Projeto de Computação Gráfica - Fase 3

Diogo Braga A82547 João Silva A82005
Ricardo Caçador A81064 Ricardo Veloso A81919

20 de Abril de 2019

Resumo

Neste relatório é apresentada a terceira fase dum projeto no qual a intenção é desenvolver um mecanismo baseado em gráficos 3D e fornecer exemplos de uso que mostrem o seu potencial. Este projeto é desenvolvido no âmbito da unidade curricular de Computação Gráfica.

Conteúdo

1 Introdução **3**

2 Estrutura da Pasta do Projeto **4**

3 Parser XML **5**
3.1 Ficheiro 5
3.2 Funcionamento 6
3.3 Otimização de Leituras 6

4 Estruturas de Dados **8**

5 Generator **10**
5.1 Algoritmo de geração do modelo 10
5.1.1 Parsing do Patch 10
5.1.2 Criação da Superfície de Bezier 11
5.1.3 Cálculo dum Ponto 12

6 Engine **13**
6.1 Translação 13
6.1.1 Curvas de Catmull-Rom 13
6.2 Rotação 15
6.3 VBOs 15

7 Scenes **16**

8 Conclusão **17**

1 Introdução

Nesta terceira fase serão implementadas algumas funcionalidades tanto no *Generator* como no *Engine*, sendo estas:

1. Ler um ficheiro *patch* que irá conter os pontos de controlo de **Bezier** e criar o modelo respetivo (cometa).
2. Aprofundar as **translações** e as **rotações** alterando o ficheiro "XML"e o seu *parsing*.
   (a) As **translações** contendo os pontos de controlo que irão definir uma *Curva de Catmull-Rom* bem como uma variável *time* que representa o tempo (em segundos) que demora a percorrer a curva.
   (b) As **rotações** poderão ter o seu ângulo substituido por tempo sendo este o tempo que demora a realizar um rotação de 360°.
3. Desenhar os modelos utilizando VBOs aumentado assim a perfomance.

De seguida iremos apresentar figuras, algoritmos e as suas respetivas explicações de forma a ilustrar a realização das etapas apresentadas acima.

2 Estrutura da Pasta do Projeto

Para um entendimento mais claro da estrutura do projeto, achamos por bem referenciar a estrutura da pasta do projeto. O projeto entregue contêm para além do relatório, 4 pastas como é possível verificar na seguinte figura.

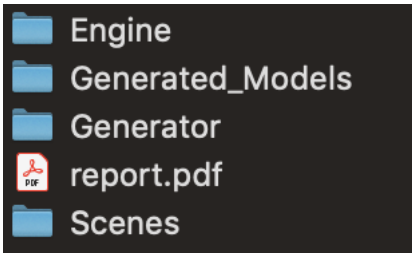

Figura 1: Estrutura da pasta.

Na pasta **Engine** residem os ficheiros relativos ao programa *engine*, bem como as bibliotecas (.h) criadas para o efeito. Contém ainda um ficheiro de configuração *cmake* e os ficheiros relativos à biblioteca *tinyxml2*.

Na pasta **Generator** residem os ficheiros relativos ao programa *generator*. Contém também um ficheiro de configuração *cmake*. Em relação à fase anterior acrescenta a existência de um ficheiro .cpp que irá servir como um parser para o ficheiro *patch* presente na pasta **Generated_Models**.

Na pasta **Generated_Models** residem os ficheiros que contêm os pontos gerados para cada figura, criados pelo programa *generator*. Relativamente à fase anterior acrescenta a existência de um ficheiro *patch* que irá conter os pontos de controlo.

Na pasta **Scenes** reside um ficheiro *XML* que contem a estrutura do sistema solar desenvolvido para esta fase do projeto.

3 Parser XML

3.1 Ficheiro

Devido aos requerimentos desta fase foi necessário alterar o ficheiro XML que provinha da segunda fase. As alterações aconteceram devido, por exemplo, à necessidade de incluir no ficheiro os pontos de controlo que, mais tarde gerados, dariam como resultado a curva de Catmull-Rom.

Importante referir que também nesta fase todos os sub-grupos herdam as transformações geométricas presentes no grupo ao qual pertencem. Nesta fase, estas transformações são: translações, rotações e escalas.

As escalas mantêm o mesmo funcionamento da fase anterior. Por outro lado, as translações e as rotações têm funcionamentos diferentes.

As translações funcionam tendo em conta um conjunto de 8 pontos que define uma curva de Catmull-Rom, e uma compontente *time* que define o tempo que o objeto demora a realizar a curva. No caso das rotações, estas podem estabelecer o ângulo através da componente habitual *angle*, ou através da componente *time*.

Na figura 2 é possível visualizar um excerto do ficheiro XML referente à Terra e à Lua.

```
<group> <!-- Earth & Moon -->
      <translate time="18.25">
            <point X="140" Y="0" Z="0" />
            <point X="98.88" Y="0" Z="-98.88" />
            <point X="0" Y="0" Z="-140" />
            <point X="-98.88" Y="0" Z="-98.88" />
            <point X="-140" Y="0" Z="0" />
            <point X="-98.88" Y="0" Z="98.88" />
            <point X="0" Y="0" Z="140" />
            <point X="98.88" Y="0" Z="98.88" />
      </translate>
      <group> <!-- Earth -->
            <rotate time="1" axisX="0" axisY="1" axisZ="0" />
            <models>
                  <model file="planet.3d" R="0.145" G="0.192" B="0.388" />
            </models>
      </group>
      <group> <!-- Moon -->
            <rotate angle="5.145" axisX="0" axisY="0" axisZ="1" />
            <translate time="1.35">
                  <point X="20" Y="0" Z="0" />
                  <point X="14" Y="0" Z="-14" />
                  <point X="0" Y="0" Z="-20" />
                  <point X="-14" Y="0" Z="-14" />
                  <point X="-20" Y="0" Z="0" />
                  <point X="-14" Y="0" Z="14" />
                  <point X="0" Y="0" Z="20" />
                  <point X="14" Y="0" Z="14" />
            </translate>
            <rotate time="27" axisX="0" axisY="1" axisZ="0" />
            <models>
                  <model file="moon.3d" R="0.412" G="0.388" B="0.376" />
            </models>
      </group>
</group>
```

Figura 2: Exemplo dum ficheiro usado no Parser.

3.2 Funcionamento

Tendo em conta as alterações referidas no ponto anterior modificamos também o funcionamento do *parser*.

1. Executa a função "parser_XML"que abre o ficheiro "scene.xml"e verifica se a leitura foi correta.
2. Caso se verifique inicia a função "parseGroup".
3. Nesta função alteramos o funcionamento para o caso do elemento *child* seja uma rotação ou uma translação.
4. Caso seja uma translação:
   (a) Obtém o valor *time* e atribui-o à figura.
   (b) De seguida itera sobre os 8 pontos definidos para a curva de Catmull-Rom retirando os valores X,Y e Z.
5. Caso seja uma rotação:
   (a) Obtém os valores X,Y e Z e atribui-os à figura.
   (b) De seguida, dependendo de qual o valor definido (*time* ou *angle*) atribuímo-lo à figura.
6. Por fim, retorna a árvore criada.

3.3 Otimização de Leituras

Ao longo dos vários testes realizados à cena que representa o sistema solar, reparamos que o tempo desde a invocação do programa **Engine** até ao aparecimento do sistema solar no ecrã era mais longo do que o comum. Notámos então que aquando do parsing do ficheiro XML, sempre que encontrávamos um nome do ficheiro abríamo-lo e e líamos o seu conteúdo.

Ora a operação de leitura de ficheiros não é tão eficiente quanto esperada pelo que necessitamos de criar uma estrutura que contivesse o nome do ficheiro, o número de triângulos presentes neste, e ainda o vetor contendo todos os pontos deste.

Desta forma sempre que fosse necessário ler informação dum ficheiro que já tivesse sido lido anteriormente, bastava aceder à estrutura retirar o necessário, poupando assim, operações de leitura em ficheiros desnecessárias.

A estrutura utilizada foi:

```
vector<pair<string,pair<int,vector<Point>>>> figure_points;
```

Figura 3: Estrutura utilizada para armazenar conteúdo de ficheiros.

A **string** usada representa o nome do ficheiro, o **int** representa o número de triângulos presentes no ficheiro, e o **vector**<**Point**> contém os pontos necessários para a figura.

O algoritmo usado, na função *loadFigure* para tratar a nova estrutra foi:

1. Verifica se o nome do ficheiro passado como parâmetro existe na estrutura:
   (a) Se sim:
      i. Copia o conteúdo lá existente para a nova figura.
      ii. Retorna a figura.
   (b) Se não:
      i. Lê o conteúdo do ficheiro.
      ii. Coloca conteúdo na nova figura.
      iii. Coloca conteúdo na estrutura para poder ser carregado posteriormente por figuras que usem o mesmo ficheiro.
      iv. Retorna a figura.

4 Estruturas de Dados

No seguimento da fase anterior, mantivemos a estrutura geral intacta pelo que continuamos a representar o sistema solar lido do ficheiro **XML** como uma árvore, onde o primeiro nodo representa o Sol e os planetas vão ser os seus *filhos*. Em consequência, as luas vão ser *filhos* dos planetas e *netos* do Sol. Tal estrutura pode ser vista na seguinte figura.

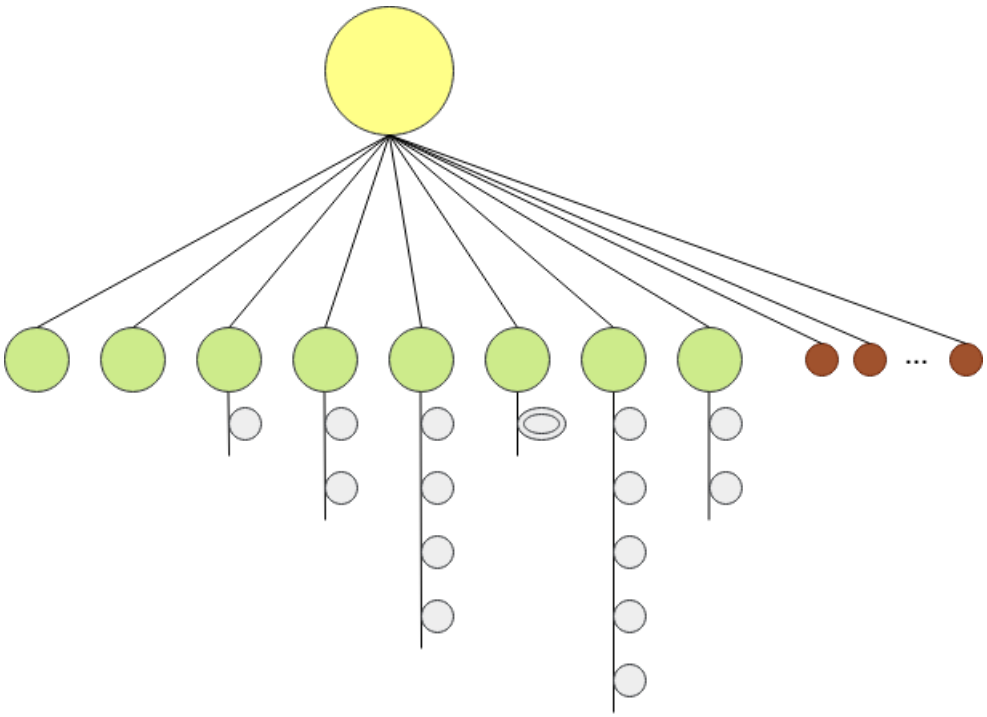

Figura 4: Visão geral da estrutura principal Tree.

Todas as principais classes necessárias mantiveram-se iguais à fase anterior à exceção de três, que são respetivamente a *Translation*, a *Rotation* e a *Figure*.

```
class Translation{
    public:
        Translation() :empty(true) {}
        bool empty;
        float time;
        vector<Point> points;
        float p[POINT_COUNT][3];
```

Figura 5: Estrutura Translation.

Foi necessário adicionar a variável **time** que representa o tempo que a trans-

lação vai demorar a realizar uma volta ao Sol. Foi também necessário substituir 3 variáveis que eram respetivamente **x, y, z** e representavam uma translação estática, por um **vector<Point>** que guarda os pontos de controlo necessários para a realização da curva de Catmull-Rom.

```
class Rotation{
    public:
        Rotation() :empty(true), angle(-1), time(-1) {}
        bool empty;
        float angle,x,y,z,time;
```

Figura 6: Estrutura Rotation.

Para a classe *Rotation* foi necessário adicionar uma variável **time** que define o tempo que uma rotação sobre o próprio deve demorar.

```
class Figure{
    public:
        string name;
        int num_triangles;
        vector<Point> points;
        Translation translation;
        Rotation rotation;
        Scale scale;
        Color color;
};
```

Figura 7: Estrutura Figure.

Por último à classe *Figure* foi adicionada uma variável **name** que representa o nome do ficheiro que continha os pontos para que a figura fosse construída. Esta variável foi necessária para distinguir qual das figuras era o cometa (explicado mais à frente), e para uma de otimização no parsing do XML (explicado anteriormente).

5 Generator

Para esta fase foi necessária uma evolução no **Generator**. Este tinha que ser capaz de criar um novo modelo baseado em **Patches de Bezier**.

Para proceder à realização dum modelo tendo em conta um patch de Bezier é preciso passar como parâmetro o ficheiro que contêm os pontos de controlo **(patch)**, o nível de **tessellation** e por último o **nome do ficheiro** onde será guardado o resultado.

```
./Generator bezier teapot.patch 10 cometa.3d
```

Figura 8: Exemplo de invocação.

O resultado que será guardado em ficheiro serão todos os pontos necessários para a criação dos triângulos que posteriormente modelam a superfície cúbica.

5.1 Algoritmo de geração do modelo

O algoritmo desenvolvido divide-se em 3 partes essenciais. A primeira é o **parsing do patch**, a segunda é a **criação da superfície de Bezier** que necessita da terceira parte que é o **cálculo dos pontos** da superfície.

5.1.1 Parsing do Patch

Para a fase parsing do patch tivemos em conta o habitual formato destes patchs, descrito na seguinte imagem.

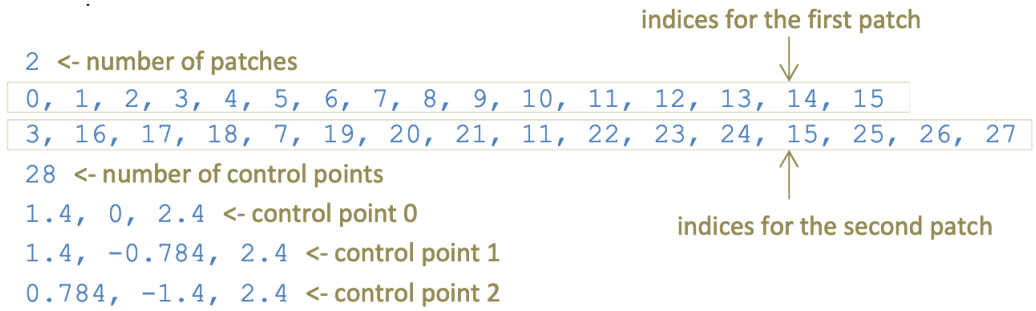

Figura 9: Exemplo dum formato dum patch.

O algoritmo utilizado para fazer parsing ao ficheiro foi:

1. Abre o ficheiro (patch) passado como parâmetro.
2. Lê a primeira linha (number os patches) e cria um array que irá conter cada patch (patches).
3. Realiza ciclo de leitura dos patches e armazena-os no array patches.
4. Lê linha seguinte aos patches que representa o número de pontos de controlo (number_control_points).
5. Realiza ciclo de leitura dos pontos, e coloca-los no array control_points.

5.1.2 Criação da Superfície de Bezier

Para criar a superfície de Bezier é preciso ter em conta a função de parsing apresentada anteriormente, e ainda a função de cálculo dum ponto que está presente na subsecção seguinte.

O algoritmo utilizado para a criação da superfície foi:

1. Faz parsing do ficheiro (patch).
2. Para cada patch existente (patch_number):
   (a) Para todos os níveis de tessellation (tessellation_v) entre 0 e tessellation, calcula-se:

$$v = \frac{tessellation\_v}{tessellation}$$

      i. Para todos os níveis de tessellation (tessellation_u) entre 0 e tessellation, calcula-se:

$$u = \frac{tessellation\_u}{tessellation}$$

      ii. Calculam-se os dois triângulos necessários, recorrendo ao algoritmo apresentado na subsecção seguinte (Cálculo dum Ponto):

      Primeiro triângulo:

      P1 $\Rightarrow$ (patch_number, u, v)

      P2 $\Rightarrow$ (patch_number, u + $\frac{1}{tessellation}$, v)

      P3 $\Rightarrow$ (patch_number, u + $\frac{1}{tessellation}$, v + $\frac{1}{tessellation}$)

      Segundo triângulo:

      P1 $\Rightarrow$ (patch_number, u + $\frac{1}{tessellation}$, v + $\frac{1}{tessellation}$)

      P2 $\Rightarrow$ (patch_number, u, v + $\frac{1}{tessellation}$)

      P3 $\Rightarrow$ (patch_number, u, v)

3. Escreve resultados em ficheiro.
4. Limpa memória.

5.1.3 Cálculo dum Ponto

Para que um ponto da superfície de Bezier seja calculado é necessário que sejam passados como parâmetros, o **número do patch** em que este se encontra, os pontos calculados anteriormente que se ligam à tessellation, **u** e **v**.

O algoritmo para o cálculo dum ponto foi:

1. Criámos 2 arrays, um para u e outro para v, e em cada índice dos arrays residem polinómios de Bernstein que têm em conta o índice em questão.

$$B_i = t^i(t-1)^{3-i}\binom{3}{i}, \forall i \in \{0,4\}, t \in \{u,v\}$$

```
bpu[4] = { powf(1-u, 3), 3*u*powf(1-u, 2), 3*powf(u, 2)*(1 - u), powf(u, 3) };

bpv[4] = { powf(1-v, 3), 3*v*powf(1-v, 2), 3*powf(v, 2)*(1 - v), powf(v, 3) };
```

2. Para cada índice de bpu:
   (a) Para cada índice de bpv:
      i. Calcula o índice do patch a ser usado.
      ii. Calcula o índice do ponto de controlo utilizando o índice calculado anteriormente.
      iii. Para cada coordenada do ponto a ser calculado, adiciona-se a multiplicação do respetivo ponto de controlo com os polinómios de Bernstein correspondentes a u e v.
3. Retorna o ponto a ser usado para a criação da superfície.

6 Engine

Nesta fase foi necessário evoluir o **Engine**, tanto no momento da aplicação da translação e da rotação, como no momento de desenhar os modelos. Nas secções seguintes vamos, portanto, abordar as mudanças que foram necessárias fazer em relação à fase anterior.

6.1 Translação

Tendo em conta que o enunciado requeria um modelo dinâmico com movimento dos planetas e dos seus satélites naturais, foi necessário estabelecer as curvas que estes seguiriam, ou seja, as órbitas.

Desta forma, a translação foi aplicada de maneira diferente, e tal vai ser abordado a seguir com a explicação das curvas de Catmull-Rom, pois foi este o tipo de curvas que foi estabelecido criar.

6.1.1 Curvas de Catmull-Rom

Uma curva de Catmull-Rom é uma curva que através da especificação de 4 pontos, desenha um troço entre 2 pontos. Nesta secção vamos explicar a forma como tal acontece.

1. Criação duma matriz dos pontos estabelecidos para a curva no ficheiro XML (8 no nosso caso), com a separação das compontentes de cada ponto ⇒ p[8][3]
2. Array que vai conter a posição de um ponto ⇒ pos[3]
3. Array que vai conter a derivada de um ponto ⇒ deriv[3]
4. Desenho da curva de Catmull-Rom:
   - Através do modo GL_LINE_LOOP, são desenhados segmentos de reta com auxílio da função getGlobalCatmullRomPoint;
   - Especificação dos vértices com 3 componentes;
5. getGlobalCatmullRomPoint:
   - Função que devolve os pontos da curva, que são espaçados num intervalo [0,1[⇒ global t
   - Estabelecimento do valor real do t, através do global t
   - Obtidos os indices dos próximos 4 pontos da matriz p que, em conjunto com o t, vão calcular o pos e o deriv através da função getCatmullRomPoint;
6. getCatmullRomPoint:
   - Matriz de Catmull-Rom ⇒ m

$$\begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{bmatrix}$$

- Multiplicação da matriz m pelo vetor que agrega cada componente dos 4 pontos anteriormente obtidos;
- Multiplicação da matriz resultante pela seguinte matriz, para cada compontente do posição do ponto ⇒ pos

$$\begin{bmatrix} t*t*t & t*t & t & 1 \end{bmatrix}$$

- Processo igual, mas multiplicado pela seguinte matriz, e a resultar para cada componente na derivada do ponto ⇒ deriv

$$\begin{bmatrix} 3*t*t & 2*t & 1 & 0 \end{bmatrix}$$

- Desenho do ponto / Translação do ponto:
  - Estabelecimento de um time (global t) utilizando a função glutGet(GLUT_ELAPSED_TIME);
  - Invocação da função getGlobalCatmullRomPoint com o time estabelecido, a variável pos, e a variável deriv;
  - Translação para o ponto usando o resultado da variável pos, utilizando a função glTranslatef;

Este processo apresentado é realizado em todos os objetos presentes no sistema solar. No entanto, e tendo em conta o enunciado, é tambem pedido que exista um cometa a realizar uma trajetória entre os planetas já existentes.

Além de executar o algoritmo anterior, o cometa realiza também um processo que faz com que este esteja sempre tangente à orbita. Tal faz sentido porque o cometa não tem rotação sobre si mesmo, e desta forma usa a sua cabeça como direção na qual se dirige. Também como requerido no enunciado, utilizamos o patch do teapot para representar o cometa no sistema solar.

Importante referir que todos os outros objetos no sistema solar realizam outro género de rotação, que vai ser apresentado depois desta secção.

O algoritmo também utilizado no cometa para o colocar tangente à curva é o seguinte:

1. Criação dum array que vai representar a matriz de rotação ⇒ m[16]
2. Criação do vetor up ⇒ { 0.0,1.0,0.0 }
3. Criação do vetor leftCometa[3];
4. Cálculo dos produtos externos necessários;
5. Normalização dos vetores em causa;
6. Estabelecimento da matriz de rotação que vai juntar as componentes dos vetores agora calculados de modo a preparar a rotação;
7. Aplicação das rotações na matriz para direção orientada à derivada;

Importante mencionar que decidimos tornar negativo o vetor que dá a direção ao objeto teapot, de forma a que o bico do teapot pudesse dar a sensação da cauda do cometa.

6.2 Rotação

O processo de rotação, na secção anterior já mencionado, é realizado o mais de acordo com a realidade possível. Portanto, o grupo achou boa ideia colocar os planetas a rodarem sobre si mesmos, diferente portanto do algoritmo utilizado no cometa.

Desta forma, a rotação (exceto no cometa) é aplicada da seguinte forma:

1. Caso o time esteja definido no ficheiro XML:
   (a) Estabelecimento do angle utilizado através do seguinte método glutGet(GLUT_ELAPSED_TIME)*360/(time*1000);
2. Rotação através do angle e das coordenadas x,y,z , utilizando a função glRotatef;

6.3 VBOs

Nesta fase é requerido que os modelos sejam desenhados com VBOs, ao invês do que acontecia nas fases anteriores.

Para tal, no momento do desenho das figuras é realizado o seguinte processo:

1. Geração de um *Vertex Buffer Objects*;
2. Especificação do objeto associado ao VBO gerado ⇒ *GL_ARRAY_BUFFER*
3. Alocação de memória para um array de floats v com o espaço necessário para os pontos da figura:
   - sizeof(float) × num_triangles × 3 (num_vertices) × 3 (num_coordinates)
4. Ciclo de carregamento dos pontos da estrutura de dados para o array de floats:
   - v[i++] = it->x;
   - v[i++] = it->y;
   - v[i++] = it->z;
5. Preenchimento do *GL_ARRAY_BUFFER* com os pontos do array v;
6. Transformação num array de vértices;
7. Desenho dos triângulos com base no array processado.

7 Scenes

Nesta fase do projeto é requerido um modelo dinâmico do sistema solar, com o sol, os planetas e os satélites naturais do mesmo. Além disso, o sistema solar deve incluir também um cometa com uma trajetória definida através duma curva de Catmull-Rom. Este cometa deve ser criado tendo por base os pontos de controlo do **Patch de Bezier** correspondente ao *teapot*. Desta forma o cometa terá, inevitavelmente, a forma dum *teapot*.

De seguida, nas figuras 10 e 11, são disponibilizados exemplos estáticos do sistema solar, onde é possível verificar a presença de todos os objetos requisitados.

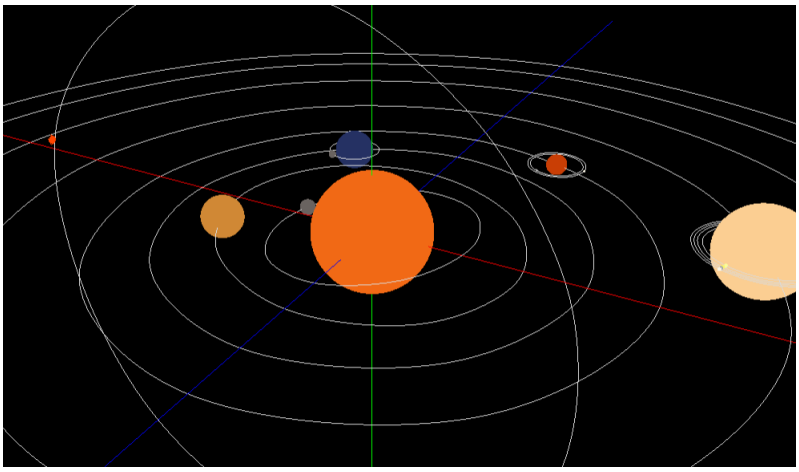

Figura 10: Possível perspetiva da cena.

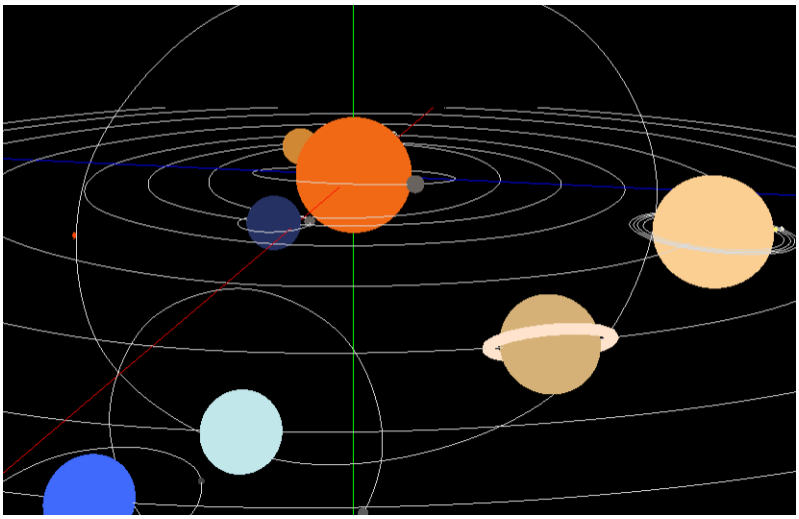

Figura 11: Possível perspetiva da cena.

8 Conclusão

Após a realização desta fase do projeto o grupo sente que o nível de confortabilidade relativamente às curvas *Catmull-Rom* aumentou exponencialmente.

Em relação à utilização de VBOs, tudo correu como planeado.

Existiu alguma dificuldade na realização da etapa referente ao *Generator*, mas através da matéria leccionada o grupo conseguiu reunir as condições necessárias para o bom entendimento das superfícies de **Bezier**.

Por fim, sentimos que o projeto está pronto para avançar para a última fase.