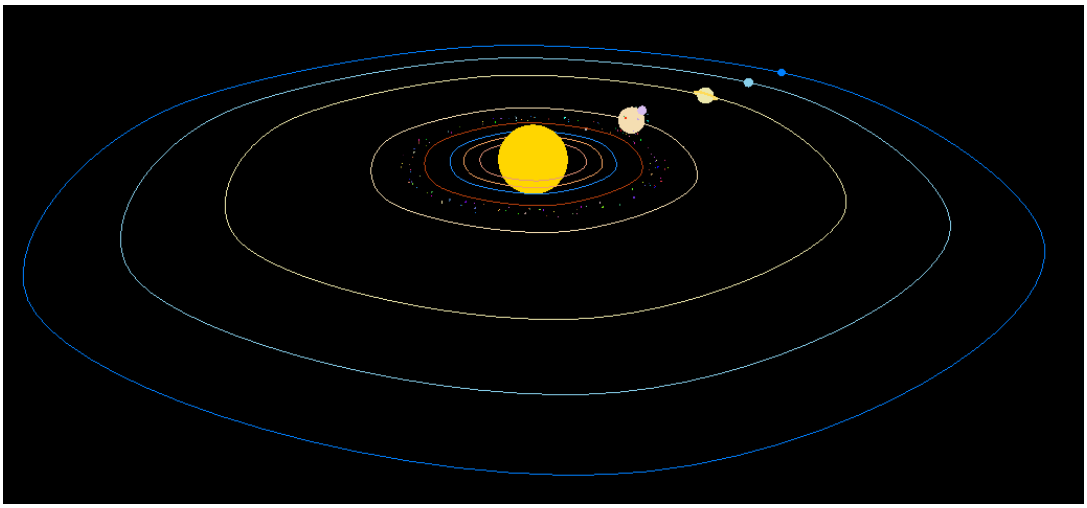
Universidade do Minho

Trabalho de Computação Gráfica

Fase 3



Grupo 7:
-João Ferreira, A50193;
- Luís Azevedo, A66704;
- Luís Albuquerque, A79010;
-Rafaela Pinho, A77293.
Docente: António José Borba Ramires Fernandes .
Ano Letivo: 2017/2018

Índice

Introdução

Nesta fase do projeto foram introduzidas novas funcionalidades nas translações e rotações dos objetos. Para além das translações e rotações “normais” passa agora a ser possível passar um parâmetro “tempo” de forma a definir uma curva de Catmull-Rom no caso das translações e quanto às rotações, o valor de “tempo” define o tempo que um objeto demora a rodar 360º graus. Esta nova capacidade é utilizada através do ficheiro XML, fornecendo para cada objeto quais as transformações necessárias para que ele se apresente no ambiente gráfico da forma pretendida.

Os modelos passam a ser desenhados através de VBOs em detrimento do modo imediato até agora utilizado.

Generator

Tal como na segunda fase o nosso *generator* continua a receber como argumentos os parâmetros necessários à criação das figuras geométricas e um nome, e cria um ficheiro de texto com o nome passado como parâmetro, onde guarda as coordenadas dos vários pontos que a figura necessita para ser gerada.

Continua a ser possível a criação das figuras da fase anterior.

Possibilidades de utilização:

- ./generator Plane x z [filename]
- ./generator Box x y z d [filename]
- ./generator Sphere radius slices stacks [filename]
- ./generator Cone radius height slices stacks [filename]
- ./generator Torus raioI raioE rings slices [filename]
- python3 Patch_Beizier.py 40 teapot.patch teapot.3d

Em que **x**,**y**,**z** são dimensões e **d** é o número de divisões.

Note-se que apenas são permitidos valores positivos nos parâmetros numéricos

Engine

Tal como nas fases anteriores, recebe um único parâmetro, o nome de um ficheiro com extensão *XML* previamente criado, onde estão os nomes dos ficheiros texto que contêm as coordenadas para criação das figuras geométricas bem como as possíveis transformações geométricas.

Neste caso temos o ficheiro ***config.xml*** e para correr o *engine* temos 2 hipóteses:

1. **make engi**
2. **./engine config.xml**

Nesta fase foram feitas algumas alterações ao nosso *engine*:

- As coordenadas dos pontos são agora guardadas através de um *map* que guarda os pontos por ordem, "coords":
map<int,vector<float>> coords.

O XML é lido, com recurso ao *parser* ***TinyXML***, e quando são encontrados nomes de ficheiros estes são abertos e as coordenadas contidas são guardadas no *map* através da função ***readCoord***.

- Relativamente à fase anterior foram adicionados os parâmetros ***time*** e ***points*** na estrutura das transformações geométricas.
A estrutura passa agora a conter também:

- **time:** Parâmetro que contém o valor necessário para as novas transformações. Pode ser utilizado nas translações ou rotações.
- **points*:*** Vetor de pontos a serem utilizados nas "novas" translações para a criação das curvas de Catmull-Rom.

Esta estrutura é preenchida recursivamente pela função ***parserXML***.

- À função ***execTransform*** foram adicionadas funcionalidades para a execução da translação utilizando curvas de Catmull-Rom e da rotação. Se as transformações tiverem o parâmetro "time" com valor zero é executada uma translação/rotação normal, caso contrário são utilizadas as curvas ou o cálculo do tempo da rotação.

- Ao *engine* foi também adicionada uma câmara em modo FPS para navegar no modelo. É possível alternar entre a utilização do rato ou do teclado para controlar a câmara pressionando a tecla 'c'.

Tal como na fase anterior, depois de os parâmetros e coordenadas necessários para as transformações e criação das figuras serem armazenados, respetivamente, na estrutura e no *map,* entramos na função ***renderScene*** através do ***GLUT***. Nela percorremos a nossa estrutura através de um ciclo.

Exemplos do translate e rotate

```
<group>
    <color R="1" G="0.86" B="0.35" />
    <translate time="538.525" >
    <point X = " -600.0 " Y = " 0.0 " Z = " 0.0 " />
    <point X = " -300.0 " Y = " 0.0 " Z = " 520.2 " />
    <point X = " 300.0 " Y = " 0.0 " Z = " 520.2 " />
    <point X = " 600.0 " Y = " 0.0 " Z = " 0.0 " />
    <point X = " 300.0 " Y = " 0.0 " Z = " -520.2 " />
    <point X = " -300.0 " Y = " 0.0 " Z = " -520.2 " />
</translate>
    <rotate angle="-60" axisX="1" />
    <models>
        <model file="ring.3d" />
    </models>
</group>
```

Figura 1- Exemplo de translate com tempo e rotate normal.

```
<group>
    <color R = "0.333224" G="0.492866" B="0.600152"/>
    <translate  X="35.751633" Y="20.000000" Z ="10.168388"/>
    <scale X="2.104796" Y="2.104796" Z ="2.104796"/>
```

Figura 2- Translate normal.

```
<group>
    <color R="0.91" G="0.59" B="0.48" />
    <translate time="29.3233" >
        <point X = " -109.0 " Y = " 0.0 " Z = " 0.0 " />
        <point X = " -54.5 " Y = " 0.0 " Z = " 94.503 " />
        <point X = " 54.5 " Y = " 0.0 " Z = " 94.503 " />
        <point X = " 109.0 " Y = " 0.0 " Z = " 0.0 " />
        <point X = " 54.5 " Y = " 0.0 " Z = " -94.503 " />
        <point X = " -54.5 " Y = " 0.0 " Z = " -94.503 " />
    </translate>


    <rotate time="19.387"  axisY="1" />

    <models>
        <model file="mercury.3d" />
    </models>
</group>
```

Figura 3- Translate e rotate com tempo.

VBOs

Um VBO (Vertex Buffer Object) é uma opção de OpenGL que nos permite passar propriedades de vértices (posição, vetores normais, etc) para a placa gráfica e assim fazer *render* de forma não imediata. Os modelos são então desenhados com apenas uma chamada à função ***glDrawArrays.***

Para a sua implementação nesta fase do trabalho utilizamos um *buffer*: **GLuint buffers[1].** Alteramos também a nossa função ***renderScene.*** Em vez de criarmos os objetos passando os seus vértices dos pela função ***glVertex3f***, utilizamos a funções ***glBindBuffer*** e ***glBufferData*** para ativar e carregar o *buffer.* Vamos também contando o número de vértices das figuras à medida que vão sendo carregadas no *buffer*.

De seguida são chamadas as funções ***glVertexPointer*** para definir a semântica e ***glDrawArrays*** para desenhar o que está no *buffer* através de *GL_TRIANGLE_STRIPs*.

De forma a poder utilizar esta funcionalidade de OpenGL, chamamos na ***main*** as funções ***glGenBuffers*** com o número de *buffers* e o seu nome, e ***glEnableClientState*** com o parâmetro *GL_VERTEX_ARRAY.*

Catmull-Rom

As curvas de Catmull-Rom são curvas cúbicas, utilizadas no motor para definir trajetórias. A nossa implementação permite o uso de um número arbitrário de pontos de controlo, passados através do ficheiro XML. No mínimo devem ser especificados 4 pontos de controlo.

Para a correta criação das curvas são utilizadas as funções ***cross***, ***normalize***, ***multVectorMatrix***, ***multMatrixMatrix***, ***getCatmullRomPoint***, ***renderCatmullRomCurve*** e ***catmullRom.***

As funções *cross*, *normalize*, *multVectorMatrix*, *multMatrixMatrix* são funções que foram fornecidas pelo professor para o trabalho da aula prática sobre curvas. Estas são usadas para manipular matrizes e vetores.

A função **getGlobalCatmullRomPoint** calcula e devolve um ponto da curva, para isso recebe um vetor de pontos, um tempo global, uma matriz de posição e outra com as suas derivadas.

A função ***renderCatmullRomCurve*** desenha a curva usando segmentos de reta através do **GL_LINE_LOOP**. Esta recebe um vetor de pontos.

A função ***catmullRom*** recebe um vetor de pontos e o tempo. Esta calcula o tempo da trajetória dos planetas e coloca-os a fazer esse moviemto.

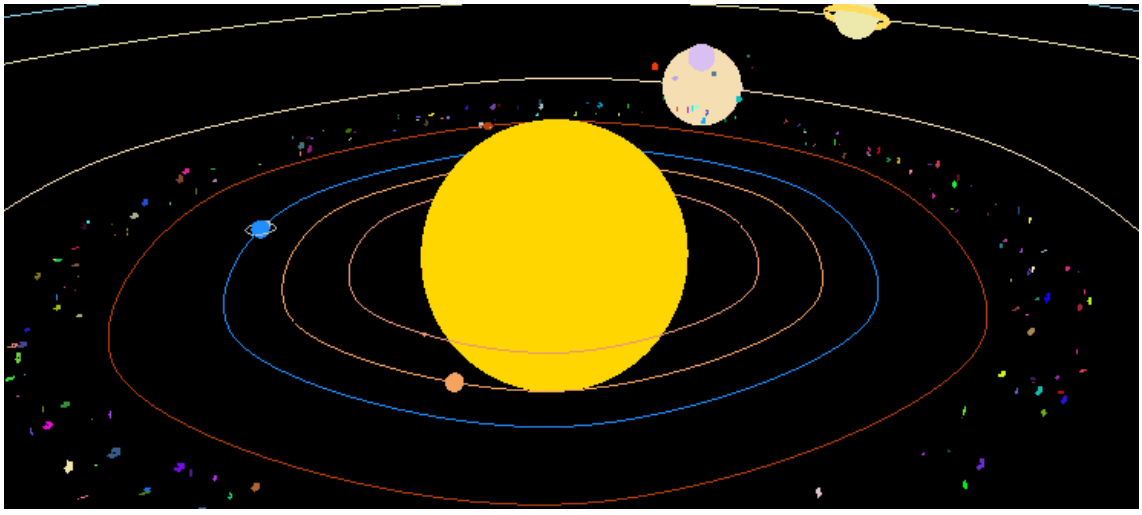


Figura 4- Demonstração das curvas de translação.

Patches de Bézier

As Patches de Bézier servem essencialmente para conseguirmos desenhar uma figura mais complexa. Para se gerar uma Patch de Bézier existem, algumas normas, tal como demonstra a figura.

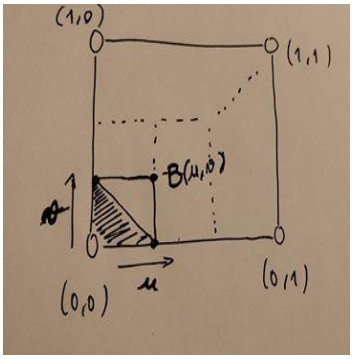
```
2  <- number of patches
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
3, 16, 17, 18, 7, 19, 20, 21, 11, 22, 23, 24, 15, 25, 26, 27
28  <- number of control points
1.4, 0, 2.4  <- control point 0
1.4, -0.784, 2.4  <- control point 1
0.784, -1.4, 2.4  <- control point 2
0, -1.4, 2.4
1.3375, 0, 2.53125
1.3375, -0.749, 2.53125
```

indices for the first patch

indices for the second patch

E para processar uma Patch há basicamente duas partes, uma onde lemos e guardamos valores, e outra onde fazemos contas e geramos o ficheiro .3d pretendido.

Para a primeira fase é lida a primeira linha do ficheiro, que segundo as normas, corresponde ao número de paches. Depois são lidas tantas linhas quantas o valor que acabamos de ler. Guardamos esses valores (índices) numa matrix, cujo índice corresponde ao número da patch e o valor ao conjunto dos 16 índices que formam essa patch. Depois disso lemos outra linha que corresponde ao número de Pontos de controlo. E fazemos sensivelmente o mesmo que fizemos para os índices.



Depois de termos tudo guardado começamos a parte de calcular os pontos, temos que calcular o B(u,v) para todos os pontos do quadrado, em que o lado do quadrado é dado por 1/slices (primeiro parâmetro que a função Patch_Bezier). Este B(u,v) é dado pela formula da seguinte imagem:

$$B(u,v) = [u^3 \quad u^2 \quad u \quad 1]M\begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Onde

$$U = [u^3 \quad u^2 \quad u \quad 1] \text{ and } M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

A matriz P é uma matriz de ordem 4x4 onde cada valor é dado pela matriz que guardamos dos pontos, com o índice dado pela matriz de índices.

Tanto a matriz M, como a P e a M transposta é algo que é processado uma única vez. O resto é andar com o 'u' e com o 'v' percorrendo os pontos de forma a se desenhar cada mini quadrado, ou seja 2 triângulos. Primeiro o ponto B(u,v), depois o B(u+1/slices,v) e por fim o B(u,v+1/slices), para o outro triângulo é B(u+1/slices,v) depois B(u+1/slices,v+1/slices) e por fim B(u,v+1/slices). Depois é calcular o x o y e o x, indice 0,1,2 respetivamente, do resultado do B(u',v').

O único cuidado que é necessário se ter é quando se multiplica a matriz UM por P, uma vez que a matriz P aparentemente é uma 4x4 em que os elementos são um ponto (x,y,z,1), e o cuidado vem uma vez que com essa facto ela torna -se uma matriz 4x4x4, mas a multiplicação de UM por M nao é uma multiplicação de matrizes normal, uma vez que os últimos 4 valores correspondem a um ponto. Ou seja a multiplicação tem que ser de um escalar por um ponto, portanto multiplicar cada componente do ponto por aquele escalar.

Esta parte do trabalho foi feita em python, pois estávamos a ter alguns problemas e decidimos usar uma linguagem que já tivesse bibliotecas como o numpy que multiplica matrizes e que nos fez poupar tempo. E para o efeito é igual.

É compilado como python3 Patch_Bezier <slices> <patch> <ponto.3d>.

Compilação

Para além dos ficheiros necessários à criação das figuras é também fornecida uma *Makefile* para compilação e execução do projeto.

Para gerar as figuras necessárias para o nosso modelo do Sistema Solar basta fazer "**make gera**" e de seguida, "**make engi**" para executar o *engine* utilizando o ficheiro XML com as transformações necessárias.

Sistema Solar

Utilizamos como base o sistema solar usado na fase anterior. Decidimos retirar a translação de -380 no eixo do Z que tínhamos e fizemos uma escala de 0.3 em todos os eixos.

Para cada planeta acrescentamos uma translação com tempo para todos girarem á volta do sol. Para ser um pouco mais realista o movimento de translação, consultamos o site "http://arifabitami.blogspot.pt/2012/03/rotacao-e-translacao-dos-planetas-do.html".

Aplicamos 1/3 da velocidade de translação até Marte, para Júpiter dividimos por 10, Saturno dividimos por 20, para Úrano por 50 e para Neptuno por 90.

As rotações aplicamos só em alguns planetas e também dividimos por 3 o tempo referido no site.

PLANETA	TEMPO DE TRANSLAÇÃO (em dias)
MERCÚRIO	87,97/3 = 29,3233
VÊNUS	224,7/3 = 74,9000
TERRA	365/3 = 121,6667
MARTE	386,98/3 = 128,9933
JÚPITER	4330/10 = 433
SATURNO	10767,5/20 = 538,375
URANO	30664/50 = 613,28
NEPTUNO	60225/90 = 669,1667
LUA	27,77 (Valor arbitrário)

Conclusão

Nesta fase do trabalho prático aprofundamos conhecimento sobre desenvolvimento gráfico, em particular em relação às **curvas de Catmull-Rom** e aos **Patches de Bezier**.

Para além do referido, fomos capazes, de alterar a criação do nosso modelo de modo imediato para VBOs e assim melhorar a sua eficiência no que toca à utilização de recursos.

Apesar de conseguirmos implementar VBOs e as curvas, sentimos algumas dificuldades em colocar a funcionar.

Com todo o nosso esforço para tentar por as **Patches de Bezier** a funcionar não conseguimos que desenhar o *teapot*.

Como trabalho futuro teremos de melhorar as câmaras e tentar aperfeiçoar as curas e pôr as **Patches de Bezier** a funcionar.