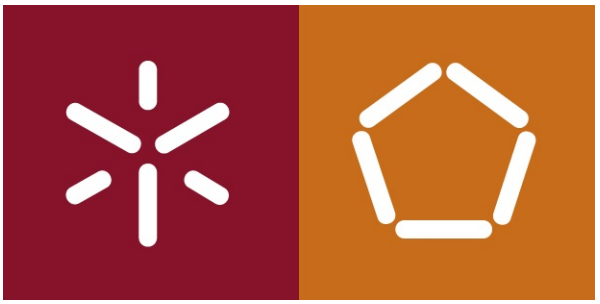

Universidade do Minho

Visualização e Iluminação 1

Terrain Generation

Autores:
Bruno Arieira
Daniel Vieira
João Palmeira

Números Aluno:
A70565
A73974
A73864

19 de Janeiro de 2019

Resumo

Um **fractal** é uma forma geométrica não regular que tem o mesmo grau de não regularidade em todas as escalas. Neste trabalho abordámos este sistema, mais especificamente dois dos seus tipos, o **Fractal Brownian Motion (FBM)** e **Hybrid Fractal**. Estes dois tipos de algoritmos são usados, no nosso trabalho, para a geração de terreno processual onde mostrámos as respetivas diferenças e o modo de implementação de cada um.

Conteúdo

Lista de Figuras

1 Introdução

Este trabalho prático foi realizado no âmbito da unidade curricular **Visualização e Iluminação I**, com principal objetivo de analisar a implementação de um determinado algoritmo utilizado no contexto de geração de um terreno processual, onde nos expressamos relativamente aos seus prós e contras, referenciando uma outra alternativa possível.

Terrain Generation é um método baseado na geração de um determinado terreno, cujo resultado pode se basear num dado mapa de alturas definido (exemplo, textura que tenha implícito ruído) ou até mesmo em funções que façam a geração desse ruído de forma aleatória.

Para a inicialização deste trabalho tivemos de efetuar várias pesquisas para encontrar algoritmos interessantes por forma a abordar perspetivas diferentes de geração de terreno. Assim, encontramos um documento, que nos despertou a atenção, desenvolvido por o professor e cientista da computação *Forest Kenton Musgrave*. De maneira a dar seguimento a este documento, baseamos o nosso trabalho em dois diferentes tipos de algoritmos, o ***Fractal Brownian Motion*** e o ***Hybrid Fractal***, sendo este último o método utilizado como principal referência.

Relativamente ao desenvolvimento deste relatório do trabalho prático, decidimos começar por introduzir o tema fulcral de **fractais**, fazendo referência á sua designação, e tipos que foram usados para este projeto. De seguida, passamos a especificar o modo como foi feita a implementação deste projeto, baseando nos no *vertex* e *fragment shader* de cada um dos tipos especificados, onde por fim, falamos da análise do desempenho da execução do nosso projeto, das dificuldades encontradas e conclusões retiradas.

2 Desenvolvimento

2.1 O que é Fractal?

A palavra fractal foi derivada de uma palavra latina *fractus* que significa fração, partido. Os fractais são figuras geométricas não euclidianas.

Um fractal é uma forma geométrica não regular que o mesmo grau de irregularidade em todas as suas escalas, pode ser dividido em partes, cada uma das quais semelhante ao objeto original. Diz-se que os fractais têm infinitos detalhes, são geralmente auto-similares e de escala. Em muitos casos um fractal pode ser gerado por um padrão repetido, tipicamente um processo recorrente ou iterativo.

Os fractais, em relação à computação gráfica, são imagens muito complexas geradas por um computador a partir de uma única fórmula. Estes são criados usando iterações, o que significa que uma fórmula é repetida com valores ligeiramente diferentes repetidas vezes, levando em conta os resultados da iteração anterior.

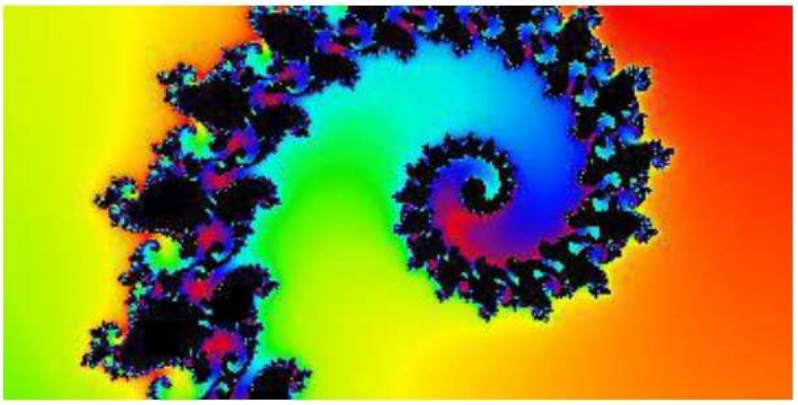

Figura 1: Computer graphics fractal

2.1.1 Geração de Fractais

Para gerar fractais, como referido anteriormente, pode-se utilizar a mesma forma repetidamente, como mostrado na figura seguinte. Na figura (a) é apresentado um triângulo equilátero. Na figura (b), podemos ver que o triângulo é repetido várias vezes para criar uma forma de estrela. Na figura (c), podemos ver que a forma da estrela na figura (b) é repetida várias vezes para criar uma nova forma.

Podemos fazer um número ilimitado de iterações para criar uma forma desejada. Em termos de programação, a recursividade é usada para criar essas formas.

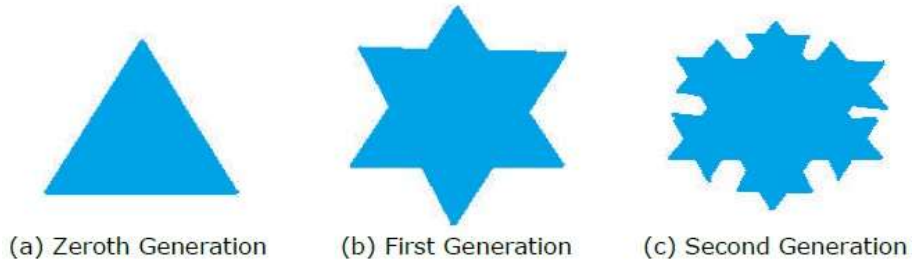

Figura 2: Geração de fractais

Os fractais geométricos lidam com formas encontradas na natureza que possuem dimensões não inteiras (fracionarias) ou fractais. Para construir geometricamente um fractal auto-similar determinístico (não aleatório), começamos com uma dada forma geométrica, chamada de **iniciador**. As sub-partes do iniciador são então substituídas por um padrão, chamado **gerador**.

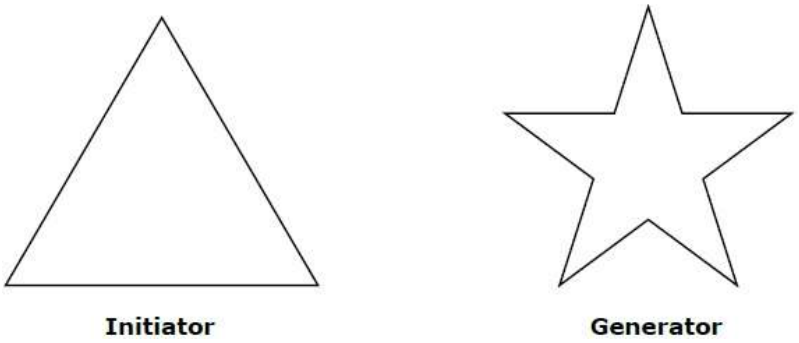

Figura 3: Iniciador e Gerador

2.2 Tipos de Fractais Utilizados

Relativamente aos vários algoritmos que existem para a criação de fractais, neste trabalho iremos dar ênfase a dois deles: *Fractal Brownian Motion* e *Hybrid Fractal*.

2.2.1 Fractal Brownian Motion (FBM)

Uma onda é uma flutuação ao longo de tempo de alguma propriedade. Para uma onda há duas caraterísticas: amplitude e frequência. E é com base nestes conceitos que podemos explicar o **FBM**.
Na música sabemos que cada nota tem uma frequência definida em que cada frequência segue um padrão a que chamamos de escala, sendo que metade ou o dobro de uma frequência corresponde a um `octave`.
Um bom gerador de números aleatórios produz números que não são relacionados e que não mostram nenhum padrão entre eles. Portanto podemos usar a aleatoriedade para programar comportamentos orgânicos, semelhantes à vida real. No entanto, a aleatoriedade como princípio único

de orientação não é necessariamente natural. E é com este pensamento que chegamos ao **Ruido Perlin**.
Este ruído é muito parecido com uma onda sinusoidal no entanto tem uma aparência mais orgânica porque produz uma sequência naturalmente ordenada ("suave") de números aleatórios. O gráfico abaixo mostra o ruído de *Perlin* no decorrer de um determinado tempo, sendo que o eixo x representa o tempo. Observe a suavidade da curva.

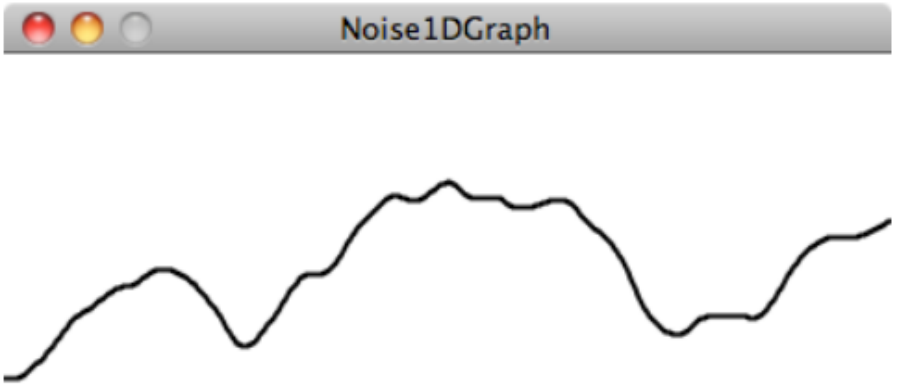

Figura 4: Ruido Perlin

Usando o ruído anterior, podemos adicionar iterações diferentes de ruído(*oitavas*), em que cada um deles são adicionados sucessivamente as frequências (*lacunaridade)* bem como a retirada da *amplitude*. Com isto podemos obter detalhes mais precisos.

2.2.2 Hybrid Fractal

O tipo de fractal descrito acima é um método homogéneo e isotrópico, já que aplica-se o mesmo em todo o terreno, e em todas as direções desse mesmo terreno. Se nos deparar-mos com o que realmente acontece relativamente a fenómenos fractais, por exemplo, as montanhas não são uniformes, assim como não contém a mesma rugosidade em todos os lugares. Com isto, surgiu a necessidade da criação de funções mais heterogéneas com principal objetivo de obter maior realismo, para geração de terrenos.

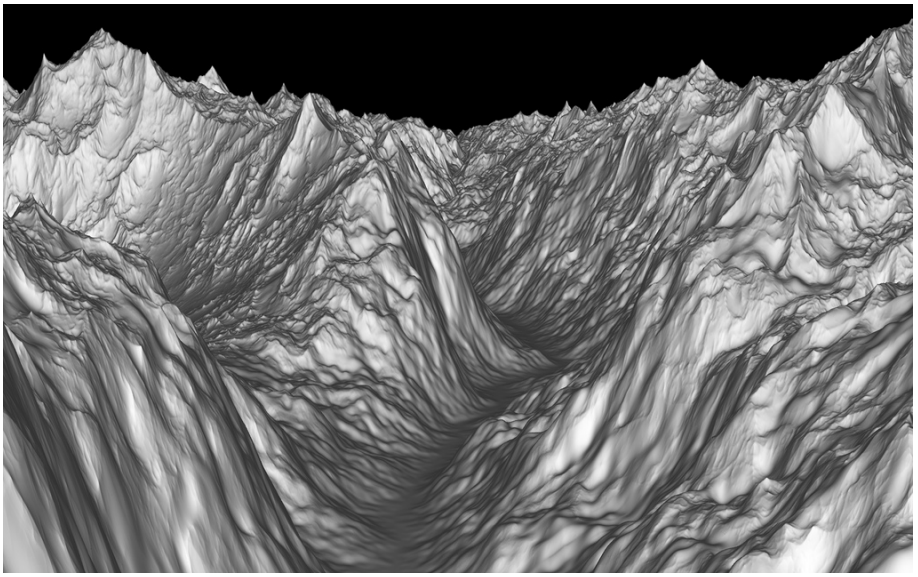

Figura 5: Exemplo de uma implementação Hybrid MultiFractal

Na implementação desenvolvida por *Musgrave*, a ideia é que o piso dos vales são lisos quando comparados com as montanhas. Assim sendo, foi necessário fazer algumas alterações na função, alterações estas que permitem a geração de um terreno com as características mais realistas como será explicado mais á frente.

3 Implementação

Para o desenvolvimento do tema *Terrain Generation*, para além de utilizarmos as duas técnicas mencionadas anteriormente, baseamos-nos numa técnica relacionada com o ruído onde para cada ponto de uma grelha é calculada a altura que cada um dos vértices irá ter.

Ken Perlin implementou um algoritmo designado por *Perlin Noise* em 1983 e melhorou esse mesmo algoritmo designando-o por *Simplex Noise*. Optamos por utilizar a versão melhorada nesta implementação visto que a sua eficiência é superior ao anterior algoritmo.

O algoritmo de *Simplex Noise* utilizado nesta implementação foi retirado de um repositório do Prof. António Ramires, logo não é da autoria do grupo, tal como as funções relativas aos fractais que foram retiradas de um repositório na Internet do trabalho realizado por *Forest Kenton Musgrave*.

De resto todo o código é da autoria do grupo e foi implementado em GLSL utilizando a Nau3D como suporte para o uso dos dois *shaders* utilizados para cada tipo de implementação (FBM e *Hybrid Fractal*): *vertex shader* e *fragment shader*.

Nas secções seguintes, vai ser analisado o código de cada um dos *shaders* desenvolvidos para as duas implementações e explicando cada um deles.

3.1 Vertex Shader

Começando pelo *vertex shader* onde o esqueleto do código é idêntico tanto na implementação FBM como na *Hybrid*. É neste *shader* que através da função de ruído *Simplex* e, consoante a versão, através da função FBM ou *Hybrid* se vão efetuar os cálculos para a altura de cada vértice devolvendo para o *pipeline* a posição final (altura) de cada vértice da grelha. O cálculo de cada altura é feito essencialmente através das funções FBM ou *Hybrid* que recebem como parâmetro a posição de um vértice (apenas as coordenadas x e y), a lacunaridade e os *octaves* que funcionarão para o cálculo do valor do *noise* do ponto, tomando em atenção que será criado nestas funções um *array* de frequências consoante o número de *octaves* para auxiliar o cálculo do valor descrito acima.

Como foi dito anteriormente, a função do ruído também participa no cálculo através da invocação da função *snoise* por parte das duas anteriores que irá elaborar um conjunto de valores para os diferentes valores de *octaves* e de lacunaridade, onde posteriormente serão somados após efetuada a multiplicação pela frequência correspondente. A frequência utilizada para multiplicar pelo valor retornado pela função *snoise* determina, na sua essência, a variação do valor retornado. Dito de outra forma, uma baixa frequência resultaria num terreno com variações mais suaves, e uma alta frequência em variações muito mais repentinas. Somando os valores de *noise* retornados pela função *snoise* calculados com frequências diferentes, o resultado final inclui vários tipos de variações, permitindo, no contexto de *terrain generation*, criar vários fenómenos geológicos de dimensões diferentes.

Por fim, será retornado o valor da soma do conjunto de valores referidos antes por uma das funções (FBM ou *Hybrid*).

3.1.1 Fractal Brownian Motion (FBM)

Devido à complexidade da função **FBM** para um melhor entendimento da mesma decidimos usar um exemplo que consegue, no nosso ponto de vista, demonstrar de forma concreta da formação relativa a um fractal. Por agora pensemos numa função que corta uma bola de queijo em pedaços, e tendo isso em mente, iremos construir um fractal a partir disso, ou seja, iremos cortar a bola de queijo em inúmero pedaços de diferentes tamanhos e diferentes formas, onde mais tarde serão compactados, tanto em altura como em largura, formando de novo uma bola de queijo embora com uma nova disposição de todos os pedaços cortados. Através deste processo metafórico foi construída a ideia de como formar o fractal.
Cada tamanho do pedaço irá ser a **frequência** da função e a sua altura a **amplitude**. Com esta explicação podemos falar da função propriamente dita. No inicio iremos ter um *array* em que cada índice do *array* irá ter uns valores diferentes devido a frequência definida que irá receber. Através de um ciclo, iremos percorrer todos os *octaves* e retornar um valor que será a multiplicação entre o **snoise** e o valor de cada índice do *array* criado anteriormente.

Consequentemente, o que queremos fazer com esta função é gerar n valores compostos pelo ruído, através da função da **snoise**, e por partes de cada fractal criado no ciclo anterior.

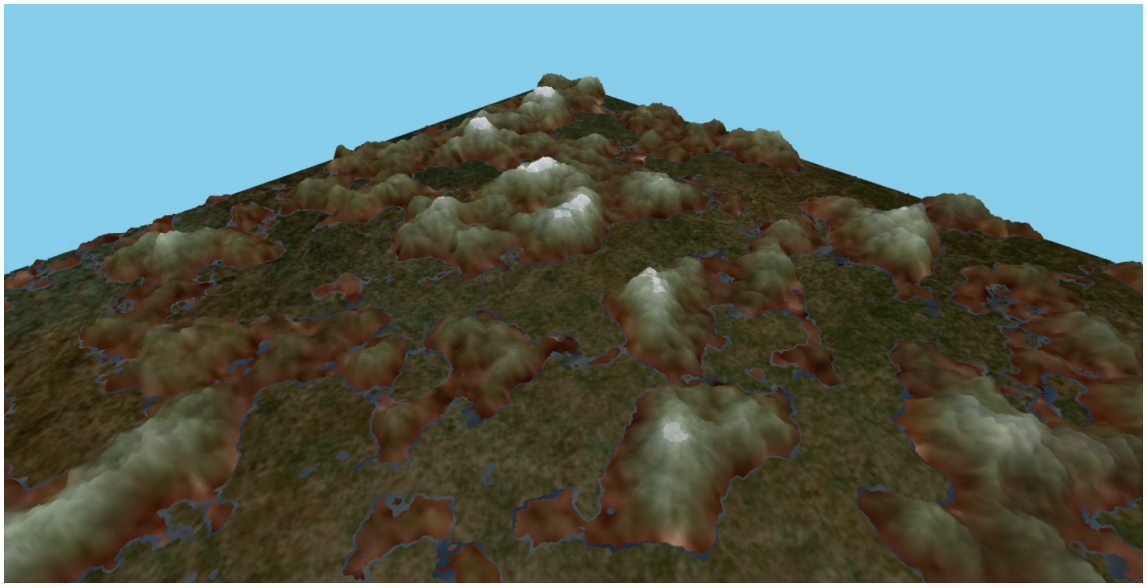

Figura 6: Exemplo de terreno gerado utilizando o método FBM

3.1.2 Hybrid Fractal

O que difere entre o **Hybrid Fractal** e o **Fractal Brownian Motion (FBM)** é que no primeiro há uma diferença clara entre as *octaves*, sendo que na primeira *octave* vai depender do primeiro índice do array, logo de uma parte de um possível fractal, sendo que os restantes irão ser diferentes visto que vai usar todos os valores do array antes criado. Como mencionamos anteriormente, vai haver uma clara diferença entre o primeiro nível (piso dos vales) e o resto, sendo o primeiro mais liso que os restantes, e consegue-se devido a uma diferença entre a primeira *octave* e os restantes

visto que neste últimos irá haver uma mistura de várias partes de fractais. É de notar que o *offset* é aplicado á função *snoise*, com objetivo de mover o seu intervalo de [-1,1] para algo mais próximo de [0,2].

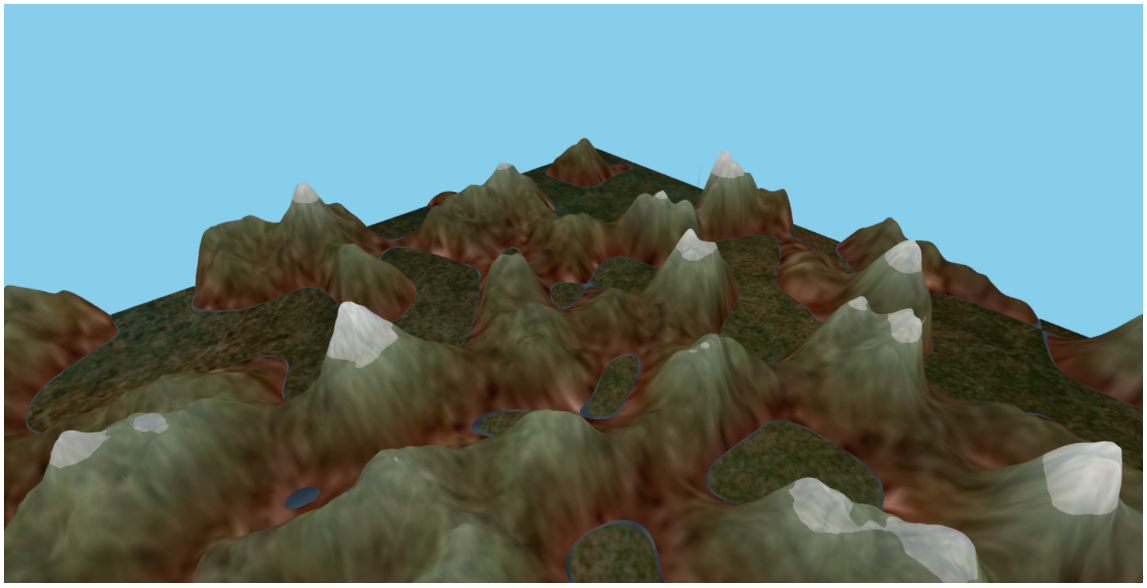

Figura 7: Exemplo de terreno gerado utilizando o método *Hybrid*

3.1.3 Câmera

Ainda no *vertex shader* de ambos os casos fora modelados os parâmetros da câmera através de duas técnicas. Numa primeira instância às componentes x e z da câmera são lhe adicionadas as mesmas componentes relativas a cada vértice, uma vez que desta forma a grelha que define o terreno irá acompanhar o movimento da câmera dando ao utilizador uma sensação de "terreno infinito"como é visível no projeto.

Numa segunda instância, é utilizada a função *floor* para as posições da câmera garantindo que os vértices da grelha mudem de posição apenas quando a câmera avança uma unidade no plano XZ, uma vez que a esta pode tomar posições decimais. Caso esses valores sejam tomados, os triângulos do terreno mudam de posição de forma continuada sempre que existe movimento da câmera conferindo ao mesmo um efeito mais "ondulatório".

3.2 Fragment Shader

Relativamente a este *shader* é de realçar que é utilizado para calcular a cor de cada pixel através das texturas que fornecemos. De referir que são efetuadas várias misturas das texturas que fornecemos para que as transições entre texturas não transmitam a olho nu uma sensação transição abrupta, mas sim uma sensação de transição suave, mais conforme o que acontece no mundo real. Estas misturas foram elaboradas através da escolha de duas das texturas fornecidas que funcionam como parâmetros de início e fim da interpolação e com um valor de interpolação aleatório entre as duas

texturas fornecido através do cálculo de um *float* aleatório consoante a altura onde vai ser aplicada a mistura.

De realçar que também foi elaborada uma função de suavização para aplicar a essas mesmas texturas de forma a dar relevância aos tons mais realistas e mais naturais das texturas utilizadas. O objetivo desta estratégia é fazer com que no terreno como um todo sejamos capazes de distinguir perfeitamente os diferentes tipos de terreno consoante a altura do mesmo e que este esteja com um aspeto geral realista.

4 Análise de Desempenho

Relativamente a este tópico decidimos testar várias grelhas com diferentes divisões e tamanhos ao longo dos eixos e, com ajuda da Nau3D, geraram-se *logs* de cada teste. Desta forma, testamos a escalabilidade dos *shaders*, utilizando uma placa gráfica *NVIDIA GeForce GTX 1060* de 6GB (DDR5) dedicados, onde os tamanhos e o número de divisões testados foram 4092, 2048 e 8184 para ambos, num total de 9 grelhas diferentes testadas. Visto que usamos a função **snoise**, que segundo o autor da mesma é eficiente, fizemos questão de testar e portanto aumentamos o número de `octaves`. Para cada aumento não houve muita diferença no tempo de execução, relativamente ao algoritmo FBM, pelo que podemos confirmar a afirmação do autor. Já para o algoritmo *Hybrid* o mesmo não acontece.

No ficheiro **XML** fizemos as respetivas alterações do comprimento não havendo alterações significativas. Só com valores elevadíssimos como 20000 em `LENGTH` é que houve uma mudança drástica de tempo (mais do dobro) e uma diminuição drástica de FPS. Sendo que para a *Hybrid* para o comprimento de 8184 já se denotava uma diferença bastante significativa do seu tempo de execução.

Modificamos os valores de **DIVISIONS** de forma a tentar compreender como o tipo de fractal FBM se comporta: 2048 com 0.346000 ms, 4096 com 0.373000 ms e 8184 com 0.394000 ms. Para o mesmo número de **DIVISIONS**, para o outro tipo (*hybrid*): 2048 com 0.636000 ms, 4096 com 1.658000 ms e 8184 com 6.838000 ms, notando-se uma diferença significativa, situando-se esta diferença no calculo das normais no *vertex shader*.

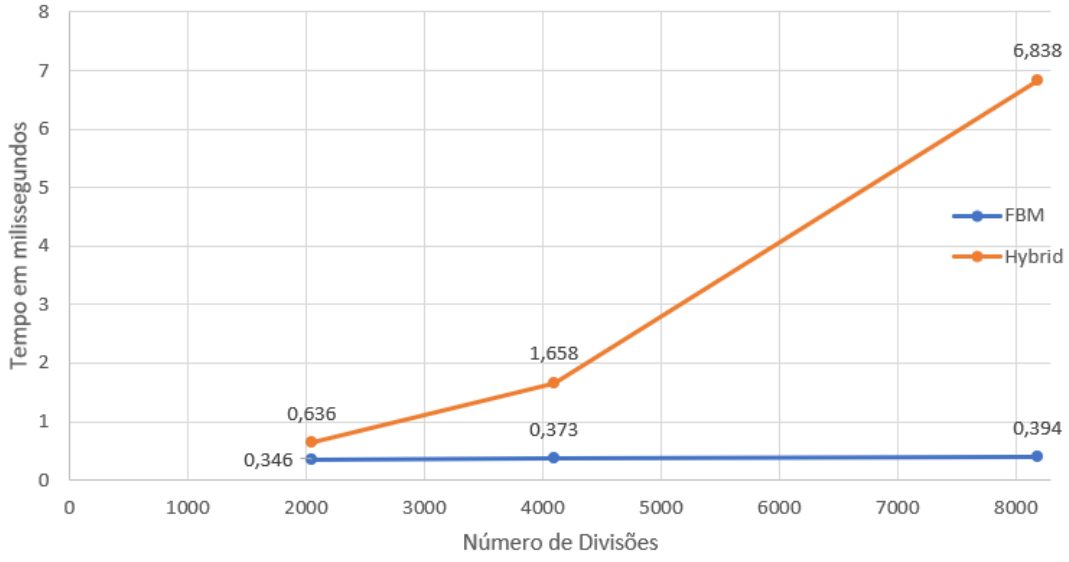

Figura 8: Gráfico ilustrativo dos tempos de execução de cada um dos algoritmos abordados

Após uma análise do gráfico, chegamos à conclusão que à medida que aumentamos o número de divisões, o tempo de execução da versão ***Hybrid*** aumenta de uma forma bastante mais acentuada do que da versão **FBM**.

5 Considerações

Tomando em consideração que a implementação do trabalho não foi o único foco do grupo, achamos necessário salientar que existem alguns aspetos que são necessários melhorar na nossa abordagem do tema.

Primeiramente o aspeto visual do terreno gerado. O grupo acha que o produto final é agradável embora tendo noção que existem vários aspetos a melhorar tais como as misturas e as transições das texturas de modo a tornar o terreno ainda mais realista. Uma vez que apesar de termos utilizado várias texturas, a transição, por exemplo, entre o terreno normal (área mais esverdeada) e a neve não foi a melhor e talvez fosse necessário criar mais irregularidades de modo a transparecer ao utilizador que o que foi gerado corresponde mais ao que existe no mundo real.

Outra situação que não conseguimos implementar, foi no sentido da geração de terreno automática em relação á fixação das texturas, isto é, conseguimos gerar o terreno de forma automática, mas á medida que a câmara se move, o terreno vai sendo gerado e as texturas vão-se movendo, algo que não deveria acontecer.

6 Conclusão

Com a realização deste trabalho prático, inserido no âmbito da unidade curricular de Visualização e Iluminação I, em relação aos conteúdos apreendidos nas aulas práticas, podemos concluir positivamente relativamente á execução deste projeto, tanto relativamente aos algoritmos utilizados para o desenvolvimento do projeto bem como a criação do *vertex* e *fragment shader* para cada um deles.

De modo a criamos várias soluções para o tema escolhido, o grupo decidiu seguir a metodologia do Sr. *Ken Musgrave* que elaborou diferentes métodos com o auxílio de fractais para responder ao problema do ruído do terreno. Por isso, ao utilizarmos as técnicas FMB e *Hybrid*, o grupo chegou à conclusão que a segunda técnica gera um terreno mais agradável, visto que existe mais irregularidades no terreno tornando-o mais realista.

Após efetuadas todas as considerações, o grupo conclui que o projeto realizado está de acordo com as expectativas, pois após analisar os resultados visuais e de desempenho chegamos à conclusão que os principais objetivos foram alcançados com sucesso. Concluindo, apesar de existirem vários aspetos que podem ser melhorados de forma a criar mais realismo no terreno, o objetivo de criar um programa capaz de gerar terreno aparentemente infinito em tempo real foi alcançado.

Referências

[1] https://www.classes.cs.uchicago.edu/archive/2015/fall/23700-1/final-project/MusgraveTerrain00.pdf

[2] https://thebookofshaders.com/13/