Relatório Terceira Fase Trabalho Prático - Computação Gráfica

Grupo 34 :
António Luís de Macedo Fernandes (a93312)
José Diogo Martins Vieira (a93251)
João Silva Torres (a93231)
Ricardo Lopes Santos Silva (a93195)

1 Maio 2022

Contents

1 Introdução **3**

2 Alterações no projeto **3**

2.1 Generator . . . 3

2.2 Engine . . . 3

3 Generator **3**

3.1 Superfícies de Bezier . . . 3

3.1.1 Leitura do ficheiro . . . 3

3.1.2 Curva de Bezier . . . 4

3.1.3 Criação das patches . . . 5

4 Engine **5**

4.1 Translação periódica em curva Catmull-Rom . . . 5

4.2 Rotação períodica . . . 6

4.3 VBO . . . 6

4.4 First Person Camera . . . 7

5 Resultados obtidos **7**

6 Sistema Solar dinâmico **8**

7 Conclusão **10**

1 Introdução

Nesta terceira fase do projeto da cadeira de Computação Gráfica foi necessário efetuar alterações no trabalho feito nas fases anteriores. Desta forma, tivemos de criar curvas de *Bezier* desenhar figuras através de patches. Para além disso tivemos de definir também curvas de *Catmull-Rom* para a translação periódica do cometa, em forma de bule.

Ao longo deste relatório vamos explicar de forma detalhada as mudanças realizadas, bem como o tudo o que foi acrescentado.

Numa fase final, teremos uma pequena secção de conclusões em que falaremos acerca da opinião geral do grupo.

2 Alterações no projeto

2.1 Generator

Para o generator, foi necessário a adição do modelo de bezier, para tal, para além do cálculo dos pontos, que será explicado posteriormente, foi preciso uma nova função de leitura de ficheiros devido à particularidade destes. Também foram simplificadas as restantes funções de geração de pontos, de modo a torná-las mais simples.

2.2 Engine

Para o engine, foi necessária a implementação de uma translação segundo um número de pontos, que serão feitas segundo uma curva de Catmull-Rom, assim como o número de segundos que demora a percorrer, e ainda o alinhamento, ou não, segundo a curva. Assim como, o número de segundos que demora a efetuar uma rotação. Também foi feita uma melhor organização e otimzação do engine, para tal, foram adicionados VBO's para o desenho dos modelos e foram criadas classes para lidar com o parser do XML e a câmara.

3 Generator

Nesta secção, vamos explicar com mais detalhe as alterações referidas anteriormente.

3.1 Superfícies de Bezier

3.1.1 Leitura do ficheiro

O ficheiro para as patches de bezier é dividido em duas partes:

- Número de patches e dos índices que as constituem
- Número de pontos e os mesmos

Assim, começamos por ler o número de patches, que indicará o número de linhas relativas às patches, para cada linha será feita a leitura dos índices. A leitura das patches será guardada num vector, que irá guardar para cada patch um vector de inteiros, correspondente aos índices.

A outra parte, será para os pontos. Esta parte é igual à dos outros ficheiros, dado um número de pontos, iremos guardar os pontos pela ordem que aparecem no ficheiro.

Tendo guardada toda a informação do ficheiro, passamos para o cálculo de cada patch. Cada patch tem 16 pontos, com estes 16 pontos, iremos dividir em conjuntos de 4, estes 4 pontos irão formar um curva de bezier.

3.1.2 Curva de Bezier

Para descobrir os pontos de uma curva de bezier, é usado a interpolação linear, que consiste num conjunto novo de pontos que se enconcontram entre dois pontos já conhecidos.

Para a explicação da descoberta dos pontos na curva de Bezier, temos a figura a baixo como referência.

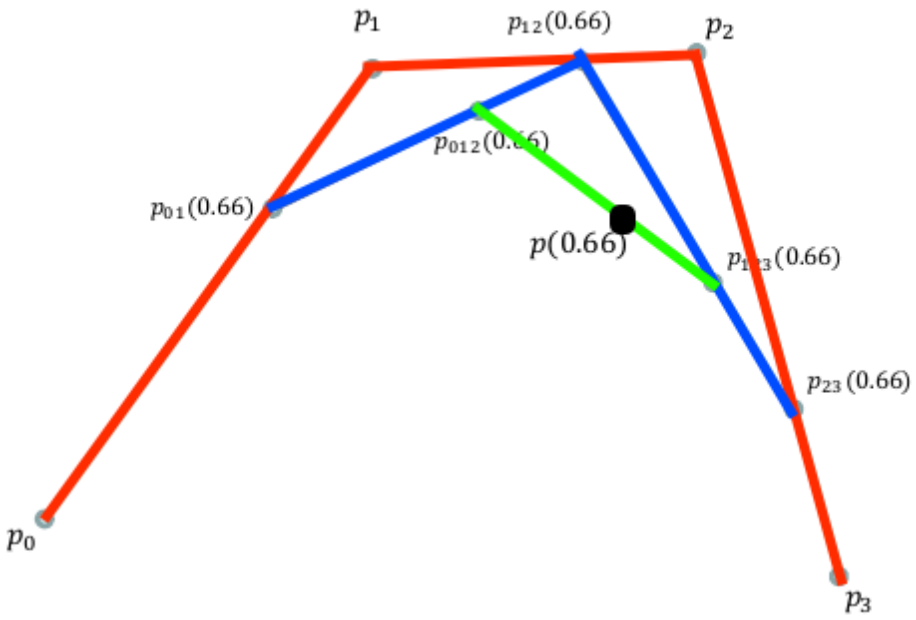

Figure 1: Curva de Bezier

Dados os pontos p0,p1,p2, e p3, conseguimos obter 3 novos conjuntos de pontos através da interpolação linear (linhas a vermelho), os pontos entre: p0 e p1, p1 e p2, p2 e p3.

Tenhamos agora um valor t, neste exemplo será 0.66, que será um valor entre 0 e 1, e nos indicará o ponto a obter para cada conjunto.

Obtemos assim 3 novos pontos, os pontos p01, p12 e p23. Com o mesmo raciocínio da interpolação linear, obtemos 2 conjuntos (linhas a azul).

Com o mesmo t, obtemos dois novos pontos p012 e p123, este dois pontos formam um novo conjunto (linha a verde). Com o mesmo t, iremos buscar o ponto final, que será assim um ponto da curva de bezier.

Transportando este raciocínio para a matemática, e sabendo que a interpolação linear entre dois pontos,p0 e p1, é dada por:

$p_{01}(t) = (1-t)p_0 + tp_1$

conseguimos traduzir a figura acima por:

$p_{01}(t) = (1-t)p_0 + tp_1$

$p_{12}(t) = (1-t)p_1 + tp_2$

$p_{23}(t) = (1-t)p_2 + tp_3$

$p_{012}(t) = (1-t)p_{01} + tp_{12}$

$p_{123}(t) = (1-t)p_{12} + tp_{23}$

$p(t) = (1-t)p_{012} + tp_{123}$

Conseguimos deduzir então a seguinte fórmula:

$p(t) = t^3P_3 + 3t^2(1-t)P_2 + 3t(1-t)^2P_1 + (1-t)^3P_0$

Passando para a forma matricial, temos:

$$p(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Figure 2: Cálculo do ponto na curva de bezier, através de matrizes

Faz-se o cálculo acima, e temos assim o ponto na curva de bezier.

3.1.3 Criação das patches

Sabendo como calcular os pontos da curva de bezier, podemos agora proceder para a criação das patches.

Dado um número inteiro, tesselation, que corresponde ao número de partes que cada patch terá, por exemplo, sendo 10, dará origem a uma patch de 10x10.

Para o cálculo da patch serão então necessárias duas variaveis, u e v, que começa em 0 e aumentam em $\frac{1}{tessselation}$. Para as 4 curvas da patch, dado um valor u, iremos obter o ponto respetivo para cada uma das curvas. Com esses 4 pontos, iremos formar uma nova curva, e para essa curva, dado um valor v, obtemos o ponto final que irá pertencer à patch.

No final de cada patch, após termos todos os pontos calculados, iremos fazer um double loop, para obter assim, os dois triângulos que formarão cada parte da patch.

4 Engine

Vamos, então, explicar as alterações feitas ao engine que mencionamos anteriormente com mais detalhe.

4.1 Translação periódica em curva Catmull-Rom

Também foi necessária a implementação de uma nova forma de translação. Neste caso, das translações com tempo, foi necessário recorrer às curvas de *catmull* para delinear a primitiva e estas seriam percorridas num intervalo de tempo estabelecido. Então, para respeitar estes requisitos foram considerados um conjunto de pontos, no mínimo 4 lidos do ficheiro *XML* para, então, se delinear a curva.

Assim, iremos adicionar uma nova variável à nossa classe de translação, que irá guardar um vetor de pontos. Ao inicializar esta classe, caso este vetor não seja vazio, iremos calcular os pontos necessários para desenhar a curva e guardá-los. Assim, este cálculo, é apenas feita uma vez, e a curva é desenhada posteriormente com o auxílio de `GL_LINE_LOOP`.

Para o tempo, iremos guardar uma variável para saber o tempo necessário para percorrer a curva. Iremos também ter uma variável no engine.cpp que é inicializada através da função `glutGet(GLUT_ELAPSED_TIME)`, que nos irá dizer o ínicio do programa. E a cada iteração do *renderScene*, iremos obter o valor do `glutGet(GLUT_ELAPSED_TIME)` e subtrair pelo guardado anteriormente, obtendo assim o tempo, este tempo,t, será passado à função de translação, e esse valor será normalizado através do seguinte cálculo:

```
   int start = t/time;
t -= start*time;
```

Irá depois ser passado à função, o valor, $\frac{t}{time}$. Onde time será a variável guardada que corresponde ao número de segundos necessário para realizar a translação.

A função getGlobalCatmullRomPoin irá calcular o ponto, para tal, calculamos o valor global de t, e através dele, vamos buscar os 4 pontos para o cálculo.

O cálculo do ponto é semelhante ao da bezier, muda a matriz para o cálculo. Neste caso, calculamos não só a posição, como também a direção, que será usada caso o align seja True.

$$p(t) = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_o \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

4.2 Rotação períodica

Através da gestão do tempo, já explicada anteriormente, iremos guardar uma varíavel *time*, que corresponderá ao número de segundos que serão necessários para fazer uma rotação completa. Assim, antes de efetuar a ação, verificamos se o valor do time é diferente de -1(valor default, significa sem rotação periódica) iremos obter o ângulo, mulitiplicando o valor de $\frac{t}{time}$, por 360, e após isto, aplicar a rotação.

4.3 VBO

Implementamos também os VBO's para o desenho de todos os models. Para tal, foi necessário uma reestruturação. Anteriormente, tínhamos uma variável `map<string, vector<Ponto>> mapFilesPontos`, que para cada ficheiro iria guardar a lista de pontos associada, tendo cada modelo depois uma referência para o vetor de Pontos que teria de desenhar, evitando assim a repetição da leitura e do gasto de memória desnecessário.

Assim, tornou-se mais simples a implementação dos VBOs.

Mudamos apenas o mapa para `map<string, pair<unsigned int,unsigned int>>`, onde o valor irá corresponder a um par onde o primeiro número é usado para a referência do buffer, e o segundo para o número total de pontos guardados. Adicionamos também ao model um par, `pair<unsigned int,unsigned int>`, que faz referência a um para do mapa.

Inicialmente no parser do XML, para cada ficheiro, o par corresponderá ao size do map até então (de modo a ter sempre um valor diferente) e de -1, que indica que ainda não foi efetuada a leitura do ficheiro. Assim para cada modelo, verifica-se se já foi efetuada a leitura (segundo elemento do par diferente de -1), caso sim, o par do model será o par do mapa. Caso não, será feita a leitura, guardando todos os pontos num vector de float, e no fim, tratamos do VBO.

Criamos o VBO com a glGenBuffer, com o endereço do primeiro elemento do par. Depois utilizamos as funções glBindBuffer e glBufferData para fazer a cópia do vetor.

Atualizamos o par, com o devido número de pontos. E fazemos correspondência do par do model com este.

Para o desenho, para cada model, verificamos o par associado. Fazemos o glBindBuffer, com o primeiro elemento do par, que corresponderá ao vetor que queremos desenhar, usamos glVertexPointer para indicar que um ponto serão 3 floats. E por fim, utilizamos a função glDrawArrays com o modo `GL_TRIANGLES`, para desenhar triângulos, e indicamos o número total de pontos, que será o segundo elemento do par.

4.4 First Person Camera

Nesta fase, implementamos também a first person camera, de modo a ser mais fácil navegar pelo sistema solar. Para tal, é necessário guardar um vector que irá corresponder à orientação da câmara, este vetor no íncio é fácil de determinar, pois será apenas, o lookAt - posição inicial.

Para manobrar a câmara, precisamos de associar dois ângulos, beta e alpha, que anteriormente foram descobertos para saber o ângulo do centro para a posição da câmara, neste caso é o contrário, para isso multiplica-se beta por -1 e alpha acrescenta-se PI.

Assim, há duas hipóteses para a câmara: olhar em redor, ou mover frente/trás(zoom).

Para olhar em redor, guardamos a posição inicial do X e Y, quando o utilizador clica no botão do lado esquerdo. E consoante a posição do rato, calculamos o deltaX e deltaY, que serão a diferença do X e Y da posição guardada para a atual. Com estes valores, calculamos o novo alpha e beta, e alteramos o vetor da direçao consoantes os valores.

Para o mover a câmara, verificamos apenas o deltaY, neste caso a direção não muda. Adicionamos à posição que estamos, o vetor direção multiplicado pelo valor do deltaY com a sensibilidade (esta pode ser alterada). Atualiza-se assim a posição.

5 Resultados obtidos

Nesta secção, iremos demonstrar e comentar os resultados obtidos para os ficheiros de teste desta fase.

Para o primeiro ficheiro obtemos o seguinte resultado:

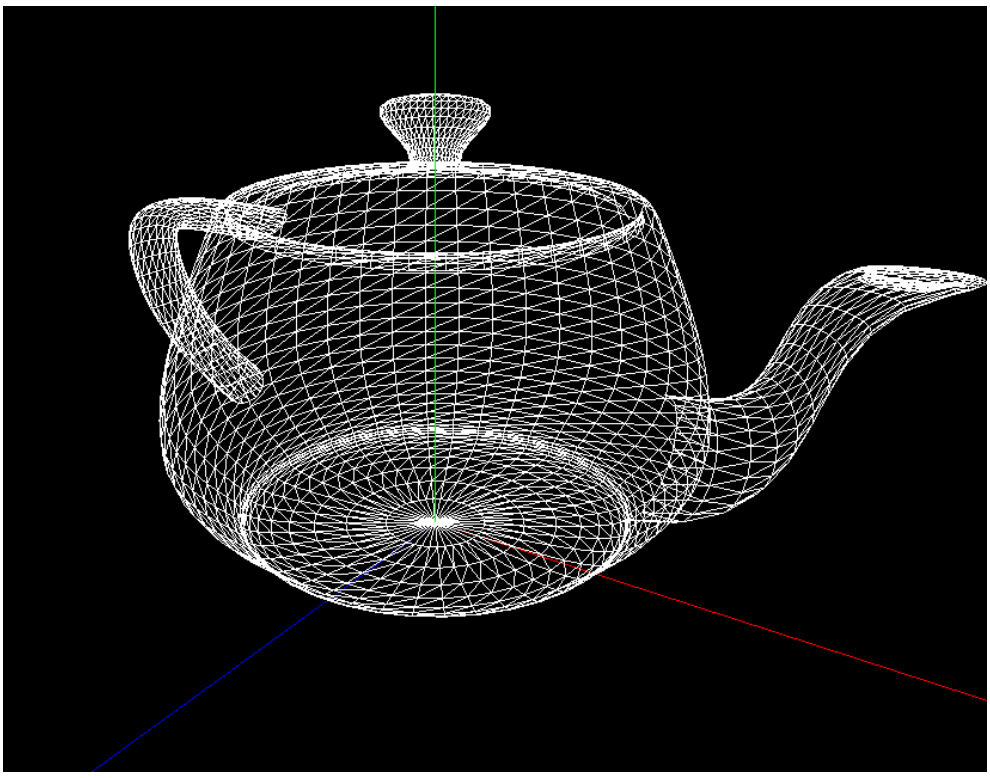

Figure 3: Primeiro ficheiro de teste

Para o segundo ficheiro de teste o resultado obtido foi o seguinte:

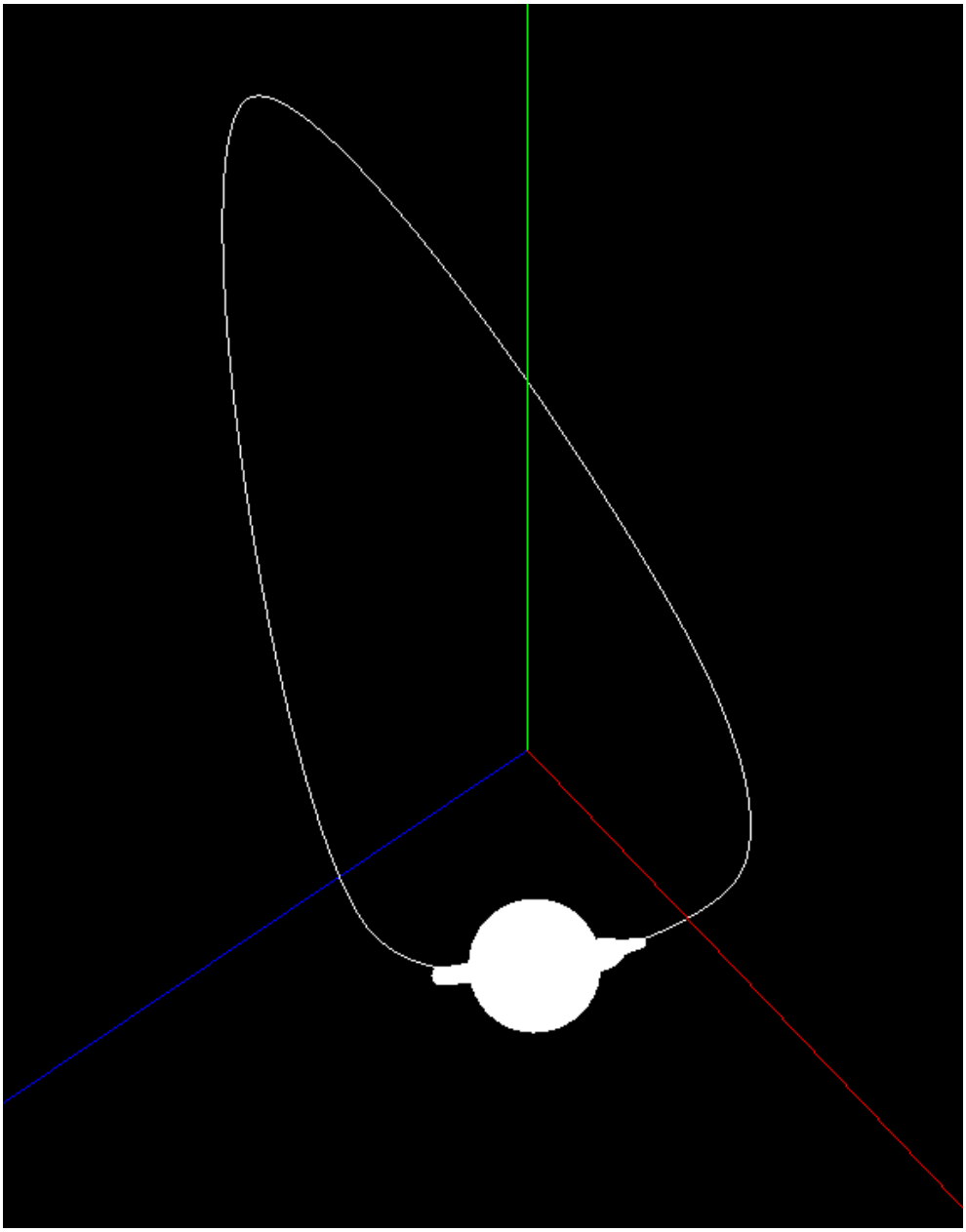

Figure 4: Segundo ficheiro teste

Com isto, estamos contentes com os resultados obtidos com estes ficheiros já que correspondem aos cenários objetivo, delineados pelos videos fornecidos.

6 Sistema Solar dinâmico

Quanto ao nosso sistema solar, criámos um cenário mais realista com os planetas e luas a girar sob a sua órbita, é de salientar que Vénus e Urano giram no sentido contrário aos outros planetas e isto também está representado no nosso cenário.

Adicioná-mos o cometa solicitado na forma de *teapot*. Este também cumpre a sua órbita.

No relatório não conseguimos demonstrar as órbitas e as rotações tanto dos planetas como das luas e da cintura de asteróides mas enviamos com o código o ficheiro *xml*.

Então o nosso cenário fica de acordo com a seguinte imagem:

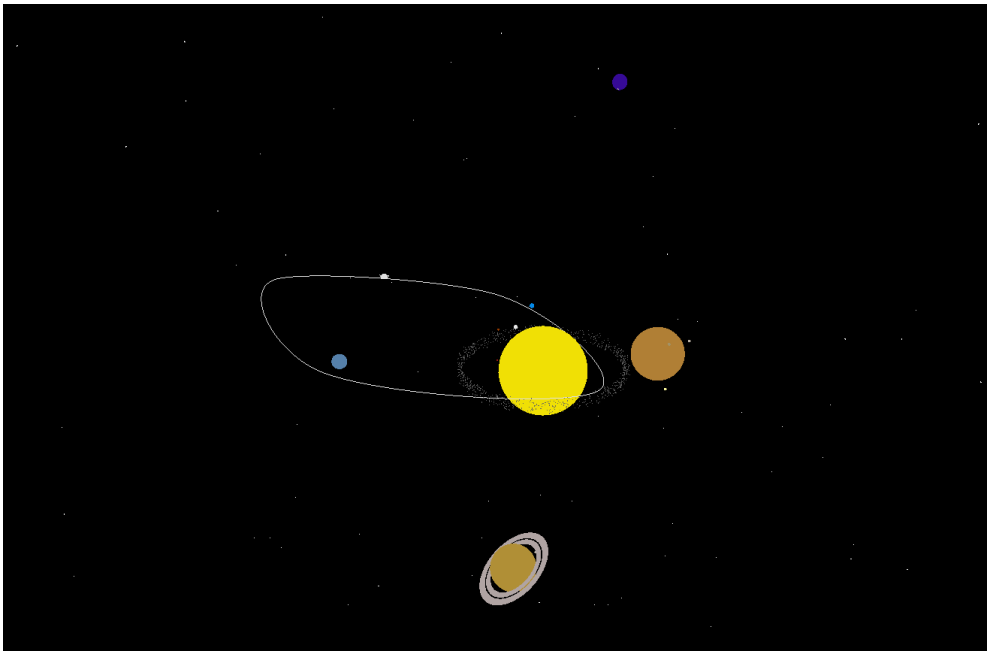

Figure 5: Sistema Solar

E a vista de cima do sistema fica da seguinte maneira:

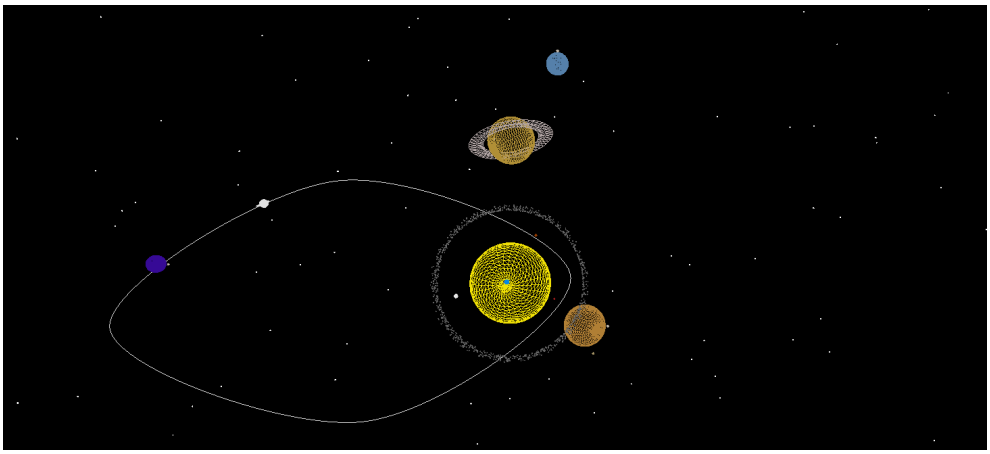

Figure 6: Vista de cima do Sistema Solar

7 Conclusão

Nesta terceira fase, consideramos que este projeto foi-nos útil para aprofundar o conhecimento adquirido em aula, nomeadamente na criação de curvas através dos métodos matemáticos lecionados nas aulas, bem como na otimização e aumento da performance do nosso projeto com a utilização de VBOs.

Por outro lado, a reestruturação do projeto achamos ter sido bem conseguida, pois o projeto ficou melhor organizado, tornando mais fácil a definição de novas funções e a procura de informação e conteúdo. Além disso, estamos contentes com o produto final do nosso sistema solar no qual conseguimos adicionar o que foi proposto, colocando os planetas e as luas a girar de acordo com a órbita delineada e tendo adicionado com sucesso o cometa pedido.

Assim, do nosso ponto de vista esta terceira fase teve um aproveitamento positivo, as dificuldades sentidas foram ultrapassadas, os requisitos propostos cumpridos, os extras consideramos relevantes e na próxima fase há aspetos a melhorar que são alcançáveis.