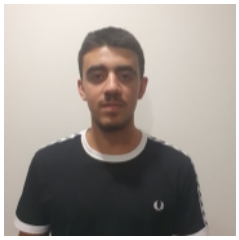
Computação Gráfica

Trabalho Prático

Fase 1 - Primitivas Gráficas

LEI - 2023/2024

João Barroso	Maurício Pereira	Jorge Teixeira	Hugo Dias
A95195	A95338	A100838	A100758




Grupo 08

Universidade do Minho

Índice

1 Introdução

No contexto deste projeto, foi-nos atribuída a tarefa de desenvolver um mecanismo tridimensional fundamentado num ambiente gráfico, dividido em quatro fases. Este relatório incide sobre a primeira fase do projeto, na qual nos foi solicitado implementar duas aplicações: um gerador, apto a criar ficheiros com os dados necessários para o modelo, e um motor, responsável por ler ficheiros de configuração no formato XML e apresentar os modelos desejados para visualização.

1.1 Classes

1.1.1 Point

Como forma de desenvolver o mecanismo em 3D será necessário primeiramente definir uma forma de representar as posições no espaço. Para isso, decidimos utilizar a unidade básica de geometria, o Point. Point esse que armazenará informações de 3 coordenadas relativas aos eixos x,y e z.

1.1.2 Triangle

Para evitar a repetição de vértices, assunto que será tratado mais adiante, esta classe representa um triângulo que inclui as posições no vetor de pontos de cada ponto que o compõe, gerado ao criar a primitiva gráfica.

1.1.3 VBO

Para evitar problemas de eficiência na nossa engine, recorremos à criação de uma classe representativa de um VBO (Vertex Buffer Object). A classe em si será responsável por armazenar dois buffers de um dado modelo. Um destes buffers vai ser usado para armazenar os vértices, e outro para guardar os índices dos vértices dos triângulos.

1.1.4 World

A responsabilidade de ler toda a configuração proveniente do ficheiro no formato XML estará sob o engine. E toda a informação por ele lida será. posteriormente, armazenada num objeto denominado por World, que vai por sua vez guardar todas as configurações relativa à câmara, janela e grupos associados a este.

1.1.5 Model

Como forma de fazer a representação das primitivas gráficas, à frente explicadas em detalhe, optamos por criar uma classe denominada por Model. Classe esta que irá conter, por sua vez, 2 estruturas. A primeira servirá para guardar todos os vértices da primitiva a si associada, e outra para guardar os triângulos que o vão compor.

1.2 Ferramentas Utilizadas

1.2.1 TinyXML2

Além da biblioteca OpenGL, utilizamos também a biblioteca TinyXML2 para simplificar o processo de análise sintática do ficheiro que contém as configurações gráficas, bem como a indicação dos ficheiros previamente gerados que serão carregados e renderizados.

2 Aplicações

2.1 Generator

Como mencionado anteriormente, o gerador deve criar os ficheiros que contêm a informação sobre as primitivas gráficas a desenhar, ou seja, neste caso, o conjunto de vértices que as compõem, de acordo com os parâmetros recebidos.

2.1.1 Parsing dos argumentos

É realizado primeiramente, um parsing dos argumentos passados como input, com o objetivo de verificar se o sólido que se pretende construir é realmente possível.

2.1.2 Geração do modelo

Após a verificação do passo anterior, segue-se a geração do sólido pedido. É realizado o cálculo dos vértices da primitiva gráfica escolhida, fase esta que requer uma especial atenção, para conseguirmos fornecer ao motor um modelo o mais eficiente possível. Para alcançar esse objetivo, exploramos duas abordagens diferentes.

Primeiramente, a aplicação gerava apenas os pontos de todos os triângulos presentes na primitiva a ser construída. O problema desta abordagem foi ser muito pouco eficaz, uma vez que resultava em muitos pontos repetidos devido à presença de um mesmo ponto em vários triângulos. Ora todos estes pontos repetivos acabavam por consumir uma quantidade grande de memória o que não se demonstrava ser o ideal, o que nos levou a pensar noutra abordagem.

Finalmente, optamos por uma abordagem que procurava equilibrar um pouco a abordagem anterior, isto é, permitir a repetição dos vértices mas de uma forma moderada para que ainda assim podessemos garantir a localidade espacial. Assim continuamos a conseguir ter a localidade espacial necessária, e poupamos a memória que com a outra abordagem seria gasta em demasia.

Depois da geração do modelo, com todo o cálculo necessário para os vértices e triângulos, é devolvido o objeto "Modelo".

2.2 Engine

A aplicação engine processa um ficheiro de configuração em XML, utilizando a biblioteca TinyXML2 para interpretar os dados e armazená-los num objeto "World". Este motor é responsável por ajustar as configurações da câmara, definir a janela de visualização e representar os resultados gráficos dos ficheiros de modelos 3D gerados pelo "generator".

Para garantir um bom desempenho, a aplicação utiliza Vertex Buffer Objects (VBOs). Estes são buffers armazenados na memória da placa gráfica que permitem que os modelos sejam renderizados diretamente pela mesma, evitando transferências frequentes de dados entre a CPU e a GPU. Durante a leitura dos ficheiros de modelo, são criados e preenchidos dois buffers - um para os vértices e outro para os índices - para cada modelo.

Uma funcionalidade importante implementada é o modo explorador da câmara, que facilita a visualização de diferentes faces das primitivas gráficas. Isso também ajuda no processo de depuração, permitindo visualizá-las de ângulos diferentes e circundá-las para verificar a conformidade com a regra da mão direita.

O modo explorador da câmara pode ser ativado ou desativado premindo a tecla "c", sendo que está desativado por padrão. As teclas de seta são utilizadas para ajustar os ângulos vertical e horizontal da câmara, enquanto as teclas "+"e -"são usadas para aproximar ou afastar a câmara do objeto.

Outra funcionalidade implementada é a contagem do número de frames por segundo, o que facilita a comparação das diferentes estratégias de geração de modelos apresentadas anteriormente.

3 Primitivas Gráficas

Para esta fase, as primitivas geométricas implementadas foram o plano, a caixa, a esfera e o cone.

3.1 Plano

A criação de um plano requer a definição de dois parâmetros importantes: o número de divisões e o comprimento dos lados. O número de divisões irá influenciar o tamanho do plano. Considerando n o número de divisões, o plano irá ter $n \times n$ quadrados sendo cada um destes visto como a junção de dois triângulos.

O desenho dos pontos que definem os triângulos segue um padrão específico, partindo da esquerda para a direita e de frente para trás. A primeira aresta a ser construída é a mais à esquerda (com menor valor de x), e essa aresta é construída de frente para trás (com maior valor de z para menor valor de z). A forma será gerada com base num ponto inicial, com os restantes pontos a serem calculados através de dois ciclos aninhados que percorrem o eixo do X e do o Z. Cada um destes ciclos tem n iterações, que correspondem ao número de divisões, anteriormente mencionados.

É importante realçar ainda que o plano será desenhado no plano XZ, mantendo a coordenada de Y com o valor 0, e centrado na origem, pelo que as coordenadas de cada ponto correspondem a metade do valor de X e a metade do valor de Z.

3.2 Caixa

Para a criação desta primitiva gráfica, são pedidos dois parâmetros: o comprimento dos lados e o número de divisões - tal como no plano. Podemos ver a caixa como a união de 6 planos concorrentes paralelos a $x0z$, $y0z$ e $x0y$, que representam as suas 6 faces.

Para cada plano iremos ter a mesma abordagem explicada anteriormente em relação à criação dos pontos, isto é, para cada plano existe um ponto inicial e dois ciclos aninhados a contruír os restantes pontos.

3.3 Esfera

A criação da esfera já é mais complexa, uma vez que são pedidos três parâmetros: o raio, o número de camadas e o número de fatias. Ao criar a esfera, ela deve estar centrada na origem. Além disso, é importante lembrar que a esfera é dividida em $n \times m$ retângulos, cada um formado por 2 triângulos, onde n é o número de camadas e m é o número de fatias.

A esfera é construída com base em coordenadas esféricas, isto é, através de um raio e dois ângulos α e β (inicialmente definidos como 0) tendo o seu processo de criação a começar do centro para as pontas. Para gerar esta figura, utilizamos uma estratégia iterativa sobre os parâmetros de número de camadas e número de fatias. Cada iteração dos ciclos corresponde a uma camada e uma fatia da esfera, respectivamente. Durante cada iteração, são calculados os vértices correspondentes e adicionados ao vetor de vértices.

3.4 Cone

Para a criação do cone, são pedidos quatro parâmetros: o raio, a altura, o número de camadas e o número de fatias. O cone vai estar centrado na origem e a base no plano XZ. Este será dividido em $n \times m$ retângulos, cada um constituído por 2 triângulos, onde n é o número de camadas e m o número de fatias.

A sua realização foi dividida em duas partes: a base e a superfície lateral. Primeiro são gerados os pontos da base. Esta é feita com triângulos no plano XZ direcionados para o

centro da circunferência, ou seja, todos os triângulos partilham o ponto $(0, 0, 0)$. O número de triângulos depende do número de fatias verticais, tendo em conta que quanto maior for o seu número, mais "redonda"ficará a circunferência.

De seguida, é preciso construir os triângulos que formam a superfície lateral do cone. Estes triângulos são construídos por camadas, de baixo para cima. Para tal, de camada para camada, o raio diminui e as coordenadas de y aumentam proporcionalmente.

4 Resultados Finais

4.1 Testes

Nesta secção, encontram-se os testes que obtivemos e que em todos deram o resultado esperado. em baixo ficam as prints dos resultados dos diferentes cenários referidos anteriormente:

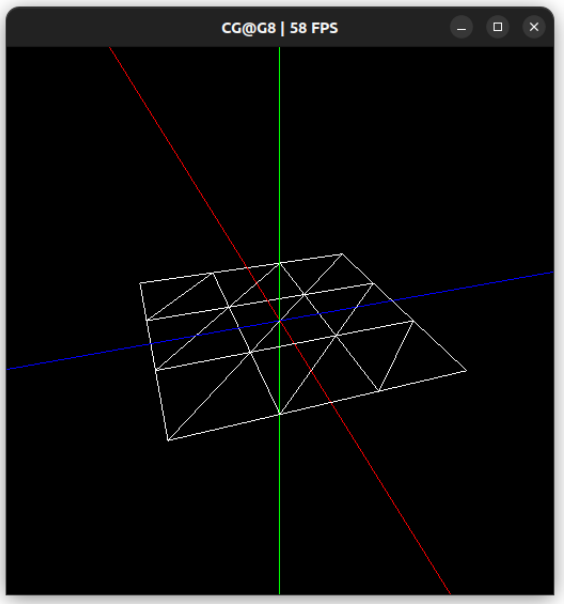


Figura 1: Plano

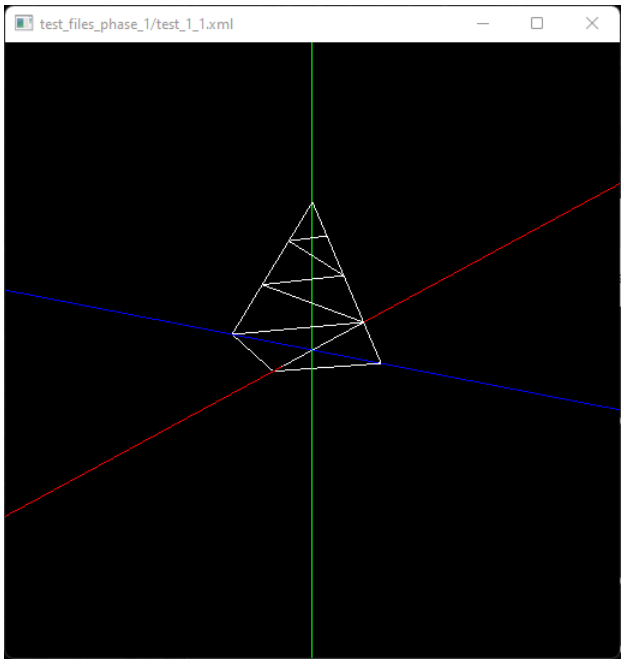


Figura 2: Cone

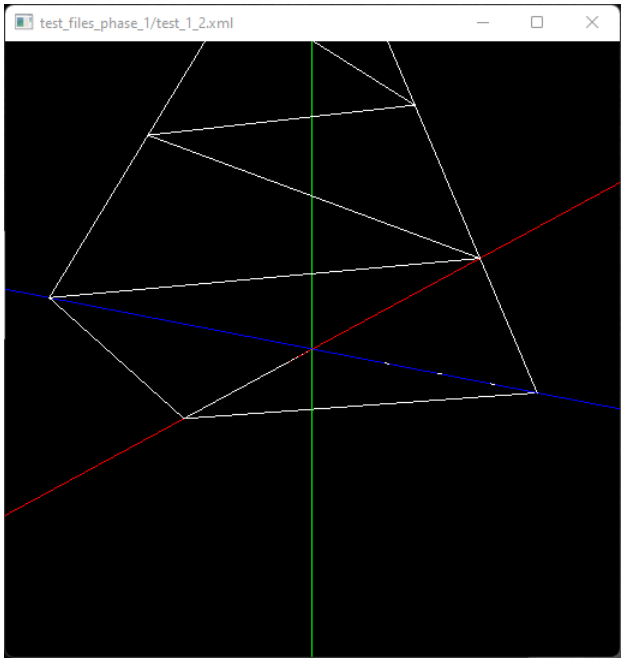


Figura 3: Cone Aproximado

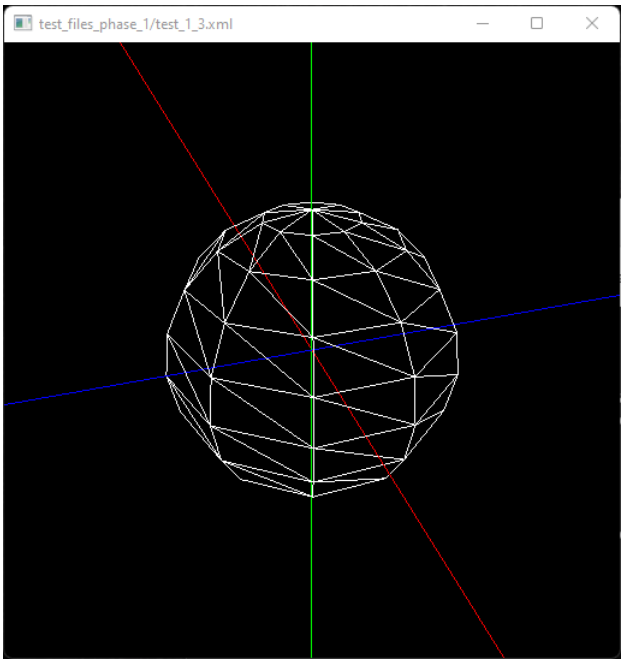


Figura 4: Esfera

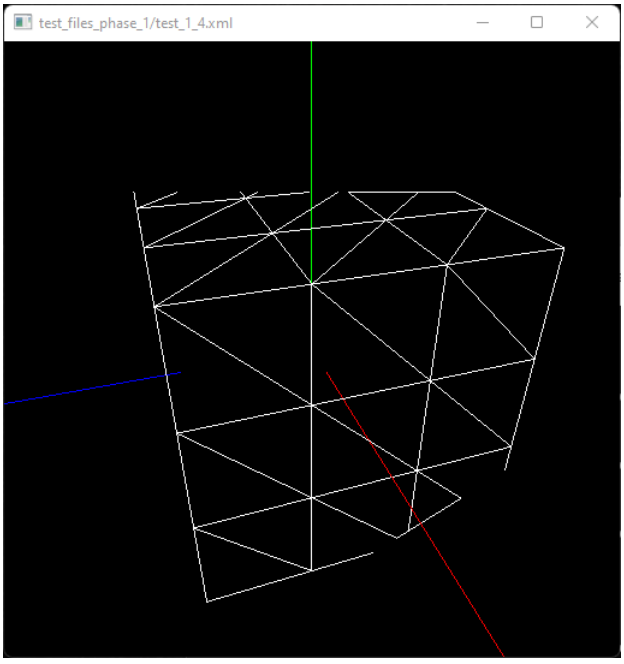


Figura 5: Caixa

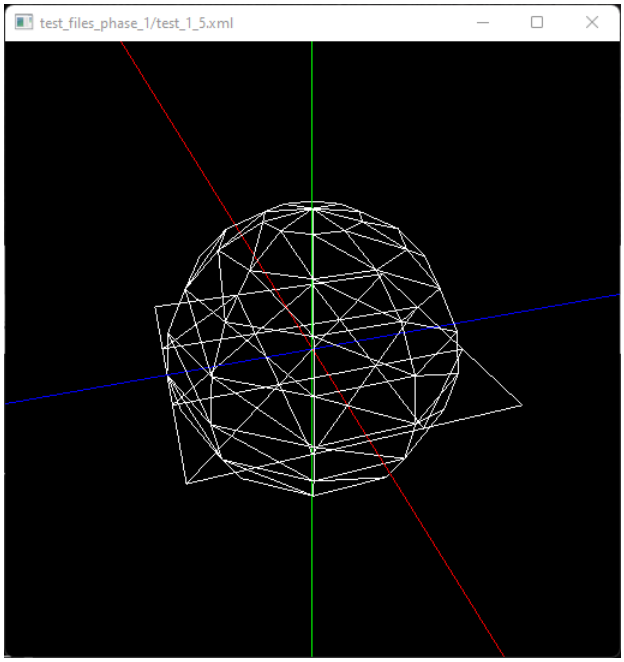


Figura 6: Plano + Esfera

Como já foi referido, todos os resultados obtidos foram os esperados. Tivemos algumas dificuldades a conseguir obtê-los, mas ficamos satisfeitos com o resultado final. Nas próximas fases, temos como objetivo adicionar primitivas gráficas extras para desenvolver ainda mais as nossas capacidades em computação gráfica.

4.2 Diferentes perspetivas dos Resultados Obtidos

Nesta secção mostramos alguns dos testes extra que fizemos. Como forma de desenvolver mais as nossas capacidades gráficas decidimos criar novos ficheiros .xml para os resultados obtidos e alterar alguns valores de forma a obter resultados visualmente mais apelativos das figuras que nos foram propostas desenvolver, ou seja, uma representação mais afastada e de maior dimensão. Os resultados finais encontram-se nas figuras abaixo:

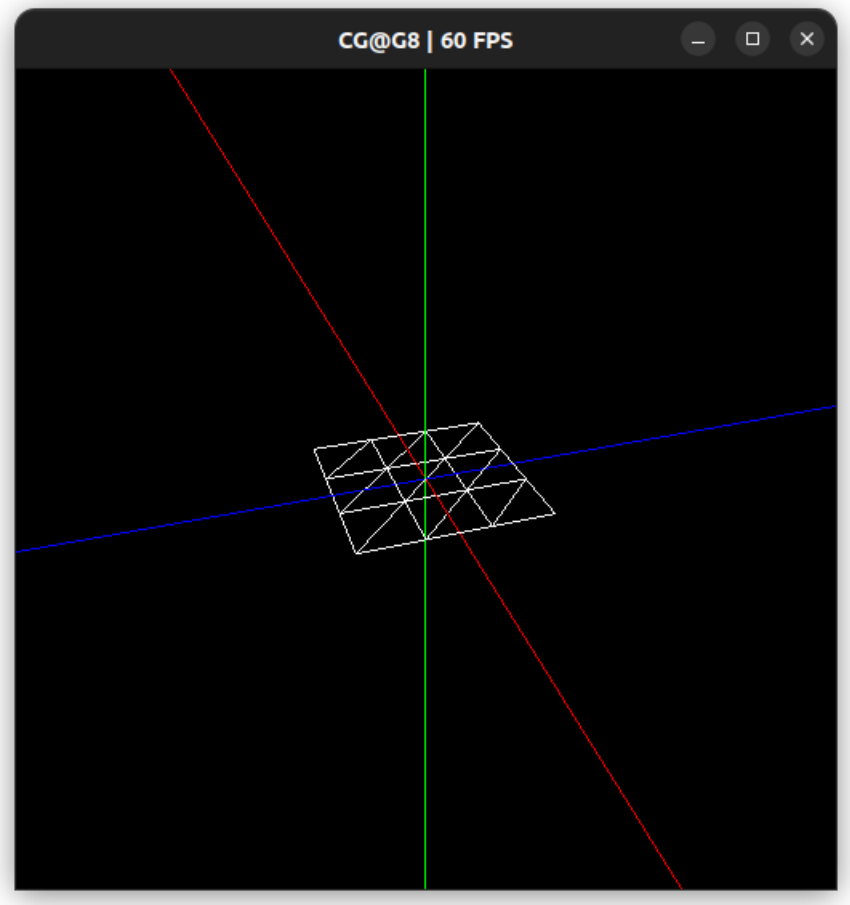


Figura 7: Plano Perspetiva Extra

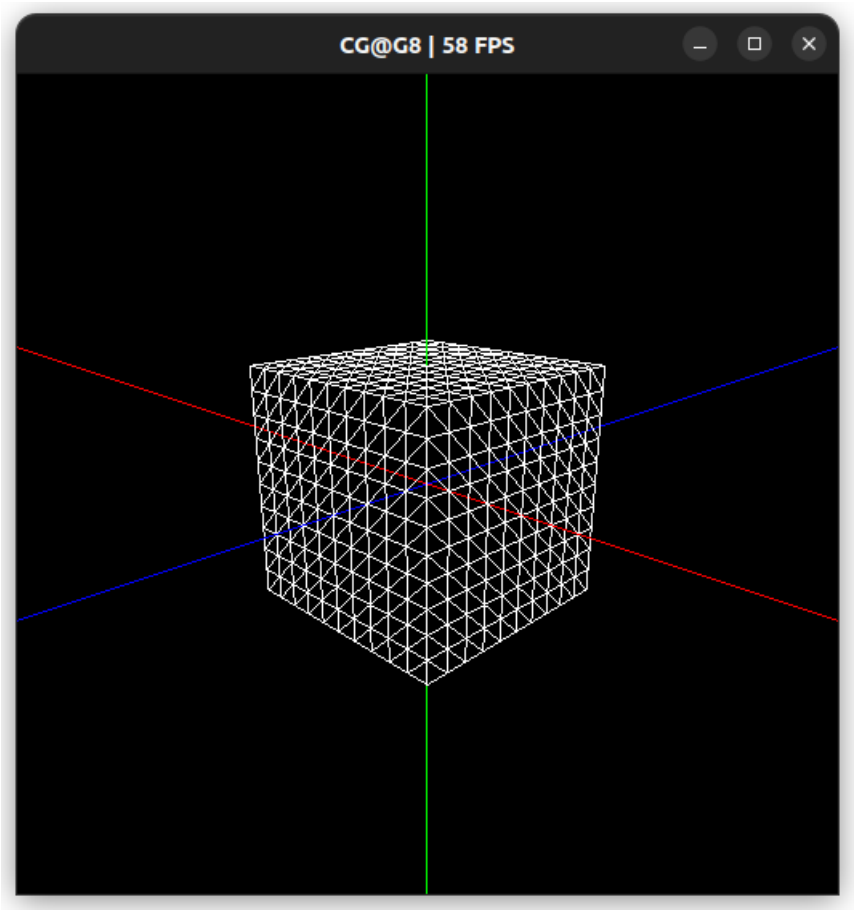


Figura 8: Caixa Perspetiva Extra

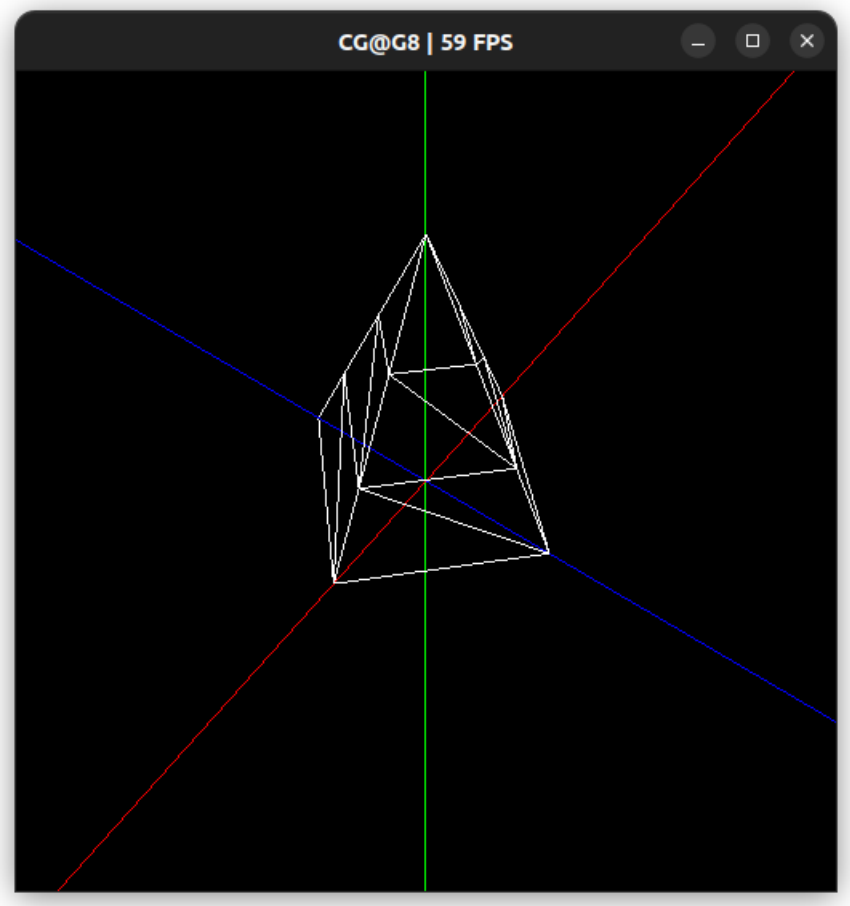


Figura 9: Cone Perspetiva Extra

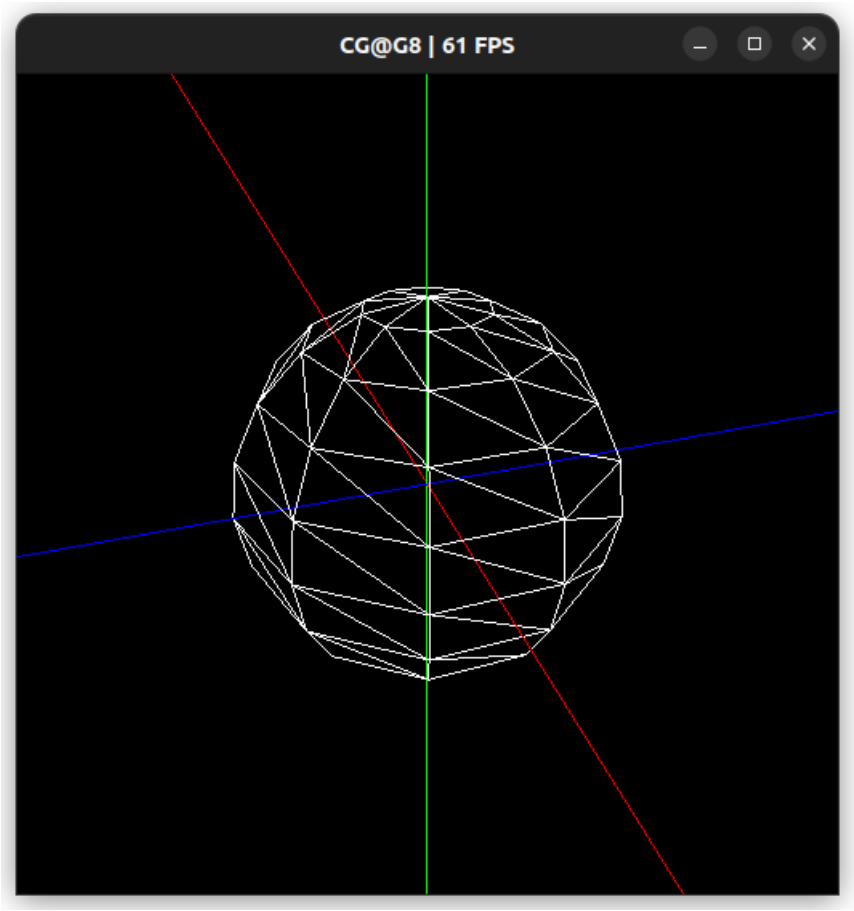


Figura 10: Esfera Perspetiva Extra

5 Conclusão

A conclusão desta etapa inicial do projeto possibilitou a consolidação dos conhecimentos transmitidos em sala de aula, especialmente em relação à utilização e domínio de ferramentas e conceitos relacionados à Computação Gráfica, como OpenGL e Glut. Além disso, adquirimos familiaridade com uma nova linguagem, C++, e exploramos novas áreas, como a leitura de arquivos com extensão XML. Globalmente, consideramos que alcançamos plenamente os objetivos definidos para esta fase e que o trabalho realizado foi enriquecido com funcionalidades adicionais, como a introdução de uma nova primitiva gráfica (o cilindro), o modo exploratório, a implementação dos Vertex Buffer Objects e a contagem dos frames por segundo. Uma das metas que não conseguimos atingir foi a construção da primitiva gráfica torus, mas isso permanece como um objetivo para a próxima fase. É importante destacar que, ao longo deste período, o grupo manteve um foco constante na eficiência e no desempenho das duas aplicações desenvolvidas. Portanto, fazemos uma avaliação positiva e estamos bastante satisfeitos com o trabalho realizado.