

Trabalho Prático CG - 23/24

Diogo Paiva¹[A100760], João Magalhães¹[A100740], Jorge Rodrigues¹[A101758], e
Rodrigo Gomes¹[A100555]

Universidade do Minho, Braga Gualtar, Portugal

Abstract. A pedido da UC de Computação Gráfica, fomos conceber um Motor Gráfico 3D que será capaz de exibir o nosso Sistema Solar.

Keywords: Computação Gráfica · Motor Gráfico 3D · Sistema Solar

1 Introdução

No nicho da computação gráfica, os motores 3D são extremamente impactantes na criação de ambientes imersivos, que contribuem para diversas atividades humanas, tais como, os videojogos, o cinema, os simuladores e outras centenas de aplicações interativas. Este projeto servirá então como ensaio do grupo para construir uma ferramenta capaz de construir e expor cenas 3D.

O desenvolvimento será dividido em 4 partes, com progresso em cascata, que deverá acompanhar o conteúdo lecionado nas aulas da unidade curricular. Desta forma, conceitos como primitivas gráficas, transformações geométricas, *VBO's*, normais e texturas são implementadas em concordância temporal com as aulas práticas.

Este trabalho serve, acima de tudo, para aplicar os conhecimentos da representação 3D assimilados ao longo da unidade curricular, com um extra de, no final, ser capaz de gerar uma cena representativa do nosso sistema solar, através de um simples ficheiro *XML*.

2 Fase 1 – Primitivas gráficas

2.1 Gerador

O Gerador foi o primeiro modelo desenvolvido a pedido da **Fase 1**. Este programa tem como propósito apenas gerar as primitivas gráficas que vão ser descritas nesta mesma secção. O funcionamento do programa é simples, consiste apenas em receber como argumento a primitiva e os respetivos parâmetros requeridos para a construção da mesma.

Antes de especificar as técnicas usadas para a geração de modelos, achamos relevante mencionar como é que estes ficheiros das figuras a três dimensões são guardados. Para esse efeito, optamos por escrever para um ficheiro, ponto a ponto, com a respetiva ordem que cada polígono deve ser desenhado. Esta abordagem é menos eficiente a nível de memória, mas bastante rápida no desenho das figuras.

2.1.1 Plano

O plano foi a primeira primitiva desenvolvida pelo grupo. Esta figura assenta no plano XoZ , isto é, o valor relativo à componente y de todos os pontos do plano é 0. Este é gerado tendo em conta dois valores, sendo estes o tamanho do lado do plano mais o número de subdivisões por lado. Deste modo, o plano assemelha-se a um grelha.

O desenho desta primitiva é trivial, basta apenas ter em atenção a ordem como os triângulos são desenhados. Seguem as fórmulas que permitem a geração desta estrutura:

$$passo = lado / subdivisoes \quad (1)$$

$$x = -lado/2.0 + i * passo \quad (2)$$

$$z = -lado/2.0 + j * passo \quad (3)$$

Na expressão 2 e 3, o i e o j correspondem aos índices dos ciclos, começando ambos em 0. O $-lado/2.0$ surge pelo facto do plano ser centrado na origem.

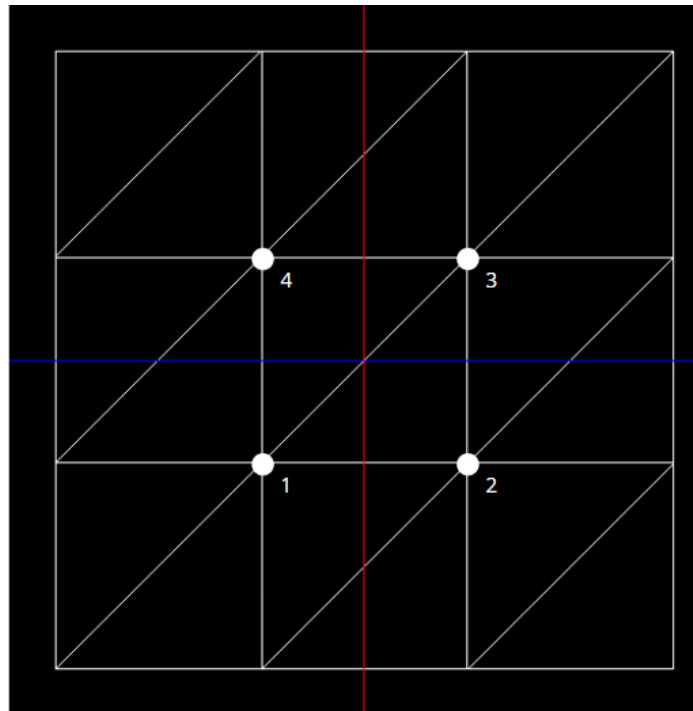


Figura 1: Plano com 2 unidades de lado e 3 subdivisões

Para terminar, vamos ainda mencionar a ordem da escrita dos pontos, que segue a seguinte ordem: Primeiro triângulo: 1, 2 e 3; Segundo triângulo: 1, 3 e 4

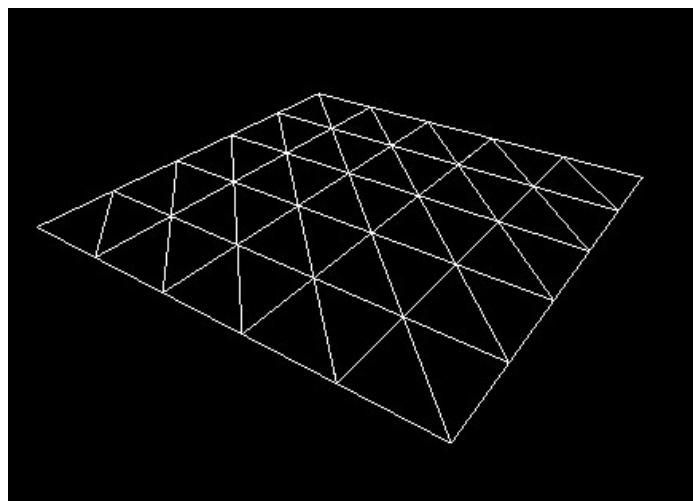


Figura 2: Plano com 5 unidades de lado e 5 subdivisões

2.1.2 Caixa

A caixa foi desenvolvida logo a seguir ao plano, isto porque uma caixa não passa de 6 planos idênticos aos desenvolvidos no ponto anterior. Esta figura continua a estar centrada na origem, mas com um novo obstáculo que recai sobre a orientação dos triângulos desenhados.

As fórmulas matemáticas são idênticas à anterior. Claro que, começando pela tampa e a base, o valor da componente y vai ser diferente do plano da secção anterior. Além disso, a orientação da tampa vai ser diferente da orientação da base do sólido. Com isto, podemos dividir o desenho do cubo em 3 etapas: Desenho da tampa e base; Desenho da face da frente e da face de trás; Desenho da face da direita e da face da esquerda.

Começando pela tampa e pela base, seria redundante repetir a face da *tampa*, isto porque a única coisa que varia face ao plano é o valor da componente y . Contudo, têm alguma relevância mostrar a base. As fórmulas matemáticas utilizadas são na mesma a fórmula 1, 2 e 3, mas a fase de desenho é ligeiramente diferente. Este caso da base é um dos casos onde queremos que seja a chamada parte de trás que fique à mostra.

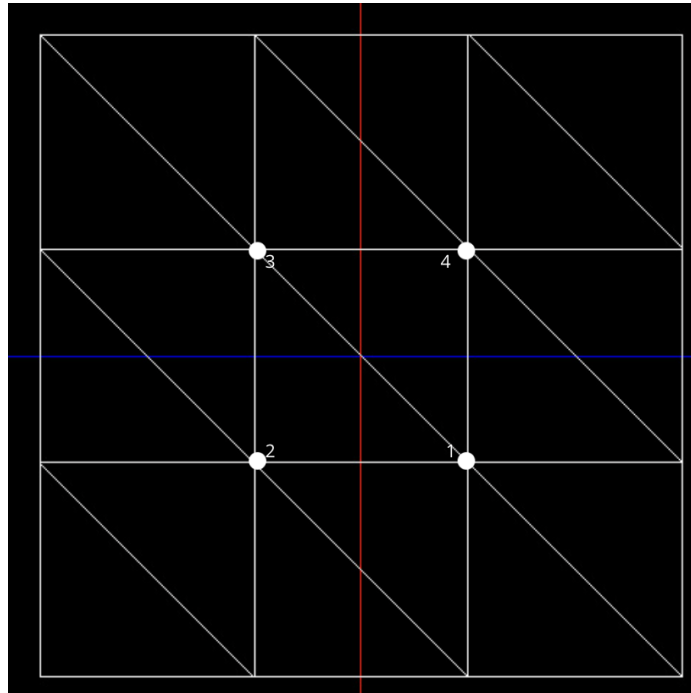


Figura 3: Base da caixa com 2 unidades de lado e 3 subdivisões

Assim, a ordem de desenho foi a seguinte: Primeiro triângulo: 3, 2 e 1; Segundo triângulo: 3, 1 e 4

De forma singela, os triângulos seguem uma orientação *clockwise*, algo que vai ser repetido para outras 2 faces da caixa. Os números dos pontos estão

espelhados ao da tampa, dando uma impressão que estamos mesmo a seguir a ordenação *counter-clockwise*.

Avançando para as restantes faces, por questões de simplificação, vamos apenas mostrar a face paralela ao plano XoY .

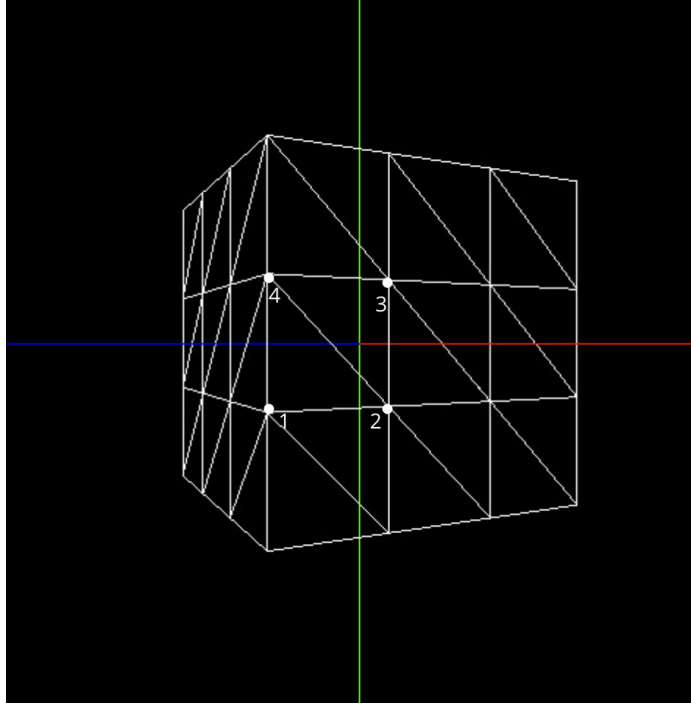


Figura 4: Base da caixa com 2 unidades de lado e 3 subdivisões

Começando pela matemática, precisamos das seguintes fórmulas:

$$passo = lado / subdiviso es \quad (4)$$

$$x = -lado / 2.0 + i * passo \quad (5)$$

$$y = -lado / 2.0 + j * passo \quad (6)$$

$$z = -lado / 2.0 \quad (7)$$

É muito semelhante com o que tem sido exposto até agora, apenas muda o plano ao qual a face é paralelo e, como geramos as faces aos pares, mudamos também o valor da componente invariante, neste caso, o z .

Relativamente à ordem de desenho dos triângulos, a que optada foi a seguinte: Primeiro triângulo: 4, 1 e 2; Segundo triângulo: 4, 2 e 3.

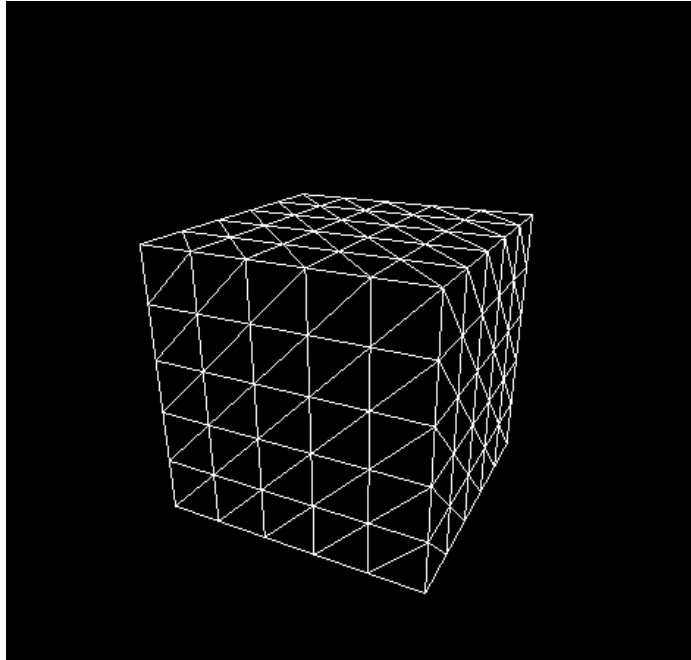


Figura 5: Caixa com 5 divisões e 1 unidade de lado

2.1.3 Esfera

Um dos maiores desafios desta fase foi perceber como desenvolver sólidos curvos com polígonos. Assim, como praticado nas aulas, a abordagem que escolhemos foi o uso de coordenadas esféricas que, muito facilmente, tornam o processo em algo iterável, uma qualidade desejada na computação. Para a geração da primitiva, são fornecidos parâmetros relativos ao raio, o número de fatias e também o número de camadas.

Relativamente às coordenadas esféricas, neste referencial surgem três componentes: (r, θ, φ) que podem ser convertidas para coordenadas cartesianas (usadas pelo motor) através das seguintes fórmulas.

$$z = r \cos \theta \cos \varphi \quad (8)$$

$$x = r \sin \theta \cos \varphi \quad (9)$$

$$y = r \sin \varphi \quad (10)$$

Fonte: https://www.songho.ca/opengl/gl_sphere.html

Além destas fórmulas, é preciso utilizar os valores das *fatias* e das *camadas*, para calcular o passo, de certa forma semelhante às primitivas já desenvolvidas. As fórmulas utilizadas foram as seguintes:

$$sliceStep = 2.0 * \pi / slices; \quad (11)$$

$$stackStep = \pi / stacks; \quad (12)$$

Com estas expressões, fica facilmente computável o sólido com dois ciclos, semelhante ao que foi construído anteriormente.

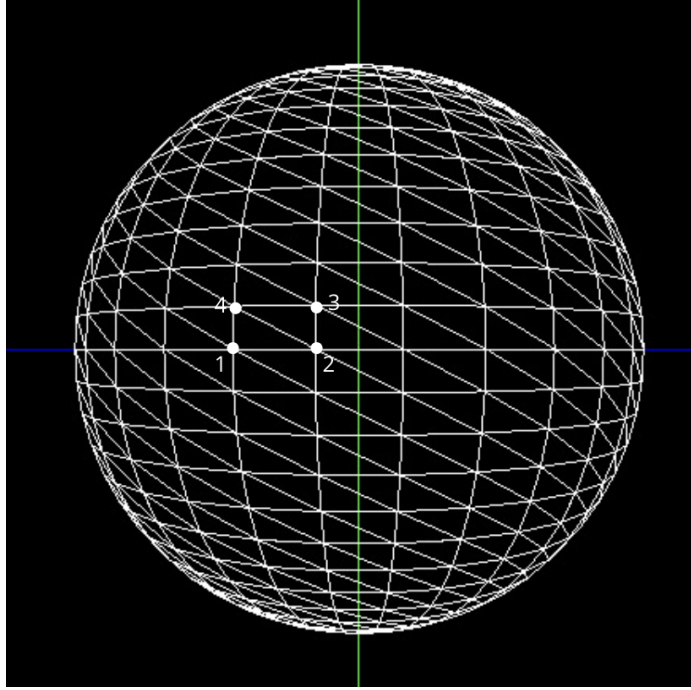


Figura 6: Exemplo de uma esfera

As nossas esferas são construídas de baixo para cima, formando quadrados, assim como está ilustrado na figura. Relativamente à ordem de desenho dos triângulos, optamos pela seguinte organização: Primeiro triângulo: 2, 3 e 4; Segundo triângulo: 4, 1 e 2.

2.1.4 Cone

Finalmente, surge o cone. Esta é a primitiva mais complexa de se desenvolver, isto porque a abordagem para desenhar a sua face lateral segue uma lógica notável. O seu desenho pode ser dividido em duas partes: A base do cone e a face lateral. Para este efeito, o gerador precisa de receber os parâmetros para o raio, a altura, o número de fatias e o número de camadas, relativamente semelhante à esfera.

Começando com a base, a fórmula 11 será novamente utilizada, e adicionamos uma nova fórmula:

$$x = radius * \sin \theta \quad (13)$$

$$z = radius * \cos \theta \quad (14)$$

A partir desta fórmula, é possível calcular os três pontos necessários da base do cone, dividida em fatias. Para esse efeito, visto que o cone está centrado na origem, sabemos sempre que um dos vértices do polígono será a origem e também que, os restantes podem ser calculados com recurso às equações 13 e 14, que facilmente é computável. O terceiro ponto consistem em adicionar o *step* ao θ .

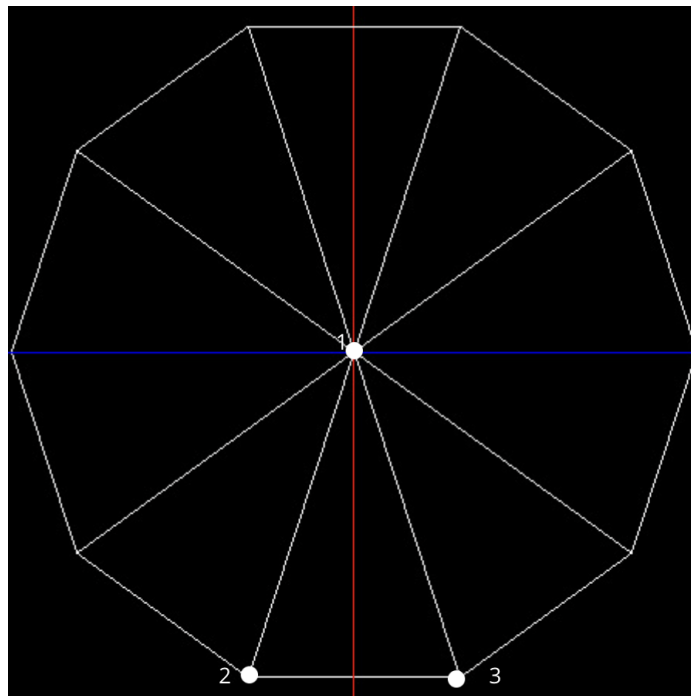


Figura 7: Base de um cone com 10 fatias

Desta forma, cada polígono segue a ordem 1, 2 e 3, desenhando assim triângulo a triângulo da base.

Avançando para a face lateral, a matemática fica significativamente mais complexa do que a primeira parte da construção do cone.

Começando com a matemática, um novo conceito é introduzido, sendo este a questão do uso de dois raios diferentes. Assim, já que o cone está dividido em camadas, consideramos o *bottomradius* como o raio da camada mais inferior que estamos a usar para o desenho, e o *topradius* como o raio da camada imediatamente em cima. Só desta forma conseguimos calcular os pontos com precisão para desenhar a figura. As fórmulas que nos ajudam a calcular estes dois raios são as seguintes:

$$topRadius = (altura - alturaAtual - stackStep)/(altura/raio); \quad (15)$$

$$bottomRadius = (altura - alturaAtual)/(altura/raio); \quad (16)$$

$$y = alturaAtual \quad (17)$$

$$y = alturaAtual + stackstep \quad (18)$$

Facilmente percebemos que a diferença entre ambas as fórmulas está na inclusão da *stackStep* para chegar ao raio mais pequeno, isto porque construímos a figura da base até ao topo. Será ainda preciso utilizar as fórmulas 13 e 14, para, de forma semelhante à base, conseguir calcular os pontos.

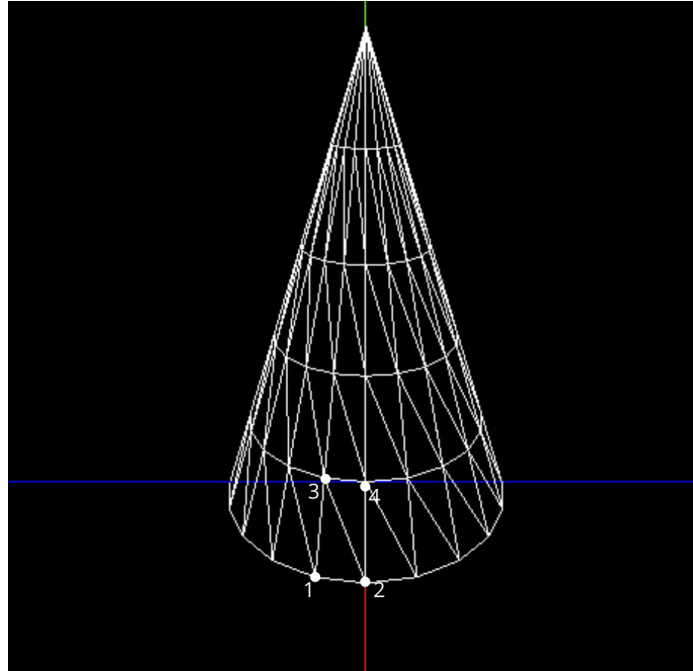


Figura 8: Face lateral de um cone

Para desenhar cada triângulo, a ordem foi a seguinte: Primeiro triângulo: 2, 4 e 3; Segundo triângulo: 3, 1 e 2.

2.1.5 Anel

Por fim, o grupo, com o intuito de consolidar todo o conhecimento adquirido na construção de figuras, apresenta o anel. Este é uma primitiva de complexidade

semelhante à construção da base de um cone. O seu desenho pode ser dividido em duas partes: o anel interior e o anel exterior. Para isso, o gerador precisa de receber os parâmetros do raio exterior, raio interior e o número de fatias.

Ambos os anéis têm construções semelhantes. Por conveniência, o grupo decidiu demonstrar apenas a construção do anel interior. Para o restante da figura, seria apenas necessário alterar o tamanho do anel. A construção do anel interior seguiu as formulas supramencionados no cone com uma ligeira alteração. Dado que não existia o ponto comum (origem) a todos os triângulos, foi necessário calcular mais um ponto para substituir o papel desse ponto origem na construção do triângulo.

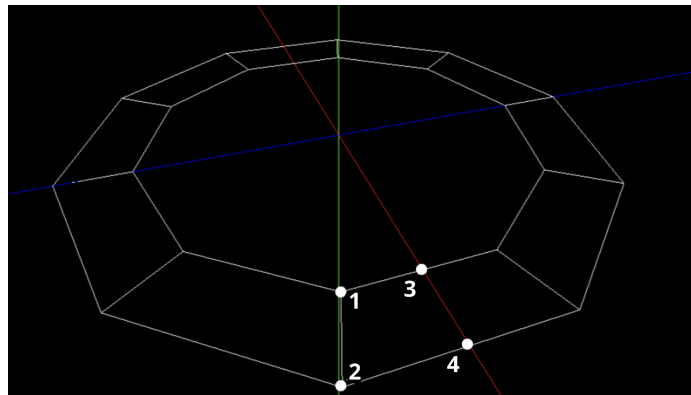


Figura 9: Face externa e interna de um anel

Para a construção individual de cada triângulo foi seguida a seguinte ordem : Primeiro triângulo:1,2,4 Segundo triângulo:4,3,1

2.2 Motor

O Motor é a componente que renderiza a cena que, por sua vez, foi escrita previamente num ficheiro alvo no formato *XML*.

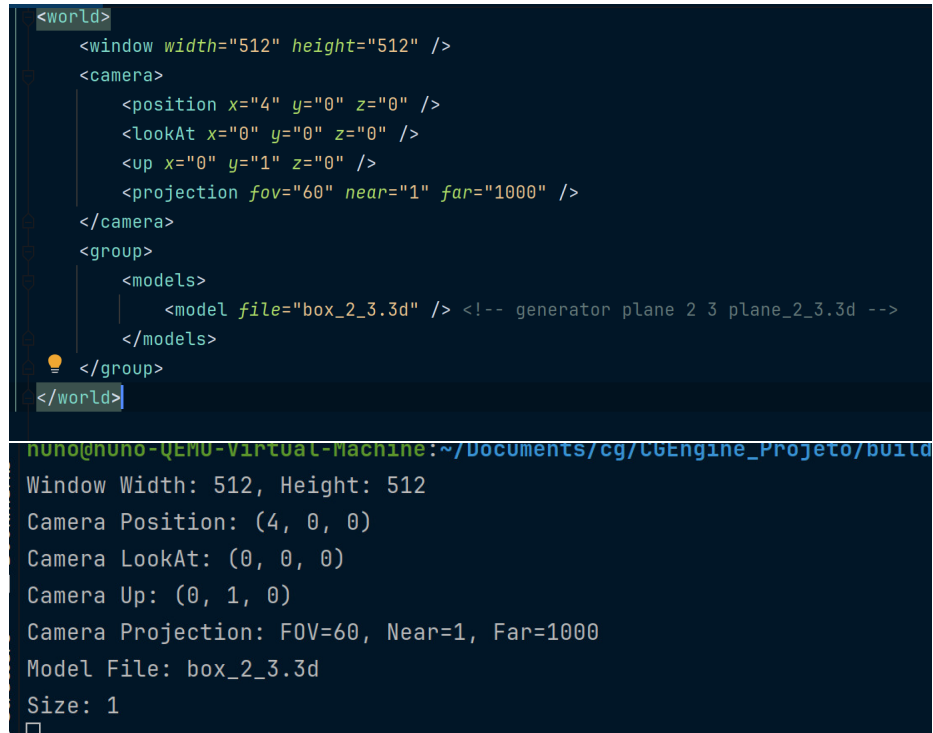
Para a primeira fase, o motor é bastante elementar, isto porque as funcionalidades requeridas não precisam de grande lógica e de configurações extras da biblioteca *Glut*. Contudo, podemos dividir esta secção em duas partes, sendo estas o *Parser* e o *Motor 3D*.

2.2.1 Parser

Este elemento do motor tem apenas uma única função que corresponde a ler a cena do formato *XML* e carregar os dados num formato que o Motor entenda. Para este efeito, o grupo optou por seguir a recomendação do enunciado e usar a biblioteca *tinyXml* que agilizou bastante o processo de textitparsing . Nesta secção, achamos que faça sentido referir também as classes de utilidades diversas que definimos, com o intuito de organizar a informação.

Começando pela classe **Ponto**, apenas guarda informações das coordenadas de um ponto, com a vantagem acrescida de poder ter métodos que serão particularmente úteis para a próxima fase.

Mais importante para o motor, desenvolvemos a classe **Cena** que aglomera todas as componentes para a cena que vai ser representada, mais concretamente, informações sobre a janela, a câmara e os modelos.



```

<world>
  <window width="512" height="512" />
  <camera>
    <position x="4" y="0" z="0" />
    <lookAt x="0" y="0" z="0" />
    <up x="0" y="1" z="0" />
    <projection fov="60" near="1" far="1000" />
  </camera>
  <group>
    <models>
      <model file="box_2_3.3d" /> <!-- generator plane 2 3 plane_2_3.3d -->
    </models>
  </group>
</world>

```

```

nuno@nuno-QEMU-Virtual-Machine:~/Documents/cg/CgEngine_Projeto/build
Window Width: 512, Height: 512
Camera Position: (4, 0, 0)
Camera LookAt: (0, 0, 0)
Camera Up: (0, 1, 0)
Camera Projection: FOV=60, Near=1, Far=1000
Model File: box_2_3.3d
Size: 1

```

Figura 10: Cena lida apartir do ficheiro *XML*

2.2.1 Motor 3D

Com a informação relativa à cena, fica fácil construir o motor que, por sua vez, segue o standard de uma aplicação que utiliza a biblioteca *Glut*.

Começando com o desenho dos modelos, a lógica está a nível do gerador, o motor apenas desenha os polígonos pela ordem que estão no ficheiro gerado pela a sua contraparte. Isto pode ser reduzido a um ciclo que itera *três a três*.

A câmara e a janela têm uma construção semelhante. Apenas acedem à informação da cena. A começar pela janela, a cena possui informação relativa à altura e largura da janela. Já a câmara, requiere de um pouco de mais informação, mais concretamente, como a projeção será feita, utilizando os campos *Fov*, *Near* e *Far*, a sua posição, será apenas um ponto e, finalmente, o ponto para onde a câmara estará apontada.

3 Extras

Fora os requisitos da fase, o grupo decidiu implementar as seguintes funcionalidades ao programa:

3.1 Movimento da Câmara

Com *input* do teclado, um utilizador consegue mover a câmara pela cena, o que torna a experiência de utilização do motor muito mais imersivo. Também é possível rodar a câmara, o que dá uma experiência muito parecida com aquilo que temos na realidade. Ainda é possível ativar e desativar os eixos, uma funcionalidade particularmente útil para desenvolver este relatório.

3.2 Contador de Frames

Como o desempenho é essencial para motores gráficos, optamos por implementar esta funcionalidade que imprime na consola o número de *frames* que a nossa cena está a ser renderizada. Achamos que esta funcionalidade será especialmente importante para as fases futuras.

4 Conclusão

No desenvolvimento deste projeto, os membros do grupo tiveram a oportunidade de experimentar tecnologias, aplicar conceitos, tudo com liberdade para a implementação escolhida.

Até ao momento, podemos apenas retirar conclusões relativas às práticas que foram desenvolvidas. Nesta etapa, o *Gerador* foi o maior obstáculo, isto porque as primitivas requerem um traquejo com expressões matemáticas que ficaram um pouco esquecidas desde o início da licenciatura. Além disso, existiu também a questão das orientações dos polígonos que não foi tão dramático como o problema anterior. O *Motor*, nesta fase, é ainda muito simples, sendo que pouca atenção foi dirigida para esta parte. Esperamos que nas fases futuras, este cenário seja invertido.

Em suma, conseguimos cumprir com os objetivos da primeira fase, relativa ao desenho das primitivas, sempre com alguma atenção ao que será o projeto final e ao que é lecionado na unidade curricular.