Universidade do Minho
Escola de Engenharia

Computação Gráfica

Fase 2

Trabalho Prático

Grupo 9

João António Redondo Martins - a96215

João Pedro Antunes Gonçalves - a95019

Miguel de Sousa Braga - a97698

14 de abril de 2023

Conteúdo

Lista de Figuras

1 Introdução

O objetivo principal da segunda fase do trabalho prático da cadeira de Computação Gráfica foi construir um modelo estático do Sistema Solar com base numa hierarquia de planetas e luas (modelos) e transformações geométricas descritas em ficheiro XML que se aplicam sobre estes. O conceito de grupo, presente no ficheiro, permite-nos agrupar objetos que são afetados pelas mesmas transformações. Pelo facto de os grupos estarem definidos de forma recursiva, é ainda possível definir subgrupos dentro de um grupo. Nesse caso, cada um dos subgrupos adiciona à sua própria lista de transformações as transformações dos grupos "pai".

Neste relatório, na secção 2, começamos por apresentar o processo de desenvolvimento que nos levou a conseguir os resultados que mais tarde apresentamos na secção 3. Para além das funcionalidades relativas aos objetivos mencionados acima (subsecções 2.1 e 2.2), são ainda expostas algumas funcionalidades extra definidas pelo grupo (subsecção 2.3).

2 Processo de Desenvolvimento e decisões tomadas

2.1 Implementação das transformações

De modo a suportar as várias transformações pedidas (translações, escalas e rotações), começamos por atualizar o nosso parser do ficheiro XML para que siga a seguinte estrutura de um grupo:

```
<group>
    <transform>
        <translate x="1" y="0" z="0">
        <scale x="0.5" y="0.5" z="0.5">
        <rotate angle="30" x="0" y="1" z="0">
    </transform>
    <models>
        <model file="sphere_1_100_100.3d"/>
    </models>
    <!-- Subgroups -->
    <group>
        <transform>
            <translate x="0" y="1" z="0">
        </transform>
        <models>
            <model file="sphere_1_100_100.3d"/>
        </models>
    </group>
    <group>
        <transform>
            <translate x="0" y="0" z="1">
        </transform>
        <models>
            <model file="sphere_1_100_100.3d"/>
        </models>
    </group>
</group>
```

Assim, criamos uma nova classe, chamada *Group*, que contém, para cada grupo, em forma de vetor, as transformações, os modelos e os subgrupos que partilham as suas transformações de forma hierárquica. Esta classe exporta, na sua API, os métodos *readXML*, que faz *parsing* da *tag group* do ficheiro XML, e *drawGroup*, que percorre recursivamente a estrutura de grupos, aplicando as transformações pela ordem que estas aparecem no ficheiro XML, utilizando para isso as primitivas do *glut*.

Por exemplo, para o ficheiro definido acima, a sequência de operações seria a seguinte:

```
glPushMatrix()
glTranslatef(1, 0, 0)
glScale(0.5, 0.5, 0.5)
glRotate(30, 0, 1, 0)
drawSphere() // Importa o modelo .3d da esfera
glPushMatrix()
glTranslatef(0, 1, 0)
drawSphere()
glPopMatrix()
glPushMatrix()
glTranslatef(0, 0, 1)
drawSphere()
glPopMatrix()
glPopMatrix()
```

De modo a aumentar a abstração de dados, criamos, para as transformações, uma classe abstrata com o método *applyTransformation*, que deverá ser implementado por cada classe que implemente uma qualquer transformação. Este método deverá ser o responsável por chamar a função apropriada da interface do *glut*.

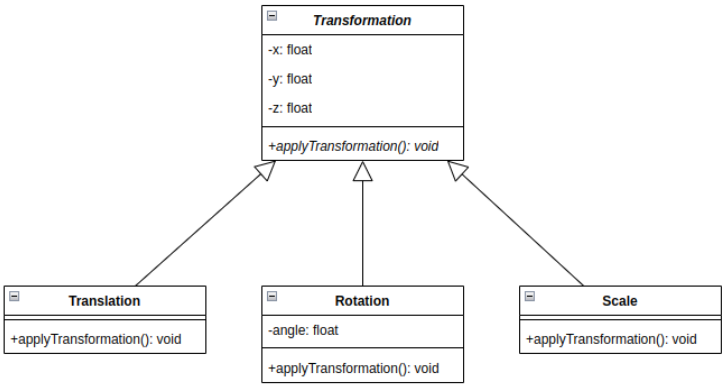


Figura 1: *Transformations*

2.2 Modelação do Sistema Solar

Após algumas pesquisas sobre as distâncias relativas entre os objetos do Sistema Solar, elaboramos a tabela seguinte:

	Diâmetro (km)	**Distância Média do Sol (UA)**	**Escala relativamente à Terra**
Mercúrio	4879	0.39	0.38
Vénus	12104	0.72	0.95
Terra	12742	1	1
Marte	6779	1.52	0.53
Júpiter	139822	5.20	11.2
Saturno	116460	9.58	9.4
Úrano	50724	19.18	4
Neptuno	49244	30.07	3.8

Tabela 1: Planetas presentes na cena

	Diâmetro (km)	**Planeta Originário**	**Escala relativa ao planeta Terra**	**Distância ao Planeta (UA)**
Lua	3474.8	Terra	0.27	0.00257
Fobos	22.2	Marte	0.00054	0.0000626
Deimos	12.4	Marte	0.00015	0.000157
Europa	3122	Júpiter	0.245	0.0045
Ganimedes	5268	Júpiter	0.4135	0.00716
Calisto	4821	Júpiter	0.378	0.0126
Io	3642	Júpiter	0.286	0.00282
Titã	5150	Saturno	0.404	0.00818
Reia	1528	Saturno	0.124	0.00353
Tetis	1062	Saturno	0.124	0.00197
Titânia	1578	Úrano	0.124	0.00292
Oberon	1522	Úrano	0.12	0.00391
Tritão	2706	Néptuno	0.223	0.00237

Tabela 2: Luas presentes na cena

Assim, para manter alguma fidelidade das distâncias relativas reais, mas proporcionar uma réplica que permita a visualização de todo o espaço planetário, recorremos às seguintes escalas:

- Distância ao Sol - Distância (UA) * 1000 = 1 unidade
- Distância do planeta ao satélite natural - Distância (UA) * 10000 = 1 unidade
- Tamanho dos planetas e dos satélites naturais - Escala relativamente á terra * 10 = 1 unidade

Para além dos planetas, foram desenhados também os satélites naturais principais dos mesmos. A posição destes deixa de ser relativa ao centro do Sistema Solar (Sol) e passa a ser relativa ao planeta que orbita, enquanto que, o seu tamanho é relativo à terra.

Foram ainda desenhados alguns objetos auxiliares como a cintura de Kuiper, cintura de asteroides e os anéis de Saturno, bem como as trajetórias (órbitas) de cada um dos planetas e luas. Abaixo, encontra-se um excerto do ficheiro XML que modela o Sistema Solar estático. Nele estão representados a Terra e Lua, juntamente com a sua órbita.

```
<group> <!-- Terra -->
    <transform>
        <translate x="2100" y="0" z="0"/>
    </transform>
    <group>
        <transform>
            <scale x="10" y="10" z="10"/>
        </transform>
        <models>
            <model label="Terra" file="sphere_1_100_100.3d"/>
        </models>
    </group>
        <group> <!-- Lua -->
            <transform>
                <translate x="0" y="0" z="35.7"/>
                <scale x="2.7" y="2.7" z="2.7"/>
            </transform>
            <models>
                <model label="Lua" file="sphere_1_100_100.3d"/>
            </models>
        </group>
        <group> <!-- Lua Órbita-->
            <models>
                <model file="lua_orbita.3d"/>
            </models>
        </group>
    </group>
```

Consultando a tabela 1, podemos ver que a Terra dista 1 Unidade Astronómica do Sol, pelo que irá sofrer uma translação de 1*1000 unidades relativamente ao Sol. Devido ao Sol encontrar-se no centro do Sistema Solar e ter um tamanho de 1100 unidades (é cerca de 110 vezes maior que a Terra), acrescentamos 1100 unidades a essa translação, completando os 2100 do excerto acima. A escala do planeta Terra será de 1*10 unidades uma vez que o seu tamanho relativamente a si próprio é de 1 unidade.

Consultando a tabela 2, reparamos que Lua tem um tamanho igual a 27% do tamanho da Terra, pelo que a escala a utilizar será 0.27*10 = 2.7. Já a sua posição distará 0.00257*10000 + 10 = 35.7 unidades (somamos ao raio da Terra a distância da Lua relativamente à Terra multiplicada por 10000). A órbita da Lua será colocada centrada na Terra, com um raio ajustado ao tamanho e posição da Lua.

2.2.1 Anéis de Saturno

Os anéis de Saturno foram construídos à custa da primitiva gráfica *torus*, apresentada na primeira fase. A colocação deste modelo é feita no mesmo grupo que o modelo que representa o planeta Saturno. Os anéis podem sofrer sucessivas rotações em torno dos eixos principais, deixando o plano onde os anéis estão assentes oblíquo em relação ao plano xOz.

Para serem gerados, o *torus* sofreu uma ligeira alteração. Anteriormente, começávamos por gerar o *torus* pelos pontos com a menor coordenada y, o que nos impedia de obter um *torus* em formato de disco, se possuisse apenas duas *stacks*. Nesta fase, começamos por gerá-lo a partir de um dos pontos da superfície no plano xOz, permitindo assim obter o resultado esperado, demonstrado na figura 9.

2.2.2 Cintura de asteroides e Cintura de Kuiper

A cintura de asteroides e a cintura de kuiper são duas zonas do sistema solar constituídas por formações rochosas de tamanho variável. A primeira, entre Marte e Júpiter estende-se por cerca de 1 U.A (2.2 U.A a 3.2 U.A). A segunda, mais afastada do centro, situa-se após Neptuno e estende-se por cerca de 20 U.A (30 U.A a 50 U.A do Sol)

Para desenhar este conjunto de objetos rochosos, adicionamos uma nova *tag XML*, a *tag ring*, com os seguintes parâmetros:

- **file** - nome do ficheiro com o modelo;
- **outer** - raio mais exterior do anel;
- **inner** - raio mais interior do anel;
- **n** - número de objetos a gerar;
- **minScale** - escala mínima que cada objeto pode tomar;
- **maxScale** - escala máxima que cada objeto pode tomar;
- **minVAngle** - ângulo vertical míximo;
- **maxVAngle** - ângulo vertical máximo.

Importa realçar que a localização de cada objeto gerada por esta instrução é pseudo-aleatória, dependendo de 4 valores gerados aleatoriamente: o ângulo em torno do eixo dos y, o ângulo em torno do eixo dos z (ângulo vertical), a distância ao centro e a escala. Os valores são gerados, semi-aleatoriamente através de uma determinada *seed* e deverão estar dentro dos limites pre-estabelecidos nos parâmetros lidos no *XML*. Por exemplo, o ângulo vertical deverá estar entre o *minVAngle* e o *maxVAngle*. A única exceção é o ângulo em torno do eixo dos y, que varia entre 0 e 360º.

De forma a dar a cada objeto uma caraterística única e realista decidimos, de forma provisória, adicionar 3 atributos quer à *tag ring* quer quer à *tag model*. Os atributos tem o nome 'red', 'green' e 'blue' e definem, cada um, numa escala de 0 a 1, a proporção dessa cor (vermelho, verde ou azul) na cor final (rgb).

2.2.3 Desenho das órbitas

O desenho das órbitas de cada planeta foi feito recorrendo mais uma vez à primitiva *torus*, desenvolvida na fase anterior. Aproximando o raio interior ao raio exterior do torus, conseguimos que o raio da secção de corte do torus fosse bastante reduzido em comparação com as dimensões do planeta. Para além disso, teve que ser aumentado o número de *slices* (cortes verticais) devido ao perímetro elevado do anel.

Uma desvantagem deste método foi a necessidade de criar modelos diferentes para cada uma das órbitas dos diferentes planetas. Aqui não foi possível aplicar escalas, uma vez que assim não seria possível manter constante o raio da secção de corte do torus.

2.3 Extras

Para além dos objetivos definidos para esta fase do projeto, o grupo procurou implementar ainda outras funcionalidades consideradas relevantes, como diferentes modos de câmara, menus e ainda extensões para o ficheiro XML.

Para a câmara, foi criado um módulo separado que encapsula as diferentes operações a que a mesma está sujeita, como mover-se, alterar a direção ou alternar entre os diferentes modos disponíveis.

2.3.1 Câmara em modo FPS

Para além do modo explorador, implementado na primeira fase, é agora possível movimentar a câmara em modo primeira pessoa (FPS). Este modo carateriza-se por uma maior liberdade de movimentos, sendo possível movimentar a câmara para além da esfera que carateriza o modo explorador.

No modo FPS a câmara é caraterizada pelo vetor *up*, pela posição P e por um vetor que define a direção do olhar (na imagem, o vetor d). O vetor d é controlado, tal como no modo explorador, pelos ângulos *alfa* e *beta* que definem o ângulo horizontal (com o eixo do z) e o ângulo vertical (com o plano xOz).

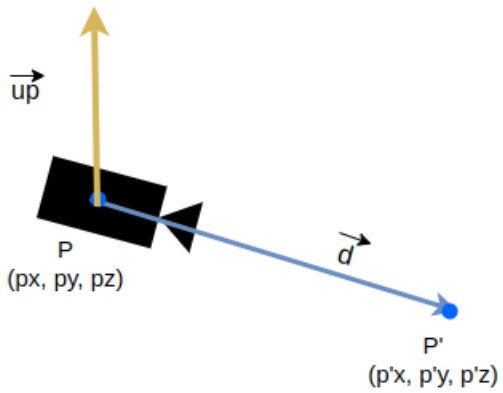


Figura 2: Caraterização da Câmara

As teclas associadas à rotação da câmara são as setas (*arrows*). O movimento vertical (aumentar e diminuir o beta) é controlado com as setas *up* e *down* e o movimento horizontal (aumentar e diminuir o alfa) é controlado com as setas *right* e *left*.

O utilizador pode ainda movimentar a câmara para a esquerda, para a direita, para a frente e para trás, utilizando, para isso, respetivamente, as teclas 'a', 'd', 'w", 's'. O movimento para a frente e para trás resulta de somar o vetor d ou o seu simétrico à posição atual da câmara, resultando numa nova posição P'. Para movimentar a câmara para a esquerda e para a direita, é necessário calcular um outro vetor perpendicular aos dois da imagem.

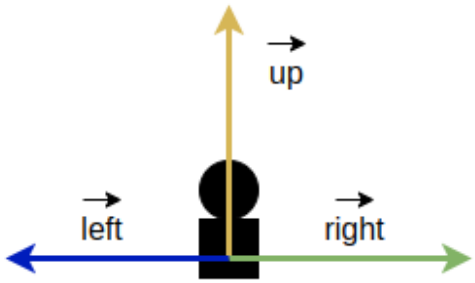


Figura 3: Vetores *lefte right* da câmara

A imagem mostra-nos uma câmara orientada com o vetor d para 'dentro' do ecrã, o que fica evidenciado pela posição dos vetores *left* e *right*. Para o cálculo destes vetores utilizamos o produto vetorial (*cross product*) entre os vetores *up* e *d*. A ordem (*up* x *d* ou *d* x *up*) tem importância uma vez que define qual o vetor resultado. Pela regra da mão direita, ao calcularmos *d* x *up* obtemos o vetor *right*. Já se a ordem for *up* x *d*, obtemos o vetor *left*.

Para além destes movimentos, o utilizador pode ainda mover a câmara para cima e para baixo, utilizando as teclas *HOME* e *END*. O vetor que controla o movimento vertical é o vetor *up*. Todos estes movimentos têm associados um fator de escala (*step*). Aumentando este fator, através das teclas '1' e '2', o utilizador consegue deslocar-se mais rapidamente pelo Sistema Solar.

2.3.2 Câmara em modo explorador controlada pelo movimento do rato

Para complementar os diferentes modos de câmara, foi adicionado ainda um modo de movimentação com recurso ao movimento do rato. É um modo semelhante ao modo explorador, com a diferença de que os ângulos alfa e beta são incrementados/decrementados tendo em conta a deslocação vertical e horizontal do rato pelo ecrã, quando pressionado o seu botão esquerdo. É possível também aumentar e diminuir o raio da esfera de movimento facilmente, utilizando o movimento do rato quando pressionado o seu botão direito.

Para alternar entre os modos da câmara basta premir a tecla 'm'. Neste momento estão presentes dois modos de câmara, o modo explorador (movimentado através do rato ou do teclado) e o modo primeira pessoa. Foi objetivo do grupo que as transições entre os diferentes modos fossem o mais suaves possíveis. Para mudar do modo exploradar para o modo FPS, é necessário fazer o cálculo do vetor *d* e dos ângulos *alfa* e *beta* de orientação do olhar da câmara, ou seja, os ângulos são medidos a partir do referencial da câmara. Já para voltar ao modo

explorador, é necessário cálcular o raio da esfera e os ângulos *alfa* e *beta* relativos ao espaço global, para assim posicionar a câmera corretamente na superfície esférica. Em seguida, apresentamos a forma como os ângulos são calculados:

```
void Camera::calculateAlfa(){
    if (d.getX() == 0 && d.getY()==0 && d.getZ()==0) {
        float vx = this->position.getX() - this->lookAtPosition.getX();
        float vz = this->position.getZ() - this->lookAtPosition.getZ();
        float norma = sqrt( x: pow( x: vx,  y: 2) + pow( x: vz,  y: 2));
        alfa = acos( x: vz / norma);

        if (vx < 0 && vz < 0) {
            alfa += M_PI / 2;
        } else if (vx < 0) {
            alfa += 3 * M_PI / 2;
        }
    } else {
        float vx = this->lookAtPosition.getX() - this->position.getX();
        float vz = this->lookAtPosition.getZ() - this->position.getZ();
        float norma = sqrt( x: pow( x: vx,  y: 2) + pow( x: vz,  y: 2));
        alfa = acos( x: vz / norma);

        if (vx < 0 && vz < 0) {
            alfa += M_PI / 2;
        } else if (vx < 0) {
            alfa += 3 * M_PI / 2;
        }
    }
}
```

Figura 4: Função de cálculo do ângulo *alfa*

No caso do ângulo *alfa*, para além da projeção sobre o plano horizontal, é necessário ter em conta o contradomínio da função *arccos*, que é de 0 a π. Logo, é preciso verificar o quadrante onde se encontra e ajustar o ângulo corretamente.

```
void Camera::calculateBeta(){
    if(d.getX() == 0 && d.getY()==0 && d.getZ()==0) {
        beta = asin( x: (this->position.getY() - this->lookAtPosition.getY()) / this->cameraRadius);
    } else {
        beta = asin( x: (this->lookAtPosition.getY() - this->position.getY()) / this->cameraRadius);
    }
}
```

Figura 5: Função de cálculo do ângulo *beta*

No caso do ângulo *beta*, não é preciso ter em conta o quadrante, visto que o contradomínio da função *arcsen* é de $-\pi/2$ a $\pi/2$, que são os únicos valores permitidos para este ângulo.

Por fim, é importante realçar que o que difere os dois modos é a forma como o ângulo é calculado, ou seja, a partir de qual referencial. Assim, basta alterar a forma como o vetor é calculado, alterando a seu sentido, que os restantes cálculos são feitos de igual forma.

2.3.3 Menus

De forma a dar ao utilizador uma maior liberdade para costumizar o ambiente gráfico foram acrescentados menus, com recurso à função *glutCreateMenu*. Estes menus são ativados quando o botão do meio do rato é pressionado. É possível:

- ativar e desativar a visualização dos eixos coordenados;
- deslocar a câmara para cada um dos componentes do Sistema Solar;
- alterar o *polygon mode*;
- ativar e desativar a informação da câmara, que aparece em rodapé.

Na imagem abaixo podemos ver a aplicação de algumas das funcionalidades acima referidas. Faltou mencionar a informação de debug apresentada na zona inferior do ecrã que apresenta informações relacionadas com o posicionamento da câmara (posição, ângulos, *look at*, etc).

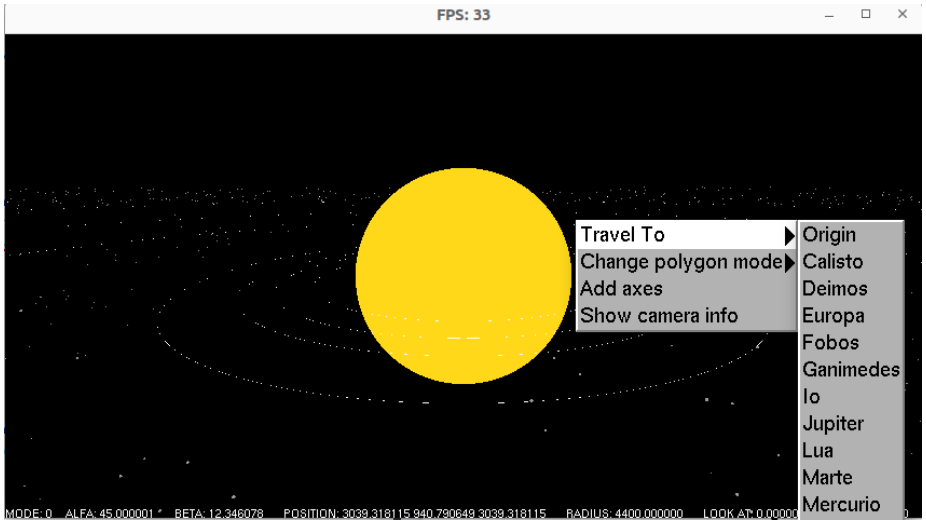


Figura 6: Menus e Informação de Debug

2.3.4 VBOs e FPS

Para aumentar a *performance* da nosso *engine*, principalmente devido à elevada quantidade de primitivas gráficas que o Sistema Solar contém, utilizámos VBOs. Assim sendo, alteramos o programa *generator* de modo a criar os ficheiros das primitivas tendo em consideração a localidade no acesso à memória.

De formar a explicar como funciona o generator após esta atualização, vamos utilizar o cone como exemplo, recorrendo à seguinte figura:

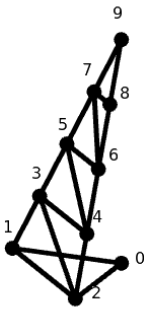


Figura 7: *Slice* do cone

Para garantirmos localidade no acesso à memória, cada *slice* vai ter os seus vértices ordenados de forma a que os índices de cada triângulo sejam seguidos. Quando geramos o cone, para cada *slice*, começamos pelo triângulo da base e depois iteramos pelas stacks, com exceção da última, adicionando os pontos pela ordem indicada e criando os triângulos em função dos índices, sabendo que temos uma variável que nos guarda o índice atual. Por fim, na última stack adicionamos o último ponto e criámos o último triângulo. Nas próximas iterações iremos repetir o mesmo processo, repetindo alguns vértices para ganharmos em desempenho.

De modo a analisar o desempenho da nossa aplicação, calculamos, para cada segundo, o número de frames renderizados. O valor calculado é depois apresentado, em forma de *String* como título da janela.

2.3.5 *Teleports*

Para uma melhor interação e navegação na cena, demos ao utilizador a possibilidade de se teletransportar para os modelos que desejar. Isto compensa o facto de o Sistema Solar ser bastante vasto, permitindo uma mais rápida navegação pelos seus elementos. Para isso, acrescentamos ao ficheiro *XML* um

atributo *label* à *tag model*. Essa label irá identificar, no menu dos teletransportes, cada um dos objetos presentes no nosso espaço. Deste modo, ao selecionar um determinado modelo nesse mesmo menu, a câmara é colocada em modo explorador e o seu *look at* é alterado para o centro do objeto selecionado. O raio de exploração é também alterado com base na escala a que objeto foi sujeito.

O cálculo do centro do objeto segue as normas da aplicação de transformações (translações, escalas e rotações) a pontos com o uso de matrizes. Por exemplo, para saber o resultado de aplicar uma translação segundo o vetor (1,2,3) a um ponto realizamos o seguinte cálculo.

$$\begin{bmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{1}$$

Para aplicar uma rotação (apenas está implementada em relação a um dos eixos coordenados), realizamos um cálculo semelhante. Muda apenas a matrix da transformação (que depende do ângulo alfa da rotação).

Por outro lado, as escalas têm o efeito de não afetar o centro do objeto. Apenas fazem alterar o raio da esfera de observação em torno desse objeto. Assumindo que os fatores x, y e z da escala são iguais, o novo raio é dado pela seguinte fórmula:

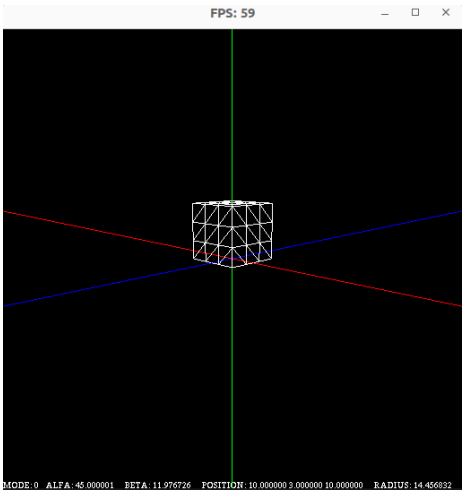
$$r' = s_{factor} * r \tag{2}$$

Importa ainda referir que a ordem de aplicação das matrizes aos pontos deve ser a ordem contrária à utilizada pelo glut.

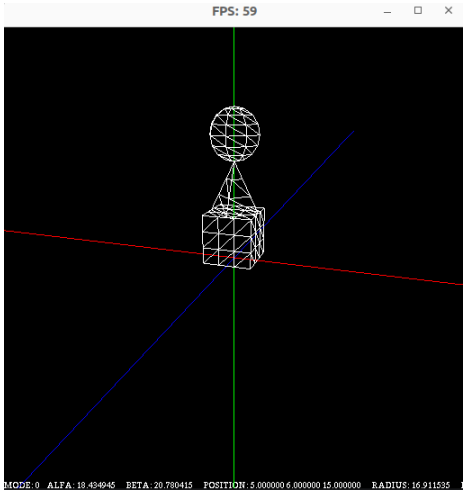
3 Resultados obtidos

3.1 Testes fornecidos

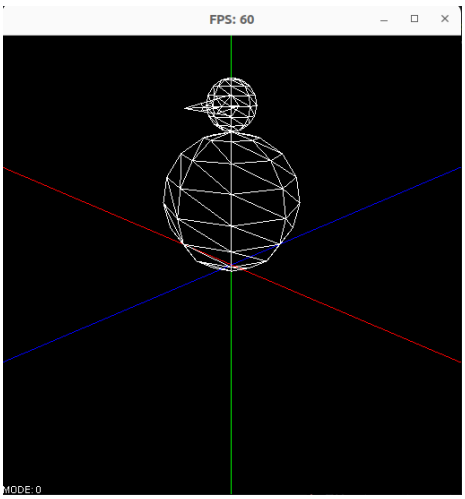
Quanto aos testes fornecidos, foram comparados os resultados obtidos com o programa *engine* com as imagens que mostram o resultado esperado. Todos os testes obtiveram sucesso.



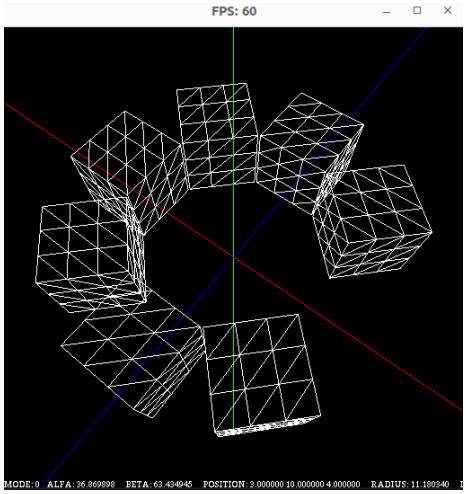
(a) Teste 1



(b) Teste 2



(c) Teste 3



(d) Teste 4

3.2 Modelo do Sistema Solar

Nesta secção expõe-se alguns resultados obtidos com os diferentes modos de câmara no Sistema Solar construído.

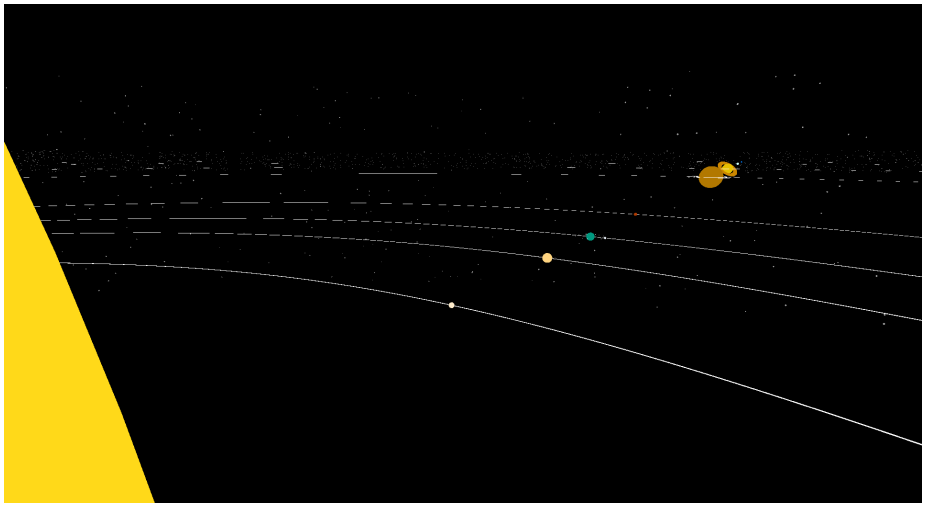


Figura 8: Câmara próxima do Sol

Nesta imagem podemos ver a totalidade do Sistema Solar, desde o Sol até à cintura de *Kuiper*, passando pelos planetas telúricos, pela cintura de Asteroides e pelos planetas gasosos. A imagem foi capturada com a câmara em modo FPS, localizada perto da superfície do Sol.

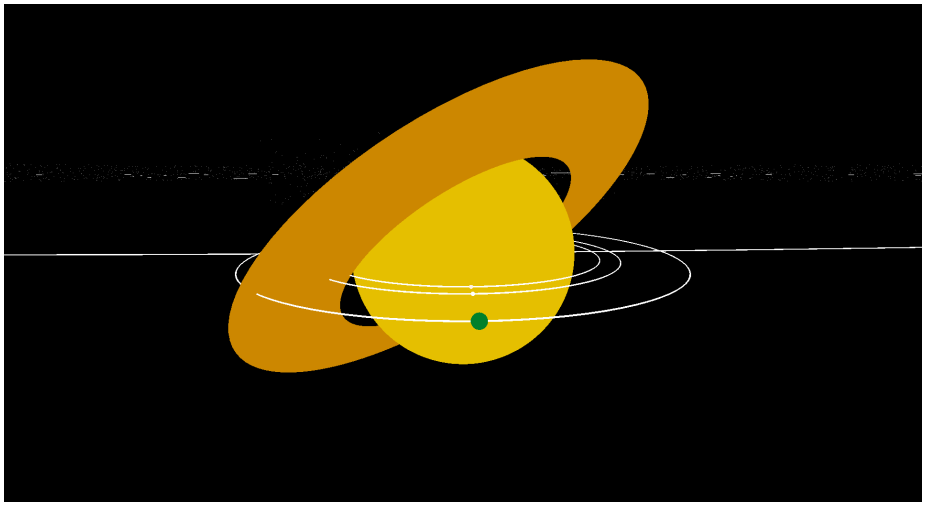


Figura 9: Câmara perto de Saturno

Na segunda imagem capturamos Saturno e as suas 3 principais Luas (Titã, Reia e Tetis), bem como os seus anéis e as órbitas dos satélites referidos.

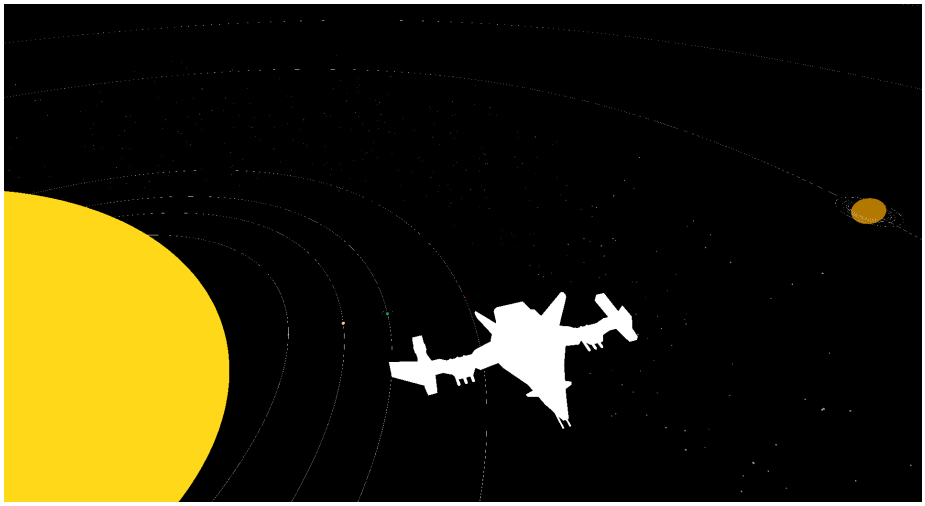


Figura 10: Câmara nave espacial

Nesta imagem, mostramos o resultado da colocação de um objeto importado do blender (formato obj). Neste caso, a nave espacial mostrada foi colocada na posição (2000, 2000, 2000) do nosso espaço.

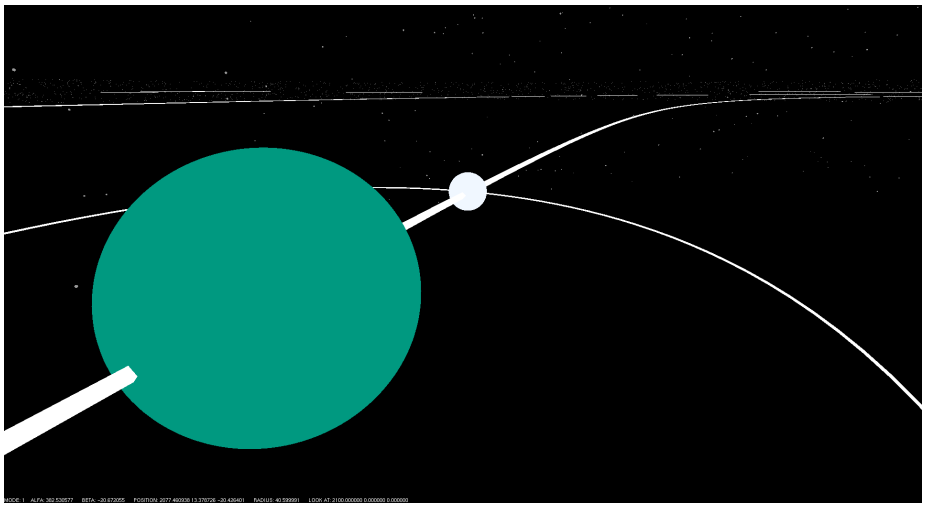


Figura 11: Câmara Terra

Podemos agora ver a Terra e a Lua, bem como a sua órbita. A captura foi realizada com a câmara em modo primeira pessoa.

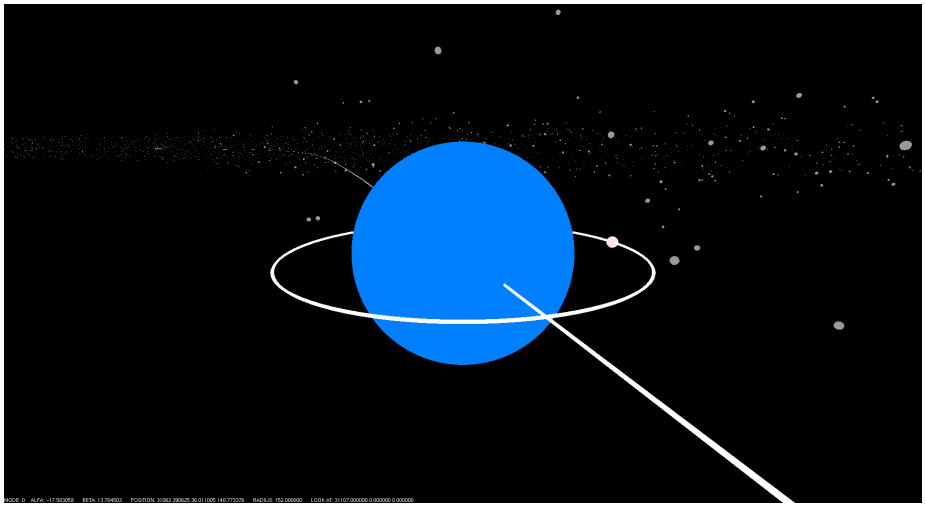


Figura 12: Câmara Neptuno

Na imagem acima vemos Neptuno e a sua maior lua, Tritão. Ao fundo, podemos ver a cintura de Kuiper e os seus numerosos asteroides.

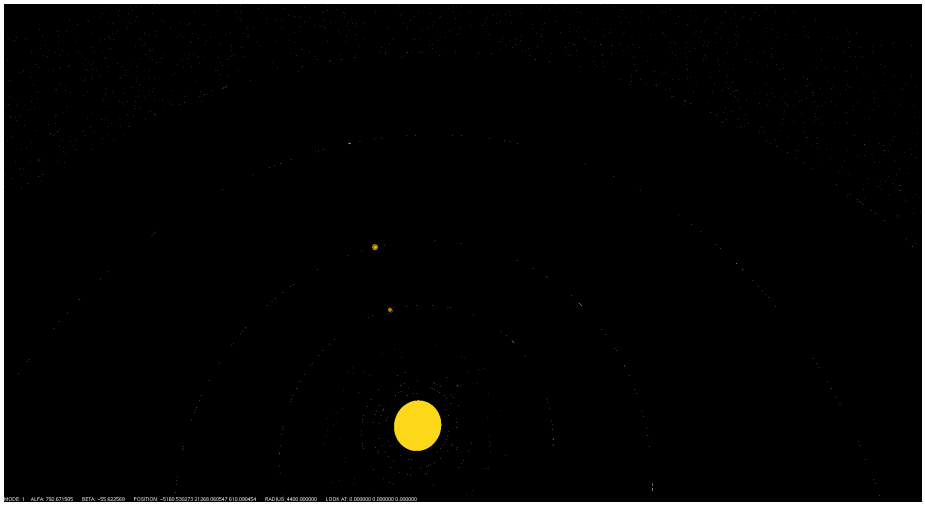


Figura 13: Câmara aérea

Por fim, é apresentada uma vista de cima do nosso Sistema Solar. As órbitas não se apresentam totalmente nítidas aqui devido à grande distância a que

esse objeto se encontra relativamente à câmara (e devido também ao tamanho reduzido do mesmo)

4 Conclusão e balanço da segunda fase

Em conclusão, nesta segunda fase aplicamos, no programa *engine*, transformações geométricas como translações, rotações e escalas com o objetivo de criar um modelo para o Sistema Solar. Para além dos objetivos principais, acrescentamos funcionalidades auxiliares que melhoram a forma como a aplicação pode ser manipulada, dando um maior poder de controlo ao utilizador. Com estes acrescentos, pudemos explorar ainda mais as funcionalidades que o *glut* oferece, aprofundando os conhecimentos adquiridos quer nas aulas teóricas, quer nas aulas práticas, pelo que fazemos um balanço bastante positivo desta etapa do projeto.