Procedural Terrain Generation

Different Approaches in Terrain Generation

Diogo Emanuel da Silva Nogueira
a78957@alunos.uminho.pt

Mestrado Integrado em Engenharia Informática
Universidade do Minho - Departamento Informática, Braga

RESUMO

A criação de uma superfície geográfica tridimensional tem sido um campo de constante evolução ao longo dos anos, existindo cada vez mais uma procura de novas e melhoradas formas de realizar esse processo.
Este artigo tem como objetivo basilar discutir abordagens existentes para todo o processo do *Procedural Terrain Generation*. Isso engloba as muitas funções que são usadas para gerar "ruído" (algo que se provou excecionalmente útil em PTG e síntese de textura), bem como uma referência de outras abordagens utilizadas na criação de terreno. O artigo conclui com uma síntese do material discutido, dando também destaque das áreas para pesquisas futuras.

1. INTRODUÇÃO

A área informática que se foca em cuidar de grande parte dos aspetos visuais nas novas tecnologias é a Computação Gráfica. A mesma engloba um conjunto de técnicas que permitem a geração de imagens a partir de modelos computacionais de objetos reais, objetos imaginários ou de dados recolhidos por equipamentos na natureza. Importante ainda salientar que as imagens geradas possuem uma infinidade de aplicações, como por exemplo produzir animações, jogos e etc.

Neste artigo pretende-se avaliar as diferentes abordagens existentes no uso de algoritmos evolutivos durante processos de *Terrain Generation* em jogos.

Um dos aspetos mais importantes na maioria dos jogos é o terreno que o jogador irá percorrer. Criar um mundo de jogo virtual que envolve texturas e cenários adequados para um jogador explorar requer tempo e recursos, o que pode aumentar significativamente os custos de desenvolvimento.

O *Procedural Terrain Generation* pode ajudar a diminuir esse entrave através do uso de algoritmos pseudoaleatórios. O termo PTG destina-se então à criação de conteúdo por um sistema automatizado, ao invés de ser produzido manualmente. A geração quer do terreno, quer da textura pode ser feita através de algoritmos para produzir *noise*. Embora esse ruído possa parecer ser totalmente aleatório, é, na verdade, frequentemente o resultado de um conjunto pseudoaleatório de procedimentos conhecido como função de ruído. O *noise* produzido por essas funções é útil porque tem estrutura, ao invés de ser totalmente aleatório.

Além das funções de ruído, outras abordagens são utilizadas na criação do terreno, como o uso de *Heightmaps*, Agentes de *Software* e Técnicas de Modelagem de Erosão, técnicas estas que serão também abordadas/mencionadas ao longo do artigo.

2. PROCEDURAL TERRAIN GENERATION

A maioria das técnicas de PTG seguem um método consistente e sistemático para obter uma geometria semelhante às características naturais do terreno. O processo começa com uma forma geométrica básica representada por uma malha poligonal 3D.

Tradicionalmente, os algoritmos de PTG são aplicados a planos horizontais compostos por uma rede de vértices. Cada vértice recebe um valor de elevação relativo à sua altura acima deste plano. Inicialmente, todos os vértices recebem um valor de elevação zero e formam um terreno totalmente plano. Neste ponto, a malha é sistematicamente deformada pela introdução da variação pseudoaleatória ao valor de elevação de cada vértice por meio do uso de uma função matemática determinística. Todas as técnicas procedimentais de PTG diferem na função exata usada para calcular essa variação na elevação e nas formas

específicas pelas quais essa função pode ser parametrizada.

O resultado é uma geometria de terreno completamente nova, representada pela malha deformada com sucesso. Este processo está exemplificado na figura abaixo anexada [13].

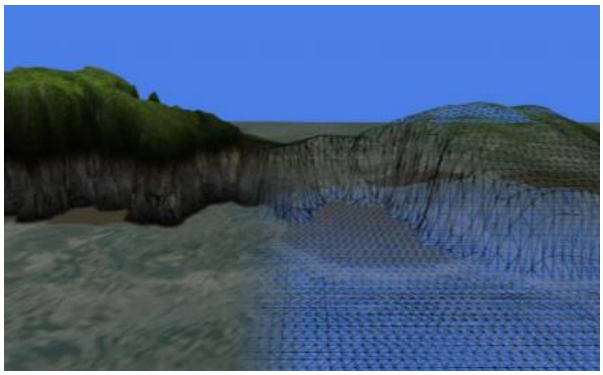

Figura 1 *Procedural Terrain Generation* com a *mesh* totalmente visível [13]

2.1. HEIGHTMAPS

Os dois tipos de apresentação digital de terreno mais comuns num espaço tridimensional são a Grade Bidimensional com um Mapa de Altura e Matriz Tridimensional onde o espaço é dividido em voxels (Um voxel representa um valor em um *gride* regular em um espaço tridimensional).

O tipo Voxel permite gerar estruturas de terreno complexas, mas requer mais espaço de armazenamento. As diferenças visuais são apresentadas na figura 2 [10].

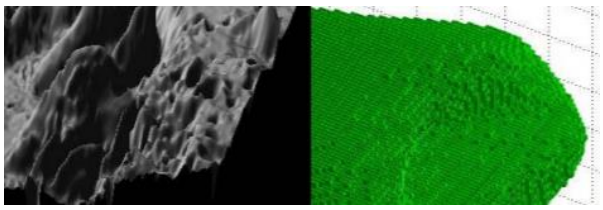

Figura 2 *Heightmaps* vs *Voxels* [10]

Heightmap é usado como uma estrutura de dados principal para armazenar o valor da elevação de cada vértice na geometria 3D da *mesh* na PTG. O mesmo usa uma rede bidimensional de valores numéricos para representar o valor de elevação de um vértice em coordenadas específicas.

Tradicionalmente, os *Heightmaps* são criados na forma de um arquivo de imagem, usando os valores de cor individuais de cada pixel para representar a altura do vértice correspondente na *mesh*. Pixéis completamente pretos representam vértices na elevação mais baixa possível na *mesh*, enquanto que pixéis completamente brancos representam o oposto.

Uma grande limitação deste método é o facto de cada pixel representar apenas um único valor de elevação, o que torna os *Heightmaps* incapazes de representar recursos de terreno mais complexos, como por ex. Cavernas ou Saliências, onde qualquer vértice localizado dentro dele teria de representar a altura do terreno na superfície e abaixo da mesma [12].

2.2. NOISE

Os algoritmos de ruído são usados para atribuir valores reais de elevação a cada pixel no *Heightmap* numa grande maioria das técnicas de PTG. Ao gerar *Heightmaps*, a entrada para essas funções de ruído são as coordenadas de cada vértice e a saída é o valor de elevação de cada vértice.

O método mais básico para produzir esses valores é simplesmente iterar por todo o *Heightmap* e atribuir valores aleatórios a cada pixel, conceito este conhecido como *White Noise*. Essa abordagem, no entanto, resulta em valores de elevação inconstantes e, consecutivamente, num terreno irregular.

A solução para este dilema está em fornecer um mecanismo por meio do qual cada vértice possa ter consciência da elevação dos vértices próximos. Isso permitiria que as restrições fossem colocadas em variações bruscas e garantiria que quaisquer transições na elevação fossem suaves e constantes.

Algoritmos de *noise* projetados com este conceito produzem o que é conhecido como ruído coerente. Na Figura 3 é ilustrada a diferença entre ruído branco e ruído coerente visualmente.

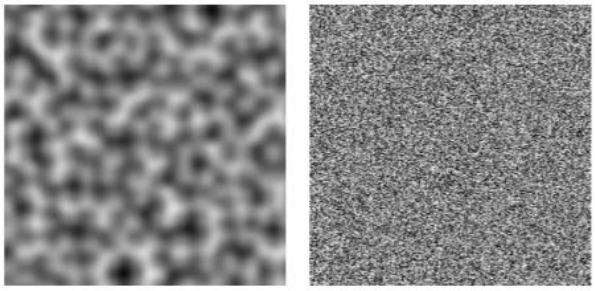

Figura 3 Ruído Branco vs Ruído Coerente [13]

No entanto o valor de ruído coerente apresenta algumas restrições:

- Qualquer valor de entrada tem o mesmo valor de saída;
- As mudanças no valor de entrada resultam em mudanças no valor de saída [2].

2.3. SPHERICAL TERRAINS

Quando as superfícies se tornam esféricas os métodos de *Heightmaps* e algoritmos de *noise* não funcionam, já que as *meshses* planas precisam apenas de terrenos de forma retangular de comprimento e largura limitados.

Ao escolher um terreno esférico, a principal preocupação é a distribuição uniforme dos vértices, uma vez que a densidade dos vértices numa determinada área determina a quantidade de detalhes geométricos que podem ser aplicados ali. Se uma malha concentra muitos vértices em uma área particular e menos em outras, o detalhe geométrico da malha não será uniforme [11].

Existem diversas maneiras de representar uma esfera num espaço 3D, sendo que a mais usual passa por utilizar o método de espelhar o sistema de coordenadas geográficas de latitude e longitude.
Apesar de ser um método simples, este método não resulta bem na aplicação de *Heightmaps* porque os vértices tornam-se mais densos e concentrados perto dos polos.

Outra abordagem está no uso de uma forma poliédrica regular e na sua transformação para se aproximar de uma esfera (quase) perfeita. Poliédricos regulares são formas geométricas tridimensionais compostas por faces de polígonos regulares, como triângulos equiláteros ou quadrados. O poliedro regular mais conhecido seja o cubo, que apresenta seis quadrados uniformes como faces.

Para aproximar de uma esfera, cada um dos quadrados pode ser subdividido em quatro novos, adicionando-se novos vértices nos pontos médios de cada aresta e no centro do quadrado original.

Estando os detalhes necessários alcançados, a posição de cada vértice pode (e deve) ser ajustada de forma a que esteja a uma unidade de distância do centro volumétrico do cubo. O resultado é então uma aproximação da esfera unitária que pode depois ser dimensionada e deformada usando um *Heightmap*.

Outro exemplo é a utilização da subdivisão recursiva de outro poliédrico regular, o icosaedro regular, que é composto por doze (12) vértices e vinte (20) triângulos equiláteros como faces. Nesse caso, esses triângulos equiláteros são subdivididos da mesma maneira que os cubos na abordagem anterior, resultando numa aproximação geométrica ligeiramente diferente de uma esfera perfeita.

Devido à sua uniformidade intrínseca, este método de refinar recursivamente os poliedros regulares garante que a malha resultante distribua uniformemente os detalhes geométricos por toda a esfera. Um exemplo pode ser analisado na Figura 4 presente abaixo.

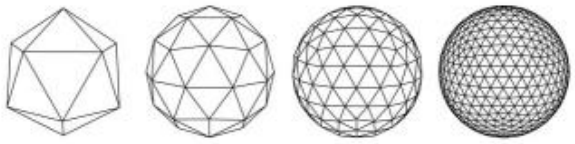

Figura 4 Aproximação Icosaédrica de uma Esfera [13]

3. NOISE FUNCTIONS

Algumas das principais funções de *noise* que foram utilizadas anteriormente na geração de PTG incluem:

- *Diamond Square;*
- *Value Noise;*
- *Perlin Noise;*
- *Simplex Noise;*
- *Worley Noise.*

Veja-se, de forma mais detalhada, a base de cada algoritmo e o modo como funcionam no subcapítulo a seguir.

3.1. DIAMOND SQUARE ALGORITHM

Consiste num algoritmo de *noise* coerente usado para gerar mapas de altura. Este método utiliza o princípio do deslocamento do ponto médio para atribuir valores de elevação a cada coordenada. O algoritmo começa com uma grade de coordenadas quadradas com as quatro coordenadas de canto inicializadas com valores pseudoaleatórios.

Em seguida, calcula a média da elevação desses quatro pontos mais ou menos num deslocamento pseudoaleatório. Este valor é então

atribuído à elevação do ponto no centro direto da grade. A grade é subdividida em quatro quadrados menores e o processo é repetido para cada um deles. Isso é feito até que cada valor na grade tenha sido atribuído um valor de elevação [5].

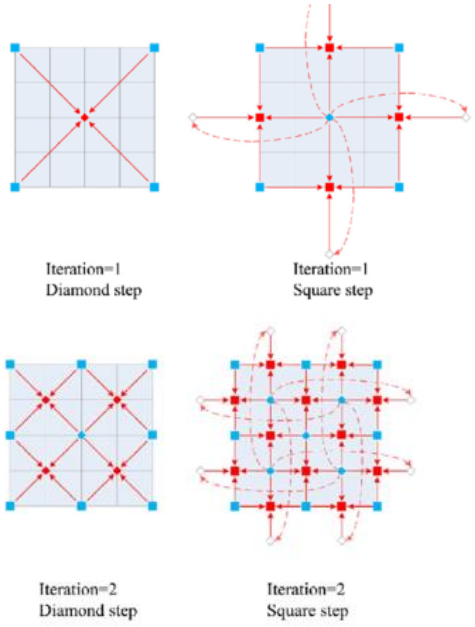

Figura 5 Passos do *Diamond Square Algorithm* [15]

3.2. VALUE NOISE ALGORITHM

O *Value Noise* atribui valores pseudoaleatórios a cada ponto no *Heightmap*. O valor final da elevação de cada ponto é obtido pela interpolação do seu valor com os dos pontos circundantes. Isso dá a cada ponto a consciência dos valores de elevação dos seus vizinhos que são característicos de ruído coerente. Diferentes variações de *Value Noise* podem usar várias funções de interpolação para atingir esse mesmo resultado.

Na Figura 6 é demonstrado como sete *octaves* de *Value Noise* se apresentam no *Heightmap*.

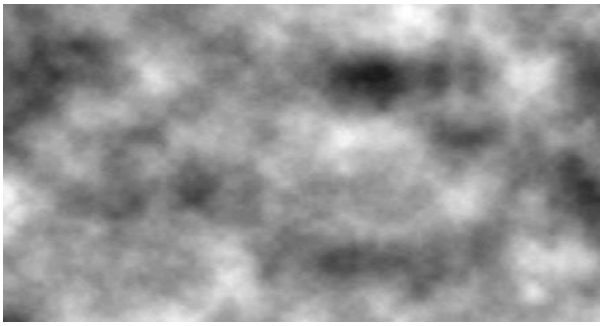

Figura 6 Sete *octaves* de *Value Noise* renderizadas sob a forma de *Heightmap* [12]

O *Perlin Noise* é um dos algoritmos mais antigos de ruído coerente que pode ser usado para criar *Heightmaps*.

Projetado por Ken Perlin no ano de 1983, o ruído de Perlin funciona criando primeiro uma grade de vetores gradientes de comprimento unitário originados em cada intersecção que aponta em direções pseudoaleatórias.

A posição de cada valor de ruído é calculada e colocada dentro de cada célula de modo que cada coordenada seja cercada por quatro pontos de grade, cada um com seu próprio vetor gradiente (Figura 7).

Em seguida, quatro vetores de distância são calculados representando a distância das coordenadas do valor do ruído e os cantos da célula circundante.

Após isto, os produtos escalares entre cada vetor gradiente e cada vetor distância correspondente são calculados e interpolados para produzir o valor final do ruído [8].
Como o cálculo de cada valor de ruído depende dos vetores de distância, cada ponto recebe a consciência do valor dos seus pontos vizinhos e o ruído coerente é assim obtido [3].

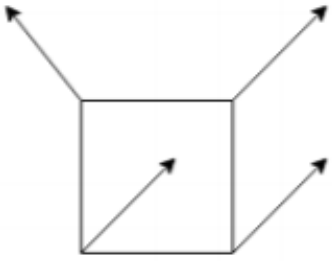

Figura 7 Cubo de unidade com um vetor de gradiente em cada canto [16]

3.3. SIMPLEX SQUARE ALGORITHM

Simplex Noise é outro algoritmo de ruído coerente projetado por Ken Perlin em 2001 que serve como uma otimização do algoritmo *Perlin Noise* [9].

Este algoritmo simplifica o *Perlin Noise*, através do uso de uma grade *simplex* em que cada célula passa a ser um triângulo ao invés de um quadrado. Isso resulta em menos vetores de distância necessários para calcular cada valor de altura, pois cada célula passa a ter apenas 3 cantos.

Além disso, o *Simplex Noise* utiliza um processo de soma em vez de uma interpolação para calcular os

valores finais de ruído. O *Simplex Noise* é menos complexo, em termos computacionais, do que o *Perlin Noise* e, dessa forma, acabar por possuir uma melhor escalabilidade.

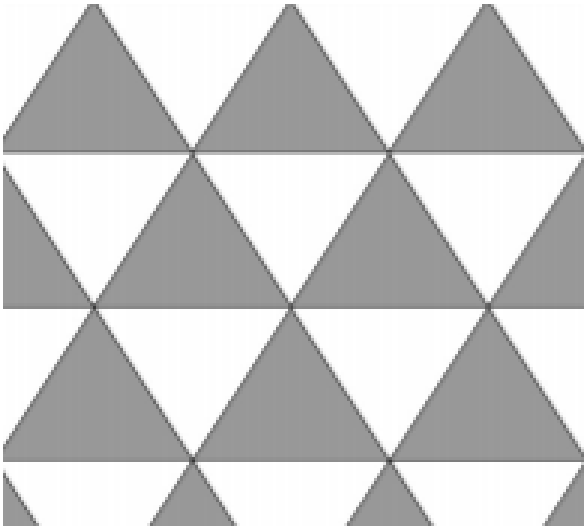

Figura 8 Exemplificação do *Simplex Square Algorithm* [16]

3.4. WORLEY NOISE ALGORITHM

Worley Noise é uma função de ruído introduzida por Steven Worley em 1996. Na Computação Gráfica, é usado para criar texturas que são criadas automaticamente com precisão arbitrária e não precisam ser desenhadas à mão. O *Worley Noise* chega perto de simular texturas de pedra, água ou células biológicas.

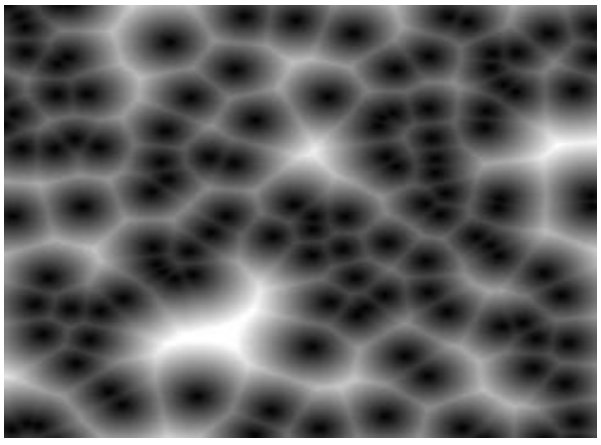

Figura 9 Imagem de Resultados Obtidos através do *Worley Noise* [6]

De forma a compactar a informação das cinco funções de ruído mencionadas anteriormente, a Tabela 1 propõe assim uma comparação com base na velocidade, requisitos de memória e a qualidade do ruído que produzem.

Algoritmo	Velocidade	Qualidade	Requisitos Memória
Diamond Square	Muito Rápido	Moderado	Alto
Value Noise	Lento-Rápido*	Baixo-moderado*	Muito Baixo
Perlin Noise	Moderado	Elevado	Baixo
Simplex Noise	Moderado**	Muito Elevado	Baixo
Worley Noise	Variável	Único	Variável

Tabela 1 Tabela Comparativa das Funções de Ruído [12]

* Depende da função de Interpolação usada

** Escala melhor para dimensões mais altas do que o *Perlin Noise*

A função de ruído mais adequada irá variar entre os Sistemas de *Terrain Generation*:

- Se a velocidade for essencial, o Algoritmo *Diamond Square* é uma boa opção, dado que é muito mais rápido do que todos os outros;
- O *Value Noise* é uma ótima alternativa se houver falta de memória e também é facilmente personalizável devido à sua natureza simples;
- O *Worley Noise*, por sua vez, fornece ruído de forma muito mais exclusiva em termos visuais comparativamente às outras funções de ruído, podendo ser ideal/desejável em determinados cenários;
- Para a qualidade geral, o *Simplex Noise* é a escolha ideal, embora o clássico *Perlin Noise* provavelmente seja suficiente na grande maioria das circunstâncias.

4. OUTRAS ABORDAGENS PARA O PROCEDURAL TERRAIN GENERATION

Mencionando de seguida os agentes de *software*, os mesmos fornecem uma alternativa interessante às funções de ruído para gerar terreno. Uma das grandes vantagens que eles produzem é que

são totalmente controláveis e podem ser usados em um sistema PTG que trabalha com parâmetros escolhidos pelo designer, ao invés de números aleatórios [7].

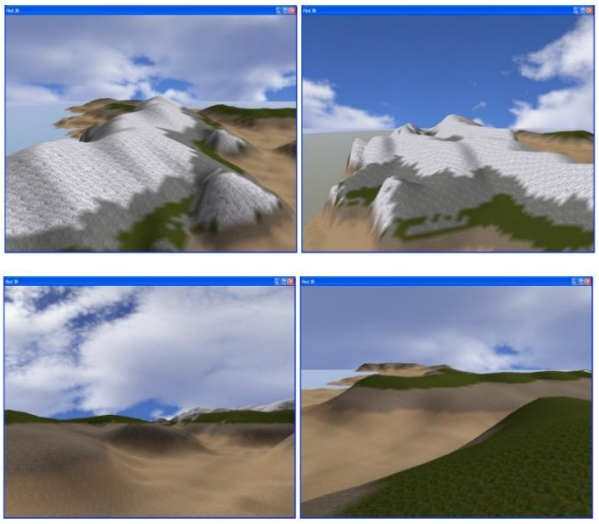

Figura 10 Exemplos de *Terrain Generation* por Agentes de *Software* [14]

Os Modelos de Erosão com base física são uma forma útil de dar ao terreno de jogo uma aparência mais desgastada e natural. Embora a erosão hidráulica seja notavelmente mais lenta que a erosão térmica e requeira aproximadamente três vezes a quantidade de memória, ela produz resultados significativamente melhores.

Simular erosão hidráulica na GPU pode ajudar a superar as limitações gerais de velocidade do algoritmo [1] [4].

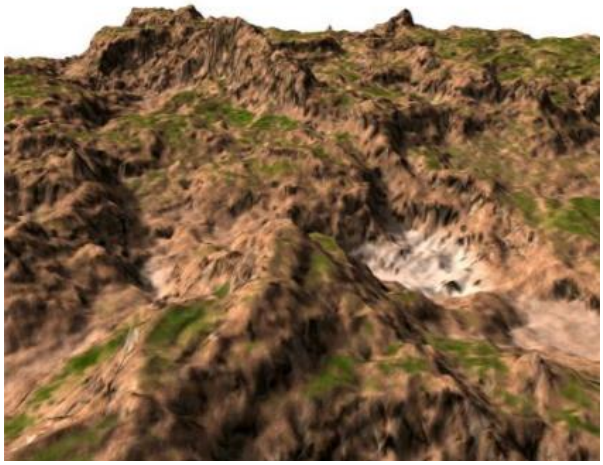

Figura 11 Terrenos de Modelos de Erosão [6]

Por último, os Algoritmos Evolutivos (EA) podem adicionar um certo controlo ao processo de criação de terreno (PTG), fornecendo uma alternativa às abordagens tradicionais tipicamente aleatórias. Isso permite não apenas a criação de uma grande variedade de terrenos, mas também a criação e o refinamento de famílias de terrenos semelhantes.

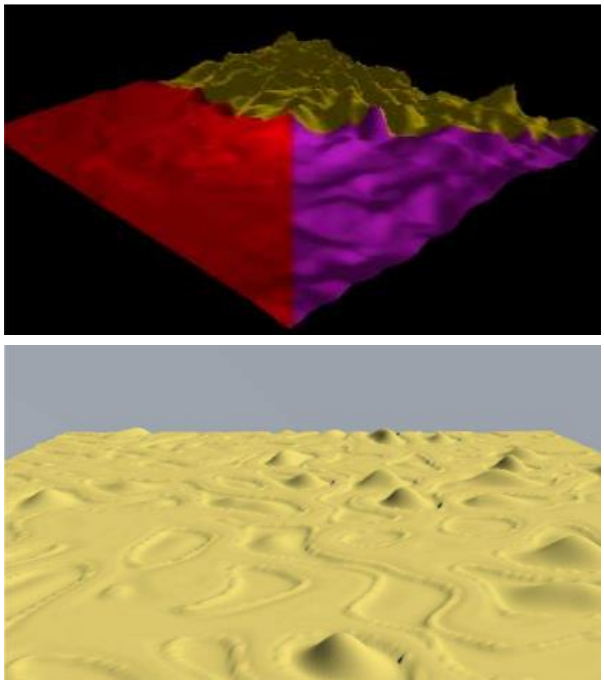

Figura 12 Exemplos de terrenos com uso de Algoritmos Evolutivos [17]

5. CARACTERÍSTICAS DE TERRENOS APLICADAS AO RAMO DOS JOGOS

O tipo de jogo onde o terreno virtual deverá ser usado deve ser o fator determinativo para ditar quais as características necessárias do terreno em si. Olsen [6] descreve alguns requisitos para que uma técnica de PTG seja útil em tempo real:

- O autor afirma que o terreno deve conter áreas planas suficientes para os personagens do jogo se consigam mover de forma realista;
- O terreno deve possuir uma alta conectividade, de forma que os jogadores possam atravessar o máximo possível do seu mapa;
- O terreno também precisa de colinas, penhascos, vales e outros recursos para promover a estratégia do jogo e, com isso, tornar os mapas visualmente mais atrativos e desafiadores.

Embora esses critérios sejam verídicos para muitos tipos de jogos, cada tipo poderá/deverá ter os seus

próprios requisitos de escala e organização de recursos.

Por exemplo, um Jogo de Tiro em primeira pessoa, o Call of Duty: Modern Warfare 3 (Infinity Ward, 2011), tem normalmente mapas em pequena escala contendo uma alta densidade de características do terreno e objetos virtuais para impedir a linha de visão (Figura 18).

Já os jogos *Role Playing*, mais conhecidos como RPG, como o The Elder Scrolls V: Skyrim (Bathesda Game Studios, 2011), têm, normalmente grandes ambientes com cidades dispersas, incentivando assim a exploração de um mundo muito mais expansivo (Figura 19).

Além disso, alguns gêneros de jogos, como Simuladores de voo, não precisam dos mesmos requisitos de travessia, mas exigem que o terreno pareça realista de uma perspetiva aérea.

Figura 13 Call of Duty: Modern Warfare 3 [18]

Figura 14 The Elder Scrolls V: Skyrim [19]

6. CONCLUSÕES E OBSERVAÇÕES FINAIS

Atualmente, existe uma grande variedade de técnicas eficazes que podem ser aplicadas na criação de conteúdo de jogos. As *Noise Functions* discutidas ao longo deste artigo são excecionalmente úteis para criar/desenvolver um terreno inicial e cada uma possui diversas características úteis e únicas entre si.

As simulações com base física não são necessariamente adequadas para produzir terreno por conta própria, mas podem oferecer uma maneira de dar forma a um mundo virtual e adicionar uma camada adicional de realismo sobre o mesmo.

Relativamente aos Algoritmos Evolutivos, as suas técnicas foram aplicadas à criação de conteúdo de jogos com sucesso e podem ser usadas para personalizar o conteúdo para jogadores individuais e, portanto, o seu uso no PTG deve ser investigado mais a fundo.

Relacionando diretamente com o ramo dos *Videogames*, uma das abordagens mais populares é baseada nos Algoritmos Fractais devido a uma série de razões:

- Redução no custo de produção dos terrenos;
- Podem reduzir bastante o tamanho do *download* e do posterior armazenamento do jogo;
- Redução do uso de memória em toda a GPU, tornando-os ideais para *videogames*;
- Permitem uma grande criatividade, o que torna todo o jogo mais divertido, interessante e estimulante para o utilizador.

Por outro lado, a principal desvantagem dos algoritmos de criação de PTG é que não é fácil criar resultados realísticos em comparação com outros métodos como o *Laser Scanning, que se centra n*uma tecnologia de medição e digitalização remota 3D de alta precisão.

Consequentemente, embora aproximações de paisagens possam ser facilmente controladas, gerar um local específico em uma paisagem não é uma tarefa propriamente banal/trivial.

Outros problemas típicos desses algoritmos incluem a modelagem de detalhes específicos em paisagens, como a criação de cavernas e saliências realistas.

Uma noção final a ser considerada daqui para frente é a maneira pela qual o conteúdo gerado

procedimentalmente pode ser avaliado. O conteúdo produzido por um sistema PTG pode ser difícil de avaliar, pois o que torna o terreno do jogo "bom" é uma questão bastante subjetiva aos olhos de muitas entidades.

Além disso, pode ser desafiador comparar a qualidade de diferentes algoritmos e técnicas de criação de terreno, já que a saída final de um sistema PTG é fortemente afetada pelas técnicas de *rendering* usadas.

7. REFERÊNCIAS BILIOGRÁFICAS

[1] Archer, T. (2011). Procedurally Generating Terrain. *3rd Int. Conf. Comput. Support. Educ.*

[16] Andersson, M., Berger, K., Bengtsson, F. B., Gelotte, B., Sagdahl, J. G., & Kvarnström, S. (2017). Procedural Generation of a 3D Terrain Model Based on a Predefined Road Mesh. Bachelor of Science Thesis in Applied Information Technology.

[2] Bevins, J. (s.d.). *Libnoise Glossary*. Obtido de http://libnoise.sourceforge.net/glossary/index.html#coherentnoise

[14] Doran, J., & Parberry, I. (2010). Controlled Procedural Terrain Generation Using Software Agents. IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES, VOL. 2 (2). doi:10.1109/TCIAIG.2010.2049020

[3] D. Maung, Y. S. (2012). Procedural Textures Using Tilings with Perlin Noise. *17th International Conference on Computer Games (CGAMES)*.

[4] Decaudin, X. M. (2007). Fast Hydraulic Erosion Simulation and Visualization on GPU. *Pacific Graph*.

[15] Depeng, L., Li, Z., Bing, L., & Wenhao, C. (2017, maio). Infrared Small Target Detection in Heavy Sky Scene Clutter Based on Sparse Representation. Infrared Physics & Technology 85. doi:10.1016/j.infrared.2017.05.009

[5] H.-R. Wang, W.-L. C.-L. (2010). An Improving Algorithm for Generating Real Sense Terrain and Parameter Analysis Based on Fractal. *International Conference on Machine Learning and Cybernetics*.

[6] Olsen, J. (2004). Realtime procedural terrain generation. *Department of Mathematics And Computer Science (IMADA) University of Southern Denmark*.

[7] Parberry, J. D. (2010). Controlled Procedural Terrain Generation Using Software Agents. *IEEE Trans. Comput. Intell. AI Games, Vol. 2 (2)*, 111–119.

[8] Perlin, K. (1985). An Image Synthesizer. *12th Annual Conference on Computer Graphics and Interactive Techniques*.

[9] Perlin, K. (2002). Improving Noise. *29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)* .

[10] Petrovas, A., & Baušys, R. (2019). Automated Digital Terrain Elevation Modification By Procedural Generation Approach. *Open Conference of Electrical, Electronic and Information Sciences*. doi:10.1109/eStream.2019.8732171

[17] *Raffe, W. L., Zambetta, F., & Li, X. (2012). A Survey of Procedural Terrain Generation Techniques using Evolutionary Algorithms. WCCI 2012 IEEE World Congress on Computational Intelligence. doi:10.1109/CEC.2012.6256610*

[11] R. Kooima, J. L. (2009). Planetary-Scale Terrain Composition. *IEEE Transactions on Visualization and Computer Graphics, vol. 15 (5)*, 719–733.

[12] Rose, T. J., & Bakaoukas, A. G. (2016). Algorithms and Approaches for Procedural Terrain. *8th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*.

[13] Vitacion, R. J., & Liu, L. (2019). Procedural Generation of 3D Planetary-Scale Terrains. *International Conference on Space Mission Challenges for Information Technology*. doi:10.1109/SMC-IT.2019.00014

8. REFERÊNCIAS WEB

[18] Black Tuesday, Call of Duty. Obtido de https://callofduty.fandom.com/wiki/Black_Tuesday

[19] The Elder Scrolls V Pictures, News Articles, Videos. Obtido de https://gadgets.ndtv.com/tags/the-elder-scrolls-v