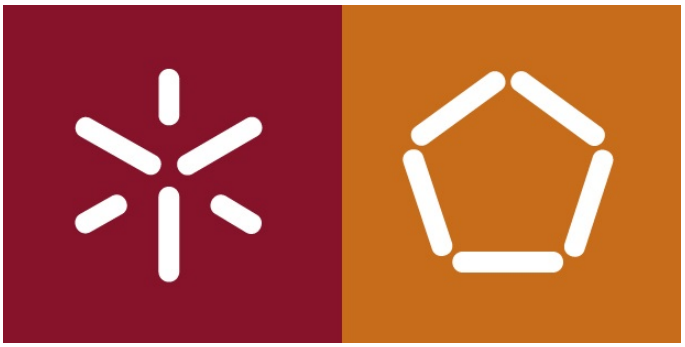


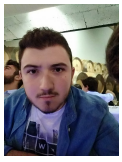
Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Computação Gráfica

Relatório do Projeto Prático - Parte 3

Autores :
Diana Costa (A78985)

José Oliveira (A78806)

Marcos Pereira(A79116)

Vitor Castro(A77870)


29 de Abril de 2018

Conteúdo

1 Introdução

Este relatório refere à terceira fase da unidade curricular de Computação Gráfica, que visa a construção de um sistema solar dinâmico.

O *generator* da fase anterior será modificado para produzir triângulos com base em *Bezier patches*. Serão também usadas curvas de *Catmull-Rom*, extendendo os elementos *translate* e *rotate*, com uma determinada duração. O *engine* processará o ficheiro, fazendo a leitura e desenho usando VBOs.

Assim, este relatório explicará, através de figuras e excertos de código, o processo realizado. Na Secção 2 será apresentas as novas funcionalidades do *Generator*. Na Secção 3 as modificações relativas ao *Engine*. Por último, será analisado o formato de leitura e processamento dos dados, na Secção 4.

2 Generator

Nesta terceira fase, era requerido que se desse movimento ao molde do sistema solar produzido na fase anterior. Para isto, foi necessário especificar um ficheiro XML, e alterar o *Engine* da fase anterior do trabalho prático.

O ficheiro XML é onde o grupo especifica a duração das translações dos diferentes objetos do sistema, bem como o ponto inicial de cada uma das curvas de *Catmull-Rom*. Para isso, o *generator* receberá como parâmetros o ficheiro (*patch*) que contém os pontos de controlo de Bezier e nível de *tesselation*, gerando um ficheiro que segue a seguinte estrutura:

- Sol;
  - Planeta;
  - Translação, rotação, escala;
    - Lua;
    - Translação, rotação, escala;

2.1 Ficheiro *patch*

A leitura do ficheiro *patch* segue a seguinte ordem, para uma operação bem sucedida:

- Abrir ficheiro;
- Lê o número total de *patches* indicado e regista o valor;
- Ler *patch* um a um;
- Lê o número de pontos de controlo e regista o valor;
- Ler as coordenadas de cada ponto de controlo e armazenar as mesmas.

2.2 Criação de superfície de *Bezier*

O algoritmo usado para a criação de cada uma das superfícies de *Bezier* é o seguinte:

- Iterar sobre os *patches* e respetivas *tesselations*;
- Calcular o primeiro ponto, avançando em uma unidade (inferior ao *tesselation* máximo); Calcular o segundo ponto, tendo em conta o *tesselation* atual; Calcular os pontos com base nos calculados neste passo;
- Armazenar os pontos calculados num ficheiro.

2.3 Cálculo dos pontos de *Bezier*

O algoritmo usado para o cálculo de um ponto de *Bezier* é o seguinte:

- Criar array de polinomiais de *Bernstein*, para os pontos u e v;
- Iterar pelo polinómio sobre u e, consequentemente, por v, de acordo com o *patch* atual.
- Multiplicar as coordenadas do ponto de controlo pelos array definido anteriormente.

3 Engine

O *Engine*, que tinha como objetivo ler um ficheiro XML, que continha as referências e coordenadas geradas com o *Generator* e dar como output o modelo do sistema solar, vê-se agora alterado para ler o novo ficheiro e dar como resultado o sistema solar com movimento e através de VBOs.

A organização do XML é idêntica à anterior, seguindo-se agora um conjunto de explicações que visam esclarecer as ligeiras diferenças.

3.1 Estruturas de Dados

A árvore que representava a organização do ficheiro XML é idêntica, tendo agora adicionados os elementos de **rotação**, **translação** e **escala**.

O primeiro nó da árvore é o Sol, e os respetivos filhos são os planetas, que por sua vez têm como filhos os seus satélites naturais. Entre o quarto e o quinto planeta tem-se a cintura de asteroides que é também um "filho"do Sol. Acresce agora um cometa que faz translação relativamente ao Sol.

A implementação do desenho anterior é feita com auxílio de um conjunto de *structs*, apresentadas de seguida:

3.1.1 Scale

```
struct Scale {

        bool vazio = true;
        float x, y, z;

};
```

A *struct* Scale contém as coordenadas que permitem verificar o escalamento de uma figura. O booleano serve para informar se a figura vai ser escalada ou não. O acesso vai ser direto e por isso as variáveis são públicas.

3.1.2 Rotation

```
struct Rotation {

    bool vazio = true;
    float x, y, z, angulo = -1, tempo = -1;

};
```

A *struct* Rotation contém também as coordenadas, tendo o *angulo* e o *tempo* para representar, dependendo do necessário, o período de rotação de um astro.

3.1.3 Translate

```
struct Translate {

    bool vazio = true;
    float tempo;
    float** pontos;
    int numPontos;
    float** matrix;

};
```

A *struct* Translate contém o tempo destinado à translação de um astro em torno do seu referencial, sendo o caminho definido pela matriz *pontos*. A matriz *matrix* é necessária para calcular as transformações ao longo do movimento.

3.1.4 Figure

```
struct Figure {

    string nome;
    int vboIndice = -1, triangulos = 0;
    Scale escala;
    Rotation rotacao;
    Translate translacao;

};
```

A *struct* Figure associa a rotação, translação e escala definida para o modelo a desenhar, cujo nome estará representado em *nome*. Os campos *vboIndice* e *triangulos* servem para gerir questões de memória.

3.1.5 Tree

A estrutura principal da segunda fase, denominada de *modelTree* foi abandonada, dando lugar às anteriores definições. Como elemento unificador, existe agora a *Tree*, que terá a *Figure* mãe e os filhos correspondentes.

```
struct Tree {

        Figure modelo;
        std::vector<Tree> subArvores;

};
```

Estas árvores são formadas na leitura do XML, explicitada de seguida.

4 Leitura e representação

4.1 Ficheiro XML

O ficheiro XML criado tem como o objetivo representar o sistema solar proposto, contendo os respetivos planetas e satélites naturais, bem como a duração da translação e rotação de cada um deles. É definido também o ponto inicial à trajetória, para as curvas de *Catmull-Rom*. A *scene* é composta por um *group* que contém as várias translações (e respetiva duração), rotações (e respetiva duração), escalas e ponto inicial da curva, a ser aplicadas à respetiva figura indicada na tag *model*. Cada *group* por sua vez pode ter outros *group* como filhos, herdando o estado atual.

Em baixo, segue-se um excerto do ficheiro XML criado pelo grupo.

```
<scene>
 <group a="Sol">
      <group>
                <rotate time="1" axisX="0" axisY="1" axisZ="0" />
                <scale X="1" Y="1" Z="1" />
                <models>
                        <model file="sol.3d" />
                </models>
     </group>
     <group a="Mercurio">
                <translate time ="3.5" X="1.1" Y="0" Z="0" >
                        <point X="0.283569" Y="0.000000" Z="0.688008" />
                        <point X="0.539929" Y="0.000000" Z="0.613315" />
                        <point X="0.763001" Y="0.000000" Z="0.510178" />
                        <point X="0.938760" Y="0.000000" Z="0.374079" />
                        <point X="1.047162" Y="0.000000" Z="0.216213" />
                        <point X="1.098829" Y="0.000000" Z="0.043854" />
                        <point X="1.080416" Y="0.000000" Z="-0.130166" />
                        <point X="0.994311" Y="0.000000" Z="-0.297044" />
                        <point X="0.837566" Y="0.000000" Z="-0.445196" />
                        <point X="0.636566" Y="0.000000" Z="-0.565311" />
                        <point X="0.403939" Y="0.000000" Z="-0.650743" />
                        <point X="0.136869" Y="0.000000" Z="-0.694380" />
                        <point X="-0.136864" Y="0.000000" Z="-0.694380" />
                        <point X="-0.403934" Y="0.000000" Z="-0.650744" />
                        <point X="-0.645561" Y="0.000000" Z="-0.565313" />
                        <point X="-0.846562" Y="0.000000" Z="-0.445198" />
                        <point X="-0.994308" Y="0.000000" Z="-0.297048" />
                        <point X="-1.070515" Y="0.000000" Z="-0.130170" />
                        <point X="-1.096830" Y="0.000000" Z="0.043850" />
                        <point X="-1.045164" Y="0.000000" Z="0.215309" />
                        <point X="-0.927764" Y="0.000000" Z="0.374076" />
                        <point X="-0.752006" Y="0.000000" Z="0.510175" />
                        <point X="-0.528935" Y="0.000000" Z="0.612413" />
                        <point X="-0.272565" Y="0.000000" Z="0.677007" />
                        <point X="-0.000007" Y="0.000000" Z="0.700000" />
                </translate>
                <rotate angle="-45" axisX="0" axisY="0" axisZ="1" />
                <group>
                        <rotate time="16" axisX="0" axisY="1" axisZ="0" />
                        <scale X="0.04" Y="0.04" Z="0.04" />
                        <models>
                                <model file="planeta.3d" />
```

```
                    </models>
            </group>
    </group>
(...)
```

Considera-se, assim, que todos os astros serão como "satélites"uns dos outros. Tem-se que o sol será a referência principal, que terá na sua orbita como satélites todos os planetas, que por sua vez terão, cada um, as suas luas em órbita.

4.2 Sistema de Leitura

Após serem feitas as alterações no ficheiro XML, foi necessário alterar o sistema de leitura já criado na fase anterior. Tendo em conta a necessidade de implementar VBOs, passa-se a explicar o fluxo de funcionamento de um caso normal.

- Abrir o ficheiro "scene.xml";
- Depois de encontrar o primeiro elemento, percorre esse mesmo através da função *getGroup*;
  - Cria nova árvore, com *Figure* sendo o elemento atual;
  - Itera pelas subarvores;
  - Em cada subarvore, caso identifique um *translate* preenche os campos respetivos daquela *Figure*. O mesmo para o *rotate* e para o *scale*.
- Retorna a árvore criada, com as respetivas subarvores.

4.3 Renderização

O *rendering* é agora ligeiramente mais complicado, uma vez que é necessário ter em conta os movimentos de translação e rotação dos astros, bem como o uso de VBOs.

Assim, o procedimento adotado é o seguinte, elemento a elemento da árvore criada anteriormente:

- Abrir com *glPushMatrix*;
- Verificar a existência de transação e, caso haja, efetuar as transições necessárias, usando a *matrix* de acordo com o tempo decorrido. Recorre a *glTranslatef*;
- Verificar a existência de rotação e, caso haja, efetuar a devida rotação em função do tempo. Recorre a *glRotatef*;
- Verificar a existência de escala e, caso haja, efetuar a mesma. Recorre a *glScalef*;
- Verificar o *vboIndex* da figura, de forma a representar esta. Usa de *glBindBuffer* e *glDrawArrays*;
- Fechar com *glPopMatrix*.

5 Debug

De maneira a verificar os resultados obtidos, foram adicionadas algumas funcionalidades à interface do programa. A tecla F1 foi definida para renderizar apenas as arestas das figuras, enquanto que a F2 define o modo de renderização que inclui apenas os pontos dos modelos. A tecla F3 representa-os de forma preenchida. As teclas W, A, S, D, Q, E e setas permitem visualizar o redor do modelo.

Com a ajuda destas funcionalidades foram descobertos alguns problemas ao longo do desenvolvimento do programa, com a dificuldade natural de um problema como este.

6 Conclusão

A terceira fase do projeto foi de complexidade muito superior à segunda, tendo o grupo feito um grande esforço conjunto para perceber como aplicar os conhecimentos. Esta serviu de elemento de consolidação de conhecimentos adquiridos nas aulas e também de aprendizagem, tendo o grupo dialogado com outros colegas exteriores ao mesmo, e discutido as melhores soluções para os projetos com quem partilhou opiniões.

Foram encontradas algumas dificuldades iniciais no preenchimento da *scene*, principalmente com quais os pontos das Catmull-Rom a colocar. Para isso foi desenvolvido um programa auxiliar, utilizando a base feita nas aulas práticas.

No desenho das imagens e das translações, foi necessário decidir como é que ia ser armazenada a informação e tornou-se complicado, por vezes, conseguir ser eficiente na utilização das estruturas definidas. De facto, a curva de aprendizagem foi grande.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu melhorar a capacidade de comunicação e entreajuda, para uma tarefa que, tendo sido concluída com sucesso, foi muito recompensadora.

7 Anexos

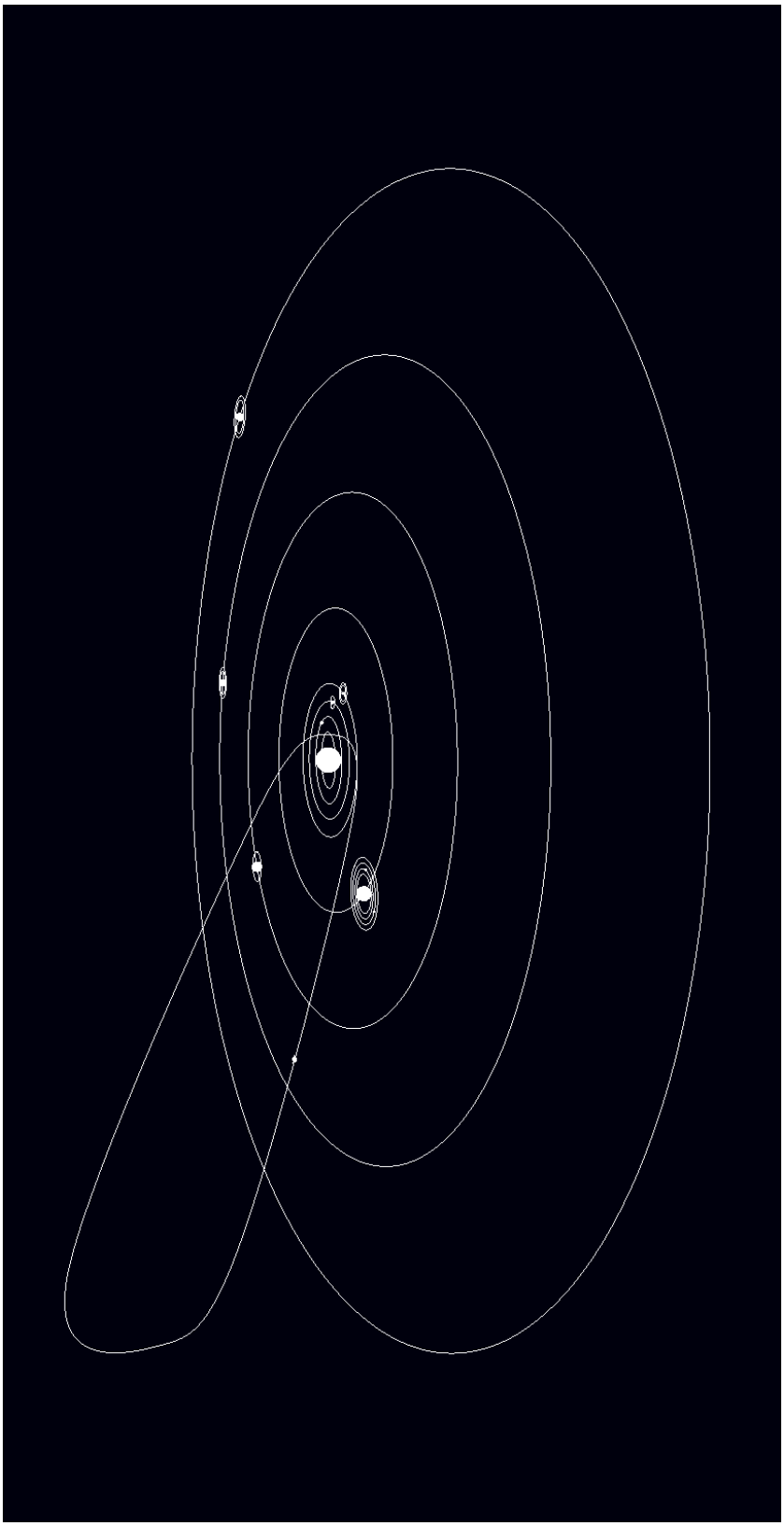


Figura 1: Sistema solar - Resultado final