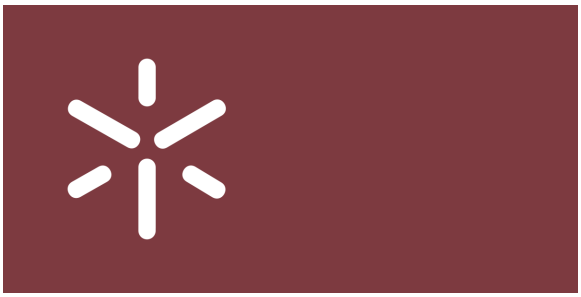


Universidade do Minho

Fase 3 - Curves, Cubic Surfaces e VBOs

Trabalho Prático de Computação Gráfica

Mestrado Integrado em Engenharia Informática

João Amorim A74806

02 de Maio de 2021

1 Introdução

No âmbito da Unidade Curricular de Computação Gráfica, foi proposto que se desenvolvesse um cenário baseado em gráficos 3D. Este projeto consiste em 4 fases, existindo vários objetivos para esta terceira fase.

Em primeiro lugar, irão ser feitas modificações ao Engine, para que os modelos sejam desenhados com recurso a VBOs, assim como efetuadas modificações nas transformações geométricas de translação e rotação com o objetivo de criar um Sistema Solar dinâmico, com recurso a curvas Catmull-Rom e a novas variáveis para cada transformação.

Posteriormente serão implementadas funcionalidades no Generator para a criação de um modelo baseado em Patches de Bézier.

Este relatório tem como objetivo descrever toda a aplicação, ajudando a executar a mesma, contendo obviamente uma descrição de todo o trabalho desenvolvido.

2 Generator

Nesta fase, um pouco ao contrário da última, foi necessário implementar novas funcionalidades no Generator, uma vez que para a demo scene, é pedido que seja incluído um cometa, com uma trajetória defida por curvas Catmull-Rom e que seja construído usando Patches de Bezier, com os pontos de controlo para o bule de chá, fornecidos no ficheiro teapot.patch. De seguida, será explicado o processo para a criação do mesmo.

2.1 Patch de Bezier

2.1.1 Processamento do ficheiro input

Para o processamento do ficheiro input, ou seja, do ficheiro teapot.patch, era necessário que numa primeira fase fosse percebida a estrutura do mesmo. Depois de uma observação do ficheiro, concluí-se que este era composto por:

- **nPatches** - Na primeira linha temos o número de patches.
- As próximas linhas, num total de **nPatches**, têm cada uma 16 números inteiros que correspondem aos indíces de cada um dos pontos de controlo que fazem parte desse patch.
- **cPoints** - Número de pontos de controlo
- As restantes linhas correspondem então aos pontos de controlo, num total de **cPoints** linhas.

2.1.2 Estruturas de Dados

Depois de analizada a composição do ficheiro, era necessário decidir que estruturas de dados utilizar para guardar a informação. Estas estruturas são arrays de arrays.

Os 16 inteiros correspondentes a um determinado patch são guardados num array de arrays que armazena inteiros, *index[i][j]*, onde i representa o índice do patch em questão, que varia entre 0 a **nPatches** e onde j corresponde aos indíces, variando de 0 e 15.

Para guardar os pontos de controlo, utiliza-se o mesmo. É utilizado um array de arrays capaz de armazenar floats, nomeadamente *points[i][j]*, onde i representa o índice do ponto de controlo, que varia entre 0 a **cPoints** e onde j representa em que ponto estamos, se X, Y, ou Z, ou seja, variando de 0 a 2.

Uma vez armazenados todos os dados, é possível passar ao processamento de todos estes dados.

2.1.3 Processamento dos Patches

Antes de passarmos à explicação do algoritmo que traduz o ficheiro de input, num modelo geométrico, temos que primeiro entender como funcionam as curvas de Bézier.

Para a criação de uma curva de Bézier, é necessário 4 pontos, definidos num espaço 3D, ou seja, constituídos por três coordenadas x, y e z.

Esses pontos só por si não geram a curva, uma vez que para processar a mesma, é preciso combinar estes pontos com alguns coeficentes.

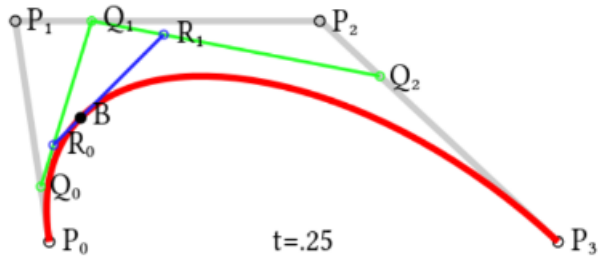


Figure 1: Exemplo de uma curva de Bézier

Sendo esta curva definida por uma equação, existe uma variável l que varia entre 0 e 1. O resultado da equação de qualquer t correspondente a uma determinada posição na curva. A equação mencionado corresponde a:

$B(t) = (1-t)^3 * P0 + 3 * t * (1-t)^2 * P1 + 3 * t^2 * (1-t) * P2 + t^3 * P3$

Onde P0, P1, P2 e P3 são os pontos de controlo. Resolvendo a equação substituindo t por vários valores entre 0 e 1, consegumos então descubrir toda a curva. A partir deste momento, já somos capazes de desenvolver o algoritmo para resolver este problema, uma vez que a única diferenca é que teremos 16 pontos de controlo em vez de 4.

Para tal, aplica-se a fórmula de Bézier, mas em vez de teros só um parâmetro t, temos dois, um xx horizontal e yy vertical. Ambos variam entre 0 e 1.

Posteriormente, aplicamos uma função que calcula a curva de Bézier para cada coordenada (xx,yy) no sentido de yy iterando até yy igual ao valor de tecelagem. Depois de efetuados esses cálculos, é aplicada a fóruma de Bézier aos resultados obtidos anteriormente. Este ciclo pertence a outro ciclo que

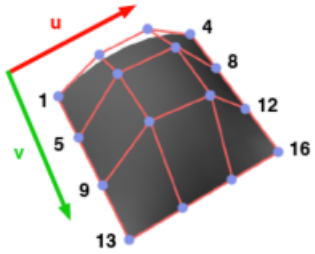


Figure 2: Patch de Bézier com os 16 pontos de controlo

está a iterar sobre xx até xx igual ao valor de tecelagem. Em cada iteração, quer xx quer yy incrementao para $(1.0/valordetecelagem * variável)$.

Finalmente, é necessário escrever para o ficheiro os pontos obtidos. De salientar que quanto maior for o valor de tecelagem, mais complexa será a figura.

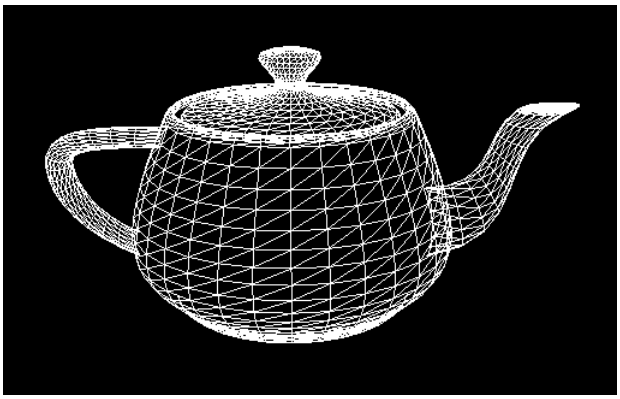


Figure 3: Modelo do Teapot obtido

3 Engine

No engine, foram efetuadas bastantes alterações. Foi necessário alterar a leitura de ficheiros XML e implementar a utilização de VBOs e Catmull-Rom Curves. Nesta secção serão apresentadas todas estas alterações.

3.1 Leitura do XML

A leitura XML continua com a mesma estrutura, onde cada uma é introduzida num grupo, tendo sido efetuadas algumas alterações no modo de leitura das transformações geométricas de translação e rotação.

Para a rotação, foi substituído o ângulo pelo tempo (**time**), sendo este tempo o número de segundos necessários para uma rotação de 360^{o} em torno do eixo especificado.

Para a translação, foram substituídos os campos X, Y e Z, passando a ser fornecido um parâmetro **time** correspondente ao número de segundos que são necessários para percorrer completamente a curva Catmull-Rom.

Ainda dentro da translação foi introduzido um ciclo para captar os pontos que definem essa mesma curva.

3.2 VBOs

Tal como já mencionado, nesta fase foi necessária a implementação de VBOs, ou Vertex Buffer Objects, para o desenho dos diferentes modelos. Estes são uma funcionalidade oferecida pelo OpenGL, que nos permitem inserir informação sobre os vértices diretamente na placa gráfica do nosso dispositivo.

Estes fornecem-nos uma melhor performance uma vez que a renderização é feita de imediato, visto que a informação já se encontra na gráfica, diminuindo assim a carga do processador.

No que diz respeito à implementação em si, de cada vez que é feito o parsing de um modelo, os pontos desse mesmo são inseridos num vetor de floats, sendo incrementada uma variável, **nVBOS** que nos diz quantos buffers terão que ser utilizados.

A função *prepareData()*, que é executada na main, aloca os apontadores para os VBOs, especifica qual o VBO ativo e copia de seguida, com a função *glBufferData*, o vetor com os pontos para o VBO. Depois de executada esta função, teremos então uma cópia do vetor na memória gráfica.

```
void prepareData() {

    buffers = (GLuint*)malloc((sizeof(GLuint)) * nVBOS);
    glGenBuffers(nVBOS, buffers);

    //Para cada buffer i
    for (int i = 0; i < nVBOS; ++i) {
        //Set buffer active
        glBindBuffer(GL_ARRAY_BUFFER, buffers[i]);
        //Fill buffer
        glBufferData(GL_ARRAY_BUFFER, sizeof(float) * vertVBOs[i].size(), vertVBOs[i].data(), GL_STATIC_DRAW);
    }
}
```

Figure 4: Função prepareData()

Finalmente, na classe Draw, é criado o funcionamento para o desenho. Inicialmente indica-se qual o VBO ativo, depois define-se a semãntica dos dados no VBO e finalmente, com glDrawArrays, é realizado o desenho dos triângulos presentes no VBO. Tal como na primeira fase, o vetor de Operations irá conter as operações de desenho que serão chamadas a partir do método run da classe Draw, na RenderScene.

```
void Draw::run() {

    glBindBuffer(GL_ARRAY_BUFFER, buffers[iVBO]);
    iVBO++;
    glVertexPointer(3, GL_FLOAT, 0, 0);

    glDrawArrays(GL_TRIANGLES, 0, vertices);
}
```

Figure 5: Método run na classe Draw

3.3 Transformações Geométricas e Curvas Catmull-Rom

3.3.1 Rotação

Para podermos implementar as novas formas de rotação, foi necessário acrescentar uma variável **time** para podermos calcular o tempo que demora a fazer uma rotação de 360^{o}. Para isto, aplicamos as seguintes fórmulas para implementar a rotação do objeto:

Com o uso da função glutGet, conseguimos determinar o tempo decorrido na execução do programa. Como este vai aumentando ao longo do tempo, é necessário estabelecer um limite para este valor. Usamos então o resto da divisão do tempo decorrido pelo tempo dado multiplicado por

```
void Rotation::run() {

    if ((int)time == 0) {
        glRotatef(this->angle, this->x, this->y, this->z);
    }
    else {
        float rAux = glutGet(GLUT_ELAPSED_TIME) % ((int)time * 1000);
        float r = (rAux * 360) / (time * 1000);
        glRotatef(r, x, y, z);

    }
}
```

Figure 6: Cálculos para a implentação da Rotação

1000(tempo em milisegundos). Conseguimos assim de seguida determinar a amplitude que o objeto vai rodar no momento multiplicando o valor calculado anteriormente pelos 360 graus, dividido novamente pelo tempo passado como parãmetro, mais uma vez multiplicado por 1000. Desta maneira, é possível passar o valor do tempo à função glRotatef.

3.3.2 Translação

Para se poder implementar as novas formas de translação, foi preciso modificar a classe Translacao, introduzindo novas variáveis, nomeadamente o tempo e dois vetores de pontos, um vetor **trans**, que guarda os pontos lidos do ficheiro XML e outro vetor **pontosCatmull** que contém os pontos de uma curva Catmull-Rom.

A inserção dos valores no vetor **pontosCatmull** é feita usando a função designada por **geraPontosCurva**, que utiliza funções auxiliares como **getCatmullRomPoint** e **getGlobalCatmullRomPoint**.

Tal como na rotação, é necessário fazer os mesmos cálculos para o tempo da translação, utilizando o parâmetro **time** passado na translação.

Depois de efetuados os cálculos, é chamada a função **getGlobalCatmullRomPoint** com o tempo calculado e com os vetores auxiliares *res*, que contém as coordenadas do ponto da próxima translação, e *deriv*, que contém a derivada do ponto anterior. No final, utilizando a função *glTranslate3f* com os valores presentes no vetor *res*, é efetuada a translação.

Para cada curva, foram defindos no total 12 pontos, sendo que o primeiro ponto possúi como coordenadas X=0, Y=0 e Z=r, onde r indica o raio do centro do Sistema Solar ao centro da corpo celeste. Com este raio e a partir do primeiro ponto, são calculados os restantes 11 pontos para a defingão da curva. O valor dos restantes pontos foi obtido com o uso de um script em

Java, utilizando como equações para as coordenadas de cada ponto, as que estão presentes na seguinte figura (para y=0):

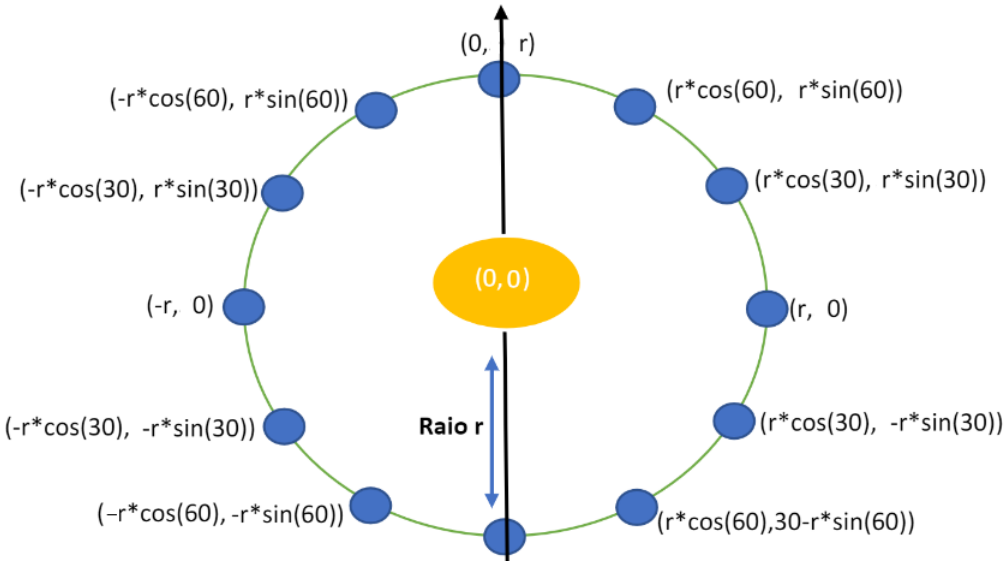


Figure 7: Equações para obtenção dos pontos para a definição das curvas Catmull-Rom

3.4 Classes

Nesta secção, serão mostrados os headers das classes que foram atualizadas nesta fase. Todas as restantes mantiveram a mesma estrutura.

3.4.1 Draw

```
#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#include <vector>
#include "Point.h"
#include "Operation.h"

#ifndef DRAW_H
#define DRAW_H

class Draw : public Operation {

    int vertices;

public:
    Draw(int vertices) {
        this->vertices = vertices;
    };
    void run();
};


#endif //DRAW_H
```

Figure 8: Classe Draw

3.4.2 Rotation

```
#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#ifndef ROTATION_H
#define ROTATION_H

#include "Operation.h"

class Rotation : public Operation {
    float time, angle, x, y, z;

public:
    Rotation();
    Rotation(float time, float angle, float x, float y, float z);
    float getTime() { return this->time; }
    float getAngle() { return this->angle; }
    float getX() { return this->x; }
    float getY() { return this->y; }
    float getZ() { return this->z; }
    void setTime(float tempo) { this->time = tempo; }
    void setAngle(float angle) { this->angle = angle; }
    void setX(float x) { this->x = x; }
    void setY(float y) { this->y = y; }
    void setZ(float z) { this->z = z; }
    void run();
};

#endif //ROTATION_H
```

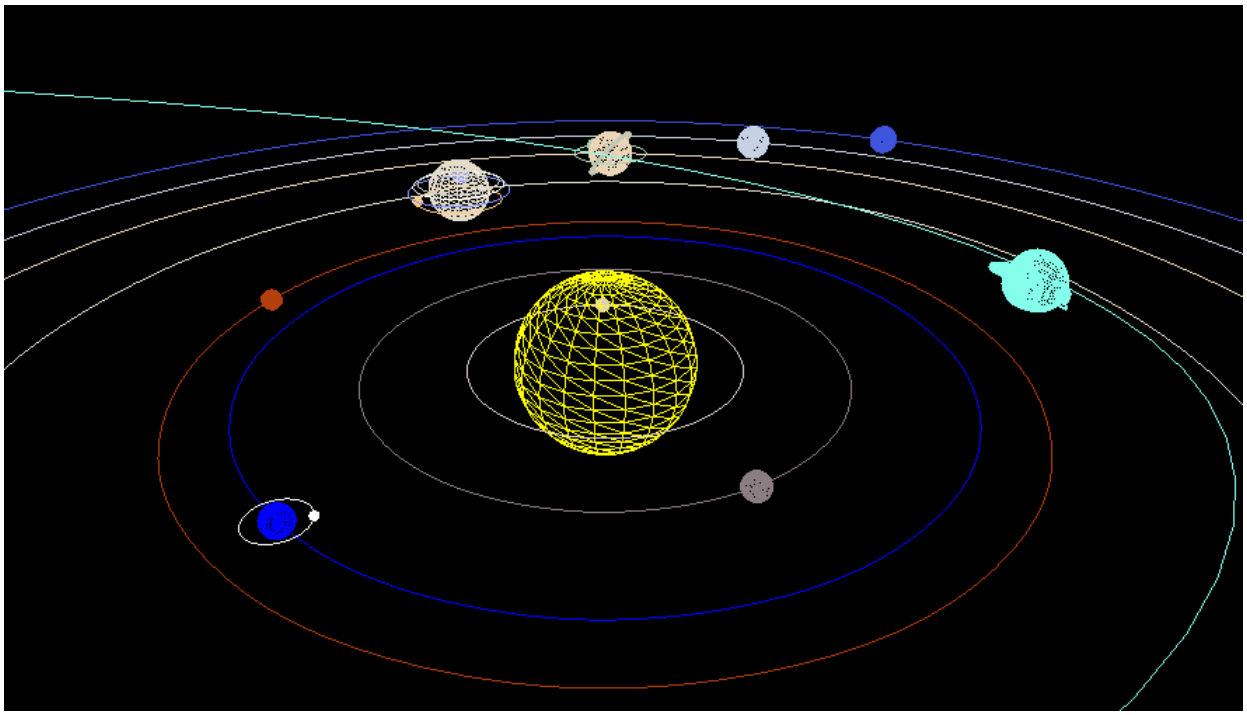
Figure 9: Classe Rotation

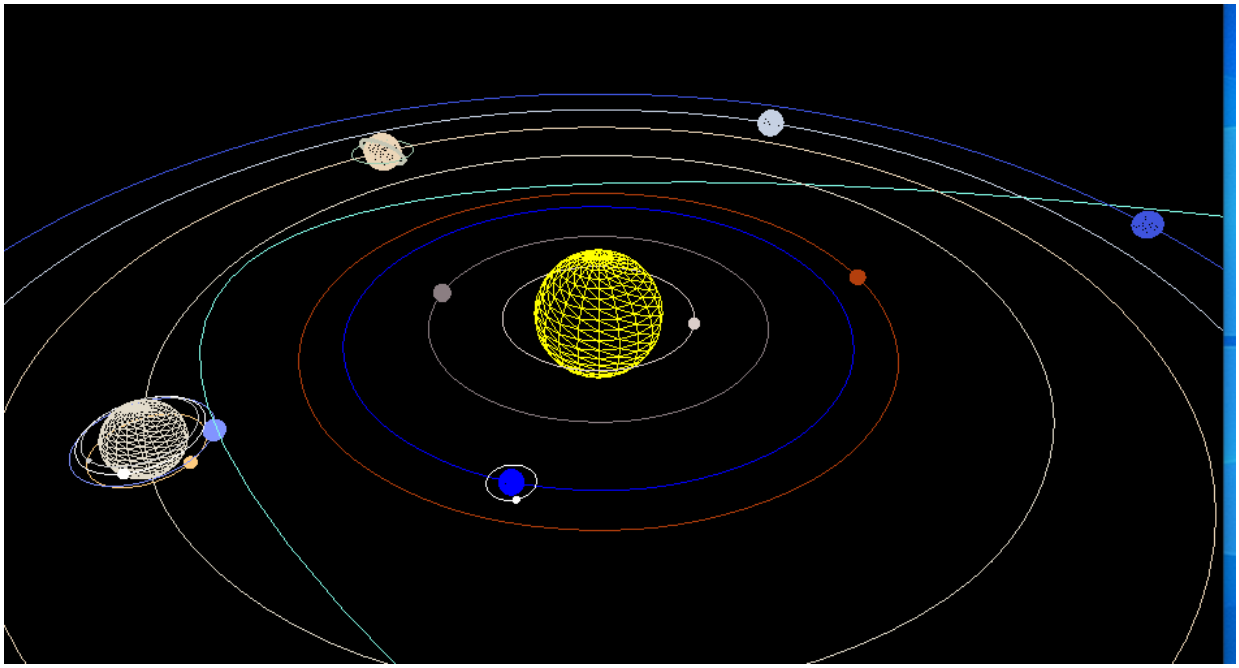
3.4.3 Translacao

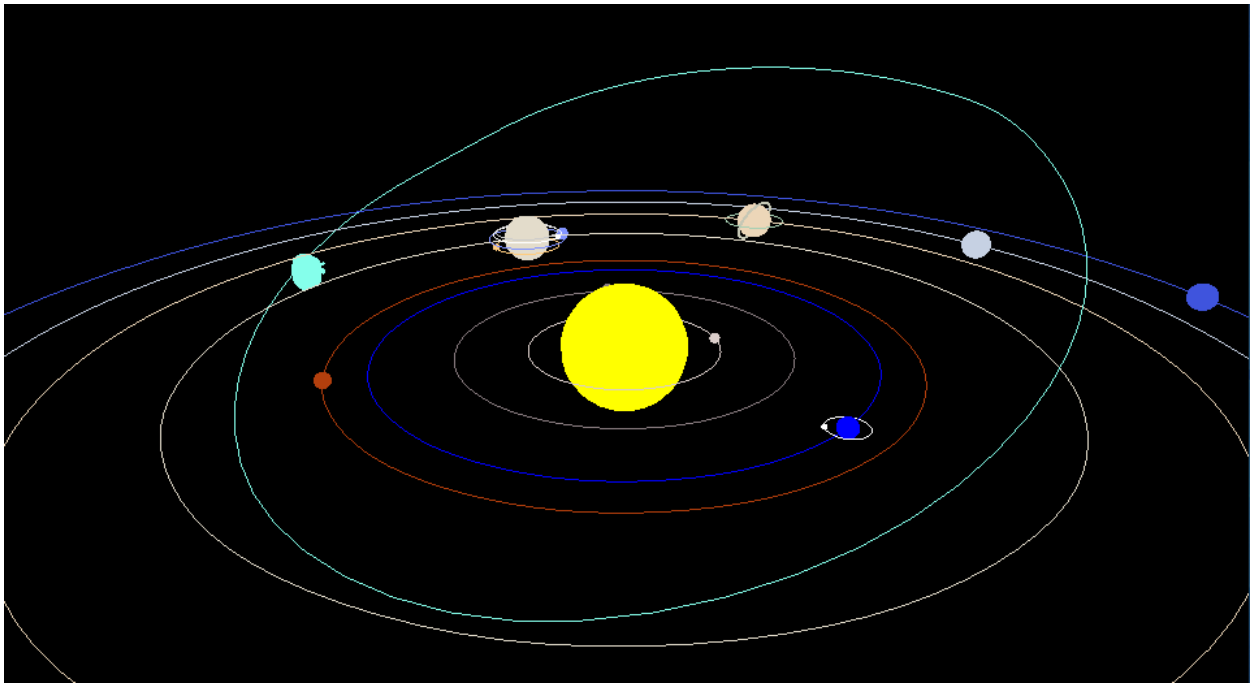
```
#include <stdlib.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif

#ifndef TRANSLACAO_H
#define TRANSLACAO_H
#include <vector>
#include "Operation.h"
#include "Point.h"

class Translacao: public Operation{
    float x, y, z;
    float tempo;
    std::vector<Point> trans;
    std::vector<Point> pontosCatmull;
    float cima[3]{};

public:
    Translacao();
    Translacao(float x, float y, float z, float t, std::vector<Point> trans);
    float getX() { return this->x; }
    float getY() { return this->y; }
    float getZ() { return this->z; }
    float getTempo() { return this->tempo; }
    float* getCima() { return this->cima; }
    std::vector<Point> getTrans() { return this->trans; }
    std::vector<Point> getPontosCatmull() { return this->pontosCatmull; }
    void setX(float x) { this->x = x; }
    void setY(float y) { this->y = y; }
    void setZ(float z) { this->z = z; }
    void setTempo(float x) { this->tempo = x; }
    void setTrans(std::vector<Point> t) { this->trans = t; }
    void setPontosCatmull(std::vector<Point> pC) { this->pontosCatmull = pC; }
    static void multMatrixVector(float* m, float* v, float* res);
    void getCatmullRomPoint(float t, float* p0, float* p1, float* p2, float* p3, float* pos, float* deriv);
    void getGlobalCatmullRomPoint(float gt, float* pos, float* deriv);
    std::vector<Point> geraPontosCurva();
    void run();
};


#endif //TRANSLACAO_H
```

Figure 10: Classe Translacao

4 Resultado Final







5 Conclusão

No final desta fase, posso dizer que estou satisfeito com o trabalho realizado. Esta foi claramente a fase mais dificil até agora, mas também aquela em que mais aprendi.

Os objetivos foram cumpridos e no final foi apresentado um Sistema Solar animado, cujos objetos seguem curvas Catmull-Rom, com o respetivo cometa cujo modelo foi criado utilizando Patches de Bezier a partir do modelo do Teapot e com a utilização de VBOs.

Penso que alguns aspetos poderiam ser melhorados, mas estando a trabalhar sozinho, penso que faço um balanço bastante positivo daquilo que foi implementado nesta fase.