

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Computação Gráfica
Texturas e Iluminação

Bárbara Cardoso (a80453) Márcio Sousa (a82400)
Pedro Mendes (a79003)

12 de março de 2022

Conteúdo

1 Introdução

Este trabalho foi proposto no âmbito da unidade curricular de Computação Gráfica, e tem como objetivo desenvolver um motor gráfico genérico para representar objetos a 3 dimensões. Este relatório tem como contexto ambos os relatórios anteriores.

Nesta quarta fase do projeto estendemos as capacidades do motor gráfico anteriormente definido com a adição de luzes e texturas. Foram também feitas algumas otimizações a este.

O gerador de modelos foi também estendido para calcular para cada uma das primitivas os vetores normais e as coordenadas de textura das mesmas.

Por fim, o modelo de teste utilizado (O Sistema Solar) foi alterado para demonstrar estas capacidades, fazendo uso de texturas para os planetas e Sol, assim como colocar iluminação neste.

2 Generator

O output do *generator* foi alterado para incluir as normais e as coordenadas de textura correspondentes aos pontos dos triângulos calculados. Em seguida, passamos a explicar o processo do cálculo das normais e das coordenadas de textura para cada uma das primitivas.

2.1 Nova Primitiva: Cilindro

Foi adicionado ao gerador a primitiva cilindro. Este é definido pelo raio da base, altura e número de *slices*.

Em cada iteração são definidos quatro pontos usando coordenadas polares, estes quatro pontos são os cantos da *slice* respetiva, depois usando estes são desenhados quatro triângulos, um para a base, outro para o topo e por fim outros dois para a fazer o retângulo que compõem a *slice*.

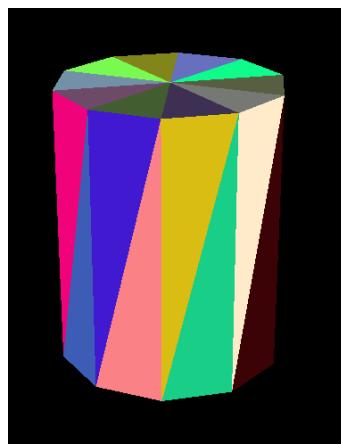


Figura 1: Cilindro gerado com 10 slices

2.2 Vetores Normais e Coordenadas de Textura

2.2.1 Plano

Como um plano está situado no plano xOz e as normais são sempre vetores perpendiculares, as normais deste são vetores unitários com apenas a componente y positiva.

As coordenadas de textura para o plano são calculadas de forma muito simples. Como o plano e a textura tem a mesma forma, o mapeamento é direto, assim, os cantos da textura são mapeados para os cantos do plano.

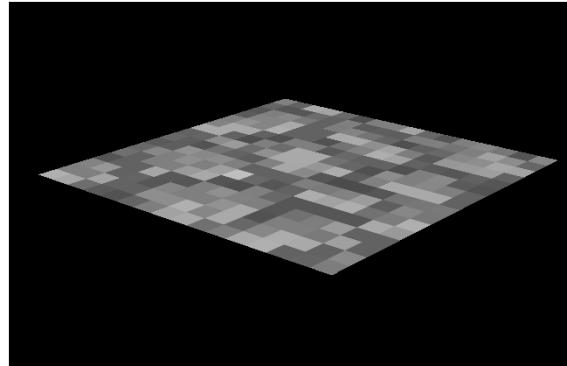


Figura 2: Plano com textura

2.2.2 Cubo

As normais de um cubo também são simples de calcular, pois cada uma das faces é paralela a um dos planos (xOz , xOy , yOz), logo as normais dessa face são perpendiculares ao plano correspondente.

Para as texturas começou-se por definir uma fórmula que mapeasse as coordenadas de cada face para valores compreendidos entre, 0 e $\frac{1}{3}$ e 0 e $\frac{1}{2}$, dependendo do eixo pretendido. Depois é somado um *offset* a estes valores dependendo da secção da textura que se pretende mapear.

Assumindo que o ponto de vista é colocado em $(0,0,1)$, as coordenadas de textura da face da frente são $x \in [0, \frac{1}{3}]$ e $y \in [0, \frac{1}{2}]$ logo o *offset* destes é nulo. No entanto, para a face da esquerda, as suas coordenadas de textura são $x \in [\frac{2}{3}, 1]$ e $y \in [\frac{1}{2}, 1]$, logo o *offset* destes é $\frac{2}{3}$ e $\frac{1}{2}$, respetivamente.

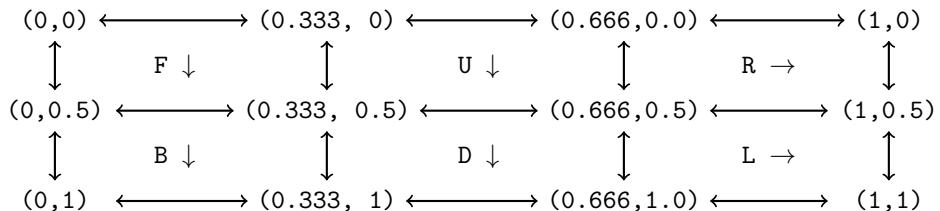
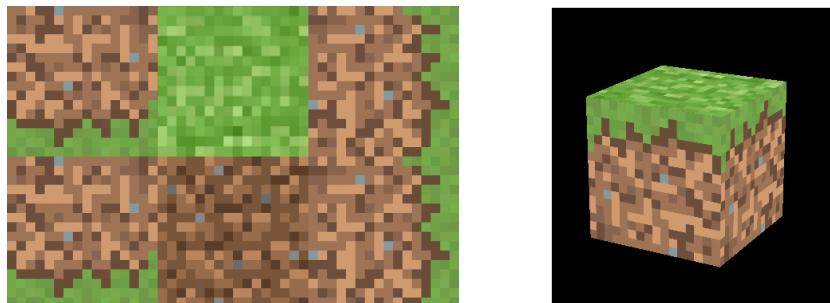


Figura 3: Esquema de como a textura deve ser desenhada.



(a) Exemplo de uma textura para o cubo.

(b) Cubo com textura.

Figura 4: Cubo.

2.2.3 Esfera

Sendo que a esfera está sempre centrada na origem, para calcular as normais desta, é preciso apenas normalizar as coordenadas dos pontos de forma a obter as normais.

A textura usada para aplicar à esfera é retangular, assim, o raciocínio utilizado para aplicar a textura é imaginar que a esfera é cortada em altura e “espalmada” sobre a textura, de seguida é apenas necessário mapear diretamente a textura sobre estar interpretação da esfera.

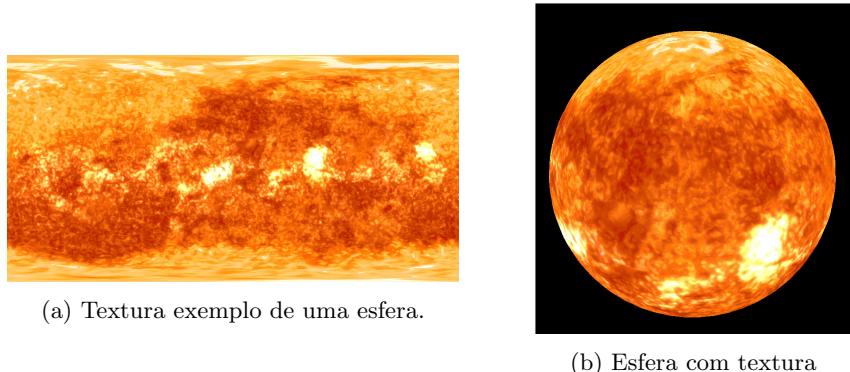


Figura 5: Esfera

2.2.4 Cone

O cálculo das normais do cone é feito em duas partes, primeiro são calculadas as normais da base, que por definição são o vetor unitário $(0, -1, 0)$. A segunda parte corresponde ao lado do cone, para o qual é calculado o ângulo entre qualquer ponto do lado do cone e a base do mesmo através da $\arctan(\frac{radius}{stacks})$, este ângulo define a inclinação das normais laterais. Depois disto, basta utilizar coordenadas polares para determinar as componentes x e z do vetor e normalizar.

Para aplicar uma textura a um cone foi definido o formato da figura ??, ou seja, a metade esquerda da imagem é a base do cone, e a metade direita é a textura do lado do cone. Depois a textura é dividida no mesmo número de *slices* e *stacks* que o cone tem e faz-se o mapeamento.

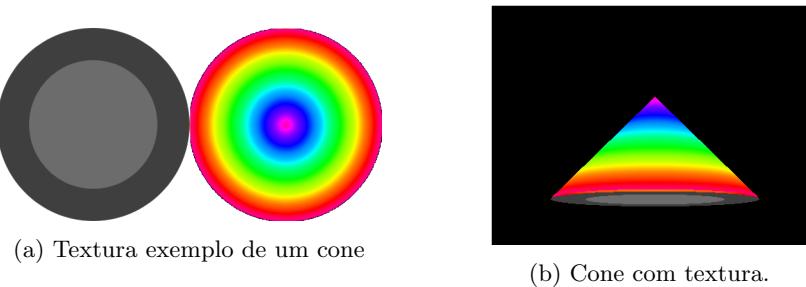


Figura 6: Cone.

2.2.5 Cilindro

As normais dos cilindro são relativamente simples, para a base e topo as normais são sempre $(0, -1, 0)$ e $(0, 1, 0)$ respetivamente, para o lado, é apenas necessário normalizar as coordenadas dos pontos respetivos depois de colocar o seu y a 0 (para que seja paralelo ao plano xOz).

Para as coordenadas de textura é foi utilizado o *template* da figura ???. Para os lados divide-se o retângulo pelo número de *slices* e sempre que se avança de *slice* soma-se essa quantidade ao x da textura. O y so toma dois valores 0 e 0.625. Por fim, o topo e a base são obtidas através de

coordenadas polares, somando ao centro de cada uma $\sin(\alpha) \times 0.1875$ para o x e $\cos(\alpha) \times 0.1875$ para o y .



Figura 7: Texturas do cilindro.

2.2.6 Torus

As normais do torus fazem uso da mesma técnica utilizada para a esfera, (Ver ??) apenas alterado o centro, agora a origem já não é o ponto de referência mas sim os centros de cada anel do torus, visto que o raciocínio para o cálculo dos pontos deste já era vetorial, foi apenas necessário normalizar os vetores.

Para as coordenadas de textura, o mapeamento é também similar ao da esfera, imagina-se que a *mesh* do torus é espalmada sobre a textura deste, e assim o mapeamento é direto. Percorrer a textura ao longo do eixo dos y implica percorrer o torus à volta do seu perímetro enquanto que no eixo dos x desenha-se cada anel do torus.

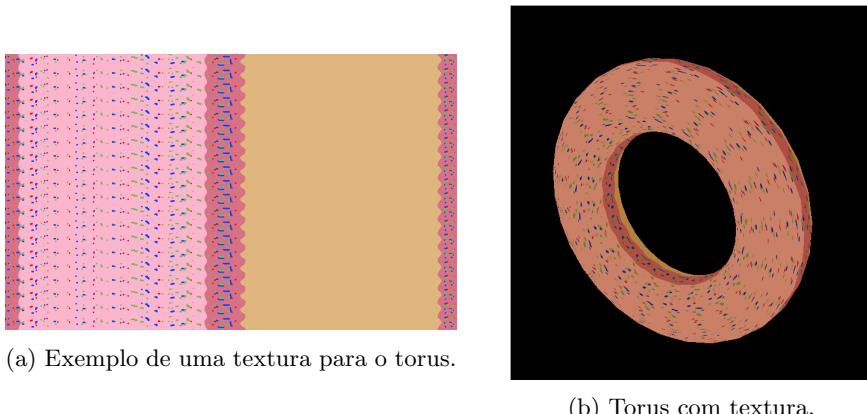


Figura 8: Torus.

2.2.7 Bezier Patches

Para calcular as normais do *Bezier patch* usa se o mesmo algoritmo de calculo usado para os pontos, mas em vez de calcular um ponto calcula-se dois vetores tangentes a este e com o produto

escalar encontra-se o vetor normal ao ponto.

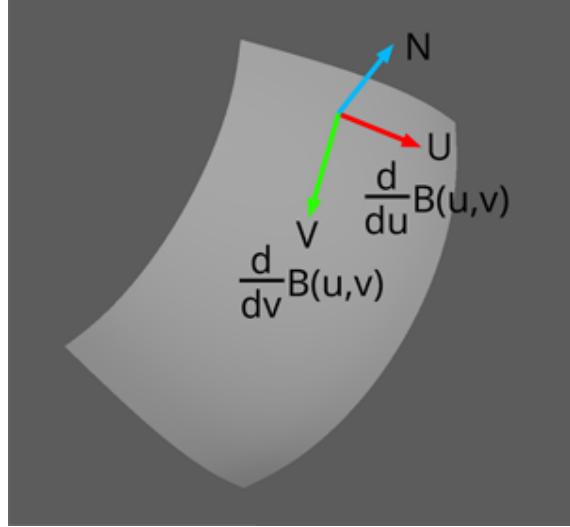
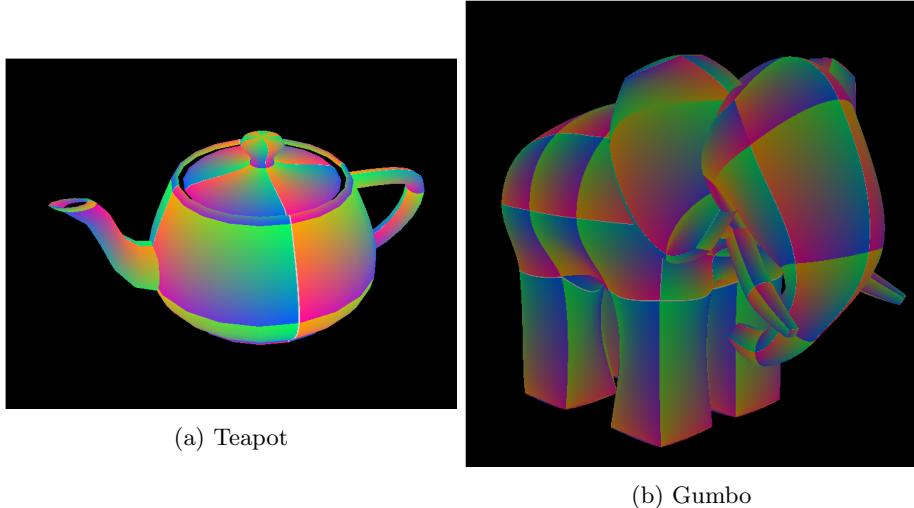


Figura 9: Calculo da normal através dos vetores tangentes. (Source [?])

Para as coordenadas de textura, simplesmente mapeia-se a textura toda para cada superfície de *Bézier*. Ficando assim repetida tantas vezes quantos os patches do modelo.



(a) Teapot

(b) Gumbo

Figura 10: Exemplos de modelos gerados por patches.

3 Engine

O *engine* foi estendido para possibilitar a renderização de modelos com superfícies mais complexas, permitindo definir se um objeto tem uma superfície difusa, especular, emissiva e/ou ambiente, assim como definir uma textura para a cobrir.

3.1 Sem texturas

A entrada de um *model* no XML pode agora incluir as componentes acima referidas através das propriedades `ambi[RGB]`, `diff[RGB]`, `spec[RGB]` e `emis[RGB]`. Por exemplo,

```
<model file="models/sphere.3d" ambiR="0.2" diffB="0.8" />
```

Todas as componentes têm valores por defeito de $(0, 0, 0)$ exceto a componente difusa que tem $(1, 1, 1)$ por defeito. Num esforço de manter *backwards compatibility* e cenas de fases anteriores continuarem a funcionar, quando nenhuma componente é especificada a cor do grupo é passada como valor da componente emissiva dos seus *models*.

3.2 Com texturas

Para além das componentes, pode ser passado também o ficheiro de onde carregar a textura do modelo utilizando o atributo **texture**. Por exemplo

```
<model file="models/sphere.3d" texture="assets/8k_jupiter.jpg" />
```

4 Lighting

Este programa oferece 3 tipos de luzes, pontual, direcional e *spotlight*. Estas são afetadas por transformações geométricas como qualquer outro objeto e podem ser definidas tanto ao nível de cena (**<scene>**) como ao nível de um qualquer grupo, tendo em atenção que apenas 8 luzes podem ser carregadas pelo programa.

Cada uma com os seus parâmetros específicos que serão delineados nas subsecções seguintes.

4.1 Point Light

Para definir uma luz pontual tem de ser passado o atributo **type** com o valor **POINT**. Os parâmetros admitidos pela luz pontual são:

- **Posição:** Através dos atributos **posX**, **posY**, e **posZ**.
- **Cor:** Através dos atributos **R**, **G** e **B**.

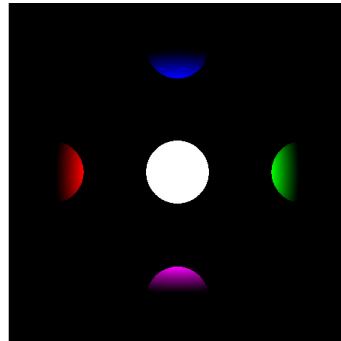


Figura 11: Demonstração da luz pontual, uma fonte de luz no centro (marcada com uma esfera emissiva) rodeada de quatro esferas de cores diferentes.

4.2 Directional Light

Para definir uma luz direcional tem de ser passado o atributo **type** com o valor **DIRECTIONAL**. Os parâmetros admitidos pela luz pontual são:

- **Direção:** Através dos atributos **dirX**, **dirY** e **dirZ**.
- **Cor:** Através dos atributos **R**, **G** e **B**.

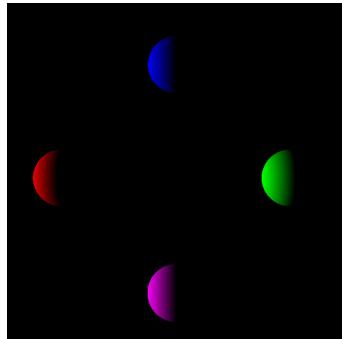


Figura 12: Demonstração da luz direcional, uma fonte de luz em direção a x positivo rodeada de quatro esferas de cores diferentes.

4.3 Spotlight

Para definir uma luz *spotlight* tem de ser passado o atributo `type` com o valor SPOT. Os parâmetros admitidos pela luz pontual são:

- **Posição:** Através dos atributos `posX`, `posY`, e `posZ`.
- **Direcção:** Através dos atributos `dirX`, `dirY` e `dirZ`.
- **Cor:** Através dos atributos `R`, `G` e `B`.

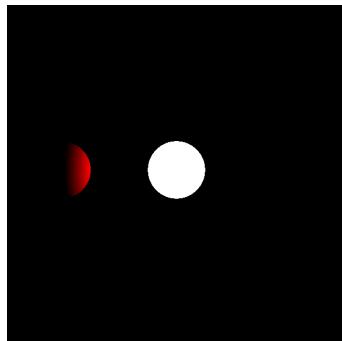


Figura 13: Demonstração da *spotlight*, uma fonte de luz no centro (marcada com uma esfera emissiva) rodeada de quatro esferas de cores diferentes.

5 Sistema Solar

O sistema solar foi alterado para aplicar uma textura a cada astro e uma luz pontual no centro do Sol. Alguns detalhes a notar são o cuidado em colocar sempre a mesma face da lua virada para a Terra e ainda a um habitante em Marte.

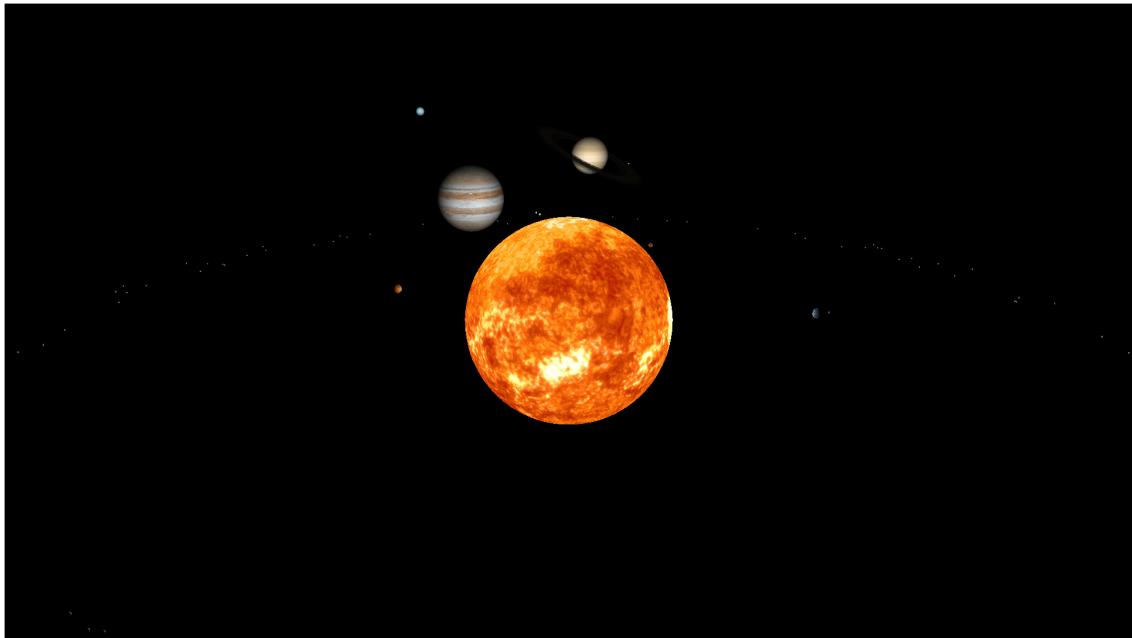


Figura 14: O sistema solar final.

6 Optimizações

Para otimizar a utilização de memória do programa, foi criada uma *cache* de modelos e texturas, assim cada conjunto de pontos, normais e coordenadas de textura, correspondente a um ficheiro .3d, é carregado apenas uma vez para o GPU e reutilizado para todos os modelos que os utilizam. O mesmo raciocínio foi feito para as texturas, para que não fosse lido para memória mais do que uma cópia de cada textura.

Para o caso do sistema solar, como todos os planetas utilizam o mesmo modelo unitário da esfera, independentemente de quantos astros sejam colocados no modelo, o ficheiro `sphere.3d` é lido apenas uma vez.

7 Conclusões e Trabalho Futuro

Em conclusão, a área de computação gráfica apresenta desafios bastante diferentes das outras áreas de ciências de computação, tanto no que toca a gestão de memória e de recursos de computação, como também nos conhecimentos de geometria e álgebra.

Como trabalho futuro, poderiam ser feitas várias otimizações como oclusão e a utilização de índices para melhorar o desempenho dos VBOs.