Licenciatura em Ciências da Computação

Computação Gráfica

Ultima Fase

Filipe Barbosa A77252

Luís Bigas A76964

Hugo Ferreira A78555

8 de Maio de 2019

Conteúdo

Capítulo 1

Introdução

1.1 Introdução

No âmbito da unidade curricular Computação Gráfica, foi proposta a criação de um motor gráfico.

Este contendo as capacidades de desenhar num espaço tridimensional objetos construidos por triângulos, estes representados um conjunto de três pontos.

O motor possibilita ainda a manipulação da iluminação da cena a ser desenhada e aplicação de texturas nos objetos.

Capítulo 2

Generator

O generator cria um ficheiro que contem os dados necessários para o desenho dos objetos, num ficheiro com o nome escolhido pelo utilizador.
No ficheiro resultante da execução do generator, podemos considerar que existem "grupos"de 3 linhas cada, cada um destes grupos representa um vértice da figura representada.
A primeira linha representa a posição do vértice, a segunda linha contem a normal deste que será utilizada para o calculo da iluminação, e por fim na terceira linha temos o ponto correspondente a posição numa textura.

2.1 Compilação

- make gen

2.2 Utilização

Geração dos objetos para serem utilizados no programa principal.

- ./generator sphere 1 10 10 sphere.3d
- ./generator box 1 1 1 10 box.3d

2.2.1 Objetos

Plane

O plano é calculado pela função **draw_plane2()**
Este é composto por 2 triângulos, formando um plano centrado na origem dos eixos X e Z.

As normais no plano são iguais para todos os vértices (0,1,0), pois o plano apenas tem uma face direcionada para o eixo Y.

As coordenadas de textura estão definidas diretamente, sendo o vértice (-x,-x) a coordenada (0,0) e o vértice (x,x) a coordenada (1,1).

Box

A caixa é calculada pela função **draw_box2()**.
A caixa é composta por 4 planos, cada um que representa uma das faces da mesma. Cada um destes planos contem múltiplos triângulos, o seu numero dependendo do nível de tesselação definida pelo utilizador.

Cada um dos vértices é calculado dentro de dois ciclos, um ciclo percorre o numero de triângulos na "horizontal"e outro na "vertical".

As normais são iguais dentro de cada uma das faces da caixa, dependendo da orientação da mesma.
A face "frontal"tem as normais na direção do eixo Z, no topo da caixa, as normais têm as normais na direção do eixo Y.

As coordenadas de textura são iguais para cada uma das faces. Multiplicando o inverso do valor de tesselação pela posição do vértice a ser calculado.

Sphere

A esfera é calculada pela função **draw_sphere()**.
Cada um dos seus vertices é calculado dentro de dois ciclos, um que percorre as posições na horizontal(slices) e o outro percorre as posições na vertical(stacks).

Cada um dos vértices é calculado com as equações:
x = raio * cos(beta) * sin(alpha)
y = raio * sin (beta)
z = raio * cos(beta) * cos(alpha)
Sendo beta e alpha calculados no inicio de cada ciclo.

As normais são calculadas de forma idêntica a cada um dos vértices, com a diferença de que não realizamos a multiplicação pelo raio, desta forma obtemos um vetor que aponta do centro da esfera ao vértice.

As coordenadas de textura são calculadas pela multiplicação do passo do ciclo da função pela fração 1/(numero de stacks/slices).

Capítulo 3

Engine

O Programa Engine executa a leitura do ficheiro XML descritor de uma cena, a leitura dos ficheiros gerados pelo Generator e realiza a impressão no ecrã.

3.1 Compilação

- make eng

3.2 Utilização

- ./engine NomeFicheiro.xml

3.2.1 Controlo

- J - Afasta a câmara do centro
- K - Aproxima a câmara do centro
- Setas - Move a câmara em torno do centro da cena, no sentido da seta pressionada

3.3 Funcionamento

3.3.1 Estruturas utilizadas

- lightScene

  Guarda a posição e tipo de luz

- translateT

  Guarda os pontos de controlo que definem uma curva Catmull-Rom, na qual um objeto se irá movimentar.

- rotateT

  Guarda as coordenadas ao longo que uma rotação irá ser realizada, e o tempo que esta demora.

- modelPoints

  Guarda os pontos que representam os objetos a ser desenhados, a sua cor, e propriedades de iluminação a ser aplicado ao mesmo.

- drawInfo

  Esta estrutura é a responsável por indicar ao programa o que deve ocorrer em ordem, Imprimir um objeto, realizar popMatrix, translação, etc...

3.3.2 Parsing

Para realizar este processamento, utilizando a biblioteca TinyXML2, verificamos primeiro se existe o nodo **<scene>**. Sendo este encontrado é chamada a função **ParserRoot()** que irá percorrer os nodos **<group>** e **<lights>** exteriores.
Após é chamada a função **ParserGroup()** que processará o conteúdo de cada um dos groups, assim como a possível chamada recursiva para estes.
Caso seja lido do o elemento **<models>** é então utilizada a função **ParserModels()** que percorre esta Tag e envia cada um dos seus elementos à próxima função **ParseObject()**.

Para guardar o conteúdo lido com estas funções foi utilizado um vector de estruturas que guarda em ordem que comandos funções do FreeGlut deverão ser chamadas e em que ordem.

3.3.3 Rendering

Após realizado o parsing, em primeiro lugar caso tenhamos definido algum tipo de iluminação, com função **setLight()** ativamos a iluminação geral no programa e a ativação de cada uma das luzes definidas, até 8 luzes no máximo.

De seguida percorremos cada um dos modelos definidos, e em cada um destes, caso esteja definida uma textura, indicado pela flag **Ftexture**, os dados da textura são guardados no GLuint *texID.

Agora, já temos todos os elementos necessários para iniciar a impressão da cena em si.
A função **newrenderScene** é chamada em cada frame do programa. Esta inicia um ciclo que percorre o vector de estrutura **drawInfo** e executa o indicado por esta mesma, chamado as funções apropriadas.

Caso a estrutura indique para realizar o desenho de um objeto, então é feita a chamada da função **drawVBO**, esta recebe a estrutura representante de um modelo.
Nesta função é realizado o carregamento dos vértices, normais e coordenadas de textura para Buffers e a aplicação da iluminação a cada um dos objetos.

3.3.4 Notas sobre a Engine

Caso no ficheiro XML não estejam definidas luzes, irá ser utilizada a definição normal de cor. Caso contrário, esta definição tona-se redundante, e são utilizados os parâmetros de luz difusa, emissiva, especular e ambiente.

Capítulo 4

Ficheiro XML

4.1 Tags Disponíveis

- <scene>
- <lights>

  Define o inicio da definição da iluminação.
- <light type=”tipo de luz”posX=”posição em X”posY=”posição em Y”posZ=posição em Z”>
- <group>
- <scale>
- <rotate>
- <translate>
- <models>
- <model>

Apenas é permitido uma Tag **scene** e uma Tag **lights**.
Por outro lado podemos ter multiplos **group** e **group's** dentro de outros **group's**.

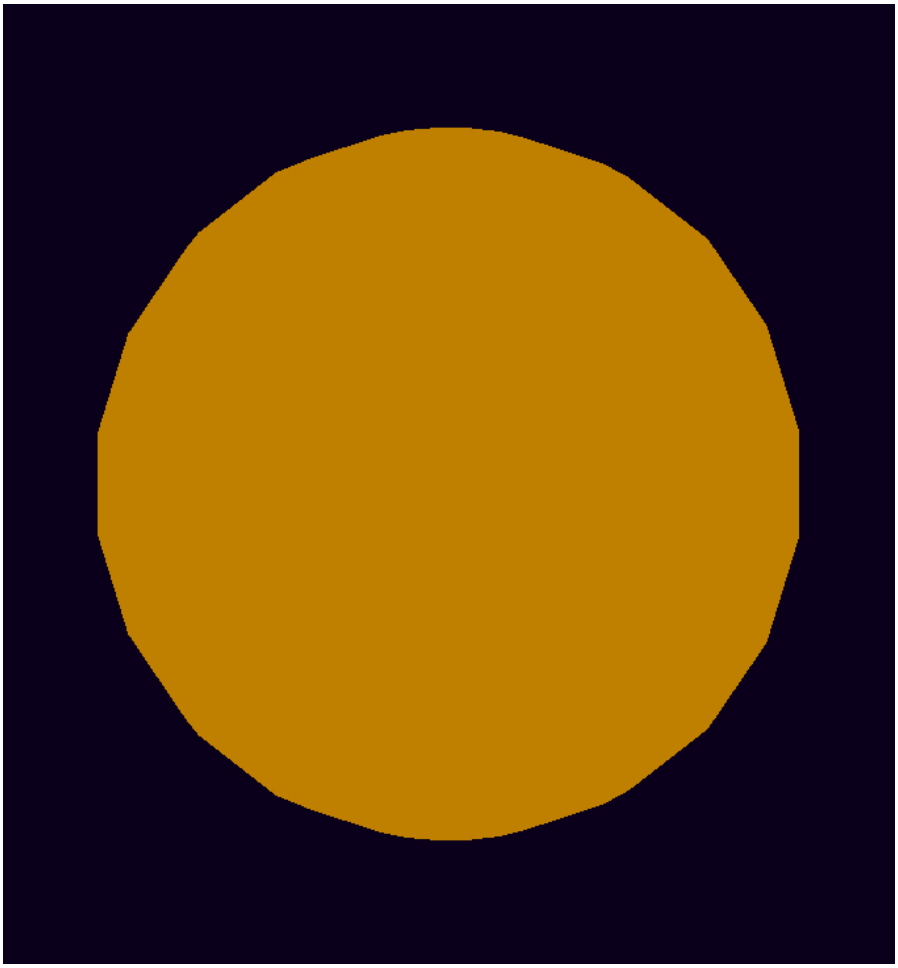
Capítulo 5

Programa em Funcionamento

Por fim podemos demonstrar o funcionamento do nosso programa através de alguns exemplos.

5.1 Sem texturas nem luzes

```
<scene>
    <group>
        <scale X="30" Y="30" Z="30" />
        <models>
            <model file="sphere.3d" R="0.75" G="0.5" B="0" />
        </models>
    </group>
</scene>
```



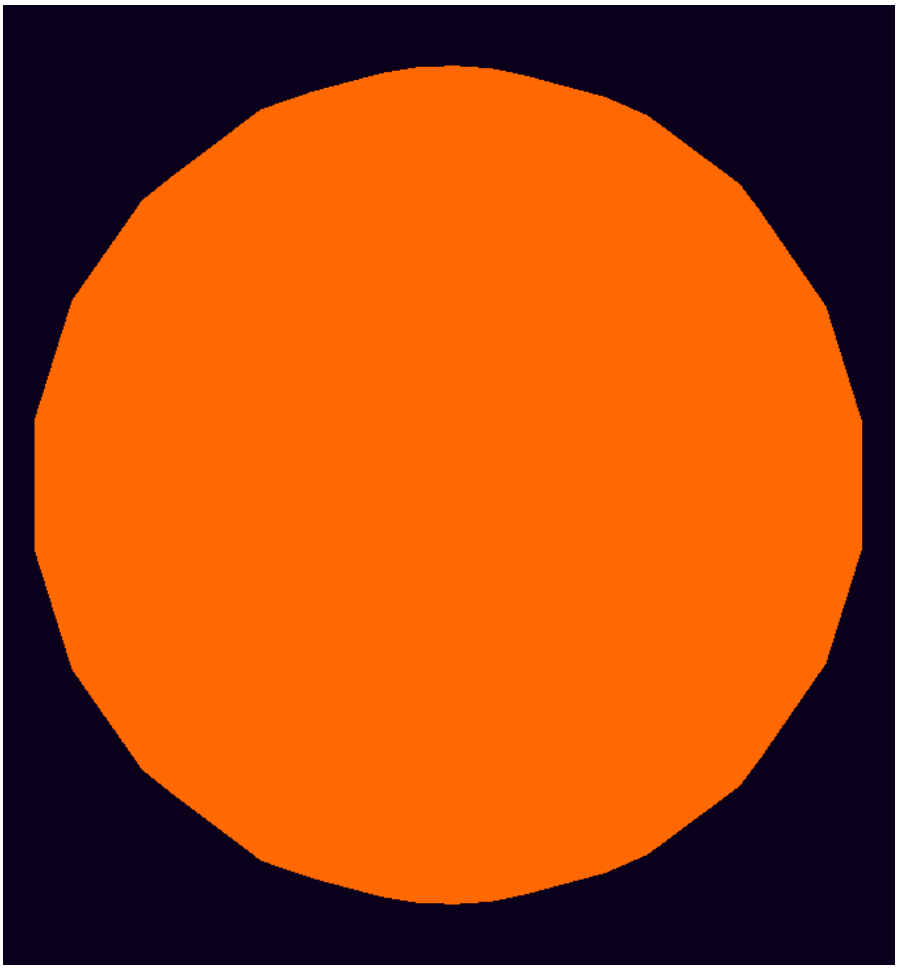
5.2 Exemplo de iluminação

```
<scene>
    <lights>
        <light type="POINT" posX="0" posY="0" posZ="0" />
    </lights>

    <group>
        <scale X="30" Y="30" Z="30" />
        <rotate time="50" axisX="0" axisY="1" axisZ="0" />
        <models>
            <model file="sphere.3d" emisR="1" emisG="0.4117" emisB="0" />
```

```
        </models>
    </group>

</scene>
```



Capítulo 6

Sistema Solar

```
<scene>
    <lights>
        <light type="POINT" posX="0" posY="0" posZ="0" />
    </lights>
    <group>
        <!-- SOL -->
        <group>
            <scale X="30" Y="30" Z="30" />
            <rotate time="50" axisX="0" axisY="1" axisZ="0" />
            <models>
                <model file="sphere.3d" texture="2k_sun.jpg" emisR="1" emisG="1" emisB="1" /
            </models>

        </group>

        <!-- MERCURIO -->
        <group>
            <rotate time="100" axisX="0" axisY="1" axisZ="0" />
            <translate X="0" Y="0" Z="35" />
            <rotate time="10" axisX="0" axisY="1" axisZ="0" />
            <models>
                <model file="sphere.3d" texture="2k_mercury.jpg" diffR="1" diffG="1" diffB="
            </models>
        </group>

        <!-- VENUS -->
        <group>
            <rotate time="150" axisX="0" axisY="1" axisZ="0" />
            <translate X="0" Y="0" Z="45" />
            <scale X="1.5" Y="1.5" Z="1.5" />
            <rotate time="10" axisX="0" axisY="1" axisZ="0" />
```

```
    <models>
        <model file="sphere.3d" texture="2k_jenus_surface.jpg" diffR="1" diffG="1" d
    </models>
</group>

<!-- TERRA -->
<group>
    <rotate time="200" axisX="0" axisY="1" axisZ="0" />
    <translate X="0" Y="0" Z="60" />
    <scale X="2" Y="2" Z="2" />
    <rotate time="20" axisX="0" axisY="1" axisZ="0" />
    <models>
        <model file="sphere.3d" texture="2k_earth.jpg"  diffR="1" diffG="1" diffB="1
    </models>
</group>

<!-- MARTE -->
<group>
    <rotate time="270" axisX="0" axisY="1" axisZ="0" />
    <translate X="0" Y="0" Z="70" />
    <scale X="1.7" Y="1.7" Z="1.7" />
    <rotate time="20" axisX="0" axisY="1" axisZ="0" />
    <models>
        <model file="sphere.3d" texture="2k_mars.jpg" diffR="1" diffG="1" diffB="1"
    </models>
</group>

JUPITER
<group>
    <rotate time="350" axisX="0" axisY="1" axisZ="0" />
    <translate X="0" Y="0" Z="85" />
    <scale X="10" Y="10" Z="10" />
    <rotate time="5" axisX="0" axisY="1" axisZ="0" />
    <models>
        <model file="sphere.3d" texture="2k_jupiter.jpg" diffR="1" diffG="1" diffB="
    </models>
</group>

SATURNO
<group>
    <rotate time="400" axisX="0" axisY="1" axisZ="0" />
    <translate X="0" Y="0" Z="110" />
    <scale X="9" Y="9" Z="9" />
    <rotate time="5" axisX="0" axisY="1" axisZ="0" />
    <models>
        <model file="sphere.3d" texture="2k_saturn.jpg" diffR="1" diffG="1" diffB="1
```

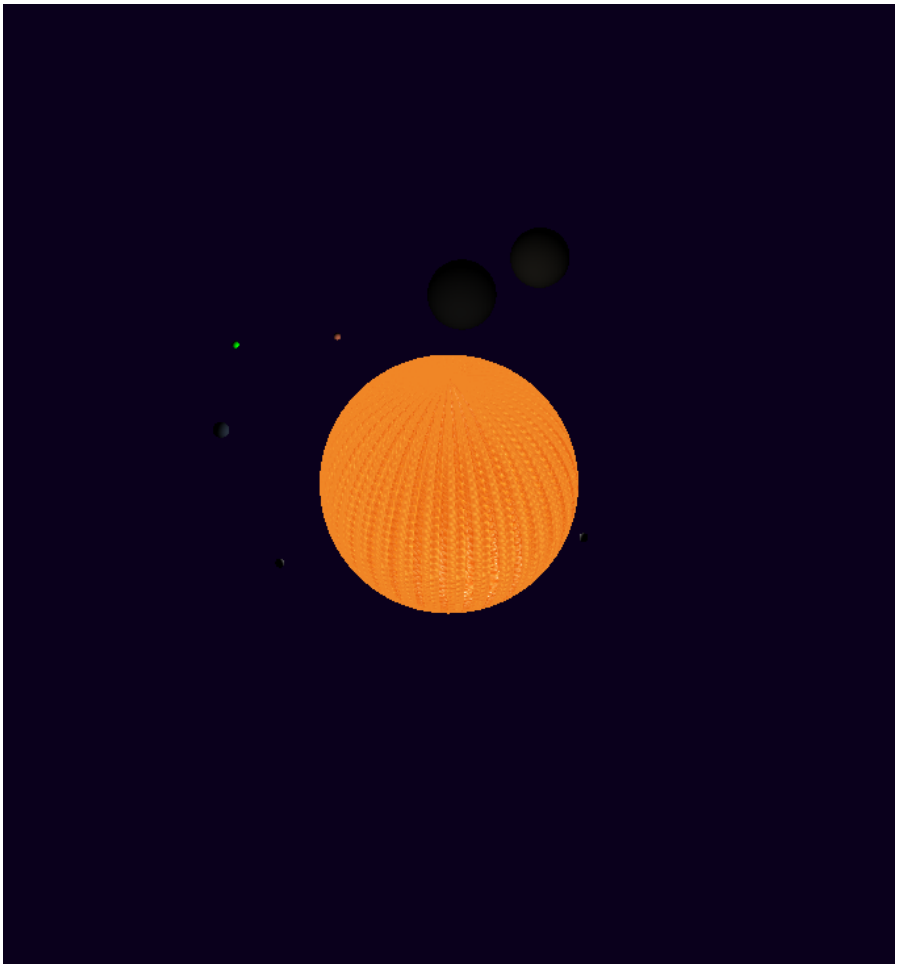
```
            </models>
        </group>

    </group>

    <group>
        <translate time="50" >
            <point X="40" Y="0" Z="200" />
            <point X="-40" Y="0" Z="200" />
            <point X="-70" Y="0" Z="0" />
            <point X="-40" Y="0" Z="-200" />
            <point X="40" Y="0" Z="-200" />
            <point X="70" Y="0" Z="0" />
        </translate>
        <models>
            <model file="sphere.3d" diffR="0" diffG="1" diffB="0"/>
        </models>

    </group>

</scene>
```



Capítulo 7

Conclusão

Ao longo da realização deste trabalho tivemos a oportunidade de aprender vários métodos matemáticos de calculo geométrico para aplicação em programas gráficos, como diferentes métodos de desenho, calculo de iluminação, com o auxilio de matrizes.
Aprendemos ainda como é realizada a execução a nível de hardware, interação entre o processador e placa gráfica.

Apesar de não termos atingido todos os objetivos impostos pelo enunciado do trabalho, penso que com algum tempo seria possível resolver os problemas, como calcular de forma correta as coordenadas de textura e os patches de bezier.