Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica

Phase 3 – Curves, Cubic Surfaces and VBOs

Autores:

João Cabo	A75064	
José Sousa	A74678	

29 de Abril de 2018

Conteúdo

1 Introdução

Este projeto é o resultado de a realização da terceira fase proposta na unidade curricular **Computação Gráfica**. Esta fase manteve muitas das funcionalidades criadas nas fase anteriores, embora algumas tenham sido alteradas no processo de realização desta fase.

Começando pelas mudanças realizadas ao **generator**, nesta fase foi pedido que o gerador conseguisse criar um modelo através de ***Bezier patches***. O generator passará a poder receber os seguintes parâmetros:

- **.patch file** - Este ficheiro irá conter o número de patches, os índices dos vários patches, o número de pontos de controlo e os pontos de controlo.
- **nível de tessellation**
- **.3d file** - Ficheiro que contém a lista de triângulos que definem esta superfície

Passando agora para o **engine**, este irá sofrer várias mudanças, devido a ter de suportar novas funcionalidades. Os elementos **translate** e **rotate** que se encontram nos ficheiros XML também irão mudar. O elemento **translation** possui agora um conjunto de pontos que definirão uma ***catmull-rom curve*** e um tempo (em segundos), que indica quanto demora a realizar um volta completa a essa curva. Conseguimos assim criar animações baseadas nessas ***catmull-rom curves***. O elemento **rotation** terá agora um tempo (substituirá o ângulo), que indica o número de segundos que o objeto demora a completar uma rotação de 360 graus em torno de um eixo definido. Como consequência destas alterações, tanto o parser que é responsável por ler e processar os ficheiros como o modo que é processada a informação terá de ser alterado, de forma a satisfazer estes novos requisitos.

A última modificação que o engine irá ter está relacionado com o modo como os modelos são desenhados. Ao contrário da fase anterior, onde os modelos eram desenhados de forma imediata, estes passarão a ser desenhados com o auxílio de VBOs

Tudo isto tem como objetivo principal gerar um modelo do Sistema Solar um pouco mais realista que o elaborado na fase anterior, já que este passará de um modelo estático a um modelo com animações.

2 Arquitetura

2.1 Classes novas ou alteradas

2.1.1 Generator

Tal como explicado na fase anterior esta é a aplicação responsável por gerar os vértices de cada modelo e criar um ficheiro para cada um deles. Para esta fase foi introduzido um novo método de construção de modelos com curvas de Bézier, sendo assim foi necessário adicionar, ao gerador, funcionalidades para geração dos vértices neste novo método.

2.1.2 Engine

Tal como na fase anterior a classe engine tem diversas funcionalidades. Permite apresentar as várias primitivas e interagir com estas, através de alguns comandos. Nesta fase foi necessário alterar o motor, dada a alteração no ficheiro XML e à implementação das curvas Catmull-Rom. Alterou-se o método do parsing e também a estruturação dos dados durante a leitura.

2.1.3 Action

Classe abstrata que contém toda a informação relativa às ações que vão ser aplicadas às primitivas. Foi necessário alterar esta classe, devido às translações e às rotações. A translação agora passa a ter também um tempo correspondente ao número em segundos que demora a percorrer a curva Catmull-Rom, e um conjunto de pontos que define essa curva. A rotação necessita também de um tempo, que equivale ao número de segundos que demora a a fazer uma rotação de 360º sobre o eixo pretendido.

2.1.4 Patch

Classe que armazena os pontos de controlo de cada patch. Esses pontos são essenciais para o processamento das curvas de Bézier.

2.2 Classes inalteradas

2.2.1 Vertex

Esta classe manteve-se inalterada. Representa um vértice de um triângulo, composto pelas coordenadas (x,y,z).

2.2.2 Shape

Tal como a classe Vertex também se mantém inalterada. Representa um modelo 3D, composta por um conjunto dde vértices, sendo esses os necessários para a criação do mesmo.

2.2.3 Plane

Classe criada na primeira fase, que se manteve inalterada. A Classe Plane, com o devido algoritmo,permite-nos obter os vértices necessários para a criação de um plano.

2.2.4 Box

Classe criada na primeira fase, que se manteve inalterada. A Classe Box com o devido algoritmo, permite-nos obter os vértices necessários para a criação de uma caixa.

2.2.5 Sphere

Esta classe é exatamente igual à que foi criada na primeira fase. Permite-nos obter os vértices necessários para a criação de uma esfera.

2.2.6 Cone

Classe criada na primeira fase, que se manteve inalterada. A Classe Cone, com o devido algoritmo, permite-nos obter os vértices necessários para a criação de um cone.

2.2.7 Torus

Classe que, com o devido algoritmo, nos permite obter os vértices necessários para a criação de um torus.

2.2.8 Group

Classe que tem como objetivo armazenar toda a informação relativa a um grupo. Um grupo contém a seguinte informação:

- Modelos do grupo(shapes)
- Transformações geométricas e Cor(actions)
- Filhos (para se formar uma hierarquia um grupo pode ter outro grupo dentro de si, estes grupos são denominados childs)

2.2.9 Parser

A Classe Parser contém todos os métodos que são necessários para realizar a correta leitura e interpretação do ficheiro XML. Esta classe trata de armazenar, na estrutura indicada para isso, toda a informação relevante do ficheiro XML que é passado como input.

2.3 TinyXML2

Ferramenta utilizada para fazer o parsing dos ficheiros XML.

3 Generator

3.1 Bézier Patches

3.1.1 Leitura do patch file

Para enteder como se processa a leitura do ***patch file*** é necessário incialmente perceber o formato deste. O ficheiro apresenta o seguinte formato:

- **Número de patches** - A primeira linha do ficheiro irá conter o número de patches deste
- **Índices dos patches** - As **n** (número de patches) próximas linhas irão conter, para cada um dos patches, uma sequência de 16 números que correspondem ao número de cada um dos pontos de controlo que constituem o patch em questão.
- **Número de pontos de controlo** - A próxima linha contém o número de pontos de controlo
- **Pontos de Controlo** - As seguintes linhas contém os pontos de controlo na posição que corresponde a um índice de 0 a **n** (número de pontos de controlo)

O primeiro passo foi descobrir quantos patches continha o ficheiro de input. Foi criado um array de arrays, onde cada posição correspondia a um patch. Foram processados os índices de cada patch e armazenados em um array que foi guardado na posição que corresponde ao seu patch no array de arrays. A seguir realizou-se o mesmo procedimento para os pontos de controlo, onde cada posição do novo array que foi criado continha os 3 pontos que compõem o ponto de controlo.

3.1.2 Processamento dos patches

Para percebermos como funciona o algoritmo que traduz os **Bezier patches** para os respetivos modelos, é necessário perceber como funcionam as **Bezier curves** e como estas funcionam.

Para desenhar uma **Bezier curve** são necessários 4 pontos. Estes pontos são designados pontos de controlo e são definidos por 3 coordenadas: x, y e z. Ao termos este conjunto de pontos, podemos criar esta curva através de combinação destes mesmos pontos juntamente com alguns coeficientes.

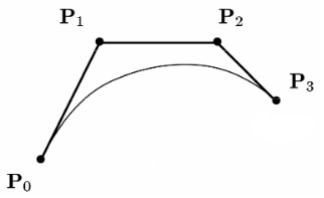

Figura 1: Bezier curve e 4 pontos de controlo

Esta curva pode ser tratada como uma curva paramétrica, ou seja, que é definida por uma equação e que, por consequência, uma variável associada a si, designada por parâmetro. Este parâmetro é identificado pela letra **t** (tessellation) e, devido a esta curva ser uma **Bezie curve**, esta variável varia entre 0 e 1. A equação da curva entre os valores 0 a 1 representa o percorrer da curva do início até ao fim. O resultado da equação para qualquer t dentro do intervalo [0:1] corresponde a uma determinada posição da curva. Se queremos visualizar a curva paramétrica, o que é necessário fazer é calcular o resultado da equação da curva à medida que vamos aumentando o valor de t usando um certo intervalo, obtendo assim vários pontos da curva.

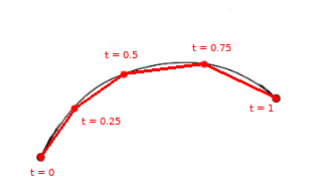

Figura 2: Curva com pontos calculados para 4 valores de t

De forma a obter um resultado mais preciso e uniforme, tudo o que é preciso fazer é diminuir o tamanho desses intervalos aumentando assim o número de pontos calculados.

O próximo passo é saber como é que é possível calcular estes pontos. A forma da curva corresponde ao resultado da combinação de vários pontos de controlo com alguns coeficientes. Como tal, surge a seguinte equação da curva:

```
P(t) = P1*x1 + P2*x2 + P3*x3 + P4*x4;
```

Onde **P1, P2, P3 e P4** são os pontos de controlo da curva de Bezier e **x1, x2, x3 x4** são os coeficientes que irão decidir a contribuição que cada um dos pontos de controlo dá para o cálculo de uma dada posição da curva.

É fácil de ver que quando **t=0**, o primeiro ponto da curva (resultado da equação) coincide com o ponto de controlo **P1** (x2=x3=x4=0). O mesmo acontece quando **t=1**, o último ponto da curva coincide com o ponto de controlo **P4** (x1=x2=x3=0).

Tirando estes 2 casos, quando o valor t se encontra entre 0 e 1, passa a ser necessário calcular os vários coeficientes, usando para isso o valor de t e as seguintes equações:

- x1 = $(1-t)^3$
- x2 = $3 * t * (1-t)^2$
- x3 = $3 * t^2 * (1-t)$
- x4 = t^3

Assim, conseguimos calcular a posição de um determinado t, substituindo o t nestas 4 equações pelo que precisamos, calculando assim os 4 coeficientes, que serão multiplicados posteriormente pelos 4 pontos de controlo.

Iremos agora passar para o processamento dos **Bezier Patches**. O princípio é bastante parecido ao usado nas **Bezier curves**, sendo que a principal diferença é o número de pontos controlo que passam de 4 pontos de controlo para 16 pontos de controlo, que podem ser vistos como uma grelha de 4x4 pontos de controlo.

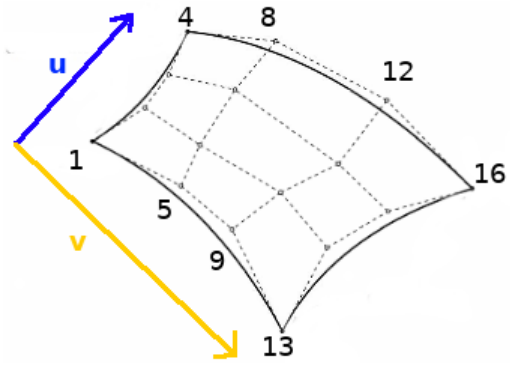

Figura 3: Bezier patch com 16 pontos de controlo

Neste caso, ao contrário das curvas que apenas continham um parâmetro **t**, passaremos a ter dois parâmetros: o parâmetro **u** para nos movimentarmos na horizontal da grelha e o parâmetro **v** para nos movimentarmos na vertical da grelha. Ambos estes parâmetros variam entre 0 e 1.

Para calcular os pontos que correspondem às coordenadas (u,v) do patch, foi necessário considerarmos cada linha da grelha 4x4 como sendo **Bezier curves** independentes. Vamos agora usar um dos pârametros (u) para calcular o ponto correspondente em cada uma destas curvas usando o algoritmo de cálculo anteriormente explicado.(u passa a ser tratado como o t do algoritmo anterior)

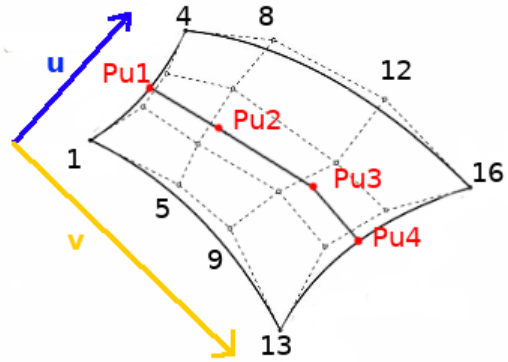

Figura 4: Pontos correspondentes a um dado valor de u para cada uma das curvas

Através deste processo obtemos 4 pontos, que podem ser vistos como os 4 pontos de controlo de uma nova **Bezier curve** orientada na direção de v. Utilizando agora o segundo parâmetro (v), podemos calcular, tal como fizemos anteriormente, o ponto final definido por essa curva. Este ponto corresponde à posição da superficie de Bezier para um dado par de valores (u,v).

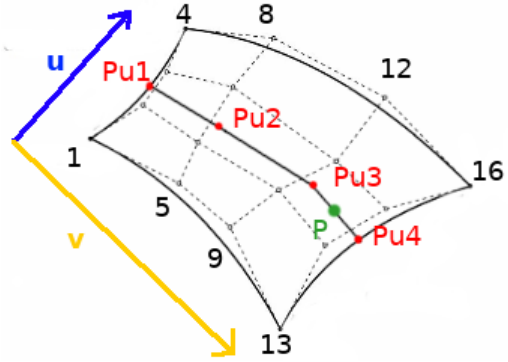

Figura 5: Ponto P correspondente a um dado v e calculado a partir dos 4 pontos anteriores

Dependendo do nível de **tessellation** que é dado como input, serão calculados os pontos para os pares (u,v) dos vários patches e colocados num ficheiro numa determinada ordem, de forma a formar um conjunto de triângulos que formem a superfície correta. Quanto maior o nível de **tessellation**, maior será o número de divisões de u e v, o que implica um maior número de pontos calculados. Como consequência a superfície que vai ser gerada vai ser mais precisa e uniforme.

4 Engine

4.1 Vexter Buffer Objects

Nesta fase passamos a usar VBOs para desenhar todos os modelos, que até aqui era desenhados de modo imediato. Esta é uma funcionalidade oferecida pelo OpenGL, que fornece métodos capazes de inserir os vértices diretamente na placa de vídeo do computador.
A principal vantagem deste método é a melhoria substancial na permormance, devido ao facto de agora a informção passar a residir na placa de vídeo e não na própria memória do sistema, podendo assim ser diretamente renderizada pela placa. Após a introdução deste método no projeto, podemos observar um aumento bastante grande de fps(frames per second). Para implementar VBOs necessitamos de criar vertex buffers, que são arrays nos quais vão ser inseridos todos os vértices do modelo que tencionamos desenhar.
As funções criadas para isto foram a Shape::readyUp() que tem como função criar o vertex buffer e
preenche-lo, e gerar o respetivo modelo, e a Shape::draw() que gera o modelo já guardado na placa de vídeo.

4.2 Curva Catmull-Rom

4.2.1 Rotate

Como já foi referido acima foi introduzida uma nova variável **time** à rotação. Esta indica o tempo em segundos, que uma rotação de 360º demorará. Com este, é-nos possível calcular o ângulo da rotação a cada momento.

```
tmp = (glutGet(GLUT_ELAPSED_TIME) % (int)(time * 1000);
r = (tmp*360) / (time * 1000);
```

O valor de retorno de glutGet(GLUT_ELAPSED_TIME) será o tempo passado desde a chamada do **glutInit()**, em milisegundos. O facto de deste retorno ser em milisegundos faz com que tenhamos de converter o valor do time, que se

encontra em segundos, multiplicando por 1000. O resto da divisão inteira do tempo decorrido desde o início pelo tempo de rotação será usado como limite da rotação. Dividindo este valor pelo time em milisegundos, ficamos com o coeficiente, que multiplicado por 360º, equivale ao angulo que vai ser introduzido na **glRotatef()**.

```
glRotatef(r,axisX,axisY,axisZ);
```

4.2.2 Translate

Na Translação foi adicionado também a variável time que equivale ao tempo, em segundos, que demora a completar uma volta, um array que é necessário para colocar e alinhar o objeto com a curva, um vetor de pontos referentes aos pontos de controlo da curva e um vetor referente aos pontos finais da translação.

```
float x;
float y;
float z;
float time;
float aux[3];
vector<Vertex*> points;
vector<Vertex*> curvePoints;
```

Foi necessário criar alguns métodos, como o **getGlobalCatmullRomPoint()** que utiliza uma lista de pontos para calcular uma trajetória, traçcando tangentes nos pontos dados e posteriormente calcular a linha que passa por esses mesmos pontos e pelas suas derivadas.

5 Resultado Final

5.1 Teapot

Resultado gerado com o ficheiro do Patch fornecido pelo professor:

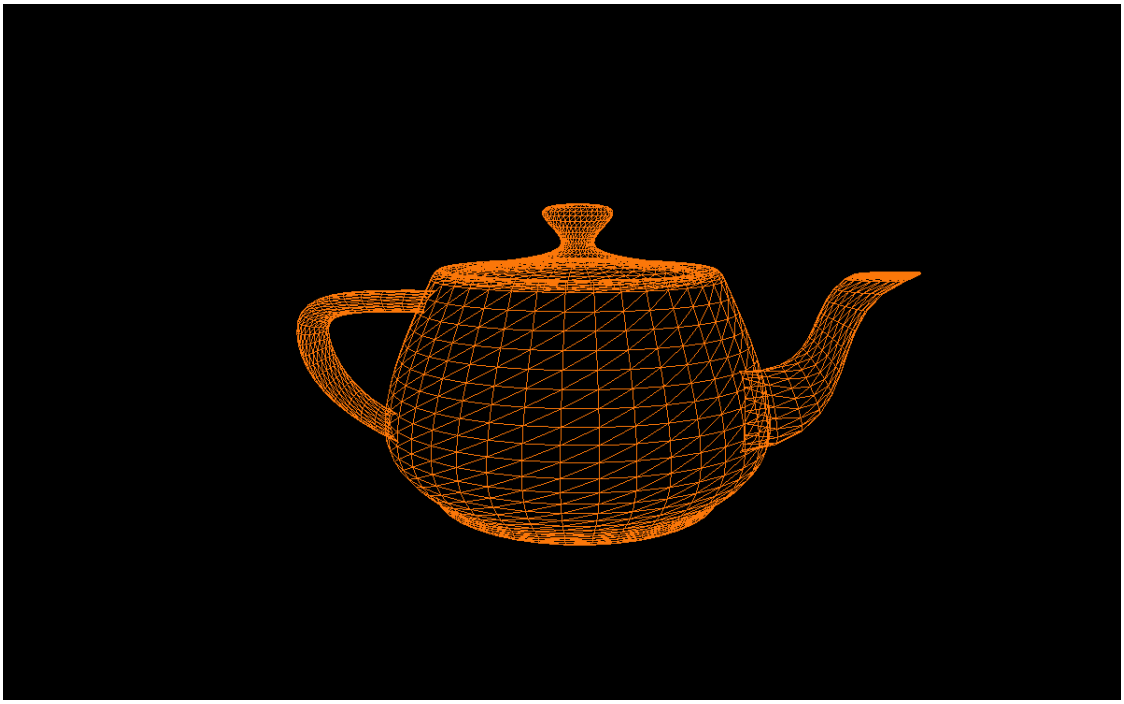

Figura 6: Visualização do teapot por linhas

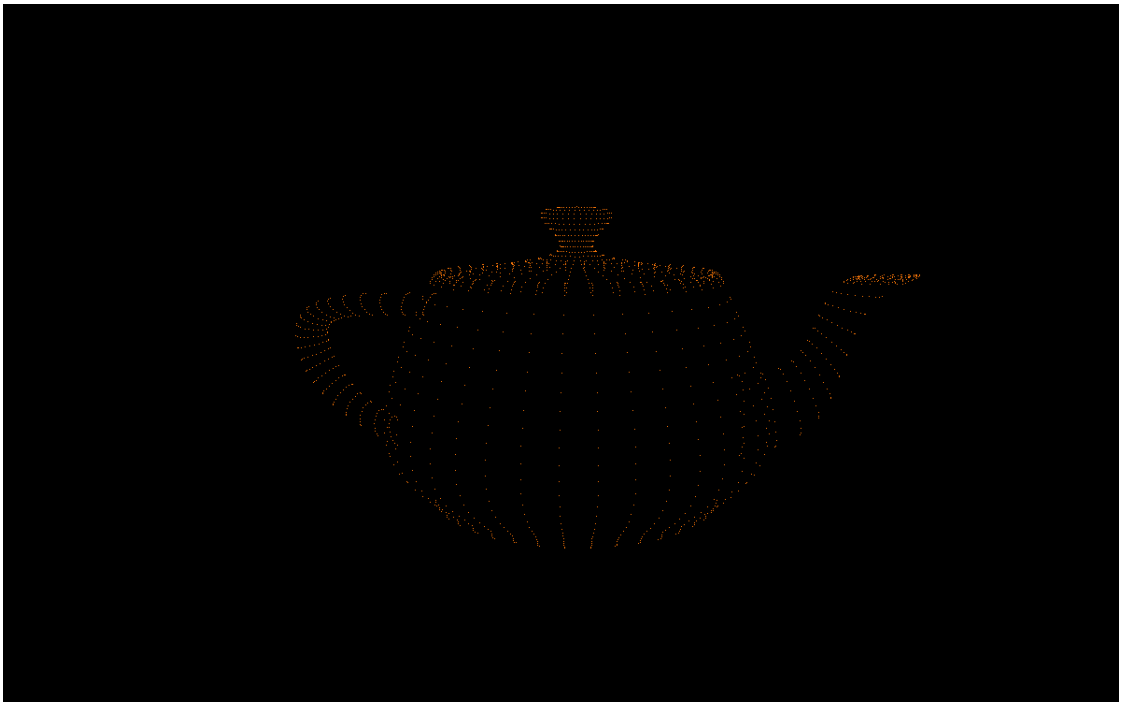

Figura 7: Visualização do teapot por pontos

Figura 8: Visualização do teapot preenchido

5.2 Tea Party

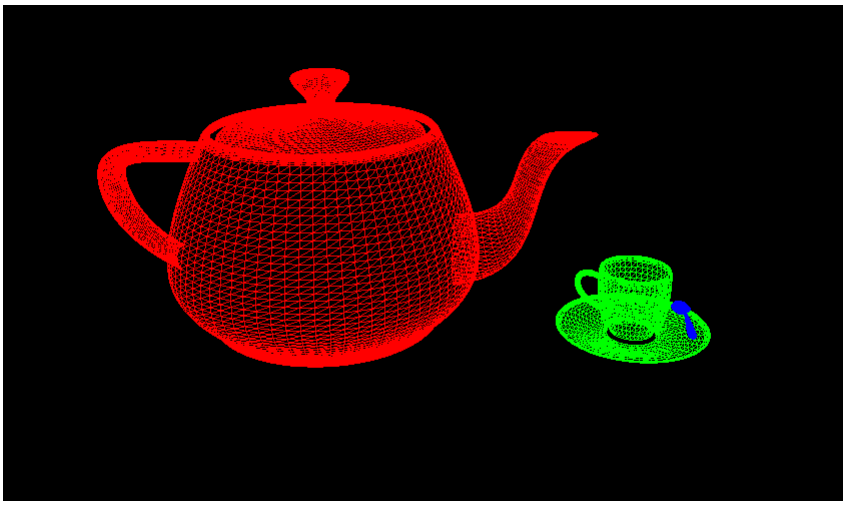

Figura 9: Visualização do teapot, teacup e teaspoon por linhas

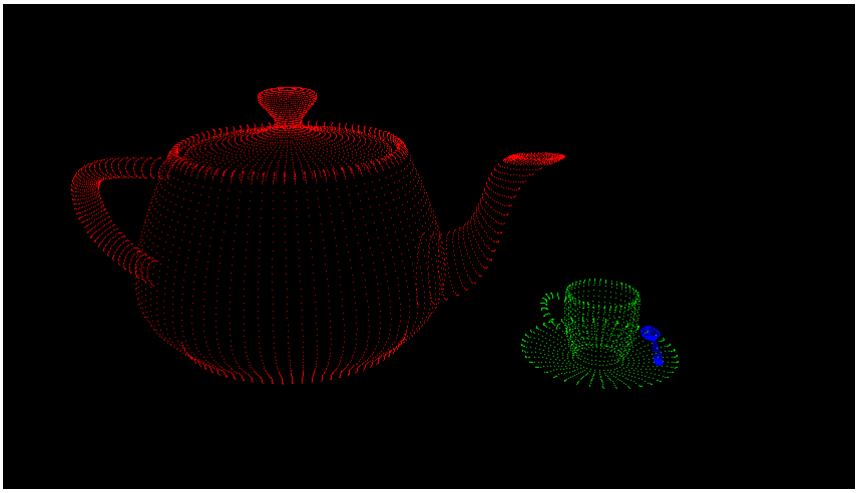

Figura 10: Visualização do teapot, teacup e teaspoon por pontos

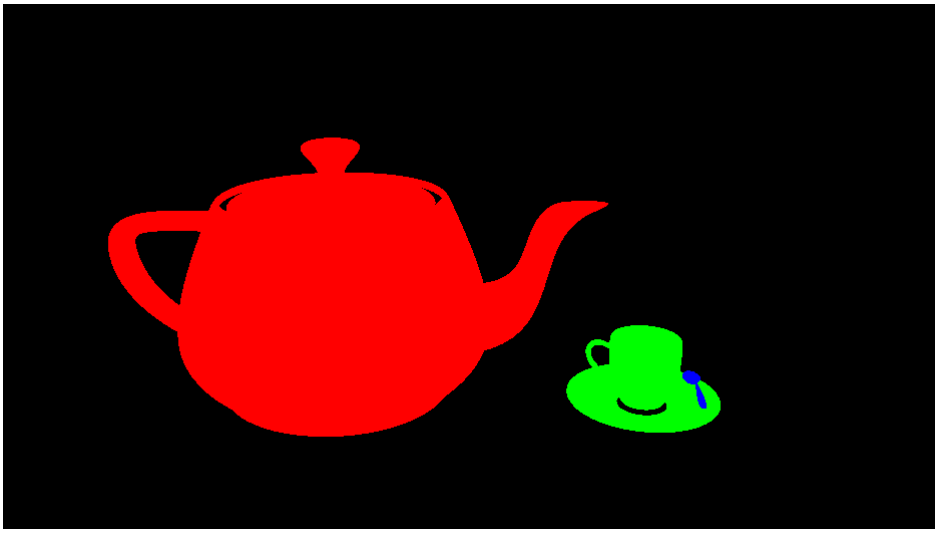

Figura 11: Visualização do teapot, teacup e teaspoon preenchidos

5.3 Sistema Solar

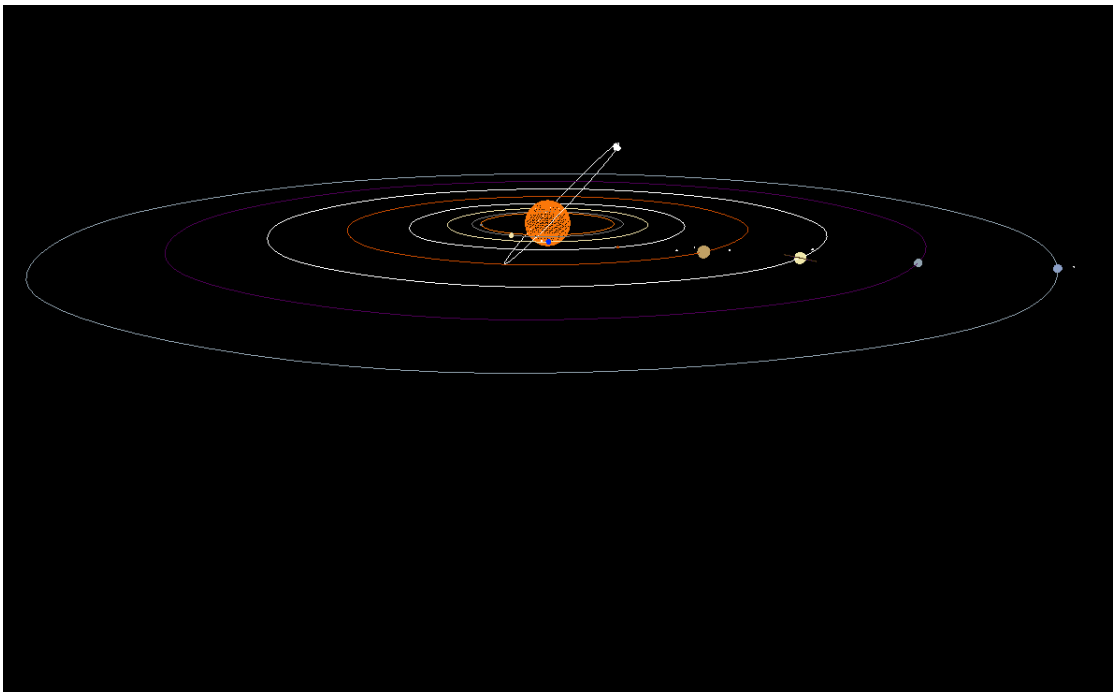

Figura 12: Visualização do Sistema Solar

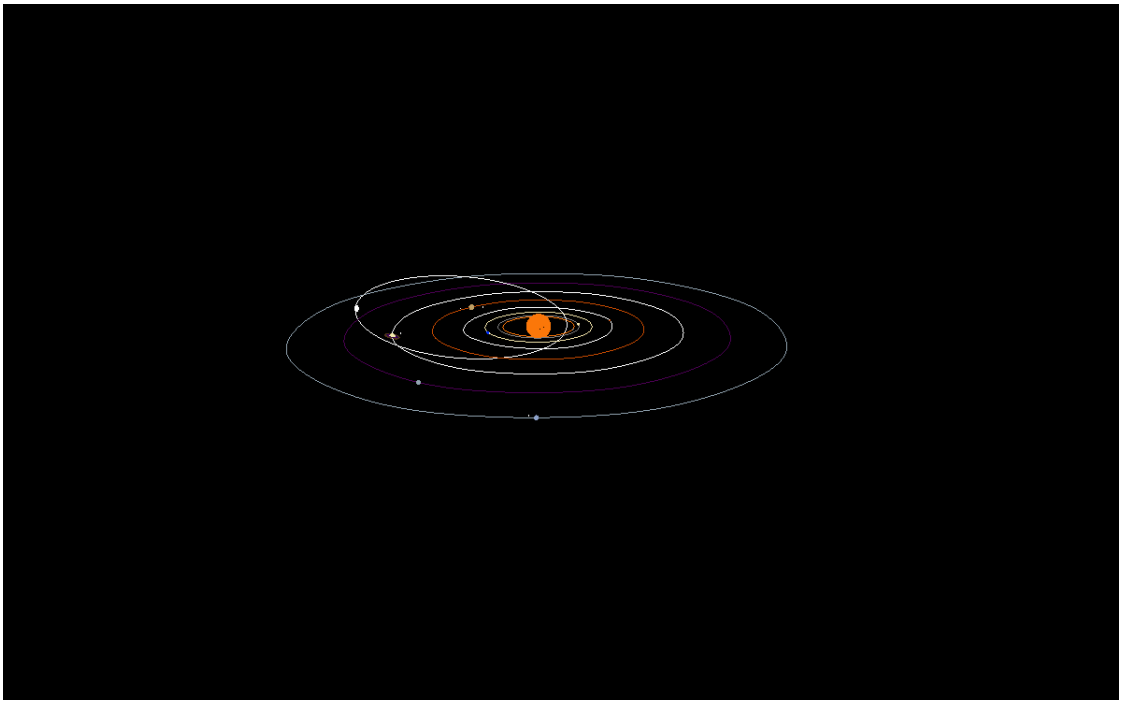

Figura 13: Sistema Solar visto de longe

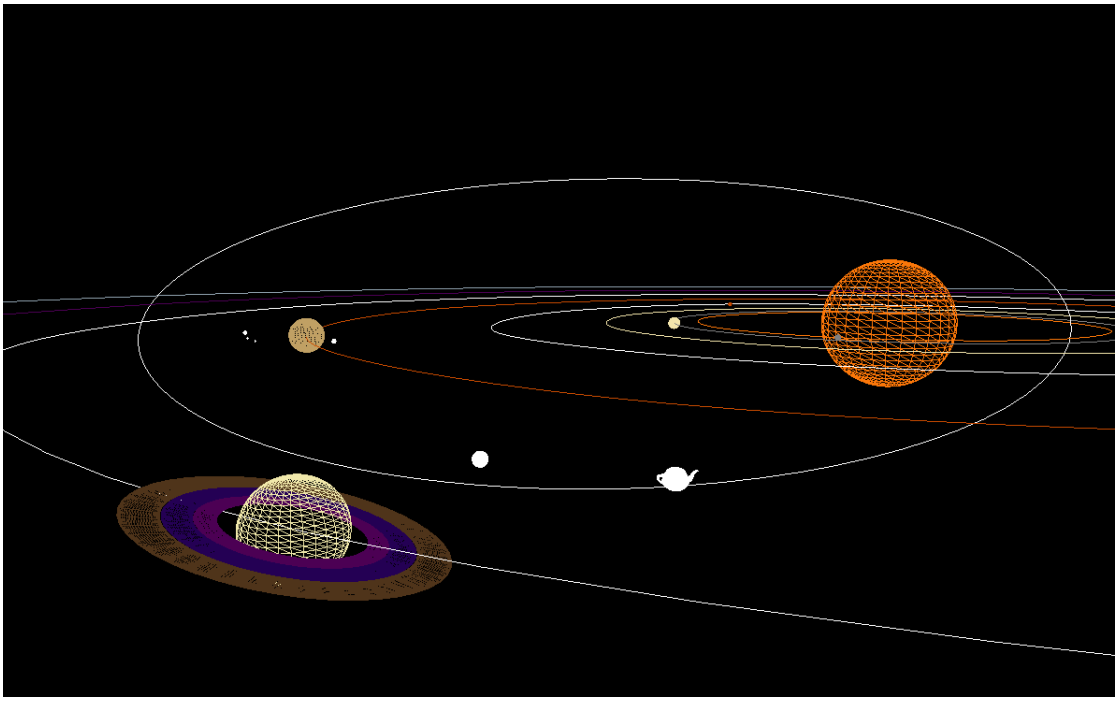

Figura 14: Visualização do no cometa(teapot)

6 Conclusão

A elaboração da terceira fase do projeto foi bastante mais complexa que a fase anterior. Isto deve-se principalmente aos diferentes algoritmos que tiveram de ser usados e ao baixo nível de conhecimento que tinhamos sobre como os deveriamos implementar. Durante a elaboração desta fase foi possivel compreender como usar os Bezier patches, as Catmutll-Rom curves e os VBOs e como consequência ganhar conhecimentos sobre estes.

O objetivo principal desta fase foi assim cumprido, já que foi desenvolvido um modelo do Sistema Solar que é agora dinâmico (como pedido no enunciado)

Concluimos assim a terceira fase do projeto, esperamos que a última fase permita acabar o projeto com um resultado final bastante agradável.