

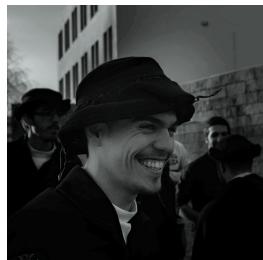
**Universidade do Minho**

# Visualização e Iluminação I

MIEI - 4º ANO - 1º SEMESTRE

UNIVERSIDADE DO MINHO

## TRABALHO PRÁTICO



João Leal  
A75569



Pedro Almeida  
A74301



António Lopes  
A74357

2 de Fevereiro de 2020

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Perlin Noise</b>	<b>3</b>
2.1	Algoritmo . . . . .	3
2.1.1	Definição da <i>grid</i> . . . . .	3
2.1.2	Cálculo do produto escalar . . . . .	4
2.1.3	Interpolação entre os valores calculados . . . . .	5
2.1.4	Fractal Brownian Motion . . . . .	6
<b>3</b>	<b>Aplicação do Algoritmo</b>	<b>7</b>
3.1	Plano . . . . .	7
3.2	Esfera . . . . .	7
<b>4</b>	<b>Variaveis</b>	<b>8</b>
<b>5</b>	<b>Texturas</b>	<b>9</b>
<b>6</b>	<b>Resultados</b>	<b>12</b>
<b>7</b>	<b>Conclusão</b>	<b>14</b>

# Capítulo 1

## Introdução

Os avanços notáveis na computação gráfica permitem simular mundos virtuais em constante crescimento com um nível mais alto de realismo, que pode até ser criado em tempo real, de forma procedural. Uma parte fundamental desses mundos virtuais são os terrenos que nestes assentam. Assim sendo e, após uma análise à lista de temas possíveis para este trabalho prático acabamos por optar pelo *Terrain generation*, de forma a explorar como este pode ser gerado proceduralmente e como certos parâmetros influenciam a geração do mesmo. Para este âmbito recorremos à técnica de Perlin Noise, frequentemente usada na geração de terrenos e outros materiais.

Este relatório irá descrever o desenvolvimento do projecto prático da Unidade Curricular de **Visualização e Iluminação I** onde se pretende que o grupo para além da implementação do algoritmo, relativo ao tema escolhido, seja também capaz de o analisar conseguindo apontar quais os pontos fortes e também os seus aspectos negativos.

Esperamos assim aquando da finalização deste projecto, que tenhamos atingido um bom patamar da sua resolução e que obtenhamos um bom resultado final que vá de encontro ao esperado no enunciado. Queremos também que com o mesmo, adquirirmos novos conhecimentos e consolidemos aqueles que nos foram sendo transmitidos aos longo do semestre nas aulas práticas da UC.

# Capítulo 2

## Perlin Noise

O *Perlin Noise* é um algoritmo extremamente poderoso, frequentemente usado em jogos e filmes, na geração de conteúdo gráfico procedimental, influenciando o mesmo para sempre. Esta técnica pode ser usada na geração de qualquer tipo de material ondulado ou textura ondulatória, por exemplo, efeitos de fogo, água, nuvens ou até mesmo em terrenos procedimentais (i.e., Minecraft).

Na sua forma básica, este tipo de ruído tem a mesma aparência geral que uma onda sinusoidal, onde a amplitude e a frequência variam um pouco, mas a amplitude permanece razoavelmente consistente e é restrita a uma faixa bastante estreita em torno de uma frequência central. Contudo, não é tão regular como uma onda sinusoidal, tornando mais fácil de criar uma aparência aleatória.

### 2.1 Algoritmo

O *Perlin Noise* é normalmente implementado como uma função bidimensional, tridimensional ou quadridimensional, podendo ser definido para qualquer outro número de dimensões. Esta implementação geralmente envolve três etapas:

- Definição de uma *grid* com vetores gradiente aleatórios;
- Cálculo do produto escalar entre os vetores gradiente;
- Interpolação entre os valores calculados;

#### 2.1.1 Definição da *grid*

Para cada ponto da *grid*, criamos um vetor gradiente aleatório. Este vetor (normalmente de comprimento uniforme, por exemplo, 1) aponta numa direção aleatória a partir do seu ponto de origem na *grid*, como podemos ver na figura abaixo.

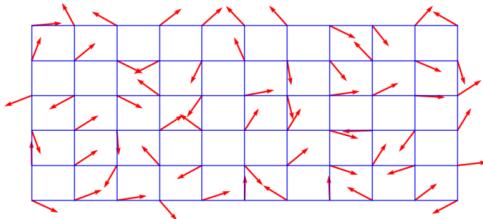
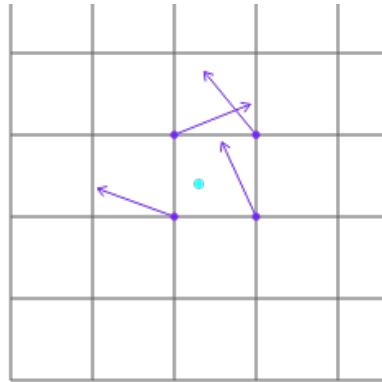


Figura 2.1: Grid e vetores gradiente

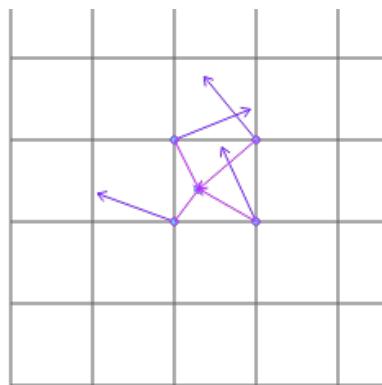
Os vetores são chamados de gradientes porque a função de ruído terá uma inclinação positiva (isto é, aumentará) na direção de cada vetor gradiente.

## 2.1.2 Cálculo do produto escalar

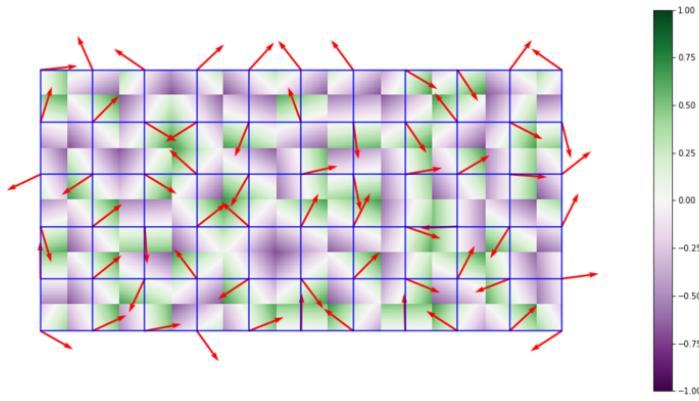
Para cada pixel  $P$  para o qual precisamos calcular o ruído, avaliamos uma função usando os quatro pontos da *grid* vizinhos  $Q$  e os seus vetores gradiente. Na figura seguinte, o pixel é marcado como um ponto azul ciano e os quatro pontos vizinhos da *grid* são marcados como pontos roxos, assim como os respectivos vetores gradiente.



Para cada ponto vizinho  $Q$ , pegamos no produto escalar do gradiente  $G$  em  $Q$  com o vetor de distância  $(P - Q)$ . O resultado do produto escalar contribui para o valor do ruído em  $P$ . Aqui os vetores de distância  $(P - Q)$  são mostrados em rosa:

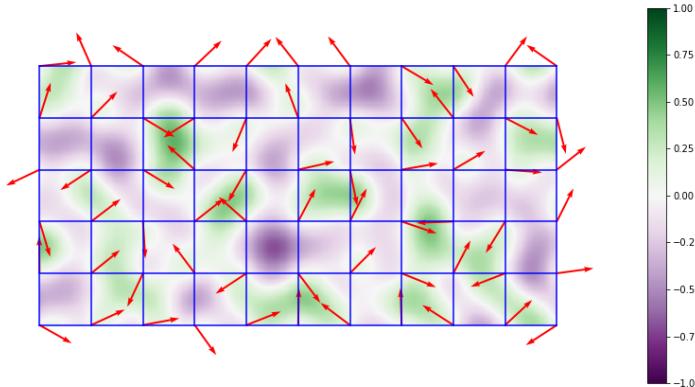


Apartir destes cálculos obtemos uma *grid* que mostra as diferentes contribuições para o ruído em  $P$ .



### 2.1.3 Interpolação entre os valores calculados

O último passo é a interpolação entre os  $2^n$  produtos escalares de ponto computados nos nós da célula da *grid* que contém o ponto de argumento. Este passo é realizado usando uma função que possui zero primeiras derivadas (e possivelmente também segunda derivada) nesses mesmos nós da grade. Assim sendo, em pontos próximos aos nós da grade, a saída aproximará o produto escalar mais cedo. Isso significa que a função de ruído passará pelo zero em cada nó e terá um gradiente igual ao gradiente do nó da grade pré-processado. Essas propriedades conferem ao ruído Perlin a sua escala espacial característica e uma contribuição de ruído, em cada ponto, mais suave.



Assim, com o uso desta técnica aplicada a diferentes dimensões é possível obter resultados variados, indo desde ondas de ruído, a texturas naturais (i.e., fogo) ou a mapas 3D com montanhas e cavernas.

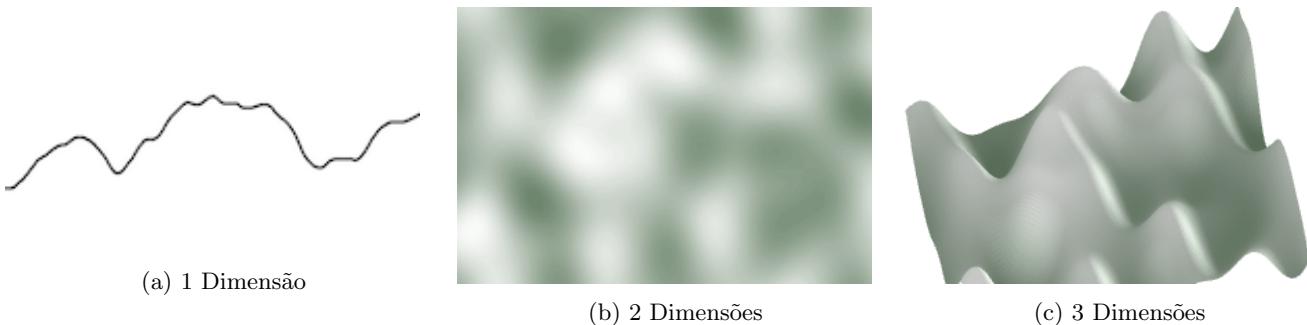
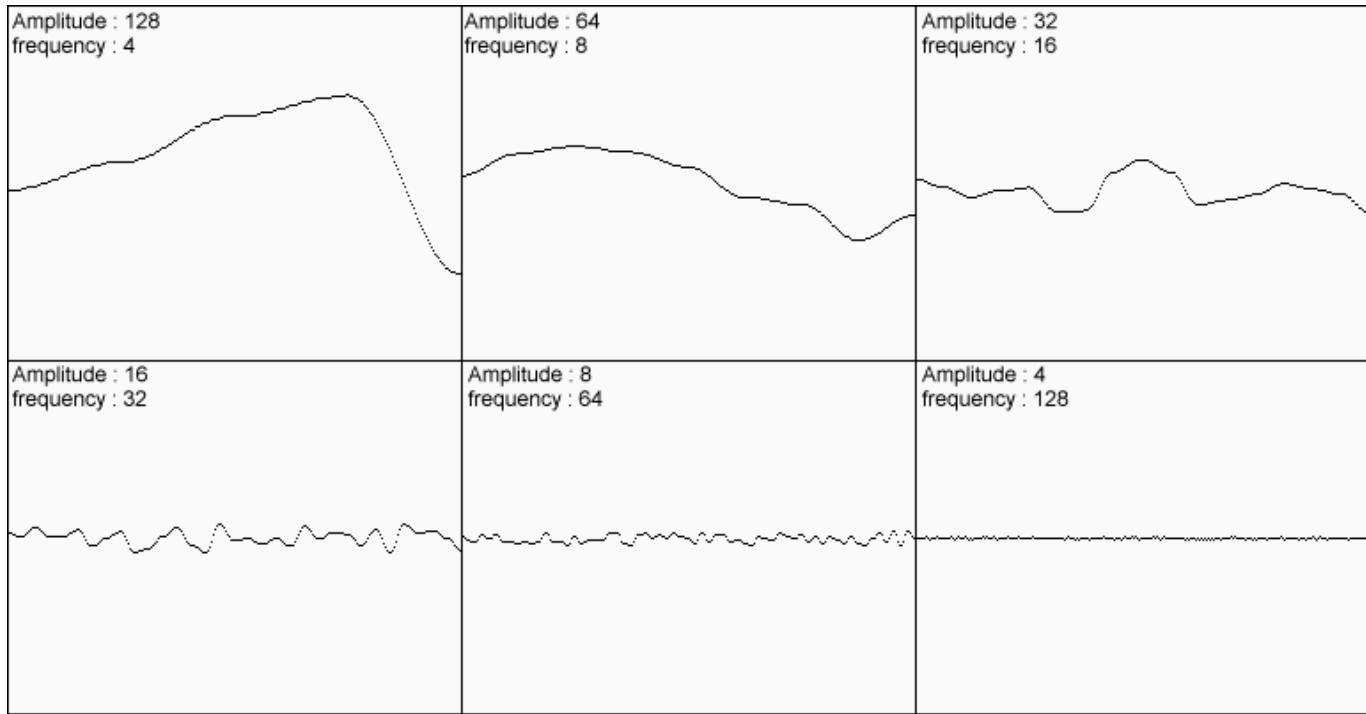


Figura 2.2: Imagens em *grayscale*

Neste projeto iremos usar uma versão do algoritmo, com um conjunto de rotinas de shaders, de forma a produzir ruído necessário. O algoritmo usado pode ser encontrado em (<https://github.com/ashima/webgl-noise/wiki>).

## 2.1.4 Fractal Brownian Motion

Ao adicionar diferentes iterações de ruído (oitavas), onde sucessivamente incrementamos a frequência e diminuímos a amplitude do ruído, obteremos uma maior granularidade no ruído. Esta técnica é normalmente chamada de “Fractal Brownian Motion” (**fBM**).



Assim com esta técnica iremos produzir um resultado mais próximo do realista pois o resultado final irá apresentar uma maior granularidade. É devido a estas suas características que a fazem ser bastante utilizada para a geração procedural de terrenos pois uma maior granularidade equivale a um maior nível de detalhe.

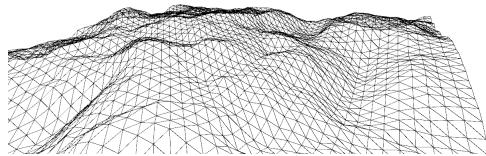
# Capítulo 3

## Aplicação do Algoritmo

O ruído por si só é apenas um conjunto de números. Para tornar este conjunto útil e poder trabalhar com este, é necessário atribuir-lhe um significado. A primeira coisa que podemos pensar é fazer os valores de ruído produzido, corresponder às elevações do nosso terreno(também chamado de "*height-map*"). Tendo isto em conta, decidimos aplicar este conceito em dois objetos diferentes: um plano e uma esfera.

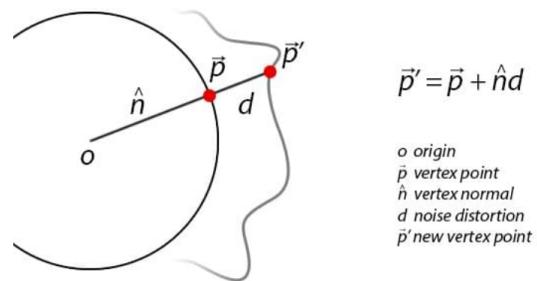
### 3.1 Plano

Para esta parte, simplesmente basta atribuir a  $Y$  o valor de ruído gerado de forma a produzir efeitos como o seguinte:



### 3.2 Esfera

Já nesta fase, por se tratar de uma esfera, é necessário ter em conta em que direção será projetada esta elevação. Para resolver este problema, usamos a normal do vértice para obter esta mesma direção, como podemos observar na seguinte figura:



# Capítulo 4

## Variaveis

Sendo este um projecto de *procedural terrain generation* irá existir um conjunto de variaveis que consoante o seu valor irá afetar a forma como o terreno é gerado e como o ruído gerado aleatoriamente pode ser controlado. Visto que o algoritmo tem grande peso na criação e aplicação do noise **fBm** estas variaveis estarão relacionadas com esse mesmo processo. Iremos agora apresentar quais essas variáveis que poderão ser modificadas no nosso programa e quais as implicações que a alterações das mesmas provocam no resultado final.

- **roughness** (float) - Representa o incremento da frequência em cada oitava.
- **base\_roughness** (float) - Representa a frequência inicial com que entra no ciclo de oitavas.
- **persistence** (float) - Esta variável determina quanto cada octave contribui para a estrutura geral do mapa de ruído. Se a persistence for 1, todas as octaves contribuem da mesma forma.
- **octaves** (int) - Esta variável representa os níveis de detalhe do ruído de Perlin, que diminui ou expande detalhes.
- **minValue** (float) - Esta variável à medida que aumenta, diminui o nível de relevo do planeta.
- **strength** (float) - Esta variável à medida que aumenta, o relevo da superfície do planeta também aumenta.
- **noise\_x** (float) - Parametro x relativo à coordenada do centro do ruído, que permite alterar o ruído ao longo do objeto.
- **noise\_y** (float) - Parametro y relativo à coordenada do centro do ruído, que permite alterar o ruído ao longo do objeto.
- **noise\_z** (float) - Parametro z relativo à coordenada do centro do ruído, que permite alterar o ruído ao longo do objeto.

Como esperado, grande parte das variáveis presentes estão correlacionadas com a geração do ruido que irá criar o relevo no nosso mundo onde com a alteração dos mesmos consigamos obter melhores resultados de realismo para o sistema.

# Capítulo 5

## Texturas

Após a geração e aplicação do noise ao nosso modelo, de maneira a criar as parecenças com os vários tipos de terrenos presentes no planeta, selecionamos diversas Textura de modo a aplicar cada especificamente a cada caso. Logo teremos texturas desde agua que irá ser aplicada o nível mais baixo para a representação dos oceanos e possíveis Lagos, até a uma textura de Neve para os pontos de maior elevação no cenário que irá representar os cumes montanhosos. No total serão utilizadas 6 texturas diferentes para representar as várias diferenças de níveis possíveis no relevo do terreno.

**Water** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado de 0, representando a água do planeta, sendo assim, a primeira camada da superfície do planeta.



Figura 5.1: Textura Water

**Sand** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado inferior a 0.07 e superior a 0.02, representando a areia/terra do planeta, sendo assim, a segunda camada da superfície do planeta.



Figura 5.2: Textura Sand

**Grass** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado inferior 0.6 e superior a 0.15, representando a vegetação do planeta, sendo assim, a terceira camada da superfície do planeta.



Figura 5.3: Textura Grass

**Rock** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado inferior 0.75 e superior a 0.65, representando as rochas do planeta, sendo assim, a quarta camada da superfície do planeta.



Figura 5.4: Textura Rock

**Mountain** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado inferior 0.9 e superior a 0.85, representando as montanhas do planeta, sendo assim, a quinta camada da superfície do planeta.



Figura 5.5: Textura Montain

**Snow** - Esta textura é usada para os pontos onde a variável "elevation" tiver o resultado superior a 0.9, representando a neve/gelo do planeta, sendo assim, a sexta camada da superfície do planeta.



Figura 5.6: Textura Snow

Mas como é sabido e também observável no exterior, as variações entre as diferentes camadas não é algo drástico, existem zonas de transição entre as mesmas onde esta mais suave. Para reproduzir esse efeito de modo a levar o grau de realismo presente no projecto, decidimos aplicar o *smoothstep* entre as várias transições entre texturas de modo a criar o tal efeito de transição mais próximo da realidade. Com esta técnica conseguimos então melhorar substancialmente o nível de realismo e qualidade do terreno gerado.

# Capítulo 6

## Resultados

Neste capítulo iremos apresentar agora os resultados finais obtidos aquando a finalização do projecto onde mostraremos como a implementação do algoritmo funcionou na prática.

Iremos mostrar o panorama geral da vista da scene ao correu o projecto e depois iremos mostrar mais em detalhe a forma da criação dos triângulos no sistema.

De inicio temos a imagem do planeta gerado através do nosso algoritmo já com valores selecionados para melhor verificar as potencialidades do mesmo.

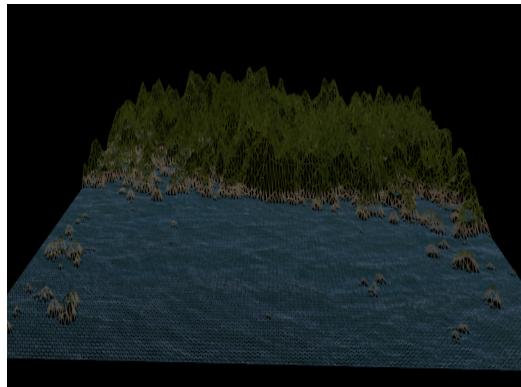


Figura 6.1: Scene Principal - Plano



Figura 6.2: Scene Principal - Esfera

Com a próxima imagem temos uma visão aproximada sobre o resultado do ruido sobre a esfera, criando o relevo de terreno. Note-se ainda também o wireframe gerado para o mesmo onde também conseguimos observar melhor a transição de textures aplicadas consoante a elevação.

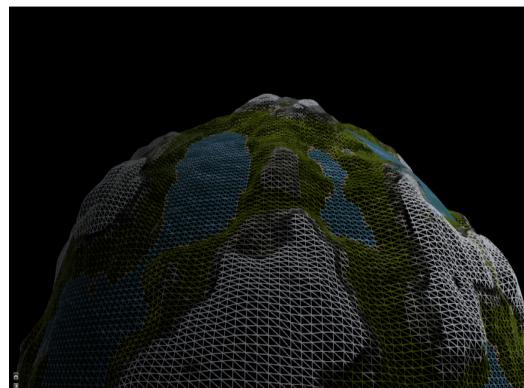


Figura 6.3

# Capítulo 7

## Conclusão

Damos então por concluido a apresentação do projecto desenvolvido para a UC de Visualização e Iluminação I, onde conseguimos aplicar as várias técnicas que foram sendo expostas ao longo do semestre e que permitiu ao grupo atingir um patamar que consideramos satisfatório mas que contudo, ainda tem margem para melhoramentos.

Para além das ferramentas já conhecidas previamente, tivemos também que proceder ao estudo de quais as mais adequadas para o problema selecionado sendo esta uma das partes mais desafiantes como o estudo do *Perlin Noise* e a sua aplicação via a técnica *fBm* que nos ocupou algum tempo de estudo e experimentação mas que após essa fase, conseguimos aplicar no nosso projecto com sucesso criando assim a nossa canvas para o terreno no mundo.

Para trabalho futuro, de modo a elevar ainda mais o nível do projecto, poderíamos proceder a colocação de sombras no terreno de forma a criar um maior realismo nas escarpas montanhosas e também ”povoar” certas zonas com objectos como arvores para dar outro dinamismo visual, assim como introduzir tesselação no projeto.