Computação Gráfica (3º ano de MIEI)

Quarta Fase

Relatório de Desenvolvimento

André Gonçalves (a80368)

João Queirós (a82422)

Luís Alves (a80165)

Rafaela Rodrigues (a80516)

15 de Maio de 2019

Resumo

Este relatório inicia-se com uma breve contextualização, seguindo-se a descrição dos problemas propostos e o que deve ser desenvolvido para os solucionar. De seguida são descritas as alterações efetuadas às duas aplicações previamente desenvolvidas: o gerador e o motor gráfico. Para o gerador é indicado o método de obtenção de normais e coordenadas de textura de cada primitiva, e para o motor gráfico são indicadas as alterações realizadas para suportar iluminação e texturas. Por fim, é apresentado a nova cena do Sistema Solar, terminando este relatório com uma conclusão sobre o trabalho desenvolvido, apontando os seus pontos fortes e fracos, bem como dificuldades sentidas.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito da Quarta Fase do Trabalho Prático da Unidade Curricular de Computação Gráfica, que se insere no 2º semestre do 3º ano do primeiro ciclo de estudos do Mestrado Integrado em Engenharia Informática.

1.2 Objetivos e Trabalho Proposto

Pretende-se com este relatório formalizar toda a análise e modelação envolvida na construção da solução ao problema proposto, incluindo estruturas de dados e algoritmos utilizados.

Nesta quarta fase foi proposta a geração de modelos com coordenadas de textura e normais para vértice, sendo que o motor gráfico deverá ser capaz de ler e interpretar essas coordenadas, ativando assim as funcionalidades de iluminação e textura.

Para além disso, também deverá ser possível definir cores e fontes de luz no ficheiro de configuração da cena.

1.3 Resumo do trabalho a desenvolver

Uma vez que todos os modelos deverão passar a ter texturas e normais, todas as primitivas terão de ser alteradas para se poder gerar as respetivas coordenadas. No entanto, será apenas necessário calcular o que vai ser adicionado, mantendo-se as soluções anteriores.

Relativamente ao motor gráfico, este deverá ser capaz de armazenar estas novas informações provenientes do ficheiro .3d, bem como carregar imagens do ficheiro XML e ser capaz de interpretar cores e luzes desse mesmo ficheiro, ativando as respetivas funcionalidades no OpenGL.

Capítulo 2

Arquitetura da Solução

Esta fase pode ser vista como tendo duas componentes de desenvolvimento distintas: a geração das normais e texturas dos modelos e a alteração do motor gráfico para suportar texturas e iluminação. As alterações feitas são reportadas de seguida.

2.1 Gerador

Tendo em conta que agora são inseridas mais linhas por modelo, será necessário alterar a estrutura do ficheiro .3d para ser possível ler as novas informações. Por isso, primeiramente surgem os vértices, com cada componente (X, Y e Z) separada por um espaço e cada vértice separado por uma quebra de linha. De seguida surgem os índices, sendo que cada índice é separado por uma quebra de linha. Findos os índices, surgem as normais de cada vértice, com cada componente (X, Y e Z) também separada por um espaço e cada normal separada por uma quebra de linha. Por fim, surgem as coordenadas de textura, com cada componente (X e Y) separada por um espaço e cada par separado por uma quebra de linha.

2.1.1 Primitivas da 1ª Fase

Para cada primitiva da 1ª fase, foram adicionadas as normais de cada vértice gerado e as coordenadas de textura. Isso não implicou nenhuma alteração à sequência de vértices/índices gerada anteriormente, excetuando o cone. As alterações feitas são as que se apresentam de seguida.

Plano

As normais do plano são obtidas de forma muito simples, uma vez que são iguais para os 4 vértices. São elas `(0, 1, 0)`.
Relativamente às coordenadas de textura, para cada extremidade do plano, corresponde uma extremidade da textura a aplicar.

Caixa

Relativamente às normais da caixa, para cada face as normais de todos os vértices são iguais, sendo elas:

- (0,1,0) para a face de cima
- (0,-1,0) para a face de baixo

- (-1,0,0) para a face da esquerda
- (1,0,0) para a face da direita
- (0,0,1) para a face da frente
- (0,0,-1) para a face de trás

Quanto às texturas, esta será replicada igualmente por todas as faces, sendo que par cada face, a cada vértice (i,j) corresponde a coordenada (i/divisões) na textura.

Cone

Relativamente ao cone, tivemos que refazer a forma de de listar os vértices, sendo que primeiramente são calculados todos os vértices da base, e por fim todos os vértices da superfície lateral. Assim, é possível definir as normais para as bordas da superfície lateral, que têm vértices comuns à base do cone. Repetindo-os, já não existe esse problema.
A normal de todos os vértices da base do cone é (0,-1,0), sendo que a cada vértice da base (centro,slice) é associada a coordenada da textura (i / slices, centro), sendo que caso seja o centro a coordenada y é 1, e caso não seja é 0.
Relativamente à superfície lateral, as componentes x e z do vetor normal são iguais aos valores de x e z dos vértices onde são calculadas, mas normalizadas, sendo possível obtê-las através das seguintes equações:

$$x = cos(slice * 2\pi/slices) \tag{2.1}$$

$$z = sin(slice * 2\pi/slices) \tag{2.2}$$

Relativamente à componente y, recorremos a um esquema visível na figura 2.1. Assim, é possível obter a componente y através das seguintes equações:

$$\frac{h}{r} = tan(\alpha) \tag{2.3}$$

$$\alpha = atan(\frac{h}{r}) \tag{2.4}$$

$$\beta = \pi - \frac{\pi}{2} - \alpha \tag{2.5}$$

$$y = sin(\beta) \tag{2.6}$$

Quanto às coordenadas de textura, estas têm para o vértice (i,j) valor (i / slices, j / stacks).

Esfera

Relativamente à esfera, as normais a cada vértice são facilmente obtidas através da normalização dos componentes x, y e z de cada vértice, isto é, dividindo o seu valor pelo raio.
Quanto às coordenadas de textura, para cada vértice (i,j), a sua textura será ((slices - j) / slices, (stacks - i) / stacks).

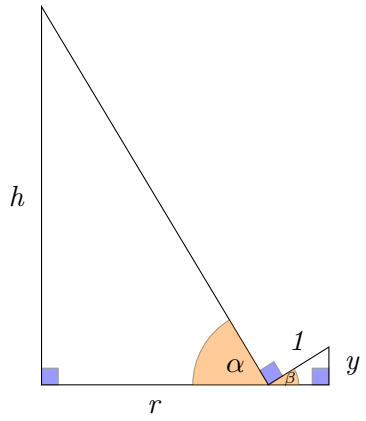

Figura 2.1: Cálculo da componente y da normal da superfície lateral do cone

Patches de Bezier

Para calcular as normais de um dado ponto (u,v) de uma superfície de Bezier, recorremos às seguintes equações

$$\frac{\partial B(u,v)}{\partial u} = [3u^2, 2u, 1, 0]MPM^TV^T \tag{2.7}$$

$$\frac{\partial B(u,v)}{\partial v} = UMPM^T[3v^2, 2v, 1, 0]^T \tag{2.8}$$

Tendo as derivadas parciais, o vetor normal a qualquer ponto na superfície é o resultado normalizado do produto externo dos vetores tangentes ao ponto (sendo eles o resultado das duas equações acima).

Relativamente às texturas, a um ponto (u,v) corresponde a textura (u,v), uma vez que u e v variam entre 0 e 1, tal como a textura.

2.2 Motor gráfico

Em relação ao motor gráfico, foi necessário alterá-lo de forma a poder interpretar as novas funcionalidades previstas para o ficheiro de configuração XML. As alterações foram aplicadas ao *parser*, ao modelo e a duas novas classes: *light* e *color*. Nas secções seguintes são abordadas em mais detalhe.

2.2.1 *Light*

De acordo com a documentação do *OpenGL*, uma luz é definida por ter uma posição, uma componente ambiente, difusa e especular, podendo também ser emulada uma lanterna, através da definição de um expoente, direção e corte.

Assim, sempre que for adicionada uma nova luz no *parser*, esta terá os valores por defeito do *OpenGL*, à exceção daqueles que forem redefinidos no ficheiro .xml.

O vetor de posição tem 4 componentes, uma vez que a quarta componente, caso seja 0, indica que a luz é direcional, e os valores de x, y e z representam a direção. Caso não seja, então a luz é posicional, e os valores de x, y e z representam a posição da luz no plano de coordenadas homogéneo.

Para o caso de uma luz posicional, o seu raio de iluminação poderia ser limitado através da definição da direção e corte, sendo o corte o ângulo até ao qual a luz será emitida, simetricamente. Para além disso,

também é possível controlar a distribuição da quantidade de luz emitida pelo *spotlight* através da definição do expoente.
Uma luz também tem uma componente difusa, ambiente e especular, sendo que a primeira é representativa do que é tipicamente conhecido como "a cor da luz"; a segunda é representativa da quantidade de luz ambiente que a fonte de luz fornece à cena e a terceira afeta a cor especular dos objetos iluminados, sendo que para um efeito realista esta deverá ter parâmetros idênticos à componente difusa.

2.2.2 *Color*

Relativamente à cor, esta é composta por 4 componentes: difusa, especular, emissiva e ambiente, sendo que cada um destes componentes toma 4 valores, representativos de uma escala RGBA (Red-Green-Blue-Alpha).
Por defeito, cada nova cor toma os mesmos valores que um material criado para o *OpenGL*.

2.2.3 *Parser*

Foi necessário alterar o *parser* de forma a precaver a nova estrutura dos ficheiros .3d e das novas tags presentes no ficheiro de configuração xml. Assim, surgem duas novas funções, a `parseLights` e `parseColor`.
Assume-se que todas as fontes de luz presentes num ficheiro de configuração se encontram como filhos da *scene*, estando encapsuladas dentro da tag *lights*. Uma luz poderá ter os seguintes atributos:

1. type - representa o tipo da luz (poderá ser SPOT, POINT ou DIRECTIONAL)
2. posX/Y/Z - representa a posição da luz, ou direção se o tipo for DIRECTIONAL
3. ambRGB - representa o valor de luz ambiente para cada componente RGB
4. diffRGB - representa o valor de luz difusa para cada componente RGB
5. specRGB - representa o valor de luz especular para cada compoente RGB
6. dXYZ - caso seja do tipo SPOT, poderá ter estes atributos que definem a direção do *spotlight*
7. exp - caso seja do tipo SPOT, define o expoente associado à dispersão da luz pelo foco
8. cut - caso seja do tipo SPOT, define o ângulo para o qual emite luz

Caso o número de luzes seja superior a 8, o *parsing* falha e o motor não é executado.
Relativamente aos modelos, cada um poderá ter associado uma textura e componentes que definem o material e respetiva interação com as fontes de luz.
Os componentes são os seguintes:

1. diffRGB - representa o valor da luz difusa refletida pelo modelo
2. specRGB - representa o valor da luz especular, criando *highlights* no modelo
3. emRGB - representa a emissividade do modelo, simulando que o modelo emite luz.
4. ambRGB - representa o valor da luz ambiente refletida pelo modelo

Caso alguma ou nenhuma destas componentes estejam presentes num modelo, este assume os valores por defeito do *OpenGL*.

2.2.4 *Engine*

No motor em si, as alterações feitas foram mínimas. Aquando da inicialização dos VBOs de cada modelo passou a considerar-se as suas coordenadas de textura e normais, e caso existam texturas associadas a um modelo também são carregadas na fase de inicialização recorrendo ao DevIL.

Para além disso, também são ativadas todas as luzes carregadas no motor, bem como renderizadas aquando da chamada da função de *display* do *Glut*.

2.3 Sistema Solar

No que diz respeito ao sistema solar, as alterações relativamente à fase anterior prendem-se com a utilização de texturas para os planetas e satélites, bem como para o sol. Para além disso, também foi definida uma fonte de luz para o Sol, estando a sua emissividade definida para 1 nos campos RGB.

De seguida apresentam-se algumas capturas do sistema solar desenvolvido.

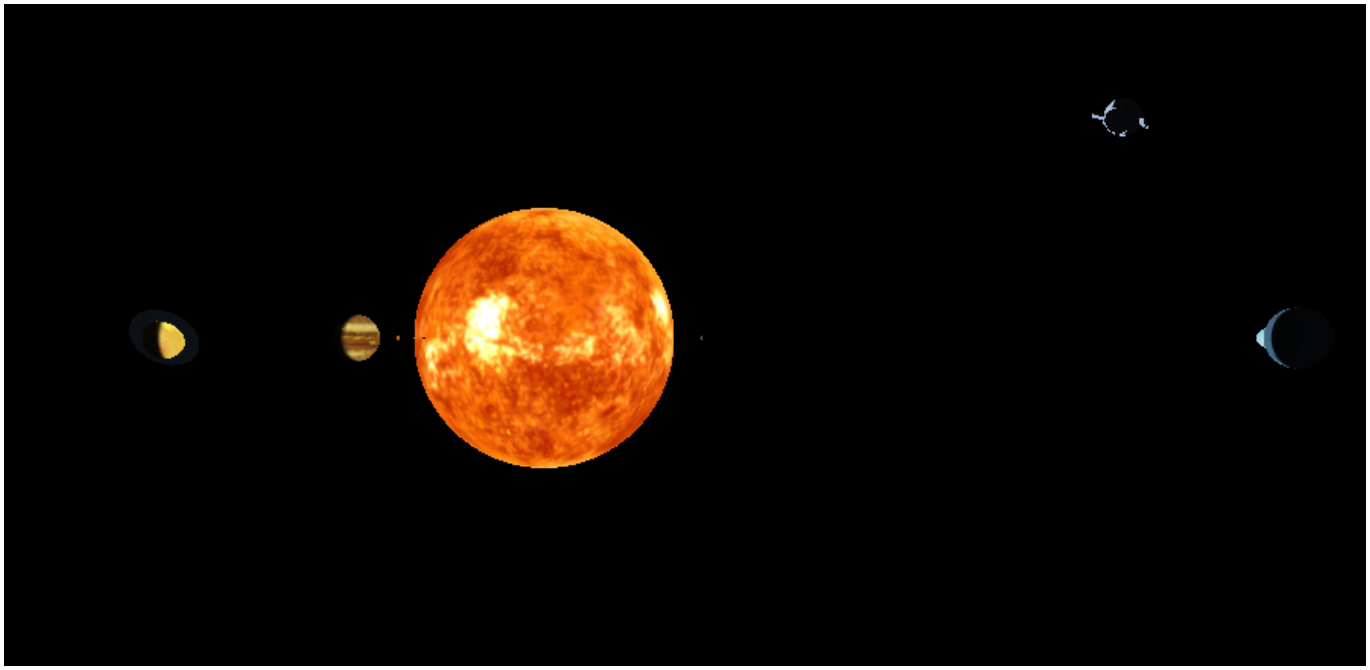

Figura 2.2: Sistema Solar

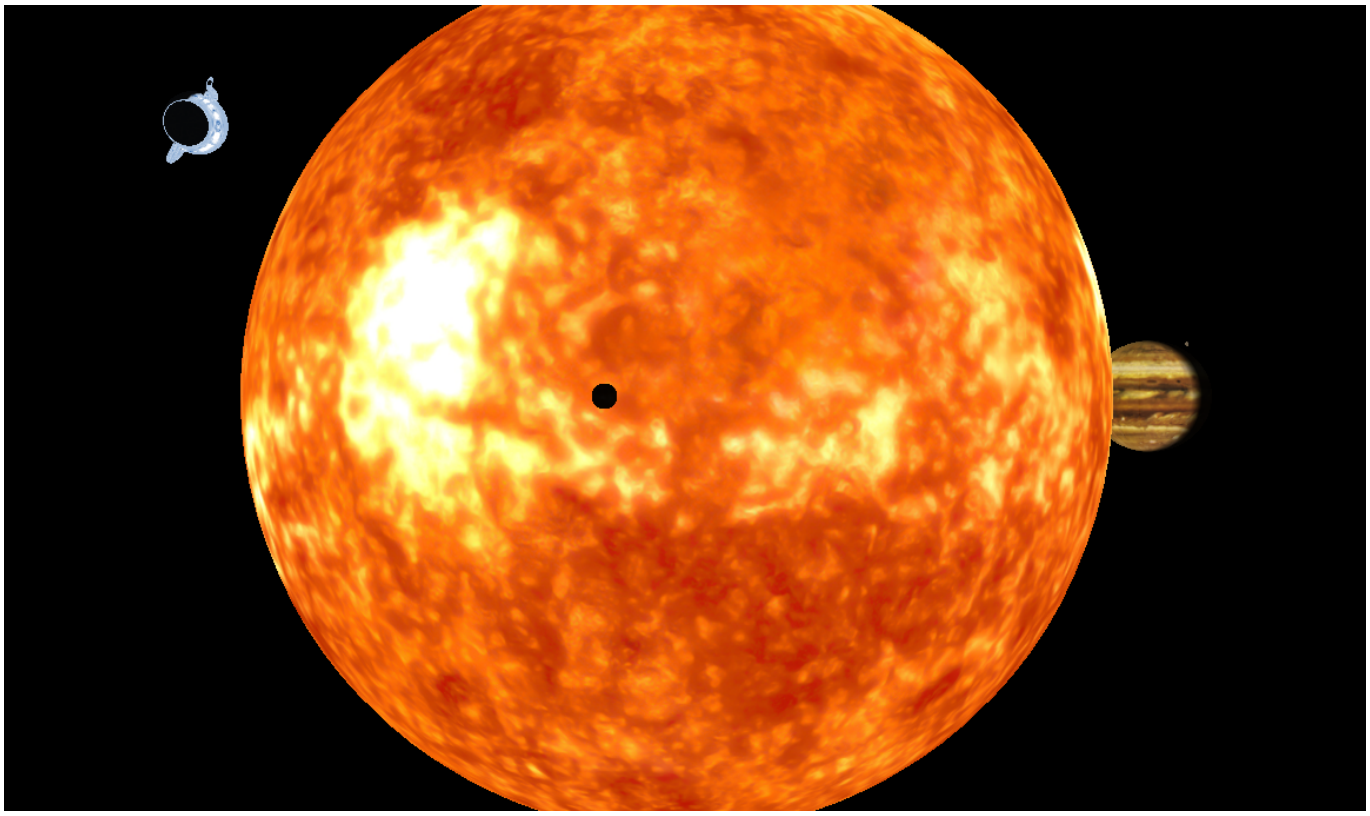

Figura 2.3: Sistema Solar com Teapot e Júpiter

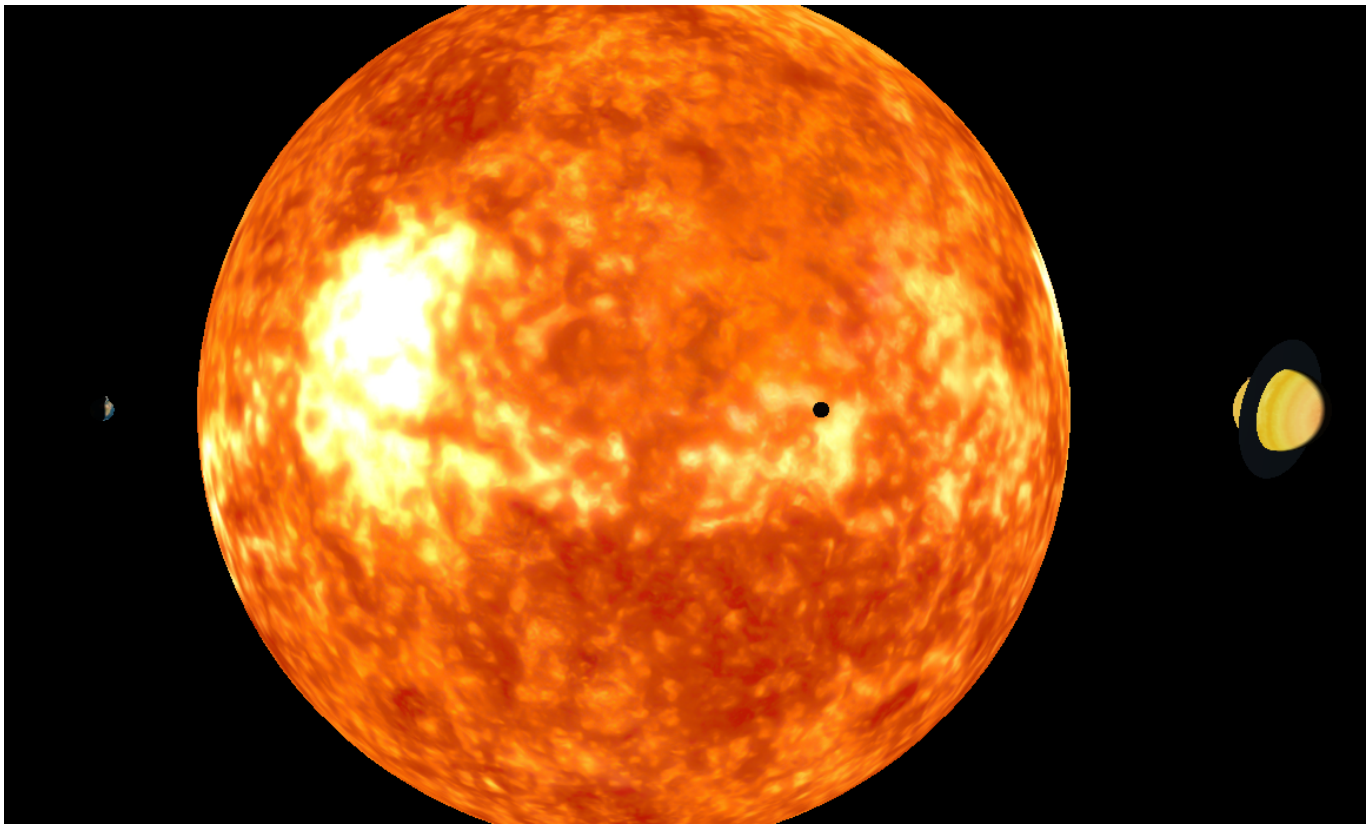

Figura 2.4: Sistema Solar com Terra e Saturno

Capítulo 3

Conclusão

Finda esta quarta e última fase, foi possível cumprir novamente com todos os requisitos pedidos. Infelizmente, não nos foi possível adicionar qualquer tipo de funcionalidades extra, uma vez que o tempo se revelou curto para o nosso grupo para o fazer. Ainda assim, tendo em conta o trabalho desenvolvido, consideramos que o resultado final corresponde às expectativas do grupo, uma vez que foi possível representar um sistema solar muito agradável e minimamente fiel à realidade.

Para além disso, a modularidade do código desenvolvido também se revelou uma vantagem que as fases anteriores trouxeram para esta última, minimizando o código necessário de alterar e acrescentar.