

# Computação Gráfica - Fase 3

25 de Abril de 2024

## Grupo 2

Daniel Pereira  
A100545

Duarte Ribeiro  
A100764

Francisco Ferreira  
A100660

Rui Lopes  
A100643

## Introdução

Este relatório descreve a terceira fase do projeto de Computação Gráfica. Nesta fase foram implementadas novas funcionalidades, tais como as transformações temporais - no caso específico da translação, com recurso a curvas de *Catmull Rom*. Foi também implementado o suporte à geração de modelos com *VBOs* com índices, melhorando, assim, a performance da renderização de cenas mais complexas. Por fim, foi implementado o suporte à geração de modelos mais complexos, em termos geométricos, com recurso a *Bezier Patches*.

## 1. Simulação de tempo

Com grande foco desta fase, uma componente temporal teve de ser adicionada ao projeto. Esta é responsável por simular o tempo passado, permitindo a implementação das várias funcionalidades temporais que foram pedidas.

Como já tínhamos implementado o processamento do tempo na fase anterior para as acelerações e desacelerações da câmara, foi relativamente simples adicionar a componente temporal ao projeto.

Agora, a `Engine` guarda também o tempo atual, que é atualizado a cada *frame*. A cada *frame* é somado ao tempo atual o tempo que passou desde o último *frame*, permitindo assim a simulação do tempo. Com esta abordagem também é possível controlar a velocidade do tempo através de um fator de escala e também pausá-lo.

```
void Engine::Run()
{
    float currentTime = glfwGetTime();
    while (!glfwWindowShouldClose(m_window))
    {
        ...
        const float newTime = glfwGetTime();
        const float timestep = newTime - currentTime;
        ...
        currentTime = newTime;
        m_simulation_time += timestep * m_time_speed;
        Render();
        ...
    }
}
```

## 1.1. Integração no ImGui

Desta forma, foi adicionada uma aba no *ImGui* [1] que permite fazer a gestão do tempo. A aba de simulação tem um *slider* que permite controlar a velocidade do tempo e um *checkbox* que permite pausar o tempo.

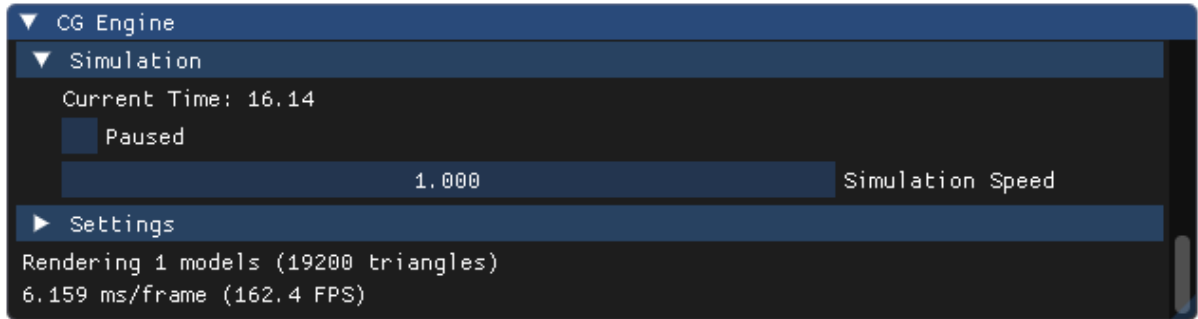


Figura 1: Aba de Simulação no *ImGui*

## 2. Novas transformações

Com a simulação de tempo implementada, podemos passar agora às novas transformações temporais. Para isto, uma pequena reestruturação no código das transformações foi necessária. Agora, a matriz resultado de cada transformação é dependente do tempo e não será uma matriz fixa. Para simplificação de implementação, a matriz de transformação não é mais guardada e alterada só quando necessário como era na fase anterior, mas sim calculada a cada *frame*.<sup>1</sup>

### 2.1. Rotação temporal

Uma das novas transformações adicionada foi a rotação temporal. Esta permite que um grupo seja transformado de forma a que este rode em torno de um eixo (arbitrário) fazendo uma rotação completa em um determinado intervalo de tempo.

A implementação desta transformação é simples, pelo que, foi apenas necessário utilizar a matriz de rotação com um ângulo que é calculado com base no tempo:

$$\mathcal{R}_{\mathcal{T}}(t, \Delta t, x, y, z) = \mathcal{R}\left(\frac{2\pi t}{\Delta t}, x, y, z\right)$$

Sendo que  $\Delta t$  é o tempo que demora a completar uma rotação completa,  $t$  o tempo atual,  $x$ ,  $y$  e  $z$  os eixos de rotação, e  $\mathcal{R}$  a função que calcula a matriz de rotação (enunciada no relatório da fase 2).

#### 2.1.1. Integração no ImGui

Como regra geral deste projeto, esta transformação também foi adicionada no *ImGui*, onde é possível controlar o tempo que demora a completar uma rotação completa e o seu eixo de rotação.

---

<sup>1</sup>Como os cálculos para as transformações não são muito complexos, a performance não é afetada.

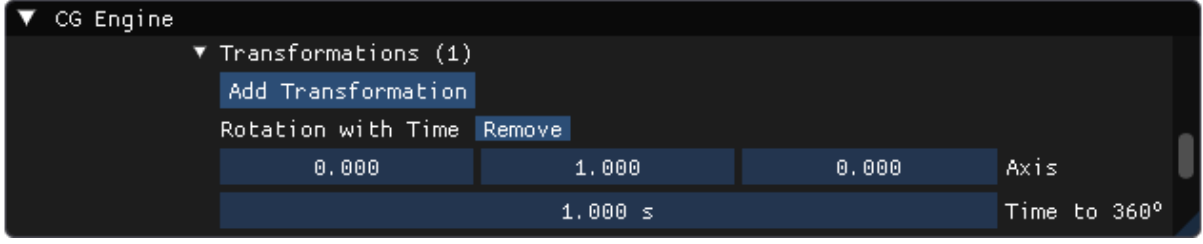


Figura 2: Rotação temporal no *ImGui*

## 2.2. Translação temporal (Catmull Rom)

A outra transformação temporal adicionada foi a translação temporal entre pontos. Esta permite que um grupo se mova entre vários pontos de controlo, de forma que a sua trajetória seja suave e contínua.

Para a implementação desta transformação, foi utilizado o algoritmo de *Catmull Rom* para a interpolação da posição de um ponto entre 4 pontos de controlo. Caso hajam mais de 4 pontos de controlo, a trajetória é calculada a partir dos 4 pontos de controlo mais próximos do ponto do tempo atual.

O ponto final e a sua derivada são calculados a partir das seguintes fórmulas:

$$\mathcal{C}(P_0, P_1, P_2, P_3) = \begin{pmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

$$\mathcal{P}(t, P_0, P_1, P_2, P_3) = \mathcal{C}(P_0, P_1, P_2, P_3)(t^3 \ t^2 \ t \ 1)$$

$$\mathcal{P}'(t, P_0, P_1, P_2, P_3) = \mathcal{C}(P_0, P_1, P_2, P_3)(3t^2 \ 2t \ 1 \ 0)$$

Sendo  $P_0, P_1, P_2, P_3$  os pontos de controlo,  $t \in [0.0, 1.0]$  o tempo atual,  $\mathcal{P}$  a posição atual e  $\mathcal{P}'$  a derivada nessa posição.

Com a posição calculada, o resultado da transformação será apenas uma matriz de translação para esse ponto.

Esta transformação também tem um parâmetro opcional de alinhamento à curva, que usa a derivada calculada para esse efeito.

$$\begin{aligned} \overrightarrow{X}_i &= \|\mathcal{P}'(t, P_0, P_1, P_2, P_3)\| \\ \overrightarrow{Z}_i &= \|\overrightarrow{X}_i \times \overrightarrow{Y}_{i-1}\| \\ \overrightarrow{Y}_i &= \|\overrightarrow{Z}_i \times \overrightarrow{X}_i\| \\ M &= \begin{pmatrix} \overrightarrow{X}_{i_x} & \overrightarrow{Y}_{i_x} & \overrightarrow{Z}_{i_x} & 0 \\ \overrightarrow{X}_{i_y} & \overrightarrow{Y}_{i_y} & \overrightarrow{Z}_{i_y} & 0 \\ \overrightarrow{X}_{i_z} & \overrightarrow{Y}_{i_z} & \overrightarrow{Z}_{i_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Sendo  $i$  a iteração atual e  $i - 1$  a iteração anterior<sup>2</sup>.

<sup>2</sup>Na primeira iteração, quando  $i = 0$ ,  $\overrightarrow{Y}_{i-1} = (0, 1, 0)$ .

Esta matriz  $M$  é multiplicada com a matriz de translação calculada, para a finalidade de rotação de alinhamento do grupo à curva do caminho caso o parâmetro esteja ligado.

A transposta dessa matriz resultado é enviada para o *OpenGL* [2], usando o `glMultiMatrixf()`, completando assim a transformação.

### 2.2.1. Mostrar caminho da curva

Para facilitar a visualização do comportamento da translação temporal, também foi adicionada a renderização do caminho da curva de Catmull Rom. Este caminho é calculado a partir das fórmulas anteriores.

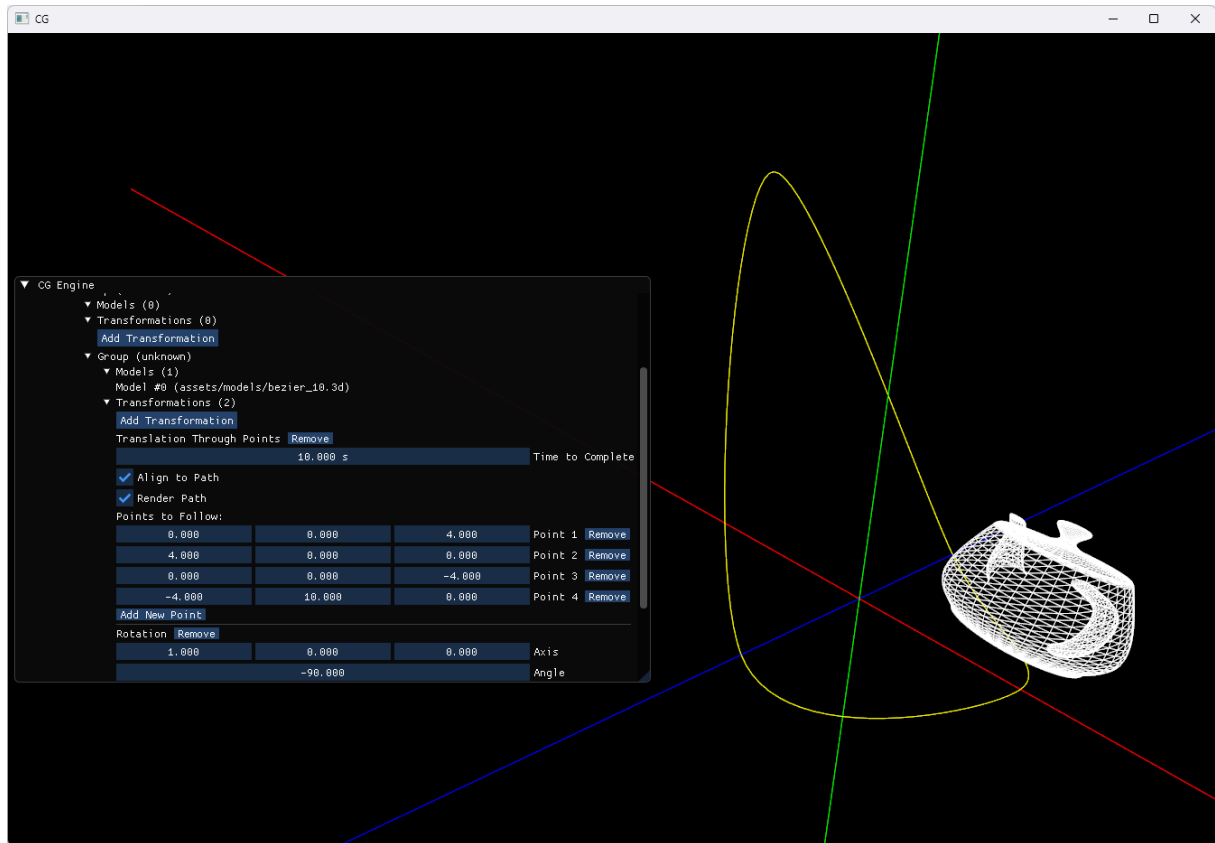


Figura 3: Caminho da curva de Catmull Rom

O caminho é formado por pontos<sup>3</sup> ligados entre si, espaçados uniformemente ao longo da curva. Estes pontos são enviados para o *OpenGL* e são renderizados como `GL_LINE_LOOP`. A forma de como são enviados para o *OpenGL* será explicada num capítulo posterior.

### 2.2.2. Integração no ImGui

Esta transformação também é completamente editável em tempo real no *ImGui*.

<sup>3</sup>Para já são 100 pontos, um valor não configurável.

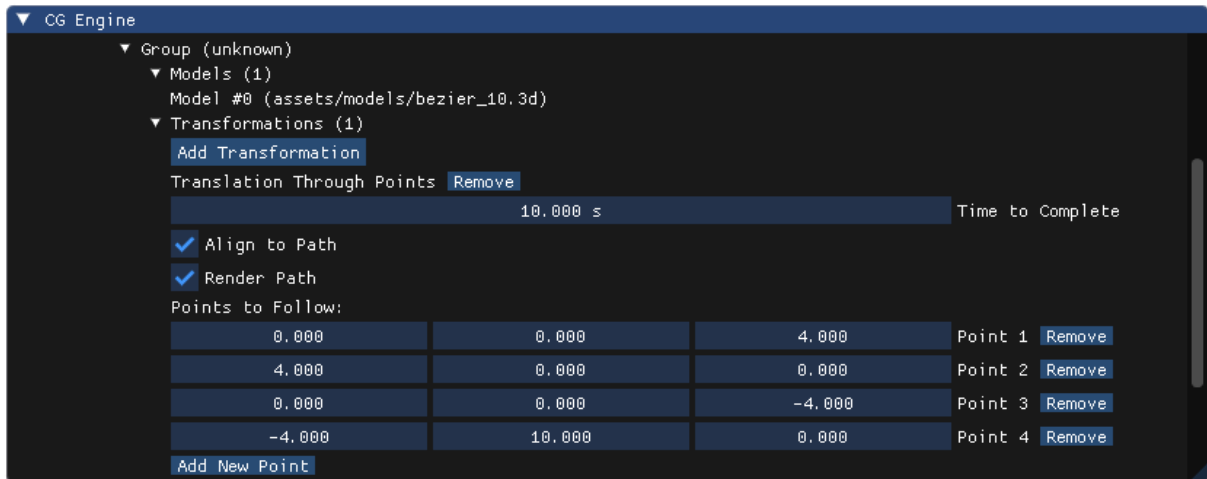


Figura 4: Transformação temporal no *ImGui*

Como se pode ver, é possível adicionar pontos, remover pontos, esconder/mostrar o caminho para cada transformação individualmente (também é possível esconder globalmente na aba da **Settings**), trocar entre alinhar e não alinhar o modelo ao caminho e alterar o tempo do trajeto completo.

Caso a transformação tenha menos de 4 pontos de controlo, uma mensagem de aviso é mostrada ao utilizador que a transformação não tem efeito até adicionar pelo menos os 4 pontos necessários.

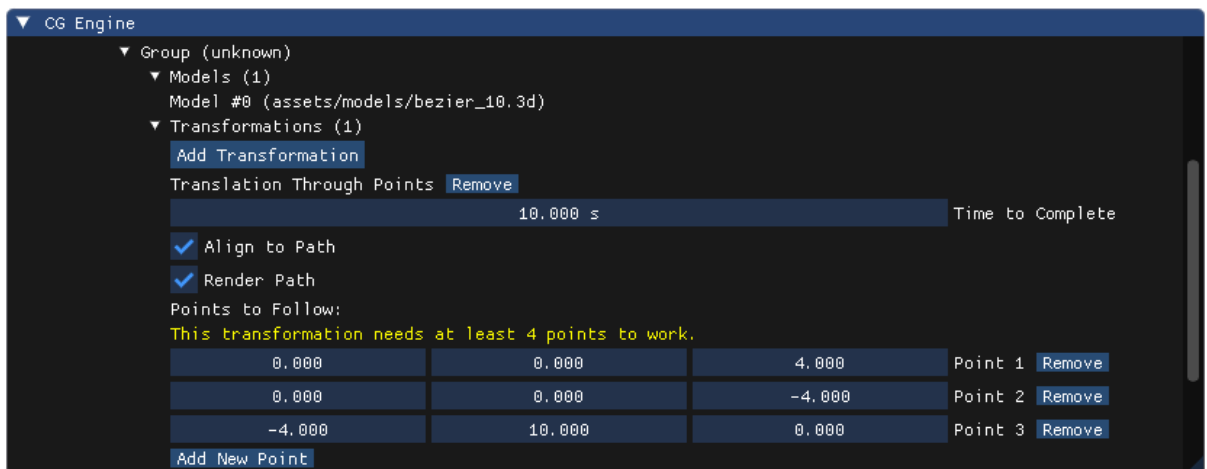


Figura 5: Transformação temporal no *ImGui* com menos de 4 pontos de controlo

### 3. *VBOs* com Índices

Devido ao grande número de vértices que a cena do sistema solar tem vindo a ter, a renderização no modo imediato estava a tornar-se um fator limitante. Com isto, foi implementado o uso de *VBOs* com índices para a renderização dos modelos.

Contudo, para esse efeito, a sua implementação não foi trivial visto que os modelos não tinham informação de índices. A solução que o grupo optou foi alterar o formato de ficheiros *.3d* e a implementação da geração desses para incluir a informação dos índices.

O novo formato de ficheiros `.3d` agora inclui o número de vértices e o número de índices no início do ficheiro, e de seguida a lista de vértices e a lista de índices.

```
4 6

-1 0 -1
1 0 -1
-1 0 1
1 0 1

0 1 2
2 1 3
```

Figura 1: Exemplo de um ficheiro `.3d` com índices

Os índices são (opcionalmente) agrupados por triângulo em cada linha para melhor visualização.

Na próxima fase este formato terá de ser alterado novamente para guardar informações de normais. A geração dos modelos já foi implementada para fácil adição das normais.

A geração de modelos levou grandes mudanças e teve grande consideração na poupança de pontos, evitando duplicações de vértices onde não é necessário.

Na *engine*, o modelo agora é representado por uma lista de vértices e uma lista de índices. Os vértices e índices são carregados para *buffers* da GPU e são renderizados posteriormente com `glDrawElements()`.

```
void Engine::renderModel(uint32_t model_index, size_t index_count)
{
    glBindBuffer(GL_ARRAY_BUFFER, m_models_vertex_buffers[model_index]);
    glVertexPointer(3, GL_FLOAT, 0, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, m_models_index_buffers[model_index]);
    glDrawElements(GL_TRIANGLES, index_count, GL_UNSIGNED_INT, 0);
}
```

### 3.1. Considerações nas gerações

De modo geral, para que o cálculo dos índices seja facilitado, as novas gerações de modelos passam por uma primeira parte de cálculo de vértices, e depois de todos os vértices calculados, são calculados os índices a partir das posições dos vértices.

Isto vai assegurar que na próxima fase do trabalho, bastará adicionar a geração de normais na mesma parte de cálculo de vértices.

#### 3.1.1. Do Plano

O plano, por ser o tipo de geometria mais simples, também tem o processo de nova geração mais simples. O cálculo de índices passa simplesmente por fazer com que pontos fora das bordas apontem para o mesmo vértice, que é partilhado entre vários triângulos.

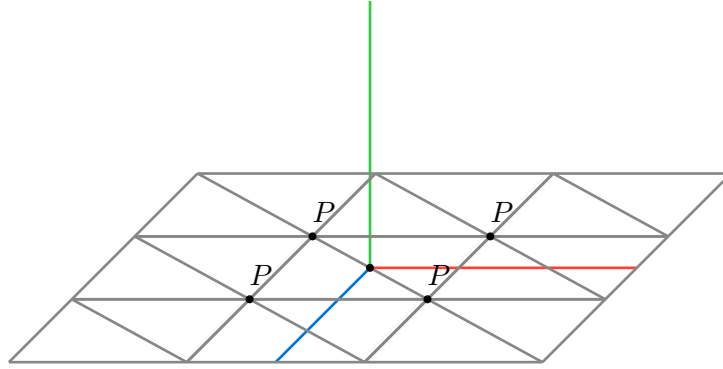


Figura 6: Pontos partilhados na geração de um plano

### 3.1.2. Da Esfera

A geração da esfera, agora com índices, passou a ser uma das mais complicadas. A nova geração de índices tem em conta que vértices do topo e do chão da esfera são partilhados entre todos os *slices* e, ao fim de uma volta completa, os pontos da *slice* atual são agrupados<sup>4</sup> com a *slice* inicial. Com isto, nenhum vértice está duplicado.

Também tem em consideração que na primeira e na última iteração da *stack* só adiciona um triângulo em vez de dois, como foi enunciado no relatório da primeira fase, no capítulo **Problema dos polos da esfera**.

### 3.1.3. Da Caixa

Usando a estratégia de translações de matrizes a partir da geração do plano, a geração da caixa manteve-se praticamente inalterada, já que a lógica de geração dos pontos está toda na geração do plano. Tem consideração de **não** agrupar os pontos adjacentes entre os planos (as arestas do caixa) visto que esses terão normais diferentes.

### 3.1.4. Do Cilindro

No cilindro é onde tem mais duplicações de vértices. Como os pontos que unem as bases às laterais terão duas normais diferentes (uma de uma das bases e uma da lateral), esses pontos tiveram de ser duplicados. Mesmo assim, o vértice do centro do topo e do centro da base são apenas adicionados uma vez. Vértices das laterais também são reutilizados quando formam triângulos adjacentes. Também como na esfera, na geração ao fim da volta completa os pontos do último *slice* são agrupados com o primeiro *slice*.

### 3.1.5. Do Cone

O cone segue uma união da lógica do cilindro e da esfera. Tem em conta o agrupamento de vértices no fim da volta completa, do centro da base e do topo do cone, duplicação de pontos na união da lateral e da base devido às normais, e geração de apenas um triângulo na última *stack* de cada *slice*.

<sup>4</sup>Por agrupado, entende-se que os índices que formam o triângulo correspondente partilham o mesmo vértice.

### 3.2. Suporte a modelos OBJ

O suporte a modelos *Wavefront OBJ* foi continuado. Agora, ao carregar um modelo desse tipo, os índices são usados individualmente e como está no ficheiro (sem cálculo de vértices adicionais). Melhorias de desempenho foram notadas em modelos mais pesados.

### 3.3. Caminho de translação temporal com *VBOs*

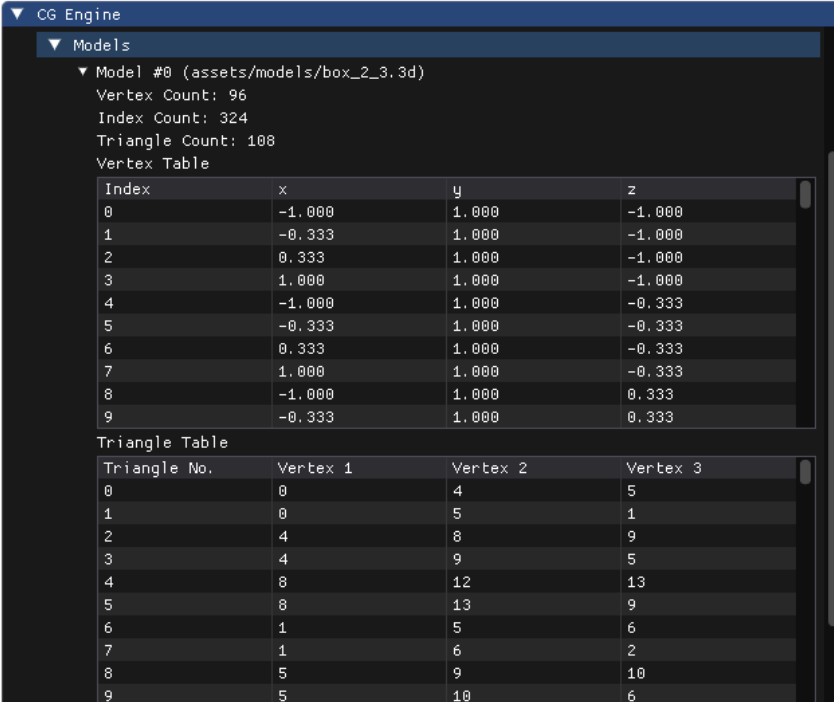
Como enunciado num capítulo anterior, a translação temporal tem a funcionalidade de mostrar o caminho por onde o trajeto da translação acontece. Com grande ênfase na performance desta fase e a possibilidade de haver uma grande quantidade de planetas/satélites com esta transformação no sistema solar, se tais pontos do trajeto forem enviados em modo imediato poderão trazer grande perda de desempenho.

Com isto, cada transformação destas tem o seu *buffer* na GPU para onde são enviados tais pontos quando há alguma alteração nos parâmetros da transformação. A sua renderização, em semelhança aos modelos, é feita com uso de *VBOs* (sem índices visto estar a ser renderizado no modo `GL_LINE_LOOP`).

```
glColor3f(1.0f, 1.0f, 0.0f);
glBindBuffer(GL_ARRAY_BUFFER, translation.render_path_gpu_buffer);
glVertexPointer(3, GL_FLOAT, 0, 0);
glDrawArrays(GL_LINE_LOOP, 0, 100);
glColor3f(1.0f, 1.0f, 1.0f);
```

### 3.4. Nova visualização de modelos no *ImGui*

No *ImGui*, a visualização de modelos também foi alterada para acomodar o novo formato. Agora uma tabela de vértices e uma tabela de triângulos (índices agrupados de 3 em 3) são mostrados, tal como o número total de vértices, índices e triângulos do modelo.



The screenshot shows the 'CG Engine' window with a 'Models' tree. Under 'Model #0 (assets/models/box\_2\_3.3d)', the following statistics are displayed: Vertex Count: 96, Index Count: 324, and Triangle Count: 108. Below these are two tables: 'Vertex Table' and 'Triangle Table'.

| Index | x      | y     | z      |
|-------|--------|-------|--------|
| 0     | -1.000 | 1.000 | -1.000 |
| 1     | -0.333 | 1.000 | -1.000 |
| 2     | 0.333  | 1.000 | -1.000 |
| 3     | 1.000  | 1.000 | -1.000 |
| 4     | -1.000 | 1.000 | -0.333 |
| 5     | -0.333 | 1.000 | -0.333 |
| 6     | 0.333  | 1.000 | -0.333 |
| 7     | 1.000  | 1.000 | -0.333 |
| 8     | -1.000 | 1.000 | 0.333  |
| 9     | -0.333 | 1.000 | 0.333  |

| Triangle No. | Vertex 1 | Vertex 2 | Vertex 3 |
|--------------|----------|----------|----------|
| 0            | 0        | 4        | 5        |
| 1            | 0        | 5        | 1        |
| 2            | 4        | 8        | 9        |
| 3            | 4        | 9        | 5        |
| 4            | 8        | 12       | 13       |
| 5            | 8        | 13       | 9        |
| 6            | 1        | 5        | 6        |
| 7            | 1        | 6        | 2        |
| 8            | 5        | 9        | 10       |
| 9            | 5        | 10       | 6        |

Figura 7: Tabela de vértices e triângulos do modelo no *ImGui*



### 3.5. Diferenças de performance (*Benchmarks*)

Para comparar diferenças da implementação de renderização imediata em comparação com a renderização com *VBOs* com índices, foi escolhido o sistema solar da fase anterior. Para referência, esta cena tem 149940 triângulos de 181 modelos e foi renderizada numa resolução de  $2560 \times 1369$ .

Não foi possível escolher uma cena mais recente, visto que os modelos sofreram alterações nos índices, pelo que, por exemplo, não temos desenvolvido geração de modelos sem índices para *patches* de *Bezier*, que são usados no sistema solar desta fase.

| Sistema solar               | Performance                                | Ambiente de testes |                |
|-----------------------------|--|--------------------|----------------|
|                             |  | GPU                | RTX 2060 SUPER |
| Sem <i>VBOs</i>             | 1.254 ms/frame $\leftrightarrow$ 797.6 FPS | CPU                | Ryzen 7700x    |
| Com <i>VBOs</i> com índices | 0.314ms/frame $\leftrightarrow$ 2930.2 FPS | RAM                | DDR5 6000MHz   |

Tabela 1: Resultados do *Benchmark*

Acreditamos que a diferença seria ainda maior caso fosse usado a cena do sistema solar desta fase, que tem 432 modelos com um total de 218400 triângulos, cuja performance no mesmo ambiente ronda os 2000FPS.

## 4. *Bezier Patches*

Nesta fase foi implementada a capacidade do programa *generator* de gerar modelos a partir de *Bezier Patches*. Estas *Bezier Patches* encontram-se num ficheiro `.patch` e seguem o formato enunciado pela equipa docente.

De forma análoga às fases anteriores, mas neste caso específico, basta correr o *generator* da seguinte forma: `./generator patch <ficheiro.patch> <tessellation level> <output.3d>`. O segundo parâmetro, a tesselação, consiste no número de divisões que cada *Bezier Patch* terá, ou seja, quanto maior o número, mais detalhado será o modelo. No entanto, é importante denotar que um número grande de tesselação<sup>5</sup> irá levar a um número de vértices gerados muito elevado, que faria com que a renderização de tal modelo fosse pesada desnecessariamente com poucos ganhos de fidelidade visual.

### 4.1. Estrutura de um ficheiro *.patch*

```
2
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
3, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27
28
1.4, 0, 2.4
1.4, -0.784, 2.4
0.784, -1.4, 2.4
...
0, -1.4375, 2.53125
1.5, 0, 2.4
1.5, -0.84, 2.4
```

Figura 2: Exemplo de um ficheiro *.patch*

<sup>5</sup>Um número acima de 10, por exemplo.

A primeira linha do ficheiro indica o número de total de *Bezier Patches*. De seguida, cada linha indica os índices dos 16 pontos de controlo que formam a *Bezier Patch*. A linha imediatamente a seguir a todas essas indica o número de pontos de controlo. Por fim, cada uma das linhas seguintes indica as coordenadas dos pontos de controlo.

Assim, o *parser* para ler um ficheiro `.patch` é relativamente simples, visto que o mesmo segue um formato bem definido.

## 4.2. Geração de modelos

A geração de modelos dá-se ao nível do *patch*, onde, para cada um dos *patches* computamos os seus pontos a partir da seguinte fórmula:

$$\mathcal{P}(u, v) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M \begin{pmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{pmatrix} M^T \begin{pmatrix} v^3 \\ v^2 \\ v \\ 1 \end{pmatrix}$$

Sendo  $P_{ij}$  os pontos de controlo do dado *patch*,  $u, v \in [0, 1]$  e

$$M = M^T = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Os pontos são calculados por cada componente  $x, y, z$ .

Esta matriz,  $M$ , é derivada a partir dos coeficientes dos polinómios de *Bersntein*<sup>6</sup>. Neste caso, os polinómios de grau 3, visto que estamos a lidar com curvas com 4 pontos de controlo cada.

De notar que, uma vez que  $u, v \in [0, 1]$ , a geração de pontos é feita para todos os valores das ditas variáveis com passo de  $\frac{1}{\text{tessellation}}$  a cada iteração. De notar também que, o cálculo  $MAM^T$ , onde  $A$  é a matriz dos pontos de controlo, é feito apenas uma vez para cada *patch*, visto que o mesmo é sempre constante.

Finalmente, após os pontos estarem todos computados, o problema reduz-se a agrupá-los de forma a formarem triângulos.

Essencialmente, por cada *patch* um ponto irá formar um retângulo com os seus vizinhos diretos, tal como demonstra a figura seguinte. Se indexarmos os pontos com base nos valores de  $u$  e  $v$  usados para os gerar, um dado ponto  $P_{ij}$  será agrupado com os pontos  $P_{(i+1)j}$ ,  $P_{i(j+1)}$  e  $P_{(i+1)(j+1)}$  para formar dois triângulos.

---

<sup>6</sup>Estes coeficientes podem ser derivados, por exemplo, a partir do Triângulo de Pascal.

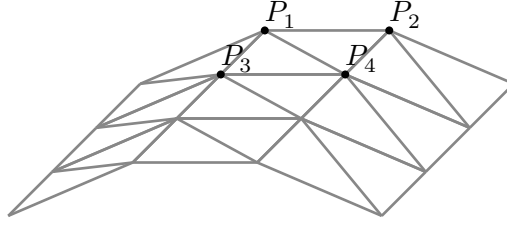


Figura 8: Exemplificação do agrupamento dos pontos em triângulos

Neste caso,  $P_1$  será gerado com  $u = 0.333$  e  $v = 0$ ,  $P_2$  com  $u = 0.666$  e  $v = 0$ ,  $P_3$  com  $u = 0.333$  e  $v = 0.333$  e  $P_4$  com  $u = 0.666$  e  $v = 0.333$ .

O processo de indexação de vértices passa essencialmente no mesmo que a geração do plano.

## 5. Sistema Solar com rotações temporais e asteroides

Agora que temos a capacidade de gerar modelos a partir de *Bezier Patches*, de renderizar grandes quantidades de vértices devido à implementação de *VBOs* com índices e de aplicar transformações temporais, podemos finalmente dar vida ao sistema solar.

### 5.1. Adição do tempo

Pelo *dataset* dos planetas e satélites do sistema solar usado na fase anterior, já temos a informação necessária para tempos de rotações e translações de todos os corpos celestes.

Agora, com a capacidade de simular o tempo, podemos aplicar rotações e translações de forma que os corpos celestes se movam e rodem de acordo com o tempo.

#### 5.1.1. Rotação dos planetas

Para simular a rotação do planeta em torno do seu eixo, basta aplicar uma transformação de rotação temporal com o tempo de rotação do planeta.

Este tempo teve de ser convertido para um tempo que seja visualmente agradável, visto que o tempo de rotação real de um planeta é muito grande.

A função que foi usada para mapear o tempo real para o tempo visual foi a seguinte:

$$\Delta r_{\text{visual}} = 5 \log_2(\Delta r_{\text{real}})$$

Sendo  $\Delta r_{\text{visual}}$  o tempo de rotação visual,  $\Delta r_{\text{real}}$  o tempo de rotação real e  $\log_2$  o logaritmo<sup>7</sup> na base 2.

Esta função tem a propriedade logarítmica que ajuda a que os planetas que rodam exponencialmente mais devagar (por exemplo, Vénus com um dia a durar 5832 horas) tenham a sua rotação minimamente visível, apesar de mais lenta.

Por fim, é adicionada a transformação de rotação temporal no eixo  $(0, 1, 0)$ .

---

<sup>7</sup>Caso  $\Delta r_{\text{real}}$  for negativo, i.e. o planeta roda no sentido contrário,  $\Delta r_{\text{visual}}$  será negativo.

### 5.1.2. Ângulo de inclinação de planetas

De forma a ainda tornar mais real a apresentação, os planetas também são rodados ligeiramente de acordo com o seu ângulo de inclinação que também consta no *dataset*.

Esta rotação é estática e feita no eixo  $(1, 0, 0)$ .

### 5.1.3. Translação dos planetas em torno do sol

A translação dos planetas em torno do sol é o efeito que mais se destaca no sistema solar. Para tal, é aplicada uma translação temporal com o tempo de translação do planeta, usando a mesma fórmula de mapeamento de tempo que a rotação.

$$\Delta t_{\text{visual}} = 5 \log_2(\Delta t_{\text{real}})$$

Como a translação necessita de pontos de controlo, estes são calculados a partir de uma circunferência com o raio da distância do planeta ao sol. A partir daí, são retirados pontos de controlo igualmente espaçados ao longo da circunferência. Para já o número de pontos está fixo a 10, que já dá uma translação que parece ter a trajetória de uma circunferência.

Esta translação podia ser feita através de uma rotação temporal, mas a translação faz com que seja possível que o planeta tenha caminhos de translação mais complexos, como por exemplo, translações elípticas, que pretendemos implementar na próxima fase.

### 5.1.4. Transformações nos satélites

Os satélites do sistema solar também têm translações e rotações em torno dos seus planetas. Estas são feitas de forma análoga às dos planetas, mas com tempos diferentes. Como o *dataset* não tem informação sobre esses tempos, estes foram gerados a partir do tempo de translação do planeta:

$$\Delta t_{\text{satelite}} = \frac{2\Delta t_{\text{visual}}}{5}$$

$$\Delta r_{\text{satelite}} = \frac{\Delta t_{\text{visual}}}{5}$$

Sendo  $\Delta t_{\text{satelite}}$  o tempo de translação à volta do planeta do satélite,  $\Delta t_{\text{visual}}$  o tempo da translação do planeta à volta do Sol e  $\Delta r_{\text{satelite}}$  o tempo de rotação do satélite.

Com estes parâmetros conseguimos fazer com que os satélites tenham movimento visualmente apelativo em relação aos planetas.

## 5.2. Cintura de Asteroides

Baseado na cintura de asteroides do sistema solar, foi adicionado um grupo de asteroides que se movem em torno do sol. Estes asteroides têm translações elípticas, que são feitas a partir de uma translação temporal similarmente à dos planetas.

Cada asteroide tem a sua própria translação, com um tempo aleatório entre 30 e 35 segundos e uma posição aleatória na cintura. Esta posição pode variar em todos os eixos.

Para dar uso à performance adquirida com a implementação de *VBOs* com índices, os asteroides são gerados a partir do patch do *teapot*. Ou seja, um *teapot* é um asteroide. O número de asteroides é configurável na geração do sistema solar, mas como versão final desta fase, o número de asteroides é de 100. Isto é, 100 *teapots* a moverem-se em torno do sol, uma cintura de *teapots* portanto.

Estes *teapots* são gerados com uma tesselação de 1 (mínimo) para que a renderização seja rápida. Visualmente não se nota a diferença visto que os asteroides são pequenos em relação aos tamanhos dos planetas.

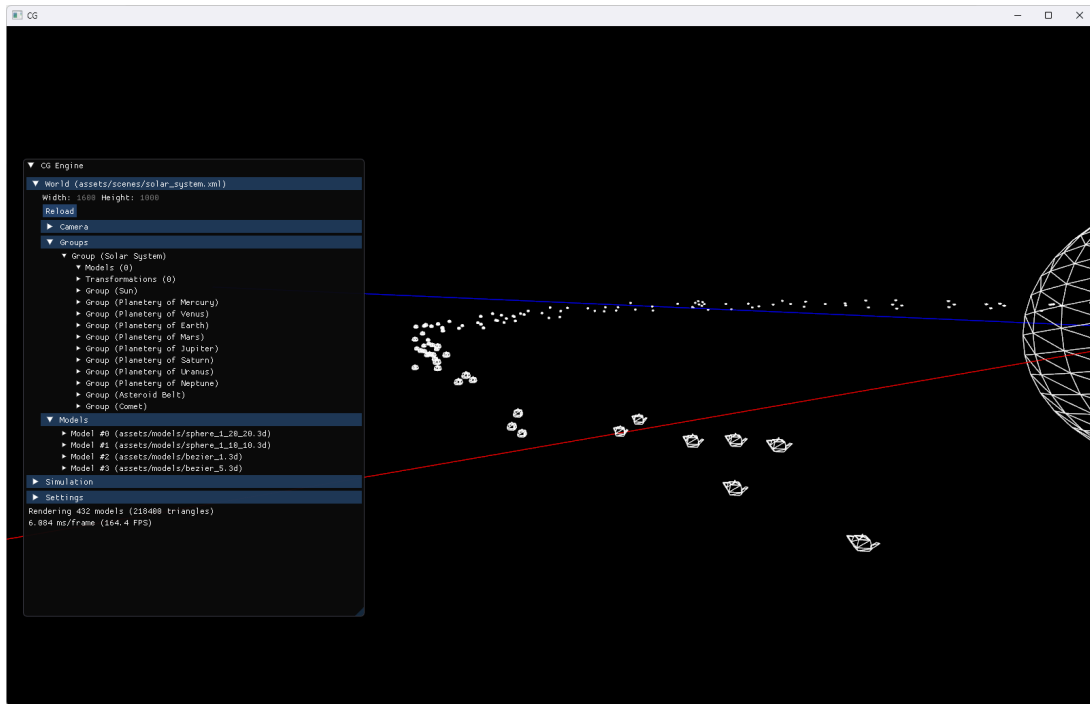


Figura 9: Cintura de asteroides

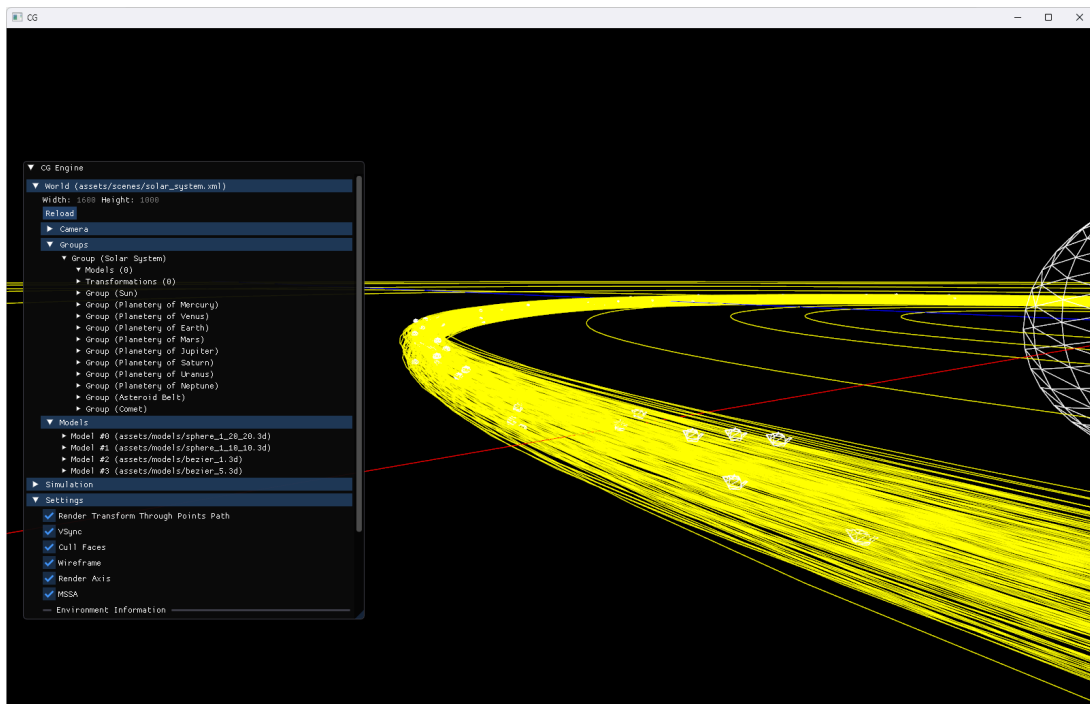


Figura 10: Cintura de asteroides com trajetória de transformações temporais

### 5.3. Cometa

Como requisito deste enunciado, também foi adicionado um cometa que percorre uma trajetória elíptica entre perto de Marte e perto da trajetória de Úrano. Os pontos da translação do cometa

foram calculados de forma semelhante aos planetas, com também 10 pontos de controlo, mas esses pontos foram calculados de forma que o cometa tenha uma trajetória elíptica usando as fórmulas de uma elipse. Numa próxima fase pretendemos expandir as trajetórias elípticas para os planetas também.

Este cometa tem como modelo também um *teapot* mas com mais tesselação, 5, já que o seu tamanho é maior.

O grupo do cometa tem então como transformações:

- Translação no eixo  $x$  para ficar descentralizado com o sistema solar
- Translação temporal com a trajetória elíptica
- Rotação de  $-\frac{\pi}{2}$  no eixo  $(1, 0, 0)$  para o *teapot* ficar perpendicular ao plano da elipse
- Rotação temporal de 20 segundos no eixo  $(1, 0, 0)$  para dar o efeito de rotação ligeira do cometa em torno de si mesmo.

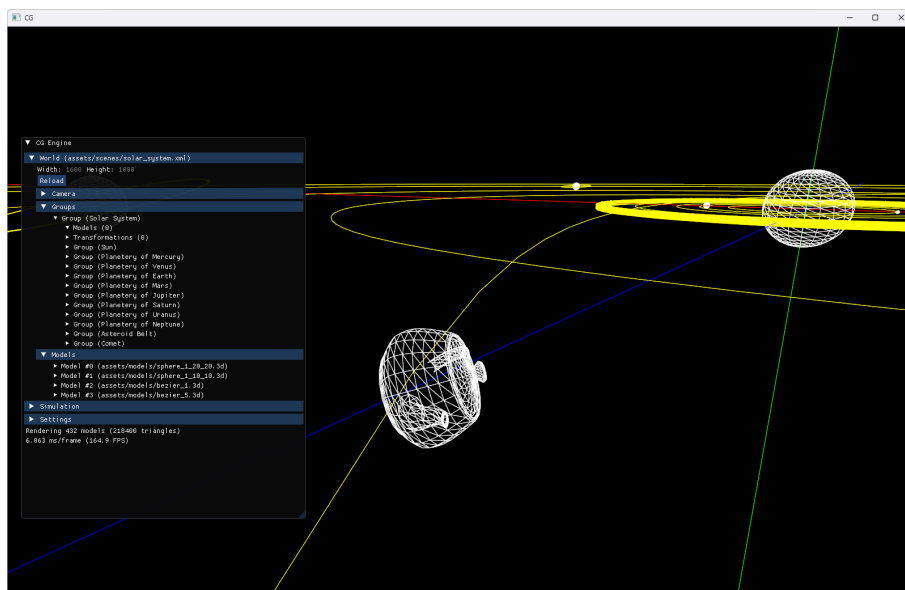


Figura 11: Cometa *teapot*

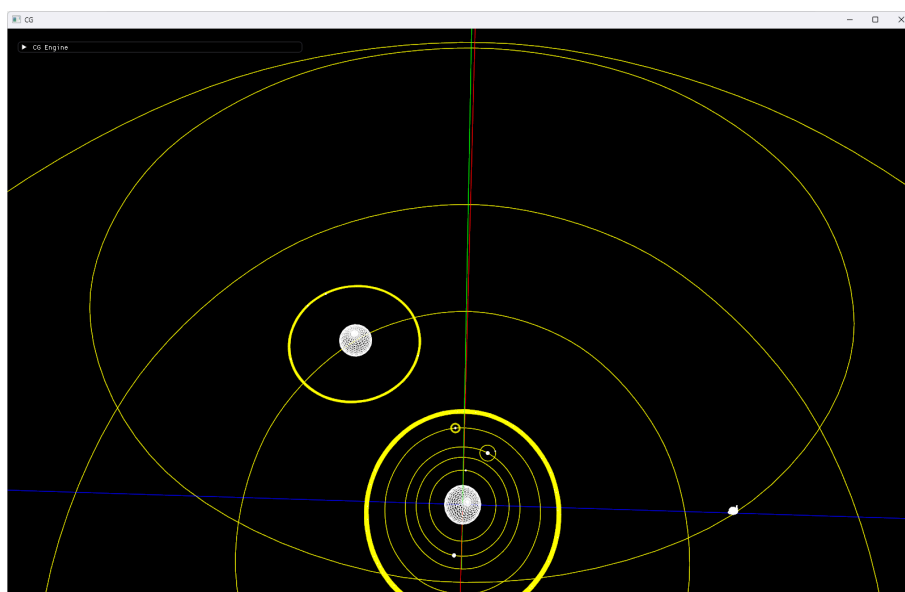


Figura 12: Cometa *teapot* com vista de cima

## 5.4. Nomes dos corpos celestes

Para facilitar a identificação dos corpos celestes, foi adicionado um parâmetro de nome nos grupos. Este parâmetro é opcional.

```
<group name="Planet Earth">  
    ...  
</group>
```

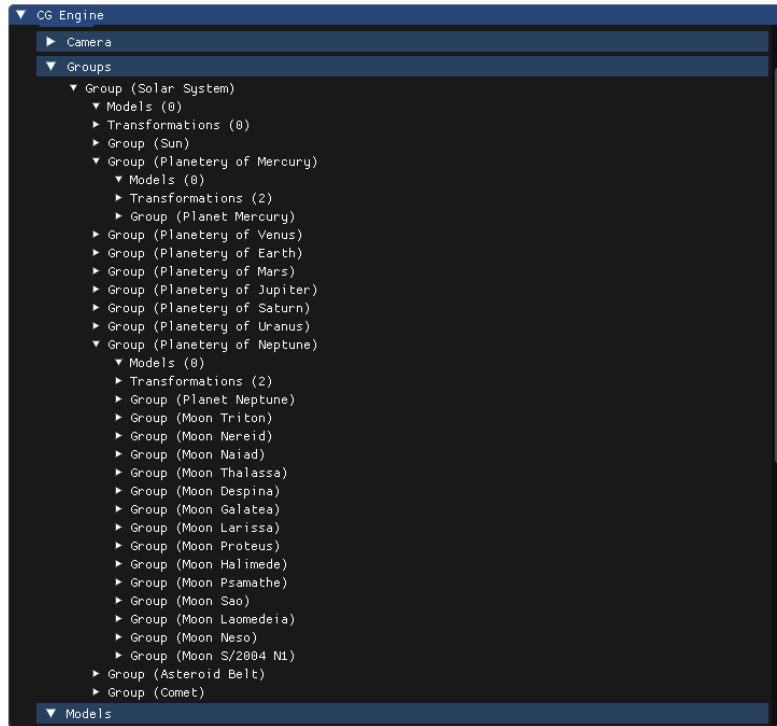


Figura 13: Nomes de grupos no *ImGui*

## 5.5. Estrutura final do sistema solar

Com isto o novo sistema solar pode ser gerado e ele tem a seguinte estrutura:

- Grupo Principal
  - Sol
    - Escala (para o seu tamanho)
    - Rotação (para a sua inclinação)
    - Rotação temporal (para a rotação em torno do seu eixo)
  - Grupo Planetário 1
    - Translação temporal (para o movimento de translação no sistema solar)
    - Rotação (para a sua inclinação)
    - Grupo do Planeta
      - Escala (para o seu tamanho)
      - Rotação temporal (para a rotação em torno do seu eixo)
    - Grupo do Satélite 1
      - Translação temporal (para o movimento de translação em torno do planeta)
      - Escala (para o seu tamanho)
      - Rotação temporal (para a rotação em torno do seu eixo)
  - Grupo de Asteroides

- Asteroide 1
  - Translação temporal (para o movimento em torno da cintura)
  - Rotação de  $90^\circ$  no eixo  $x$  (para alinhar o *teapot* horizontalmente)
  - Escala (para o tamanho de um asteroide)
- Grupo do Cometa
  - Translação (para deslocamento fora do centro)
  - Translação temporal (para seguir o seu trajeto)
  - Rotação (para alinhar o *teapot* horizontalmente)
  - Rotação temporal (para rotação em torno de si mesmo)



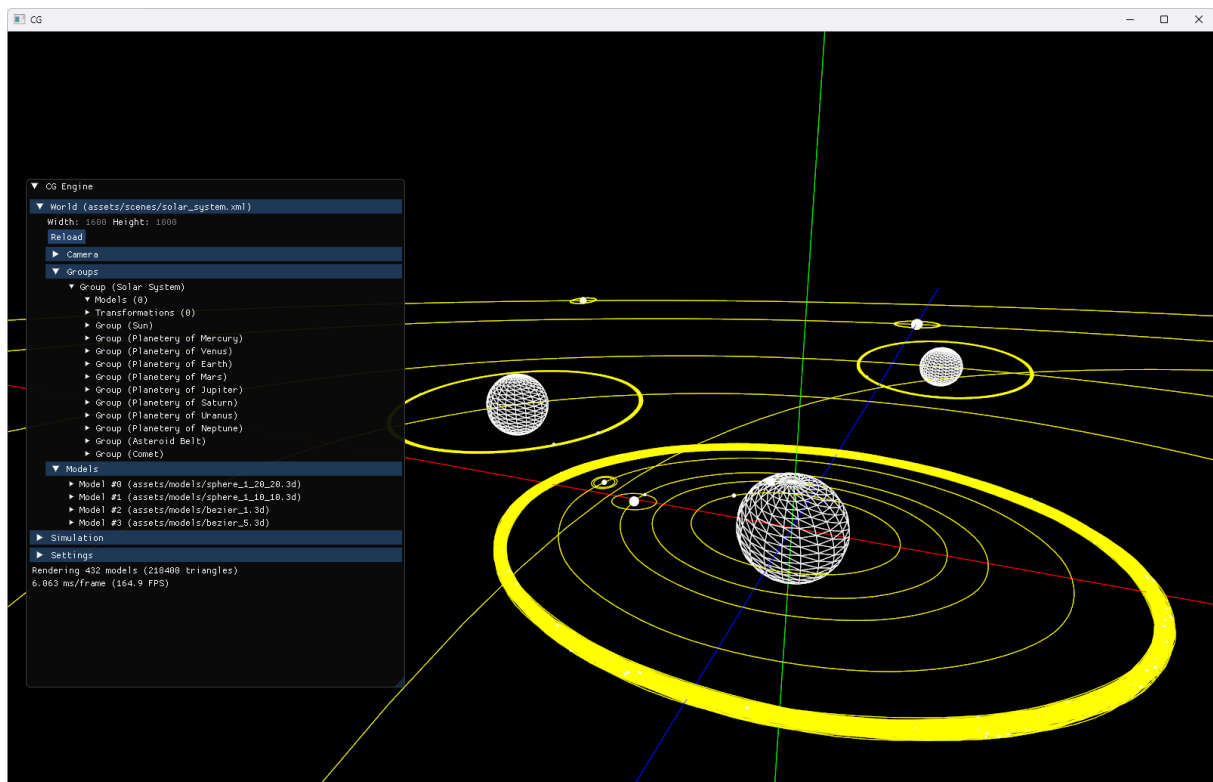


Figura 14: Sistema solar com trajetória de transformações temporais

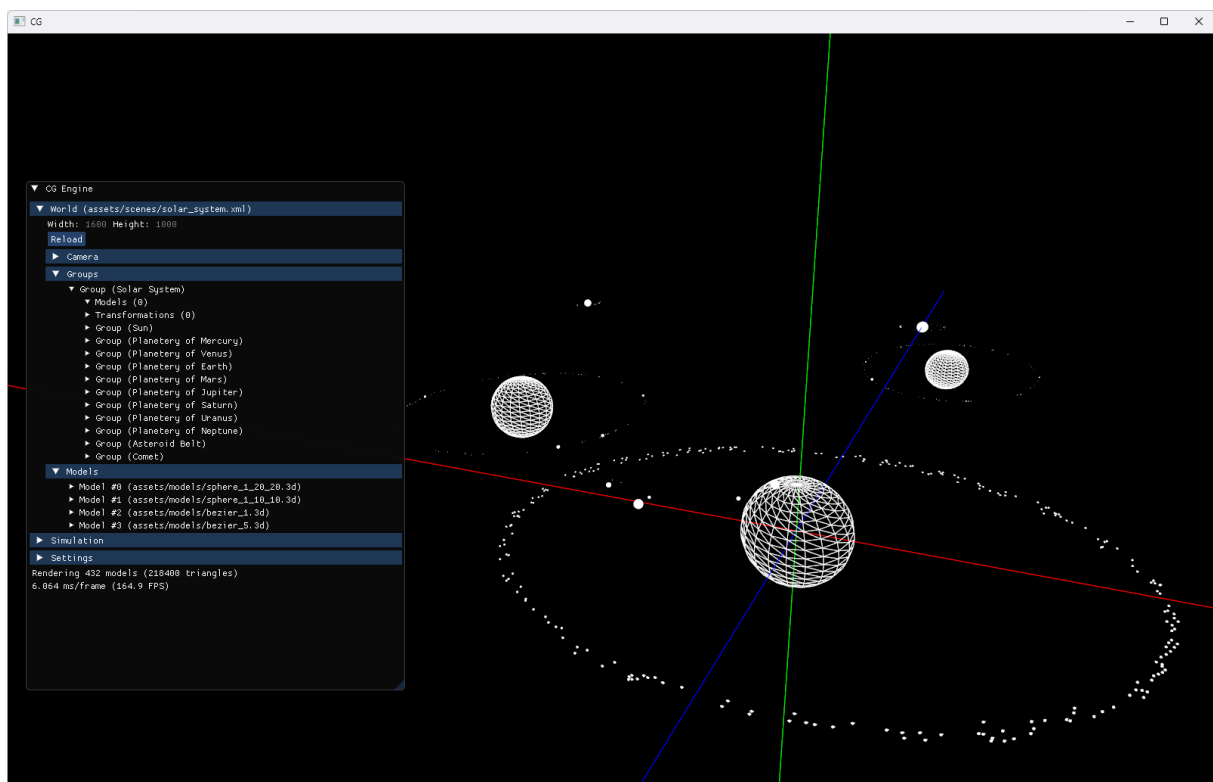


Figura 15: Sistema solar final da terceira fase

## Conclusão

De um modo geral, o projeto está a avançar de acordo com o planeado. A implementação das transformações temporais, a geração de modelos com *VBOs* com índices e a geração de modelos com *Bezier Patches* foram implementadas com sucesso. A implementação dos *VBOs* provou ser uma melhoria significativa na performance do programa, especialmente em modelos mais complexos.

De facto, à medida que as fases vão avançado, o grupo tem vindo a adquirir imenso conhecimento sobre este mundo da computação gráfica e está ansioso para a fase final.

## Bibliografia

- [1] ocornut, «ocornut/imgui». [Em linha]. Disponível em: <https://github.com/ocornut/imgui>
- [2] «OpenGL - The Industry Standard for High Performance Graphics». [Em linha]. Disponível em: <https://www.opengl.org/>