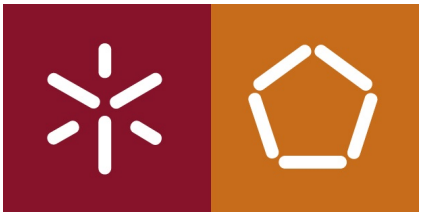

Universidade do Minho
Mestrado Integrado em Engenharia Informática
3ºano - 2º Semestre

Computação Gráfica

Relatório sobre a Fase 2

Grupo 47




a83732 – Gonçalo Rodrigues Pinto
a84197 – João Pedro Araújo Parente
a84829 – José Nuno Martins da Costa
a85059 – Diogo Paulo Lopes de Vasconcelos

29 de Março de 2020

Conteúdo

Lista de Figuras

1 Introdução

No 2º semestre do 3º ano do Curso de Engenharia Informática da Universidade do Minho, existe uma unidade curricular denominada por Computação Gráfica, que tem como objectivo ajudar os estudantes caracterizar as transformações geométricas e os referencias utilizados na computação gráfica, aplicar transformações para construção de modelos geométricos complexos e posicionamento da câmara, dar a conhecer algoritmos de iluminação local e global tais como Gouraud, Phong, Ray-tracing, Radiosity and Virtual Point Lights, têm também como meta que os estudantes apliquem texturas e definam coordenadas de textura, além de permitir uma análise de soluções do ponto de vista do desempenho recorrendo a profilers, utilizando apropriadamente soluções de eliminação de geometria, recorrendo a partição espacial e por fim aplicação de análise de algoritmos para geração de sombras

O presente trabalho pretendeu desenvolver um mecanismo 3D baseado em mini gráficos de cenas e fornecendo exemplos de uso que mostrem o seu potencial.

2 Contexto

Partindo do trabalho realizado na fase 1 seguindo abordagem de utilizar matrizes para representar os modelos que depois a aplicação, Generator, criada gera documentos com as informações do modelo (guardadas em bytes de forma a reduzir o espaço ocupado por dados num determinado ficheiro e ao mesmo tempo obter mais desempenho na leitura e escrita dos dados nos ficheiros).

Consequentemente criou-se uma outra aplicação, Engine, que recebe um documento de configuração, escrito em XML, que efectua a leitura do mesmo, ficando a conhecer todos os documentos com extensão .3d (criado pela aplicação referida anteriormente) que tem de representar, esta lê esses ficheiros que conhece efectuando a passagem de cada um para uma matriz igual a que a aplicação *generator* criou para escrever os pontos que gerou e posteriormente *engine* efectua a representação no OpenGL percorrendo essa matriz pela ordem correta.

3 Fase 2 - Transformações Geométricas

Nesta fase, foi nos requerido a criação de cenas hierárquicas usando transformações geométricas. Uma cena é definida como uma árvore em que cada nó contém um conjunto de transformações geométricas (translação, rotação e escala) e opcionalmente, um conjunto de modelos. Cada nó também pode ter nós filhos.

As transformações geométricas podem existir apenas dentro de um grupo e são aplicadas a todos os modelos e subgrupos. É de realçar que a ordem das transformações geométricas é relevante.

Para representar estas transformações efectua a adição de mais informação ao ficheiro XML que é recebido pela aplicação Engine, como se mostra na figura abaixo.

```
<scene>
    <group>
        <translate X=1 />
        <models>
            <model file="sphere.3d" />
        </models>
        <group>
            <translate Y=1 />
            <models>
                <model file="cone.3d" />
            </models>
        </group>
    </group>
</scene>
```

Figura 1: Exemplo de um ficheiro XML de configuração para esta fase.

3.1 Abordagem

Utilizando todas as classes previamente criadas decidimos apenas acrescentar/alterar a forma como o aplicação Engine efectua a leitura do documento XML, deixando a parte referente à leitura do documento com extensão .3d e posteriormente representação em formato matricial da mesma forma procedida anteriormente.

De forma a realizar as transformações geométricas requeridas nesta fase com o documento de configuração XML idêntico ao da figura 1, criou-se/redefiniu-se estruturas de dados para armazenar a informação que os ficheiros XML possuem, são essas:

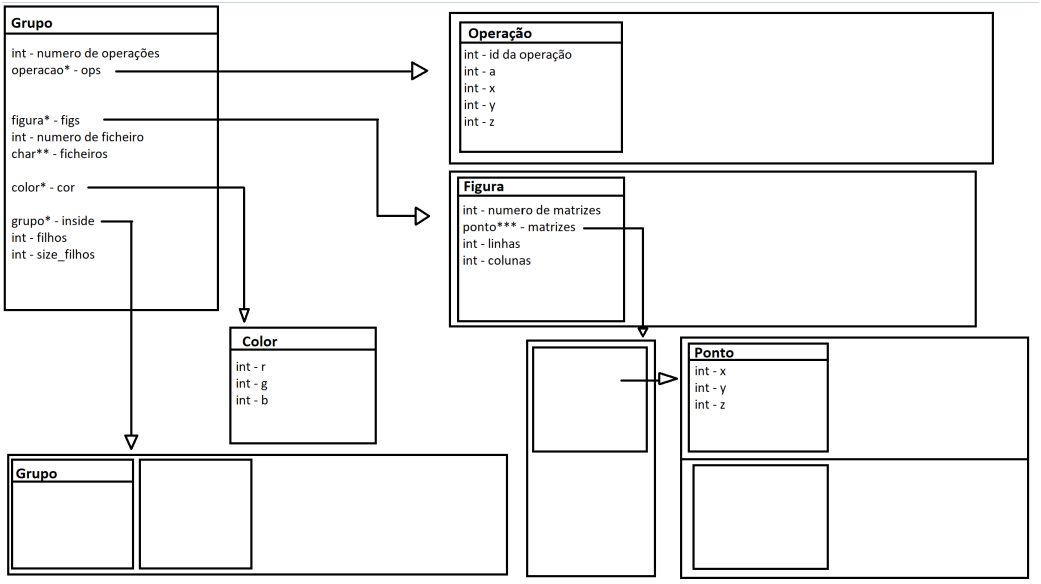

Figura 2: Esquema com as diferentes estruturas de dados utilizadas para armazenar a informação que os ficheiros XML possuem.

- **grupo**
  - Conjunto de estruturas de dados Operação designada de *ops*;
    - Identificador da Transformação Geométrica (Rotação, Escala, Translação);
    - Um número decimal designado de a que representa o ângulo de rotação, em graus;
    - Um número decimal designado de x que representa a coordenada X do vector da transformação em causa;

    * Um número decimal designado de y que representa a coordenada Y do vector da transformação em causa;
    * Um número decimal designado de z que representa a coordenada Z do vector da transformação em causa;
- Número total de operações (*numero_op*);
- Conjunto de estruturas de dados figura designada de *figs*;
    * Número total de matrizes (*numero_matrizes)*;
    * Conjunto de estruturas de dados dimensão designada de *di*;
        · Número de linhas (*linhas*);
        · Número de colunas (*colunas*);
    * Array de uma matriz de pontos designada de *matrizes*;
        · Número decimal designado de x;
        · Número decimal designado de y;
        · Número decimal designado de z;
- Número de ficheiros (*numero_ficheiros*);
- Apontador para um conjunto de estruturas de dados do tipo char designado de *ficheiros* para guardar os diferentes modelos;
- Conjunto de estruturas de dados Color designada de *cor*;
    * Número inteiro designado de r que especifica o novo valor de vermelho para actual cor;
    * Número inteiro designado de g que especifica o novo valor de verde para actual cor;
    * Número inteiro designado de b que especifica o novo valor de azul para actual cor;
- Conjunto de estruturas de dados grupo designada de *inside* (permitindo que haja vários grupos encadeados);
- Número de filhos que este grupo tem (*filhos*);
- Total de filhos (*size_filhos*);

3.2 Aplicação

A aplicação Engine recebe um documento de configuração, escrito em XML, posteriormente efectua a função (*abrir_xml_file* em OurXML.cpp) que recebe a localização do documento XML como também uma estrutura de dados do tipo grupo para esta função preencher com informação.

Esta função utilizando o auxílio do TinyXML2 que é um analisador XML C++ simples, pequeno e eficiente que pode foi facilmente integrado nesta aplicação. Após ser feito a carregamento do documento para tinyxml2 efectuamos análise das diferentes marcas presentes no documento XML após ser encontrado a marca "scene"esta procura de seguida pela marca "group"alocando espaço de memória para o grupo que recebeu como parâmetro e indicando que ainda não foi utilizado nenhum espaço. Posteriormente entra num ciclo em busca da marca "group"onde efectua a alocação de espaço caso seja necessário como efectua sempre que encontre a marca, a função (*parseGrupo* em OurXML.cpp, recebendo actual marca onde se encontra no ficheiro XML e a estrutura de dados grupo) que efectua análise das marcas dentro de um "group"se encontrar uma marca como "translate", "rotate", "scale"efectua a criação de uma estrutura de dados operacao onde coloca o id da operação igual a marca que encontrou e coloca nos elementos (x,y,z) os atributos da marca correspondentes, caso necessário efectua alocação de espaço para a sequencia de operações. Caso encontre a marca "models"executa a operação (*parseModels* em OurXML.cpp, que recebe marca atual XML onde se encontra e um apontador para guardar os nomes dos ficheiros que encontrar) que foi feita na fase anterior de leitura dos nomes dos ficheiros presentes no documento XML. Por fim, o nosso grupo decidiu acrescentar uma tag adicional que permite analisar a cor de um dado grupo pode ter para isso se ao executar a análise das marcas XML encontramos a marca "color"efectua a criação da estrutura color onde preenche os atributos da estrutura com os elementos da marca correspondentes. No final de cada chamada recursiva efectua-se a ligação das diversas estruturas permitindo o encadeamento das diferenças marcas encontradas no XML.

Previamente antes de desenhar o documento recebido aplicação vai percorrer todos os números de grupos recebidos passando para a função *preencheListaGrupo* que recebe um grupo efectua a tradução do ficheiro em matriz através da aplicação de *lee3d* presente no Our3d.cpp caso este grupo tenha mais grupos dentro de si efectua a recursivamente a função em causa.

Por fim, na função *renderScene* na parte de desenhar os objectos percorre todos os grupos inocando a função *desenhoListaGrupo* com os diferentes grupos presentes, esta função vai efectuar as transformações geométricas presentes do conjunto do tipo operacao dentro deste grupo de seguida preenche com a cor correspondente, posteriormente vai utilizar a função previamente criada na fase 1 *desenha* que faz o desenho da matriz nas diferentes figuras e caso este grupo tenha mais grupos dentro de si efectua a recursivamente a função em causa.

3.3 Demo Scene

A cena de demonstração necessária para esta fase foi um modelo estático do sistema solar, incluindo o sol, planetas e luas definidos em uma hierarquia.

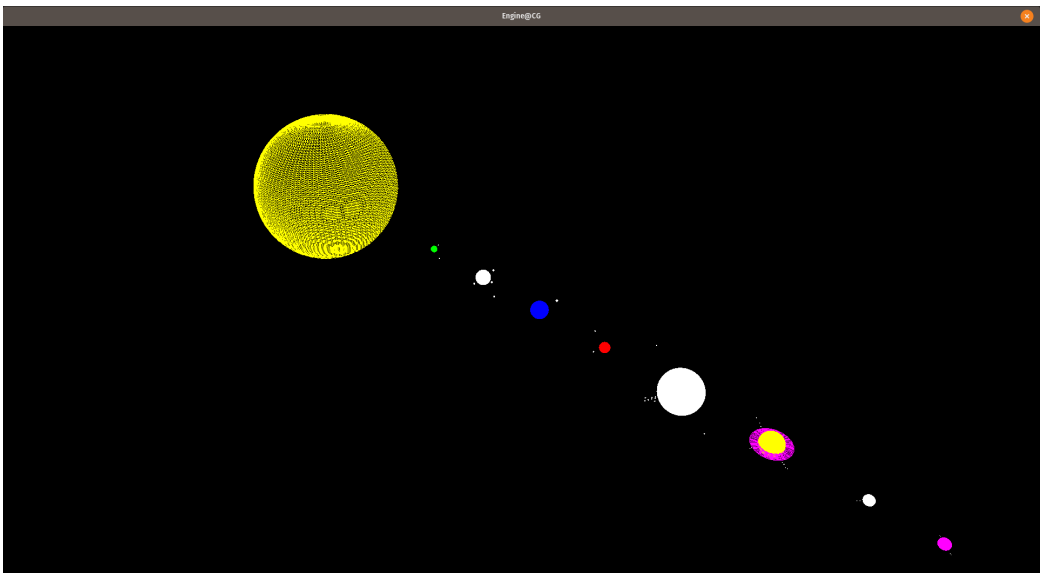

Figura 3: A nossa demonstração do Sistema Solar.

Para conseguirmos desenhar o Sol colocou-se no ficheiro XML que é fornecido à aplicação *Engine* uma marca *group*, onde o modelo a desenhar é uma esfera, sendo que antes de desenhar este modelo colocamos uma marca do tipo *scale* o que equivale a uma transformação geométrica de multiplicar a matriz actual por uma matriz de escala geral de modo a aumentar o tamanho desta, a designada Escala, e uma marca *color* permitindo que esta esfera possua a cor amarela. Para desenhar os diversos planetas e respectivas luas utilizou-se o encadeamento de marcas *group*, por exemplo para o planeta mais próximo do sol, Mercúrio, utilizou-se uma marca do tipo *translate* equivalente

a uma transformação geométrica do tipo Translação, de seguida uma marca *color* para colorir a esfera e uma marca *scale* para ajustar o seu tamanho e claro a marca que permite desenhar o modelo (esfera), sendo que tivemos em particular atenção a distância real de Mercúrio ao Sol para a transformação geométrica de deslocação, Translação, como também o raio real do Sol e do Mercúrio efectua a proporção entre eles através da transformação geométrica de ajustamento, Escala.

```
<scene>
    <group>
        <!--Sol-->
        <scale X="10" Y="10" Z="10"/>
        <color R="255" G="255" />
        <models>
            <model file="/home/jpedro/Desktop/CG_testes/3d/esfera.3d" />
        </models>

        <group>
            <translate X="15"/>
            <group>
                <!--Mercurio-->
                <color  G="255" />
                <translate X="10" />
                <scale X="0.035" Y="0.035" Z="0.035"/>
            <group>
                <models>
                    <model file="/home/jpedro/Desktop/CG_testes/3d/esfera.3d" />
                </models>
            </group>
```

Figura 4: Representação do Sol e de Mercúrio no formato XML.

4 Conclusão

O presente relatório descreveu, de forma sucinta, a resolução desta fase mediante os requisitos apresentados.

Consideramos que os principais objectivos foram cumpridos.

Sentimos que a realização deste projecto consolidou os nossos conhecimentos em ferramentas associadas à computação gráfica como o OpenGL e o GLUT, permitiu adquirir conhecimentos da linguagem C++, essencial para o desenvolvimento desta fase e conseguimos obter uma noção dos algoritmos que estão associados à criação de algumas primitivas gráficas.

Em suma, esperamos que o trabalho realizado até ao momento seja essencial e fulcral para as fases seguintes do projecto.