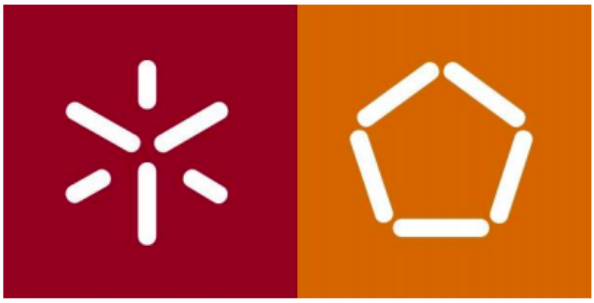

Universidade do Minho

Mestrado Integrado em Engenharia Informática

29/05/2020

Computação Gráfica
Fase 4

António Gonçalves (A85516)
João Fernandes (A84034)
Eduardo Conceição (A83870)
Rita Rosendo (A84475)

Contents

1 Introdução

Este relatório irá descrever a quarta e última fase do projeto da cadeira de Computação Gráfica. Nesta fase do trabalho, foi-nos proposto que, a partir da segunda fase, construíssemos um programa que conseguisse ler e efetuar vários tipos de iluminação diferentes, para além de permitir atribuir texturas às diversas figuras.

2 Correções

Uma vez que na entrega anterior não conseguimos executar o movimento de translação através da utilização de curvas Catmull, achamos importante resolver este problema antes de avançar para a 4ª fase.

Pouco depois da entrega da terceira fase, apercebemo-nos de que alguns dos valores utilizados para o calculo da função getCatmullRomPoint estavam errados, por exemplo, na atribuição dos valores da matriz A. Para além disso, ao incializarmos uma nova Translação, esta ficava com o valor *t* a 0, o que fazia com que o calculo de alguns valores fosse uma divisão por 0, o que é impossível. Assim passamos a inicializar este valor com -1, sendo este valor alterado na altura de fazer o *parsing*.

Concluindo, passamos a certificar que existia uma *tag* "translacao" antes de iniciar o *parse* da sua informação. Para isso, simplesmente verificamos se o *FirstChildElement* do xml atual tem o texto "translacao".

Para conseguirmos os movimentos dos satélites em torno dos seus planetas foi utilizada a mesma estratégia dos planetas em torno do Sol. Neste caso, a distância entre o centro do planeta principal e o centro do satélite será o raio da circunferência que representa a sua órbita. Assim, a partir do planeta principal, calculamos 8 pontos à distância do raio para definir a orbita.

Para finalizar a parte 3 do projeto faltava também o movimento do bule ao longo da curva, sendo ainda preciso definir os pontos da sua trajetória. Estes foram criados aleatoriamente.

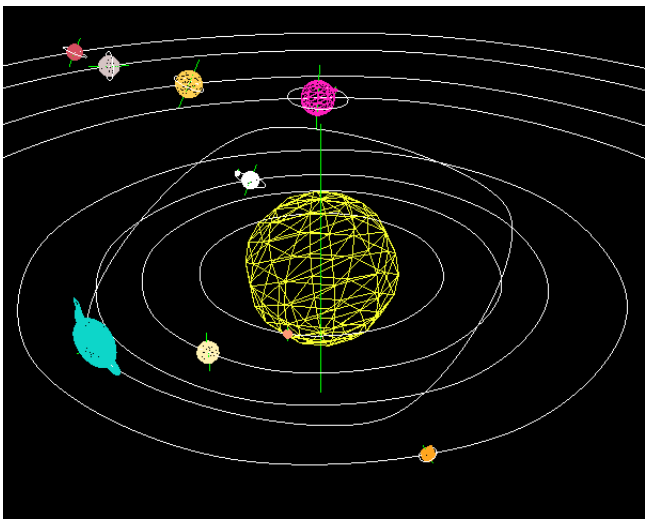

Figure 1: Resultado pretendido para a fase 3 do Projeto

3 Exposição do Problema

3.1 *Generator*

Para que se possa utilizar texturas e iluminção, temos antes de, para cada modelo criado pelo gerador, calcular as normais e as coordenadas da textura.

3.2 *Engine*

Após ler as coordenadas de textura e as normais dos pontos, estas terão de ser processadas, de forma a poderem ser utilizadas para definir as texturas e as luzes. Para além disso, teremos de fazer o *load* das texturas.

4 Resolução

4.1 *Generator*

Nesta secção iremos descrever como calculamos as coordenadas das texturas e das normais para cada um dos objetos suportados pelo *Generator*.

4.1.1 Plane

- **Normais**: A normal de um plano é básica. Uma vez que este é desenhado perpendicular ao eixo YY, então as coordenadas das suas normais são apenas (0,1,0) para a face superior e (0,-1,0) para a face inferior.
- **Texturas**: Como apenas existem 4 pontos a ter em conta no plano (os seus vértices), então as coordenadas são igualmente básicas.

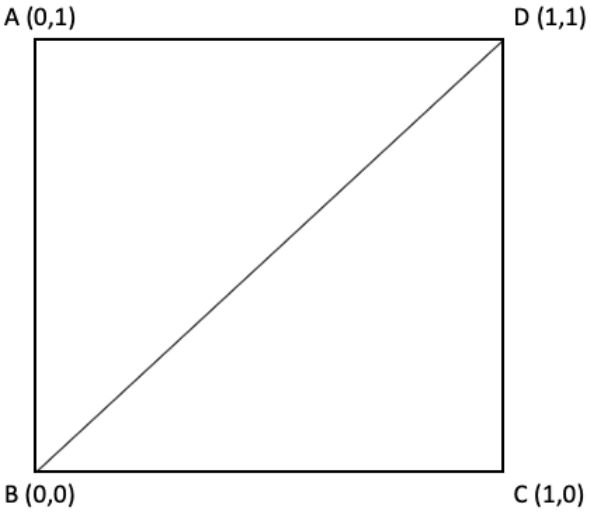

Figure 2: Coordenadas das texturas para o Plano

4.1.2 Box

- **Normais**: A normal vai variar de face para face. Assim sendo, vai ser equivalente ao vetor unitário do eixo perpendicular à face, com sentido para onde a face estiver virada. Ou seja, por exemplo, a face da frente da caixa (que estaria virada para nós se estivermos no eixo do ZZ) terá normal (0,0,-1), e o seu interior tem normal (0,0,1), ou seja, no sentido oposto. Isto também se aplica às faces das divisões.

- **Texturas**: As texturas funcionam da mesma forma que funcionam no plano, aplicando os cálculos a cada uma das faces. Temos sempre que ter em consideração a orientação do vetor normal, ou seja, temos de estar "de frente" ao plano da face para calcularmos as coordenadas como fazíamos com o plano.

4.1.3 Cone

O cone está dividido em duas parte: a base e os lados. Como tal, iremos dividir a explicação do mesmo em duas.

Base:

- **Normais**: A normal da base cone, uma vez que este está virado para baixo e é perpendicular ao eixo YY, é em todos os pontos da base (0,-1,0).
- **Texturas**: A textura que aplicamos à base do cone (e também aos seus lados) é circular. Como tal, tendo em conta que o círculo que constitui a textura se encontra centrado na imagem em (0.5,0.5), realizamos os seguintes cálculos para obter as coordenadas:

```
float tx = 0.5 * cos(a) + 0.5;
float tz = 0.5 * sin(a) + 0.5;
```

  Estes advém da simples aplicação de coordenadas circulares, num círculo de raio 0.5 e centro em (0.5,0.5), com o ângulo a tendo sido explicado na parte 1 deste projeto.

Lados:

- **Normais**: As normais do lado do cone não são tão triviais de calcular como eram nos outros casos. Nos lados, os componentes X e Y do vetor normal serão iguais aos valores das componentes X e Y do ponto em que aplicamos o vetor a dividir pela distância que estes têm com o eixo YY. No entanto, o componente Y é diferente.

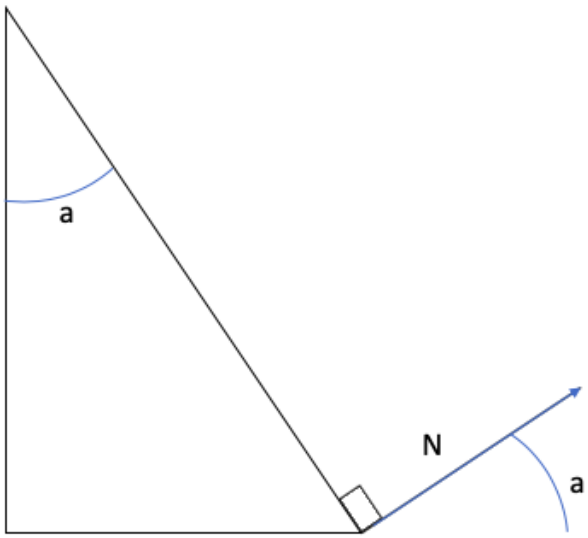

Figure 3: Perfil do cone

Nesta figura, podemos ver que o ângulo que o vetor faz com o plano XZ é, para qualquer ponto na lateral do cone, igual ao ângulo que o plano tangente à lateral do cone faz com o eixo YY, e podemos obtê-lo com a função inversa da tangente, utilizando a altura como cateto adjacente e o raio da base como cateto oposto. Tendo este ângulo, podemos obter a componente Y da normal aplicando o seno.

```
nx = sin(a);
ny = sin( atan( (float)radius / (float)height ));
nz = cos(a);
```

- **Texturas**: O cálculo das texturas funciona de forma semelhante à base. No entanto, temos de ter em conta a *stack* e a *slice* em que estamos. Para isto, aplicamos os seguintes cálculos:

```
float tx1 = ((float)i/(float)stacks) * cos(a) + 0.5;
float tz1 = ((float)i/(float)stacks) * sin(a) + 0.5;
```

4.1.4 Sphere

- **Normais**: Para calcular as normais, seguimos um cálculo bastante simples, pois, para um ponto na superfície da esfera, o seu vetor normal é igual às suas coordendadas, normalizadas, ou seja:

```
nx = cos(beta)*sin(alpha);
ny = sin(beta);
nz = cos(beta)*cos(alpha);
```

  De notar que alteramos os nomes dos ângulos que tínhamos em fases anteriores para melhor refletir o nome utilzado por padrão. Ao compararmos os valores com os das coordenadas, podemos ver que os obtemos simplesmente dividindo cada componente pelo raio da esfera.

- **Texturas**: Para obter as coordenadas das texturas, que seriam geradas a partir de uma imagem retangular, aplicamos os seguintes cálculos:

```
texX=(a/slices);
texZ=(h/stacks);
```

  Neste caso, *a* corresponde à variável que é utilizada para percorrer a esfera na horizontal, e *h* uma variável auxiliar que é incrementada ao percorrer a esfera na vertical. Posto isto, cada componente das coordenadas das texturas correspondem a um passo ao longo das *stacks* e das *slices*.

4.1.5 Bezier Patch

- *Normais*: Para calcular a normal de um ponto numa malha de Bezier, teremos de utilizar a seguinte fórmula:

```
N = dU x dV
```

  Em que *dU* e *dV* correspondem, respetivamente, às derivadas parciais das componentes *u* e *v* da *patch*, no ponto em questão, sendo que estas são calculadas da seguinte forma:

$$dU = [3u^2 2u 10] * M * P\_matriz * M^T * [v^3 v^2 v 1]^T$$
$$dV = [u^3 u^2 u 1] * M * P\_matriz * M^T * [3v^2 2v 10]^T$$

  Sendo *P* a matriz com os pontos de controlo a ter em consideração e *M* a matriz:

$$\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

  Assim, tendo estes dois vetores, o seu produto externo normalizado corresponde ao vetor normal no ponto em questão.

- **Texturas**: As coordendas das texturas aqui são calculadas com base no *u* e no *v*. Uma vez que estes valores variam entre 0 e 1, como os valores das coordenadas de texturas, podemos utilizá-los como coordenadas para as texturas, sendo que:

```
tx = u;
ty = v;
```

4.1.6 Ficheiro

Para esta fase, o ficheiro normal que o *Generator* criava não é alterado. No entanto, para além deste ficheiro, é criado um auxiliar na mesma diretoria, que terá o mesmo nome que o ficheiro original, com a extensão ".lat" (*Lights and Textures*). Cada linha deste ficheiro contém 5 números reais, sendo os primeiros três as componentes de um vetor normal e os outros dois as coordenadas das texturas, para o ponto da linha correspondente no ficheiro das coordenadas. Isto significa que, por exemplo, na linha 10 do ficheiro "sphere.3d" estão as coordenadas de um ponto e na linha 10 do ficheiro "sphere.3d.lat" encontramos a respetiva normal e coordenadas da textura.

4.2 Engine

Nesta secção trataremos de descrever o processo para desenvolver o pedido para esta fase no que toca à *Engine*.

4.2.1 Iluminação

Apesar de não termos consegui implementar a iluminação no trabalho final, iremos explicar qual a nossa estratégia para o fazer.

Nesta fase do projeto foi-nos pedido que conseguissemos implementar 3 tipos de iluminação diferentes: Direcional, Pontual e *Spot*. Para guardar as informações relacionadas com estes tipos iriamos criar uma estrutura, tendo esta as componentes necessárias para cada tipo, e um inteiro identificador do tipo. Assim, seria preciso guardar a posição da luz, definir as componentes *ambient* e *diffuse* e todas as restantes componentes necessárias para definir a fonte de luz.

Antes da leitura seria necessário inicializar esta estrutura, com valores genéricos para que mais tarde a falta de um valor lido não criasse problemas.

Um exemplo de uma póssivel leitura do ficheiro *xml* seria:

```
<Luzes>
    <Luz Tipo="PONTUAL" posX="20" posY="20" posZ="20"\>
<Luzes\>
```

A leitura desta informação seria feita da mesma forma que é efetuada para as restantes informações.

Na altura da geração das luzes seria usada a estrutura para que, em função do tipo de iluminação, se pudesse representar a luz pretendida.

Uma vez que não conseguimos completar esta ideia, acabamos por simplesmente inicializar uma *GL_LIGHT0* e os restantes comandos necessários para criar um ambiente onde é possível observar o efeito de fontes de luz, obtendo assim uma luz branca simples.

```
void renderScene(){

    float pos[4] = {1.0, 1.0, 1.0, 0.0};
    glClear( mask: GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(camX,camY,camZ,
              centerX,centerY,centerZ,
              upX,upY,upZ);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);

    float white[4] = { 1,1,1,1 };
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
```

Figure 4: Iluminação simples

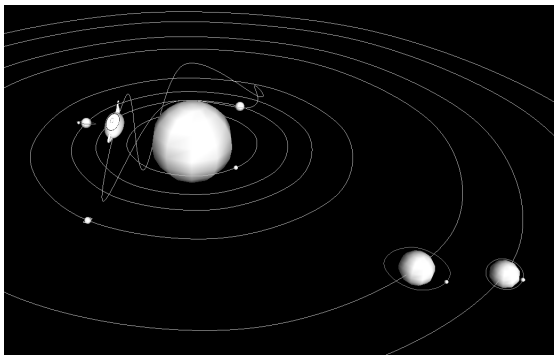

Figure 5: Resultado da Inicialização da Iluminação

4.3 Leitura

A leitura dos ficheiros de pontos não foi alterada. No entanto, foi adicionada a capacidade de ler os ficheiros com as normais, e estas são guardadas num *buffer* num *GLuint* na *struct Group*. O mesmo se aplica às coordenadas das texturas.

Para ler o ficheiro da textura em si, da imagem, digamos, adicionamos a função *loadTexture*, que foi retirada *verbatim* da nossa resposta ao Guião 11 das aulas práticas. Esta função recebe o nome do ficheiro e devolve um inteiro apontador para onde a textura foi *bound*.

4.4 Texturas

Uma vez que o processo descrito na secção de leitura conclui, os objetos e suas texturas são desenhadas pela função *draw*, e as texturas são chamadas uma a uma para serem uma vez mais *bound* quando o seu objeto correspondente é chamado. Para alé das texturas, são *bound* também os apontadores de *GLuint* para as coordenadas das normais e das texturas, juntamente com os pontos que já pasavam por este processo desde a fase anterior.

Os objetos que utilizamos para a demonstração com o Sistema Solar são apenas esferas ou um bule, que é feito com *patches* de Bezier. Para isto, disponibilizamos várias texturas das superfícies dos vários astros em questão, que são todas retangulares.

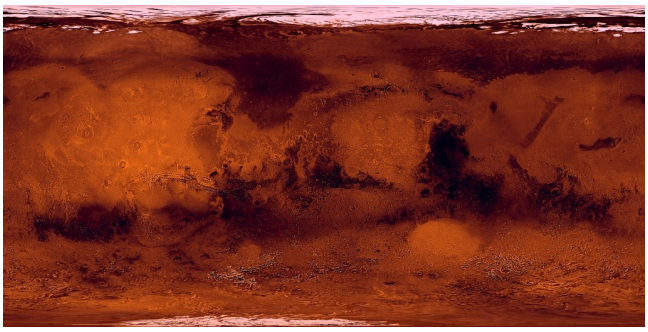

Figure 6: Textura de Marte

4.5 Ficheiro XML

Em relação à fase anterior, o ficheiro de *input* não foi muito alterado. Uma vez que concluímos a fase 3, a posição dos planetas passou a definir-se dentro da curva gerada, e assim não será necessário fazer *translate* dos mesmos para a sua posição inicial. Assim, removemos todas as translações do ficheiro, podendo, mesmo assim, efetuar translações como anteriormente. No caso de efetuar uma translação de um planeta, o que se verificaria seria uma translação de toda a "órbita", desviando assim o seu centro para o ponto de translação.

Para aproximar o Modelo de um mais real, decidimos alterar os valores do tempo que cada planeta demora a completar uma volta ao sol. Assim, por exemplo, a Terra, que demora aproximadamente 365 dias a dar a volta ao Sol, tem um valor de *tempo* no *XML* de 36. Seguimos esta lógica para todos os planetas, mas não para os satélites, sendo estes valores aleatórios.

Outra alteração foi que, uma vez que as figuras possuem texturas, retiramos as modificações das cores. Ao alterar a cor, a textura iria ficar com uma ligeira alteração, conforme a cor aplicada. Mais uma vez, simplesmente retirámos essa informação do ficheiro *XML*, sendo na mesma possível alterar a cor dos objetos, caso o ficheiro de *input* tenha indicação para tal. Para finalizar, acrescentámos à linha onde tinhamos o nome do ficheiro o nome da textura a utilizar, encontrando-se estas numa pasta chamada Texturas.

4.5.1 Funcionalidades Adicionais

Uma vez que o movimento dos planetas mais distantes é pouco perceptível, decidimos criar 2 inteiros, *rot* e *trans*, que irão alterar a velocidade das animações de rotação e de translação dos mesmos. Estas mudam de valor com as Teclas 'Z' e 'X', para diminuir e aumentar, respetivamente.

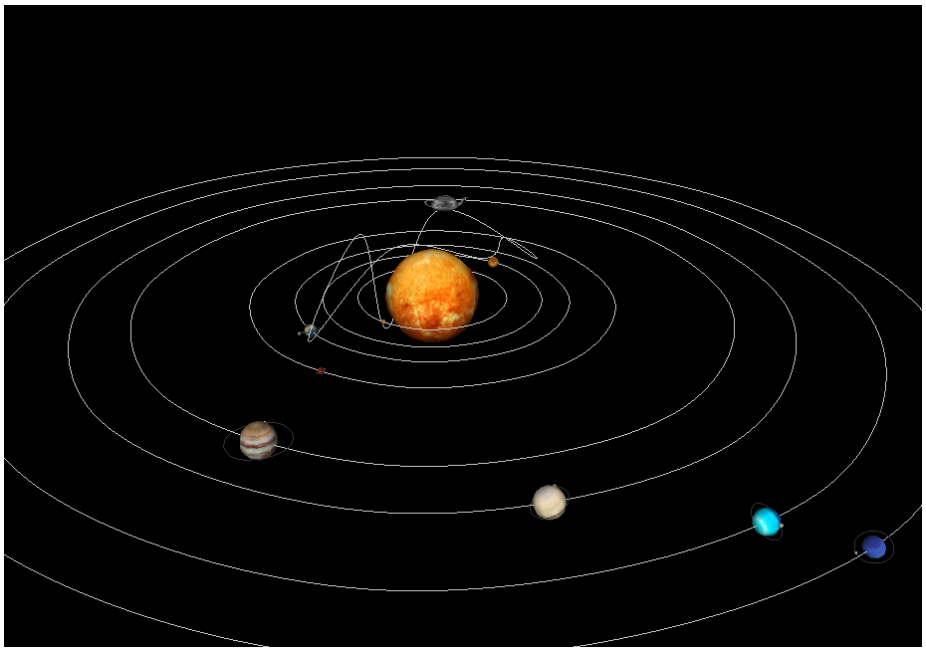

Figure 7: Resultado Final

5 Conclusão

Apesar de não termos feito a parte da iluminação, achamos que o resto do projeto esta correto. Conseguimos implementar a utilização de texturas, bem como a criação e leitura dos vetores normais e coordenadas de textura necessárias para tal.

Perdemos um pouco de tempo na correção dos erros da parte 3, o que afetou o nosso resultado final, sendo os erros que nos fizeram entregar essa fase incompleta erros bastante simples, mas que na altura não conseguimos detetar. Contudo, após os resolver achamos que conseguimos perceber melhor essa mesma parte da matéria.

Conseguimos ainda ultrapassar um erro que estava a tornar o modelo quase inutilizável, fazendo com que a leitura das texturas terminasse, algo que não acontecia na nossa primeira resolução, aumentando assim a *performance* do modelo. Concluindo, pensamos que o nosso trabalho poderia ter sido mais desenvolvido, principalmente no que toca à iluminação e interface com o utilizador, mas que mesmo assim conseguimos alcançar uma boa parte do objetivo proposto.