

**Universidade do Minho**  
Escola de Engenharia

COMPUTER GRAPHICS

## VISUALIZATION AND ILLUMINATION

### PRACTICAL ASSIGNMENT

---

## TERRAIN GENERATION

---



Hugo Cunha **A84656**



Joana Gomes **A84912**



Luís Ferreira **A86265**

Braga, 17 de fevereiro de 2021

# **Conteúdo**

<b>1 INTRODUÇÃO</b>	<b>2</b>
<b>2 CONTEXTUALIZAÇÃO</b>	<b>2</b>
<b>3 RUÍDO</b>	<b>3</b>
<b>4 ILUMINAÇÃO</b>	<b>5</b>
<b>5 TEXTURAS</b>	<b>7</b>
<b>6 INTERFACE</b>	<b>9</b>
<b>7 CONCLUSÃO</b>	<b>10</b>

## 1 INTRODUÇÃO

No âmbito da Unidade Curricular de **Visualização e Iluminação**, foi-nos proposto a implementação e análise de um algoritmo na área de Computação Gráfica. De entre os tópicos sugeridos, escolhemos o tema **Terrain Generation**, por unanimidade de atração pelo tema.

Como apoio no desenvolvimento do projeto, decidimos usar a **Nau3D**, sendo esta um *graphics engine* que nos fornece variadas *features* que achamos pertinentes para o avanço do trabalho.

## 2 CONTEXTUALIZAÇÃO

Um dos *assets* que nos é fornecido pela *Nau3D* é uma **grid** usada como base do nosso algoritmo de *terrain generation*. Como seria de esperar, esta **grid** é criada como um plano, o que faz com que todas as suas normais apontem para cima ( $x=0, y=1, z=0$ ).

Em termos de *shaders* utilizamos apenas dois no nosso trabalho: um *Vertex Shader* e um *Fragment Shader*. No *Vertex Shader* é aplicado o ruído aos pontos, assim como o cálculo das normais iniciais dos pontos e a direção real da luz. Já no *Fragment Shader* são aplicadas as texturas, assim como aplicadas novas normais baseadas na textura.

Para tornar o terreno mais customizado, aplicamos um **ruído** em todas as coordenadas **y** da **grid**, de forma a modelar a sua altura, como vamos aprofundar futuramente na secção 3. Intuitivamente, é necessário um novo cálculo das normais, de modo a se adaptar às diversas inclinações do terreno, que iremos analisar também, mais à frente (na secção 4).

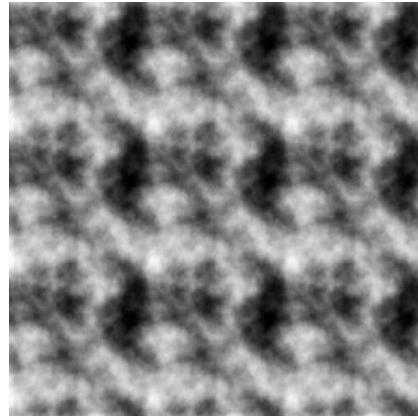
Através dos diferentes níveis de altura, foram escolhidos quatro níveis de **textura**, onde não só serão aplicadas as suas texturas *diffuse*, mas também serão tidas em consideração as suas normais (confrontar secção 5).

### 3 RUÍDO

Como já foi referido na contextualização, foi necessário aplicar um ruído para a atribuição de uma nova altitude nos pontos de coordenada  $y$  da *grid*.

Para este fim, recorremos a uma função de ruído em que, a partir das outras coordenadas do ponto, é possível calcular alturas "aleatórias". Foram feitos dois tipos de implementações, uma com base numa textura e outra através de métodos conhecidos, sendo estes o *Fractal Brownian Motion (FBM)* e o *Simplex*. A implementação destes métodos teve como base códigos encontrados *online*, sendo que o primeiro foi implementado F. Kenton Musgrave ([3]), e o segundo faz parte de uma biblioteca de efeitos GLSL ([2]). O FBM é um processo gaussiano de tempo contínuo baseado em fratais e o *Simplex Noise* um método para construir uma função ruído semelhante ao *Perlin noise* mas com menos artefactos direcionais, em dimensões maiores e com um *overhead* computacional mais baixo.

No caso do ruído baseado numa textura, foi desenvolvido um algoritmo simples e estático que apenas realiza uma normalização dos pontos da *grid* de modo a estes ficarem entre 0 e 1 e abrangearem todos os valores da textura. Os únicos fatores dinâmicos neste ruído são a possibilidade de mudar a altura máxima dos pontos do terreno, assim como um valor mínimo da altura.



**Figura 1:** Textura de ruído  
[4]

O ruído criado com FBM e *Simplex noise* já é mais dinâmico e por sua vez pos-

sibilita a dinamização de alguns dos seus parâmetros, como é o caso da escala do ruído, da sua altura máxima, do ruído ser mais suave ou mais acentuado, entre outros. Por outro lado, o *Simplex noise* introduz alguns artefactos, sendo criados "picos" indesejados na nossa cena. Como este problema só acontece em pontos esporádicos, para o resolvermos foi criado um *smoother* que basicamente vai buscar as alturas dos quatro pontos adjacentes, calculando a sua média. Esta média é comparada com uma variância de ruído que pode ser alterada dinamicamente, através da interface, e caso o valor absoluto da sua altura a subtrair pela média for maior do que essa variância, é trocado o valor anterior da altura pela média.



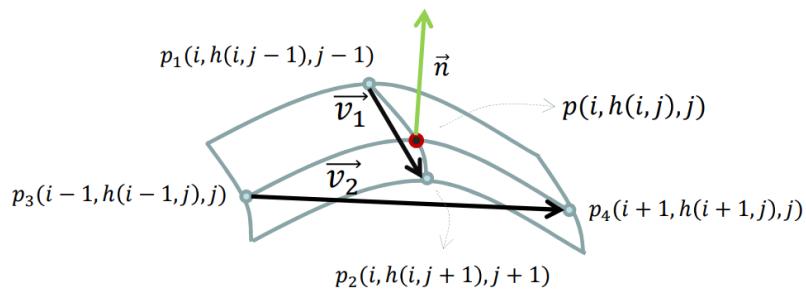
**Figura 2:** Cena com artefacto



**Figura 3:** Cena sem artefacto

## 4 ILUMINAÇÃO

Para criar um cena realista é fundamental que esta reaja à luz de forma dinâmica, ou seja, é necessário recalcular as normais da *grid*, já que, como foi referenciado anteriormente, estas são inicializadas com vetores a apontar no sentido positivo do eixo dos *yy*.



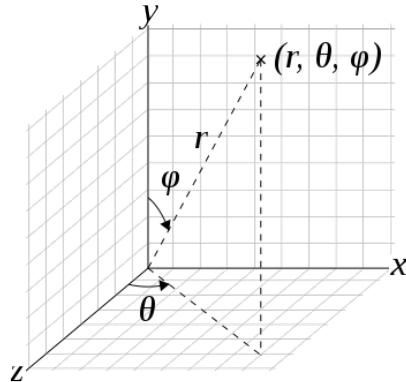
**Figura 4:** Derivadas parciais

[1]

Como se pode ver na figura 4, foram usadas derivadas parciais, recorrendo a valores dos quatro pontos adjacentes. É calculado um vetor entre cada par destes pontos, estando estes nos lados opostos do ponto a ser processado, e através de um produto vetorial calculamos a sua normal.

Para finalizar o cálculo das normais, este valor é normalizado e multiplicado pela matriz-transformação das normais (que é fornecida pela **Nau3D**) e normalizado outra vez. Há que ter em conta que, como os artefactos indesejados criados pelo ruído são tratados previamente, atualizando o valor da altura do ponto, estes artefactos não interferem no cálculo das normais.

Foi ainda criada uma luz dinâmica para recriar os diversos estados do sol durante o dia. Para este efeito, foi necessário passar a direção da luz (também esta fornecida pela *Nau3D*) de coordenadas cartesianas para esféricas, tomando uso do ângulo  $\phi$  para fazer variar esta direção em torno do eixo dos *yy*.



**Figura 5:** Coordenadas esféricas

```

1 float r = sqrt(pow(l_dir.x,2) + pow(l_dir.y,2) + pow(l_dir.z,2));
2 float phi = atan((sqrt(pow(l_dir.x,2)+pow(l_dir.y,2)))/l_dir.z) + (time-12)
   /12*pi;
3 float theta = atan(l_dir.y/l_dir.x);

```

O código acima representa a mudança do vetor direção de coordenadas cartesianas para esféricas. Pode assim ser observada uma variável `time` que vai dinamizar e representar o tempo em 24 horas, sendo que, quando `time == 12`, este encontra-se direcionado para a parte negativa do eixo dos `yy`.

No final, este vetor luz é passado de novo para coordenadas cartesianas, sendo de seguida normalizado, multiplicado pela matriz-transformada `view` e finalmente normalizado novamente.

De sublinhar que o referido até agora nesta secção foi implementado na parte do *Vertex Shader*.

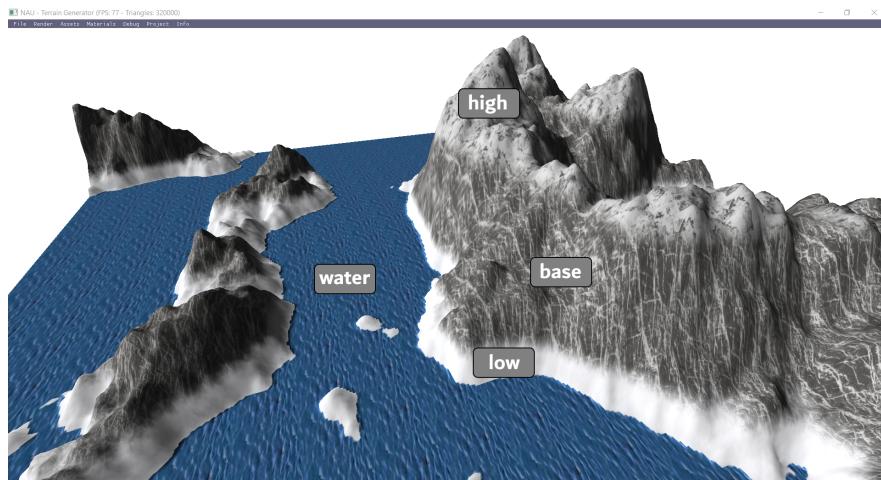
Na parte do *Fragment Shader*, para simular **Bump Mapping**, foram carregadas as normais das texturas correspondentes e usadas para complementar as normais do ponto previamente calculadas, aplicando um multiplicador a estas normais da textura para que estas não tenham grande influência no valor final normal, já que isto traduziria um resultado pouco realista. Este multiplicador pode ser ajustado dinamicamente através da interface.

Para o cálculo da intensidade da luz, é então aplicada uma constante de luz ambiente e somado ao valor máximo entre zero e o produto interno da normal (ainda agora calculada) e a direção da luz, é multiplicado por um `gamma` de valor dinâmico, e finaliza com uma soma com o valor de luz ambiente, mais uma vez cedido pela *Nau3D*.

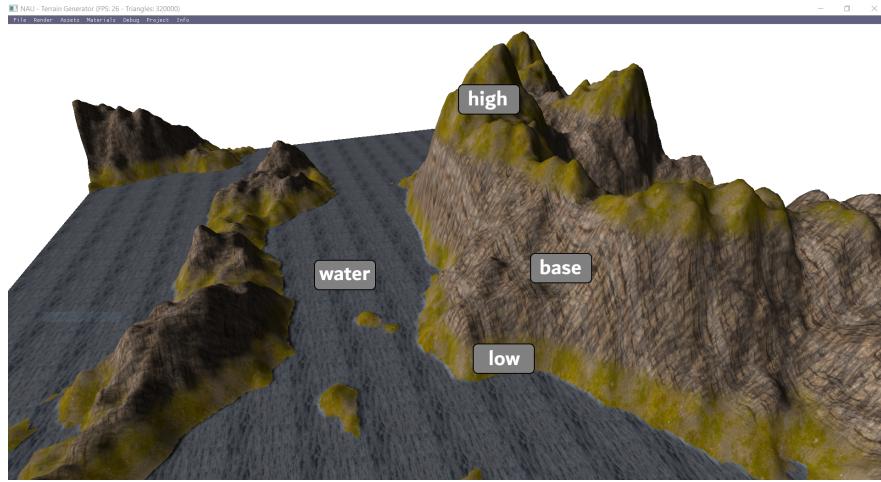
## 5 TEXTURAS

As texturas são somente tratadas no lado do *Fragment Shader*. Foram utilizadas 17 texturas diferentes, sendo que uma delas foi utilizada no cálculo ruído, e das outras 16 texturas 8 são difusas e as outras 8 são para as respetivas normais. Cada cena utiliza apenas metade da texturas, sendo criadas todas estas para que seja possível dinamizar a execução programa, escolhendo entre duas cenas. A quantidade de *tiles* de texturas utilizadas pode ser dinamizado através da interface.

Dependendo da cena ativa no momento, são carregadas as 8 texturas referentes a essa cena, onde cada par de texturas vai simbolizar a parte da água (water), a parte mais baixa (low), a parte intermédia (base) e a parte mais alta (high).



**Figura 6:** Visualização das diferentes texturas na cena 0



**Figura 7:** Visualização das diferentes texturas na cena 1

As texturas são assim aplicadas consoante a altura de cada ponto, podendo os quatro níveis em que queremos que as texturas apareçam serem definidos dinamicamente através da interface.

De modo a não haver uma mudança abrupta entre texturas, foi usada uma função *smoothstep*. Esta função possibilita-nos o *smoothing* entre as fronteiras das texturas. Para o cálculo destas é ainda aplicado um factor *random* baseado no ruído de uma textura, para acrescentar variabilidade na fronteira inter-textura. O resultado desta função vai-nos dar um valor entre 0 e 1, que será usado na função *mix* para escolher os valores das duas texturas. Por último, é multiplicado este valor calculado pelo valor da intensidade calculado na secção 4.



**Figura 8:** Transição entre texturas diferentes

## 6 INTERFACE

Como foi mencionado várias vezes ao longo dos capítulos anteriores, a nossa solução é bastante dinâmica. Esse dinamismo é proveniente da nossa Interface. Decidimos, então, criar *sliders* para valores que achamos importantes que pudessem ser alterados em tempo real, podendo ser visualizados na Figura 9.

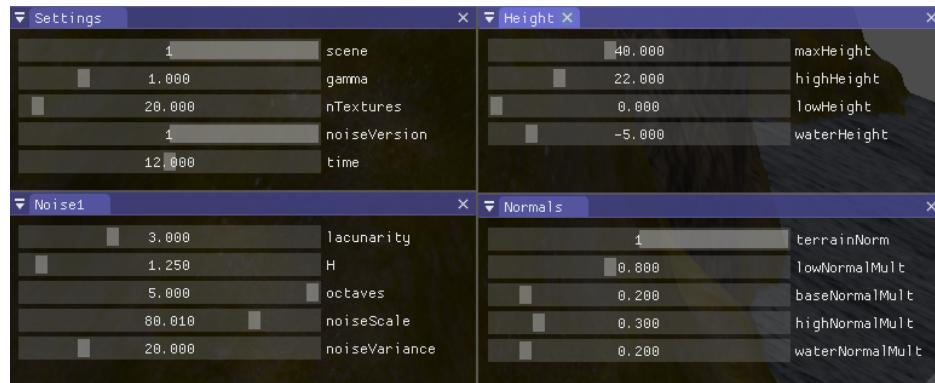


Figura 9: Interface

Nas **Settings** é possível escolher entre as duas cenas previamente escolhidas, através do *slider* *scene*. Temos um valor *gamma* que serve como fator multiplicador da intensidade da luz na cena, para o caso de ser necessário escurece-la ou clareá-la. Existe, também, um valor *nTextures* que deixa escolher a quantidade de vezes que uma textura se repete ao longo de um eixo da *grid*. Para a mudança dos dois tipos de *noise* existentes (textura e função) é utilizado o *slider* *noiseVersion*. Em último nas **Settings** encontra-se o valor *time* que pode variar entre 0 e 24 e simboliza as diversas posições da luz (sol) durante o dia.

A janela **Noise1** está apenas ativa quando o valor de *noiseVersion* se encontra no valor 1. Nesta janela temos 3 valores de alteração da função **FBM** do ruído, sendo estes a *lacunarity*, *H* e *octaves*, que devido à nossa combinação com o *simplex noise* têm alterações na *grid* bastante complexos. O valor *noiseScale* limita-se a esticar ou retrair o ruído. E por último temos o *slider* *noiseVariance* que, como referido anteriormente, tem como objetivo retirar os artefactos criados pela função de *simplex noise*.

Na divisão da **Height** encontramos um valor para decidir o tamanho máximo que queremos que as montanhas do nosso terreno tenham através do valor

`maxHeight`. Temos também os diversos valores para decidir a que nível de altura queremos que as texturas se encontrem, sendo 4 texturas difusas por cena e 3 interceções entre elas, temos também 3 *sliders* (`highHeight`, `lowHeight` e `waterHeight`).

Por último temos a janela **Normals**, onde se encontra um *slider* para decidir se queremos aplicar as normais da textura na cena de nome `terrainNorm`. Os restantes 4 *sliders* (`lowNormalMult`, `baseNormalMult`, `highNormalMult` e `waterNormalMult`), apenas funcionam caso o *slider* anterior se encontrar no valor 1 e servem como multiplicador às normais da textura para se quando aumentar ou diminuir a sua relevância para a normal final.

## 7 CONCLUSÃO

Com a realização deste trabalho prático foi-nos possível aplicar e aprofundar bastante (perante um tema específico) os conteúdos lecionados nas aulas de Visualização e Iluminação, tanto em termos do desenvolvimento em grupo do projeto em si e dos Shaders, como na apreciação crítica e melhoramento dos algoritmos utilizados.

O grupo conclui que pode ser feita uma consideração positiva deste projeto, já que, fazendo uma análise dos resultados visuais e de desempenho, depreende-se que os principais objetivos definidos para uma adequada geração de um terreno em *real-time* foram alcançados com sucesso e, deste modo, a realização do *assignment* foi de acordo com as expectativas.

## Referências

- [1] António Ramires Fernandes. «Terrain II». 2019. URL: [https://elearning.uminho.pt/bbcswebdav/pid-992208-dt-content-rid-2893419\\_1/courses/1920.H506N2\\_2/p10.pdf](https://elearning.uminho.pt/bbcswebdav/pid-992208-dt-content-rid-2893419_1/courses/1920.H506N2_2/p10.pdf).
- [2] *GLSL Sandbox Effect*. acedido em 15/2/2021. URL: <http://glslsandbox.com/e#58663.0>.
- [3] F. Kenton Musgrave. *FBM noise*. acedido em 15/2/2021. URL: <https://engineering.purdue.edu/~ebertd/texture/1stEdition/musgrave/musgrave.c>.
- [4] xd\_nitro. *Textura de perlin noise*. acedido em 15/2/2021. URL: <https://vvvv.org/contribution/tiled-perlin-noise-generator>.