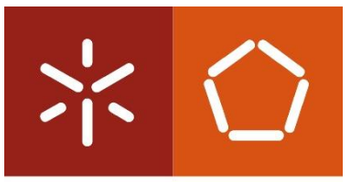

Universidade do Minho

Computação Gráfica

Fase 4

Benjamim Coelho, Henrique Neto, Leonardo Marreiros e
Júlio Alves

e-mail: {a89616,a89618,a89537,a89468}@alunos.uminho.pt

30 de maio 2021

1 Introdução

Para esta fase tínhamos objetivos para o engine e para o generator. No engine, tínhamos de implementar as infraestruturas para permitir o uso de luzes e texturas, e consequentemente tinhamos de implementar os mecanismos para poder diferenciar os varios materiais envolvidos. No generator, temos de o modificar de forma a permitir que a aplicação seja capaz de calcular para cada modelo as normais de cada ponto e as respetivas coordenadas num plano de textura em duas dimensões.

2 Arquitetura do projeto

2.1 Engine

2.1.1 Models

Os modelos forem mais uma vez extensivamente mudados pois para alem dos novos valores associados aos normais e ás texturas, estes agora encontram-se a ser implementados por *Triangle Strips*, resultando assim numa nova classe *StripedModel* que como todos os modelos, é uma extensão da classe abstrata *Models*. Adicionalmente foi necessário implementar um modelo para tratar especificamente da *skybox* da aplicação.

Semelhante ao que tínhamos na fase anterior, cada modelo é instanciado uma única vez em memoria de maneira a não desperdiçar recursos desnecessáriamente.

StripedModel

Esta classe trata-se de um novo modelo que é desenhado com recurso a *triangle strips*. Desta forma, este modelo pode chegar a instanciar 4 buffers virtuais correspondentes ás coordenados dos pontos, ás normais repetivas, ás coordenadas no espaço de texturas e aos índices das respetivas *strips*. Para alem disto, existe um vetor de inteiros que indica de maneira sequencial o comprimento de cada strip a ser desenhada.

Desta forma, é possível desenhar um modelo de forma a obter a iluminação esperada e consequentemente a respetiva textura.

Skybox

A skybox corresponde a uma interface que faz recurso aos *stripped models* e ás *textures* para criar uma caixa em volta da cena. Desta forma esta classe instância referencias ao respectivo modelo da cena e ás texturas envolvidas no seu desenho.

Para desenhar a respetiva caixa, as *strips* do modelo sao dividas em 6 blocos contíguos ("top", "bottom", "left", "right", "front", "back") no qual se associam a respetivas texturas. De seguida esta é centrada na posição câmara e desenhada sem ter em consideração o *buffer* de profundidade. Desta forma, a caixa fica estática e independente ao seu tamanho, tendo a consequência de ser necessário desenha-la antes da cena correspondente para que tudo o que a suceda a sobreponha caso necessite.

Para especificar a skybox no ficheiro xml é necessário apresentar este elemento na cena:

$< skybox\ model = "modelo"\ culling = ("inverted"\ ou "normal")\ top = "textura"\ bottom = "textura"\ left = "textura"\ right = "textura"\ front = "textura"\ back = "textura" / >$

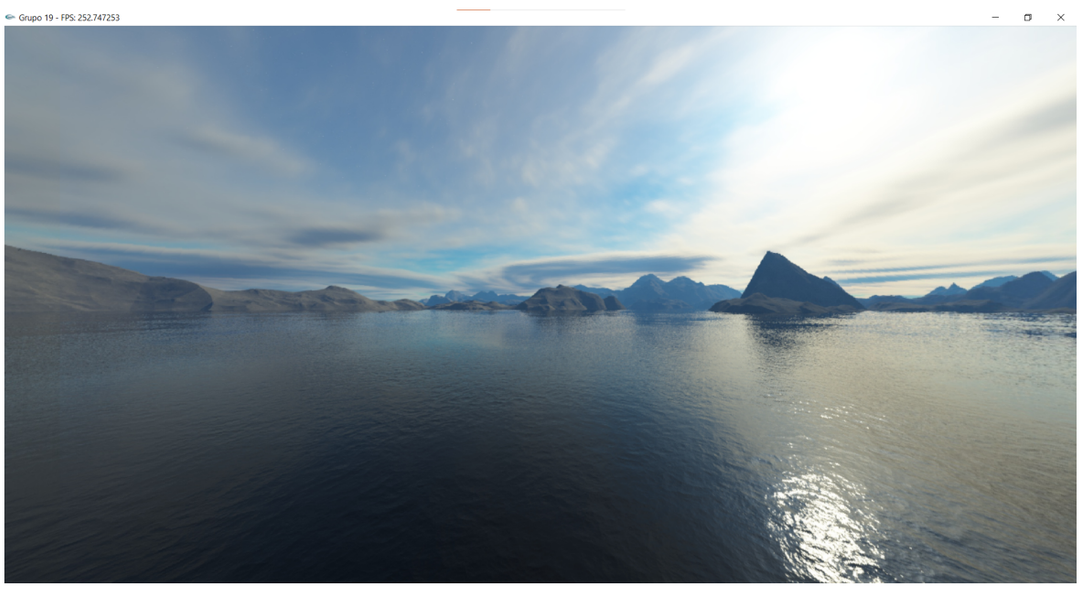

Figura 2.1: Exemplo de uma *skybox* no engine

2.1.2 Textures

O nosso programa permite também, neste momento, a utilização de texturas em mapeamentos de duas dimensões, através da utilização do seguinte argumento. $<$ *modeltexture* = "*textura*"/ $>$ É de salientar que a nossa aplicação só guarda cada textura uma vez em memória para o caso em que uma textura seja utilizada várias vezes, teremos apenas uma instância em memória dessa textura, o que, obviamente, nos permite poupar em memória.

2.1.3 Materials

Tendo em conta a nova implementação das luzes, o *engine* possui agora a capacidade de ler e processar materiais constituídos pelas respeticas componentes ambiente, difusa, especular, brilho e emissiva.

2.1.4 Lights

Em conjunto com os materiais referiados anteriormente, um novo tipo de elemento que permite deinir fontes de luz na cena. Para tal for criada uma classe *Light* que faz partido da classe *LightParameters* para poder renderizar a luz na cena. Dado às limitações da versão usada do opengl, só é possível instanciar no máximo 8 luzes.

LightParameters

Esta classe especifica os parâmetros evolvidos na rasterização da luz. Com isto esta classe instancia variáveis para as componentes ambiente, difusa, especular, *spotDirection*, *attenuationFactor*, *spotExponent*, *spotCutOff* e o tipo de atenuação. Quando se chama o metodo *set(GLenum light)*, a classe, os parâmetros instanciados são aplicados à luz especificada no enum fornecio.

2.1.5 Groups

Para esta fase, permitimos que os *Groups* tenham referencias ás texturas e aos materiais. Desta forma, estas são referenciadas no momento em que são desenhados os modelos.

2.1.6 Scene

De maneira a suportar os novos elementos, a classe *Scene* instância duas novas listas, correspondentes aos materiais e texturas presentes nesta juntamente com a instância da skybox respectiva, se esta for especificada no ficheiro *xml* respetivo.

2.2 Generator

Nesta fase, o *Generator* foi alterado de forma a gerar as coordenadas das normais e as coordenadas das texturas de cada ponto. Além disso, foi alterada também a forma como os ficheiros são escritos passando para *strips*

2.2.1 Novo formato do ficheiro 3d

Na fase anterior do projeto, havíamos alterado o formato dos ficheiros 3d para que não houvesse repetição de pontos e nesta fase decidimos voltar a fazer alterações nos ficheiros 3d, uma vez que passamos a utilizar strips para a construção de triângulos, para que haja uma redução de um terço no número de indíces escritos. Isto fará com que haja um aumento relevante na performance do programa pois terá bastantes menos indices para ler e já os terá em memória.

2.2.2 Calculo das Normais

Plane

Um plano apenas tem duas normais: (0,1,0) para os pontos da face voltada para cima e (0,-1,0) para os pontos da face voltada para baixo.

Box

Quanto ao cubo, a cada face irá corresponder uma normal: para as faces paralelas ao plano XZ, os vértices da face de cima tem normal (0,1,0) e os da face de baixo (0,-1,0); quanto às faces paralelas ao plano YZ, os vértices da face da esquerda tem normal (-1,0,0) e os da face direita (1,0,0); finalmente, para as faces paralelas ao plano XY, os vértices da face da frente tem normal (1,0,0) e os da esquerda (-1,0,0)

Cone

Os normais de um cone são divididos em duas partes: a base, e a superfície. Para a base, a normal correspondente é (0,-1,0); Para os vértices da superfície a normal correspondente é obtida pela fórmula $(r * \sin(a), \cos(arctg(h/R), r * \cos(a))$ em que R corresponde ao raio da base do cone, r ao raio da stack do ponto, h à altura do cone e a ao ângulo da subsecção do cone correspondente a uma slice.

Sphere

Para a esfera, a normal correspondente a um ponto da sua superfície é dada por $(\sin(\beta) * \cos(\alpha), \cos(\alpha), \sin(\beta) * \sin(\alpha))$.

Patch

Por fim, para determinar as normais das patches de bezier foi necessário para cada ponto calcular as tangentes a cada ponto segundo o espaço u,v segundo as formulas :

$$\frac{\delta B(u,v)}{\delta u} = [3u^2 \ 2u \ 1 \ 0] * M_B * G_B * M_B^T * V^T$$

$$\frac{\delta B(u,v)}{\delta v} = U * M_B * G_B * M_B^T * [3v^2 \ 2v \ 1 \ 0]^T$$

onde

$$G_{Bk} = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

Desta forma é possível calcular a normal ao pontos fazendo o produto vetorial entre estas tangentes e assim conseguimos calcular as normais de quase todos os pontos.

Contudo, surgem problemas em patches com pontos degenerados, ou seja, em patches em que $\exists_i, \forall_{j,k \in \{0,1,2,3\}} : P_{ij} = P_{ik}$. Nestes casos não existe solução para além de aproximar a normal. Para tal, estamos a considerar que nestes pontos, as normais ás superfícies, correspondem norma da soma das tangentes dos triângulos adjacentes. Assim, para cada triângulo adjencente são calculados dois vetores entre o ponto em questão e os restantes vértices do triângulo, e sucessivamente a normal do triangulo é obtida através do produto vetorial entre eles como são apresentados na figura 2.2.

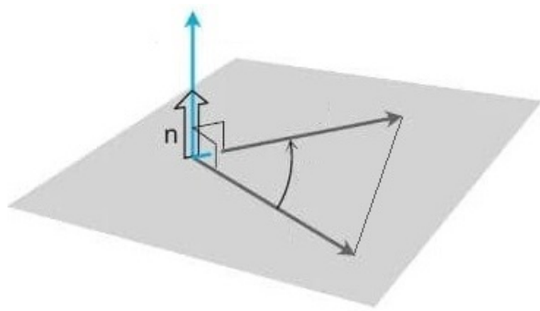

Figura 2.2: Calculo da normal no triângulo adjacente ao ponto

Por fim, as normais obtidas por todos os triângulos envolvidos são somadas e o vector resultante é normalizado. Com esta aproximação conseguimos garantir que não foi perdida a iluminação, no ponto em que ocorreu esta falha.

2.2.3 Calculo das Coordenadas de textura

Plane

Para cada ponto do plane, for mapeado os respetivos cantos (0,0);(0,1);(1,0);(1,1) para que desta forma a textura fosse interpolada em todo o plano.

Box

Para cada lado da caixa foi mapeado a textura individualmente. Desta forma, cada textura é impressa em cada lado de maneira contigua. Adicionalmente, o interpretador do modelo pode separar ambas as faces a aplicar uma textura diferente entre cada elas. Atualmente, as nossas *skyboxes* fazem uso desta propriedade.

Cone

As coordenadas de textura de um cone baseiam-se em um plano de texturas contendo dois círculos. Um encontra-se centrado no ponto (0.25,0.5) e corresponde a textura da base do cone. O outro encontra-se centrado no (0.75,0.5) e corresponde a textura da parte lateral do cone, e desta forma, a textura em cada ponto é interpolada com base na posição no circulo correspondente.

Sphere

Para calcular as coordenadas de textura da esfera, o espaço de texturas é dividido igualmente pelas várias *slices* e as *stacks* correspondentes. Desta forma para cada *slice* i e *stack* j cada coordenada de textura corresponderá ao par $(j/slices, j/stacks)$

Patch

As coordenadas de textura de um *patch* têm apenas um elemento: (u,v) onde u é o valor da tesselagem em u e v é o valor da tesselagem em v.

2.3 Utils

2.3.1 Point

Devido à necessidade de implementar os vetores normais e as coordenas, a classe *Point* definida em outras fases foi renomea-da como *SimplePoint*, sendo a classe atual uma expansão direta a essa classe, que para alem das coordenadas no espaço local, contém também a normal referente ao ponto e as respetivas coordenadas no plano de texturas.

3 Construção de cenas

3.1 Sistema Solar

Fazendo uso das novas funcionalidades implementadas, é possível agora adicionar texturas e luzes aos diferentes elementos do Sistema Solar.

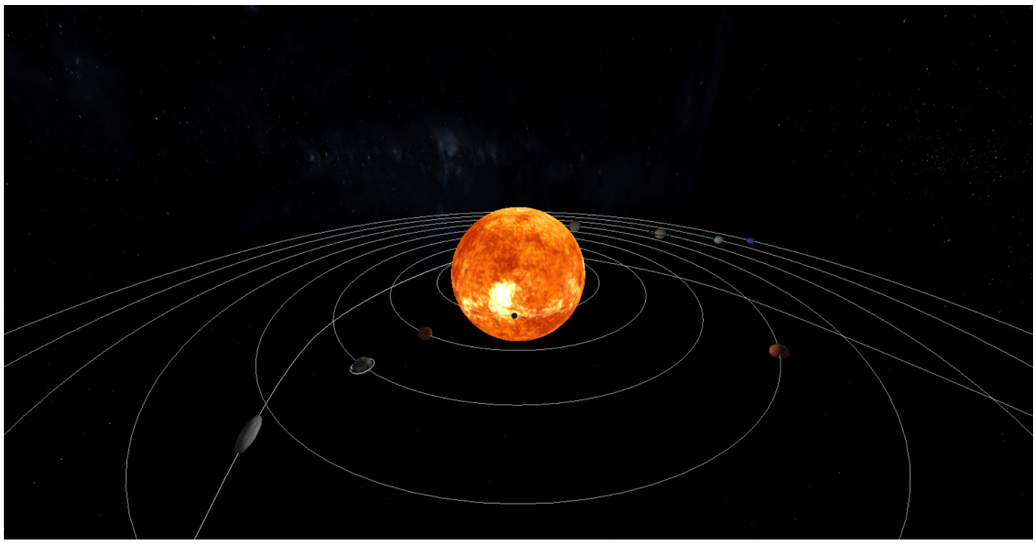

Figura 3.1: Modelo do Sistema Solar

4 Conclusão

Com o término desta fase, concluímos assim o trabalho prático de Computação Gráfica. Nesta fase, calculamos as normais e coordenadas para as texturas dos modelos que permitiu a implementação de vários tipos de iluminação e a aplicação de texturas aos modelos. Além disso implementou-se uma *skybox* e a mudança da escrita e leitura das coordenadas para *strips* o que aumentou bastante a eficiência e reduziu a quantidade de informação repetida nos ficheiros .3d.

Consideramos que com a realização desta fase, e deste trabalho prático, consolidamos bem os conhecimentos adquiridos nas aulas.

Como trabalho futuro gostaríamos de implementar uma nova câmara e implementar a primitiva do tórus.