

Mestrado Integrado em Engenharia Informática

Unidade Curricular de Tecnologias de Segurança

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Terrain Generation*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Elaborado por:**

* André Germano A7
* Leandro Salgado A70949
* Luís Costa A73434
* Sofia Carvalho A7

Índice

[1 Introdução 3](#_Toc504771791)

[2 Geração de terreno em tempo real 4](#_Toc504771792)

[2.1 Implementação 4](#_Toc504771793)

[2.2 Análise de Desempenho 4](#_Toc504771794)

[2.3 Considerações 5](#_Toc504771795)

[3 Conclusão 6](#_Toc504771796)

[4 Referências 7](#_Toc504771797)

# Introdução

# Geração de terreno em tempo real

De modo a demonstrar a geração de terreno em tempo real, decidiu-se que se utilizaria uma técnica baseada em ruído para calcular a altura do terreno em determinados pontos.

Para tal, utilizou-se um algoritmo de ruído criado por Ken Perlin, designado por *simplex noise*.

## Implementação

Utilizando a Nau3D para suporte, a técnica foi implementada em GLSL, fazendo uso de três *shaders*: *vertex shader*, *geometry shader*, e *fragment shader*.

O *vertex shader* trata do cálculo da altura de cada vértice, fazendo uso de uma função de ruído *simplex 2D[[1]](#footnote-1)*, utilizando depois alguma turbulência para ter um aspeto mais natural. Nesta implementação, o *vertex shader* trata ainda de decidir a cor dos vértices dependendo da sua altura, e aplica um *threshold* para que os vértices não tenham uma altura negativa.

As normais de superfície são calculadas por triângulo, pelo *geometry shader*, que também altera as cores dos vértices caso a normal do triângulo esteja muito próxima de ser ortogonal a um vetor unitário vertical.

Por último, o *fragment shader* utiliza as normais de superfície para calcular a iluminação de cada triângulo.

Os três *shaders* são depois utilizados numa *scene* simples que consiste numa grelha 2D de 1024x1024, com 1024 divisões em ambos os eixos.

## Análise de Desempenho

De modo a testar a escalabilidade dos *shaders*, foram testadas várias grelhas, com diferentes tamanhos e divisões ao longo dos eixos e, com ajuda da *Nau3D,* geraram-se *logs* de cada teste. Os tamanhos e o número de divisões testados foram 512, 1024 e 2048 para ambos, num total de 9 grelhas diferentes testadas.

Variando apenas o número de divisões em cada eixo, ao duplicar o número de divisões, verifica-se um aumento de aproximadamente 4 vezes no tempo de execução, consistente com o aumento de triângulos resultante da duplicação do número de divisões. **(requer confirmação)**

Variando apenas o tamanho da grelha, existem pequenas variações no tempo de execução, na ordem das décimas de milissegundo. No entanto, à medida que o tamanho da grelha se aproxima do número de divisões, verifica-se uma diminuição do tempo de execução. **(incluir alguns logs em anexo, como exemplos)**

Em termos de eficiência da implementação desenvolvida, a função de ruído é descrita pelo autor como sendo rápida. Isto pode ser comprovado aumentando os *octaves* na turbulência, por exemplo, de 1 para 8, em que um aumento de cálculos do ruído em 8 vezes resulta num aumento do tempo de execução negligenciável ou até inexistente ao nível dos milissegundos. Da mesma forma, um aumento de 8 *octaves* para 16 resulta num pequeno aumento do tempo de execução, com um mínimo aumento do detalhe. A partir deste ponto, o aumento dos *octaves* tem impacto cada vez maior na *performance* sem qualquer aumento significativo de detalhe.

Em termos de instruções condicionais, as funções contendo *ifs* foram reescritas de forma a atingir o mesmo resultado sem *ifs*. No entanto, não se observaram reduções significativas nos tempos de execução, uma vez que estas não englobam grandes blocos de código.

## Considerações

Ao contrário de outras técnicas existentes, a técnica implementada não requer nem permite a utilização de dados previamente gerados, sendo que os valores de altura são todos calculados em *runtime*.

Pela análise de desempenho, é possível inferir que esta implementação não escala bem para *meshes* muito densas. No entanto, seria possível melhorar o desempenho utilizando *tessellation* para reduzir o detalhe nos limites do *view frustrum* mais afastados da câmara. Porém, o tamanho da *mesh* não influencia o desempenho de forma significativa.

A função de ruído utilizada é descrita pelos autores como sendo rápida, paralelizável, e independente de dados externos. Não utiliza tabelas de *hash* nem texturas, como outras implementações.

# Conclusão

# Referências

1. *Simplex noise*, de Ken Perlin,modificado para *GLSL*, autoria de Ashima Arts [↑](#footnote-ref-1)