Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 2 - Transformações Geométricas

Daniel José Silva Furtado A97327
Ricardo Lopes Lucena A97746
Ricardo Silva Machado Araújo A96394
Nuno Miguel Leite da Costa A96897

A97327 A97746 A96394 A96897

Conteúdo

1 Introdução

A próxima fase do nosso projeto requer uma abordagem mais sofisticada, onde temos como principal objetivo incorporar novas funcionalidades ao desenvolvimento das figuras geométricas criadas na fase anterior. Para isso alteramos a estrutura XML, permitindo assim efetuar diversas transformações geométricas, tais como, translações, rotações e alteração de escala no referencial. Estas mudanças permitirão criar modelos mais complexos e realistas.

2 Arquitetura do Projeto

Nesta fase continuamos a utilizar duas aplicações principais: o Generator e o Engine. No entanto, devido às mudanças na estrutura XML, foi necessária a alteração dessas aplicações e também surgiu a necessidade de criar novas classes.

2.1 Generator

2.1.1 *generator.cpp*

É no **generator.cpp** que são determinadas as estruturas para cada uma das formas geométricas a representar, com o objetivo de gerar os vértices das figuras.

2.2 Engine (Carregamento dos Modelos)

Em relação à *engine* adaptamos a forma como os ficheiros *XML* são lidos. Nesta fase os ficheiros *XML* também possuem as transformações geométricas aplicadas ao modelos. Para conseguir guardar todos os modelos decidimos utilizar a classe *Group*, que é capaz de guardar todas as transformações, os modelos e também outros grupos que são relativos a outro grupo.

2.2.1 *engine.cpp*

Neste ficheiro, para implementar as modificações mencionadas acima, foi introduzida uma variável chamada *groupList*, que armazena todos os grupos encontrados no arquivo *XML* carregado. Além disso, a função de análise do arquivo *XML* também foi ajustada para interpretar corretamente a nova estrutura do arquivo.

2.3 Engine (Câmara)

Devido à complexidade do cenário exigido nesta fase do projeto, percebemos a importância de oferecer uma visualização mais clara e detalhada. Com esse objetivo, optamos por implementar uma câmara que foca no Sol (neste caso a origem).

3 Estrutura e *Parsing* do Ficheiro *XML*

3.1 Estrutura

O ficheiro *XML* contém vários elementos *group*. Cada *group* é composto por:

- ***Transform***

  O *Transform* inicializa diversas transformações dentro do seu grupo, como por exemplo:

  - ***translate***

    Este elemento apresenta três atributos: X, Y e Z. Estes atributos representam a translação dos modelos em relação ao eixo correspondente.

  - ***rotate***

    Este elemento possui quatro atributos: *ang*, X, Y e Z. A rotação, em graus, em torno do eixo formado pelos atributos X, Y e Z é representada pelo atributo *ang*.

  - ***scale***

    Este elemento é representado por três variáveis: X, Y e Z. Estes influenciam a escala dos modelos em relação ao eixo.

  - ***color***

    Este elemento é composto por três atributos: R, G e B. O atributo R influencia a instensidade da cor vermelha, o G altera a intensidade da cor verde e o B a cor azul. Usar cores diferentes no nosso projeto facilita diferenciar os diferentes planetas e luas no Sistema Solar.

- ***Models***

  - ***Model***

    Possui o nome do ficheiro que é destinado a ser processado, passado para o programa através do argumento *file*.

- ***Group***

  group permite inserir um grupo dentro de outro. Permite a existência de uma hierarquia entre objetos.

3.2 *Parsing*

Tal como na primeira fase do projeto usamos a biblioteca ***TinyXML2***. Para começar abrimos o ficheiro e obtemos o elemento *world*, depois percorremos as definições da janela e da câmera, e os vários elementos *group*, sendo que estes últimos guardamo-los num *vector* geral após serem tratados.

3.2.1 *Parsing* de um elemento *group*

Para cada grupo encontramos cada um dos elementos acima referidos.

1. **Elemento *translate***

   Em relação ao *translate* obtemos os seus atributos. Caso não seja encontrado um atributo assume-se o valor predefinido *0*. No final, a translação é adicionada ao grupo correspondente. Caso o elemento não seja encontrado a translação é ignorada para o grupo.

2. **Elemento *rotate***

   Neste elemento, obtemos também os seus atributos e assume-se o valor *0* caso não seja encontrado o valor. Depois adicionamos a rotação ao respetivo grupo. Caso o elemento não seja encontrado a rotação é ignorada para o grupo.

3. **Elemento *scale***

   Neste elemento, obtemos os seus atributos e assume-se o valor *0* caso não seja encontrado o valor. Por fim adicionamos o escalamento ao grupo em questão. Caso o elemento não seja encontrado o escalamento é ignorado para o grupo.

4. **Elemento *color***

   Neste elemento, obtemos também os seus atributos e assume-se o valor *0* caso não seja encontrado o valor. Por fim adicionamos a cor ao respetivo grupo. Caso não seja encontrado o elemento, a cor definida com o valor predefinido é *R = 1, G=1 e B=1* correspondente ao branco.

5. ***models***

   Este elemento possui um conjunto de elementos *model*. O *model* é uma parte fundamental do nosso programa, composto por dois atributos importantes: *file* e *description*. O atributo *file* indica o nome do arquivo que contém os pontos que precisam de ser carregados.

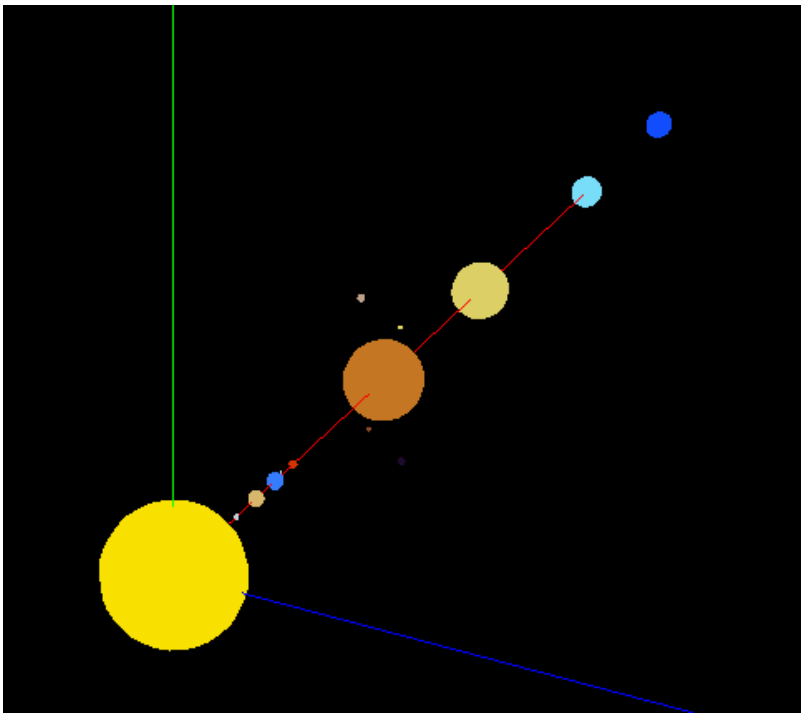
6. **Elemento *group***

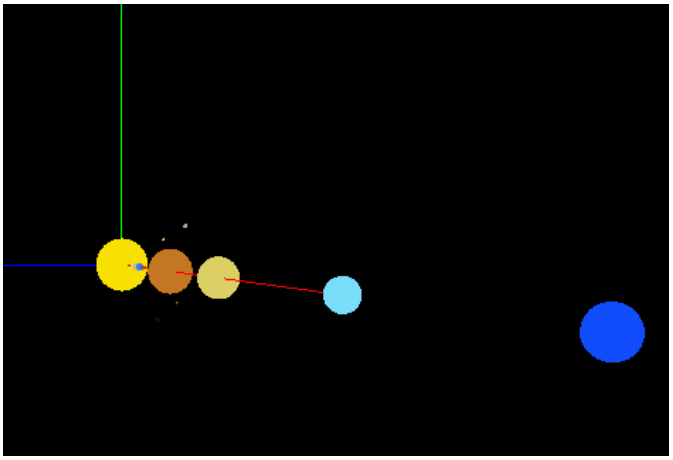
   Para este elemento usamos a recursividade invocando a função para conseguir processar todas as repetições.

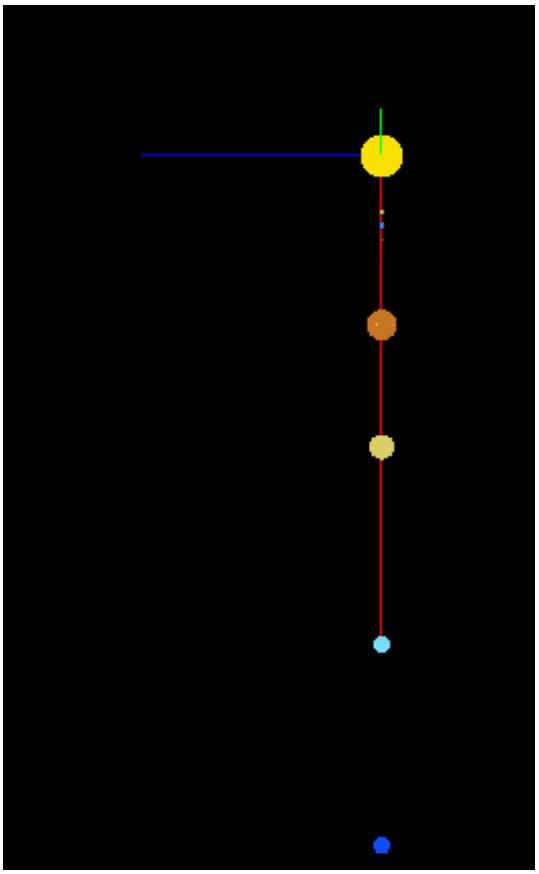
4 Construção do Cenário Final

Para começar criamos um modelo do Sistema Solar, composto pelo Sol e pelos planetas. Alinhamos os planetas ao longo do eixo Ox e o Sol na origem. Para isso, usamos uma operação de translação para posicionar cada planeta com o espaço adequado entre eles. Em termos de tamanho, todos os planetas foram baseados no mesmo ficheiro de pontos, foi necessário usar um escalamento para cada. Um aspeto a ter em conta é que o tamanho dos corpos celestes tal como a sua distância em relação ao Sol e entre eles não está feito à escala, porque seria muito difícil de observar o Sistema Solar dessa forma. Então decidimos colocar dimensões onde a representação do Sistema Solar era de fácil visualização mantendo sempre uma representação fidedigna do Sistema Solar. Após conseguirmos obter uma base sólida decidimos representar mais alguns planetas secundários, mais especificamente a Lua, quatro luas de Marte e as luas de Júpiter.

A seguir iremos mostrar algumas imagens que representam o estado final aquando desta fase do trabalho prático.







5 Aspetos a melhorar

Nas fases seguintes, pretendemos melhorar alguns aspetos que achamos fundamentais para o nosso projeto. Primeiramente, queremos rever e reformular os cálculos que usamos para formar o .xml, utilizado na construção do sistema solar. Posteriormente, pretendemos adicionar câmara livre que permitirá o utilizador navegar pelo espaço e observar todos os detalhes, e adicionar também uma forma de ao clicar num planeta, o programa mostrar ao utilizador informação relativa ao mesmo(picking). Por fim, pretendemos adicionar outra primitiva que permita criar as orbitas e os anéis dos planetas que os possuem(Júpiter, Saturno, Urano, Néptuno) e planetas anões.

6 Conclusão

Após a conclusão da segunda fase do projeto, podemos afirmar que consolidamos os nossos conhecimentos sobre as transformações geométricas que foram abordadas nas aulas. Além disso, adquirimos mais habilidade para trabalhar com a linguagem C++ e com o GLUT, à medida que fomos enfrentamos desafios e superamos obstáculos ao longo desta fase. Com o código organizado e estruturado, estamos confiantes que as próximas fases do projeto serão de um modo mais fácil de implementar pois temos uma base sólida para a implementação de novas funcionalidades. Estamos animados para continuar o nosso trabalho e esperamos adquirir mais conceitos e aprendizados valiosos ao longo do percurso.