdo	34
3.2	Continentes	35
3.3	Relevo	35
4	RESULTADOS	38
5	CONCLUSÃO E TRABALHOS FUTUROS	39
	REFERÊNCIAS	40

1 INTRODUÇÃO

1.1 Contexto e Motivação

1.2 Objetivos e Contribuição

1.3 Organização do Texto

2 REVISÃO DE LITERATURA

2.1 Funções de ruído e fractais

As formas encontradas na natureza são geralmente complexas e ricas em detalhes. Uma cadeia montanhosa, por exemplo, apresenta variações de altitude que às vezes não pode ser representada por uma função matemática, visto que ela não possui um padrão definido (ela é completamente aleatória). Em contra-partida, certas elementos naturais apresentam um padrão bem definido, como as cores na concha de um molusco ou as camadas de cascas de uma árvore.

Para simular elementos naturais e suas peculiaridades em computação gráfica, pode-se utilizar fractais e funções de ruído. Fractais simulam elementos que possuem um padrão definido, enquanto funções de ruído adicionam a aleatoriedade existente no mundo real. Essa seção explana sobre conceitos e técnicas relacionadas a fractais e funções de ruído.

2.1.1 Fractais

Fractais são formas matemáticas geradas recursivamente, sendo que a cada nível de recursão mais e mais detalhes da mesma forma são gerados [7]. As formas geradas são auto-similares e geralmente a cada nível de recursão a forma original é vista novamente, com algumas alterações. Fractais possuem um nível de detalhes infinito, então quando mais próximo ele é visualizado, mais detalhes ele irá gerar. A figura 2.1 ilustra o funcionamento de um fractal. É possível que fractais sejam utilizados para a geração de diversos elementos, tanto terrenos como plantas [8]. A utilização de fractais para a geração de relevo é uma técnica comum em computação gráfica, como pode ser visto na seção 2.2.

Conforme já citado, fractais apresentam um padrão de auto-similaridade, o que garante uma certa homogeneidade. As formas encontradas na natureza, porém, são em sua grande

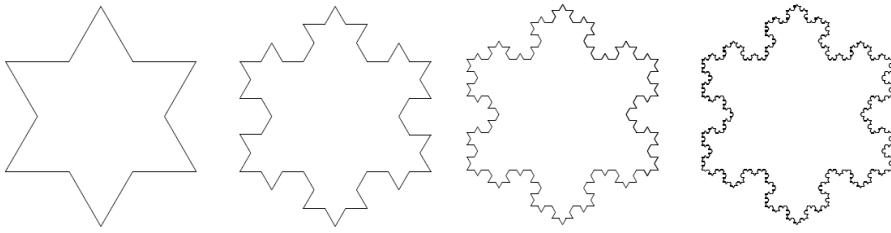


Figura 2.1: Quatro primeiras iterações do fractal de floco-de-neve de Koch [1]

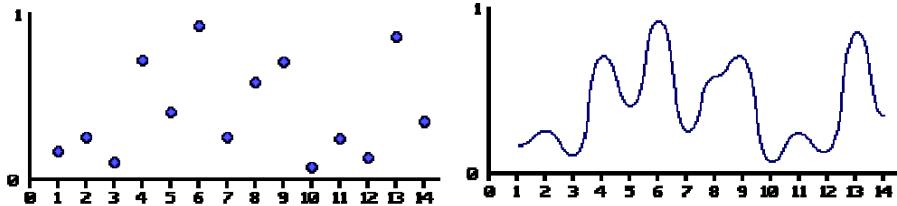


Figura 2.2: À direita, valores aleatórios gerados a partir de pontos discretos. À esquerda, interpolação suave dos pontos [6]

maioria heterogêneas, como a forma de um montanha: quando mais próximo do cume, mais pontiagudas as formas se tornam, ao passo que na base da montanha as formas são suaves e tentem a ser mais planas. Aplicando o conceito de heterogeneidade à fractais obtém-se o que é denominado *multifractal*. Em um conceito simples, multifractais são fractais com dimensões e detalhes diferenciados ao longo de sua forma e eles são obtidos fazendo com que o seu detalhamento seja calculado em função de algum outro atributo. Considerando o exemplo da montanha já citada, o nível de suavização da forma pode ser dado em função da altitude da montanha, ou seja, quanto mais próximo do cume, menos suave as formas devem ser.

2.1.2 Ruído de Perlin

O ruído de Perlin foi desenvolvido por Ken Perlin in 1982, cujo objetivo inicial era criar texturas com aparência mais natural. A função de ruído de Perlin gera um intervalo de valores que podem ser utilizados para diversos fins, dentre eles uma forma de adicionar aleatoriedade aos relevos gerados proceduralmente. Os resultados da função de ruído de Perlin são baseados no somatório dos valores de diversas funções redimensionadas, todas elas originadas de uma única função base. Essa função base é obtida a partir da interpolação de valores aleatórios gerados de forma discreta. A figura 2.2 ilustra os pontos gerados e a sua interpolação.

Partindo de um conjunto de pontos discretamente coletados, um espectro de valores

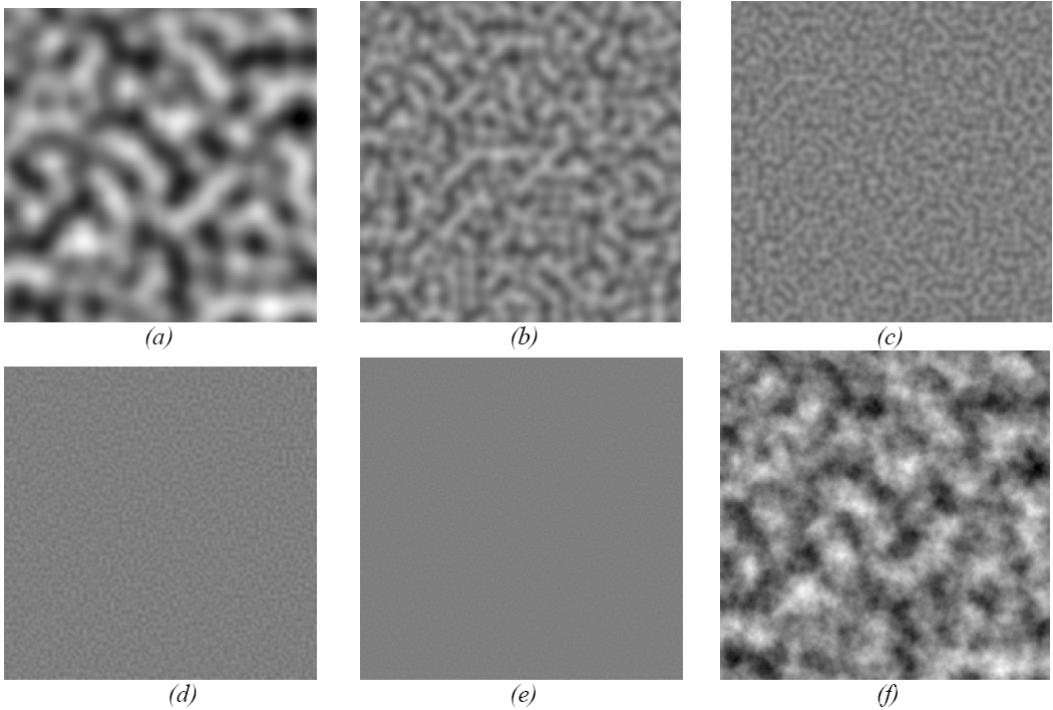


Figura 2.3: À direita, valores aleatórios gerados a partir de pontos discretos. À esquerda, interpolação suave dos pontos [6]

é criado. Em seguida, todos os pontos são interpolados entre si, criando um domínio de valores infinito. Para cada parâmetros de entrada que a função de ruído receber, um valor correspondente será encontrado na curva que foi gerada como resultado da interpolação. Para facilitar os cálculos de geração de terreno, por exemplo, pode-se condicionar a função de ruído para que os valores retornados estejam sempre contidos dentro do intervalo $[-1; 1]$.

Alterando-se a amplitude e a frequência da função base, obtém-se um conjunto de novas funções em outra escala. A maneira mais comum de obter novas funções em outra escala é dobrando a frequência e reduzindo a amplitude pela metade; cada nova função gerada é chamada de *octava*. O resultado da adição dessas diversas funções (octavas), com diferentes amplitude e frequência, é a função de **ruído de Perlin**. A figura 2.3 ilustra graficamente as funções base utilizadas para a criação da função de ruído; as ilustrações de (a) até (e) são oitavas, sendo (a) a oitava de menor frequência e (e) a oitava de maior frequência. A ilustração (f) mostra o resultado da soma de todas as oitavas, que é a função de ruído de Perlin propriamente dita.

2.2 Geração de relevo

O elemento mais básico que compõe um mundo virtual é o seu terreno. O terreno é o plano que irá guiar e servir como base para todos os demais elementos que compõem o mundo, como árvores, estradas, cidades, etc. Em computação gráfica, um das formas possíveis de se representar um terreno é através de um mapa de altura (*heightmap*). Nessa abordagem, o terreno é representado por uma matriz de pontos, sendo que cada ponto (i, j) representa a altura de um ponto do plano. Iterando-se sobre os pontos dessa matriz, conectando cada um deles através de arestas até eles formarem triângulos, é possível renderizar uma malha que pode ser utilizada como um terreno virtual.

Essa seção apresenta diversas técnicas que podem ser utilizadas para a geração de um *heightmap* que, quando renderizado, é capaz de produzir relevos muito próximos aos que são encontrados no mundo real.

2.2.1 Divisões estocásticas

Essa técnica consiste na subdivisão recursiva do *heightmap* através da utilização de números pseudo-aleatórios. Partindo de uma matriz com todos os pontos tendo a altura zero, o funcionamento do algorítimo é o seguinte:

- Uma altura escolhida aleatoriamente é atribuída a cada um dos cantos do quadrado. Essa altura é proporcional ao tamanho do quadrado.
- O quadrado é dividido em quatro quadrados menores, com as alturas dos cantos de cada quadrado sendo calculada com base na interpolação das alturas dos cantos dos quadrados vizinhos ao quadrado original.
- O algorítimo repete os dois primeiros passos para cada quadrado gerado. A recursão é quebrada quando um certo nível de detalhes é atingido.

O resultado obtido com esse algorítimo é um plano que simula as elevações montanhosas do mundo real. Embora essa abordagem seja eficiente, ela não produz resultado muito realísticos, visto que a variação de altura ao longo dos pontos é linear, o que não é comum na natureza. Além disso, é possível encontrar picos ou bordas muito pontiagudos ao longo dos cantos dos quadrados gerados, o que gera um relevo irrele [9]. A figura 2.4 ilustra um relevo gerado a partir de divisões estocásticas.

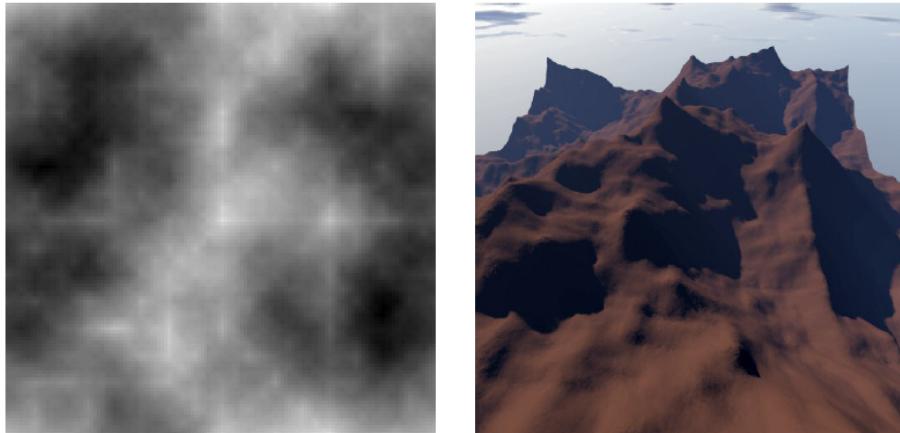


Figura 2.4: À esquerda, *heightmap* gerado a partir de divisões estocásticas; à direita, o mesmo *heightmap* renderizado em 3D [1]

Existem variações desse algorítimo que visam contornar as imperfeições descritas anteriormente. Uma dessas variações é a subdivisão em forma de diamante, que consiste em rotacionar os novos quadrados gerados em 45° em relação ao quadrado original. Outra variação é afastar os quadrados novos dos cantos altos do quadrado original, recalculando as alturas dos cantos de cada um dos novos quadrados com base em pesos, o que produz um relevo mais suave e sem picos pontiagudos.

2.2.2 Falhas geológicas

A geração por falhas geológicas simula a criação de relevo por movimentação de placas tectônicas. Partindo de um terreno com todos os pontos zerados, o algorítimo escolhe aleatoriamente uma linha que divide o terreno em duas partes. Para um dos lados, a altura dos pontos é incrementada em uma certa quantidade, enquanto a altura dos pontos do outro lado é decrementada dessa mesma quantidade. Depois disso, o valor usado para aumentar/diminuir as alturas é reduzido em uma certa quantidade e o algorítimo é repetido. No momento que esse valor chegar a zero o algoritmo termina.

O relevo gerado com esse método não apresenta elevações pontiagudas como no algorítimo de subdivisões estocásticas, porém ele é muito lento para ser utilizado. Depois que a linha divisória é escolhida, todos os pontos da matriz precisam ser atualizados e isso é feito a cada iteração do algorítimo.

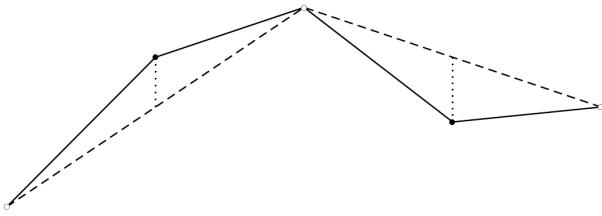


Figura 2.5: Geração de relevo pela disposição do ponto médio (perspectiva 2D) [1]

2.2.3 Deposição de sedimentos

A geração por deposição de sedimentos simula a criação de relevo por fluxos de lava. Nessa abordagem, pontos de liberação de sedimentos são escolhidos e, em cada um desses pontos, um determinado número de sedimentos é depositado. Cada partícula depositada irá tentar chegar até o ponto mais baixo da região onde foi depositada, rolando por cima das demais partículas. Se a partícula cair em um local no qual todas as posições à sua volta tem a mesma altura, então a partícula para de se movimentar.

Variando-se os pontos de deposição de partículas e a quantidade de elementos depositados em cada ponto, é possível gerar obter-se terrenos visualmente aceitáveis. Novamente o tempo de execução do algorítimo é um problema; para cada partícula depositada, faz-se necessário o cálculo de movimentação desse elemento, o que, dependendo do número de iterações, pode ter um custo computacional grande. Além disso, o algorítimo apresenta uma complexidade de implementação maior que os demais já citados.

2.2.4 Disposição do ponto médio

Essa técnica consiste em perturbar o ponto médio de um determinado segmento. Em uma abordagem 2D, por exemplo, dado um determinado segmento, o algorítimo acharia o ponto médio desse segmento e, utilizando um número pseudo-aleatório, aumentaria ou diminuiria a altura desse ponto em uma determinada quantidade. Em seguida, essa quantidade utilizada para perturbação é reduzida e o algorítimo é aplicado novamente aos dois novos segmentos gerados a partir da perturbação do ponto médio. A figura 2.5 ilustra o funcionamento do algorítimo.

2.2.5 Ruído de Perlin

Essa técnica consiste em utilizar funções de ruído de Perlin para causar perturbações no *heightmap*. A utilização de uma função de ruído de Perlin num *heightmap* não produz, por si só, um relevo visualmente parecido com aqueles encontrados na natureza; a com-

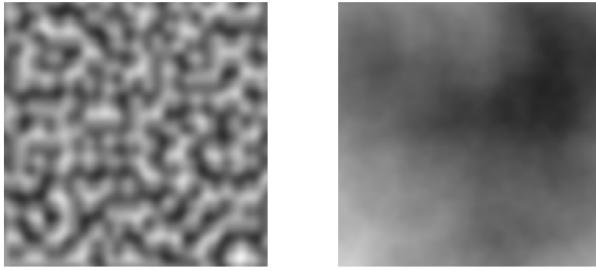


Figura 2.6: À direita, *heightmap* gerado a partir de ruído de Perlin. À esquerda, *heightmap* gerado a partir da combinação de camadas de ruído de Perlin (turbulência). [2]

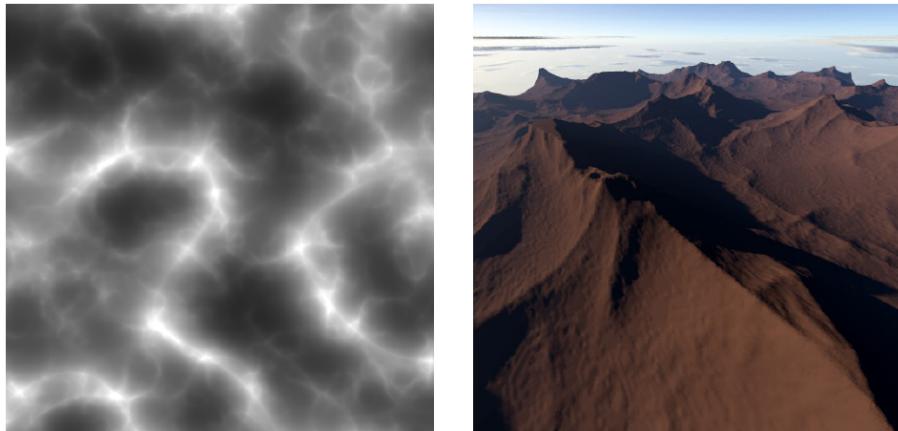


Figura 2.7: À esquerda, *heightmap* gerado a partir da técnica de ruído de Perlin *ridged*; à direita, o mesmo *heightmap* renderizado em 3D [2]

binação de diversas camadas de ruído, porém, é capaz de produzir um efeito mais natural e convincente. Cada uma dessas camadas, chamadas octavas, possui uma amplitude e uma frequência, e a combinação delas é comumente chamada de turbulência de Perlin. A figura 2.6

As características do relevo podem ser ajustadas através da modificação do número de octavas e da frequência de cada uma delas. Quanto maior for a frequência da octava adicionada, maior será a quantidade de detalhes do relevo. Uma variação para a geração de relevo através de ruído de Perlin é a técnica conhecida como ruido de Perlin *ridged*, que consiste na utilização de funções com valores absolutos em conjunto com a função de ruído original para produzir um relevo com mais "cristas". Na abordagem original, no intervalo $[-1, 1]$, que é o domínio da função de ruído, -1 indicaria um local muito baixo no mapa, ao passo que 1 indicaria o cume da montanha mais alta; na versão *ridged*, o algorítmico original é modificado para que os valores -1 e 1 gerem relevos de baixa altitude, ao passo que os valores gerados no meio do intervalo $[-1, 1]$ geram valores de grande altitude. A figura 2.7 ilustra os resultados obtidos com essa abordagem.

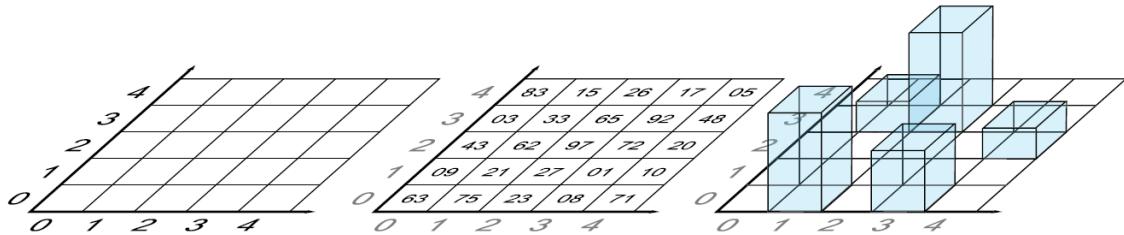


Figura 2.8: Mundo virtual dividido em células. Cada uma delas possui uma semente calculada com base em sua posição e com base uma semente global referente à cidade [14]

2.3 Mundos pseudo-infinitos

A geração de conteúdo procedimentais é um assunto antigo no campo da computação gráfica. A aplicação desse tipo de técnica na geração de um mundo virtual completo foi utilizada por [14], cujo objetivo era gerar uma cidade virtual que fosse visualmente interessante e composta por construções complexas, porém cada uma sendo criada a partir de elementos mais simples.

Na abordagem utilizada, os autores dividiram o mundo virtual numa grade composta por diversos quadrados, chamados células. As coordenadas de localização de cada célula, em conjunto com uma semente global, são utilizadas como entrada para uma função de hash [17]. O resultado dessa função é utilizado como semente para um pseudo gerador de números aleatórios e irá definir todas as características das construções que estão dentro da célula. Dessa forma, o conteúdo de uma célula é sempre o mesmo, independente de quanto o usuário caminhe pelo mundo virtual e faça a célula em questão entrar ou sair do seu campo de visão. A figura 2.8 ilustra a divisão do mundo virtual em células.

Cada uma das construções existentes é gerada pela mescla de polígonos simples escolhidos aleatoriamente. Utilizando um processo iterativo, partindo do topo até a base, em cada iteração o polígono escolhido é mesclado com o polígono anterior e, então, mais um nível (andar) é criado; esse processo garante que a construção, ao longo das iterações, cresça em altura e em largura de uma forma realística. Depois que a geometria da construção está pronta, ela é texturizada com janelas, sendo que o tipo da janela é escolhido aleatoriamente. A figura 2.9 ilustra a geração de construções.

Para garantir o uso racional de recursos computacionais, como memória e processamento, os autores utilizaram o conceito nomeado por eles de preenchimento por *view frustum*, que consiste em restringir à geração de conteúdo somente para as células que

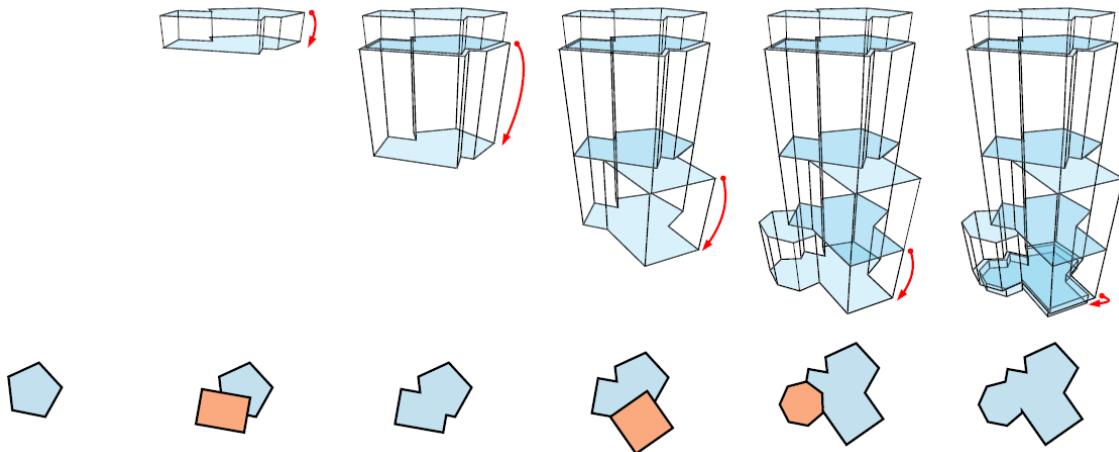


Figura 2.9: Geração de construções: cada andar é gerado pela mescla de polígonos escolhidos e centralizados aleatoriamente [14]

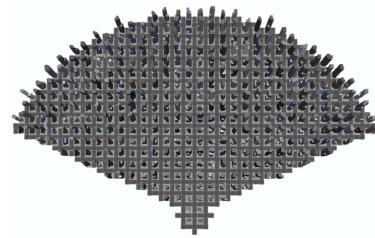


Figura 2.10: Campo de visão do usuário: apenas as células visíveis tem o seu conteúdo gerado [14]

estão dentro do campo de visão do usuário. A medida que esse anda pela cidade virtual, novas células vão sendo adicionadas ao campo de visão e seu conteúdo é gerado; quando a célula sai do campo de visão, ela é removida da memória e seus recursos são liberados. As células são posicionadas em loops quadrados ao redor do usuário e são consideradas pertencentes ao campo de visão se estão a uma certa distância do usuário e dentro de um ângulo de 120° de visão. A figura 2.10 ilustra o funcionamento do campo de visão descrito e a figura 2.11 mostra uma cidade virtual gerada.

Outro trabalho analisado foi o de [2] para a construção da ferramenta SkyCastle [5], uma *engine* para jogos online multijogador com suporte à geração procedural de mundos virtuais. No referido trabalho, o autor foca o problema de mundos virtuais cada vez maiores em jogos de computador e aplicações, o que cria a necessidade de desenvolvimento de ferramentas capazes de ajudar na geração de conteúdos realistas para esses mundos. Para a geração de relevo, o autor utiliza procedimentos parametrizados e sistemas baseados em fractais em uma abordagem de camadas: a aplicação adiciona um mapa de ruído (com amplitude reduzida) ao mapa de altura base em cada iteração.



Figura 2.11: Exemplo de cidade virtual gerada [14]

Os mapas de ruído são pré-computados e criados através de funções de ruído de Perlin, abordagem semelhante ao método de geração de relevo por deposição de sedimentos, que será abordado em mais detalhes na seção 2.2.

Depois do assunto de criação de relevo, o autor aborda a funcionalidade de texturização da malha gerada. Um dos métodos apresentados é a utilização de uma imagem com proporções muito grandes, que seria capaz de cobrir o mundo inteiro. Embora essa abordagem possa ser útil para cenários pequenos, ela não é viável para terrenos grandes ou mundos virtuais, uma vez que o tamanho da imagem poderia atingir proporções proibitivas. Para contornar esse problema, a abordagem de reticulados é sugerida [12]; nessa abordagem, uma célula de textura é criada de tal forma que ao posicionar várias células, uma do lado da outra, o plano gerado apresenta uma texturização contínua e sem falhas. O resultado obtido com essa abordagem é aceitável, porém ele não é visualmente atraente para o usuário final, uma vez que o terreno apresenta uma continuidade que não existiria no mundo real. Para conseguir um resultado visual melhor, o autor sugere a utilização de uma abordagem proposta por [13], que consiste na utilização de um conjunto de texturas pré-definidas em formato de borda juntamente com a texturização em reticulados já citada. O funcionamento do algoritmo resume-se em aplicar as texturas em borda sobre um plano já uniformemente texturizado, porém utilizando transparência nas áreas não desenháveis das texturas em borda. Dessa forma, a sobreposição das bordas sobre o plano texturizado irá produzir uma paisagem menos homogênea, o que gera um resultado visual melhor. A figura 2.12 ilustra as texturas em forma de borda e a sua utilização na

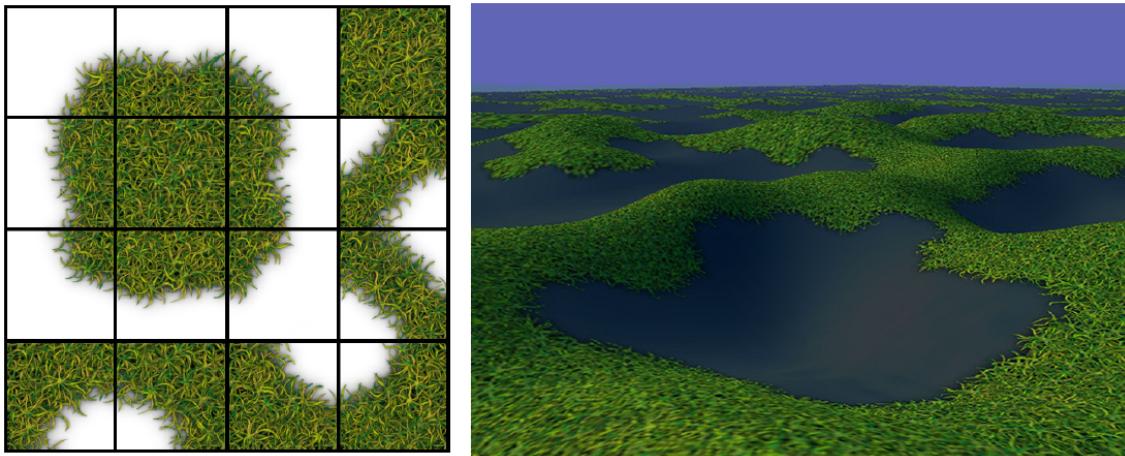


Figura 2.12: Terreno texturizado com duas camadas: bloco escuro e um conjunto de texturas em formato de borda com transparência [2]

abordagem descrita.

Para ornamentar o mundo virtual, o autor cita a utilização de plantas proceduralmente geradas através de três métodos principais: *L-System* [16], geração baseada em componentes [10] e árvores parametrizadas [15]. O algoritmo *L-System* consiste na geração de elementos a partir da interpretação de uma cadeia de caracteres, sendo que cada um desses caracteres representa uma estrutura geométrica ou operação (rotação, translação, etc); a cadeia de caracteres resultante é obtida a partir da aplicação sucessiva de regras sobre uma cadeia base. O algorítimo de geração baseada em componentes consiste na interpretação de uma árvore de elementos, sendo cada um desses elementos modificável através de parâmetros; cada componente pode possuir filhos e, também, uma descrição de qual é o elemento que pode ser usado como folha da árvore. Por fim, a geração de árvores parametrizadas é conceitualmente semelhantes à geração por componentes, exceto que a geração é orientada a ramificações; cada ramificação da árvore corresponde a um nível de recursão, sendo o tronco da árvore o nível zero; quando um ramo sofre uma divisão, os filhos resultantes dessa divisão herdam algumas características do pai (como resolução), porém eles adquirem características próprias, como o ângulo de curvatura; ao final do processo, através de um estudo dos parâmetros corretos a serem utilizados (como a quantidade de ramos, o nível de recursão, etc), é possível que uma árvore completa seja gerada.

Outro trabalho analisado é a geração de planetas procedurais através de fractais e combinações deles com outros métodos [3]. Na abordagem da autora, o mundo gerado

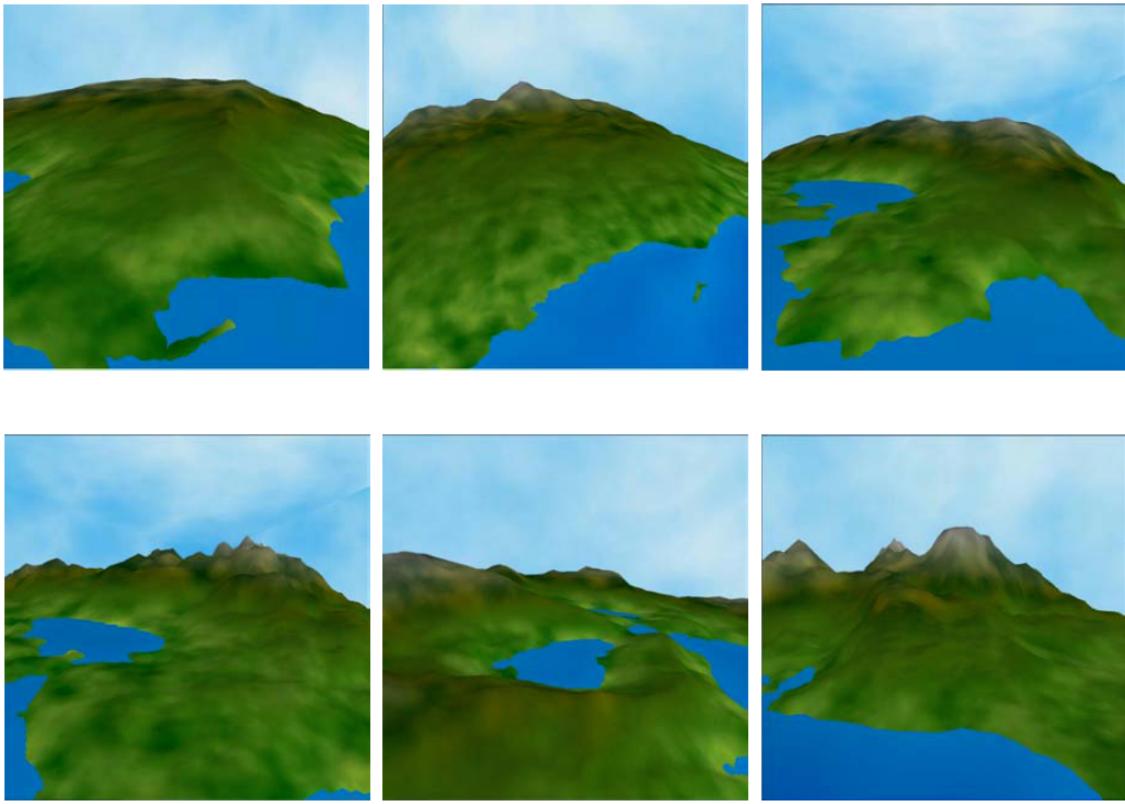


Figura 2.13: Diferentes relevos gerados pela utilização de fractais e suas combinações. [3]

não é infinito, porém ele é esférico e simula a visualização do planeta Terra. Partindo da subdivisão recursiva de um octaedro, a autora cria um mundo esférico que serve como base para a aplicação dos algorítimos de relevo. Através de uma interface, o usuário pode estipular qual algorítimo de relevo ele deseja utilizar, e o resultado desse processo é traduzido em uma lista encadeada que descreve todos os vértices do mundo. Uma lista encadeada foi utilizada para garantir que a malha resultante possa ter seus detalhes aumentados e/ou reduzidos facilmente (através das funções de inserção e remoção de elementos da lista), mesmo que essa estrutura de dados não seja a melhor escolha para busca de um elemento em específico, por exemplo. A utilização de listas encadeadas é importante porque a autora utiliza o conceito de LOD (*level of detail*) para garantir um bom desempenho da aplicação. Através dessa técnica, nos polígonos que estão longe do campo de visão do usuário são mostrados em resolução baixa (poucos triângulos), enquanto os polígonos que estão próximos ao usuário são mostrados com resolução alta (muitos triângulos), o que aumenta a riqueza de detalhes dos objetos próximos. Com a utilização da lista encadeada, a inserção/remoção de novos vértices como consequência da atuação do algorítimo de LOD torna-se uma tarefa menos complexa.

O relevo gerado pela autora é obtido através de diversas técnicas, cada uma com suas peculiaridades e resultados. Dentre as técnicas utilizadas, encontram-se: geração por falhas aleatórias, disposição do ponto médio (incluindo uma variação multifractal) e ruido de Perlin (e suas diversas variações). A figura 2.13 ilustra os relevos obtidos com essas técnicas. Depois que a malha é gerada, a autora utiliza uma combinação de técnicas para colorir os pontos da falha; as técnicas variam conforme o resultado desejado, porém todas elas são baseadas na interpolação de cores parametrizada pela altura do ponto sendo analisado. Para garantir um aspecto mais real, utiliza-se em algumas técnicas conjuntamente com ruídos e turbulência, como o ruído de Perlin.

Tento em vista que autora explora diversos métodos para geração de relevo, é importante salientar quais foram os resultados obtidos com a utilização de cada um deles frente ao problema de geração de continentes para um mundo esférico. Uma visão mostrando todos os planetas gerados é mostrada na figura 2.14. O algorítimo de geração por **falhas aleatórias** produz terrenos que parecem quebrados e diversas ilhas pequenas; de acordo com autora, os resultados obtidos em sua grande maioria são muito aleatórios e na média o algorítimo produz um grande continente e um grande oceano. O algorítimo de **disposição do ponto médio** e sua variação multifractal geram continentes grandes e compactos sem muitas ilhas pequenas, com as deformações geradas sendo uniformes e concentradas no centro dos continentes; embora a variação multifractal do algoritmo produza uma colorização aceitável (tento em vista que há grandes variações de altitude), segundo a autora os resultados não são bons quanto os obtidos com ruído de Perlin. O algorítimo de **ruído de Perlin** e sua variação multifractal apresentam resultados visuais melhores em comparação com as técnicas citadas anteriormente; a comparação entre o planeta gerado apenas com ruído de Perlin e o planeta gerado pela sua variação multifractal mostram a considerável diferença existente entre essas duas abordagens: enquanto a primeira produz relevos com variações de altitude concentradas em determinados lugares, a segunda produz relevos mais realistas com uma distribuição menos concentrada em determinados pontos. Por último, as variações *ridged* da geração por **ruído de Perlin** produzem planetas com continentes estreitos e arredondados, algumas vezes gerando baías fechadas; da mesma forma que os resultados anteriores mostraram, uma abordagem multifractal gera um relevo com variações de altitude mais distribuídas ao longo dos continentes.

Outro trabalho analisado foi a geração de terrenos procedimentais em tempo real com

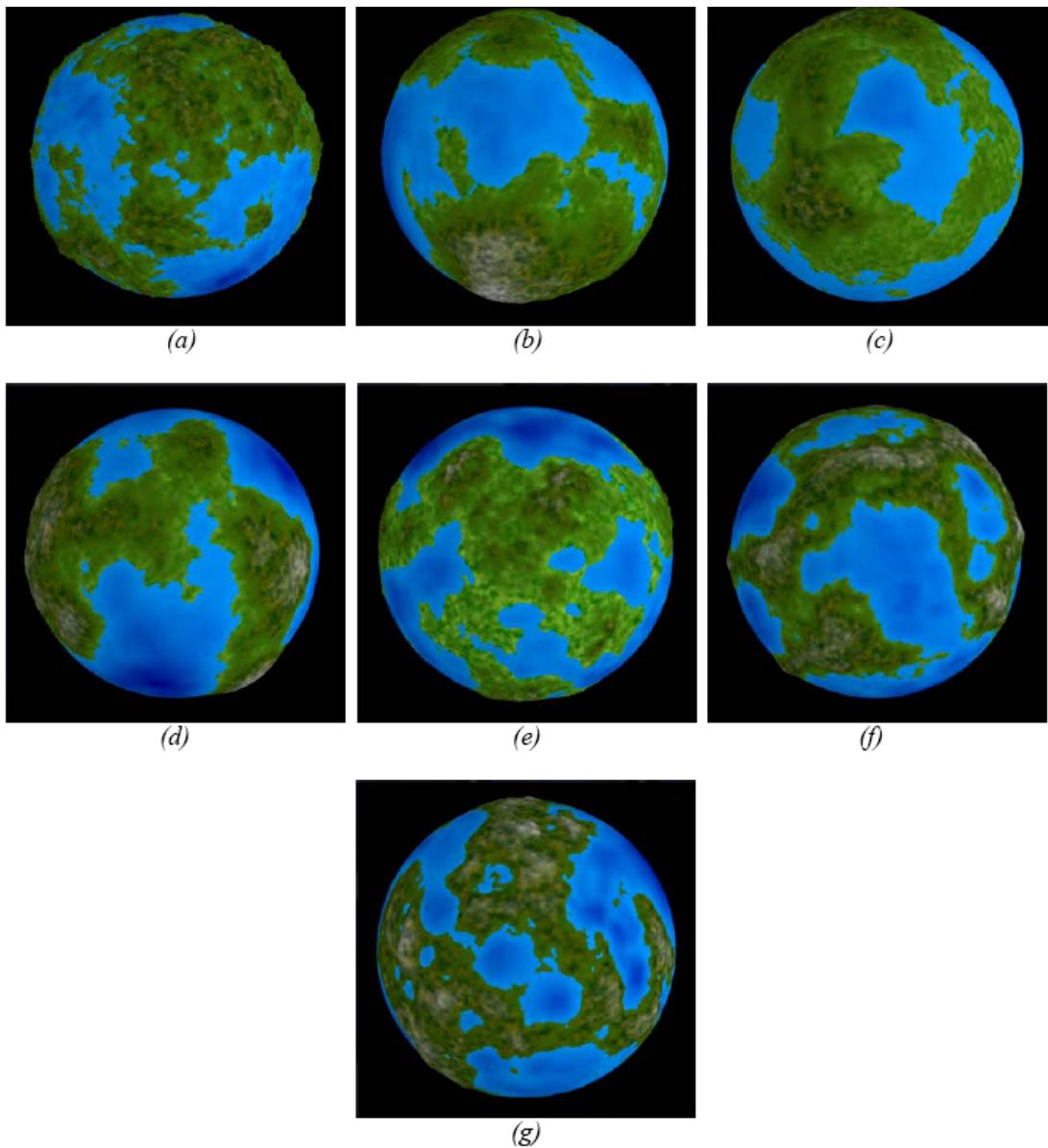


Figura 2.14: Planetas gerados com os diversos algorítimos. (a) Falhas aleatórias. (b) Disposição do ponto médio. (c) Disposição do ponto médio multifractal. (d) Ruído de Perlin. (e) Ruído de Persil Multifractal. (f) Ruído de Perlin *Ridged*. (g) Ruído de Perlin *Ridged Multifractal*. [3]

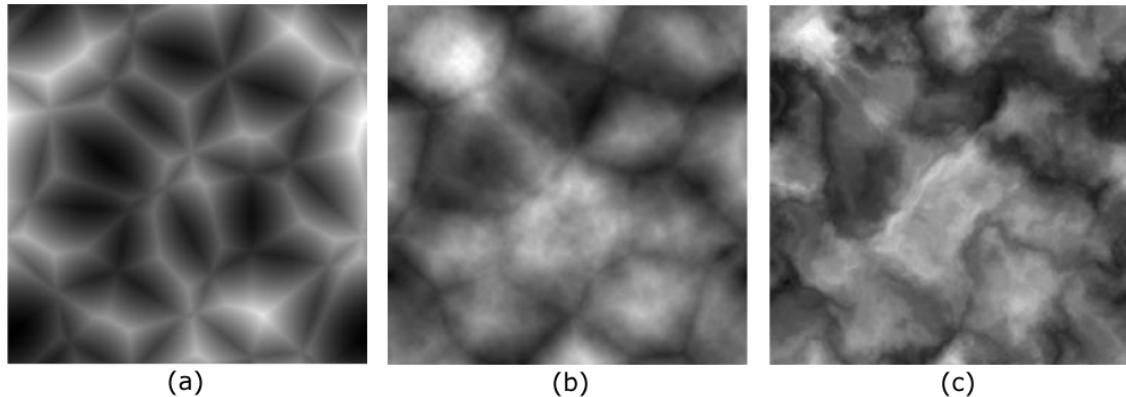


Figura 2.15: (a) Mapa de altura gerado a partir de um diagrama de Voronoi. (b) Aplicações de funções de ruído em (a). (c) Mapa resultante da aplicação de um filtro de perturbação em (b). [4]

a utilização de fractais afetados por erosão [4]. Conforme ressaltado pelo autor, o aumento do poder de processamento de computadores domésticos possibilitou que jogos de computador façam uso de simulações de erosão em tempo quase real. Utilizando essa premissa, o autor agregou à geração de relevo por fractais novas características decorrentes da aplicação de erosão, o que teve como resultado mapas mais reais e, ao mesmo tempo, customizáveis. Primeiramente, um mapa de altura foi criado a partir de um diagrama de Voronoi, sendo cada um dos vértices do diagrama chamados de *pontos de funcionalidade*; o valor de cada célula no mapa de altura é obtido através de uma combinação linear das distâncias entre os pontos de funcionalidade mais próximos. Depois que o mapa de altura é gerado, ele é combinado com um algorítimo de geração de ruído para produzir montanhas menos pontiagudas e, para remover as linhas bem definidas criadas pelo diagrama de Voronoi, um filtro de perturbação ainda é aplicado. A figura 2.15 ilustra o processo descrito.

Para a simulação de erosão no mapa de altura gerado, o autor utiliza um método híbrido de duas técnicas de erosão: a termal e a hidráulica. A erosão termal consiste na simulação de sedimentos se desprendendo de áreas mais altas e deslizando para baixo, na base do relevo. Aplicando uma série de otimizações sobre o algorítimo original proposto por [11], o autor obteve resultados satisfatórios que podem ser utilizados em uma abordagem em tempo real para jogos. A erosão hidráulica simula a deposição de sedimentos causada pelo transporte de materiais dissolvidos num fluxo de água corrente. Em uma comparação entre os dois métodos, a erosão hidráulica apresentou melhores resultados visuais, porém mesmo a sua implementação otimizada não foi mais rápida que a erosão

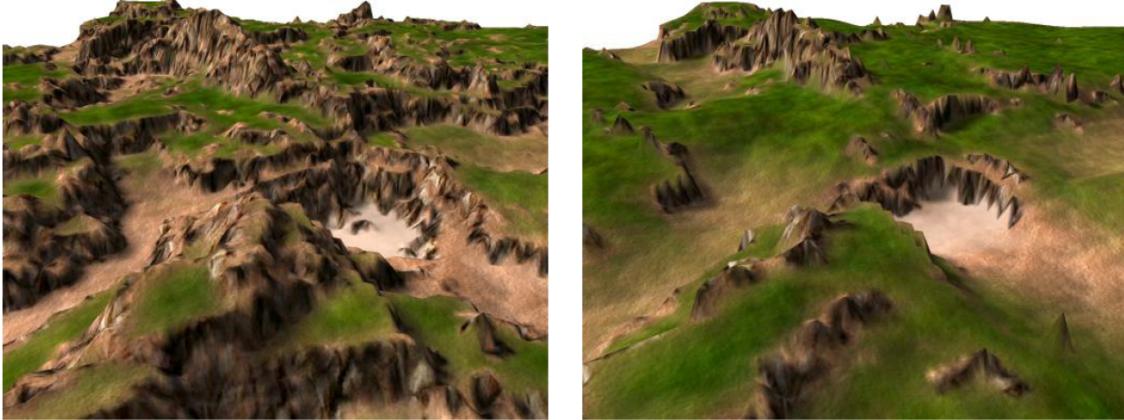


Figura 2.16: À direita, renderização de um mapa de altura sem a aplicação de erosão; à esquerda, o mesmo mapa com a aplicação de erosão. [4]

termal otimizada. O autor criou, então, uma mescla entre as duas técnicas, alterando o método de erosão hidráulica para que ele ficasse mais próximo ao método de erosão termal. O resultado obtido é uma técnica capaz de gerar resultados visuais tão bons quanto o método da erosão hidráulica, porém com o desempenho do método de erosão termal, o que permite que a técnica seja utilizada em tempo real em jogos. A figura 2.16 ilustra o resultado final obtido pelo autor.

Outro trabalho analisado foi a geração de mundos virtuais proceduralmente através de técnicas de subdivisão estocásticas [1]. O objetivo do autor foi criar um mundo virtual de grandes proporções, porém gerando o seu conteúdo sob-demanda através procedimentos parametrizáveis e com multi-resolução. Para a geração do relevo do terreno, o autor utiliza sub-divisões de uma *quad-tree* em um processo recursivo, como ilustra a figura 2.17. A forma do terreno é definida pelo ponto médio de cada uma das células, que na figura são representados pelos círculos pequenos no centro de cada quadrado. Para cada nível de subdivisão, os novos pontos médios das células filho criadas são definidos em função das nove células pai que estão ao redor da célula sendo dividida. A altura dos demais pontos da malha, como os vértices que estão nos cantos das células, é definida pela interpolação dos pontos médios mais próximos. A cada camada de detalhamento, mais células são subdivididas, e a célula que sofreu a subdivisão é substituída pelas novas células filho geradas. Na figura 2.17, o quadrado localizado no centro representa uma célula que está sofrendo subdivisões e que será substituída pelas quatro células filho resultantes da subdivisão (as células filhos são os quadrados com círculos brancos no centro). Tendo em vista que as subdivisões são baseadas em cálculos estocásticos, o autor

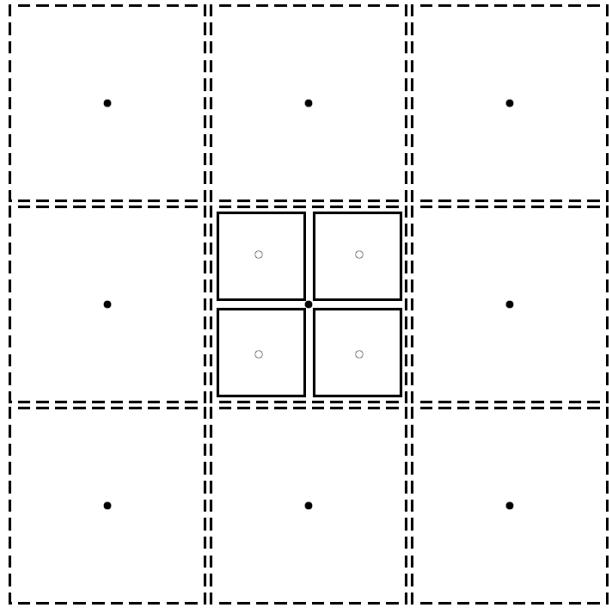


Figura 2.17: *Quad-tree* utilizada para a geração do terreno [1]

necessitou de uma forma de geração de números aleatórios que pudesse prover uma sequência confiável de números ao longo de todo o processo de subdivisão, visto que são vários níveis de detalhes, cada um deles gerando várias células com diversos pontos. Para evitar repetições de números ao longo dos vários níveis de detalhamento, o autor utilizou um gerador de números aleatórios alimentado por três elementos: as coordenadas X e Y da célula e o nível de detalhe sendo mostrado nesse momento; isso garante que um conjunto de números aleatórios seja gerado de forma satisfatória para cada um das células existentes no mundo virtual, independente do nível de detalhes exigido.

Para controlar a instanciação de cada célula, conforme a câmera se movimenta, novas células são instanciadas à frente da câmera e as células atrás da câmera, que estão muito distantes, são removidos. A figura 2.18 ilustra o funcionamento do processo através de uma árvore binária; os nós pai sempre instanciam todos os seus filhos e os nós que estão dentro de um determinada camada (nível da árvore) só serão atualizados quando a câmera atingir o ponto médio da célula pai. Como ilustrado na figura, à medida que a câmera se move para a direita, ultrapassando o ponto médio das células pai, novos nós vão sendo instanciados à direita (à medida que o nó pai entra no campo de visão), ao passo que todos os nós pai que saem do campo de visão são removidos (junto com os seus filhos). A malha final é renderizada a partir das informações conditas nas folhas da árvore, sendo que a altura de cada um desses pontos na malha é definido pela interpolação de seu ponto

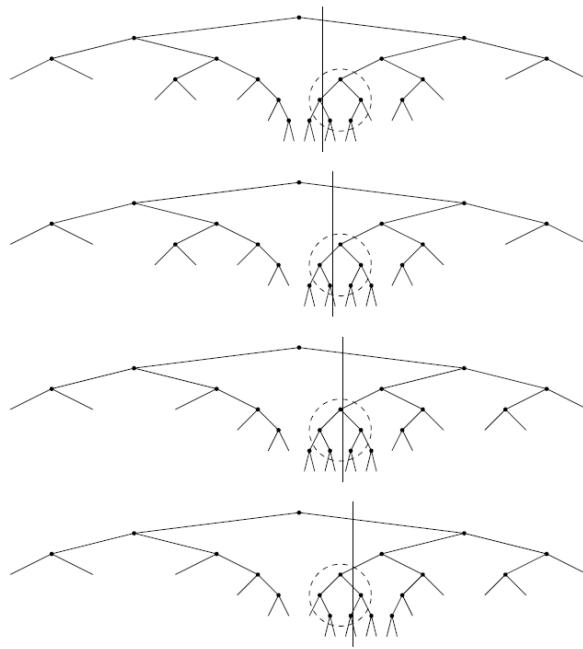


Figura 2.18: Subdivisão de células e níveis de detalhe representados por uma árvore binária em 2D. [1]

médio com a vizinhança, como descrito anteriormente.

Para garantir o desempenho no cálculo dos pontos médios da vizinhança, o autor buscou uma forma de pesquisar, com um tempo constante, os nós vizinhos duma determinada célula. Para encontrar os vizinhos imediatos a uma célula, basta que a relação de pai e filho da árvore seja utilizada, o que irá resultar em uma busca de apenas dois ponteiros (um do nó atual até o pai dele e outro do pai até o nó irmão). O problema está no cálculo para encontrar as células vizinhas mais distantes, que podem estar a uma distância arbitrária do nó atual, ou seja, vários ponteiros deverão ser consultados até que os vizinhos desejados sejam encontrados. Uma das soluções encontradas pelo autor para essa busca é utilizar uma tabela hash para indexar todos os nós de uma determinada camada (nível da árvore). Utilizando essa abordagem, o tempo de busca de qualquer nó de uma camada se tornaria constante. Outra solução encontrada é utilizar um vetor bi-dimensional para indexar os nós de uma determinada camada. Conforme apontado pelo autor, quando uma camada deve ser atualizada (porque o ponto médio de uma célula pai foi atingido), a melhor abordagem é a utilização de um vetor; mesmo que o custo para atualizar todos os nós de um vetor seja mais alto do que apenas inserir uma nova entrada na tabela hash, a busca em vetor é mais rápida do que o processo de calcular a chave de hash de cada célula a partir de suas coordenadas.

3 IMPLEMENTAÇÃO

A ideia original do trabalho foi desenvolver um mundo virtual completo, semelhante em grande parte com o mundo real. Dentre as funcionalidades previstas, encontravam-se a divisão do terreno em relevos específicos (desertos, florestas, planícies, etc), cidades/vilas, caminhos entre as cidades, rios e cadeias montanhosas. A combinação de todos esses elementos seria capaz de criar um mundo virtual muito próximo da realidade, fato que seria de suma importância para garantir um bom resultado da ferramenta.

Quando o planejamento foi finalizado, a complexidade de determinadas funcionalidades previstas tornou proibitiva a sua implementação. A grande maioria dos problemas encontrados é uma consequência da abordagem de geração dinâmica de conteúdo sob demanda (a medida que o usuário se move, novos elementos são colocados na tela). A Figura 3.1 ilustra o problema da geração de conteúdo sob demanda.

Partindo do fato que o usuário só consegue enxergar aquilo que está dentro do seu campo de visão, todos os algoritmos de geração de conteúdo, seja para relevo, caminhos ou cidades, precisam levar em consideração única e exclusivamente as informações que estão disponíveis dentro desse campo. Essa abordagem é eficiente para a utilização racional de recursos (processar somente o que o usuário está vendo), porém ela aumenta



Figura 3.1: Problema da geração de conteúdo sob demanda.

a complexidade dos algoritmos envolvidos na ferramenta.

Para o algoritmo de geração de cadeias montanhosas, por exemplo, não é possível determinar onde a cadeia termina, visto que o mundo fora do campo de visão tecnicamente não existe ainda, ele será gerado conforme o usuário avança pelo terreno. Uma abordagem seria utilizar uma função matemática que descrevesse a cadeia montanhosa, porém essa função não deveria depender de um ponto de início e fim, porque eles poderiam inexistir em um determinado momento. Se a função de geração de cadeias montanhosas não dependesse de um ponto de início e fim, ela precisaria, ao menos, depender da posição do usuário no mundo virtual para que o conteúdo correto fosse gerado. Depender de uma localização implicaria que a cadeia montanhosa gerada pela função fosse pré-posicionada no mundo virtual, o que iria contra o conceito de geração de conteúdo sob demanda.

Além disso, os algoritmos são sensivelmente afetados pelo fato de que as informações que eles recebem em um determinado instante podem desaparecer por completo na próxima iteração, visto que o usuário pode se mover e mudar o conteúdo do campo de visão. Utilizando o exemplo da geração de cadeias montanhosas, uma montanha poderia sofrer uma alteração em sua composição de forma abrupta, apenas porque os pontos que estavam sendo utilizados para a geração do relevo mudaram.

Para contornar esses problemas e focar os esforços de desenvolvimento em soluções pontuais, a geração do mundo virtual foi dividida em três grandes etapas: terreno infinito, continentes e relevo. A geração de conteúdo sob demanda afeta de forma diferenciada cada uma dessas etapas e a descrição da implementação de cada uma delas, junto com os problemas associados, é descrito nas seções seguintes.

3.1 Terreno infinito

A base para a geração do mundo virtual proposto é a possibilidade do usuário poder andar, de forma infinita, sobre a superfície do mundo e, conforme anda, visualizar novos conteúdos. A medida que o usuário anda, a ferramenta precisa ser capaz de identificar em qual local do mundo o observador se encontra para então gerar os conteúdos à sua volta.

Para solucionar esse problema, utilizou-se uma variação da técnica descrita por [14]. Na abordagem dos autores em questão, o mundo virtual pseudo-infinito é dividido em células quadradas e, à medida que o usuário anda, as células são adicionadas e/ou removidas do campo de visão. Cada célula possui um conteúdo próprio e auto-contido, ou seja, a



Figura 3.2: Organização do sistema de coordenadas do mundo virtual

célula não precisa de informações de vizinhos para gerar o seu conteúdo. Isso garante que as células não entrem em uma dependência recursiva infinita entre elas para conseguirem gerar o seu conteúdo. Além disso, essa abordagem é vantajosa para garantir o uso racional de recursos, uma vez que só serão carregados para a memória os blocos que o usuário realmente consegue ver.

Para o posicionamento do usuário no mundo virtual, os autores utilizam um vetor 3D no formato (x, y, z) . Conforme o usuário se move horizontalmente pelo mundo, as coordenadas x e y são atualizadas. Se o usuário se move verticalmente, a coordenada Z é alterada. A abordagem utilizada para a ferramenta dessa dissertação baseou-se nesses conceitos. A figura 3.2 ilustra a organização do mundo virtual.

A origem do mundo virtual é o ponto $(0, 0, 0)$ e os eixos que definem o plano horizontal são o X e Z , sendo a altura controlada pelo eixo Y . A distância máxima que o usuário consegue percorrer em qualquer um dos eixos é o número máximo suportado por um inteiro de 32 bits com sinal.

O que o jogador consegue ver na tela em um determinado momento é um pedaço do mundo virtual existente. Esse pedaço foi chamado de *view frustum*, ou campo de visão. Diferentemente do que foi feito em [14], no qual o campo de visão é um cone, o campo de visão da presente ferramenta é um quadrado centrado no usuário. A figura 3.3 ilustra o funcionamento do campo de visão.

A partir da posição (x, y, z) do usuário, a ferramenta calcula qual é o conteúdo visualizável ao redor do referido ponto. Ao chegar na borda limite do mundo, que pode ser a distância máxima de um eixo, por exemplo, o usuário é impedido de avançar e nenhum conteúdo é mostrado além da borda limite.

Inicialmente planejou-se a utilização de quadrados para dividir o mundo virtual em

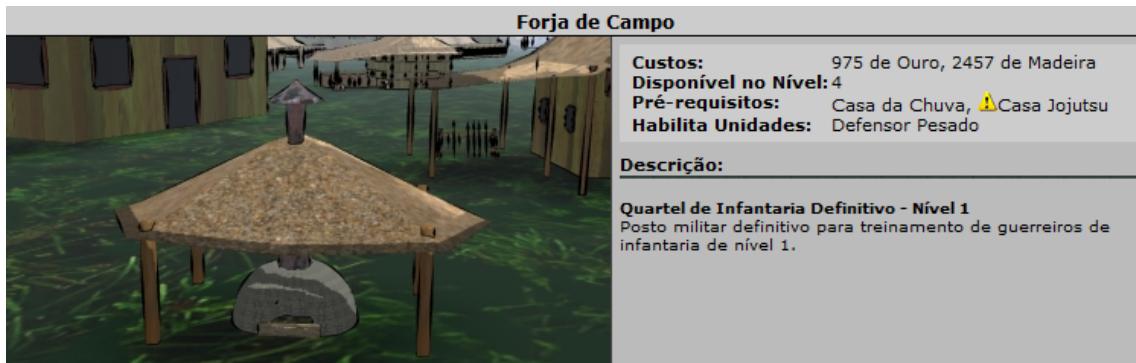


Figura 3.3: Campo de visão do usuário: a área visualizada corresponde a uma fatia do mundo virtual existente.

células, conforme é feito na outra abordagem citada anteriormente. A utilização de células garantiria que o mundo virtual fosse subdividido em blocos menores, o que viabilizaria um melhor controle sobre o que o usuário consegue ver no seu campo de visão e, também, um melhor controle sobre a geração de conteúdo. O maior problema encontrado nessa abordagem, que foi a razão pela qual ela foi abandonada, é a dependência que as células precisam ter entre si para que o terreno infinito seja gerado.

Utilizando o exemplo da geração de relevo, que é tratada em mais detalhes na seção 3.3, se o mundo virtual fosse dividido em células, cada uma delas deveria possuir um relevo perfeitamente nivelado com a célula vizinha, caso contrário o relevo gerado teria diversos "degraus". Analisando o problema um pouco mais a fundo, no caso da geração de montanhas, por exemplo, se na metade de uma célula a ferramenta decidisse que uma cadeia montanhosa deveria começar, a célula vizinha deveria obrigatoriamente ter a continuação dessa cadeia montanhosa, caso contrário a montanha em questão seria fatiada pela metade. Uma das formas de corrigir esse problema é adicionar um nível mínimo de dependência entre as células: o conteúdo de uma célula é gerado com base nas informações da própria célula e também com base em alguma "dica" da célula vizinha. No exemplo da cadeia montanhosa, a célula vizinha ao começo da cadeia saberia que o conteúdo que ela deve gerar é a continuação da cadeia montanhosa, visto que a sua célula vizinha possui o começo da cadeia.

Essa dependência de conteúdo gera um encadeamento recursivo infinito entre as células. Se a célula A, por exemplo, for gerar o seu conteúdo, ela irá fazer isso com base nas suas informações e também com base nas informações de sua célula vizinha, B. A célula B, por sua vez, só poderá informar a A o seu próprio relevo quando ela o gerar; para gerar



Figura 3.4: Mapeamento das coordenadas do mundo virtual para as coordenadas de desenho da tela

o seu relevo, ela precisa das suas informações e das informações da sua vizinha, C, e C precisa de D e assim por diante. Dessa forma, para gerar o conteúdo de A, a ferramenta teria que obrigatoriamente percorrer todas as células do mundo virtual.

Uma saída alternativa para esse problema da recursão infinita é definir um nível de consulta de informações. Embora essa abordagem limite o nível de consulta entre as células vizinhas, ela não soluciona o problema de continuidade de conteúdo. Se o conteúdo de A for gerado com seis níveis de recursão, por exemplo, quando o usuário se mover, novas células vizinhas serão consultadas para a geração do conteúdo; se a última célula que A consultou foi G, depois que o usuário se mover, G terá novas informações para o seu relevo, porque agora ela pode solicitar informações de suas vizinhas. Isso fará com que todas as células dependentes de G mudem o seu conteúdo, o que resultaria em um relevo diferente a cada movimentação do usuário. Em virtude da complexidade descrita e dos problemas mapeados, a abordagem de divisão do mundo virtual em células foi abandonada e substituída pelo modelo de campo de visão quadrado.

3.1.1 Renderização de conteúdo

Depois que o modelo de controle da geração de terrenos infinitos foi definido, iniciaram-se os trabalhos de computação gráfica para que as informações pudessem ser desenhadas na tela. Embora o mundo virtual tenha coordenadas pseudo-infinitas, o máximo de conteúdo que o usuário enxerga na tela é a área do campo de visão, que possui sempre o mesmo tamanho e coordenadas de desenho. A figura 3.4, quadrado 1, ilustra as coordenadas envolvidas no desenho do conteúdo do campo de visão.

O plano A representa o mundo virtual gerado pela ferramenta, enquanto o plano B representa a fatia do mundo virtual que o usuário consegue visualizar. A medida que

o usuário se move, o centro do campo de visão é alterado e o usuário passa a ver novos conteúdos. Independente da movimentação que o usuário faça, o plano B pode sempre ser mapeado como se estivesse na origem, porque ele é apenas uma fatia do mundo virtual. O que a ferramenta faz para desenhar esse conteúdo na tela é extrair essa fatia e, então, mapeá-la para um mapa de altura (*height map*). Depois que o mapa de altura é definido, as coordenadas dos eixos X e Z do mundo virtual que foram utilizadas durante a extração não são mais relevantes para o processo de desenho. Dessa forma, depois de extraído, o mapa de altura possui sempre as mesmas coordenadas nos eixos X e Z, que são as coordenadas de desenho da tela. A única informação do mundo virtual que é mantida é a altura de cada um dos pontos da malha do mapa de altura, conforme ilustra a figura 3.4, quadrado 2. O mapa de altura gerado é renderizado como uma malha triangular.

3.2 Continentes

– falar que utilizamos o algoritmo maluco do professor dos EUA para fazer a costa. Falar que existe o mundo é gigante o suficiente para fazer com que uma matrix que descreve água/terra seria inviável. Para solucionar esse problema, falar que utilizamos o conceito de isLang local e isLang global. O isLang global dá uma dica se o lugar é água ou terra, e o isLang local utiliza multifractais para fazer o desenho das bordas dos continentes.

– É importante frisar que essa seção é onde está a nossa contribuição na pesquisa: mundo pseudo-infinito, com relevo gerado on-the-fly e com costas de continentes com multiresolução.

3.3 Relevo

A ideia original do presente trabalho foi desenvolver um mundo virtual capaz de apresentar diversos tipos de relevos, como cadeias montanhosas, planícies, vales, desertos, florestas, etc. A complexidade associada à geração de cada um desses elementos varia conforme o nível de realismo esperado, bem como pelo nível de dinamismo do conteúdo gerado. Quanto mais presente for o conceito de geração de conteúdo sob demanda, mais difícil torna-se a tarefa de gerar conteúdos conexos e sem falhas abruptas na malha de relevo.

Mantendo-se a meta de gerar o conteúdo da forma mais dinâmica possível (sem ele-



Figura 3.5: Geração de conteúdo com base apenas as informações do campo de visão

mentos pré-posicionados, por exemplo), o primeiro grande problema encontrado durante o desenvolvimento do relevo do mundo virtual foi a forma como ele deveria ser gerado. Considerando que a ferramenta é capaz de criar um mundo pseudo-infinito, a geração de uma malha de relevo tão grande, de uma só vez, é fisicamente inviável; a quantidade de vértices que precisariam ser armazenados tornaria o consumo de armazenamento da aplicação muito grande, o que poderia ser um empecilho para a utilização da ferramenta.

Além do problema de armazenamento, outro tópico importante que foi considerado é a geração de conteúdo contínuo, ou seja, um relevo que não tenha falhas abruptas de continuidade, como uma cadeia montanhosa que termina inesperadamente. Partindo do fato que o mundo virtual não poderia ser dividido em células, tendo em vista os diversos problemas de continuidade de conteúdo, a solução para a geração de relevo deveria basear-se apenas nas informações disponíveis no campo de visão. Embora seja possível gerar relevo utilizando-se como valores de entrada somente as informações do campo de visão, novos problemas de continuidade foram encontrados.

Supondo que a geração de relevo utilize a posição do vértice no mundo virtual como semente para uma função pseudo-aleatória de cálculo de relevo; supondo também que a geração de montanhas, por exemplo, é definida por pontos chave que indicam a espinha dorsal da cadeia montanhosa. Se o algoritmo se basear apenas nesses pontos chave para calcular as montanhas, o usuário só irá enxergar a montanha quando um desses pontos chave entrar dentro do campo de visão. Isso irá causar falhas abruptas de continuidade, porque se o usuário costear a espinha dorsal da montanha, sem que os pontos chave entrem no campo de visão, ele não verá a cadeia de montanhas, sendo que ele deveria ver um relevo ascendente até o cume dessas montanhas. A figura 3.5 ilustra esse problema.

A solução encontrada para a geração de relevo foi utilizar uma função paramétrica



Figura 3.6: Função de geração de relevo parametrizável e o resultado gerado

que informa a característica de cada um dos vértices do mundo virtual. Nessa abordagem, cada ponto do mundo é utilizado como parâmetro para a função de relevo que, por sua vez, calcula a altura que esse ponto deve ter. Como resultado, tem-se uma função capaz de descrever todo o relevo do mundo virtual, independente do tamanho que ele seja; a quantidade de recursos computacionais que serão utilizados para gerar o conteúdo do campo de visão é diretamente proporcional ao tamanho do campo de visão, não ao tamanho do mundo, visto que a função de relevo utiliza as informações de cada ponto para calcular a sua característica. A figura 3.6 ilustra o funcionamento da função de relevo aplicada ao mundo virtual gerado.

Para aumentar a heterogeneidade do relevo gerado, a ferramenta utiliza duas funções de relevo, uma para o eixo X e outra para o eixo Z. Além de garantir que o relevo gerado não seja perfeitamente simétrico nos dois eixos, essa abordagem permite um maior controle sobre a geração de conteúdo. A altura de cada ponto do mapa é definido pela soma dessas duas funções de ruído.

Inicialmente utilizou-se uma combinação de funções seno e cosseno para a geração do relevo.

4 RESULTADOS

5 CONCLUSÃO E TRABALHOS FUTUROS

–Trabalhos futuros: – Melhoramento nas funções que geram relevo; – Texturização conforme altura do terreno e afins; – Rios; – GPU;

REFERÊNCIAS

- [1] A. Aho and J. D. Ullman. *Compiladores: conceitos, técnicas e implementação.* Makron Books, 1993.
- [2] A. Aho and J. D. Ullman. *Compiladores: conceitos, técnicas e implementação.* Makron Books, 1993.
- [3] A. Aho and J. D. Ullman. *Compiladores: conceitos, técnicas e implementação.* Makron Books, 1993.
- [4] A. Aho and J. D. Ullman. *Compiladores: conceitos, técnicas e implementação.* Makron Books, 1993.
- [5] A. Aho and J. D. Ullman. Compiladores: conceitos, técnicas e implementação., 1993.
- [6] arrumar. arrumar, 1989. ARRUMAR.
- [7] arrumar. arrumar, 1989. ARRUMAR.
- [8] arrumar. arrumar, 1989. ARRUMAR.
- [9] arrumar. arrumar, 1989. ARRUMAR.
- [10] Bernd Lintemann and Oliver Deussen. A modelling method and user interface for creating plants, 1998. Computer Graphics Forum.
- [11] F. Kenton Musgrave and Craig E. Kolb and Robert S. Mace. The synthesis and rendering of eroded fractal terrains, 1989. Computer Graphics, Volume 23, Number 3, July 1989, pages 41-50.
- [12] Filip Strugar. Advantage terrain, 2007. balbal.

- [13] Filip Strugar. Advantage terrain, 2007. blahblah.
- [14] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Realtime procedural generation of 'pseudo infinite' cities, 2005.
- [15] Jason Weber and Joseph Penn. Creation and rendering of realistic trees, 1995. blabla.
- [16] Prusinkiewicz Przemyslaw and Aristid Lindenmayer. The algorithmic beauty of plants, 1990. Springer-Verlag.
- [17] Thomas Wang. Integer hash function. Technical report, 2000. Available at: <<http://www.concentric.net/Ttwang/tech/inthash.htm>>.