UNIVERSIDADE DO MINHO
MIEI

COMPUTAÇÃO GRÁFICA

Terceira Fase
Curves, Cubic Surfaces and VBOs

Grupo 33:
João Nunes - a82300
Shahzod Yusupov - a82617
Luís Braga - a82088
Luís Martins - a82298

Braga, Portugal
20 de Abril de 2019

Conteúdo

1 Introdução

No âmbito da unidade curricular de Computação Gráfica foi proposta, nesta terceira fase, que se implementasse um sistema capaz de desenhar um certo conjunto de figuras já idealizadas numa primeira fase, com a nuance que desta vez essas figuras e modelos sejam baseados em fragmentos de Bezier. Para tal, foi necessário actualizar o **Generator** para receber também como parâmetro um ficheiro que tem nele definido os pontos de controlo e também o nível de tesselação. No que toca ao **Engine** foram também estendidos os elementos de rotação e translação. Para tal, serão providenciados um conjunto de pontos de forma a definir uma curva de Catmull-Rom, assim como o tempo para percorrer estas curvas.

O objectivo desta fase foi criar um sistema solar dinâmico, incluindo um cometa, construído usando fragmentos de Bezier, que percorre uma trajectória definida por uma curva de Catmull-Rom.

2 Estrutura do Projecto

Sendo esta a terceira fase de um projecto contínuo e acumulativo, naturalmente grande parte do código é mantido inalterado assim como todas as funcionalidades, que apenas são cada vez mais complexas e melhoradas.

Esta terceira fase do projecto, à semelhança das anteriores, divide-se no *generator* e no *engine*. No *generator* geram-se os pontos, guardando-os num ficheiro, que mais tarde será usado para desenhar as figuras no *engine*. Esses ficheiros encontram-se referidos num ficheiro XML que o *engine* irá ler e produzir o modelo lá especificado, neste caso, o sistema solar.

No *generator* foi necessário fazer algumas alterações devido à produção dos patches de Bezier.

No *engine* também se efectuaram algumas modificações, porque foi necessário produzir as figuras usando VBO's e introduzir movimento no modelo, criando as suas órbitas segundo curvas de Catmull-Rom.

2.1 Aplicações Principais

O sistema, tal como nas fases anteriores, possui a mesma estrutura e dois programas distintos: o *generator* e o *engine*.

2.1.1 Generator

Tal como foi referido e explicado em fases passadas, esta é a aplicação onde estão definidos os algoritmos que geram os pontos usados para construir as figuras geométricas.

Para além das funcionalidades anteriormente implementadas, foi introduzida a funcionalidade de construir modelos com base em curvas de Bezier, portanto foi necessário fazer as alterações necessárias para tal ser possível.

Também o menu de ajuda deste módulo sofreu alterações ficando com seguinte aspecto.

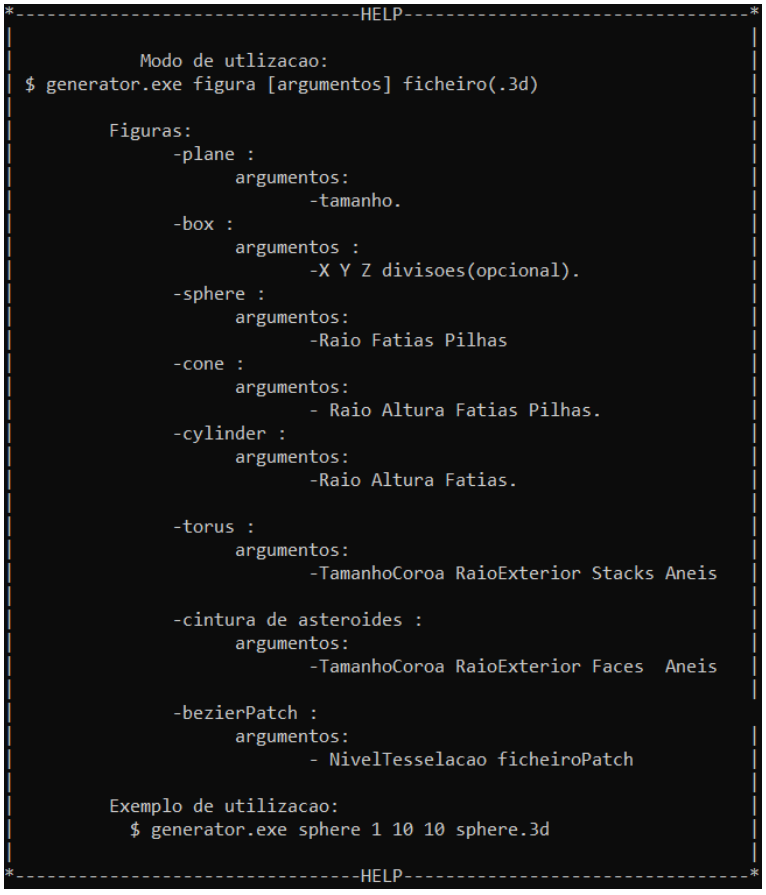

Figura 2.1: Novo aspeto do comando help no generator.

2.1.2 Engine

É no *engine* que se encontram as funcionalidades relativas ao desenho de figuras/modelos. É através deste programa que as figuras são desenhadas e apresentadas numa janela e ,é também com ele, que podemos interagir com as mesmas.

Fizeram-se várias alterações neste módulo de modo a cumprir os objetivos desta fase. As alterações foram as seguintes: no ficheiro XML t alteraram-se/criaram-se novas classes listadas na seguinte secção.

2.2 Classes Alteradas/Criadas

Nesta fase três do projeto, foi possível reaproveitar as classes criadas anteriormente, ou seja, da fase dois, que foi onde ocorreu uma maior reestruturação do código. Como tal, foi apenas necessário alterar algumas classes, e adicionar uma nova classe, de modo a cumprir com os requisitos necessários.

2.2.1 Engine/Rotate

```
1 #ifndef __ROTATE_H__
2 #define __ROTATE_H__
```

```
3
4  #include <GL/glut.h>
5
6  class Rotate {
7    float time;
8    float x;
9    float y;
10   float z;
11
12 public:
13   Rotate();
14   Rotate(float, float, float, float);
15   float getTime();
16   float getX();
17   float getY();
18   float getZ();
19   void setTime(float);
20   void setX(float);
21   void setY(float);
22   void setZ(float);
23   void apply();
24 };
25
26
27 #endif
```

No **rotate.cpp**, o principal objetivo desta classe é elaborar as rotações dos planetas sobre eles próprios, tendo sido, portanto, alterada de modo cumprir com esta finalidade.

Como tal, foram adicionadas três variáveis distinas, uma para cada eixo do referencial cartesiano (*x,y e z*), sendo que estas três variáveis irão representar o eixo sobre o qual a rotação irá ser aplicada.

De seguida, foi criado também uma variável *time*, que a contrário da fase anterior onde as rotações eram apenas elaboradas segundo o ângulo, nesta fase é necessário também indicar o número de segundos de modo a completar uma rotação sobre ele próprio.

2.2.2 Engine/Translate

```
1  #ifndef __TRANSLATE_H__
2  #define __TRANSLATE_H__
3
4  #include<vector>
5  #include "../../../src/headers/Point.h"
6  #include <GL/glut.h>
7
8  class Translate {
9    float tempo;
10   vector<Point*> trans;
11   vector<Point*> curve;
12
13 public:
14   Translate();
15   Translate(float, vector<Point*>);
16   float getTempo();
17   vector<Point*> getCurve();
18   vector<Point*> getTrans();
19   void setTempo(float);
```

```
  void setTrans(vector<Point*>);
  void setCurve(vector<Point*>);
  void getCatmullRomPoint(float, int*, float*, vector<Point*>);
  void getGlobalCatmullRomPoint(float, float *, vector<Point*>);
  void drawCurve();
  void drawCatmullRomCurve();
  void draw();

};

#endif
```

No que toca à classe **translate.cpp**, ou seja, a classe responsável por estabelecer as translations no modelo do grupo, sendo possível verificar dois arrays de pontos, um para guardar os pontos retirados no parse (*trans*) e outro que possui os pontos para desenhar a curva (*curve*). Existe também uma variável de instância relativa ao número de segundos necessários para a esfera percorrer toda a curva, que é o *tempo*, cujo intuito é simular a órbita dos planetas em torno do sol.

2.2.3 Engine/Pontos

```
#ifndef __PONTOS_H__
#define __PONTOS_H__

#include <vector>
#include <string>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "../../../src/headers/Point.h"

using namespace std;

class pontos{
  GLuint buffer, size_buffer;

public:

  pontos();
  pontos(vector<Point*> l);
  void prepare(vector<Point*>);
  void draw();

};

#endif
```

Nesta fase do projeto, é proposta a utilização do **VBO**, sendo este um recurso oferecido pelo *OpenGL*.

O principal ganho obtido por utilizar o VBO é em termos de performance, uma vez que permite guardar informação na própria memória gráfica ao invés da memória do sistema, o que permite uma renderização mais rápida e mais direta.

De modo a implementar o VBO é necessário recorrer a buffers, que se tratam nada mais nada menos do que arrays com os pontos necessários para a elaboração do modelo em questão. Então, foi criada a classe **pontos.cpp**, tendo sido necessário criar um *buffer*, onde são inseridos

os pontos necessários para a elaboração do modelo em causa, sendo preciso também passar o seu tamanho (*size_buffer*).

Em termos de funções, as duas funções principais são o *prepare* e o *draw*, sendo que a função *prepare* é responsável por criar e preencher o array com pontos, por sua vez a função *draw* está encarregue de desenhar os vários modelos que compõem o ficheiro *xml*.

2.2.4 Engine/BezierPatch

```
#ifndef __BEZIERPATCH_H__
#define __BEZIERPATCH_H__

#include <vector>
#include "../../src/headers/Point.h"

using namespace std;

class BezierPatch {

  vector<Point*> pontosControlo;

  public:
    BezierPatch();
    BezierPatch(vector<Point*>);
    vector<Point*> getPontosControlo();
    void setPontosControlo(vector<Point*>);
    void addPonto(Point*);
};

#endif
```

Criou-se também a classe **BezierPatch.cpp**, que é responsável por armazenar os pontos de controlo associados a cada patch. Tais pontos que são indispensáveis ao processamento das curvas de Bezier.

Podemos ver que existe um array de pontos (pontosControlo) usado para guardar os pontos de controlo das curvas de Bezier.

3 Explicação das novas funcionalidades

3.1 Patches de Bezier

De forma a aumentar a qualidade de desenho do projecto, foi proposto que se usasse uma nova técnica que recorre a patches de Bezier. Um patch de Bezier utiliza, 16 pontos de controlo e é relativo a uma superfície. Portanto, se possuirmos um conjunto de patches de Bezier é possível desenhar figuras complexas com um aspecto realista, deixando de serem desenhadas com faces rectas e passarem a serem desenhadas com superficíes arredondadas.

Para o efeito e como exemplo, foi fornecido um ficheiro *.patch* que servirá como modelo para estrutura de ficheiros do mesmo tipo, a qual se pode ver de seguida:

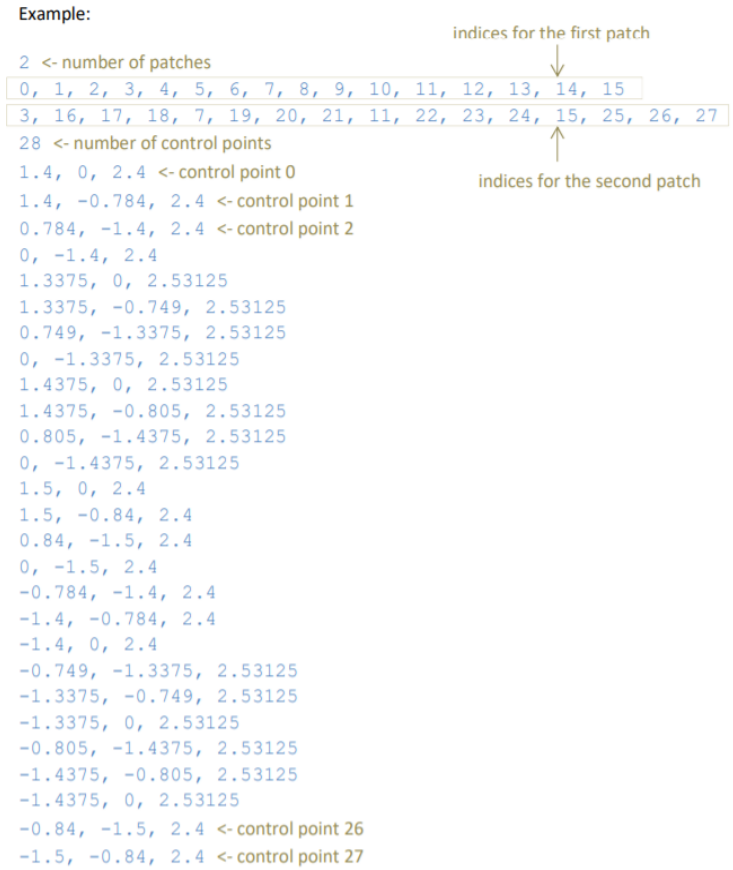

Figura 3.1: Formato do patch file.

Mas é necessário processar o patch e transformá-lo num dos ficheiros *.3d* usados para guar-

dar pontos de modelos. Para se traduzirem os patches de Bezier em modelos geométricos, convém primeiro perceber o que são curvas de Bezier.

Para criar uma curva de Bezier são necessário 4 pontos, a este pontos dá-se o nome de pontos de controlo e são constituídos pelas coordenadas x, y e z. No entanto apenas possuímos um conjunto de pontos e não uma curva, para tal temos de a calcular, usando estes 4 pontos.

Essa curva é calculada segundo uma fórmula que é expressa da seguinte maneira:

$$B(t) = (1-t)^3 B_0 + 3t(1-t)^2 B_1 + 3t^2(1-t)B_2 + t^3 B_3, t \in [0,1]$$

Que na implementação adquiriu o aspecto:

```
1  float Coef1 = (1 - v)*(1 - v)*(1 - v);
2  float Coef2 = 3 * (1 - v)*(1 - v) * v;
3  float Coef3 = 3 * (1 - v) * v * v;
4  float Coef4 = v * v * v;
5
6  float x = Coef1 * x1 + Coef2 * x2 + Coef3 * x3 + Coef4 * x4;
7  float y = Coef1 * y1 + Coef2 * y2 + Coef3 * y3 + Coef4 * y4;
8  float z = Coef1 * z1 + Coef2 * z2 + Coef3 * z3 + Coef4 * z4;
9
10 return Point(x, y, z);
```

Contudo, como se pode confirmar este código está imcompleto e mais adiante veremos o resto.

Consequentemente, é possível agora processar e obter um novo ponto (x,y,z). Em que t é representado por v e corresponde ao nível de tesselação, que dita o grau de curvatura da figura resultado.

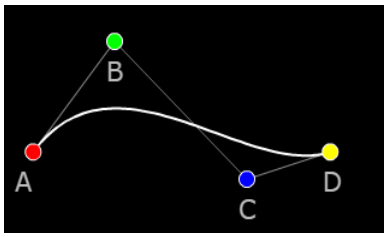

Figura 3.2: Exemplo de uma curva de Bezier.

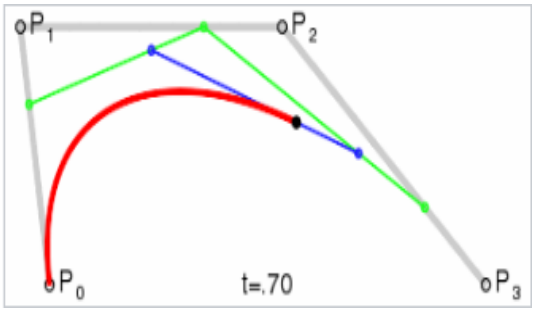

Figura 3.3: Exemplo de uma curva de Bezier.

No entanto, um patch possui 16 pontos de controlo e não 4 como uma simples curva.

No caso das curvas apenas tínhamos um parâmetro (t) para nos movimentarmos ao longo desta, mas no caso de um patch apenas um parâmetro não serve e são necessários duas variáveis para efectuar esta especificação do nível de tesselação (u,v) que calculam o grau de curvatura, ou seja, representam o mesmo papel que t só que servem para percorrer todos os pontos.

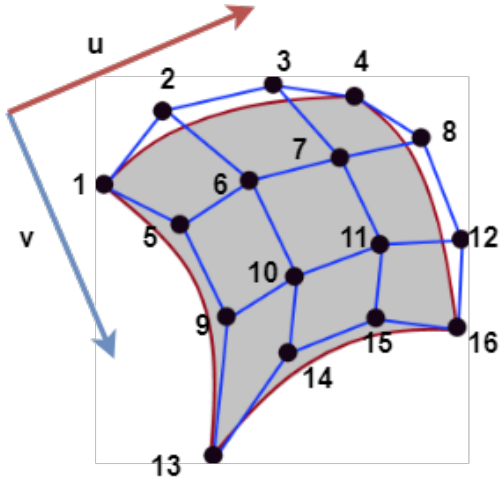

Figura 3.4: Exemplo de um patch e seu pontos de controlo.

Que depois de implementado adquiriu o aspecto:

```
  float coef1 = (1 - u)*(1 - u)*(1 - u);
float coef2 = 3 * (1 - u)*(1 - u) * u;
float coef3 = 3 * (1 - u) * u * u;
float coef4 = u * u * u;

float x1 = coef1 * B(0)->getX() + coef2 * B(1)->getX() + coef3 * B(2)->
 getX() + coef4 * B(3)->getX();
float x2 = coef1 * B(4)->getX() + coef2 * B(5)->getX() + coef3 * B(6)->
 getX() + coef4 * B(7)->getX();
float x3 = coef1 * B(8)->getX() + coef2 * B(9)->getX() + coef3 * B(10)->
 getX() + coef4 * B(11)->getX();
float x4 = coef1 * B(12)->getX() + coef2 * B(13)->getX() + coef3 * B(14)
 ->getX() + coef4 * B(15)->getX();

float y1 = coef1 * B(0)->getY() + coef2 * B(1)->getY() + coef3 * B(2)->
 getY() + coef4 * B(3)->getY();
float y2 = coef1 * B(4)->getY() + coef2 * B(5)->getY() + coef3 * B(6)->
 getY() + coef4 * B(7)->getY();
float y3 = coef1 * B(8)->getY() + coef2 * B(9)->getY() + coef3 * B(10)->
 getY() + coef4 * B(11)->getY();
float y4 = coef1 * B(12)->getY() + coef2 * B(13)->getY() + coef3 * B(14)
 ->getY() + coef4 * B(15)->getY();

float z1 = coef1 * B(0)->getZ() + coef2 *B(1)->getZ() + coef3 * B(2)->
 getZ() + coef4 * B(3)->getZ();
float z2 = coef1 * B(4)->getZ() + coef2 * B(5)->getZ() + coef3 * B(6)->
 getZ() + coef4 * B(7)->getZ();
float z3 = coef1 * B(8)->getZ() + coef2 * B(9)->getZ() + coef3 * B(10)->
 getZ() + coef4 * B(11)->getZ();
float z4 = coef1 * B(12)->getZ() + coef2 * B(13)->getZ() + coef3 * B(14)
 ->getZ() + coef4 * B(15)->getZ();
```

```
20
21    float Coef1 = (1 - v)*(1 - v)*(1 - v);
22    float Coef2 = 3 * (1 - v)*(1 - v) * v;
23    float Coef3 = 3 * (1 - v) * v * v;
24    float Coef4 = v * v * v;
25
26    float x = Coef1 * x1 + Coef2 * x2 + Coef3 * x3 + Coef4 * x4;
27    float y = Coef1 * y1 + Coef2 * y2 + Coef3 * y3 + Coef4 * y4;
28    float z = Coef1 * z1 + Coef2 * z2 + Coef3 * z3 + Coef4 * z4;
29
30    return Point(x, y, z);
```

Dito isto, é agora também possível calcular um patch e as respectiva curvas e pontos.

É também importante referir que quanto maior for o nível de tesselação, maior o número de pontos calculados o que, consequentemente, resultará numa superfície mais perfeita e realista.

3.2 Rotate

Tal como foi abordado anteriormente na explicação da classe **rotate.cpp**, foi necessário mudar a forma em como são elaboradas as rotações. A nova variável tempo, indica o número de segundos necessários para completar uma rotação de 360º em volta do eixo (x,y e z) definido. Para o efeito, foi utilizada a seguinte fórmula:

```
1 r = glutGet(GLUT_ELAPSED_TIME) % (int)(time * 1000);
2 gr = (r * 360) / (time * 1000);
```

Onde, o *glutGet(GLUT_ELAPSED_TIME)* é utilizado para medir o tempo deste o ínicio da execução da função *glutInit*. Ora, como o tempo medido pelo *glutGet* cresce numa maneira diretamente proporcional à execução do projeto, foi necessário restringir o tempo, portanto, é feito o resto da divisão inteira pelo *time* (tempo lido do ficheiro *xml*) multiplicado por mil, de modo a dar o tempo em milissegundos.

De seguida, o resultado do conjunto de operações supracitado é utilizado para descobrir o ângulo da rotação (**gr**) de modo a rodar uma porção do modelo num instante de tempo. Portanto ao dividir o *time* por mil temos a porção, de seguida o **r** (calculado anteriormente) é multiplicado por 360º e, quando aplicado com a operação anterior, retorna a amplitude sobre a qual o modelo irá rodar no próprio momento.

O **gr** é posteriormente aplicado na função *glRotate* de modo a rodar o modelo consoante as restrições anteriores.

```
1 glRotatef(gr, getX(), getY(), getZ());
```

3.3 Translate

Assim como a rotação foi alterada, também a translação sofreu alterações uma vez que esta irá ser definida por uma *Catmull-Rom Cubic Curve* , alterações essas que ocorrem na classe **translate.cpp**. Para além das novas variáveis de instância apresentadas anteriormente, existem também três novos arrays auxiliares cruciais.

- **up[3]:** Alinhar o modelo com a curva;
- **res[3]:** Ponto para a próxima translação dentro da curva;

- **deriv[3]:** Derivada do ponto anterior.

Como se pode ver na seguinte instanciação dentro da classe translate na função *draw*:

```
float res[3];
float deriv[3];
float up[3];
```

Continuando, de modo a preencher estes arrays, é preciso utilizar a função *getGlobalCatmullRomPoint* que recebe como argumento um valor **t** que é calculado como se segue:

```
te = glutGet(GLUT_ELAPSED_TIME) % (int)(tempo * 1000);
gt = te / (tempo * 1000); // t = gt
```

O princípio utilizado para calcular o **t** é o mesmo que foi utilizado no *Rotate* para o cálculo do **gr**, ou seja, novamente é calculado o tempo desde a execução da função *glutInit* através da função *glutGet(GLUT_ELAPSED_TIME)*, e como tal como anteriormente, é necessário restringir este valor ao efetuar a divisão inteira pelo *tempo*, retirado do *parse* do xml multiplicado por mil. Este valor calculado *te* é utilizado depois para efetuar a divisão pelo tempo multiplicado por mil (millisegundos) de modo a finalmente ser calculado o valor *t*.

Tendo o valor de *t* já calculado, este é passado como argumento na função *getGlobalCatmullRomPoint*, que é utilizado de modo a calcular o *gt*, são também calculados os indicies que armazenam os pontos.

```
void Translate::getGlobalCatmullRomPoint(float t, float *res, vector<Point
    *> tr) {

  int indices[4];

  int tam = tr.size();
  float gt = t * tam;
  int index = floor(gt);
  gt = gt - index;

  indices[0] = (index + tam - 1) % tam;
  indices[1] = (indices[0] + 1) % tam;
  indices[2] = (indices[1] + 1) % tam;
  indices[3] = (indices[2] + 1) % tam;

  getCatmullRomPoint(gt, indices, res, tr);
}
```

Continuando, a função *getCatmullRomPoints* é definida como se segue:

```
void Translate::getCatmullRomPoint(float t, int* indices, float* res,
    vector<Point*> tr) {
  float novo[4];
  float ttt, tt;
  int ind1, ind2, ind3, ind4;
  Point p0, p1, p2, p3;

  res[0] = res[1] = res[2] = 0.0;
  ttt = t * t * t;
  tt = t * t;

  float m[4][4] = { { -0.5f, 1.5f, -1.5f, 0.5f },
            { 1.0f, -2.5f, 2.0f, -0.5f },
            { -0.5f, 0.0f, 0.5f, 0.0f },
            { 0.0f, 1.0f, 0.0f, 0.0f } };
```

```
15
16
17    novo[0] = ttt * m[0][0] + tt * m[1][0] + t * m[2][0] + m[3][0];
18    novo[1] = ttt * m[0][1] + tt * m[1][1] + t * m[2][1] + m[3][1];
19    novo[2] = ttt * m[0][2] + tt * m[1][2] + t * m[2][2] + m[3][2];
20    novo[3] = ttt * m[0][3] + tt * m[1][3] + t * m[2][3] + m[3][3];
21
22    ind1 = indices[0];
23    ind2 = indices[1];
24    ind3 = indices[2];
25    ind4 = indices[3];
26
27    p0 = (*tr[ind1]);
28    p1 = (*tr[ind2]);
29    p2 = (*tr[ind3]);
30    p3 = (*tr[ind4]);
31
32    res[0] = novo[0] * p0.getX() + novo[1] * p1.getX() + novo[2] * p2.getX()
       + novo[3] * p3.getX();
33    res[1] = novo[0] * p0.getY() + novo[1] * p1.getY() + novo[2] * p2.getY()
       + novo[3] * p3.getY();
34    res[2] = novo[0] * p0.getZ() + novo[1] * p1.getZ() + novo[2] * p2.getZ()
       + novo[3] * p3.getZ();
35
36 }
```

Em suma a função supracitada é utilizada para calcular as coordenadas dos pontos que definem a curva de *Catmull-Rom*, sendo utilizado o cálculo de matrizes de modo a calcular esses valores, tal como se segue:

$$M = \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Como se poderá verificar no seguinte extrato de código.

```
1 float m[4][4] = { { -0.5f, 1.5f, -1.5f, 0.5f },
2                   { 1.0f, -2.5f, 2.0f, -0.5f },
3                   { -0.5f, 0.0f, 0.5f, 0.0f },
4                   { 0.0f, 1.0f, 0.0f, 0.0f } };
```

Para as restantes matrizes, utilizadas nos cálculos, temos as seguintes definições:

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

Tendo sido definida no código por:

```
1 novo[0] = ttt * m[0][0] + tt * m[1][0] + t * m[2][0] + m[3][0];
2 novo[1] = ttt * m[0][1] + tt * m[1][1] + t * m[2][1] + m[3][1];
3 novo[2] = ttt * m[0][2] + tt * m[1][2] + t * m[2][2] + m[3][2];
4 novo[3] = ttt * m[0][3] + tt * m[1][3] + t * m[2][3] + m[3][3];
```

Sendo que, o ttt anterior, e assim sucessivamente, é dado multiplicando o valor de t, calculado anteriormente na função *draw*, pelo seu grau, ou seja:

```
1 ttt = t * t * t;
2 tt = t * t;
```

Prosseguindo, foi também calculado o array que possui os valores dos pontos para definir a curva, esses valores encontram-se armazenados num array **p**, tendo isto em conta, as coordenadas dos pontos que definem a curva irão ser dadas através da multiplicação das matrizes ou seja, multiplicando o *T*M*P* sendo que o resultado irá ser armazenado no array **res**, tal como se segue:

```
res[0] = novo[0] * p0.getX() + novo[1] * p1.getX() + novo[2] * p2.getX() +
    novo[3] * p3.getX();
res[1] = novo[0] * p0.getY() + novo[1] * p1.getY() + novo[2] * p2.getY() +
    novo[3] * p3.getY();
res[2] = novo[0] * p0.getZ() + novo[1] * p1.getZ() + novo[2] * p2.getZ() +
    novo[3] * p3.getZ();
```

Após obter os valores do array **res**, sendo que cada indicie do array representa os valores na coordenada *x, y e z*, são agora passíveis de serem utilizados na função *glTranslatef* que se localiza na função *draw*.

```
glTranslatef(res[0], res[1], res[2]);
```

É de notar também a existência da função *curveRotation(deriv, up);* dentro da função *draw* uma vez que é necessário que o modelo em causa siga a orientação da curva, portanto, são utilizados os arrays **deriv** e **up**, referidos anteriormente, de modo a responder a esta necessidade.

Possuindo agora as coordenadas dos pontos que definem a translação de acordo com a curva de *Catmull-Rom*, é necessário desenhar esta mesma curva, de modo a implementar as órbitas dos planetas em torno do sol.

Como tal, é utilizada a função *drawCurve* que foi implementada da seguinte maneira:

```
void Translate::drawCurve() {

  float res[3];

  for (float t = 0; t < 1; t += 0.01) {
    getGlobalCatmullRomPoint(t, res, trans);
    curve.push_back(new Point(res[0], res[1], res[2]));
  }
}
```

Como é possível verificar, esta função utiliza o *getGlobalCatmullRomPoint* que por sua vez utiliza o *getCatmullRomPoint*, que foram explicados anteriormente, uma vez que com estas chamadas é possível obter as coordenadas do próximo ponto que define a curva para um dado valor **t**, que foi também visto o seu cálculo anteriormente.

Portanto, foi aplicado um ciclo, onde o t é incrementado em 0.01 unidades até chegar ao valor de 1, começando no 0. O que irá fazer com que a curva seja definida por 100 valores. No final, é aplicada uma função que com estes pontos, liga-os e desenha a trajetória pretendida, tal como se pode ver na função seguinte:

```
void Translate::drawCatmullRomCurve() {
  int tam = curve.size();
  float p[3];

  glBegin(GL_LINE_LOOP);

  for (int i = 0; i < tam; i++) {
    p[0] = curve[i]->getX();
    p[1] = curve[i]->getY();
    p[2] = curve[i]->getZ();
    glVertex3fv(p);
```

```
12      }
13      glEnd();
14  }
```

4 Resultados obtidos

O sistema solar dinâmico foi elaborado tendo sempre em mente a realidade, tentando sempre que possível aproximar o modelo construído, como por exemplo, na construção nas órbitas dos planetas, onde o grupo teve o cuidado de analisar o tempo real despendido numa translação à volta do sol, para cada um dos planetas, e encaixar esses tempos no modelo.

Para além das órbitas, foi adicionado também um cometa, gerado através de um ficheiro de configuração através dos *patches de bezier*, tomando a forma de um *teapot*.

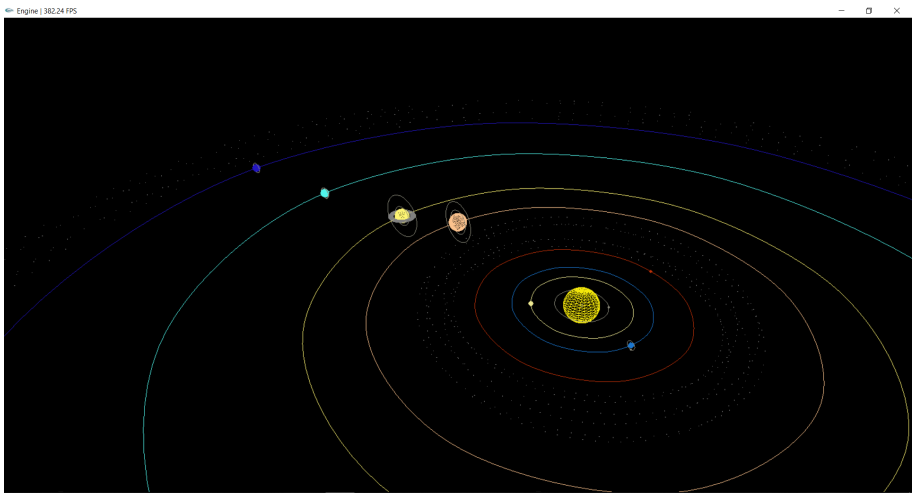

Figura 4.1: Visualização do sistema solar sem cometa.

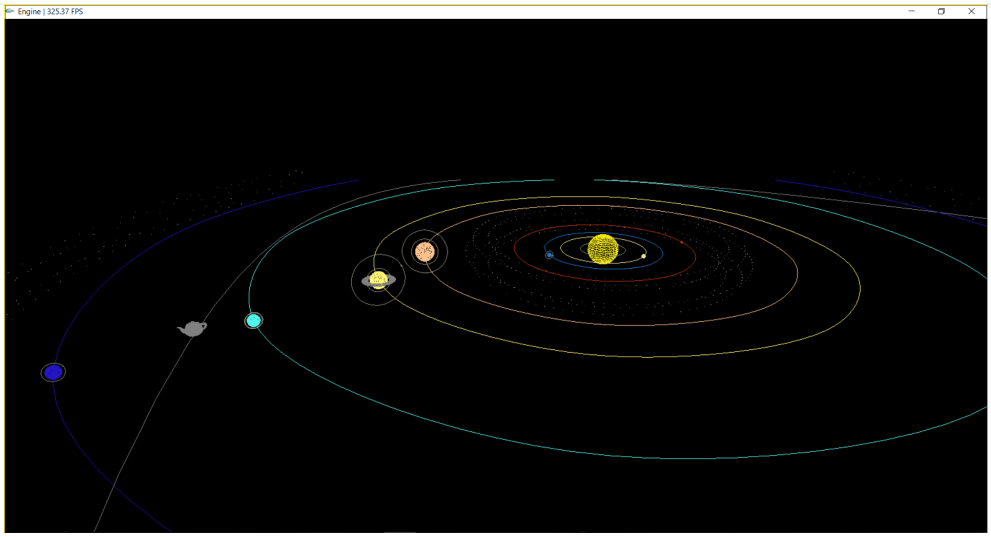

Figura 4.2: Visualização do sistema solar com cometa.

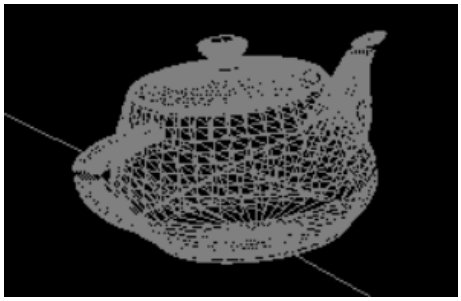

Figura 4.3: Visualização do cometa, parte lateral.

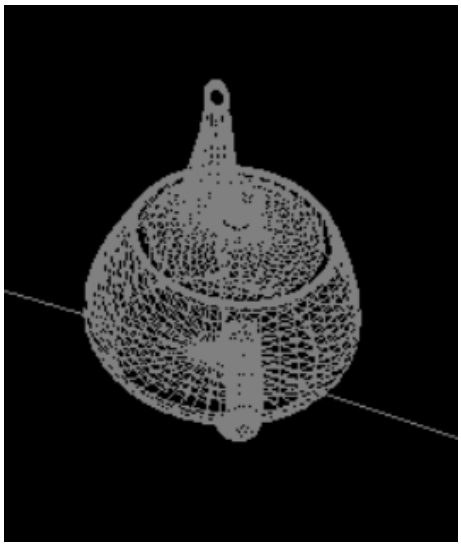

Figura 4.4: Visualização do cometa, parte de trás.

5 Conclusão

Finda a elaboração da terceira fase do trabalho, o grupo considerou que esta, ao contrário da fase anterior que se adivinhou como sendo mais fácil, foi consideravelmente mais complexo. Embora não tenha sido necessário criar tantas classes novas, ao contrário da fase anterior, foi necessário repensar e reescrever as classes já existentes sendo que, agora o nível de exigência é mais elevado, com conceitos mais complexos tais como os *Patches de Bezier* e as *Curvas De Catmull-Rom*, sendo que a maior parte das dificuldades se concentraram nestes dois conceitos.

No que toca aos *Patches de Bezier* como o grupo nunca teve nenhum contacto prévio com este conceito, surgiram várias dificuldades no que toca ao algoritmo a utilizar, sendo que após muita pesquisa e *trial and error*, chegou-se a uma solução que o grupo considera adequada.

De seguida nas *Curvas de Catmull-Rom* e uma vez que o grupo já teve contacto nas aulas práticas, embora sendo um conceito complexo e exigente, o grupo conseguiu superar este obstáculo com mais facilidade que o *stumbling block* anterior.

Contudo, tendo tudo em conta, esta fase do projeto foi crítica de modo a completar e tornar o sistema solar ainda mais fidedigno, possuindo como resultado um modelo dinâmico.

Bibliografia

[1] A Primer on Bézier Curves, https://pomax.github.io/bezierinfo/explanation

[2] Common Spline Equations for Graphics, http://blackpawn.com/texts/splines/

[3] Space Facts, https://space-facts.com/planet-orbits/

6 Anexo

```
<?xml version="2.0" ?>
<scene>
<group>
<!-- SOL !-->
<scale X="16.0" Y="16.0" Z="16.0"/>
<colour X="0.956" Y="0.909" Z="0.043"/>
<models>
<model file='../3dFiles/sphere.3d'/>
</models>
<group>
<!-- MERCURIO !-->
<translate time="21">
      <point X="1.5" Y="0" Z="0"/>
      <point X="1.0607" Y="0" Z="1.0607"/>
      <point X="0" Y="0" Z="1.5"/>
      <point X="-1.0607" Y="0" Z="1.0607"/>
      <point X="-1.5" Y="0" Z="0"/>
      <point X="-1.0607" Y="0" Z="-1.0607"/>
      <point X="0" Y="0" Z="-1.5"/>
      <point X="1.0607" Y="0" Z="-1.0607"/>
    </translate>
    <rotate time="10" X="0" Y="1" Z="0"/>
<scale X="0.05" Y="0.05" Z="0.05"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
<group>
<!-- VENUS !-->
<translate time="56">
      <point X="2.8421" Y="0" Z="0"/>
      <point X="2.0097" Y="0" Z="2.0097"/>
      <point X="0" Y="0" Z="2.8421"/>
      <point X="-2.0097" Y="0" Z="2.0097"/>
      <point X="-2.8421" Y="0" Z="0"/>
      <point X="-2.0097" Y="0" Z="-2.0097"/>
      <point X="0" Y="0" Z="-2.8421"/>
```

```
      <point X="2.0097" Y="0" Z="-2.0097"/>
    </translate>
    <rotate time="10" X="0" Y="1" Z="0"/>
<scale X="0.1244" Y="0.1244" Z= "0.1244"/>
<colour X="0.929" Y="0.909" Z="0.572"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
</group>
<group>
<!-- TERRA !-->
<translate time="90">
      <point X="3.9211" Y="0" Z="0"/>
      <point X="2.7726" Y="0" Z="2.7726"/>
      <point X="0" Y="0" Z="3.9211"/>
      <point X="-2.7726" Y="0" Z="2.7726"/>
      <point X="-3.9211" Y="0" Z="0"/>
      <point X="-2.7726" Y="0" Z="-2.7726"/>
      <point X="0" Y="0" Z="-3.9211"/>
      <point X="2.7726" Y="0" Z="-2.7726"/>
    </translate>
    <rotate time="20" X="0" Y="1" Z="0"/>
<scale X="0.1310" Y="0.1310" Z= "0.1310"/>
<colour X="0.105" Y="0.466" Z="0.815"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
<group>
<!-- SATELITE - LUA !-->
    <translate time="1">
<point X="2" Y="0" Z="0"/>
            <point X="1.4" Y="0" Z="1.4"/>
            <point X="0" Y="0" Z="2"/>
            <point X="-1.4" Y="0" Z="1.4"/>
            <point X="-2" Y="0" Z="0"/>
            <point X="-1.4" Y="0" Z="-1.4"/>
                <point X="0" Y="0" Z="-2"/>
        <point X="1.4" Y="0" Z="-1.4"/>
        </translate>
        <rotate time="10" X="0" Y="1" Z="0"/>
        <scale X="0.2727" Y="0.2727" Z="0.2727"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
</group>
<group>
```

```
<!-- MARTE !-->
<translate time="170">
<point X="6" Y="0" Z="0"/>
            <point X="4.2426" Y="0" Z="4.2426"/>
            <point X="0" Y="0" Z="6"/>
            <point X="-4.2426" Y="0" Z="4.2426"/>
            <point X="-6" Y="0" Z="0"/>
            <point X="-4.2426" Y="0" Z="-4.2426"/>
            <point X="0" Y="0" Z="-6"/>
        <point X="4.2426" Y="0" Z="-4.2426"/>
    </translate>
    <rotate time="10" X="0" Y="1" Z="0"/>
<scale X="0.0695" Y="0.0695" Z= "0.0695"/>
<colour X="0.745" Y="0.192" Z="0.035"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
</group>
<group>
<!-- Cintura de Asteroides !-->
<translate time="140">
<point X="0.1" Y="0" Z="0"/>
            <point X="0.0707" Y="0" Z="0.0707"/>
            <point X="0" Y="0" Z="0.1"/>
            <point X="-0.0707" Y="0" Z="0.0707"/>
            <point X="-0.1" Y="0" Z="0"/>
            <point X="-0.0707" Y="0" Z="-0.0707"/>
            <point X="0" Y="0" Z="-0.1"/>
        <point X="0.0707" Y="0" Z="-0.0707"/>
    </translate>
<models>
    <model file='../3dFiles/cintura.3d'/>
</models>
</group>
<group>
<!-- JUPITER !-->
<translate time="1071">
<point X="12" Y="0" Z="0"/>
            <point X="8.4853" Y="0" Z="8.4853"/>
            <point X="0" Y="0" Z="12"/>
            <point X="-8.4853" Y="0" Z="8.4853"/>
            <point X="-12" Y="0" Z="0"/>
            <point X="-8.4853" Y="0" Z="-8.4853"/>
            <point X="0" Y="0" Z="-12"/>
        <point X="8.4853" Y="0" Z="-8.4853"/>
    </translate>
<scale X="0.6294" Y="0.6294" Z= "0.6294"/>
<colour X="0.984" Y="0.760" Z="0.552"/>
```

```
<models>
    <model file=’../3dFiles/sphere.3d’/>
</models>
<group>
<!-- SATELITE - IO !-->
<translate time="10">
<point X="1.4142" Y="0" Z="0"/>
             <point X="1" Y="0" Z="1"/>
             <point X="0" Y="0" Z="1.4142"/>
             <point X="-1" Y="0" Z="1"/>
             <point X="-1.4142" Y="0" Z="0"/>
             <point X="-1" Y="0" Z="-1"/>
             <point X="0" Y="0" Z="-1.4142"/>
         <point X="1" Y="0" Z="-1"/>
     </translate>
<scale X="0.025479" Y="0.025479" Z="0.025479"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = ’../3dFiles/sphere.3d’/>
</models>
</group>
 <group>
<!-- SATELITE - EUROPA !-->
<translate time="10">
<point X="2.3869" Y="0" Z="0"/>
             <point X="1.6878" Y="0" Z="1.6878"/>
             <point X="0" Y="0" Z="2.3869"/>
             <point X="-1.6878" Y="0" Z="1.6878"/>
             <point X="-2.3869" Y="0" Z="0"/>
             <point X="-1.6878" Y="0" Z="-1.6878"/>
             <point X="0" Y="0" Z="-2.3869"/>
         <point X="1.6878" Y="0" Z="-1.6878"/>
     </translate>
<scale X="0.0218318" Y="0.0218318" Z="0.0218318"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = ’../3dFiles/sphere.3d’/>
</models>
</group>
</group>
<group>
<!-- SATURNO !-->
<translate time="2650">
<point X="15.285" Y="0" Z="0"/>
            <point X="10.8081" Y="0" Z="10.8081"/>
            <point X="0" Y="0" Z="15.285"/>
            <point X="-10.8081" Y="0" Z="10.8081"/>
            <point X="-15.285" Y="0" Z="0"/>
```

```
            <point X="-10.8081" Y="0" Z="-10.8081"/>
            <point X="0" Y="0" Z="-15.285"/>
        <point X="10.8081" Y="0" Z="-10.8081"/>
    </translate>
<scale X="0.5126" Y="0.5126" Z= "0.5126"/>
<colour X="0.992" Y="0.956" Z="0.447"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
<group>
<!-- SATELITE - REIA !-->
<translate time="10">
<point X="1.4142" Y="0" Z="0"/>
            <point X="1" Y="0" Z="1"/>
            <point X="0" Y="0" Z="1.4142"/>
            <point X="-1" Y="0" Z="1"/>
            <point X="-1.4142" Y="0" Z="0"/>
            <point X="-1" Y="0" Z="-1"/>
            <point X="0" Y="0" Z="-1.4142"/>
        <point X="1" Y="0" Z="-1"/>
     </translate>
<scale X="0.013119" Y="0.013119" Z="0.013119"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
<group>
<!-- SATELITE - TITA !-->
<translate time="10">
<point X="3.0910" Y="0" Z="0"/>
            <point X="2.1857" Y="0" Z="2.1857"/>
            <point X="0" Y="0" Z="3.0910"/>
            <point X="-2.1857" Y="0" Z="2.1857"/>
            <point X="-3.0910" Y="0" Z="0"/>
            <point X="-2.1857" Y="0" Z="-2.1857"/>
            <point X="0" Y="0" Z="-3.0910"/>
        <point X="2.1857" Y="0" Z="-2.1857"/>
     </translate>
<scale X="0.044224" Y="0.044224" Z="0.044224"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
<group>
<!-- SATELITE - Anel !-->
<rotate Angle="70" X="1" Y="0" Z="0"/>
```

```
<scale X="1" Y="1" Z="1"/>
<models>
<model file = '../3dFiles/torus.3d'/>
</models>
</group>
</group>
<group>
<!-- URANO !-->
<translate time="7500">
    <point X="22.455" Y="0" Z="0"/>
            <point X="15.8781" Y="0" Z="15.8781"/>
            <point X="0" Y="0" Z="22.455"/>
            <point X="-15.8781" Y="0" Z="15.8781"/>
            <point X="-22.455" Y="0" Z="0"/>
            <point X="-15.8781" Y="0" Z="-15.8781"/>
            <point X="0" Y="0" Z="-22.455"/>
        <point X="15.8781" Y="0" Z="-15.8781"/>
    </translate>
<scale X="0.31" Y="0.31" Z= "0.31"/>
<colour X="0.317" Y="0.960" Z="0.921"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
<group>
<!-- SATELITE - ARIEL !-->
<translate time="10">
<point X="1.4142" Y="0" Z="0"/>
             <point X="1" Y="0" Z="1"/>
             <point X="0" Y="0" Z="1.4142"/>
             <point X="-1" Y="0" Z="1"/>
             <point X="-1.4142" Y="0" Z="0"/>
             <point X="-1" Y="0" Z="-1"/>
             <point X="0" Y="0" Z="-1.4142"/>
         <point X="1" Y="0" Z="-1"/>
     </translate>
<scale X="0.025666" Y="0.025666" Z="0.025666"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
 <group>
<!-- SATELITE - UMBRIEL !-->
<translate time="10">
<point X="1.3143" Y="0" Z="0"/>
             <point X="0.9294" Y="0" Z="0.9294"/>
             <point X="0" Y="0" Z="1.3143"/>
             <point X="-0.9294" Y="0" Z="0.9294"/>
```

```
            <point X="-1.3143" Y="0" Z="0"/>
            <point X="-0.9294" Y="0" Z="-0.9294"/>
            <point X="0" Y="0" Z="-1.3143"/>
        <point X="0.9294" Y="0" Z="-0.9294"/>
    </translate>
<scale X="0.024913" Y="0.024913" Z="0.024913"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
</group>
<group>
<!-- NEPTUNO !-->
<translate time="15000">
<point X="30.75" Y="0" Z="0"/>
           <point X="21.7435" Y="0" Z="21.7435"/>
           <point X="0" Y="0" Z="30.75"/>
           <point X="-21.7435" Y="0" Z="21.7435"/>
           <point X="-30.75" Y="0" Z="0"/>
           <point X="-21.7435" Y="0" Z="-21.7435"/>
           <point X="0" Y="0" Z="-30.75"/>
       <point X="21.7435" Y="0" Z="-21.7435"/>
   </translate>
<scale X="0.3" Y="0.3" Z= "0.3"/>
<colour X="0.137" Y="0.086" Z="0.752"/>
<models>
    <model file='../3dFiles/sphere.3d'/>
</models>
<group>
<!-- SATELITE - TRITAO !-->
<translate time="10">
<point X="1.4142" Y="0" Z="0"/>
            <point X="1" Y="0" Z="1"/>
            <point X="0" Y="0" Z="1.4142"/>
            <point X="-1" Y="0" Z="1"/>
            <point X="-1.4142" Y="0" Z="0"/>
            <point X="-1" Y="0" Z="-1"/>
            <point X="0" Y="0" Z="-1.4142"/>
        <point X="1" Y="0" Z="-1"/>
    </translate>
<scale X="0.059579" Y="0.059579" Z="0.059579"/>
<colour X="0.552" Y="0.549" Z="0.486"/>
<models>
<model file = '../3dFiles/sphere.3d'/>
</models>
</group>
</group>
```

```
<group>
<!-- Cintura de Kuiper !-->
<translate time="140">
<point X="0.1" Y="0" Z="0"/>
            <point X="0.0707" Y="0" Z="0.0707"/>
            <point X="0" Y="0" Z="0.1"/>
            <point X="-0.0707" Y="0" Z="0.0707"/>
            <point X="-0.1" Y="0" Z="0"/>
            <point X="-0.0707" Y="0" Z="-0.0707"/>
            <point X="0" Y="0" Z="-0.1"/>
        <point X="0.0707" Y="0" Z="-0.0707"/>
    </translate>
<models>
    <model file='../3dFiles/cinturaFora.3d'/>
</models>
</group>
<group>
<!-- Cometa !-->
<translate time="30">
<point X="40" Y="0" Z="-80" />
<point X="36.9552" Y="0" Z="-39.8182" />
<point X="28.2843" Y="0" Z="-5.75379" />
<point X="15.3073" Y="0" Z="17.0074" />
<point X="2.44929e-15" Y="0" Z="25" />
<point X="-15.3073" Y="0" Z="17.0074" />
<point X="-28.2843" Y="0" Z="-5.75379" />
<point X="-36.9552" Y="0" Z="-39.8182" />
<point X="-40" Y="0" Z="-80" />
<point X="-36.9552" Y="0" Z="-120.182" />
<point X="-28.2843" Y="0" Z="-154.246" />
<point X="-15.3073" Y="0" Z="-177.007" />
<point X="-7.34788e-15" Y="0" Z="-185" />
<point X="15.3073" Y="0" Z="-177.007" />
<point X="28.2843" Y="0" Z="-154.246" />
<point X="36.9552" Y="0" Z="-120.182" />
</translate>
<scale X="0.2" Y="0.2" Z="0.2" />
<rotate angle="270" X="1" Y="0" Z="0"/>
<models>
<model file="../3dFiles/cometaTeapot.3d" />
</models>
</group>
</group>
</scene>
```