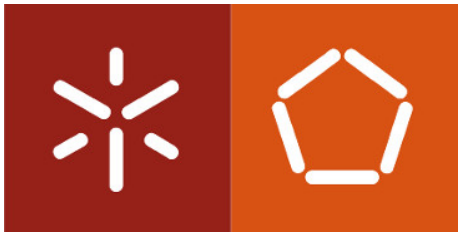
Computação Gráfica

Trabalho Prático - Fase 1

Grupo nr.	38
a83899	André Morais
a84577	José Pedro Silva
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1 Introdução

O objetivo desta segunda fase do trabalho prático consiste em melhorar o motor gráfico iniciado na primeira fase, através do processamento de ficheiros XML e a aplicação de transformações geometricas em OpenGL (translações, rotações e escalas). Estas transformações serão responsáveis pelo modo como as primitivas desenvolvidas anteriormente serão exibidas. Todo o trabalho desenvolvido nesta fase será aplicado a um modelo do Sistema Solar que incluirá o Sol, os Planetas e as respetivas Luas. Apresentaremos ainda alguns exemplos de execução a partir de vários modelos em ficheiros XML.

2 Utilização

De modo a correr a demonstração do sistema solar, é necessário dar build do programa, gerar uma esfera com raio 1, com as slices e stacks que desejar e guardar no ficheiro *sphere.3d*; gerar um torus com raio interno 0.9 e raio externo 1 e guardar no ficheiro *torus.3d* e por fim correr o programa **engine**.

3 Arquitetura

A aplicação desenvolvida está dividida em diferentes *packages*. Como podemos ver na Figura 1, existem 5 pastas diferentes onde cada nome é análogo à sua função no trabalho. Na pasta **engine** está contido o código que origina o motor da aplicação; a **generator** contém o código que através de um conjunto de argumentos, gera uma sequência de pontos tridimensionais, sendo estes guardados na pasta models, para posteriormente serem gerados pelo motor; a **rapidxml** é composta pelo código responsável por fazer *parese* do ficheiro *config.xml*; na pasta **lib** encontram-se as classes que auxiliam a construção do cenário a partir do ficheiro xml.

Figura 1: Arquitetura da aplicação.

3.1 Engine

O programa **engine** está dividido em duas fases, sendo estas, a **leitura** e extração de dados do ficheiro XML e o **desenho** da cena.

3.1.1 Primeira Fase - Leitura e Parsing

Nesta primeira fase, é lido o ficheiro *config.xml* e guardadas em memória as informações relativas às transformações de cada grupo e subgrupo.

3.1.2 Segunda Fase - Desenho

Nesta fase, o programa desenha a cena previamente guardada através de um ciclo sobre um vetor que contém cada grupo e cada transformação.

3.2 Generator

Como apresentado na fase anterior, o *generator* mantém a mesma estrutura e funcionamento, tendo sido apenas acrescentado o torus, estando a explicação do mesmo na secção 6.

3.3 Processamento do ficheiro XML

Como explicado anteriormente, a primeira fase do programa *engine* consiste em processar um ficheiro XML e guardar as informações em memória. Com o objetivo de melhor processamento, foi criada uma classe **Group** que contém um conjunto de transformações (**Transformation**), um vetor com os respetivos subgrupos (**vector<Group>**) e um vetor com os pontos tridimensionais correspondentes aos models deste grupo (**vector<Point>**).

```
class Group{
  private:
    Transformation t;
    vector<Group> subg;
    vector<Point> points;
```

Figura 2: Classe **Group**

```
class Transformation{
  private:
    Translation t;
    Rotation r;
    Scale s;
    Color c;
```

Figura 3: Classe **Transformation**

3.3.1 Parser

Como ferramenta de *parsing* utilizamos o *rapidxml* (uma vez que foi utilizado na primeira fase do trabalho). Assim, o processo torna-se simples, precisamos apenas de percorrer cada grupo, procurar e guardar as transformações pedidas (rotação, translação, escalamento e/ou mudança de cor), guardar os pontos presentes nos models apresentados e percorrer todos os subgrupos, onde são guardadas as mesmas informações. No final iremos obter um vetor com todos os grupos, onde estão guardadas todas as transformações de cada grupo e

subgrupo.

4 Algoritmo de Desenho

Com todas as informações obtidas através do processamento do ficheiro xml, é agora possível desenhar os pontos correspondentes depois de realizar as transformações, tudo isto depois de fazer **PushMatrix** e antes de fazer **PopMatrix** para cada grupo. No entanto, como as operações devem passar do grupo para os seus subgrupos, então o desenho dos subgrupos é feito antes do **PopMatrix**, como mostrado na Figura 4.

```
void drawGroup(Group g){
  glPushMatrix();

  Color c = g.getTransformation().getColor();
  Translation t = g.getTransformation().getTranslation();
  Rotation r = g.getTransformation().getRotation();
  Scale s = g.getTransformation().getScale();

  glRotatef(r.getAngle(),r.getX(),r.getY(),r.getZ());
  glTranslatef(t.getX(),t.getY(),t.getZ());
  glScalef(s.getX(),s.getY(),s.getZ());

  vector<Point> pontos = g.getPoints();

  glBegin(GL_TRIANGLES);
  glColor3f(c.getR(),c.getG(),c.getB());
  for(int i=0;((long unsigned int)i)<pontos.size();i++){
    Point p = pontos[i];
    glVertex3f(p.getX(),p.getY(),p.getZ());
  }
  glEnd();

  vector<Group> subgroups = g.getSubGroups();
  for(int j=0;((long unsigned int)j)<subgroups.size();j++){
    Group sg = subgroups[j];
    drawGroup(sg);
  }
  glPopMatrix();
}
```

Figura 4: Algoritmo de desenho para cada grupo pesente no vetor

5 Exemplo de Execução

De seguida iremos demostrar um exemplo da execução deste programa. Para isto, criamos um ficheiro config.xml onde está representado o sistema solar, seguindo este uma escala aproximada da realidade, no entanto ajustada de modo a ficar visualmente mais atrativo.

O resultado obtido foi o seguinte:

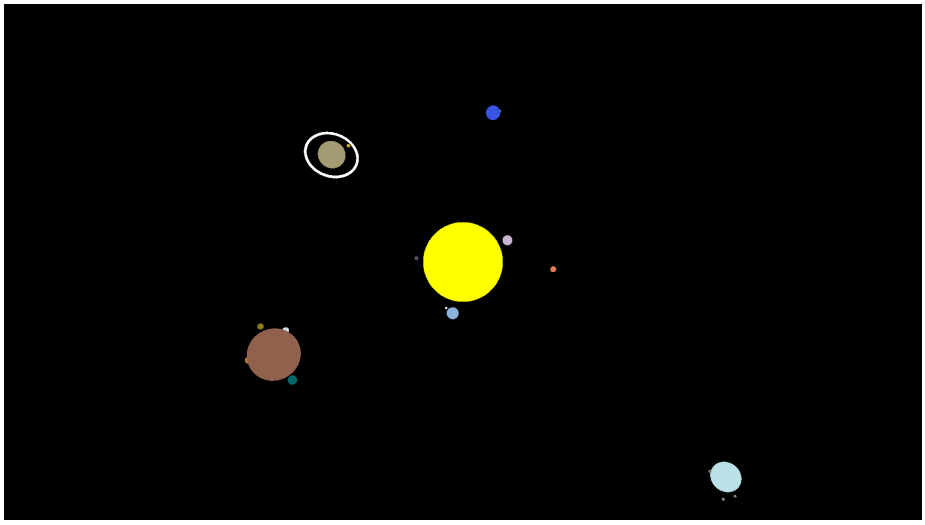


Figura 5: Sistema Solar representado pelo nosso programa com luas e planetas

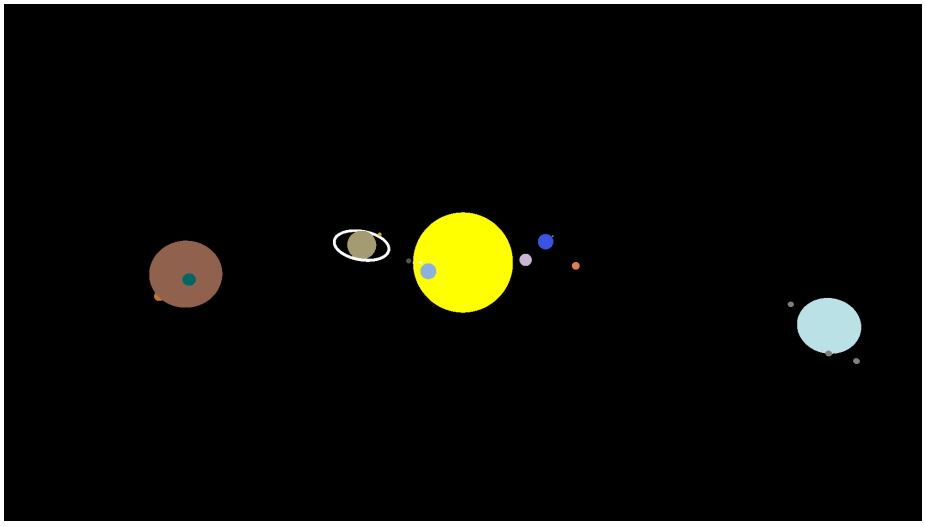


Figura 6: Sistema Solar representado pelo nosso programa com luas e planetas.

6 Extras

À medida que fomos avançando no trabalho, deparamo-nos com a necessidade de desenhar os anéis de Saturno, para resolver esta situação, decidimos gerar pontos de forma a contruir uma espécie de Torus. De modo a gerar um *torus*, é necessário fornecer o raio interior, raio exterior, o número de slices e o número de stacks. Na Figura 7 mostramos a nossa visão em relação aos raios do Torus, onde **R** representa o raio exterior e **r** representa o raio interior.

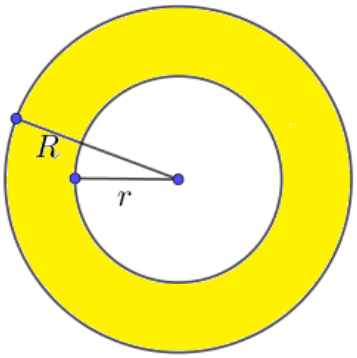


Figura 7: Visão sobre o nosso torus.

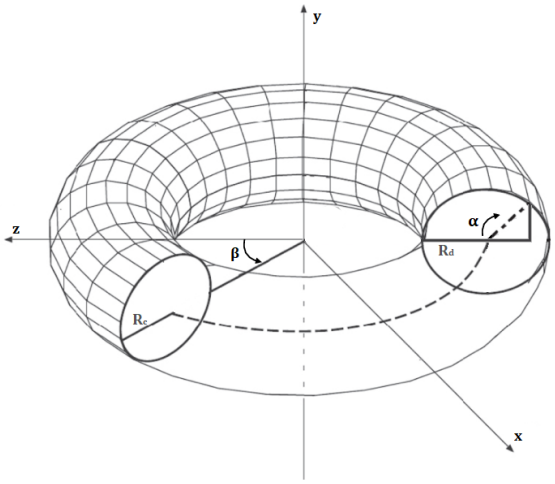


Figura 8: Ângulos e raios usados no cálculo dos pontos.

Olhando para a Figura 8, facilmente retiramos as coordenadas dos pontos.

É apenas necessário ter em conta que:

$$Rc = \frac{Re - Ri}{2}$$

$$Rd = Ri + Rc - Rc \times cos(\alpha)$$

Por necessidade, foi criada uma variável *rds* que contém o Rd do ponto seguinte do triângulo e varia do Rd apenas no ângulo:

$$rds = Ri + Rc - Rc \times cos(\alpha + \Delta\alpha)$$

Assim, as variações de α e β serão:

$$\Delta\beta = \frac{2 \times \pi Re}{nsl}$$

$$\Delta\alpha = \frac{2 \times \pi Re}{nst}$$

Posto isto, as coordenadas dos pontos dos triângulos de cada retângulo serão:

Triângulo 1:

$$x = Rd \times sin(\beta); y = Rc \times sin(\alpha); z = Rd \times cos(\beta)$$

$$x = rds \times sin(\beta); y = Rc \times sin(\alpha + \Delta\alpha); z = rds \times cos(\beta)$$

$$x = rds \times sin(\beta + \Delta\beta); y = Rc \times sin(\alpha + \Delta\alpha); z = rds \times cos(\beta + \Delta\beta)$$

Triângulo 2:

$$x = rds \times sin(\beta + \Delta\beta); y = Rc \times sin(\alpha + \Delta\alpha); z = rds \times cos(\beta + \Delta\beta)$$

$$x = Rd \times sin(\beta + \Delta\beta); y = Rc \times sin(\alpha); z = Rd \times cos(\beta + \Delta\beta)$$

$$x = Rd \times sin(\beta); y = Rc \times sin(\alpha); z = Rd \times cos(\beta)$$

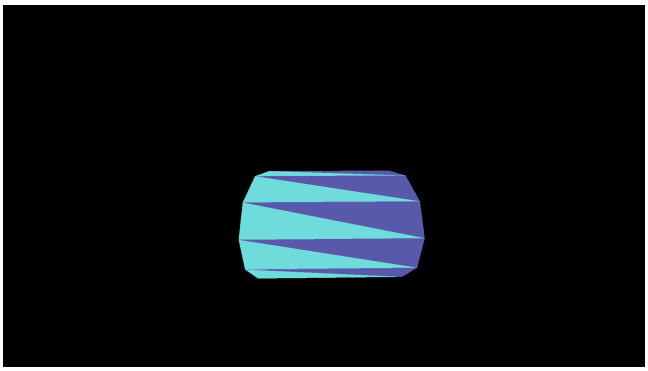


Figura 9: Resultado de uma slice.

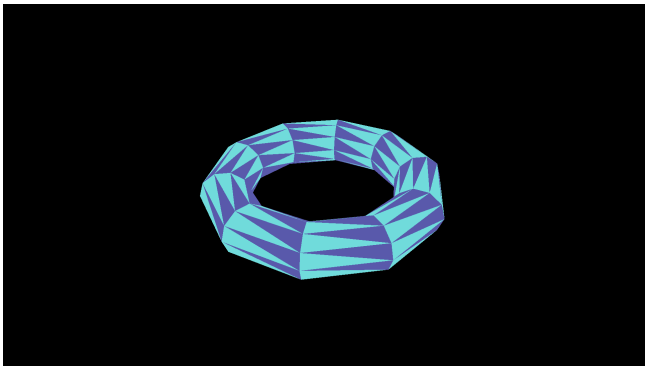


Figura 10: Resultado final.

7 Conclusão

Esta segunda fase permitiu-nos aprofundar os conhecimentos obtidos tanto nas aulas teóricas, como nas transformações geométricas, e aplicá-los a nível prático. No geral, consideramos bastante satisfatórios os resultados apresentados para esta fase, pois o Sistema Solar corresponde ao que era espectado. Em jeito de conclusão, nas fases seguintes, esperamos uma otimização a nível do desenho dos modelos com o recurso a VBO's em vez de utilizar o desenho direto a partir do uso da expressão em OpenGL "GL_TRIANGLES".