Universidade do Minho
Escola de Engenharia

Fase III
Curvas, Superfícies Cúbicas e VBO's

Trabalho Prático
Computação Gráfica

Grupo 48

Gabriela Santos Ferreira da Cunha - a97393

João Gonçalo de Faria Melo - a95085

Nuno Guilherme Cruz Varela - a96455


a97393


a95085


a96455

maio, 2023

Conteúdo

1 Introdução **3**

2 Cintura de Asteroides **3**

3 Superfícies de Bezier **3**
3.1 Leitura do Ficheiro 3
3.2 Cálculo dos Pontos 4

4 Curvas Catmull-Rom **5**

5 Transformações Dinâmicas **6**
5.1 Rotação Dinâmica 6
5.2 Translação Dinâmica 7

6 Extensões Adicionadas **8**
6.1 Geração de Pontos de uma Elipse 8
6.2 Visualização das Curvas 8
6.3 Sentido da Rotação 8
6.4 Nível de Tesselação 8
6.5 Etiqueta do Grupo 9

7 Implementação dos VBO's **9**

8 Animação do Sistema Solar **10**
8.1 Órbitas 10
8.1.1 Geração de Pontos 10
8.2 Cometa 11
8.3 Movimentação dos Elementos 11

9 Experiência do Utilizador **13**
9.1 Modo Explorador 13
9.2 Modo *Debug* 14

10 Resultados Finais **15**
10.1 Sistema Solar 15
10.2 Testes 16

11 Conclusão **17**

1 Introdução

No âmbito deste projeto, foi-nos proposto o desenvolvimento de um mecanismo 3D baseado num cenário gráfico, dividido em 4 fases.

O presente relatório é referente à terceira fase do trabalho, onde o objetivo da mesma passava pela inclusão de curvas cúbicas e superfícies de Bezier ao projeto, de forma a acrescentar dinamismo aos cenários, que até agora eram estáticos. O cenário proposto para esta etapa consiste num modelo dinâmico do sistema solar, englobando um cometa construído através de *patches* de Bezier com uma trajetória definida por uma curva Catmull-Rom.

2 Cintura de Asteroides

Como descrevemos no relatório da fase passada, um dos nossos objetivos passava por melhorar a eficiência da nossa cintura de asteroides. Anteriormente, esta era criada com recurso a uma extensão no *engine*, pelo que criavamos n VBO's na existência de n asteroides. Para solucionar este problema, decidimos passar a considerar uma cintura como uma primitiva gráfica - "belt".

Com isto, foi necessário voltar a recorrer aos elipsoides para representar os asteroides. Os parâmetros necessários para a criação da cintura correspondem aos atributos da extensão XML anteriormente usada (n, $radiusIn$, $radiusOut$, $height$ e $seed$). Para representar asteroides de diferentes tamanhos, são ainda recebidos como parâmetros os valores mínimos e máximos de comprimento de cada um para limitar a escala.

Depois de gerados os pontos pelo *generator*, o ficheiro que contém toda a cintura será lido pelo *engine* e a cintura passa a ser tratada como qualquer outra primitiva gráfica, ou seja, é preciso apenas um VBO para a sua renderização. De notar que esta alteração aumentou de forma significativa a eficiência do *engine* na presença de uma cintura.

3 Superfícies de Bezier

Nesta fase, o *generator* deve ser capaz de gerar modelos com base em superfícies de Bezier e, portanto, apto para ler um ficheiro que contém todos os *patches* e *control points* necessários à construção do modelo.

3.1 Leitura do Ficheiro

O ficheiro que contém os *patches* e os pontos de controlo possui uma estrutura específica que facilita a sua leitura.

A primeira linha do ficheiro indica o número, n, de *patches*. As seguintes n linhas contêm todos os *patches* do modelo, em que cada um é um conjunto de 16 índices para os pontos de controlo. De notar que como estamos perante curvas cúbicas de Bezier, são necessários 4 pontos para definir uma curva. Após os *patches* de Bezier, o ficheiro possui uma linha com o número de pontos de controlo. Finalmente, as restantes linhas estão reservadas para estes pontos de controlo e cada uma é um conjunto de 3 *floats* que formam um ponto.

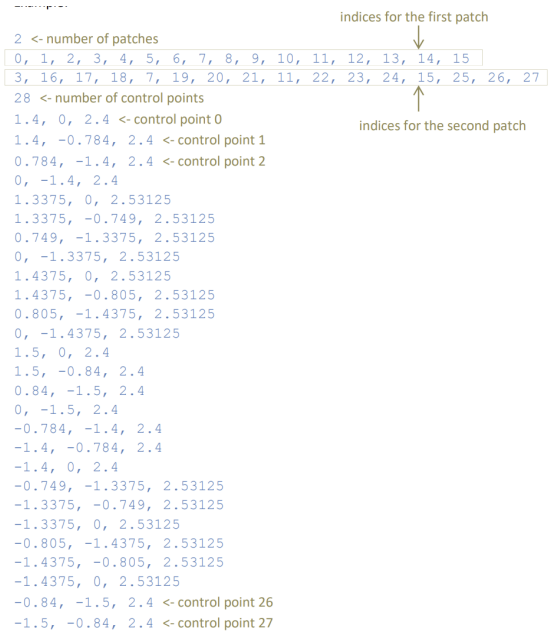


Figura 1: Exemplo de ficheiro com legenda.

No final da leitura do ficheiro, geramos uma estrutura com toda a informação dos *patches* presentes no ficheiro para nos facilitar no cálculo dos pontos.

3.2 Cálculo dos Pontos

Cada *patch* de Bezier é uma espécie de matriz composta por 16 pontos que vai ser iterada por dois valores, u e v, compreendidos entre 0 e 1. Um *patch* é ainda caraterizado pelo seu nível de tesselação que nos indica o número de divisões da nossa matriz. Desta forma, este nível acaba por influenciar na definição da superfície, isto é, quanto maior for o nível de tesselação, maior será a sua definição. Isto ocorre porque o "salto" dado pelas variáveis u e v é menor e, por isso, vão existir mais iterações, gerando um maior número de pontos.

Deste modo, resta-nos saber calcular qualquer ponto dado um u e v, ou seja, $p(u, v)$.

Consideremos as matrizes $M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ e $P = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$,
sendo esta última a matriz que contém os 16 pontos de um *patch*.

Podemos calcular qualquer ponto $p(u, v)$ através da seguinte fórmula:

$$p(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} MPM^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

A transposta da matriz M é igual à própria matriz, de onde podemos assumir que $M = M^T$. Para cada *patch* a expressão MPM^T é constante, pelo que a matriz resultante dessas multiplicações pode ser computada previamente no cálculo dos pontos de cada *patch*.

Desta maneira, conseguimos calcular todos os pontos de uma superfície de Bezier com um certo nível de tesselação e, consequentemente, gerar os triângulos para construir o modelo.

4 Curvas Catmull-Rom

Um dos objetivos desta fase passava por gerar curvas cúbicas Catmull-Rom, algo que nos viria a ser útil para animar as translações. Para tal, vimos a necessidade de calcular os pontos desta curva.

De forma a calcular os pontos, recorremos à seguinte fórmula, em que t determina a posição atual na curva e P_0, P_1, P_2 e P_3 são os quatros pontos necessários para a construção da curva, uma vez que estamos perante uma curva cúbica.

$$p(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Deste modo, conseguimos calcular qualquer ponto da curva dado um certo t. Podemos ainda calcular o vetor tangente à curva em qualquer ponto desta com a seguinte equação.

$$p'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Este vetor tangente será útil para alinhar um dado objeto com a curva, algo que será abordado na secção seguinte.

5 Transformações Dinâmicas

Nesta fase, o *engine* deve ser capaz de albergar transformações dinâmicas. Mais concretamente, este poderá receber rotações e translações estáticas e dinâmicas, sendo que se forem dinâmicas, estas devem passar a funcionar de acordo com o tempo fornecido como parâmetro.

Desta forma, tivemos de fazer alterações a nível da arquitetura das transformações, uma vez que para cada tipo de transformação vamos ter comportamentos diferentes dependendo dos parâmetros recebidos no ficheiro XML. A atualização na hierarquia das transformações pode ser descrita pela figura seguinte:

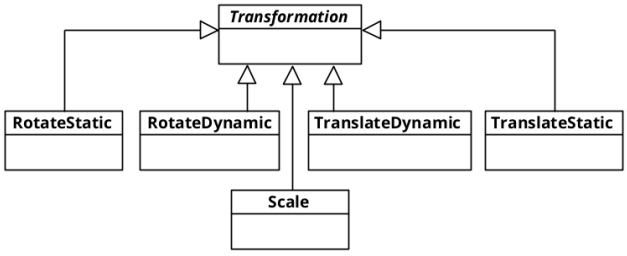


Figura 2: Hierarquia de transformações.

5.1 Rotação Dinâmica

A diferença entre a rotação estática e a rotação dinâmica traduz-se na substituição do parâmetro do ângulo de rotação por um determinado número de segundos, que significa o tempo que o objeto vai demorar a completar uma rotação.

```
<rotate time="10" x="0" y="1" z="0" />
```

O ângulo utilizado para aplicar a rotação vai, assim, depender da passagem do tempo, dada pela expressão "glutGet(GLUT_ELAPSED_TIME)". O valor obtido é dividido por 1000, o que nos permite passar para a unidade temporal de segundos. O resultado é ainda dividido pelo valor que demora a completar uma rotação.

5.2 Translação Dinâmica

Para efetuar uma translação dinâmica, é necessário um conjunto de pontos que definem uma curva Catmull-Rom e um determinado número de segundos para percorrer toda a curva. A extensão tem ainda um atributo "align" que especifica se o objeto deve ou não estar alinhado com a trajetória da curva. Esta deverá ter a seguinte estrutura:

```
<translate time="10" align="True" />
    <point x="1" y="0" z="1" />
    <point x="0.707" y="0.707" z="1/>
    <point x="0" y="1" z="1" />
    ...
    <point x="-1" y="0" z="1" />
</translate>
```

De notar que a curva deverá ter no mínimo 4 pontos, dado que estamos perante uma curva cúbica.

Relativamente à forma como lidamos com a aplicação desta transformação, primeiramente, começamos por desenhar a trajetória da curva. Para tal, calculamos os pontos da curva como explicado na secção 4. A extensão desta translação é ainda capaz de receber o nível de tesselação que será usado neste cálculo dos pontos.

Seguidamente, realizamos a translação para o ponto da curva onde o modelo deverá estar situado. Assim, utilizamos novamente a expressão que nos permite calcular o ponto da curva, mas desta vez o argumento que nos indica a sua posição na curva deverá ser calculado tendo em conta a passagem do tempo.

Por fim, caso o objeto deva estar alinhado com a curva, teremos de realizar uma rotação. Ao contrário das transformações habituais, vamos calcular a matriz da rotação para a aplicar. Deste modo, necessitamos de calcular os eixos. Tal como explicado mais acima, viríamos a necessitar do vetor tangente à curva num dado ponto da curva, que será atribuído ao eixo x. De seguida, o eixo z é calculado através do produto vetorial entre o eixo x e eixo y na iteração anterior. O novo eixo y é dado pelo produto vetorial do eixo z com o eixo x. É importante referir que todos os vetores destes eixos são normalizados e, inicialmente, o eixo y assume o valor de $(0, 1, 0)$.

A matriz resultante é obtida com base nestes eixos calculados. No entanto, é importante transformar esta matriz na sua transposta para obedecer ao formato utilizado pelo OpenGL. Finalmente, utilizamos o "glMultMatrixf" que nos permite aplicar esta transformação à matriz das transformações guardada.

6 Extensões Adicionadas

6.1 Geração de Pontos de uma Elipse

Esta extensão é responsável por gerar n pontos de uma elipse com semieixos de comprimento a e b e um ponto central, pertencente ao plano $x0z$, de coordenadas x e z. Pode, ainda, receber como parâmetro dois ângulos de rotação compreendidos entre 0 e 360 graus em relação aos eixos x e z, de forma a que a elipse adquira alguma inclinação sobre o plano de equação $y = 0$, revelando-se bastante útil para a representação das órbitas.

```
<ellipsepoints n="15" a="22.7" b="30.27" xRotationAngle="40" yRotationAngle="60"/>
```

Figura 3: Exemplo de utilização da extensão "ellipsepoints".

6.2 Visualização das Curvas

Com o objetivo de ocultar a linha de trajetória de qualquer curva Catmull-Rom, adicionamos, às translações dinâmicas, um novo atributo - "show" - que tem um funcionamento semelhante ao de um *bool*. Assim sendo, podemos ocultar uma dada trajetória concedendo um valor de "False" a este atributo, como representa a figura 4.

```
<translate time="25.2" align="True" show="False">
```

Figura 4: Exemplo de utilização do atributo "show".

6.3 Sentido da Rotação

Relativamente à rotação dinâmica de um grupo, podemos definir o sentido desta através da extensão "clockwise', que funciona também como um *bool*. Assim, caso se pretenda que a rotação se efetue de acordo com os ponteiros do relógio, este atributo deve tomar o valor de "True", sendo que, por predefinição, o sentido da rotação é o contrário.

```
<rotate time="8.2" x="0" y="1" z="0" clockwise="True"/>
```

Figura 5: Exemplo de utilização do atributo "clockwise'.

6.4 Nível de Tesselação

A translação dinâmica pode receber o nível de tesselação com que a trajetória da curva deve ser desenhada, sendo por predefinição o nível 1000. Este parâ-

metro traduz a definição da curva, na medida em que quanto maior for o seu valor, mais definida é a curva.

```
<translate time="300" align="True" show="true" tesselation="1000">
```

Figura 6: Exemplo de utilização do atributo "tesselation'.

6.5 Etiqueta do Grupo

Para facilitar a identificação de um grupo, pretendemos que o utilizador possa atribuir uma espécie de legenda ou etiqueta ao mesmo. Foi, portanto, adicionado um atributo "label" à extensão "group".

```
<group label="Earth">
```

Figura 7: Exemplo de utilização do atributo "label'.

7 Implementação dos VBO's

Os VBO's já foram previamente implementados na fase 1 do projeto. No entanto, aproveitamos para falar sobre aspetos mais relacionados com a sua implementação.

Após a inicialização do "GLEW", criamos todos os VBO's para os modelos que foram lidos do ficheiro XML. Para cada modelo são criados dois *buffers*, um para guardar todos os vértices do modelo e outro para guardar os índices. Sempre que pretendemos inserir dados no *buffer* recorremos à sequência de funções "glBindBuffer" e "glBufferData".

```
glBindBuffer(GL_ARRAY_BUFFER, vertices_buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * points.size(), points.data(), GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexes_buffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(int) * indexes.size(), indexes.data(), GL_STATIC_DRAW);

nIndexes = indexes.size();
```

Figura 8: Excerto do método "load".

Para renderizar um modelo, este dois *buffers* irão ser chamados através da função "glBindBuffer". Para o *buffer* que contém os pontos definimos a sua semântica e para índices utilizamos o "glDrawElements" que vai desenhar todos os triângulos presentes no *buffer*.

```
void Model::draw() {
    glPushAttrib(GL_CURRENT_BIT);

    if (color != NULL) {
        glColor3f(color->x / 255.0, color->y / 255.0, color->z / 255.0);
    }

    glBindBuffer(GL_ARRAY_BUFFER, vertices_buffer);
    glVertexPointer(3, GL_FLOAT, 0, 0);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexes_buffer);
    glDrawElements(GL_TRIANGLES, nIndexes, GL_UNSIGNED_INT, NULL);

    glPopAttrib();
```

Figura 9: Método "'draw".

8 Animação do Sistema Solar

8.1 Órbitas

As órbitas, anteriormente representadas por torus, são agora definidas por uma curva Catmull-Rom, pelo que pudemos passar a representá-las através de elipses, ao invés de circulares, aproximando a nossa proposta de representação do sistema solar à realidade.

8.1.1 Geração de Pontos

Para gerar os pontos da elipse que representa a órbita, empregamos a extensão "ellipsepoints", já explicada no relatório.

Sabemos que para obter os pontos de uma elipse situada no plano $x0z$ com n pontos, recorremos às equações paramétricas da elipse:

$$\begin{cases} x = a \times cos(j \times \frac{2\pi}{n}) \\ y = 0 \\ z = b \times sin(j \times \frac{2\pi}{n}), \end{cases}$$

onde $j_0 = 0$ e j é incrementado em 1 unidade a cada ponto, tomando o valor de n para o último.

A partir da fórmula base e dados o centro da elipse $(x_0, 0, z_0)$ e os ângulos de rotação sobre o eixo do y, y_{rot}, e sobre o eixo do x, x_{rot}, podemos calcular os n pontos da nova elipse, a partir do sistema:

$$\begin{cases} x = x_0 + a \times cos\ y_{rot} \times cos(j \times \frac{2\pi}{n}) - b \times sin\ y_{rot} \times sin(j \times \frac{2\pi}{n}) \times cos\ x_{rot} \\ y = b \times sin(j \times \frac{2\pi}{n}) \times sin\ x_{rot} \\ z = z_0 + a \times sin\ y_{rot} \times cos(j \times \frac{2\pi}{n}) + b \times cos\ y_{rot} \times sin(j \times \frac{2\pi}{n}) \times cos\ x_{rot} \end{cases}$$

8.2 Cometa

Uma das metas relacionadas com a animação do sistema solar consistia na criação de um cometa através de *patches* de Bezier com uma trajetória definida por uma curva Catmull-Rom.

O cometa foi feito utilizando 8 *patches* e cada *patch* viria a ter os pontos presentes na figura 10. Utilizamos esta técnica para cada octante com um $a = 0.5$, o que resultou numa espécie de uma esfera.

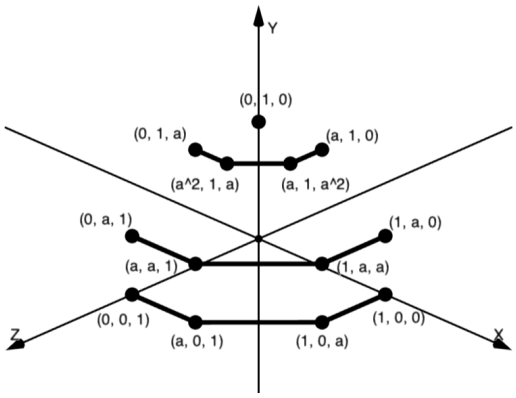


Figura 10: Pontos dos *patches* do cometa.

Um dos nossos objetivos passava por replicar o rastro do cometa. Para isso pegamos na "esfera" resultante dos *patches* de Bezier e alteramos a componente do y do ponto $(0, 1, 0)$ da figura.

Em relação à trajetória do cometa, utilizamos uma translação dinâmica para percorrer uma certa trajetória. Esta trajetória foi calculada através da extensão "ellipsepoints", para gerar uma órbita elíptica em volta do sistema solar. De notar que, não utilizamos a escala real de um cometa, uma vez que o modelo iria ficar muito pequeno e não se iria conseguir vê-lo.

8.3 Movimentação dos Elementos

Através da implementação das transformações dinâmicas, é possível visualizar os planetas, os satélites e o cometa a movimentarem-se ao longo das suas órbitas, com uma translação dinâmica, e os mesmos, à exceção do cometa, a rodarem em torno do seu próprio eixo, com uma rotação dinâmica.

A partir de uma pesquisa, foram recolhidos os seguintes dados, relativos à duração a completar uma volta à órbita e em torno de si mesmo de cada planeta e de cada satélite, respetivamente.

Planeta	**Período orbital**	**Período de rotação**
Mercúrio	88	59
Veńus	225	243
Terra	365	1
Marte	687	1.03
Júpiter	4333	0.41
Saturno	10759	0.45
Urano	30688	0.72
Neptuno	60190	0.67

Tabela 1: Períodos orbitais e de rotação de cada planeta em dias terrestres.

Planeta	**Satélite**	**Período orbital**	**Período de rotação**
Terra	Lua	27.32	27.32
Marte	Fobos	0.32	0.32
Marte	Deimos	1.26	1.26
Júpiter	Io	1.77	1.77
Júpiter	Europa	3.55	3.55
Júpiter	Ganimedes	7.16	7.16
Júpiter	Calisto	16.69	16.69
Saturno	Titã	15.94	15.94
Saturno	Reia	4.52	4.52
Saturno	Tétis	1.89	1.89
Saturno	Dione	2.74	2.74
Saturno	Encélado	1.37	1.37
Saturno	Iapetus	79.32	79.32
Saturno	Mimas	0.94	0.94
Urano	Miranda	1.41	1.41
Urano	Ariel	2.52	2.52
Urano	Umbriel	4.12	4.12
Urano	Oberon	13.46	13.46
Urano	Titânia	8.71	8.71
Neptuno	Proteu	1.12	1.12
Neptuno	Tritão	5.88	5.88

Tabela 2: Períodos orbitais e de rotação de cada satélite em dias terrestres.

Para traduzir estes valores no projeto, visto que o período orbital e rotacional têm valores bastante dispersos, optamos por usar escalas diferentes.

Assim, para o período orbital dos planetas consideramos que 1 ano (365 dias) na Terra era equivalente a 20 segundos. Para o período orbital dos satélites, que são valores bem mais reduzidos, consideramos que 1 dia terrestre era equivalente a 10 segundos.

Como utilizamos escalas diferentes para os períodos orbitais, e uma vez que a rotação dos satélites é sincronizada com a órbita em torno do seu planeta, decidimos utilizar a mesma escala dos períodos orbitais dos satélites para ambos

os períodos de rotação, de forma a poder demonstrar que é sempre o mesmo lado do satélite que fica voltado para o seu planeta.

Posto isto, procedemos à substituição das translações e rotações estáticas do ficheiro pelas dinâmicas utilizando estes valores.

9 Experiência do Utilizador

9.1 Modo Explorador

Como referimos na última fase do projeto, uma das tarefas do grupo consistia em melhorar o modo explorador desenvolvido. Assim sendo, o utilizador pode agora, pelo menu, deslocar-se para o grupo mais próximo da posição atual ou escolher o grupo sobre o qual quer efetuar a exploração. A implementação deste modo da câmara torna a experiência do utilizador muito mais acessível, na medida em que pode acompanhar e visualizar os objetos de uma forma mais detalhada, tendo acesso a uma espécie de câmara em terceira pessoa.

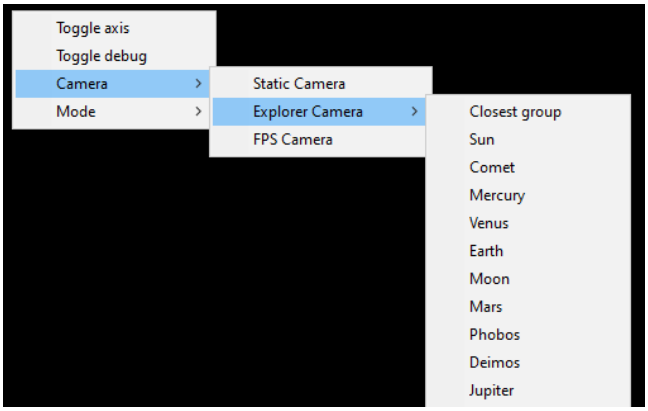


Figura 11: Menu do modo explorador da câmara.

Fazendo uso do atributo "label", verificamos quais são os grupos que têm uma etiqueta atribuída e apresentamo-los no menu, em caso afirmativo, para que o utilizador possa teletransportar-se para esse grupo. Em suma, apenas os grupos que contenham uma *label* podem ser alvo de exploração.

Para efetuar o teletransporte para o grupo mais próximo, assim que essa opção é selecionada, são calculadas as posições de todos os grupos etiquetados e é descoberto o grupo mais próximo, atualizando, em seguida, a posição da câmara para a desse grupo.

Para o cálculo das posições de todos os grupos, recorremos à multiplicação de matrizes, de modo a obter a matriz resultante das transformações. Usamos as seguintes matrizes para a translação de vetor $\vec{t}$, a escala de vetor $\vec{s}$ e a rotação sobre um eixo arbitrário (x, y, z) de ângulo θ, respetivamente:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x^2 + (1 - x^2)\cos(\theta) & xy(1 - \cos(\theta)) - z \times \sin(\theta) & z(1 - \cos(\theta)) + y \times \sin(\theta) & 0 \\ xy(1 - \cos(\theta)) + z \times \sin(\theta) & y^2 + (1 - y^2)\cos(\theta) & yz(1 - \cos(\theta)) + x \times sin(\theta) & 0 \\ xz(1 - \cos(\theta)) - y \times \sin(\theta) & yz(1 - \cos(\theta)) + x \times \sin(\theta) & z^2 + (1 - z^2)\cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Um dos desafios associados a este modo relacionou-se com o facto de o sistema solar ser agora dinâmico, sendo que é necessário uma câmara que acompanhe o objeto em movimento. Inicialmente, como o modelo era estático, as posições de cada grupo eram calculadas no início da execução do programa e armazenadas como variável de instância em cada grupo. Todavia, perante um modelo dinâmico, um grupo poderá estar a movimentar-se e, portanto, a cada *frame*, localizar-se noutra posição, o que faz com que esta variável tenha de ser atualizada com recorrência. Por conseguinte, caso o modo explorador seja selecionado para um dado grupo, a cada *frame* será calculada a posição deste grupo e instantaneamente alterados os parâmetros da câmara, não sendo mais necessária a variável de instância que guardava a posição previamente utilizada na classe de cada grupo.

9.2 Modo *Debug*

Com o objetivo de poder informar o utilizador sobre a posição onde o mesmo se encontra, criamos um novo modo, ao qual denominamos de "debug". Este recurso foi-nos essencialmente útil no processo de desenvolvimento dos teletransportes. Este modo é ativado/desativado através do menu e, quando ativo, apresenta as coordenadas dos pontos atuais dos parâmetros "position" e "lookAt" da câmara, como pode ser observado na figura 12, no canto superior esquerdo do ecrã.

Figura 12: Modo *debug*.

10 Resultados Finais

Nesta secção encontram-se os resultados finais da animação do sistema solar e, ainda, os resultados obtidos em cada teste disponibilizado no enunciado do projeto, incluíndo uma comparação com o resultado esperado no teste das superfícies de Bezier.

10.1 Sistema Solar

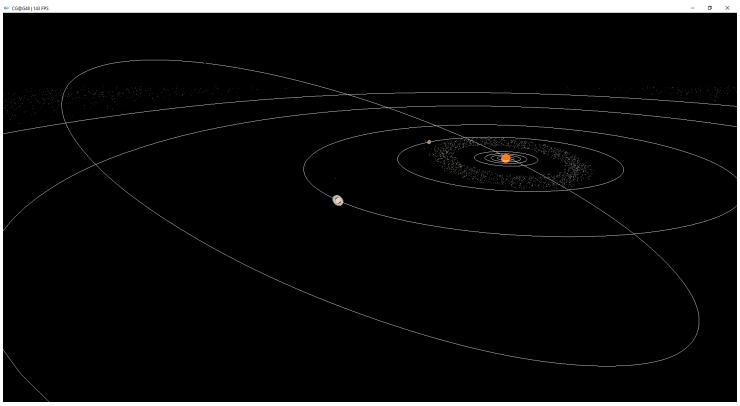


Figura 13: Órbita do Cometa.

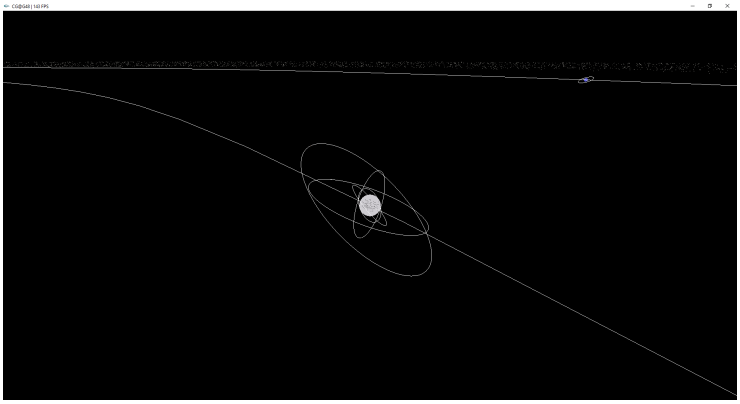


Figura 14: Órbitas dos satélites de Urano.

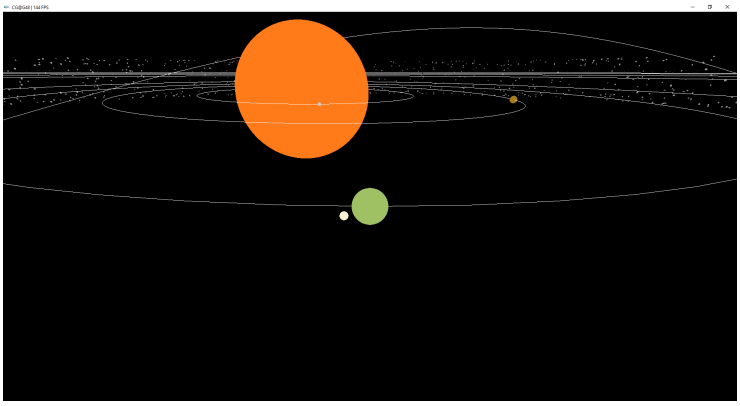


Figura 15: Terra e satélites sem órbitas.

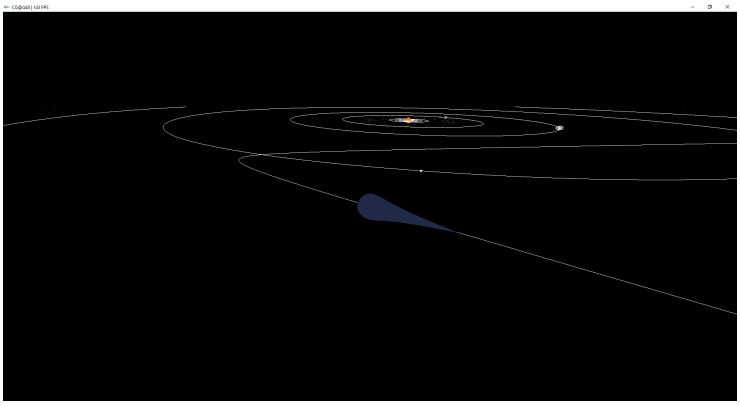


Figura 16: Cometa.

10.2 Testes

Teste 1

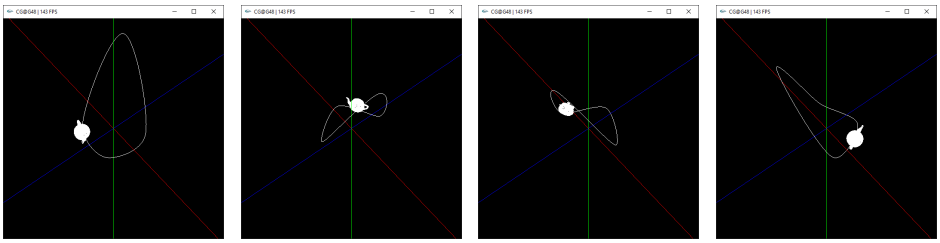
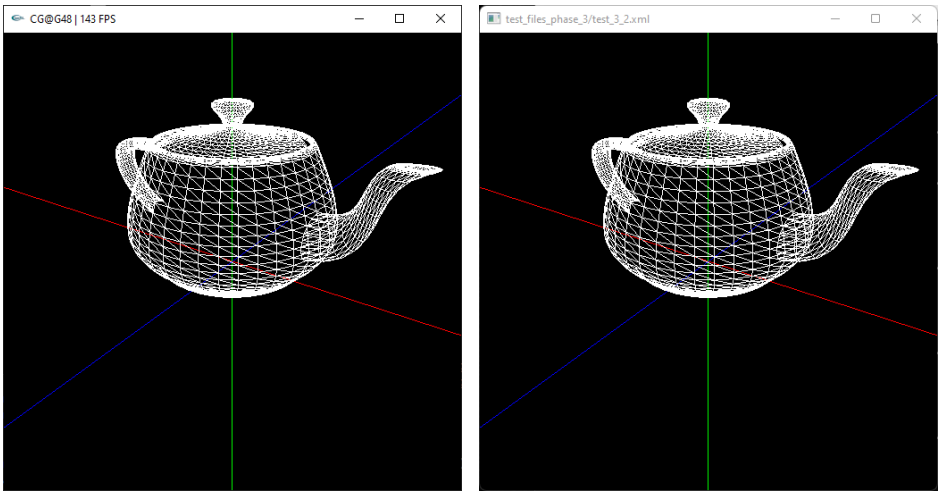


Figura 17: Resultados do Teste 1.

Teste 2



(a) Resultado obtido. (b) Resultado esperado.

Figura 18: Resultados do Teste 2.

11 Conclusão

Para concluir, consideramos que a complexidade desta fase foi maior em relação à anterior, ainda que esta tenha sido reduzida devido aos conhecimentos adquiridos nas aulas práticas.

Uma das dificuldades enfrentadas pelo grupo durante esta fase consistiu no cálculo da posição de um dado grupo, visto que para encontrarmos a sua posição, a única solução passava por recorrer a todas as transformações que o grupo sofre. De forma a calcular o ponto que indica a posição do grupo, teríamos de saber a matriz resultante de todas as transformações. Para solucionar este problema, e, mais uma vez, utilizando os conceitos teóricos aprendidos ao longo do semestre, implementamos a multiplicação das matrizes que traduzem as transformações.

De um modo geral, acreditamos que os objetivos propostos pela equipa docente para esta fase foram cumpridos e estamos satisfeitos com os resultados obtidos. Para além destes, também os objetivos do grupo, que passavam por melhorar a eficiência e o modo explorador da câmara, foram atingidos com êxito.