Tiago José de Carvalho

Geração Procedural de Terrenos

Orientador:
Marcelo Bernardes Vieira

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Juiz de Fora

Julho de 2008

Monografia submetida ao corpo docente do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como parte integrante dos requisitos necessários para obtenção do grau de bacharel em Ciência da Computação

Prof. Marcelo Bernardes Vieira, D. Sc.
Orientador

Prof. Raul Fonseca Neto, D. Sc.

Prof. Rodrigo Weber dos Santos, D. Sc.

Sumário

Lista de Figuras

Resumo

Este trabalho visa um estudo de métodos para geração procedural de terrenos tendo como principal embasamento as técnicas de ruído e turbulência descritas por Perlin, de maneira a possibilitar uma representação satisfatória de terrenos através de mapas de alturas e texturas gerados proceduralmente. São apresentados os fundamentos necessários para a compreensão do problema, a análise de soluções eficientes computacionais e as implementações práticas desses conceitos de maneira independente. São apresentadas algumas formas de se empregar tais métodos para se gerar texturas procedurais de grande aplicação em computação gráfica como madeira e mármore. Finalizando, gera-se um terreno para ser mapeado em uma superfície esférica de forma totalmente procedural.

1 Introdução

A computação gráfica é a área da computação destinada à geração de imagens em geral — em forma de representação de dados e informação, ou em forma de recriação do mundo real (BRITO, 2006).

A geração de terrenos é uma área muito interessante e de grande utilização, afinal a maioria dos ambientes virtuais baseados no mundo real necessita de um terreno.

Sua maior utilização é dada na construção de jogos, simuladores de vôo e geração de sistemas planetários.

1.1 Considerações Iniciais

Um dos modos de se gerar terrenos de forma procedural é utilizando técnicas denominadas por Fournier como impressionista para a modelagem e texturização dos mesmos. Segundo o próprio Fournier define-se *modelagem impressionista* como sendo o tipo de modelagem que utiliza funções matemáticas que não estão baseadas em leis físicas reais com relação aos fenômenos ou elementos que se deseja criar. Esta técnica recebe este nome pelo fato de que o único objetivo consiste em criar uma impressão da "aparência externa" (CLUE, 1999).

Uma das formas de se modelar um terreno é através do emprego de *ruído coerente*, o qual possui as propriedades de ser *suave e pseudo-aleatório*.

Um dos tipos de ruído coerente mais utilizado no meio acadêmico é denominado ruído de Perlin. Desenvolvido por Ken Perlin, tal ruído é amplamente utilizado devido a sua grande difusão dentro do meio acadêmico, além de possuir uma grande aplicabilidade.

Além da função de ruído, Perlin propôs uma nova função tendo como base a função de ruído. Ela foi denominada turbulência.

A função de turbulência torna-se o ponto chave para a modelagem impressionista

de elementos da natureza como nuvens, mármore, ondas para a superfície de um mar e principalmente terrenos, uma vez que os resultados obtidos são muito satisfatórios.

A geração procedural de texturas também é de grande importância na geração de terrenos uma vez que elas são capazes de se adaptar perfeitamente ao terreno.

1.2 Definição do Problema

O problema tratado neste trabalho é o da geração e texturização procedural de terrenos.

Na geração, o problema é a escolha do método mais adequado a ser empregado. Isso porque o controle da quantidade de montanhas, vales e morros no terreno depende do método. Na texturização, o método utilizado também é quem vai determinar o quão realista a textura será.

1.3 Objetivos

O objetivo primário desta monografia é pesquisar e formalizar os métodos de geração e texturização procedural de terrenos, e como objetivos secundários, advindos do objetivo primário, temos:

- apresentar matematicamente os conceitos aplicáveis na definição de ruído coerente e turbulência;
- propor uma representação de classes abstratas para geração de ruído de Perlin e turbulência;
- aplicar os conceitos e as representações propostas na implementação de um gerador de terrenos, incluindo mapas de alturas e texturas, de forma totalmente procedural;

1.4 Visão Geral

O presente trabalho tem como objetivo apresentar conceitos ligados à base matemática para se entender a geração de terrenos de forma procedural, bem como exibir as técnicas para se modelar um terreno de acordo com a estrutura de capítulos apresentada abaixo.

No capítulo dois é apresentado o conceito de sinal e ruído, bem como algumas formas de ruído. A principal delas é o ruído de Perlin, apresentado na seção 2.3.1, no qual a geração procedural de terrenos pode se basear.

Já no capítulo três são apresentadas as formas de se gerar o ruído de Perlin em uma, duas ou três dimensões. Em suma, a geração em todas as dimensões são muito semelhantes, mudando apenas alguns detalhes de uma dimensão para outra.

O capítulo quatro mostra algumas aplicações do ruído de Perlin em CG. Nele são apresentadas formas de se gerar algumas texturas utilizando-se do ruído. A geração de um terreno, foco principal deste trabalho, é mostrada na seção 4.2 onde são descritos alguns dos procedimentos e configurações utilizadas para se obter alguns terrenos, incluindo mapas de alturas e texturas.

Por fim, no capítulo cinco são descritas algumas conclusões sobre o trabalho além de serem apresentadas algumas considerações finais a respeito do ruído de Perlin.

2 Análise e Síntese de Ruído em Sinais

2.1 Sinal

Um *sinal* (PROAKIS; MANOLAKIS, 1996) é uma grandeza física que varia no tempo, no espaço ou em alguma outra variável(is) independente(s).

Matematicamente um sinal é definido como uma função de uma ou mais variáveis independentes.

Como exemplos podemos citar o som ($f(t)$) e a imagem ($f(x, y)$) dentre outros.

2.1.1 Transformada de Fourier

A *transformada de Fourier* é uma ferramenta matemática que basicamente decompõe um sinal em função de uma série de senóides. Ao ser representado por senóides, um sinal está sendo representado no *domínio da frequência*.

Para sinais periódicos, tal representação é chamada de *série de Fourier*. Para sinais de energia finita, é através da transformada de Fourier que obtemos uma representação do sinal como uma série de senóides, também chamado de *espectro*.

Um sinal representado através da transformada de Fourier, pode ser reconstruído (recuperado) completamente através de um processo inverso, sem perda de informação (GONZALES; WOODS, 1992).

Muitas sequências de sinais podem ser representadas por uma integral de Fourier (OPPENHEIM; SCHAFER; BUCK, 1999) na forma

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X\left(\exp^{j\omega}\right) \exp^{j\omega n} d\omega \tag{2.1}$$

onde

$$X\left(\exp^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n] \exp^{-j\omega n} \tag{2.2}$$

Juntas, as duas equações acima formam a *representação de Fourier* de um sinal qualquer.

2.2 Ruído

Todo sinal está sujeito a presença de ruído. Assim, *ruído* pode ser definido como qualquer degradação que afete o sinal original.

2.2.1 Propriedades do Ruído

A frequência em que um determinado sinal está contido dentro da série de Fourier define suas propriedades. Por exemplo, quando o espectro de Fourier do ruído é constante ele é chamado de *ruído branco* (GONZALES; WOODS, 1992). Essa terminação vem do fato de uma leve semelhança com a luz branca, a qual contém quase todas as frequências do espectro visível em iguais proporções.

O ruído pode ser caracterizado também pela sua probabilidade de ocorrência em cada pixel da imagem. Isso é dado pela *função densidade de probabilidade, ou FDP*. Existem várias funções FDP importantes como o *ruído Gaussiano*, o *ruído Rayleigh*, dentre outros.

2.2.2 Ruído Gaussiano

Por causa de sua característica de fácil manipulação matemática em ambos os domínios (espacial e da frequência) o modelo de ruído Gaussiano (também chamado de *ruído Normal*) é usado frequentemente na prática.

A FDP do ruído Gaussiano para z qualquer é dada por

$$p(z) = \frac{1}{\sqrt{2\pi\sigma}} \exp^{\frac{-(z-\mu)^2}{2\sigma^2}}, \tag{2.3}$$

onde z representa a intensidade do sinal, μ é a média dos valores de z e σ é o *desvio padrão*. O quadrado do desvio padrão, σ^2 é chamado de *variância* de z.

Quando a FDP é descrita pela Eq.(2.3) aproximadamente 70% dos valores de z estarão

no domínio $[(\mu+\sigma),(\mu-\sigma)]$, e aproximadamente 95% no domínio $[(\mu+2\sigma),(\mu-2\sigma)]$.

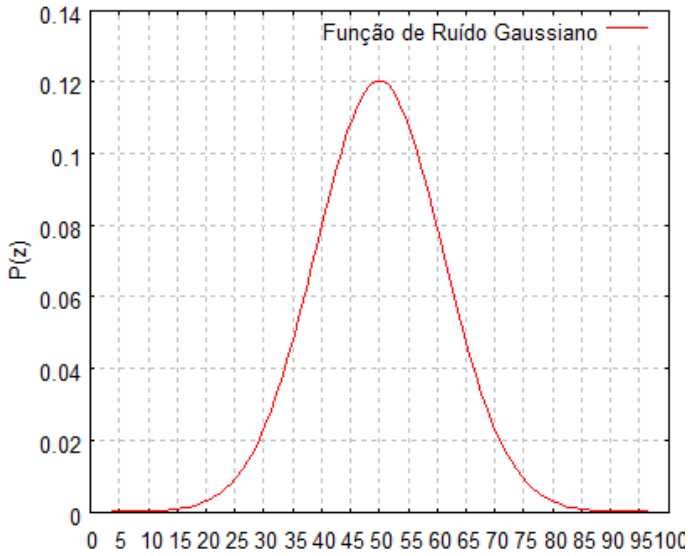


Figura 1: Gráfico do ruído Gaussiano.

2.2.3 Ruído Rayleigh

A FDP do ruído Rayleigh é dada por

$$p(z) = \begin{cases} \frac{2}{b}\exp^{\frac{-(z-a)^2}{b}} & \text{para} \quad z \geq a \\ 0 & \text{para} \quad z < a, \end{cases} \tag{2.4}$$

onde a média μ é dada por

$$\mu = a + \sqrt{\pi\frac{b}{4}}\,, \tag{2.5}$$

e a variância σ é dada por

$$\sigma^2 = \frac{b(4-\pi)}{4} \tag{2.6}$$

2.2.4 Ruído Erlang (Gamma)

A FDP do *ruído Erlang* é dada por

$$p(z) = \begin{cases} \frac{(a^b z^{b-1})}{(b-1)!}\exp^{-az} & \text{para} \quad z \geq 0 \\ 0 & \text{para} \quad z < 0, \end{cases} \tag{2.7}$$

onde a média μ é dada por

$$\mu = \frac{b}{a}\,, \tag{2.8}$$

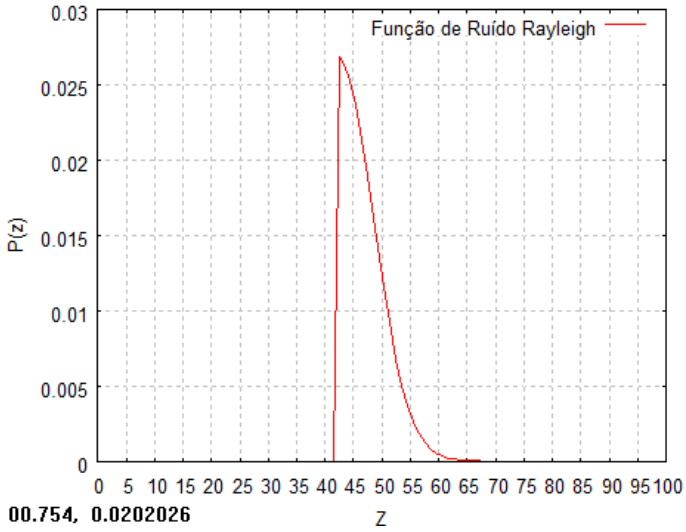


Figura 2: Gráfico do ruído Rayleigh.

e a variância σ é dada por

$$\sigma^2 = \frac{b}{a^2} \tag{2.9}$$

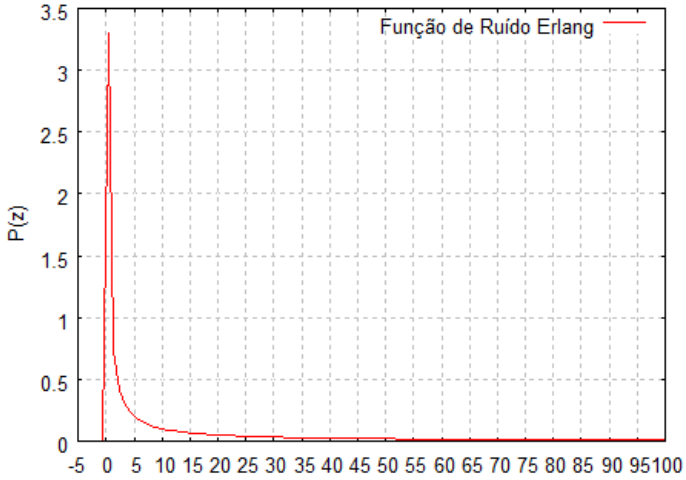


Figura 3: Gráfico do ruído Gamma.

2.2.5 Ruído Exponencial

O *ruído Exponencial* nada mais é do que um caso específico do ruído Erlang onde b = 1. Assim, a PDF do ruído Exponencial é dada por

$$p(z) = \begin{cases} a \exp^{-az} & \text{para} \quad z \geq 0 \\ 0 & \text{para} \quad z < 0, \end{cases} \tag{2.10}$$

onde a média μ é dada por

$$\mu = \frac{1}{a}, \tag{2.11}$$

e a variância σ é dada por

$$\sigma^2 = \frac{1}{a^2} \tag{2.12}$$

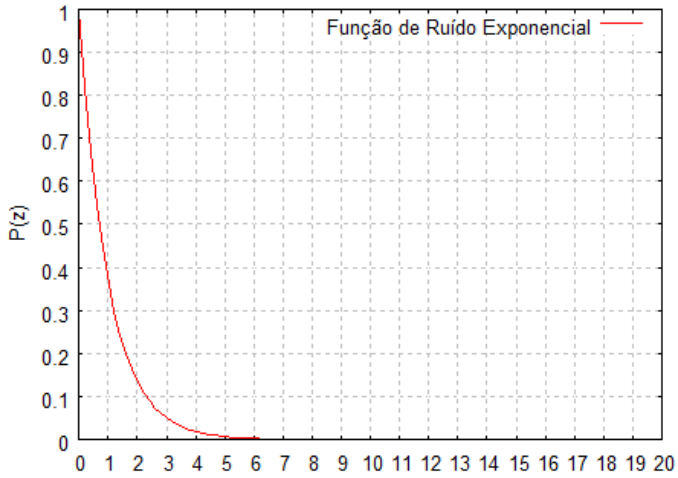


Figura 4: Gráfico do ruído Exponencial.

2.2.6 Ruído Uniforme

No *ruído Uniforme*, a FDP é dada por

$$p(z) = \begin{cases} \dfrac{1}{(b-a)} & \text{se} \quad a \leq z \leq b \\ 0 & \text{caso contrário,} \end{cases} \tag{2.13}$$

onde a média μ é dada por

$$\mu = \frac{a+b}{2}, \tag{2.14}$$

e a variância σ é dada por

$$\sigma^2 = \frac{(b-a)^2}{12} \tag{2.15}$$

2.2.7 Ruído Sal-e-Pimenta

A FDP do *ruído Sal-e-Pimenta*, é dada por

$$p(z) = \begin{cases} P_a & \text{para} \quad z = a \\ P_b & \text{para} \quad z = b \\ 0 & \text{caso contrário,} \end{cases} \tag{2.16}$$

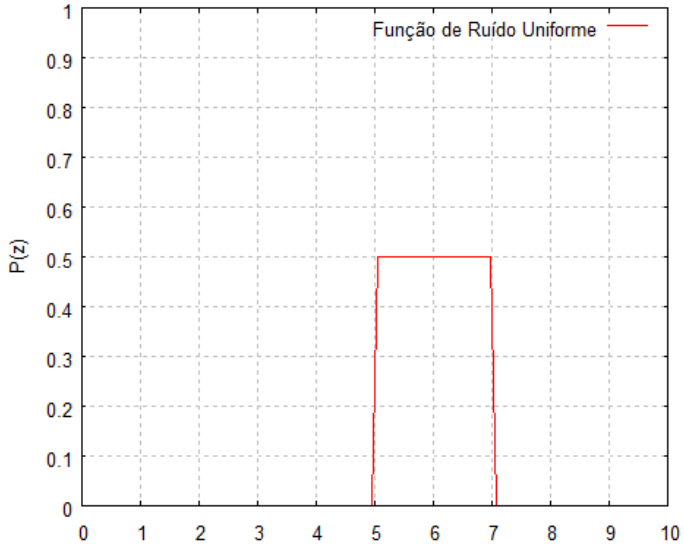


Figura 5: Gráfico do ruído Uniforme.

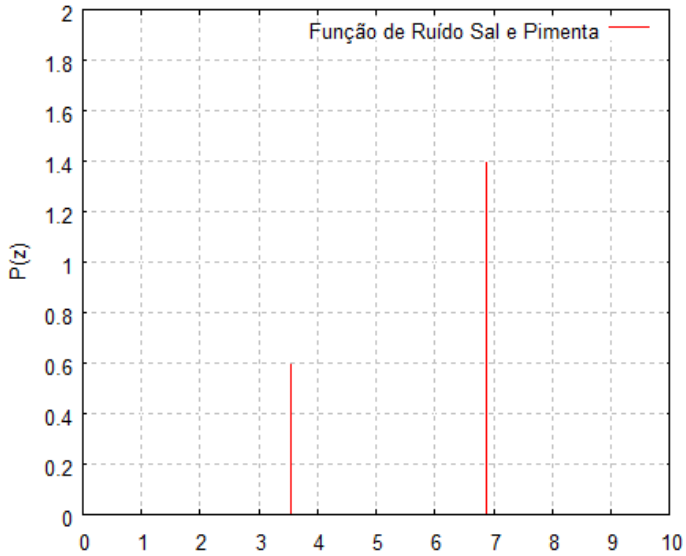


Figura 6: Gráfico do ruído Sal e Pimenta.

2.3 Ruído Coerente

Apesar da grande variedade de tipos de ruído vistos até agora, nenhum deles é suficiente para se gerar um terreno. Isso porque em um determinado sinal, qualquer um dos tipos de ruído apresentados até agora não atuam de forma a estruturar padrões desejados para geração de terrenos em Computação Gráfica (CG), como visto na Fig. 7.

Sendo assim a geração de um terreno com esse tipo de ruído se tornaria muito difícil uma vez que, não seria possível ter um controle das distorções finais (morros, vales, montanhas) a serem gerados.

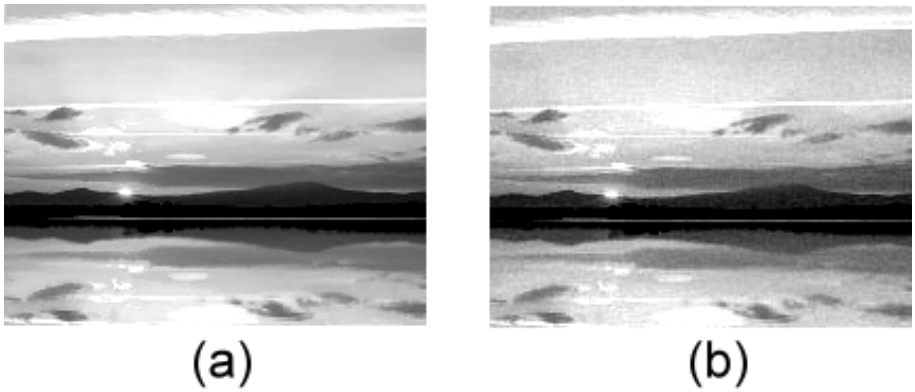


Figura 7: Ruído não coerente aplicado a uma imagem. (a) Figura sem ruído. (b) Figura com ruído Gaussiano aplicado

Por isso, para que se consiga gerar um terreno com uma aparência realista, é necessário o uso de um outro tipo de ruído, no qual é possível se ter algum controle do resultado final a ser obtido. Esse tipo de ruído é chamado *ruído coerente*.

O ruído coerente, através do uso de uma função coerente, gera valores *suaves e pseudo-aleatórios*. Suaves porque uma pequena modificação no valor de entrada irá gerar uma pequena modificação no valor de saída. Pseudo-aleatórios devido ao fato de que a aplicação do ruído em um determinado ponto do sinal, sempre resultará no mesmo valor de saída, desde que sejam mantidas as configurações (parâmetros de entrada da função) do ruído.

A diferença entre uma função coerente e um outro tipo de função que não apresenta essa característica pode ser observada nas Figuras 8 e 9 respectivamente. Na primeira, Fig. 8, é representada a saída da função seno, uma das funções coerentes mais conhecidas. Já na segunda, Fig. 9, é mostrado o gráfico de uma função n(x) não coerente também de dimensão 1.

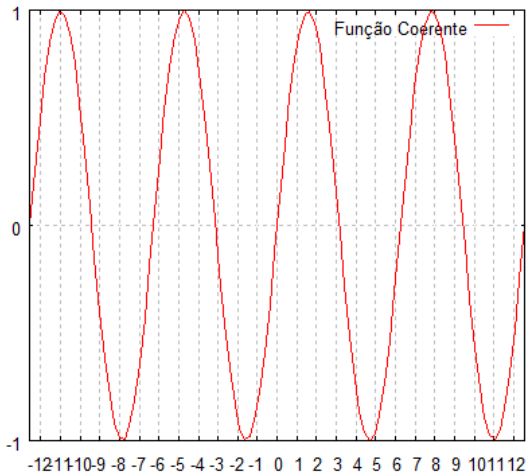


Figura 8: Gráfico da função seno, uma das funções coerentes mais conhecidas.

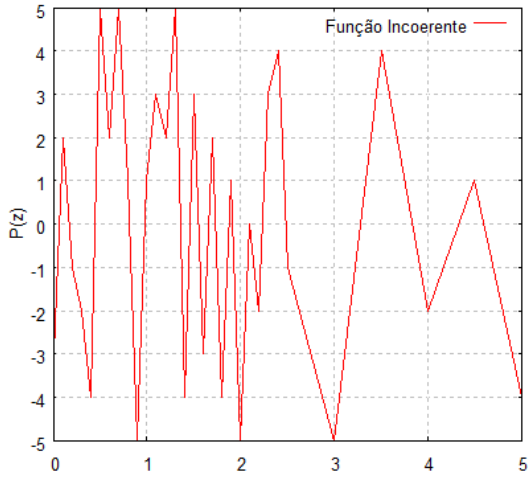


Figura 9: Gráfico de uma função não coerente.

2.3.1 Ruído de Perlin

O *ruído de Perlin* é um dos tipos de ruído coerente. Apesar de não ser a única função de ruído coerente, é uma das mais utilizadas devido a sua grande aplicabilidade e por ser bastante difundida no meio acadêmico.

Segundo Perlin (PERLIN, 1989), esse tipo de ruído permite introduzir valores aleatórios em um sinal digital sem sacrificar qualquer continuidade ou controle sobre a frequência espacial.

O ruído de Perlin pertence a classe de *processos estocásticos*, os quais são definidos como sendo uma coleção de variáveis aleatórias X_t, $t \in \mathbf{T}$, onde X_t depende exclusivamente de X_{t-1}, podendo a variável t ser discreta ou contínua. A cardinalidade do conjunto de índices $\mathbf{T}$ indica se o processo é um processo discreto ou contínuo.

2.3.1.1 Síntese do Ruído de Perlin

Alguns conceitos importantes devem ser levados em consideração antes de se prosseguir na síntese do ruído de Perlin:

- Um *reticulado* (*lattice*) é o conjunto de todos os pontos no espaço discretizado R^n, sendo que as coordenadas x_0, x_1,...,x_n são valores inteiros. Cada vértice pertencente ao reticulado possui espaçamento de uma unidade. Um reticulado em 3D pode ser visto na Fig. 10

- A cada um dos vértices pertencentes ao reticulado deve ser associado um vetor

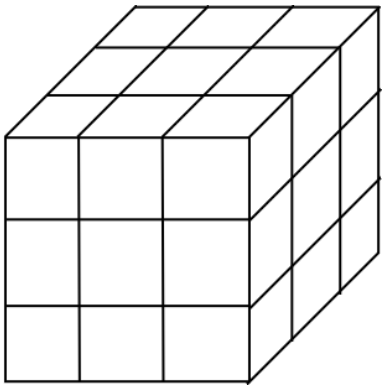


Figura 10: Representação do reticulado em 3D

denominado *vetor gradiente* $\boldsymbol{g_v}$. Para fazer isto, pode-se criar uma tabela, onde se armazenará um vetor relativo ao gradiente de cada vértice.

Perlin (PERLIN, 1985) propõe que a geração do gradiente ocorra da seguinte forma:

- Escolher pontos aleatórios que estejam dentro do cubo $[-1, 1]^{\mathsf{n}}$;
- Normalizar o vetor obtido.

Baseando-se nos conceitos apresentados, a aplicação do ruído de Perlin a um determinado ponto $p(\mathsf{x}_0, \mathsf{x}_1,...,\mathsf{x}_\mathsf{n})$, segue as seguintes etapas:

1. Enquadrar o ponto dentro da frequência f escolhida para o ruído. Para isto basta multiplicar as coordenadas de p pelo valor da frequência. Assim as coordenadas $(\mathsf{x}_0, \mathsf{x}_1,...,\mathsf{x}_\mathsf{n})$ de p serão substituídas pelas novas coordenadas $(\mathsf{x}_0 \cdot \mathsf{f}, \mathsf{x}_1 \cdot \mathsf{f},...,\mathsf{x}_\mathsf{n} \cdot \mathsf{f})$.

2. Mapear o ponto p no reticulado.

3. Encontrar as coordenadas de todos 2^{n} vértices (chamados também de *vizinhos*) que compõem a célula do reticulado em que p está mapeado, sendo n a dimensão do espaço discretizado, como pode ser visto na Fig. 11. O primeiro dos vizinhos é obtido da seguinte maneira:

$$[\mathsf{a}_0, \mathsf{a}_1,...,\mathsf{a}_\mathsf{n}] = [\lfloor \mathsf{x}_0 \rfloor, \lfloor \mathsf{x}_1 \rfloor,...,\lfloor \mathsf{x}_\mathsf{n} \rfloor]$$

Assim, os vértices que definirão a célula em que o ponto se encontra serão:

$[\mathsf{a}_0, \mathsf{a}_1,...,\mathsf{a}_\mathsf{n}]$, $[\mathsf{a}_0+1, \mathsf{a}_1,...,\mathsf{a}_\mathsf{n}]$, $[\mathsf{a}_0, \mathsf{a}_1+1,...,\mathsf{a}_\mathsf{n}]$,...,$[\mathsf{a}_0, \mathsf{a}_1,...,\mathsf{a}_\mathsf{n}+1]$, $[\mathsf{a}_0+1, \mathsf{a}_1+1,...,\mathsf{a}_\mathsf{n}]$,...
$[\mathsf{a}_0 + 1, \mathsf{a}_1,...,\mathsf{a}_\mathsf{n} + 1]$,...,$[\mathsf{a}_0, \mathsf{a}_1 + 1,...,\mathsf{a}_\mathsf{n} + 1]$,...,$[\mathsf{a}_0 + 1, \mathsf{a}_1 + 1,...,\mathsf{a}_\mathsf{n} + 1]$

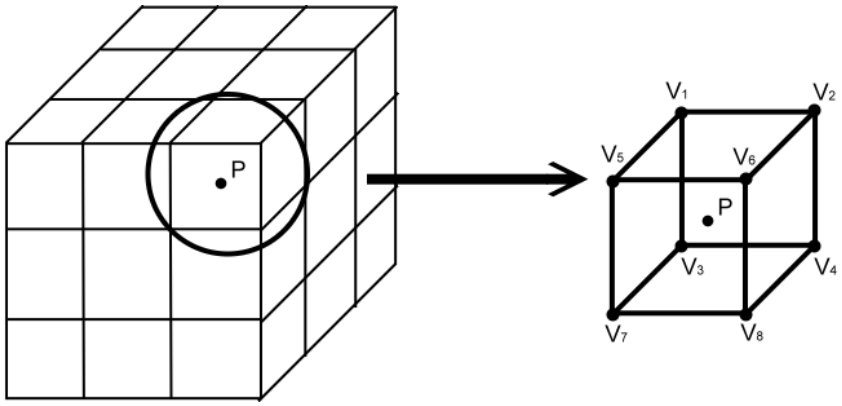


Figura 11: Mapeamento de um ponto p em uma célula do reticulado 3D à esquerda. À direita tem-se a célula em que p está mapeado e seus 2^n vizinhos.

4. Encontrar, para cada um dos vizinhos, o vetor $\boldsymbol{c_{pv}}$ que liga p ao vértice $v(a_0, a_1,...,a_n)$ do reticulado. Na Fig. 12 isso pode ser visto para um dos vizinhos.

$$\boldsymbol{c_{pv}} = (x_0 - a_0, x_1 - a_1,...,x_n - a_n);$$

5. Calcular o produto escalar e_{pv}, para cada um dos vizinhos, entre $\boldsymbol{g_v}$ e $\boldsymbol{c_{pv}}$. Esse valor é referente a influência que aquele vizinho exerce sobre o ruído final que afeta o ponto p.

$$e_{pv} = \boldsymbol{g_v} \cdot \boldsymbol{c_{pv}};$$

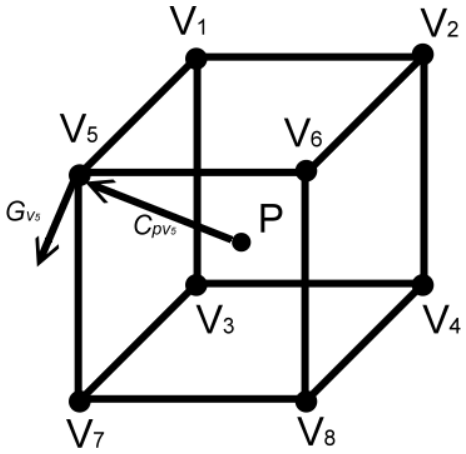


Figura 12: Representação de $\boldsymbol{g_v}$ e de $\boldsymbol{c_{pv}}$ entre p e seu vizinho de número 5 v_5 no reticulado 3D.

6. O valor final de ruído em p é obtido ao se realizar $2^n - 1$ interpolações lineares, entre os e_{pv} de todos os vizinhos. Em 3D serão realizadas sete interpolações como mostra a Fig. 13.

Para realizar as interpolações, Perlin propôs inicialmente o uso de um polinômio de grau três ($3t^2 - 2t^3$) (PERLIN, 1985) como *fator interpolante*, o qual dará o peso da interpolação em cada eixo. Porém algum tempo depois, o próprio Perlin (PERLIN, 2002) propôs que as interpolações fossem realizadas tendo como fator interpolante um polinômio de grau cinco ($6t^5 - 15t^4 + 10t^3$) para que o ruído possuísse uma aparência ainda mais suave. Tal diferença pode ser vista na Fig. 14.

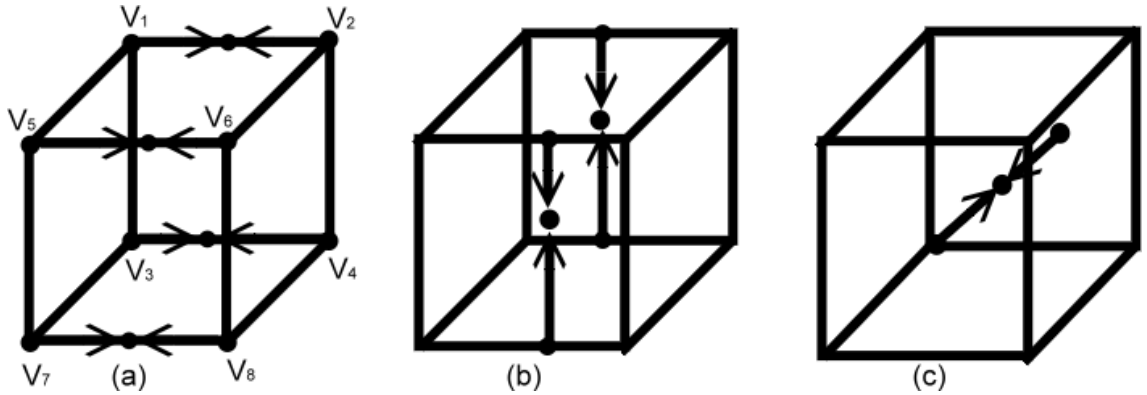


Figura 13: Representação das interpolações em um reticulado 3D para se chegar ao valor de ruído no método proposto por Perlin. (a) representa as interpolações em x, (b) as interpolações em y e (c) as interpolações em z.

7. Enquadrar o ruído dentro da amplitude a máxima proposta. Para tal, basta multiplicar o valor referente ao ruído encontrado por a.

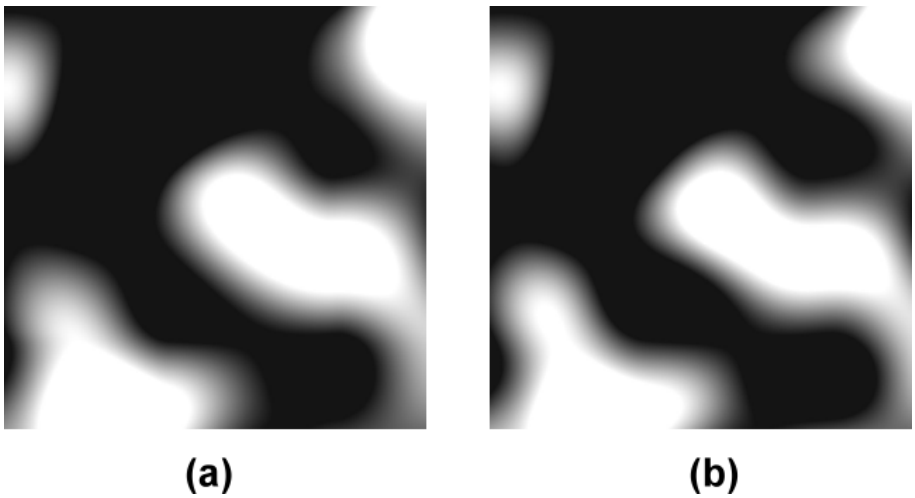


Figura 14: Figuras de ruído 2D, obtidos com baixa frequência e os mesmos parâmetros de entrada. A região totalmente preta representa que a energia do sinal naquele ponto é nula. A região totalmente branca representa que o sinal naquele ponto possui energia máxima. E os tons de cinza intermediários representam pontos de energia intermediária. (a) mostra o ruído utilizando como fator interpolante um polinômio de grau três. (b) mostra o ruído utilizando como fator interpolante um polinômio de grau cinco.

2.3.2 Turbulência

A função *turbulência* foi proposta por Perlin (PERLIN, 1985) como uma maneira de criar variados tipos de textura. No entanto, além da criação de texturas ela é de grande importância na geração de terrenos uma vez que ela consegue modelar as formas do relevo do terreno de uma maneira mais suave que a função de ruído isoladamente.

2.3.2.1 Função Turbulência

A função turbulência nada mais é do que uma iteração fractal da função ruído (CLUE, 1999), variando a cada iteração as escalas da frequência f e da amplitude a, fazendo assim com que o montante de ruído adicionado ao sinal a cada iteração, também denominado *harmônico*, seja proporcional à f e a.

Porém, ao contrário da função de ruído que gera uma deformação contínua no domínio [-1,1] o tempo todo, a função de turbulência utiliza-se apenas de valores absolutos, restringindo desse modo o domínio a [0,1] e conseguindo com isso passar uma impressão de descontinuidade que será interpretada pelo visualizador como sendo um aspecto *turbulento*. A função turbulência $T(p)$ pode ser definida como:

$$T(p) = \sum_{i=0}^{N-1} \left| r(f_i \cdot p) \cdot a_i \right|, \tag{2.17}$$

onde p é o ponto no qual se deseja aplicar a turbulência, *r()* é a função de ruído de Perlin, N é o número de harmônicos que se deseja gerar, e f_i e a_i são respectivamente a frequência e a amplitude em cada harmônico.

Uma maneira simples de se construir f de modo que se consiga uma iteração fractal satisfatória é através da utilização de potências de 2, fazendo com que em cada harmônico i, f seja igual a 2^i. Assim, podemos também definir a de uma forma bastante simples como sendo $\frac{1}{f}$.

$$T(p) = \sum_{i=0}^{N-1} \left| r(2^i \cdot p) \cdot \frac{1}{2^i} \right| \tag{2.18}$$

Uma característica importante da função turbulência é que a partir de uma determinada frequência, não ocorrem modificações significativas no sinal, o que passa a limitar o número de harmônicos. Na Fig. 15 isso ocorre no oitavo harmônico fazendo assim com que N seja oito.

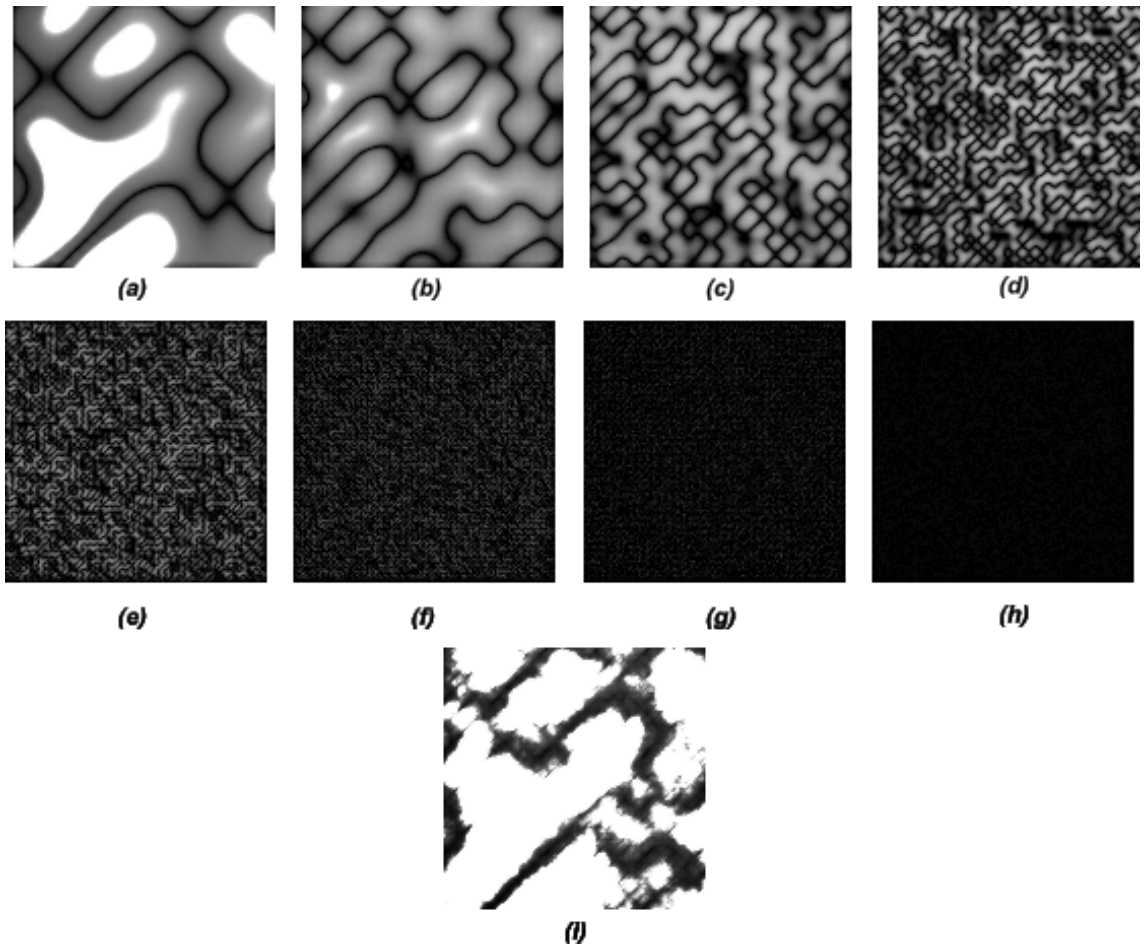


Figura 15: Função turbulência em 2D. (a)$\mathsf{f} = 1$, (b)$\mathsf{f} = 2$, (c)$\mathsf{f} = 4$, (d) $\mathsf{f} = 8$, (e)$\mathsf{f} = 16$, (f)$\mathsf{f} = 32$, (g)$\mathsf{f} = 64$, (h)$\mathsf{f} = 128$, (i) somatório de todos os harmônicos.

3 Geração do Ruído de Perlin

Tomando como base os conceitos de ruído e turbulência apresentados no capítulo anterior, serão apresentados neste capítulo meios para gerar-se o ruído de Perlin de maneira computacional.

3.1 Classe para Síntese de Ruído de Perlin

Baseado na proposta inicial deste trabalho, esta seção é destinada a descrever uma classe genérica capaz de gerar computacionalmente o ruído de Perlin.

Tal ruído pode ser gerado para `n` dimensões. No entanto, nesse trabalho serão relevantes as gerações em uma, duas e três dimensões, sendo que a implementação de cada método pode diferir de uma dimensão para outra.

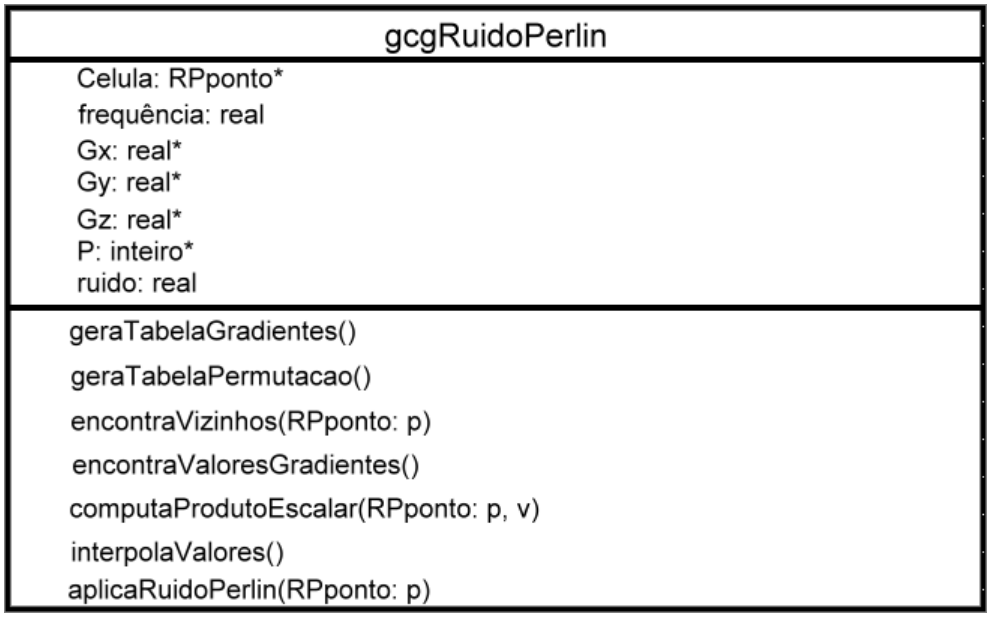


Figura 16: Classe genérica para o ruído de Perlin, independente da dimensão.

3.1.1 Ruído de Perlin em Uma Dimensão (1D)

A geração do ruído de perlin em uma dimensão é a mais fácil de ser entendida, uma vez que trabalha com uma quantidade de dados reduzida.

Para cada uma das dimensões em que o ruído de Perlin é utilizado, podemos definir a estrutura **RPponto** de um modo diferente. Em 1D, podemos definí-la com os seguintes atributos:

- x, representando a coordenada do ponto no espaço 1D;
- e, o valor de influência do ponto;

Em qualquer uma das dimensões que se está trabalhando, deve-se realizar antes de qualquer outra coisa o *pré-processamento* de alguns dados como a tabela de gradientes e a tabela de permutação.

A tabela de gradientes consiste de uma tabela responsável por guardar na memória o gradiente referente a cada um dos pontos do reticulado. Para se representar tal estrutura de dados, utiliza-se um vetor do tipo real $\boldsymbol{G}$ de n posições, onde n representa a quantidade de posições do reticulado. Perlin (EBERT et al., 2003) propõe o uso de um reticulado de 256 posições para qualquer dimensão, fazendo assim com que n seja igual a 256.

Para o preenchimento dos valores de $\boldsymbol{G}$, deve-se calcular randomicamente um valor de gradiente para cada posição do vetor. Esse valor de gradiente $\bigtriangledown$, pode ser calculado da seguinte forma:

$$\bigtriangledown = \frac{rand()}{(RM \cdot 0.5)} - 1, \tag{3.1}$$

onde *rand()* é uma função responsável por gerar um número aleatório qualquer no intervalo $[0, RM]$. Logo, $\bigtriangledown_{\mathsf{i}}$ irá gerar o valor do gradiente correspondente ao vetor $\boldsymbol{G}$ na posição i, com i variando de 0 a 255.

O pré-processamento da tabela de gradientes é de extrema importância para o desempenho final do algoritmo, uma vez que o cálculo de cada gradiente em tempo de execução acarretaria em um alto ganho de custo computacional.

A geração da tabela de permutação é outra parte extremamente importante na etapa de pré-processamento. Ela é um dos pontos do algoritmos responsável por ajudar a garantir a pseudo-aleatoriedade do ruído de Perlin uma vez que, ela auxilia na indexação de um único gradiente a cada um dos pontos do reticulado.

Pode ser vista como um vetor de inteiros $\boldsymbol{P}$ de n posições, uma vez que n representa a quantidade de posições do reticulado. Cada posição de $\boldsymbol{P}$ contém um índice da tabela de gradientes. Ela pode ser gerada de acordo com o pseudo-código abaixo:

Algoritmo 1 Pseudo-código para construção da tabela de permutação

```
for i = 0 to n do
    P[i] ← i;
end for
for i = 0 to n do
    j ← rand() mod n;
    t ← P[i];
    P[i] ← P[j];
    P[j] ← t;
end for
```

onde o operador *mod* retorna o resto da divisão de dois valores inteiros.

Ao fim do pré-processamento, a classe encontra-se pronta para aplicar o ruído de Perlin a um ponto T qualquer.

O primeiro passo para se aplicar o ruído de Perlin a T, é enquadrar T dentro da frequência utilizada, bastando para isso multiplicar a coordenada $\mathbf{x}$ do ponto pelo valor da frequência. Tal frequência, é representada por um valor real sendo geralmente utilizada uma potência de 2.

Em seguida, é necessário se fazer o mapeamento do ponto, passando-o das coordenadas do mundo para as coordenadas do espaço discretizado pelo reticulado, da seguinte forma:

$$T.\mathbf{x} = T.\mathbf{x} \cdot \frac{\mathsf{n}}{DT_x},$$

onde DT_x representa o tamanho do domínio de T no eixo x.

Após isso, é necessário se encontrar os vizinhos de T. Assim como T, cada um de seus vizinhos é do tipo **RPponto**. Como se trata de uma dimensão, T irá possuir apenas dois vizinhos, que são obtidos do seguinte modo:

$$V_0.\mathbf{x} = \text{floor}(T.\mathbf{x});$$

$$V_1.\mathbf{x} = V_0.\mathbf{x} + 1,$$

onde *floor(a)* é uma função que irá retornar a parte inteira do parâmetro *a*.

Depois de descobertos os vizinhos de T, devemos descobrir o valor do gradiente de cada um deles. Para tanto, utiliza-se da tabela de gradientes e da tabela de permutação, procedendo-se da seguinte forma:

$$V_0.\mathbf{e} = \boldsymbol{G}[(V_0.\mathbf{x} \bmod \mathsf{n})];$$

$$V_1.\mathbf{e} = \boldsymbol{G}[(V_1.\mathbf{x} \bmod \mathsf{n})],$$

A quantidade que um vizinho V irá influenciar sobre o ruído total de T, é resultado do produto da distância entre T e V pelo valor do gradiente de V:

$$V.\mathbf{e} = V.\mathbf{e} \cdot (V.\mathbf{x} - T.\mathbf{x});$$

Logo, para se obter o valor do ruído de T, basta se interpolar os valores de influência dos vizinhos através de uma interpolação linear, tendo como fator interpolante um dos dois polinômios sugeridos por Perlin. No pseudo-código abaixo tal polinômio é representado pelo termo t.

Algoritmo 2 Pseudo-código para a função interpolaçãoLinear(**RPponto**: T, V_1, V_2)

$f \leftarrow (T.\mathbf{x} - floor(T.\mathbf{x}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$ruido \leftarrow (1 - t) \cdot V_1.\mathbf{e} + t \cdot V_2.\mathbf{e}$;
return ruido;

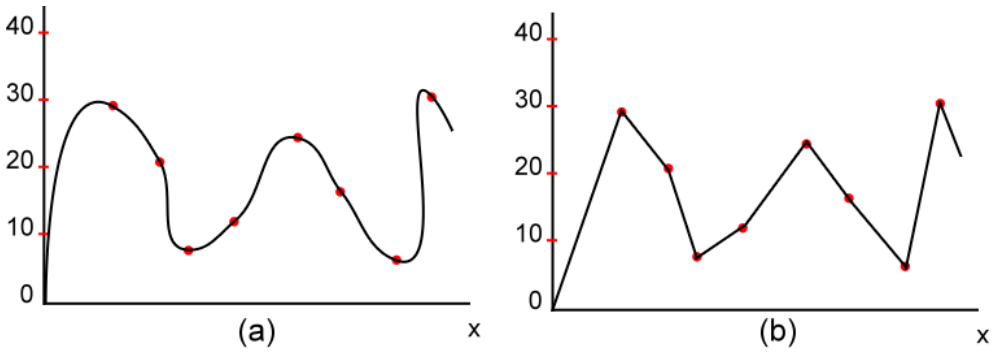


Figura 17: Ruído de Perlin em 1D. (a)Pontos interpolados utilizando um polinômio de grau três. (b)Pontos interpolados utilizando um polinômio de primeiro grau.

O próximo passo é se multiplicar o valor de ruído obtido através da interpolação pela amplitude da função, escalando desse modo o ruído dentro de um limiar definido pela amplitude. Na maioria das vezes uma maneira comum de se definir a amplitude é tomando-a como $\frac{1}{\mathsf{f}}$, onde f é a frequência utilizada.

Por fim, basta somar-se o valor do sinal original de T com o ruído obtido.

3.1.2 Ruído de Perlin em Duas Dimensões (2D)

O ruído de Perlin 2D pode ser usado para se perturbar uma função qualquer em 2D, como a imagem da Fig. 18, sendo sua aplicação muito semelhante à aplicação em 1D.

Mesmo assim, alguns pontos merecem destaque para essa dimensão.

Figura 18: Uma imagem é um ótimo exemplo de uma função 2D qualquer. O ponto T mostra dos valores do ponto referenciado na imagem.

Em 2D a estrutura **RPponto** possui um atributo a mais do que em 1D, sendo formada pelos seguintes atributos:

- **x**, representando a coordenada x do ponto no espaço 2D;
- **y**, representando a coordenada y do ponto no espaço 2D;
- **e**, o valor de influência do ponto;

Como em 1D, é necessário se realizar a etapa de pré-processamento, carregando-se para a memória a tabela de gradientes e a tabela de permutação. Ambas as tabelas são estruturadas e preenchidas exatamente como em 1D.

Para a aplicação do ruído de Perlin a um ponto T qualquer, também deve-se ajustar o ponto à frequência f. Isso é feito multiplicando-se cada uma das coordenadas de T pelo valor da frequência.

A seguir, T deve ser mapeado para o espaço discretizado pelo reticulado:

$$T.\mathbf{x} = T.\mathbf{x} \cdot \frac{\mathsf{n}}{DT_x};$$
$$T.\mathbf{y} = T.\mathbf{y} \cdot \frac{\mathsf{n}}{DT_y},$$

onde DT_x e DT_y representam o tamanho do domínio de T nos eixos x e y respectivamente. Em uma imagem isso pode ser entendido como sendo as dimensões de largura e altura respectivamente.

Após ser mapeado, como mostrado na Fig. 19, o ponto irá se encontrar dentro de uma das células do reticulado, tornando necessária a identificação da célula em que o ponto

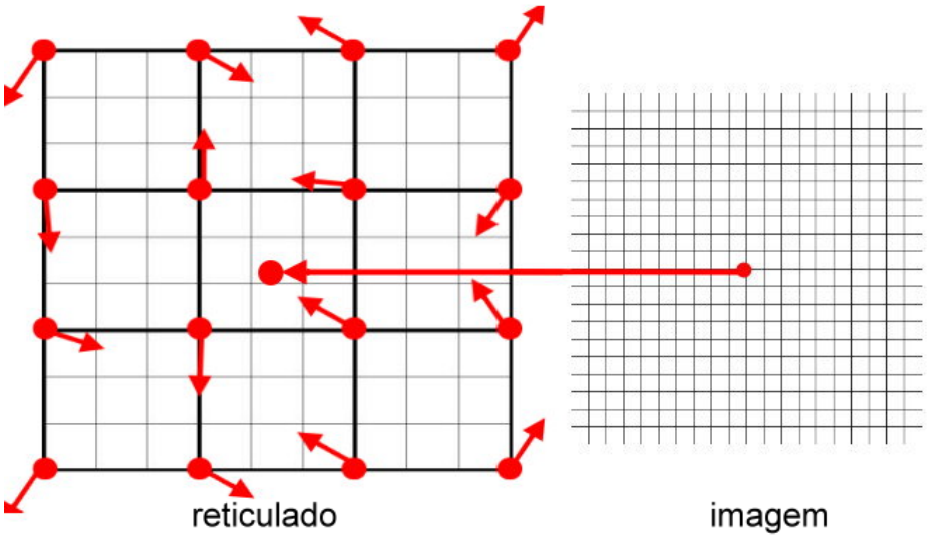


Figura 19: Exemplo do mapeamento de um ponto da imagem no espaço discretizado pelo reticulado.

foi mapeado. Para isto, devem ser encontrados todos os vizinhos de T. Nesse caso em particular pode-se definir a estrutura de uma célula como sendo formada por um vetor de quatro posições do tipo **RPponto**. A célula é composta da seguinte forma:

$$
\begin{aligned}
\boldsymbol{celula}[0].\mathbf{x} &= \text{floor}(T.\mathbf{x}) \quad ; \\
\boldsymbol{celula}[0].\mathbf{y} &= \text{floor}(T.\mathbf{y}) \quad ; \\
\\
\boldsymbol{celula}[1].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1; \\
\boldsymbol{celula}[1].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} \quad ; \\
\\
\boldsymbol{celula}[2].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} \quad ; \\
\boldsymbol{celula}[2].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1; \\
\\
\boldsymbol{celula}[3].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1; \\
\boldsymbol{celula}[3].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1;
\end{aligned}
$$

A influência que cada vizinho exerce isoladamente sobre o ponto T, é resultado de um produto escalar entre o vetor gradiente (vetores esses que são representados em 2D na Fig. 20) e o vetor que liga T a um de seus vizinhos V, sendo definida como:

$$V.\mathbf{e} = (\mathsf{GV} \cdot (V.\mathbf{x} - T.\mathbf{x})) + (\mathsf{GV} \cdot (V.\mathbf{y} - T.\mathbf{y})),$$

onde GV representa o valor do gradiente associado a V e é definido utilizando-se as tabelas de gradiente e de permutação do seguinte modo:

$$\mathsf{GV} = \boldsymbol{G}[(V.\mathbf{x} + \mathrm{P}[V.\mathbf{y} \bmod \mathsf{n}]) \bmod \mathsf{n}];$$

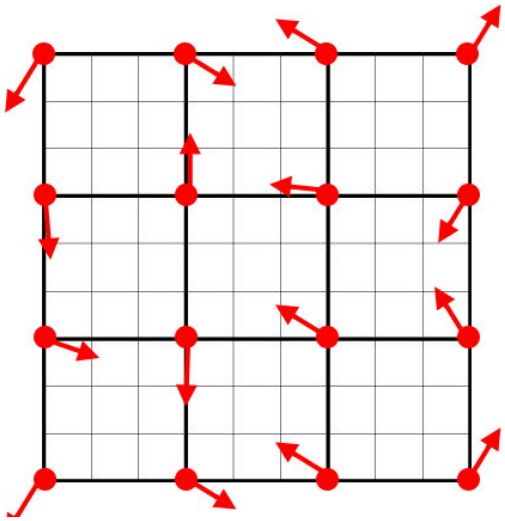


Figura 20: Representação de um reticulado 2D de 4 x 4 com o vetor gradiente (seta vermelha) referente a cada posição do reticulado.

De posse da influência que cada vizinho exerce sobre o ponto, o valor de ruído total sobre T é obtido através da interpolação bi-linear entre os valores de influência de todos os vizinhos como visto no código abaixo:

Algoritmo 3 Pseudo-código para a função interpolaçãoBiLinear(**RPponto**: T, V_1, V_2, V_3, V_4)

$f \leftarrow (T.\mathbf{x} - floor(T.\mathbf{x}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$temp1 \leftarrow (1 - t) \cdot V_1.\mathbf{e} + t \cdot V_2.\mathbf{e}$;
$temp2 \leftarrow (1 - t) \cdot V_3.\mathbf{e} + t \cdot V_4.\mathbf{e}$;
$f \leftarrow (T.\mathbf{y} - floor(T.\mathbf{y}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$ruido \leftarrow (1 - t) \cdot temp1 + t \cdot temp2$;
return $ruido$;

Em 2D também é necessário se mapear o valor de ruído obtido dentro do limiar imposto pela amplitude, exatamente como em 1D.

Por fim, basta somar-se o valor do sinal original de T ao valor de ruído obtido.

3.1.3 Ruído de Perlin em Três Dimensões (3D)

O ruído de Perlin 3D segue basicamente os mesmos passos da aplicação do ruído em 2D, porém, ocorrem algumas diferenças que devem ser levadas em consideração.

A primeira delas é na definição da estrutura **RPponto**. Ela possui um atributo a mais do que em 2D sendo composto da seguinte maneira:

- **x**, representando a coordenada x;
- **y**, representando a coordenada y;
- **z**, representando a coordenada z;
- **e**, o valor de influência do ponto;

Outra diferença que deve ser levada em consideração, é no ajuste do ponto ao domínio da frequência. Ao invés de multiplicarmos apenas as coordenadas **x** e **y**, é necessário se multiplicar também a coordenada **z** pelo valor da frequência.

Para se mapear o ponto no reticulado, deve-se proceder da seguinte forma:

$$
\begin{aligned}
T.\mathbf{x} &= T.\mathbf{x} \cdot \frac{\mathsf{n}}{DT_x};\\
T.\mathbf{y} &= T.\mathbf{y} \cdot \frac{\mathsf{n}}{DT_y},\\
T.\mathbf{z} &= T.\mathbf{z} \cdot \frac{\mathsf{n}}{DT_z},
\end{aligned}
$$

onde DT_x, DT_y e DT_z representam o tamanho do domínio de T nos eixos x, y e z respectivamente.

Quando se trabalha no espaço 3D, cada célula do reticulado é composta de oito pontos. Isso faz com que o atributo célula da classe também seja diferente do usado em 2D. Célula agora, é definido como um vetor de oito posições do tipo **RPponto**. Logo, depois de mapeado no reticulado, um ponto qualquer T irá possuir oito vizinhos, os quais são definidos da seguinte forma:

$$
\begin{aligned}
\boldsymbol{celula}[0].\mathbf{x} &= \text{floor}(T.\mathbf{x}) \quad ;\\
\boldsymbol{celula}[0].\mathbf{y} &= \text{floor}(T.\mathbf{y}) \quad ;\\
\boldsymbol{celula}[0].\mathbf{z} &= \text{floor}(T.\mathbf{z}) \quad ;\\
\\
\boldsymbol{celula}[1].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1;\\
\boldsymbol{celula}[1].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} \quad ;\\
\boldsymbol{celula}[1].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} \quad ;\\
\\
\boldsymbol{celula}[2].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} \quad ;
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{celula}[2].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} \quad ; \\
\boldsymbol{celula}[2].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} + 1; \\
\\
\boldsymbol{celula}[3].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1; \\
\boldsymbol{celula}[3].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} \quad ; \\
\boldsymbol{celula}[3].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} + 1; \\
\\
\boldsymbol{celula}[4].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} \quad ; \\
\boldsymbol{celula}[4].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1; \\
\boldsymbol{celula}[4].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} \quad ; \\
\\
\boldsymbol{celula}[5].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1; \\
\boldsymbol{celula}[5].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1; \\
\boldsymbol{celula}[5].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} \quad ; \\
\\
\boldsymbol{celula}[6].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} \quad ; \\
\boldsymbol{celula}[6].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1; \\
\boldsymbol{celula}[6].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} + 1; \\
\\
\boldsymbol{celula}[7].\mathbf{x} &= \boldsymbol{celula}[0].\mathbf{x} + 1; \\
\boldsymbol{celula}[7].\mathbf{y} &= \boldsymbol{celula}[0].\mathbf{y} + 1; \\
\boldsymbol{celula}[7].\mathbf{z} &= \boldsymbol{celula}[0].\mathbf{z} + 1;
\end{aligned}
$$

Assim como em duas dimensões, a influência de um vizinho V sobre T é resultado do produto vetorial entre o gradiente correspondente a V e o vetor que liga V à T:

$$V.\mathbf{e} = (\mathsf{GV} \cdot (V.\mathbf{x} - T.\mathbf{x})) + (\mathsf{GV} \cdot (V.\mathbf{y} - T.\mathbf{y})) + (\mathsf{GV} \cdot (V.\mathbf{z} - T.\mathbf{z})),$$

sendo GV dado por:

$$\mathsf{GV} = \boldsymbol{G}[(V.\mathbf{x} + \mathrm{P}[(V.\mathbf{y} + \mathrm{P}[V.\mathbf{z}])\mathrm{mod}\ \mathsf{n}])\ \mathrm{mod}\ \mathsf{n}];$$

O valor total do ruído sobre T em 3D, será obtido por uma interpolação tri-linear baseada nos valores de influência dos oito vizinhos da célula em que T se localiza.

As partes subsequentes à interpolação, são a inclusão do ruído no limiar da amplitude e adição do valor obtido pela interpolação ao valor inicial do sinal, exatamente igual à mesma etapa em 2D.

Algoritmo 4 Pseudo-código para a função interpolaçãoTriLinear(**RPponto**: T, V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8) (BAJAJ et al., 1994)

$f \leftarrow (T.\mathbf{x} - floor(T.\mathbf{x}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$temp1 \leftarrow V_1.\mathbf{e} + t \cdot (V_2.\mathbf{e} - V_1.\mathbf{e})$;
$temp2 \leftarrow V_3.\mathbf{e} + t \cdot (V_4.\mathbf{e} - V_3.\mathbf{e})$;
$temp3 \leftarrow V_5.\mathbf{e} + t \cdot (V_6.\mathbf{e} - V_5.\mathbf{e})$;
$temp4 \leftarrow V_7.\mathbf{e} + t \cdot (V_8.\mathbf{e} - V_7.\mathbf{e})$;
$f \leftarrow (T.\mathbf{y} - floor(T.\mathbf{y}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$temp5 \leftarrow temp1 + t \cdot (temp3 - temp1)$;
$temp6 \leftarrow temp2 + t \cdot (temp4 - temp2)$;
$f \leftarrow (T.\mathbf{z} - floor(T.\mathbf{z}))$;
$t \leftarrow (6 \cdot f^5) - (15 \cdot f^4) + (10 \cdot f^3)$;
$ruido \leftarrow temp5 + t \cdot (temp6 - temp5)$;
$return\ ruido$;

3.2 Classe para Síntese de Turbulência

De posse de uma classe para realizar a síntese de ruído de Perlin, uma classe para se aplicar a função turbulência a um ponto torna-se de fácil implementação, uma vez que a função turbulência é a aplicação da função ruído em diferentes escalas. Cada uma dessas aplicações representa uma *oitava*, sendo que, a função ruído possui apenas uma única oitava, ao contrário da função turbulência, que pode possuir inúmeras oitavas.

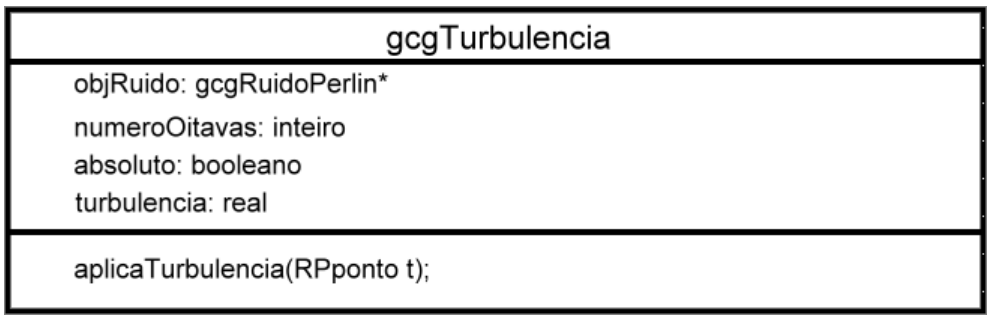


Figura 21: Classe abstrata representando a função turbulência.

Como a função turbulência tem como suporte a função ruído de Perlin, a geração de turbulência em uma, duas ou três dimensões depende exclusivamente da função de ruído que se está utilizando.

O atributo **numeroOitavas** pertencente à classe **gcgTurbulencia** representa a quantidade de vezes que a função de ruído deve ser aplicada, sendo que, em cada uma das aplicações, a frequência deve possuir um valor diferente. Na maioria das vezes a frequência é variada em função de um fator de 2, ou seja, a cada iteração a frequência é sempre o

dobro da frequência da iteração anterior.

Dentro da classe **gcgTurbulencia** o atributo **absoluto** serve para determinar se os valores gerados a cada iteração da turbulência devem ser tomados como absolutos (no caso desse atributo ser verdadeiro) ou, se o sinal do valor deve ser levado em consideração (caso contrário).

A função aplicaTurbulêcia() basicamente realiza uma iteração **numeroOitavas** vezes da função de ruído. O pseudo-código de uma função turbulência própria para o espaço 3D é mostrado abaixo:

Algoritmo 5 Pseudo-código para a função aplicaTurbulencia3D(RPponto: T, real: frequenciaInicial)

```
turbu ← 0;
for i = 0 to numeroOitavas do
  if absoluto = verdadeiro then
    turbu ← turbu + ABS(aplicaRuidoPerlin3D(T.x, T.y, T.z, frequenciaInicial));
  else
    turbu ← turbu + (aplicaRuidoPerlin3D(T.x, T.y, T.z, frequenciaInicial));
  end if
  frequenciaInicial ← frequenciaInicial · 2;
end for
return turbu;
```

onde *ABS(a)* retorna o valor do módulo de *a*.

4 Aplicações do Ruído de Perlin

Com base na teoria e nos modelos computacionais apresentados, é possível se usar o ruído de Perlin para realizar diversos tipos de tarefas, indo desde a geração de um terreno até a geração de texturas de forma totalmente procedural. Nesse capítulo serão apresentadas algumas dessas aplicações juntamente com o tipo de resultado gerado.

4.1 Ruído de Perlin e Geração de Texturas

Além de seu uso para a geração de terrenos, o ruído de Perlin também é extremamente empregado para se realizar a geração de vários tipos de texturas procedurais muito empregadas em CG.

4.1.1 Textura de Mármore

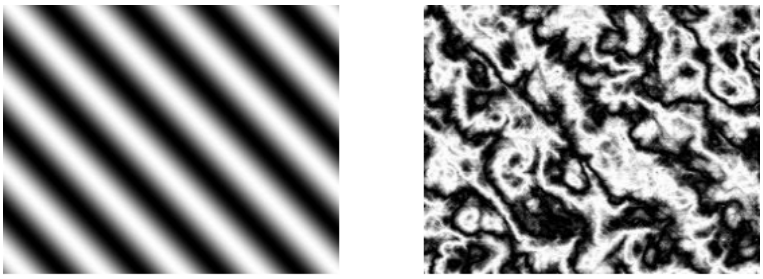
Um dos exemplos mais clássicos na utilização do ruído de Perlin para compor uma textura de forma procedural, é a geração de texturas com o aspecto de mármore.

O mármore pode ser conseguido através de uma perturbação de curvas senóides com a função turbulência como mostrado na Fig.22. Alguns valores importantes que devem ser controlados na geração são especificados por dois parâmetros: a frequência da função de ruído, e um *seed*, um número que inicializa a função rand() acima citada.

Assim, para se aplicar a textura de mármore a um ponto T qualquer de coordenadas (x,y) deve-se seguir a expressão:

$$sin(fatorMult \cdot (x + y + turbulencia(T)))$$

onde turbulencia(T) representa que foi aplicada a função de turbulência ao ponto T e *fatorMult* é um número real responsável por escalar a função seno. Graças ao termo fatorMult é possível se obter um controle sobre a quantidade de linhas diagonais geradas



(a) (b)

Figura 22: Textura de mármore gerada com os parâmetros: *seed* = 918, frequência = 0.02, escala = 3, numeroOitavas = 8, absoluto = verdadeiro. (a) mostra uma imagem da função senóide (b) mostra a função senóide perturbada pela função turbulência resultando no aspecto de mármore.

pela função seno que devem ser perturbadas para se conseguir o aspecto de mármore. Na Fig. 22 fatorMult foi usado com o valor de 0.1.

4.1.2 Textura de Madeira

Outro exemplo clássico na geração de texturas procedurais com o ruído de Perlin, é a geração de uma textura com a aparência de madeira. Ainda mais simples que a textura em forma de mármore, ela é obtida através da expressão:

$$g = noise(T) \cdot 20;$$
$$grand = g - floor(g);$$

onde noise(T) representa o ruído de Perlin aplicado ao ponto T.

Porém, mesmo utilizando uma frequência muito baixa como 0.001, por exemplo, a quantidade de curvas projetadas na superfície da textura continuava sendo muito alta como visto na Fig. 23.

Se a frequência utilizada para gerar o ruído for ainda menor, não mais se consegue obter as curvas necessárias para se gerar a aparência de madeira, como mostrado na Fig. 24.

Assim, a solução encontrada empiricamente nesse trabalho para se contornar tal problema, foi o escalamento do valor final de ruído, utilizando a própria frequência de geração, conseguindo resultado exibido na Fig. 25.

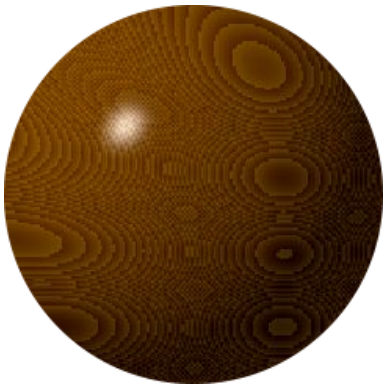


Figura 23: Textura de madeira gerada com os parâmetros: *seed* = 129675, frequência = 0.001, escala = 1. A quantidade de curvas na superfície da esfera ainda é muito alta.

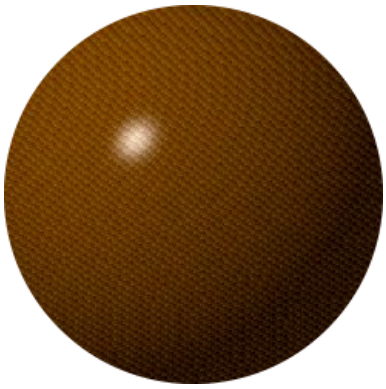


Figura 24: Textura de madeira gerada com os parâmetros: *seed* = 129675, frequência = 0.0001, escala = 1. A aparência das curvas na superfície da esfera não são mais capazes de simular a aparência de madeira.

4.2 Geração de Terrenos

Em CG a geração de terrenos pode ser utilizadas nas mais diversas aplicações, indo desde um simulador de voo à construção de um sistema planetário.

Um terreno em CG é composto por dois componentes:

- mapa de alturas
- textura

O mapa de alturas tem a função de determinar as elevações em cada ponto do terreno, enquanto que a textura, é responsável pela modelagem das cores em cada ponto do terreno.

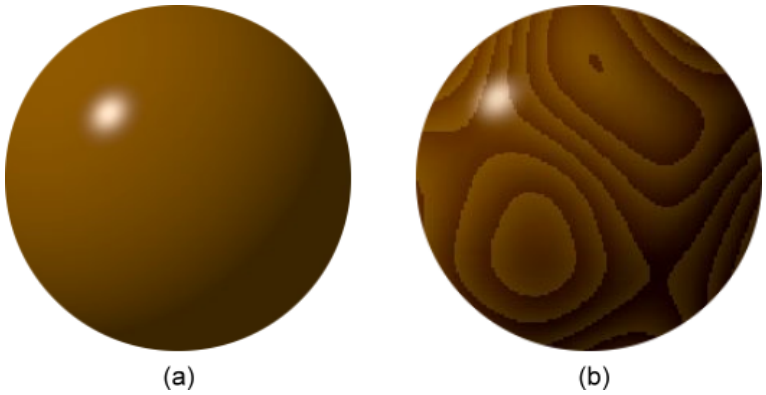


Figura 25: Textura de madeira gerada com os parâmetros: *seed* = 129675, frequência = 0.01, escala = 1, e escalando o valor de ruído obtido com a própria frequência utilizada para a geração do ruído. (a) mostra uma esfera antes da aplicação do ruído para a obtenção da textura. (b) mostra a mesma esfera depois de ser perturbada pela função de ruído de modo a se obter uma textura de madeira.

4.2.1 Geração de Mapas de Alturas

Do ponto de vista computacional o mapa de alturas pode ser visto como uma matriz **HM** de m linhas por n colunas, onde em cada posição $HM_{[a][b]}$ é guardado um valor correspondente à altura naquele ponto. Tais valores quando se trabalha com dados de oito bits, geralmente variam de 0 à 255, onde 0 representa que a altitude naquele ponto é nula, e 255 representa que a altitude naquele ponto é máxima. Um exemplo de mapa de alturas pode ser visto na Fig. 26.

Como o mapa de alturas é uma matriz bi-dimensional, pode-se utilizar da função de ruído de Perlin em duas dimensões para se perturbar tal matriz, gerando para cada posição da mesma, valores pseudo-aleatórios.

Figura 26: Mapa de alturas gerado com a função ruído de Perlin com: frequência = 0.01, *seed* = 1234, escala = 1.

Para tornar possível um controle ainda maior sobre a geração do mapa de alturas pode-se adicionar mais um parâmetro à geração, chamado *escala*, o qual deve ser multiplicado pelo valor gerado para o ruído, fazendo assim com q tenha-se um controle maior sobre as alturas geradas (intensidade dos tons de cinza gerados), como observado na Fig. 27. Já a frequência e o *seed* são responsáveis por coordenar a distribuição do ruído por todo o mapa de alturas.

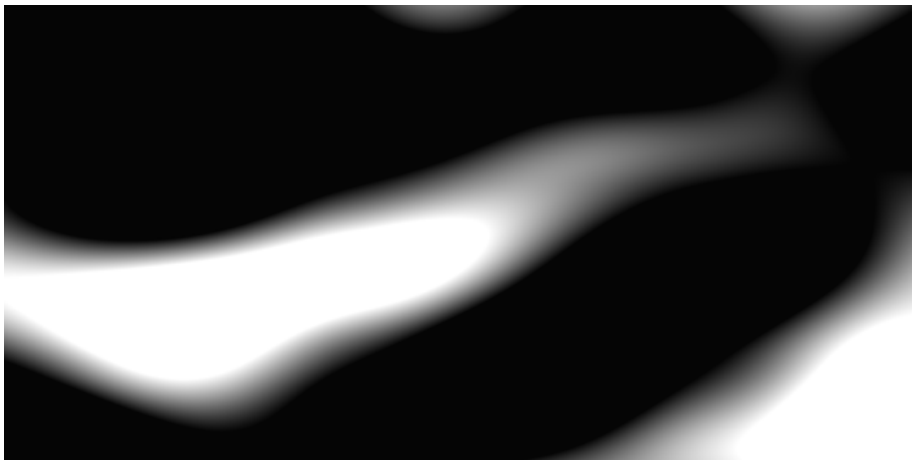


Figura 27: Mapa de alturas gerado com a função ruído de Perlin com: frequência = 0.01, *seed* = 1234 e escala = 10.

No entanto, a função de ruído sozinha não é suficiente para se modelar de forma satisfatória um mapa de alturas. Para tal, é necessário utilizar-se da função de turbulência, a qual é capaz de obter uma aparência um pouco melhor para o terreno a ser gerado.

Tal função acrescenta mais dois parâmetros ao controle da geração de mapas de alturas. O primeiro deles é conhecido como *oitava*, ou como foi definido na classe abstrata para turbulência (**gcgTurbulencia**), **numeroOitavas**. Esse nome provém do fato de que a partir da oitava iteração, com a frequência sempre dobrando, não se consegue uma modificação considerável de uma imagem gerada para a sua sucessora. Ele é responsável por definir a quantidade de aplicações do ruído de Perlin a cada ponto.

O outro parâmetro denominado *absoluto* define se serão aceitos apenas valores absolutos para turbulência ou não. A função de turbulência proposta inicialmente por Perlin (PERLIN, 1985) tomava que os valores gerados pela função turbulência sempre deveriam ser tomados por seus valores absolutos, onde os resultados podem ser vistos na Fig. 29. Porém, para a modelagem de mapas de alturas de forma procedural, é interessante se manter esse grau de liberdade, para se conseguir mapas como o aspecto da Fig. 28.

Outra diferença que também deve ser destacada para a função de ruído é que na função de turbulência determina-se apenas a frequência da primeira oitava, sendo que a

frequência vai sempre dobrando nas oitavas seguintes.

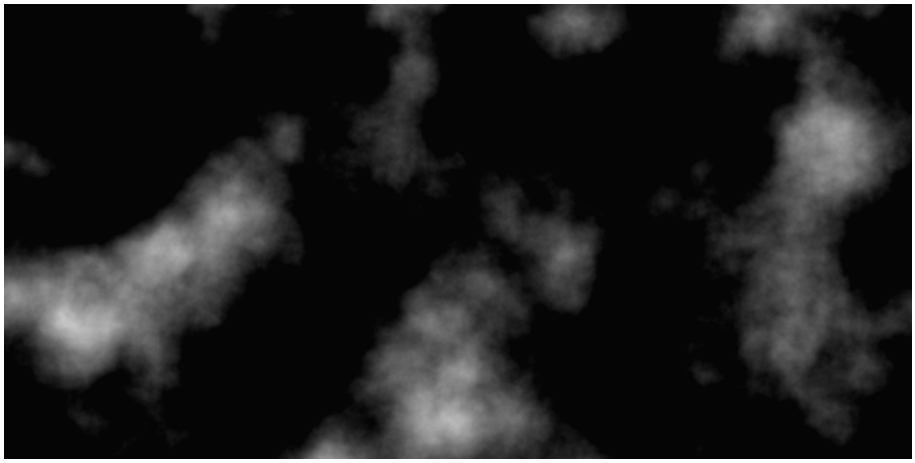


Figura 28: Mapa de alturas gerado com a função turbulência com: frequência inicial= 0.01, *seed* = 1234, escala = 3, numeroOitavas = 8 e absoluto = falso.

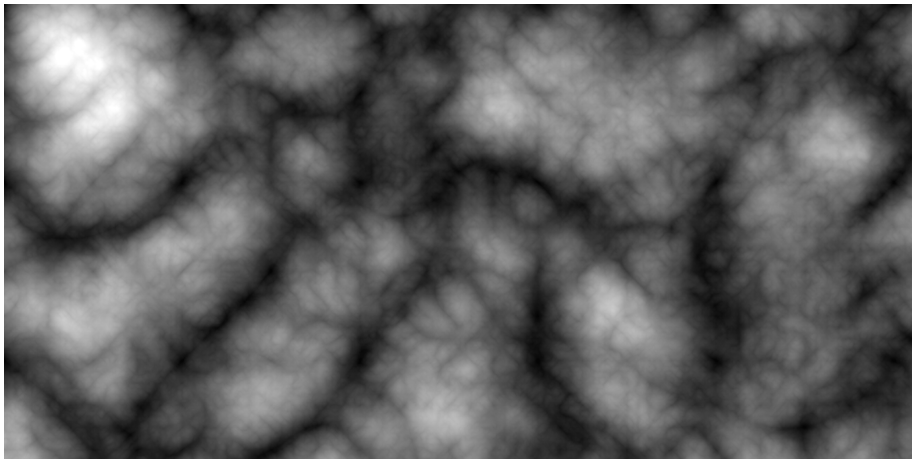


Figura 29: Mapa de alturas gerado com a função turbulência com: frequência inicial= 0.01, *seed* = 1234, escala = 3, numeroOitavas = 8 e absoluto = verdadeiro.

Para a geração de mapas de alturas que sejam exibidos em um plano, a aplicação da função turbulência em 2D mostra resultados satisfatórios. Porém, ao se tentar mapear um terreno gerado com a função turbulência 2D em uma esfera (como é necessário na geração de um planeta por exemplo), os resultados não são nada agradáveis, pois, é gerada uma grande distorção nos pólos, além de uma falta de paridade entre as alturas do extremo leste e do extremo oeste do terreno. Para compensar tais problemas, uma das soluções encontradas é se aplicar a função de turbulência em 3D na geração do mapa de alturas (CELES et al., 2003). Isso corrige os problemas acima citados, como pode ser visto na Fig. 30, fazendo com que o mapa de alturas seja perfeitamente mapeado sobre a superfície de uma esfera.

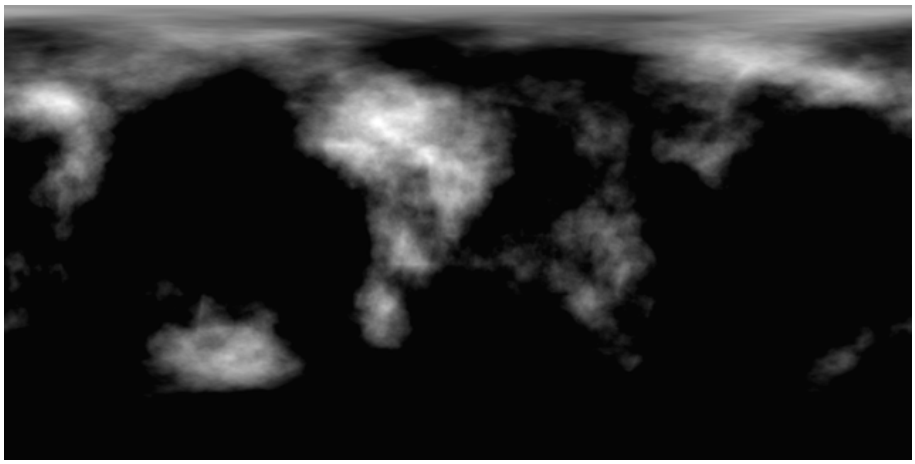


Figura 30: Mapa de alturas gerado com a função turbulência 3D com: frequência inicial= 0.01, *seed* = 1, escala = 5, numeroOitavas = 8 e absoluto = falso. Pode-se perceber um alongamento nos polos e um perfeito casamento entre as alturas de todo extremo leste com o extremo leste da imagem, tudo isso para compensar as distorções ocorridas devido ao mapeamento do mapa em uma esfera.

Logo, com base nos tópicos exibidos até agora, uma maneira simples, porém eficiente de se gerar um mapa de alturas satisfatório para a criação de planeta, é combinando-se dois ou três mapas de alturas, a exemplo da Fig. 31, gerados com turbulência 3D (com absoluto = falso), de modo a compor um único mapa final.

4.2.2 Geração de Textura Para o Terreno

A textura utilizada no terreno é de extrema importância, uma vez que é responsável por atribuir uma aparência visivelmente agradável ao terreno.

Tal textura, assim como o mapa de alturas, precisa ser gerada de forma procedural, para com isso evitar-se distorções na imagem. Um textura gerada de forma procedural se adapta a superfície do terreno não importando o quanto ela varie.

Um modo simples porém eficiente de se compor a textura do terreno é baseando-se no mapa de alturas. Pode-se dividir o mapa em intervalos de altitude com o mesmo tamanho, e a partir daí determinar-se que cada intervalo possui uma textura básica apropriada. Ao fazer referência a texturas básicas, tem-se em mente texturas de elementos isolados da natureza como água ou rocha.

Cada intervalo dentro do mapa de alturas vai possuir um limite superior que corresponde à maior altura daquele intervalo, fato que é de extrema importância para a geração de uma textura um pouco mais realista.

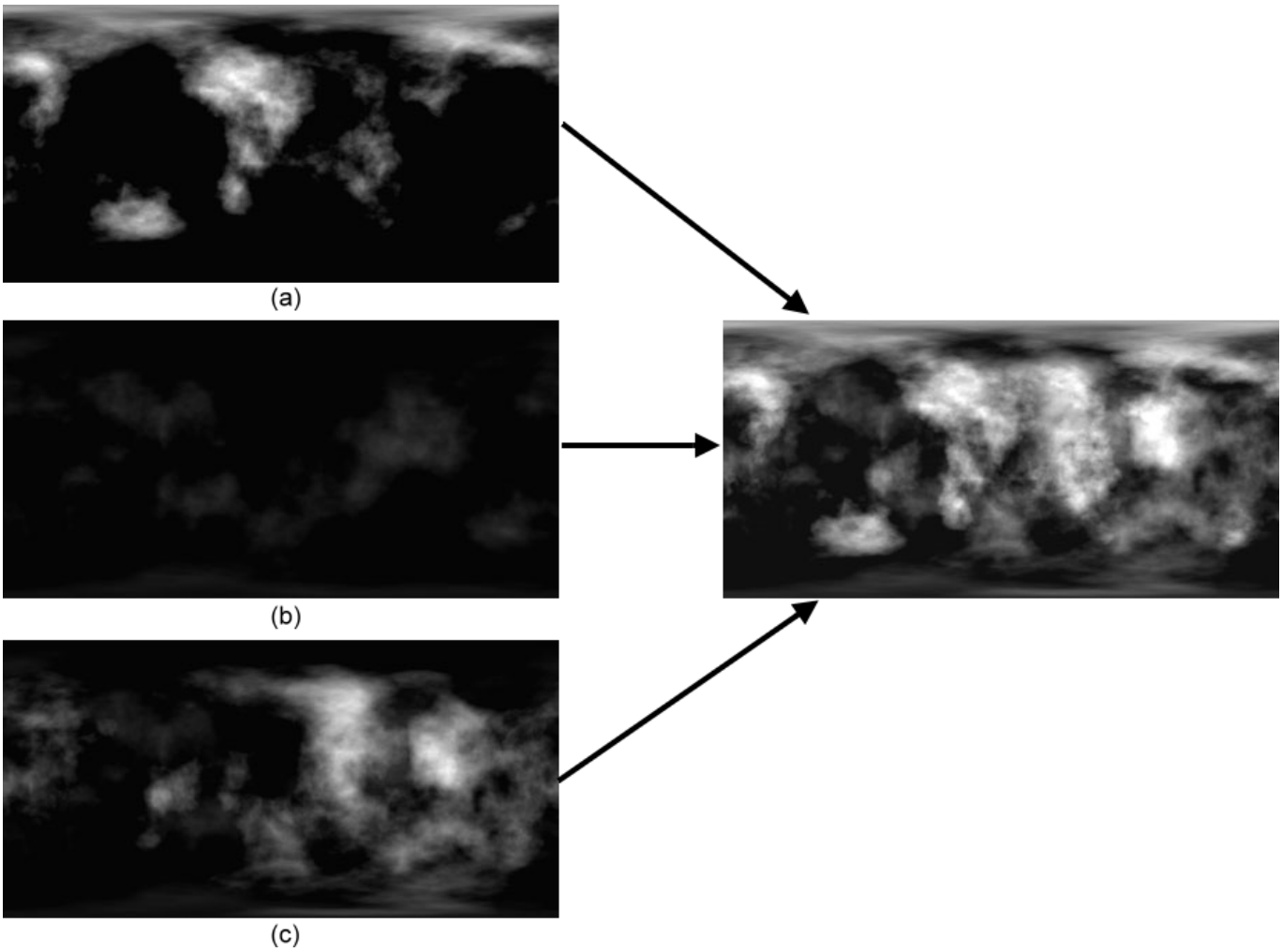


Figura 31: Mapa de alturas formado da composição de três outros mapas. (a)turbulência 3D: frequência inicial= 0.01, *seed* = 1, escala = 5, numeroOitavas = 8 e absoluto = falso. (b)turbulência 3D: frequência inicial= 0.01, *seed* = 222, escala = 1, numeroOitavas = 8 e absoluto = falso. (c)turbulência 3D: frequência inicial= 0.009, *seed* = 9876, escala = 3, numeroOitavas = 8 e absoluto = falso.

Para compor a textura de um terreno de forma procedural, basta que para cada posição do mapa de alturas, você realize os procedimentos descritos no pseudo-código abaixo: onde PTB representa a porcentagem de uma das texturas básicas, TI representa o tamanho do intervalo em que o mapa de alturas foi dividido, A representa a altura no ponto em questão no mapa de alturas, $LSTB$ representa o limiar superior para a textura básica da iteração corrente, NC representa a nova cor para a posição em questão da textura (a qual corresponde à mesma posição no mapa de alturas) e CTB representa a cor na textura básica da iteração.

Uma observação muito importante a ser feita nesse ponto é que caso as texturas básicas não possuam as mesmas dimensões do mapa de alturas é necessário se fazer um mapeamento da coordenada no domínio mapa de alturas para o domínio das coordenadas

Algoritmo 6 Pseudo-código para a geração de textura procedural para o terreno

for todas as texturas basicas **do**
  $PTB \leftarrow (TI - ABS(A - LSTB)) \div TI$;
  $NC \leftarrow NC + CTB$;
end for

da textura básica em questão.

Assim, utilizando como exemplo um mapa de alturas dividido em cinco intervalos, cujo tamanho dos intervalos corresponde a 51 e tomando como texturas básicas as texturas de água, areia, grama, rocha e neve cujos limites superiores são dados respectivamente por 51, 102, 153, 204 e 255 juntamente com um mapa de alturas dado podemos obter um resultado conforme a figura Fig. 32.

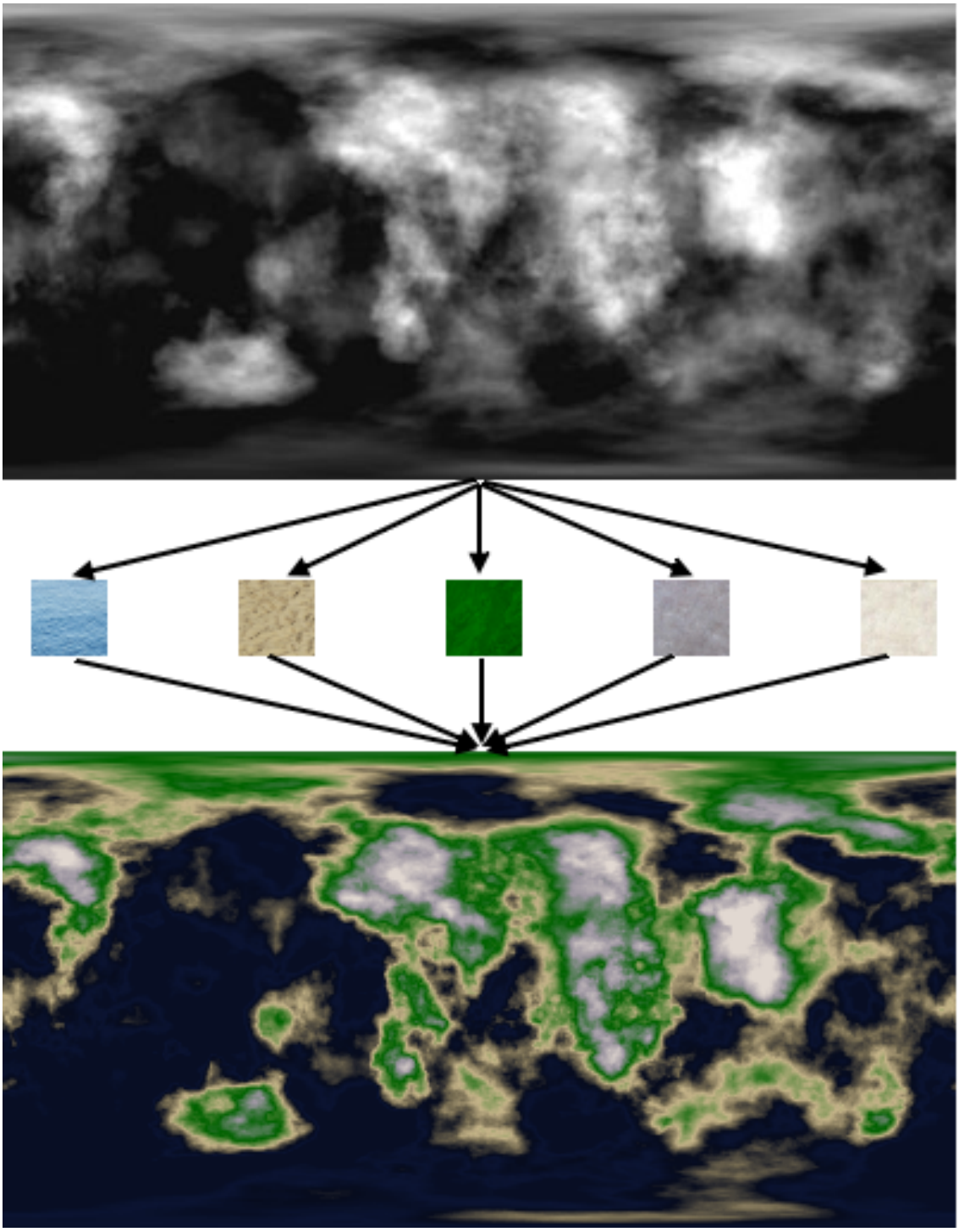


Figura 32: Geração de textura para um terreno partindo de um mapa de alturas e de cinco texturas basicas.

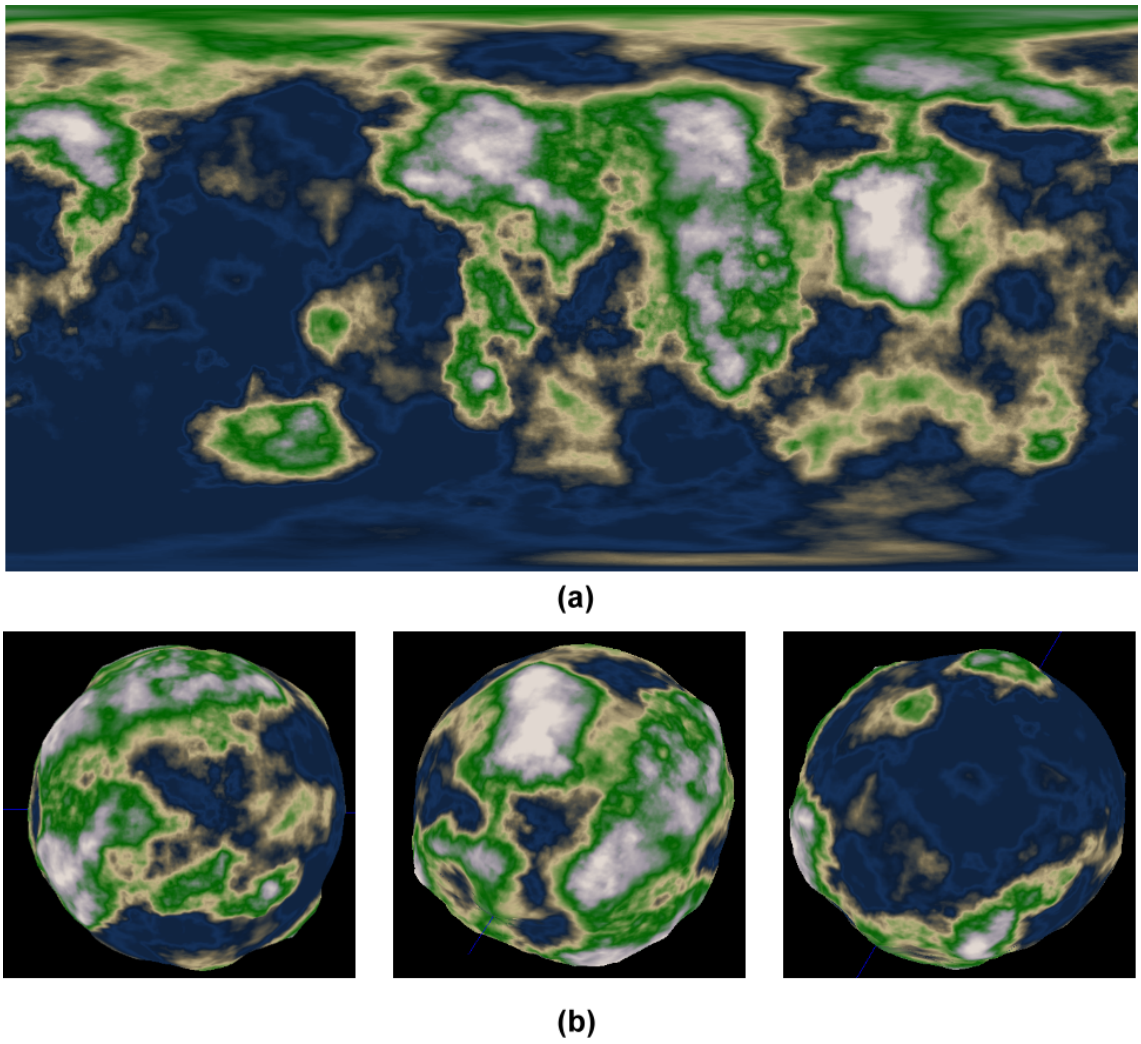


Figura 33: Terreno final obtido. (a) Mostra o terreno gerado em modo planar. (b) Mostra o mapemaento do terreno em uma esfera para a geração de um planeta.

5 Conclusão

Foram apresentados neste trabalho conceitos formais ligados a ruído e principalmente ao ruído de Perlin, além de formas para se aplicar tais conceitos à modelagem procedural de um terreno.

Como um dos objetivos do trabalho foi implementado um gerador de terrenos através das classes para geração de ruído e turbulência.

Após a implementação, pode-se observar que um dos pontos mais difíceis dentro da geração dos terrenos é o controle dos parâmetros de entrada. Isso porque é possível se controlar apenas a quantidade de elevações geradas bem como a intensidade da elevação. No entanto, a disposição do relevo é algo que só pode ser constatado empiricamente, o que gera grande demanda de tempo até se encontrar um terreno de aparência satisfatória.

Para se conseguir terrenos com aspecto realmente realista, ainda é necessário se introduzir outros tipos de efeitos como a simulação de erosão por exemplo.

O ruído de Perlin ainda pode ser explorado em outras vertentes da CG, permitindo a criação efeitos de extremamente interessantes como fogo, nuvens e fumaça.

Na parte de texturas ele também pode ser utilizado para criar muitas variações. Foram vistos aqui apenas os mais simples deles, mármore e madeira, mas há texturas de aparência extremamente realista, como pêlos, geradas através do ruído de Perlin . Em (PERLIN, 1989) podem ser vistos alguns dos mais interessantes exemplos.

Como trabalhos futuros propõe-se o estudo de métodos capazes de adicionar um melhoramento significativo ao gerador de terrenos desenvolvido. Tais melhoramentos podem ser desenvolvidos em diversas áreas: implementação de algoritmos de erosão e otimização da geração corrente para uma geração em tempo real são algumas das áreas propostas.

Referências

BAJAJ, C. et al. *Graphics Gems*. [S.l.]: Morgan Kaufmann Publishers, 1994.

BRITO, A. *Blender 3D - Guia do Usuário*. [S.l.]: Novatec Editora, 2006.

CELES, W. et al. *Graphics Programming Methods*. [S.l.]: Jenifer Niles, 2003.

CLUE, E. W. G. *Modelagem Procedimental para Visualização de Elementos da Natureza*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 1999.

EBERT, D. S. et al. *Texturing and Modeling - A Procedural Approach*. [S.l.]: Morgan Kaufmann Publishers, 2003.

GONZALES, R. C.; WOODS, R. E. *Digital Image Processing*. Second. [S.l.]: Prentice Hall, 1992.

OPPENHEIM, A. V.; SCHAFER, R. W.; BUCK, J. R. *Discrete-Time Signal Processing*. Second. [S.l.]: Prentice-Hall, 1999.

PERLIN, K. An image synthesizer. *SIGGRAPH*, v. 19, n. 3, p. 287–296, July 1985.

PERLIN, K. Hypertexture. *SIGGRAPH*, v. 23, n. 3, p. 253–261, July 1989.

PERLIN, K. Improving noise. *Computer Graphics*, v. 35, n. 3, p. 681–682, 2002.

PROAKIS, J. G.; MANOLAKIS, D. G. *Digital Sinal Processing - Principles, Algorithms and Aplications*. [S.l.]: Prentice-Hall, 1996.