Universidade do Minho
Escola de Engenharia

Fase II
Transformações Geométricas

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

abril, 2023

Conteúdo

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 segunda fase do trabalho, onde nos foi requerida a atualização da aplicação "engine", de modo a criar cenários definidos através de uma hierarquia de transformações geométricas. O cenário proposto para esta etapa resume-se num modelo estático do sistema solar.

2 Primitivas Gráficas

2.1 *Torus*

Como descrito no relatório da primeira fase, um dos objetivos do grupo passava pela implementação da primitiva gráfica *torus*. Decidimos implementá-la porque viria a ser útil para a representação dos anéis de certos planetas do sistema solar que vamos construir e para as suas órbitas.

O *torus* é caraterizado pelo seu raio externo, interno, *slices* e *stacks*. Como tal, pode ser computado com uma estratégia semelhante à utilizada para a esfera, um processo iterável pelo seu número de *slices* e *stacks*.

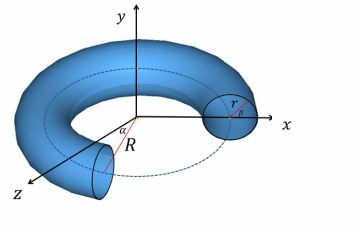


Figura 1: Estrutura do *torus*.

Assumindo que α e β são os ângulos representado na figura, $rOut$ é o raio externo e rIn é o raio interno, podemos inferir a seguinte fórmula, que determina qualquer ponto presente na superfície da primitiva:

$$P\ (rOut + rIn \times cos\beta\ \times cos\alpha,\ rIn \times sin\beta,\ rOut + rIn \times cos\beta\ \times sin\alpha)$$

Com vista a calcular os vértices e os triângulos que formam a primitiva, utilizamos dois ciclos que vão iterar pelas *slices* e *stacks*. O ângulo α e β presentes na expressão são calculados através da multiplicação do número da iteração no ciclo pela ínfima parte do ângulo($\frac{2\times\pi}{slices}$ ou $\frac{2\times\pi}{stacks}$, em que *slices* e *stacks* são o número de *slices* e *stacks*, respetivamente). De notar que, à semelhança da primeira fase, decidimos manter a boa localidade espacial, usufruída pelo *engine*, pelo que tivemos de repetir alguns vértices.

2.2 Elipsoide

O elipsoide foi outra primitiva gráfica que decidimos adicionar ao projeto. Inicialmente, a nossa ideia passava por utilizar esta primitiva com o objetivo de representar asteroides que viríamos a introduzir no sistema solar. Contudo, acabamos por renderizar os asteroides com recurso a uma esfera. A esta esfera aplicamos uma escala não uniforme, o que permite adquirir uma forma semelhante à do elipsoide. Apesar disto, optamos por não desperdiçar o trabalho feito e manter a primitiva no projeto.

Um elipsoide é caraterizado pelos comprimentos dos semieixos e pelos números de *slices* e *stacks*. A estratégia utilizada para o elipsoide é igual à da esfera, ambos diferem apenas nas fórmulas. Um elipsoide pode acabar por ser uma esfera, se os comprimentos dos seus semieixos (a, b e c) forem todos iguais.

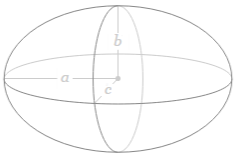


Figura 2: Estrutura do elipsoide.

Podemos definir qualquer ponto presente na superfície do elipsoide com a seguinte expressão geral, em que a, b e c são os semieixos presentes na figura e α e β são os ângulos formados pelos eixos x e z e pelos eixos x e y, respetivamente.

$$P\ (a \times cos\alpha\ \times sin\beta,\ b \times cos\beta,\ c \times sin\alpha\ \times sin\beta)$$

Novamente, recorremos a dois ciclos que vão iterar pelas *slices* e *stacks*, respetivamente. Desta forma, conseguimos obter todos os pontos e índices dos pontos para formar os triângulos que vamos escrever em ficheiro.

3 Sistema Hierárquico

A maior mudança feita, relativamente à primeira fase, foi a implementação de uma hierarquia consoante o ficheiro de configuração recebido. Uma *scene* acaba por ser uma árvore em que cada nodo é um grupo que contém, opcionalmente, transformações e modelos. Esta árvore pode ainda ter filhos que vão ser outros grupos. Estes filhos vão herdar todas as propriedades dos seus pais e, desta forma, todas as transformações aplicadas por um nodo pai vão ser também aplicadas pelo filho, daí estarmos perante uma hierarquia. Para tal, vimos a necessidade de fazer alterações nas classes "World" e "Group", que passaram a ter as seguintes estruturas:

```
class World{
    public:
        Window window;
        Camera camera;
        vector<Group> groups;
```

Figura 3: Variáveis de instância da classe "World".

```
class Group{
    public:
        vector<Group> groups;
        vector<Model> models;
        vector<Transform *> transforms;
```

Figura 4: Variáveis de instância da classe "Group".

A classe "World" passou, desta forma, a possuir um vetor de grupos, visto que uma *scene* pode ser constituída por vários grupos. Nesta fase, a classe "Group" conta agora com uma estrutura para possuir outros grupos (vetor de "Group") e outra para as várias transformações que podem ser aplicadas ao grupo e, por consequente, aos seus filhos.

Outra das mudanças realizadas foi a forma como lemos o ficheiro de configuração, uma vez que é necessário carregar toda a sua informação para memória e colocar nesta nova estrutura. Para tal, recorremos à utilização da recursividade que nos facilitou na implementação da hierarquia.

3.1 Transformações Geométricas

Nesta fase foi-nos proposta a criação de cenários hierárquicos com recurso a transformações geométricas. Desta forma, precisamos de uma maneira de representar as diferentes transformações, pelo que utilizamos uma hierarquia de classes.

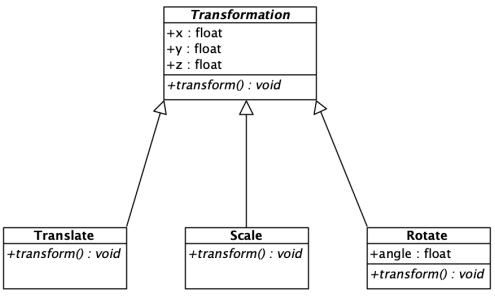


Figura 5: Hierarquia de transformações.

Como inferimos a partir da figura acima, existem 3 tipos de transformações: translação, escala e rotação. Cada uma destas transformações implementa o método abstrato "transform", da superclasse, que vai invocar o respetivo método de transformação da biblioteca "OpenGL".

Um dos métodos do nosso trabalho que achamos importante explicar é o método "drawModels" presente na classe "Group", visto que é ele que lida com toda esta hierarquia.

```
void Group::drawModels(){
    glPushMatrix();

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

    for(int i = 0; i < transforms.size(); i++){
        transforms[i]->transform();
    }

    for(int i = 0; i < models.size(); i++){
        models[i].draw();
    }

    for(int i = 0; i < groups.size(); i++){
        groups[i].drawModels();
    }

    glPopMatrix();
}
```

Figura 6: Método "'drawModels".

Primeiramente, começamos por colocar a matriz na *stack* ("glPushMatrix"), uma vez que vamos aplicar N transformações e não queremos perder a matriz original. De seguida, aplicamos a cor, se existir, e todas as transformações pertencentes ao grupo. Com as transformações aplicadas, podemos agora renderizar os modelos do grupo que irão ser afetados por estas. Finalmente, aplicamos este método recursivamente para todos os filhos, ou seja, para todos os grupos presentes no grupo atual e retiramos da *stack* a matriz inicial ("glPopMatrix"). Desta forma, conseguimos lidar com a estrutura hierárquica presente no ficheiro de configuração XML recebido como *input* e garantimos que todas as tranformações são aplicadas devidamente.

4 Extensões Adicionadas

4.1 *Color*

A primeira extensão adicionada ao nosso *parser* de ficheiros XML foi a extensão *color*, com o formato representado na figura 7.

```
<color R="246" G="241" B="213"/>
```

Figura 7: Utilização da extensão "color".

Esta extensão tanto pode ser utilizada num grupo, o que faz com que altere a cor para todos os seus modelos e consequentes filhos, como para um modelo só, alterando apenas a cor desse modelo. Desta forma, o utilizador pode facilmente alterar os valores RGB e mudar a cor dos seus modelos.

4.2 Modo de Câmara

Uma vez que existem vários modos de visualização, faria sentido o utilizador poder iniciar a representação já com uma câmara diferente da estática, que é iniciada por predefinição. Por conseguinte, não criamos propriamente uma nova extensão mas atualizamos a extensão "camera" de forma a ser, agora, seguida de um atributo "mode", como é representado na figura seguinte.

```
<camera mode="fps">
```

Figura 8: Utilização do atributo "mode".

4.3 *Belt*

De modo a construir a cintura de asteroides no sistema solar, que vamos abordar posteriormente, foi criada a extensão "belt". Esta extensão foi elaborada de modo a poder ser reutilizada para replicar qualquer primitiva em posições aleatórias em forma circular. Os atributos necessários são o ficheiro que contém a primitiva gráfica a ser replicada, o número de réplicas (n), uma *seed*, de modo a gerar semialeatoriamente as posições das primitivas, e um raio interior e exterior e um ângulo vertical para definir os limites horizontais e verticais da zona onde serão gerados os asteroides.

```
<belt file="asteroid.3d" n="2000" seed="51" radiusIn="20" radiusOut="90" verticalAngle="5"/
```

Figura 9: Utilização da extensão "belt".

5 Cenários Desenvolvidos

5.1 Sistema Solar

O cenário proposto a apresentar para esta fase foi o sistema solar. Por conseguinte, procuramos fazer uma representação próxima da realidade, implementando os planetas, os satélites principais, a cintura de asteroides e a cintura de Kuiper.

5.1.1 Escalas Utilizadas

Para obter os valores de distância e tamanho dos planetas e satélites, recorremos ao ChatGPT e à informação disponibilizada pela NASA.

Assumindo que $d_{sol} = 1\ m$ (diâmetro do Sol), recolhemos, então, os seguintes dados:

Planeta	**Diâmetro (m)**	**Distância ao Sol (UA)**
Mercúrio	0.0034	0.4
Veńus	0.0086	0.7
Terra	0.0091	1
Marte	0.0048	1.5
Júpiter	0.1	5.2
Saturno	0.084	9.5
Urano	0.034	19
Neptuno	0.033	30

Tabela 1: Diâmetro dos planetas e distância em relação ao Sol.

A proporção do tamanho dos planetas foi feita em relação ao Sol, ao qual atribuímos um raio de 6 unidades. Contudo, devido ao reduzido tamanho dos planetas obtido, foi necessário multiplicar estes valores por 5 unidades. Quanto à distância existente entre estes, definimos que 1 UA era equivalente a 30 unidades, sendo esta a distância entre a Terra e o Sol.

Planeta	**Satélite**	$\frac{d_{satelite}}{d_{planeta}}$ **(%)**	**Distância ao Planeta (UA)**
Terra	Lua	25	0.00256
Marte	Fobos	0.3	0.000063
Marte	Deimos	0.18	0.000157
Júpiter	Io	2.6	0.00281
Júpiter	Europa	2.2	0.00447
Júpiter	Ganimedes	3.8	0.00714
Júpiter	Calisto	3.4	0.01255
Saturno	Titã	4.4	0.00815
Saturno	Reia	1.3	0.00351
Saturno	Tétis	0.91	0.00196
Saturno	Dione	0.96	0.00252
Saturno	Encélado	0.43	0.00159
Saturno	Iapetus	1.26	0.02374
Saturno	Mimas	0.34	0.00124
Urano	Miranda	0.93	0.000865
Urano	Ariel	2.28	0.001275
Urano	Umbriel	2.31	0.001773
Urano	Oberon	3	0.00389
Urano	Titânia	3.12	0.002909
Neptuno	Proteu	0.85	0.000784
Neptuno	Tritão	5.50	0.002365

Tabela 2: Proporção do tamanho dos satélites e distância em relação ao planeta.

Relativamente aos satélites, a proporção do tamanho de cada um foi feita com base no planeta ao qual pertence. À exceção da Terra, que contém apenas 1 satélite, e de Mercúrio e Vénus, que não possuem satélites, viu-se a necessidade de ajustar a distância de cada satélite ao planeta sobre o qual orbita, uma vez que, em cada planeta, a distância do satélite com distância mínima era bastante reduzida e diminuir apenas essa distância não era o suficiente, visto que pretendíamos manter a escala real. Para representar o efeito de que estavam em órbita sobre o planeta, foram aplicadas rotações antes das translações, onde os ângulos foram definidos de forma a representar uma órbita visualmente agradável em cada satélite.

5.1.2 Cinturas de Asteroides

Quanto aos asteroides, como já foi referido anteriormente, estes são construídos através de uma esfera, à qual aplicamos uma escala não uniforme, uma vez que estes têm um formato irregular. Em seguida, utilizando a extensão "belt", são gerados os vários asteroides em posições aleatórias, formando a cintura. Este sistema conta com a cintura de asteroides, localizada entre as órbitas de Marte e Júpiter, e a cintura de Kuiper, que se extende a partir da órbita de Neptuno.

Um dos problemas que a criação da extensão "belt" levantou foi a forma como iríamos guardar em memória os asteroides para depois renderizá-los. Com vista a resolver isto, decidimos criar um grupo por cada asteroide e este grupo iria conter as transformações necessárias para "colocar" o asteroide na cintura. No caso de serem passados n asteroides na cintura, irão ser criados n grupos com as respetivas transformações.

Relativamente à forma como abordamos as transformações de cada asteroide, geramos um ângulo aleatório entre 0 e 360 graus, segundo o qual realizamos uma rotação segundo o vetor $(0, 1, 0)$.

```
float angle = (float) rand() / RAND_MAX;
angle = angle * 360.0; // 0 e 360
```

Figura 10: Ângulo aleatório.

Como queremos que os asteroides tenham alturas diferentes na cintura, decidimos também gerar um ângulo vertical aleatório com base no limite recebido como parâmetro. Desta forma, realizamos uma rotação segundo o vetor $(0, 0, 1)$ com este ângulo vertical.

```
float vAngle = (float) rand() / RAND_MAX;
vAngle = (vAngle * verticalAngle) - verticalAngle/2.0;
```

Figura 11: Ângulo vertical aleatório.

Precisamos, ainda, de deslocar o asteroide para a cintura. Utilizamos, por isso, uma translação segundo o eixo do x. Resta-nos saber qual é o deslocamento desta translação. Para tal, geramos uma distância tendo em conta o raio externo e raio interno recebidos como parâmetro para a construção da cintura.

```
float deltaX = (float) rand() / RAND_MAX;
deltaX = (deltaX * radiusIn) - radiusIn/2.0;
```

Figura 12: Distância aleatória.

Finalmente, para dar um aspeto irregular aos asteróides, utilizamos uma escala não uniforme com valores aleatórios entre 0 e 1.

```
transforms.push_back(new Rotate(0, 1, 0, angle));
transforms.push_back(new Rotate(0, 0, 1, vAngle));
transforms.push_back(new Translate(radiusOut + deltaX, 0, 0));
transforms.push_back(new Scale((float) rand() / RAND_MAX, (float) rand() / RAND_MAX, (float) rand() / RAND_MAX));
```

Figura 13: Transformações aplicadas na construção de um asteroide.

5.1.3 Anéis e Órbitas

Para implementar o anel de Saturno e as órbitas dos planetas, recorremos ao *torus*.

O anel contém 2 *torus*, um interior e um exterior, contidos no mesmo plano, plano este que sofre uma rotação, de forma a que fique oblíquo em relação ao plano $x0z$, sobre o qual a esfera de Saturno está assente.

Embora tenhamos noção de que as órbitas não são circulares mas sim elípticas, recorremos ao *torus* para representar as órbitas de cada planeta. Assim sendo, foi gerado um modelo diferente para cada uma das órbitas, pois o que distingue cada uma é apenas o valor do raio exterior, o que faz com que não seja possível aplicar escalas sobre um único *torus*.

5.2 Cidade

Para além do cenário exigido para esta fase, o grupo decidiu implementar um cenário extra baseado numa cidade que é, atualmente, composto por prédios, estradas, passeios, postes de luz e árvores. A nossa decisão resultou neste tema dado que é bastante abrangente, sobre o qual podem sempre vir a ser acrescentados novos elementos.

Relativamente à hierarquia do ficheiro XML, existe, novamente, um grupo que engloba todo o cenário. Também as figuras compostas por vários elementos têm um grupo que engloba todos os seus componentes, como é o caso dos edifícios, que contêm as janelas, as portas e a estrutura do mesmo. Outro exemplo são as árvores, um grupo que contém as folhas, representados por cones, e o tronco, representado por um cilindro.

6 Experiência do Utilizador

6.1 Modo FPS

Com vista a permitir ao utilizador movimentar-se e navegar ao longo do cenário, implementamos um novo modo para a câmara - modo FPS. Neste modo, podemos movimentar a câmara através das teclas "W", "A", "S" e "D", sendo que com o "SHIFT"' premido, a movimentação é mais rápida. A orientação da câmara é alterada através do movimento do rato.

Para entender como é feito o movimento e alterada a orientação da câmara, necessitamos primeiramente de perceber as propriedades da câmara, algo que vai ser fundamental para a sua compreensão. Desta forma, a câmara é caraterizada pela sua posição, vetor *up* e vetor direção, que são propriedades extrínsecas.

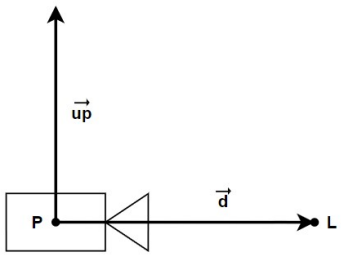


Figura 14: Propriedades extrínsecas da câmara.

6.1.1 Movimento da câmara

O movimento da câmara pode ser dividido em duas partes: o movimento para frente/trás e o movimento lateral.

Relativamente ao *forward/backward motion*, necessitamos da posição atual da câmara (ponto P) e do ponto para onde está a olhar (ponto L). Com estes dois pontos conseguimos calcular o vetor direção ($\vec{d}$), que carateriza a direção que o movimento vai tomar. Com estes três elementos, aplicamos as seguintes fórmulas, de modo a calcular o novo ponto para onde a câmara está a olhar e a sua posição.

$$\begin{cases} \vec{d} = L - P \\ P' = P + k \times \vec{d} \\ L' = L + k \times \vec{d} \end{cases}$$

De notar que o vetor $\vec{d}$ deve ser normalizado, visto que em cada "salto" queremos que a câmara se movimente com um comprimento constante. O elemento k é o fator que nos permite aumentar/diminuir a velocidade com que a câmara se movimenta.

Quanto à movimentação lateral, vamos precisar de um vetor adicional pelo qual se vai realizar o movimento. Este vetor terá de ser perpendicular ao plano formado pelos vetores $\vec{up}$ e $\vec{d}$ (direção), pelo que realizamos o seguinte produto vetorial:

$$\vec{r} = \vec{up} \times \vec{d}$$

À semelhança do movimento frente/trás, calculamos a nova posição (ponto P') e o novo ponto para onde está a olhar (ponto L') com o seguinte sistema de equações:

$$\begin{cases} \vec{d} = L - P \\ P' = P + k \times \vec{r} \\ L' = L + k \times \vec{r} \end{cases}$$

Novamente, o vetor $\vec{r}$ terá de ser normalizado e k é o fator que permite regular a velocidade.

6.1.2 Orientação da câmara

Um dos objetivos que tínhamos traçado passava por alterar a orientação da câmara através do movimento do rato.

Para tal, necessitamos de calcular o deslocamento nas componentes de x e y, que vão ditar os incrementos nos ângulos α e β, respetivamente, utilizados para calcular o novo ponto para onde a câmara vai olhar. Desta maneira, este ponto é calculado da seguinte forma:

$$\begin{cases} L_x = P_x + \sin\alpha \times \cos\beta \\ L_y = P_y + \sin\beta \\ L_z = P_z - \cos\alpha \times \cos\beta \end{cases}$$

L é o novo ponto para onde a câmara vai olhar e P é a sua posição atual. De notar que o ponto L é calculado através das coordenadas esféricas com recurso aos ângulos α e β. Com isto, conseguimos navegar livremente pelo nosso cenário com um movimento semelhante aos jogos *First Person Shooter*.

6.2 Modo Explorador

Nesta fase, também o modo explorador sofreu algumas alterações. Anteriormente, o modo consistia numa câmara que circulava em volta do ponto de origem $(0, 0, 0)$. Tendo em conta que os cenários agora não resultam apenas

numa única figura gerada na interseção dos eixos, é necessário poder explorar primitivas às quais tenham sido aplicadas translações. Assim sendo, e uma vez que podemos movimentar a câmara através do modo FPS, o modo explorador passa a circular em torno do ponto atual da câmara quando este é ativado; isto é, por exemplo, se quisermos circular em volta de um planeta devemo-nos posicionar no centro da esfera e ativar o modo explorador.

Ademais, juntamente com as "arrows" para movimentar a câmara e as teclas "+" e "-" para aproximar ou afastar a câmara do objeto, passa a ser também exequível a utilização do rato e do seu *wheel* para efetuar estas ações.

6.3 Menu

Previamente, o modo da câmara era alternado entre estático e explorador com recurso à tecla "c". Uma vez que a aplicação passa a suportar 3 modos de câmara distintos, recorremos, agora, a um menu, representado na figura 15, que pode ser obtido através de um clique no botão direito do rato.

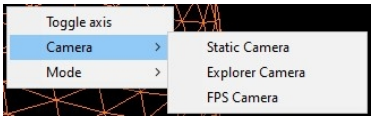


Figura 15: Modos da câmara.

Para além de permitir alterar o modo corrente da câmara, este menu permite, também, ativar ou desativar os eixos e alterar o modo de rasterização dos polígonos pertencentes às primitivas gráficas entre linhas, preenchidos ou pontos.

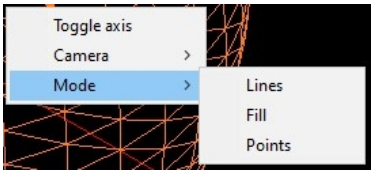


Figura 16: Modos de rasterização.

6.4 Manual de Utilização

Ao executar os programas "engine" e "generator" sem argumentos ou com o argumento "–help", o utilizador tem acesso a um manual, com vista a facilitar a utilização de cada aplicação.

```
-------------------------------------HELP-------------------------------------
USAGE: ./generator {GRAPHICAL PRIMITIVE} {ARGUMENTS} {OUTPUT FILE}
------------------------------------------------------------------------------
GRAPHICAL PRIMITIVE | ARGUMENTS                                  | OUTPUT FILE
plane               | {length} {divisions}                       | {filename}.3d
box                 | {length} {divisions}                       | {filename}.3d
cone                | {radius} {height} {slices} {stacks}        | {filename}.3d
sphere              | {radius} {slices} {stacks}                 | {filename}.3d
cylinder            | {radius} {height} {slices}                 | {filename}.3d
torus               | {radius_in} {radius_out} {slices} {stacks} | {filename}.3d
ellipsoid           | {a} {b} {c} {slices} {stacks}              | {filename}.3d
------------------------------------------------------------------------------
```

Figura 17: Guia de utilização do programa "generator".

```
-------------------------------------------------HELP-------------------------------------------------
USAGE: ./engine {FILEPATH}.xml
note: the filepath must be relative to 'build' folder
------------------------------------------------------------------------------------------------------
OPEN MENU: mouse right button
------------------------------------------------------------------------------------------------------
FPS CAMERA MODE      | MOVE: W, A, S, D      | POINT: MOUSE            | INCREASE SPEED: hold SHIFT
EXPLORER CAMERA MODE | MOVE: ARROWS or MOUSE | ZOOM: +/- or MOUSE WHEEL
------------------------------------------------------------------------------------------------------
```

Figura 18: Guia de utilização do programa "engine".

7 Resultados Finais

Nesta secção encontram-se os resultados finais obtidos nas *demo scenes* criadas pelo grupo e, ainda, a comparação entre os resultados obtidos e os resultados esperados em cada teste disponibilizado no enunciado do projeto.

7.1 Sistema Solar

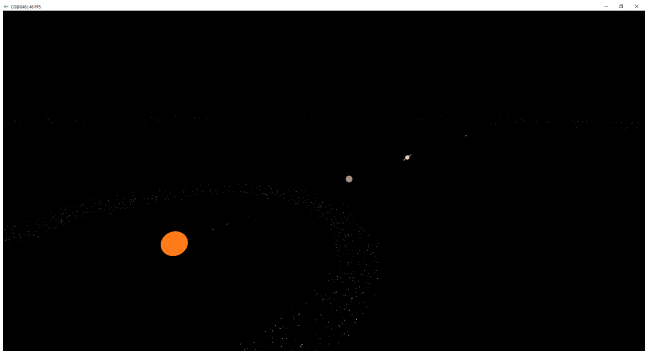


Figura 19: Sistema solar sem órbitas.

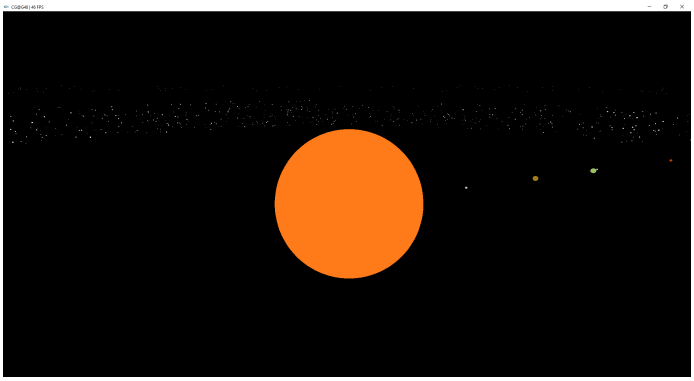


Figura 20: Sol.

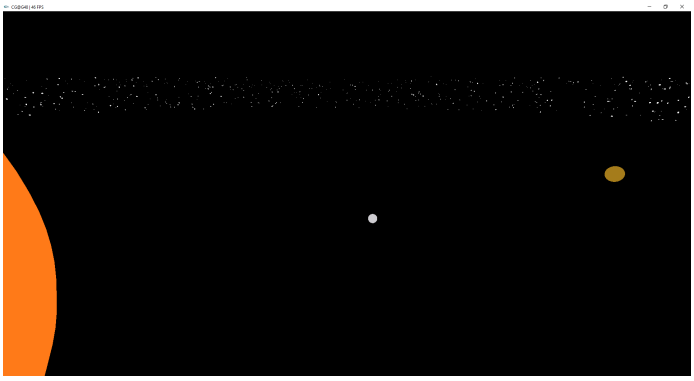


Figura 21: Mercúrio.

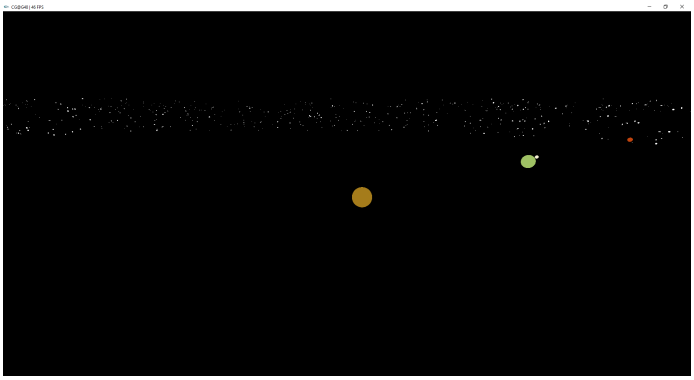


Figura 22: Vénus.

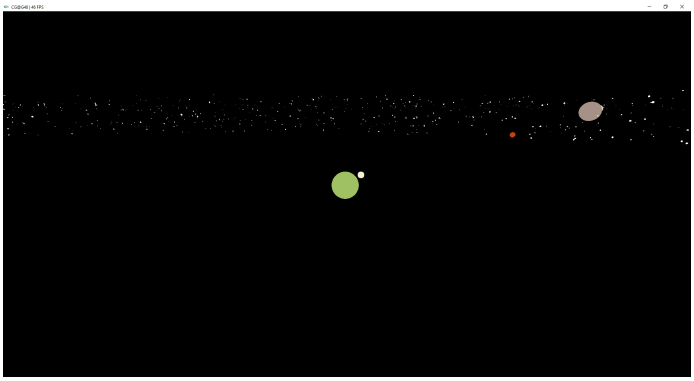


Figura 23: Terra e Lua.

Figura 24: Marte e respetivos satélites.

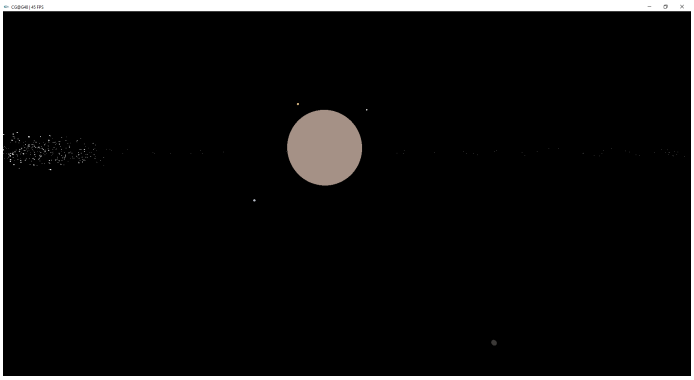


Figura 25: Júpiter e respetivos satélites.

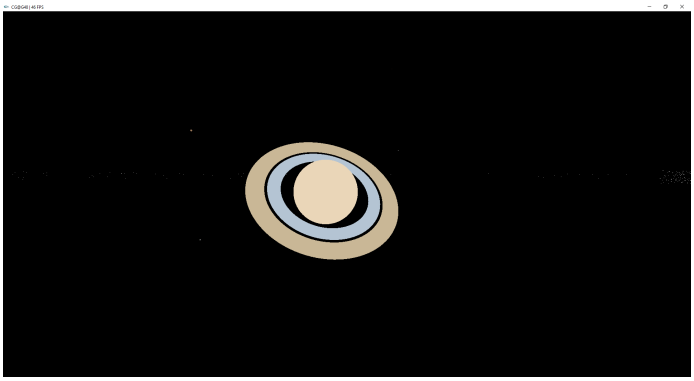


Figura 26: Saturno e respetivos satélites.

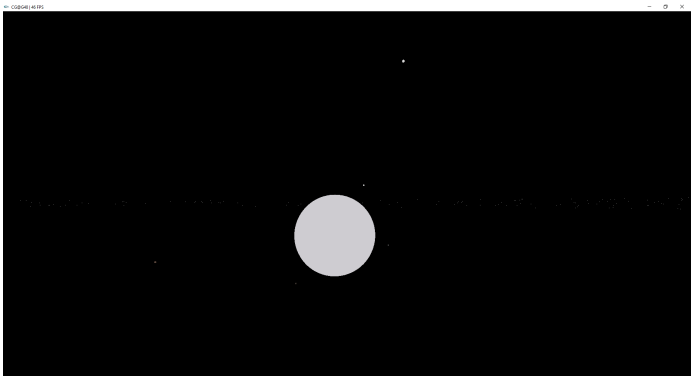


Figura 27: Urano e respetivos satélites.

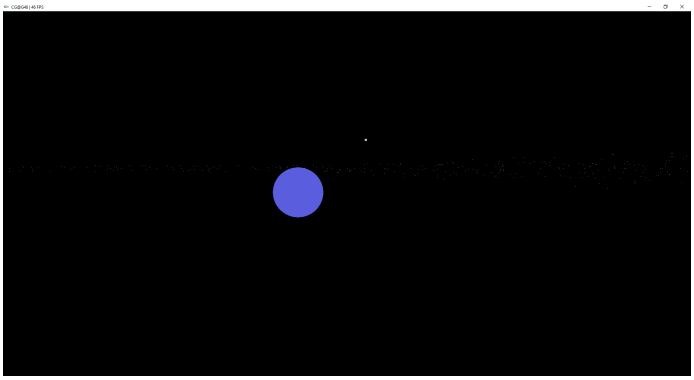


Figura 28: Neptuno e respetivos satélites.

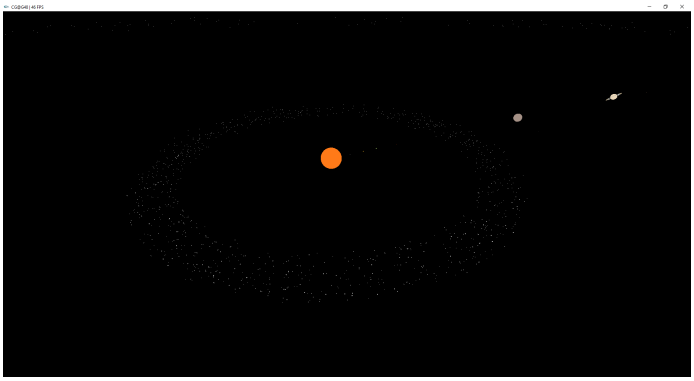


Figura 29: Cintura de asteroides.

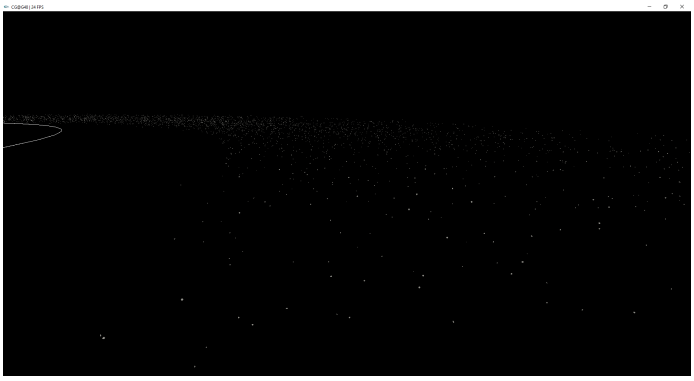


Figura 30: Cintura de Kuiper.

7.2 Cidade

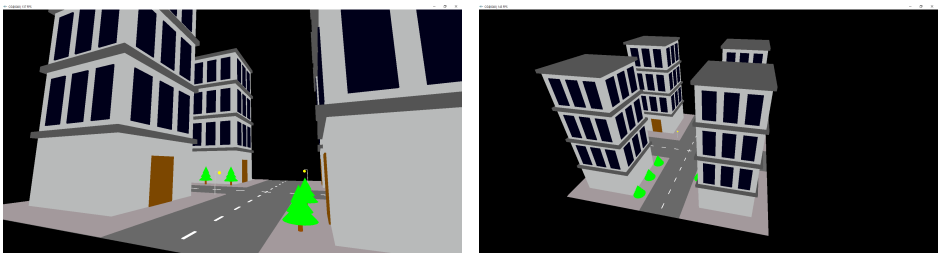
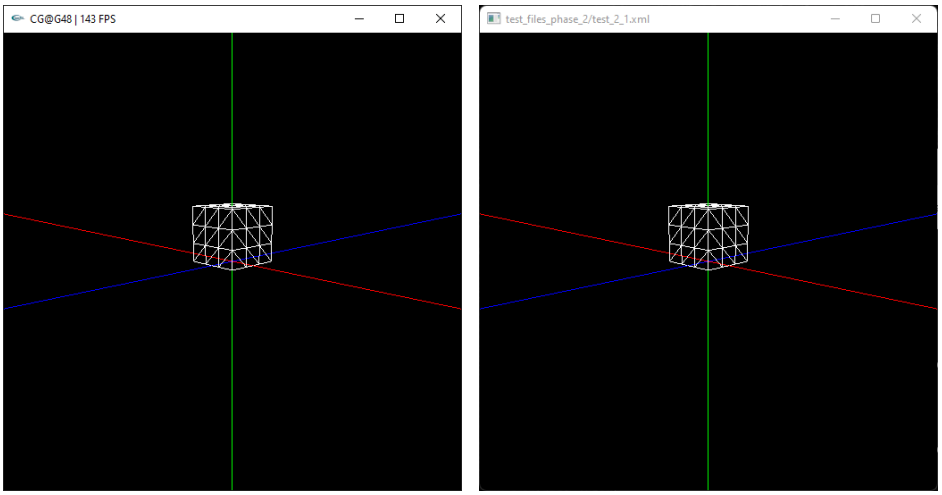


Figura 31: Cidade.

7.3 Testes

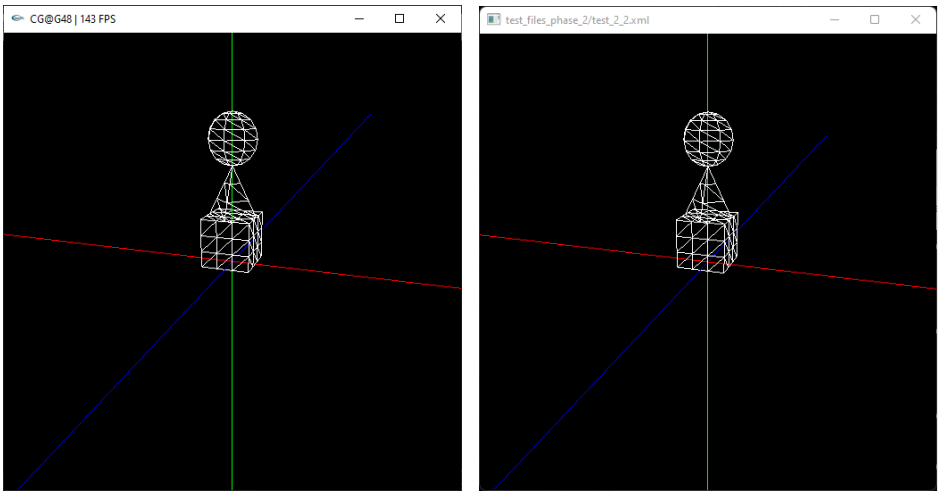
Teste 1



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

Figura 32: Resultados do Teste 1.

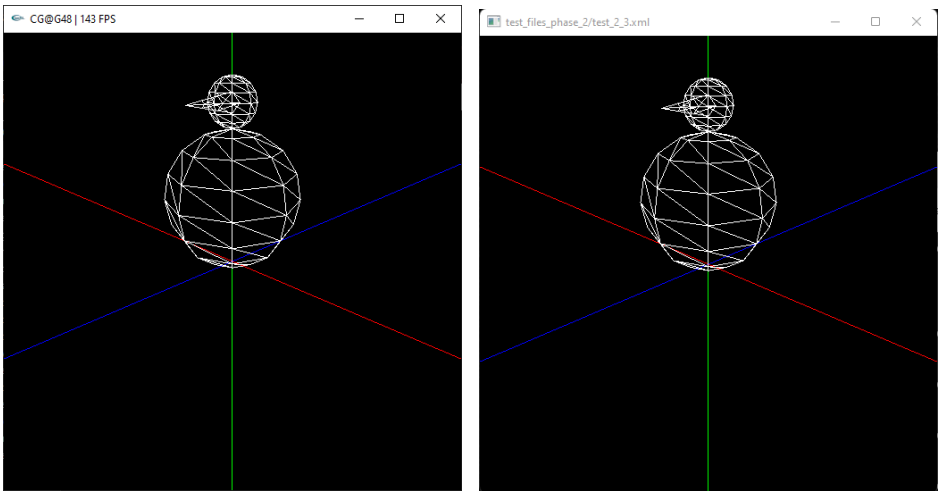
Teste 2



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

Figura 33: Resultados do Teste 2.

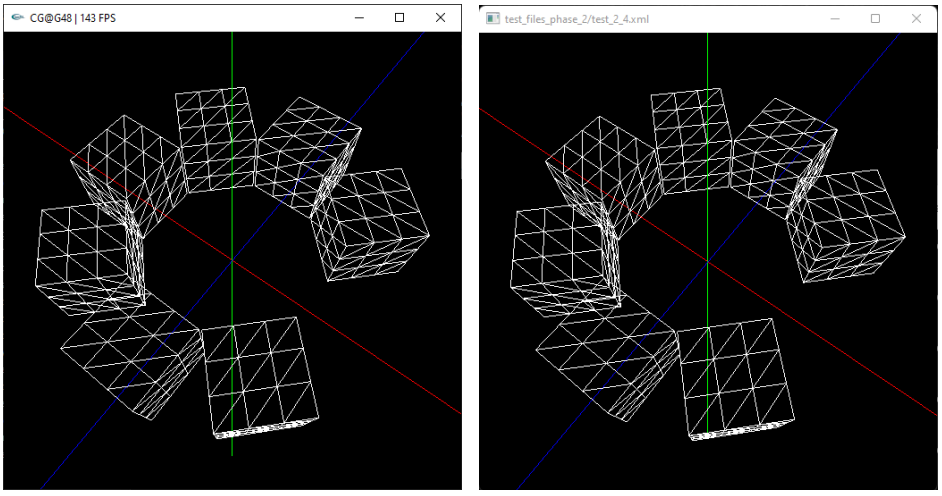
Teste 3



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

Figura 34: Resultados do Teste 3.

Teste 4



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

Figura 35: Resultados do Teste 4.

8 Conclusão

Nesta fase do projeto, consideramos, mais uma vez, que os objetivos foram cumpridos na íntegra e estamos satisfeitos com o resultado final.

A implementação das transformações geométricas e a criação do sistema solar revelaram-se tarefas simples, pelo que aproveitamos para introduzir novas funcionalidades essenciais ao projeto, maioritariamente direcionadas a proporcionar uma melhor experiência ao utilizador. Um exemplo disso é o menu para alternar os modos de rasterização e de câmara, sendo muito mais intuitivo fazê-lo da forma atual do que utilizando teclas que teriam de ser previamente memorizadas, algo que se tornaria cada vez menos viável à medida que o número de modos disponíveis aumentasse. Outro exemplo é a câmara FPS, dado que as *demo scenes* desenvolvidas são mais complexas em relação à primeira fase, incluíndo translações de várias figuras, o que faz com que seja necessário que o utilizador possa navegar ao longo de todo o cenário que, por vezes, é bastante extenso. Para além disso, foram também adicionados alguns extras ao cenário, como as cinturas de asteroides e as órbitas.

Contudo, ainda que tenha sido atualizado, concluímos que o nosso modo explorador da câmara não funciona ao nível pretendido e tencionamos vir a melhorá-lo nas próximas fases. Ao contrário de ter, primeiro, que se posicionar no ponto central sobre o qual vai circular e só, em seguida, ativar o modo explorador, pretendemos que o utilizador possa escolher a zona que quer explorar já com o modo ativo. Desta forma, terá de ser calculado o grupo do ficheiro XML de *input* mais próximo com base na posição atual da câmara.

Outro dos aspetos que pretendemos melhorar passa por aumentar a eficiência do programa aquando da utilização da extensão "belt", uma vez que utilizamos n VBO's para renderizar n asteroides, o que aumenta a ineficiência. Deste modo, pretendemos considerar uma cintura como primitiva e, portanto, passamos a utilizar apenas um VBO para renderizar toda a cintura.