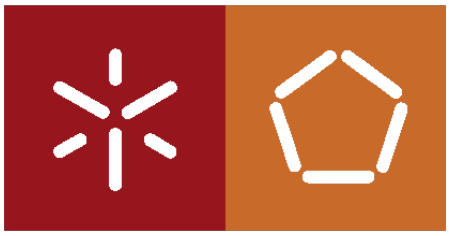

Universidade do Minho

MIEI - 3º ano - 2º semestre
Universidade do Minho

Trabalho Prático - Fase 2

Grupo 11


João Vieira
A76516


Manuel Monteiro
A74036


Bernardo Viseu
A74618


Fernando Pereira
A75496

June 4, 2019

Conteúdo

1 Introdução

1.1 Contextualização

No contexto curricular da disciplina de Computação Gráfica, foi nos proposto a criação de um mecanismo 3D com base num cenário gráfico, tendo por base todos os conhecimentos adquiridos até ao momento nas aulas e utilizando a linguagem de programação C++. Este trabalho prático está dividido em quatro fases, sendo este relatório referente à segunda fase, que tem como objetivo a conceção de algumas transformações geométricas.

1.2 Resumo

Como foi referido, esta trata-se da segunda fase do projeto prático, é natural que se mantenham algumas das funcionalidades criadas na fase anterior, e por outro lado, de modo a cumprir com os requisitos necessários, algumas tiveram de ser alteradas.

Assim, a principal mudança verificada foi na forma como o *engine* processava a informação contida no ficheiro de input *xml*. A estrutura deste ficheiro também sofre alterações, pois em vez de terem apenas o nome do ficheiro com as primitivas que se pretende exibir, este contem agora a formação de diversos grupos hierárquicos com esses ficheiros. Os grupos podem ter associados transformações geométricas (*translate, rotate* e *scale*) que serão responsáveis pelo modo como cada uma das primitivas, previamente criadas na fase anterior, serão exibidas. Para tal, não só temos de mudar a forma como lemos o ficheiro, mas também tiveram de ser criadas algumas classes para armazenar e relacionar a informação.

Tudo isto será utilizado com a finalidade de produzir um modelo estático do Sistema Solar, através do conjunto primitivas geradas e exibidas. Desta forma, para além dos requisitos mínimos requeridos, decidimos implementar algumas funcionalidades extra como a inclusão da primitiva gráfica *Torus*, para uma representação mais realista de Saturno.

O Gerador (*Generator*) é responsável por gerar a informação de cada modelo, guardando os seus vértices num ficheiro específico.

O Motor (*Engine*) tem a função de ler a configuração de um ficheiro XML e exibir os respetivos modelos.

2 Arquitetura

Em relação à primeira fase do trabalho foram criadas novas classes, algumas classes foram alteradas e as restantes mantiveram-se inalteradas. Nesta secção só faremos menção às classes novas e às que foram modificadas.

2.1 Aplicações

Estas aplicações também marcaram presença na primeira fase, mas tiveram de ser alteradas.

2.1.1 Gerador

generator.cpp: Como foi explicado na fase anterior do trabalho prático, esta aplicação é responsável por gerar os vértices das respetivas figuras geométricas, armazenando cada conjunto de vértices num ficheiro 3d. A única alteração foi a adição da primitiva Torus, passando o generator também a produzir o ficheiro **torus.3d**.

2.1.2 Motor

engine.cpp: Esta é a aplicação que permite visualizar as primitivas geométricas e interagir com elas através de comandos. Como a estrutura do ficheiro XML foi alterada, também foi necessário reajustar a aplicação de modo a realizar o parsing com sucesso. Também foi necessário modificar o modo como as formas são renderizadas, para permitir o uso de uma nova estrutura que consegue armazenar vários tipos de informação.

2.2 Classes

Todas classes da fase 1 também estão nesta fase, no entanto foi necessário acrescentar 3 classes (Parser.cpp, Action.cpp e Group.cpp) de modo a ser possível responder às novas exigências do enunciado.

2.2.1 Parser

Parser.cpp: Nesta classe estão os métodos que permitem realizar a leitura do ficheiro XML. É então realizada a leitura da informação que seja relevante e armazenada na sua correta estrutura. A ferramente utilizada foi o tinyxml2.

2.2.2 Action

Action.cpp: Contém as informações relativas às ações que vão ser aplicadas às primitivas geométricas, sendo estas:

- *Translation*;
- *Rotation*;
- *Scale*;
- *Color*.

2.2.3 Group

Group.cpp: A função desta classe é armazenar a informação relativa a um grupo. A informação é armazenada em 3 listas. Na primeira estão contidos os modelos do grupo (shapes), na segunda os filhos(groups) pois cada grupo pode ter outro grupo dentro de si e a terceira lista contém as transformações geométricas (actions).

2.2.4 Models

models.cpp: Classe que contém todos os algoritmos necessários para gerar cada forma geométrica, manipulando o seu conjunto de pontos. A diferença para a primeira fase é simplesmente a adição da nova forma geométrica **torus**.

3 Primitivas Geométricas

3.1 *Torus*

Um *Torus* é um sólido geométrico que apresenta uma forma semelhante a uma câmara de pneu. Já em termos geométricos, pode ser definido como o lugar geométrico tridimensional formado pela rotação de uma superfície circular de raio interior em torno de uma circunferência de raio exterior. Posto isto, os parâmetros para formar o *Torus* são o raio interior (r), o raio exteriror (R), o número de lados por secção radial ($nslices$) e o número de anéis ($nring$).

3.1.1 Algoritmo

Como o *Torus* é definido por duas circunferências, primeiro tivemos de decidir por quais dos eixos é que a circunferência exterior, de raio R, gira em torno. Sendo assim, ficou decidido que esta iria girar em torno do eixo Z, ou seja esta é apenas definida pelas coordenadas X e Y. A circunferência interior, de raio r, já é definida pelas 3 coordenadas.

Iterando pelos anéis, ou seja circunferência exterior, para cada anel são calculadas os deslocamentos das posições X e Y, que nos vão limitar um anel, através do ângulo de deslocamento (α_Shift), obtendo:

$$
\begin{aligned}
&ringSize = 2\pi/nrings \\
&\alpha\_Shift = i * ringSize \\
&next\alpha = \alpha + ringSize \\
&x = \cos\alpha \\
&y = \sin\alpha \\
&x\_S = \cos next\alpha \\
&y\_S = \sin next\alpha
\end{aligned}
$$

Já para cada anel temos as suas fatias, pois também se trata de uma circunferência. Para sabermos o ângulo (β_Shift) para calcular cada fatia, basta, tal como para cada anel, dividir o ângulo de rotação pelo número de fatias. Para calcular as coordenadas X e Y que delimitam cada fatia, temos de multiplicar o raio interior pelo cosseno do ângulo por fatias, e aplicar o deslocamento do raio exterior somando-o para desenhar a partir desse afastamento. Depois apenas multiplicamos pelos limites do anel, já calculado, obtendo assim as coordenadas finais. Já para a variável Z, como esta se mantêm constante na circunferência exterior contida no plano XY, apenas se vai deslocar pelo ângulo de fatia de cada anel, e tal como nas outras variáveis temos de calcular os 2 limites da fatia. Após esta divisão obtemos um rectângulo que pode ser composto por 2 triângulos em que 2 dos vértices são comuns em ambos, ou seja apenas necessitamos de 4 vértices. Com isto obtemos então:

$$
\begin{aligned}
&\beta\_Shift = 2\pi/nslices \\
&r0 = r * \cos(j * \beta\_Shift)) + R \\
&r1 = r * \cos(j + 1 * \beta\_Shift)) + R
\end{aligned}
$$

Calculando os vértices:

$$
\begin{aligned}
&v1(x1, y1, z1) && v2(x2, y2, z1) \\
&x1 = x * r0 && x2 = x\_S * r0 \\
&y1 = y * r0 && y2 = y\_S * r0 \\
&z1 = r * \sin(j * \beta\_Shift) && z1 = r * \sin(j * \beta\_Shift)
\end{aligned}
$$

$v3(x3, y3, z2)$ | $v4(x4, y4, z2)$

$x3 = x * r1$ | $x4 = x\_S * r1$

$y3 = y * r1$ | $y4 = y\_S * r1$

$z2 = r * \sin((j+1) * \beta\_Shift)$ | $z2 = r * \sin((j+1) * \beta\_Shift)$

Apresentamos uma imagem para ajudar a perceber quais os pontos que foram calculados e por fim apenas tivemos de ter em atenção à ordem de criação dos mesmos. Para melhor visualização e percepção do algoritmo, aplicámos uma rotação aos eixos.

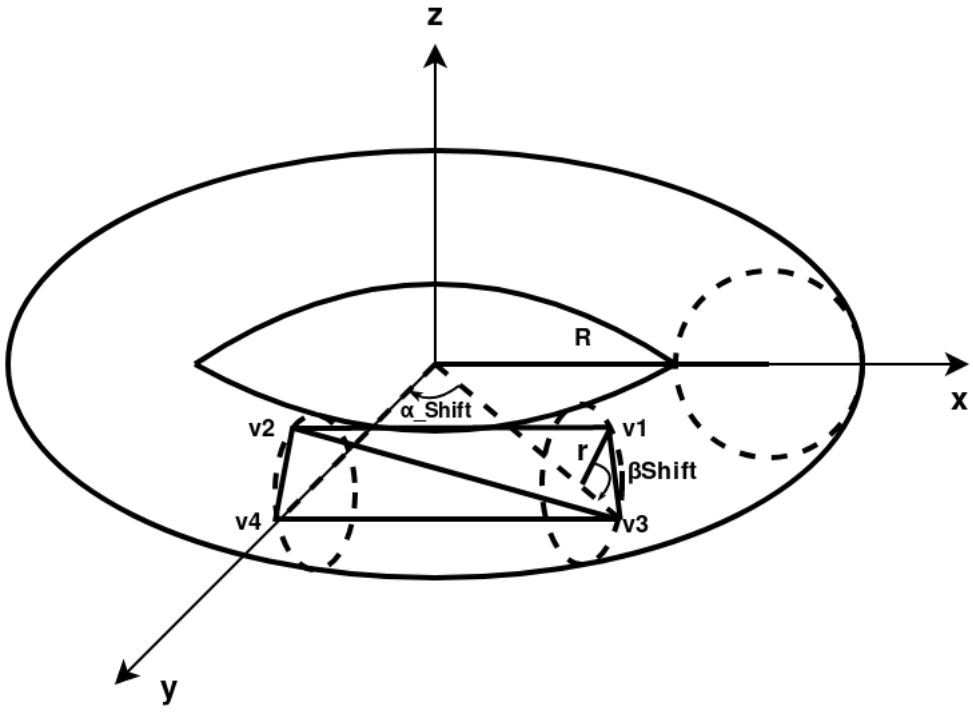

Figura 1: Ilustração da construção do *Torus*

4 Parser

Nesta classe é onde se realiza o processo de leitura dos ficheiros que contêm, em XML, o pretendido sistema solar. O processo de leitura começa a percorrer o ficheiro através da função **find_elements(XMLElement* element, Group* g)** que recebe o elemento XML que está a receber e o grupo onde a informação fica armazenada. Quando chamamos esta função pela primeira vez, sabemos que estamos a ler o primeiro elemento do ficheiro, portanto o *element* que vai receber como parâmetro é o primeiro filho do elemento **scene** que será o primeiro grupo. A seguir é necessário verificar se estamos perante uma transformação, caso seja (e dependendo de qual transformação seja), umas destas funções é chamada:

- **parse_translation(element,g)**
- **parse_rotation(element,g)**
- **parse_scale(element,g)**

- **parse_color(element,g)**

Cada uma delas adiciona a informação no grupo que está a ser percorrido.

Se não estivermos perante uma transformação, então é certo que só poderá ser uma lista de modelos, um filho ou passaremos para o elemento irmão.

No caso de ser uma lista de modelos, é chamada a função **parse_models(element)** que vai, recursivamente para cada modelo, criar uma forma *Shape* (através da função **read_file(modelName)**) onde o conteúdo é o conjunto dos pontos que originam a primitiva geométrica. A lista das *Shapes* é adicionada à lista de formas do grupo.

No caso de ser um filho, cria-se um novo grupo e insere-se na lista de filhos do grupo pai. Depois utilizamos novamente a função *find_element* onde o input será o grupo filho e o processo repete-se.

Caso nenhuma das referidas condições se verifique, o processo é repetido com o elemento irmão.

5 Estruturas de Dados

Depois de realizado todo o *parsing* do ficheiro XML, como foi explicado em cima, foi necessário definir as estruturas de dados que iriam guardar toda a informação da leitura. Desta forma, a nossa estrutura que irá guardar cada grupo lido no XML (**Group**) é definida com os seguintes argumentos:

- **vector<Shape*>** - Lista de formas, em que cada uma contém os pontos necessários para construir as figuras do grupo.
- **vector<Group*>** - Lista de grupos, pois de modo a respeitar a hierarquia definida no ficheiro de configuração, foi indispensável criar esta lista com grupos-filho, com a capacidade de herdar todas as transformações aplicadas ao grupo-pai.
- **vector<Action*>** - Lista de ações, que basicamente guarda toda as transformações aplicadas ao grupo, podendo cada ação ser, como já foi mencionado, uma *translation*, *rotation*, *scale* ou *color*.

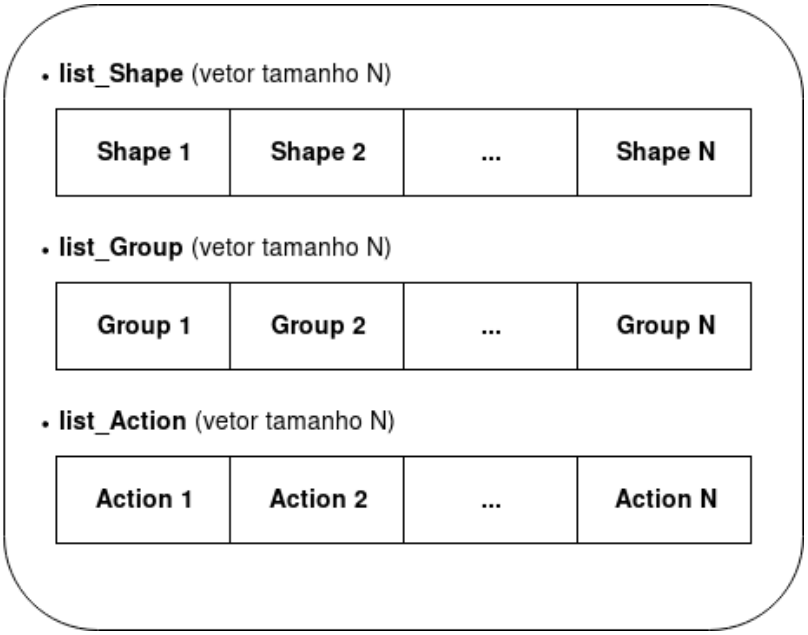

Figura 2: Estrutura de dados de cada grupo

6 Renderização do Modelo

O parser explicado na secção anterior guarda os todos os grupos numa variável chamada **scene**. Esses dados são renderizados na função **renderScene**, que tem uma pequena alteração em relação à da fase anterior. Os métodos **glPushMatrix** e **glPopMatrix** permitem guardar o estado inicial da matriz de transformação, para que após serem feitas as alterações indicadas voltemos ao estado inicial.

Começa-se por percorrer o vetor das actions e para cada ação aplica-se a função **apply** que efetua as transformações. Depois disso percorre-se o vetor das shapes e desenha-se os pontos de cada modelo através de triângulos.

Desenhado o primeiro grupo, é utilizada recursividade chamando novamente a função render para os filhos.

7 Análise do Modelo

Em relação ao modelo final, chegamos ao resultado pretendido representando à escala todos os planetas tendo em conta o raio de cada um assim como a sua distância média ao sol. Também foram alteradas as cores, e adicionados os satélites naturais mais relevantes de alguns planetas, de forma a termos um sistema solar o mais semelhante possível com a realidade. As translações efectuadas aos planetas, em relação ao Sol, foram todas efetuadas no eixo X. Já para as luas de cada planeta isso já não se verifica.

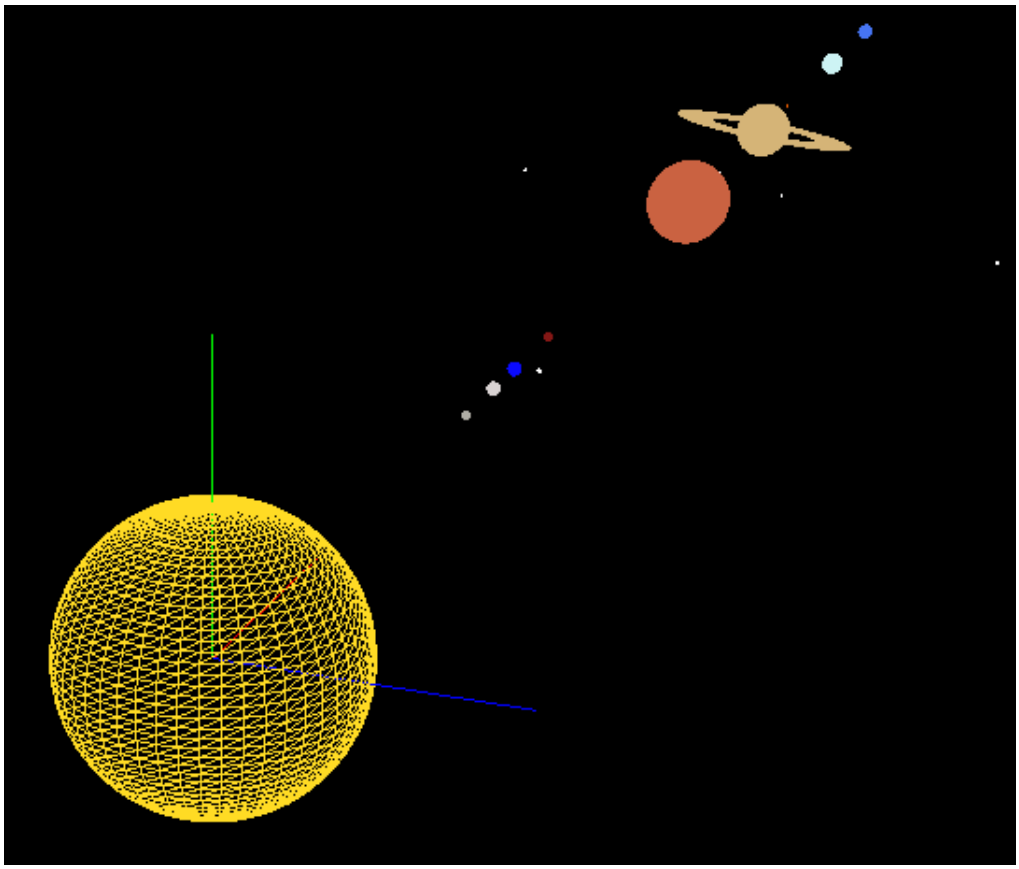

Figura 3: Visualização do Sistema Solar

8 Conclusão

Consideramos que esta segunda fase foi um melhoramento do que fizemos na primeira fase, e por isso, para além de aperfeiçoarmos os conhecimentos utilizados na primeira fase, obtivemos uma melhor noção da matéria estudada nesta cadeira, como a utilização de glPushMatrix e glPopMatrix, para o uso de translações e rotações.

Foi também importante na medida que consolidamos conhecimentos no processo do parser, movimento de câmara, e inserção de dados nas estruturas criadas.

Estamos satisfeitos com o resultado final do trabalho prático e consideramos que corresponde às expectativas que tínhamos em mente do que seria esta primeira visualização do nosso Sistema Solar.

Estamos deste modo ansiosos por começarmos a próxima fase do trabalho e sempre com a ambição de melhorar e aprender mais.