Computação Gráfica (3º ano de MIEI)

Segunda Fase

Relatório de Desenvolvimento

André Gonçalves (a80368)

João Queirós (a82422)

Luís Alves (a80165)

Rafaela Rodrigues (a80516)

25 de Março de 2019

Resumo

Este relatório inicia-se com uma breve contextualização, seguindo-se a descrição dos problemas propostos e o que deve ser desenvolvido para os solucionar. Segue-se a exposição da arquitetura da solução, sendo descritas as diversas estruturas de dados utilizadas. De seguida é explanado o processamento do ficheiro de configuração em XML e algumas das decisões tomadas para a sua leitura e interpretação. Termina-se a descrição da arquitetura com o processo de renderização das cenas carregadas e a exemplificação desse processo, com um modelo do Sistema Solar. Por fim, são apresentadas conclusões sobre o trabalho desenvolvido.

Conteúdo

1 Introdução **4**
1.1 Contextualização 4
1.2 Objetivos e Trabalho Proposto 4
1.3 Resumo do trabalho a desenvolver 4

2 Arquitetura da Solução **5**
2.1 Estruturas de Dados 5
2.1.1 Modelo 5
2.1.2 Grupo 5
2.1.3 Operação3f 6
2.1.4 Translação 6
2.1.5 Rotação 6
2.1.6 Escala 6
2.2 Processamento de ficheiro XML 6
2.2.1 Grupos 6
2.2.2 Modelos 8
2.2.3 Transformações Geométricas 8
2.2.4 Otimizações e Decisões Tomadas 9
2.3 Renderização 9
2.4 Sistema Solar 10

3 Conclusão **13**

Lista de Figuras

2.1 Ficheiro XML 7
2.2 Sistema Solar. 11
2.3 Planetas telúricos 12
2.4 Saturno (incluindo anel e uma lua) 12

Lista de Tabelas

2.1 *Map* de sequência de transformações 6
2.2 Diâmetro e órbita em escala 11
2.3 Diâmetro e órbita em escala das luas 11

Capítulo 1

Introdução

1.1 Contextualização

O presente relatório foi elaborado no âmbito da Segunda Fase do Trabalho Prático da Unidade Curricular de Computação Gráfica, que se insere no 2º semestre do 3º ano do primeiro ciclo de estudos do Mestrado Integrado em Engenharia Informática.

1.2 Objetivos e Trabalho Proposto

Pretende-se com este relatório formalizar toda a análise e modelação envolvida na construção da solução ao problema proposto, incluindo estruturas de dados e algoritmos utilizados.
Nesta segunda fase, foi proposto que fosse processado um ficheiro XML com uma hierarquia de transformações geométricas e modelos de forma a que o motor gráfico renderize uma determinada cena.

1.3 Resumo do trabalho a desenvolver

De forma a poder renderizar cenas customizadas de acordo com um ficheiro XML, será necessário atualizar o motor gráfico desenvolvido anteriormente em duas vertentes: a sua capacidade de processamento de ficheiros XML, e a estrutura de dados utilizada para armazenar informações necessárias para renderizar as cenas.

Capítulo 2

Arquitetura da Solução

Para esta fase do trabalho prático, apenas foi necessário alterar o programa *engine*, sendo que este pode ser visto como tendo dois componentes base:

- Leitura e processamento de ficheiros de configuração XML;
- Renderização das cenas resultantes.

Estes dois componentes são detalhados nas secções 2.2 e 2.3.

2.1 Estruturas de Dados

Para armazenar a informação proveniente do ficheiro XML, foram definidas 7 classes, estando 4 delas relacionadas com as transformações geométricas. Como a classe **Vértice** se manteve inalterada em relação à primeira fase, a sua especificação é aqui omitida.

2.1.1 Modelo

Um Modelo, em relação à fase anterior, passou a conter uma cor associada, de forma a ser possível distinguir primitivas que têm por base o mesmo ficheiro .3d. Tem por isso um vetor de Vértices, um nome e uma cor.

2.1.2 Grupo

Um Grupo é um objeto que contém as seguintes estruturas:

- Um vetor de **Modelos**
- Um vetor de **Grupos**
- 3 transformações geométricas (Escala, Rotação e Translação)

Uma vez que a ordem das transformações geométricas é relevante para o resultado final da renderização do grupo, foi criado um *Map* que associa uma ordem na fila à transformação geométrica. Por exemplo, se a ordem das transformações for: `R->E->T`, o *Map* correspondente será o que se encontra na tabela 2.1.
Assim sendo, o conjunto de transformações de cada grupo será aplicado aos modelos e grupos armazenados em cada Grupo.

Chave	Valor
1	”rotação”
2	”escala”
3	”translação”

Tabela 2.1: *Map* de sequência de transformações

2.1.3 Operação3f

Uma Operação3f é um objeto que contém 3 *floats*, algo que é transversal às 3 transformações geométricas, sendo que apenas a rotação requer mais 1 *float* para ser definida.

Por isso, tem as operações habituais de *get/set* desses 3 valores, sendo que este objeto é concretizado e usado através das classes **Translação**, **Rotação** e **Escala**.

2.1.4 Translação

Uma Translação é um objeto que extende o significado de um Operação3f (ver 2.1.3). Tem 0 como valor por defeito para as 3 componentes (em X, Y e Z), já que efetuar uma translação de 0 em qualquer um dos eixos é uma transformação neutra.

2.1.5 Rotação

Uma Rotação é um objeto que extende o significado de um Operação3f (ver 2.1.3). No entanto, adicionar um outro *float*, que representa o ângulo de rotação. Os 3 valores contidos na Operação3f, são os valores que definem o eixo sobre o qual é efetuada a rotação. Tem 0 como valor por defeito para as 4 componentes, uma vez que efetuar uma rotação de 0 graus sobre um vetor nulo é uma transformação nula.

2.1.6 Escala

Uma Escala é um objeto que extende o significado de um Operação3f (ver 2.1.3). Tem 1 como valor por defeito para as 3 componentes (em X, Y e Z), uma vez que efetuar uma escala de 1 sobre qualquer um dos eixos é uma transformação neutra.

2.2 Processamento de ficheiro XML

Para processarmos os ficheiros de configuração XML, recorremos à API do *tinyXML*, que permite a leitura ordenada em árvore do ficheiro.

Assim sendo, em cada árvore XML é possível encontrar as folhas, que se encontram nas subsecções 2.2.2, 2.2.3 e 2.2.1.

2.2.1 Grupos

Para cada Grupo encontrado no ficheiro XML, é efetuado o seguinte algoritmo para ser processado:

```
tag XML = primeiro filho do Grupo;

enquanto tag válida
```

```
</group>
<!--Saturno-->
<group>
    <translate X="2050.4345"/>
    <group>
      <scale X="167.3" Y="167.3" Z="167.3"/>
      <group>
          <models>
              <model file="sphere.3d" color="saturn"/>
          </models>
      </group>
      <!--Anel de Saturno-->
      <group>
          <scale X="1.7" Y="1.7" Z="1.7"/>
          <models>
              <model file="disc.3d" color="white"/>
          </models>
      </group>
    </group>
    <!--Lua de Saturno (Titan) -->
    <group>
      <translate X="128.46" Y="128.46"/>
      <scale X="7.353" Y="7.353" Z="7.353"/>
      <models>
          <model file="sphere.3d"/>
      </models>
    </group>
</group>
<!--Urano-->
<group>
```

Figura 2.1: Ficheiro XML

```
se tag == translação
    processaTranslação(grupo,tag);

senão se tag == rotação
    processaRotação(grupo,tag);

senão se tag == escala
    processaEscala(grupo,tag);

senão se tag = modelos
    processaModelos(grupo,tag,modelos);

senão se tag = grupo
    folha = novo Grupo;
    processaGrupo(folha,tag,modelos);
    adicionaGrupo(grupo,folha);

tag = tag->irmão;
```

Tendo em conta que um Grupo pode ter vários filhos, estes são processados com as respetivas transformações e modelos e são posteriormente adicionados ao Grupo pai através de um método *adicionaGrupo*.

2.2.2 Modelos

Para cada Modelos encontrado no ficheiro XML, é efetuado o seguinte algoritmo para ser processado:

```
tag = primeiro filho do Modelos;

enquanto tag válida

    nome do ficheiro = tag->atributo("file");

    se !(nome do ficheiro)
        erro;

    se (map de modelos) contém (nome do ficheiro)
        modelo = map.get(nome do ficheiro);

    senão
        cor = tag->atributo("color");

        se cor inválida
            cor = branco;

        modelo = novo Modelo (nome do ficheiro, cor);
        processa3D(nome do ficheiro, modelo);
        map.insere(nome do ficheiro, modelo);

    adicionaModelo(grupoPai,modelo);

tag = tag->irmão;
```

Assume-se por defeito que caso um Modelo não venha acompanhado de uma tag *color*, a cor por defeito será a cor branca. No processamento dos modelos é consultado um *Map* que tem como chaves o nome dos ficheiros .3d que contêm os vértices do Modelo, e no valor o respetivo Modelo já com os vértices processados e armazenados. Assim, evita-se estar a ler repetidamente o mesmo ficheiro, em situações em que é referenciado em modelos distintos.
Em relação à Primeira Fase, também se alterou o método utilizado para ler cada linha de coordenadas do ficheiro .3d, uma vez que a função utilizada (*getline()*) exigia *POSIX compliance*. Passou a utilizar-se uma série de métodos equivalentes das bibliotecas `sstream` e `fstream`.

2.2.3 Transformações Geométricas

Existem 3 possíveis Transformações Geométricas no ficheiro de configuração: **Translação**, **Rotação** e **Escala**. Por cada **Grupo**, poderá existir no máximo uma ocorrência de cada uma destas transformações. Caso exista mais do que uma, o processamento é abortado.
É descrito de seguida o algoritmo efetuado para processar transformações geométricas:

```
atribui valores default a:
```

```
    x, y, z

transformacao->x = tag->atributo("X");
transformacao->y = tag->atributo("Y");
transformacao->z = tag->atributo("Z");

transformacao = transformacao->irmão("transformacao");

se transformacao
    erro;
senão
    adicionaTransformação(transformacao,x,y,z);
```

Tanto a **Escala** como a **Translação** seguem de muito perto o algoritmo supra mencionado, sendo que apenas a **Rotação** apresenta um comportamento diferente, uma vez que o nome que define os atributos é precedido de *axis* e contém mais um atributo: ângulo de rotação. Para os valores por defeito atribuídos a cada um dos componentes, ver as respetivas secções (2.1.4, 2.1.5 e 2.1.6).

2.2.4 Otimizações e Decisões Tomadas

Aqui se listam as decisões tomadas que visam simplificar o processo de leitura e processamento dos ficheiros XML, bem como acelerá-lo:

1. A folha inicial/raiz é sempre uma `<scene>`, sendo que qualquer folha descendente dela que seja algo que não um Grupo (nomeadamente um Modelo ou Transformação Geométrica), é ignorada pelo processador do ficheiro.
2. Qualquer erro gerado pelo *tinyXML* aborta o processamento do ficheiro XML.
3. Valores inválidos nos atributos das transformações resultam em transformações identidade nesse(s) atributo(s).
4. Cada modelo carregado é armazenado num *Map*, sendo a chave o nome do ficheiro .3d e o valor o objeto Modelo.
5. Dentro de tags `<models>`, qualquer tag que não seja `<model>` é ignorada pelo processador do ficheiro.
6. As transformações geométricas dentro de um `<group>` podem localizar-se em qualquer lugar, desde que exista apenas 1 por tipo.
7. A ordem das transformações aplicadas aos modelos de um Grupo é igual à ordem pelas quais surgem no ficheiro de configuração, sendo aplicadas a todos os grupos filho.

2.3 Renderização

Para o processo de Renderização, foram efetuadas algumas alterações relativamente ao que havia sido implementado na fase anterior.
A principal é a utilização de um vetor de Grupos, ao invés de um vetor de Modelos. Assim, para cada **Grupo**, é efetuado o seguinte algoritmo para ser renderizado:

```
pushMatrix;

modelos = grupo->modelos;
filhos = grupo->grupos;

de 0 a 2:
    transformacao = obtemNésimaTransformação(grupo);

    se transformacao == translação
        glTranslate(translação->x, translação->y, translação->z);

    senão se transformacao == rotação
        glRotate(rotação->ângulo, rotação->x, rotação->y, rotação->z);

    senão se transformacao == escala
        glScale(escala->x, escala->y, escala->z);

se modelos
    para cada modelo:
        desenhaModelo;

se filhos
    para cada filho:
        desenhaGrupo;

popMatrix;
```

Para cada um dos Grupos é efetuado um `push/popMatrix` no início e fim do mesmo uma vez que se pretende que as transformações efetuadas em cada Grupo estejam limitadas a esse Grupo e respetivos filhos.

Relativamente à obtenção da N-ésima transformação de cada Grupo, esta é feita com recurso ao *Map* que mantém a sequência desejada de transformações, de acordo com o ficheiro de configuração fornecido (ver 2.1.2).

Quanto ao desenhar cada Modelo, tal como na fase anterior, o desenho é feito através do desenho de triângulos, interpretados a partir da sequência de vértices de cada Modelo. Incluiu-se nesta fase uma série de constantes que definem quais os valores RGB associados a cada planeta, sendo que um Modelo com cor "uranus"corresponde o valor em RGB (0.8, 0.8, 1), numa escala de 0 a 1.

2.4 Sistema Solar

De forma a obtermos os valores mais próximos da realidade possível, recorremos a um *website* que, dada uma escala para o Sol, disponibilizava os diâmetros e órbitas dos planetas em relação a essa escala [1].

Assim, foram obtidos os seguintes valores para os diversos planetas, tendo como base o Sol com diâmetro 2 metros (ver 2.2 e 2.3).

O grupo decidiu apenas desenhar 1 lua de cada planeta (se este tiver), devido ao grande número de astros que certos planetas possuem, como Júpiter e Saturno. Como tal, e para melhor visualização, foram escolhidas as luas com maior raio.

Planeta	Raio (mm)	Órbita (m)
Mercúrio	6.9	83.2650
Vénus	17.3	155.3415
Terra	18.3	214.9154
Marte	9.7	327.3795
Júpiter	205.4	1118.0975
Saturno	167.3	2050.4345
Urano	67.4	4124.20
Neptuno	65.2	6465.838
Plutão (we love you)	3.2	8496.3

Tabela 2.2: Diâmetro e órbita em escala

Planeta (Lua)	Raio (mm)	Órbita (m)
Terra (Lua)	4.922	23.4428
Marte (Deimos)	0.014	9.85
Júpiter (Ganymede)	7.583	219.135
Saturno (Titan)	7.353	181.673
Urano (Titania)	1.436	71.356
Neptuno (Triton)	5.768	72.996
Plutão (Charon)	2.154	5.454

Tabela 2.3: Diâmetro e órbita em escala das luas

No entanto, se representássemos estes valores à escala real, seria impercetível a visualização total do sistema solar no ecrã. Por isso, optamos por manter as relações entre os raios iguais à escala real, e multiplicar as órbitas por 10^3.

Para além disso, visto que a escala entre a lua e o respetivo planeta é muito reduzida, decidiu-se multiplicar o raio da órbita da lua por 4 e somar tanto o raio do planeta como do astro.

Para posicionar as luas, foi tida em consideração a órbita de cada planeta e o ângulo de 45º, que nos forneceu as coordenadas em X e Y (mantivemos o Z fixo para todos).

Tanto os planetas como os astros (incluindo o sol), foram desenhados com a opção G_LINE, para uma melhor visualização dos mesmos.

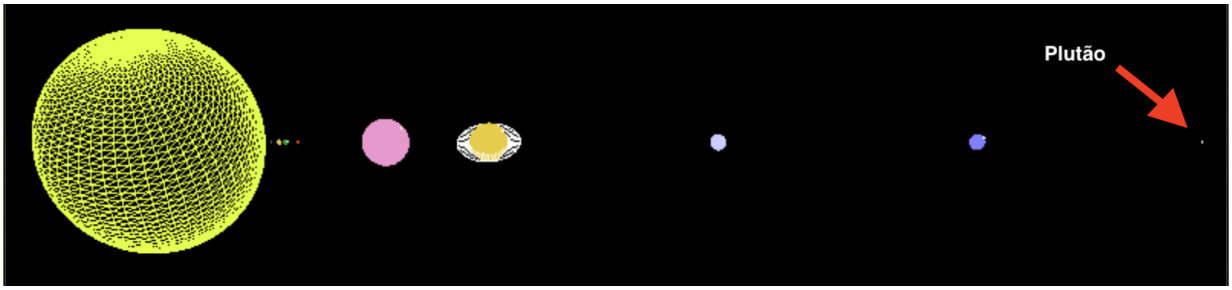

Figura 2.2: Sistema Solar.

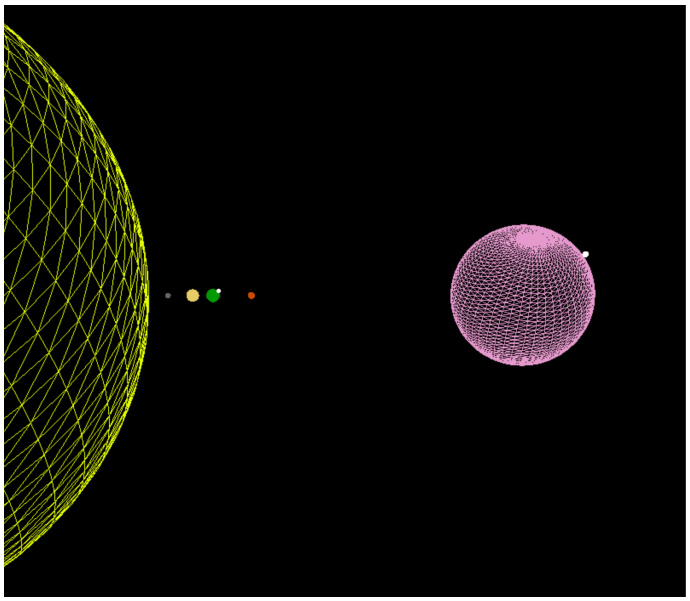

Figura 2.3: Planetas telúricos

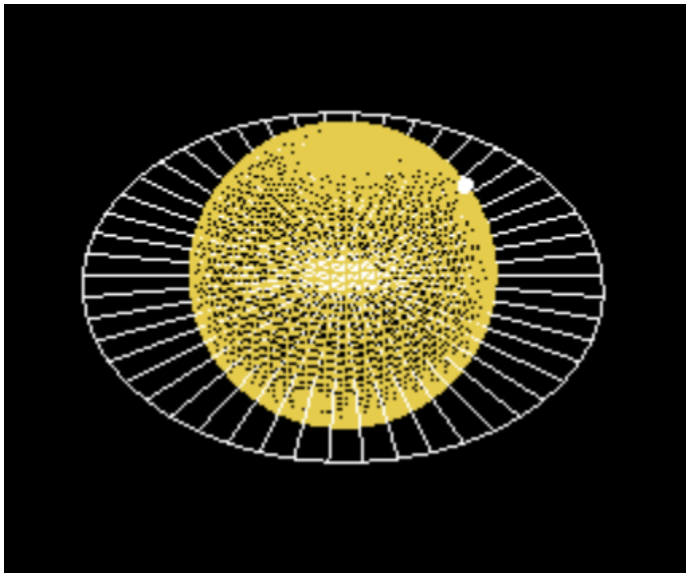

Figura 2.4: Saturno (incluindo anel e uma lua)

Capítulo 3

Conclusão

Com este trabalho pudemos aprofundar o processamento do ficheiro de configuração XML. Permitiu que aplicássemos transformações geométricas a modelos gerados pela aplicação desenvolvida na fase anterior e assim produzir cenas mais interessantes, como um sistema solar.

Foi possível também evitar o carregamento múltiplo do mesmo ficheiro .3d, o que produz melhorias a nível de desempenho no carregamento do ficheiro de configuração.

Conclui-se portanto que os objetivos definidos para esta fase foram cumpridos na íntegra.

Bibliografia

[1] Ron Hipschman. Make a scale model of the solar system and learn the real definition of "space.", 1997.