Computação Gráfica - TP1

Primitivas gráficas

02.03.2025

André Carvalho a100818,

Flávio Sousa a100715,

Vicente de Carvalho Castro a91677,

Índice

1. Introdução

A computação gráfica desempenha um papel fundamental na representação e manipulação de imagens e modelos tridimensionais. Neste trabalho prático, exploramos a geração e visualização de primitivas gráficas através da implementação de um sistema que permite criar e renderizar diversas formas geométricas, como planos, caixas, esferas e cones. Para isso, desenvolvemos um gerador de modelos que calcula os vértices de cada forma, organizando-os de maneira eficiente para posterior exibição. Além disso, implementamos uma engine baseada na biblioteca OpenGL, responsável pelo processamento e renderização das primitivas gráficas. Este documento detalha as abordagens utilizadas na geração dos modelos, o funcionamento da engine e os resultados obtidos, bem como sugestões para trabalhos futuros.

2. Generator

A sua função é gerar ficheiros onde está contida a informação relativa a cada modelo, especificamente os vértices nesta primeira fase.

2.1. Plano

O plano é um objeto bidimensional e portanto apenas está contido em dois eixos. Neste caso o nosso plano é um quadrado centrado na origem do referencial e pertence aos eixos X e Z.

A função que gera o plano recebe quatro argumentos, sendo estes o comprimento do quadrado, o número de divisões, a altura (referente à sua posição no eixo Y) e um inteiro denominado *bottom* que indica se a parte inferior do plano deve ser desenhada.

2.1.1. Cálculo dos pontos

Uma vez que o plano está centrado na origem é necessário dividir o comprimento do lado por dois dado que cada um dos vértices principais estará num quadrante (denominado *dimension2*).

Para além disso é necessário calcular o tamanho do lado de cada uma das divisões, uma vez que essas divisões são pequenos quadrados, e para isso divide-se o comprimento do lado pelo número de divisões (denominado *div_side*).

Após esses cálculos são iniciados dois *for loops*, um que itera sobre linhas (eixo Z) e um que itera sobre colunas (eixo X), onde os casos de paragem são o número de divisões.

A cada iteração são calculadas as coordenadas dos quatro pontos de cada divisão, começando por x1 e z1 que representam o canto superior esquerdo do quadrado. As coordenadas deste ponto são ambas negativas então estas são calculadas pelas seguintes fórmulas:

*x1 = -dimension2 + linha * div_side*
*z1 = -dimension2 + coluna * div_side*

Em seguida, calcula-se as coordenadas x2 e z2 onde a coordenada X se mantém, ou seja, *x2 = x1* e z2 é obtido somando o tamanho do lado de uma divisão à coordenada z1, por isso *z2 = z1 + div_side*.

No cálculo do terceiro ponto o Z mantém o valor da do primeiro ponto, portanto *z3 = z1* e x3 é calculado da mesma forma *x3 = x1 + div_side*

Por fim no cálculo do quarto ponto, o ponto calculado é diagonal ao ponto (x1, z1) e por isso obtém-se da seguinte forma *x4 = x1 + div_side, z4 = z1 + div_side*.

Posteriormente esses quadrados serão divididos em dois triângulos.

2.2. Caixa

A função *generateBox* recebe dois argumentos, o tamanho do lado e o número de divisões. Primeiramente calcula-se o tamanho do *step*, ou seja, o tamanho de cada subdivisão do cubo, dividindo o tamanho pelo número de divisões.

O cálculo dos pontos está divido em três fases. Em cada fase são calculados os pontos pertencentes a duas faces do cubo, sendo que as faces estão agrupadas da seguinte forma: superior e inferior na primeira fase, frontal e traseira na segunda fase e as laterais na terceira fase.

Como o cubo é uma forma tridimensional é necessário calcular as coordenadas nos três eixos, X, Z e Y.

2.2.1. Fórmulas utilizadas

Foram utilizadas três fórmulas para calcular os pontos, sendo elas:

Fórmula 1: *ponto = -size / 2 + ciclo * step;*
Fórmula 2: *ponto = -size / 2 + (ciclo + 1) * step;*
Fórmula 3: *ponto = size / 2*

Onde ciclo é a correspondente às letras i ou j dependendo do eixo que estiver a ser iterado, sendo que i é sempre a letra correspondente ao ciclo externo.

2.2.2. Cálculo dos pontos - 1ª fase

Primeiramente são iniciados dois *for loops* para iterar sobre linhas dos eixos X e Z, respetivamente, e calcular as coordenadas dos quatro pontos.

Como na primeira fase estamos a calcular os pontos correspondentes a duas faces, sendo elas a face superior e inferior, a coordenada Y não se altera, portanto, em qualquer ponto calculado nesta fase será aplicada a terceira fórmula.

Em x1 e z1 é utilizada a primeira fórmula

Em x2 é utilizada a segunda fórmula enquanto z2 = z1

Em z3 é utilizada a segunda fórmula, enquanto x3 = x1

E por fim, em x4 e z4 é utilizada, novamente, a segunda fórmula

Assim são obtidos os valores da face superior. Para obter os da face inferior apenas é necessário transformar a coordenada y no seu simétrico -y.

2.2.3. Cálculo dos pontos - 2ª fase

Esta fase é muito semelhante á anterior, contudo o z é a coordenada que se mantém constante ao invés do y e os ciclos operam sobre x e y, respetivamente.

Por isso, z é calculado através da terceira fórmula.

x1 e y1 são calculados utilizando a primeira fórmula.

x2 utiliza a segunda fórmula enquanto y2 = y1.

y3 utiliza a segunda fórmula, enquanto x3 = x1.

E por fim, x4 e z4 utilizam a segunda fórmula.

Novamente para obter a face oposta inverte-se a coordenada z, mantendo todas as outras.

2.2.4. Cálculo dos pontos - 3ª fase

Por fim, a coordenada x é constante e por isso utiliza terceira fórmula, enquanto z e y são iterados, sendo que desta vez, o ciclo externo itera z.

O esquema de coordenadas segue o mesmo principio:

z1, y1, y2 e z3 utilizam a primeira fórmula.

z4, y4, z2 e y3 utilizam a segunda fórmula.

Mais uma vez, as coordenadas da face oposta são calculadas através da inversão de x.

2.3. Esfera

A função principal recebe três argumentos, o raio, o número de fatias (divisões verticais) e o número de stacks (divisões horizontais).

O primeiro passo foi calcular o ângulo de cada fatia e de cada stack.

O ângulo das fatias é denominado alfa e é calculado a multiplicar o π por 2 e dividir pelo número de fatias, isto porque 2pi é o comprimento da circunferência, em radianos.

O ângulo das stacks é denominado beta e é calculado dividindo π pelo número de stacks, dado que π é a diferença do ponto mais alto da esfera, até ao ponto mais baixo

2.3.1. Cálculos dos pontos

Iniciam-se os *for loops* onde stacks é o ciclo externo e fatias o ciclo interno.

Dentro dos ciclos são calculados quatro ângulos, theta1, theta2, phi1 e phi2.

Theta1 corresponde ao ângulo de inicio de cada fatia e é calculado através da multiplicação de alfa pelo índice atual do ciclo.

Já theta2 é o oposto, representa o ângulo correspondente ao final de cada fatia. Calcula-se multiplicando alfa pelo índice atual do ciclo incrementado por um.

Phi1 e Phi2 possuem uma interação semelhante, onde phi1 é calculado subtraindo metade de π pela multiplicação do índice atual do ciclo pelo beta.

Em Phi2 apenas se soma um ao índice, antes de efetuar a multiplicação por beta.

Em cada iteração são calculados quatro pontos da seguinte forma:

*p1 = raio * sin(theta1) * cos(phi1), raio * sin(phi1), raio * cos(theta1) * cos(phi1)*
*p2 = raio * sin(theta1) * cos(phi2), raio * sin(phi2), raio * cos(theta1) * cos(phi2)*
*p3 = raio * sin(theta2) * cos(phi2), raio * sin(phi2), raio * cos(theta2) * cos(phi2)*
*p4 = raio * sin(theta2) * cos(phi1), raio * sin(phi1), raio * cos(theta2) * cos(phi1)*

Estas fórmulas são baseadas na conversão de coordenadas esféricas para cartesianas.

Por fim, estes quatro pontos irão formar triângulos que serão utilizados para desenhar a superfície da esfera.

2.4. Cone

A função que gera o cone recebe quatro argumentos, o raio da base, a altura, o número de fatias e o número de stacks.

Tal como na esfera, o cone possui o ângulo alfa, que é o dobro de π a dividir pelo número de fatias.

O seu diferencial está no facto de este ir diminuindo à medida que a altura aumenta, e por isso ele precisa de duas variáveis novas, o deltaHeight e o delta-Radius.

Estas variáveis representam a variação (delta) da altura e do raio á medida que vamos subindo no cone.

O deltaRadius calcula-se dividindo o raio da base pelo número de stacks e o deltaHeight dividindo a altura pelas stacks.

Tal como o cubo este cálculo é dividido em três fases, geração da base, geração das laterais e por fim ocorre a geração da ponta do cone.

2.4.1. Cálculo dos pontos - 1ª fase

A base está contida no plano XZ, sendo assim, a coordenada y será estática (será sempre zero).

Aqui, apenas é necessário um ciclo *for* e dois pontos, uma vez que iremos utilizar o centro da base (que é a origem do referencial) como o terceiro ponto necessário para a criação dos triângulos.

Posto isto, temos de calcular as coordenadas dos pontos (x1,z1,0) e (x2,z2,0).

Foram utilizadas as equações paramétricas onde $x = r * sin(\theta)$ e $z = r * cos(\theta)$.

Aplicando isto ao nosso caso, temos que:

$x1 = bottomRadius * sin(alpha * i)$
$z1 = bottomRadius * cos(alpha * i)$
$x2 = bottomRadius * sin(alpha * (i + 1)$
$z2 = bottomRadius * cos(alpha * (i + 1)$

2.4.2. Cálculo dos pontos - 2ª fase

Durante a formação da lateral do cone são utilizados quadrados, divididos em dois triângulos, num deles dois pontos estão numa camada e o terceiro está na camada seguinte, e no outro o contrário, desta forma conseguimos fazer com que o cone obtenha a sua forma.

Assim, o primeiro passo é criar um ciclo que itera sobre as stacks (que são as divisões horizontais do cone) e calcular o raio da camada onde está o índice do ciclo (r1) e o raio da camada seguinte (r2), bem como a altura y da camada atual (y1) e a altura y da camada seguinte (y2) utilizando as seguintes fórmulas:

$r1 = bottomRadius - j * deltaRadius;$
$y1 = j * deltaHeight;$

Para r2 e y2 apenas se incrementa uma unidade a j.

Em seguida cria-se o ciclo que itera sobre as slices (fatias verticais) que vai calcular as coordenadas de x e z necessárias para criar os pontos que serão usados para gerar os triângulos.

No total são gerados quatro pontos e são aplicadas as mesmas fórmulas utilizadas para a base,

*x1 = r1 * sin(alpha * i)*
*z1 = r1 * cos(alpha * i)*
*x2 = r1 * sin(alpha * (i + 1)*
*z2 = r1 * cos(alpha * (i + 1)*

Para os ponto (x3,z3) e (x4,z4) as fórmulas são as mesmas, apenas se substitui r1 por r2

2.4.3. Cálculo dos pontos - 3ª fase

Primeiramente define-se as coordenadas da ponta do cone, que são (0,altura,0) uma vez que o cone está centrado na origem.

Em seguida, utiliza-se novamente um ciclo que itera sobre as slices para calcular os pontos (x1, z1) e (x2, z2).

Estes pontos são calculados utilizando as seguintes fórmulas:

*x1 = (bottomRadius - stacks * deltaRadius) * sin(alpha * i)*
*z1 = (bottomRadius - stacks * deltaRadius) * cos(alpha * i*
*x2 = (bottomRadius - stacks * deltaRadius) * sin(alpha * (i + 1));*
*z2 = (bottomRadius - stacks * deltaRadius) * cos(alpha * (i + 1)*

Relembrando as equações paramétricas $x = r * sin(\theta)$ e $z = r * cos(\theta)$ podemos observar que estas expressões possuem essa mesma forma, onde o nosso r é igual a (bottomRadius - stacks * deltaRadius). Isto significa que o raio da camada é igual ao tamanho original do raio subtraído pela quantidade reduzida até chegar aquela camada.

Por fim os pontos são criados com a forma (x, height - deltaHeight,z).

3. Engine

A engine é o componente central do programa, responsável por processar e renderizar as primitivas gráficas usando a biblioteca OpenGL. Ela coordena a leitura e interpretação dos arquivos de configuração XML, que descrevem o cenário 3D, incluindo a posição da câmera, iluminação e as características dos objetos. A engine também é responsável por interagir com o utilizador por meio do GLUT, realizando a criação e exibição dinâmica das figuras 3D no ambiente virtual. Além disso, ela lida com o cálculo de projeções, as transformações de objetos e o controlo da câmera, proporcionando uma visualização interativa e eficiente da cena durante a execução. Em suma, a engine funciona como a espinha dorsal do sistema gráfico, integrando todas as funcionalidades necessárias para a construção e manipulação do cenário 3D.

3.1. Parsing de XML

O XML é usado para configurar o cenário a ser implementado com GLUT, abrangendo desde as configurações da câmera até a posição e dimensões das figuras na cena. A estrutura do XML segue uma hierarquia com quatro principais elementos: “window”, “camera”, “lights” e “group”.

Para aceder a estes nós foi utilizada a biblioteca **TinyXML** que nos permite iterar sobre eles com facilidade e extrair os dados necessários.

3.2. Controlo da câmera

O utilizador pode executar seis inputs diferentes no teclado de forma a alterar a câmera, quatro delas aplicam rotação e duas delas alteram a distância ao objeto.

A tecla *GLUT_KEY_UP* aumenta o ângulo beta, ou seja aplica uma rotação vertical para cima.
A tecla *GLUT_KEY_DOWN* diminui o ângulo beta, ou seja aplica uma rotação vertical para baixo.
A tecla *GLUT_KEY_LEFT* aumenta o ângulo alpha, ou seja aplica uma rotação horizontal para a esquerda.
A tecla *GLUT_KEY_RIGHT* diminui o ângulo alpha, ou seja aplica uma rotação horizontal para a direita.

A tecla - aumenta a distância da câmera ao objeto (zoom out).
A tecla + diminui a distância da câmera ao objeto (zoom in).

4. Resultados obtidos

Foram utilizados os ficheiros de teste disponibilizados na blackboard em que obtivemos os seguintes resultados:

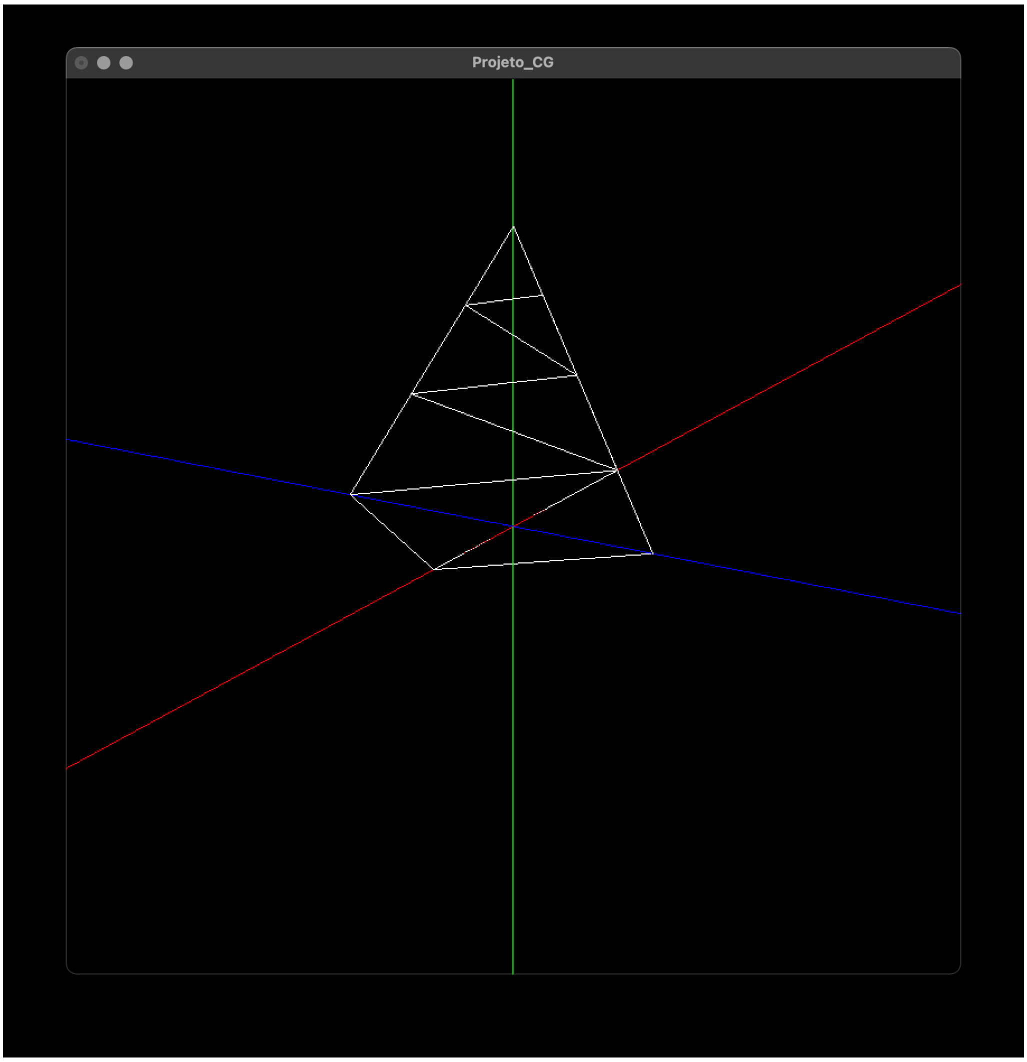

Figura 1: Cone - teste 1.1

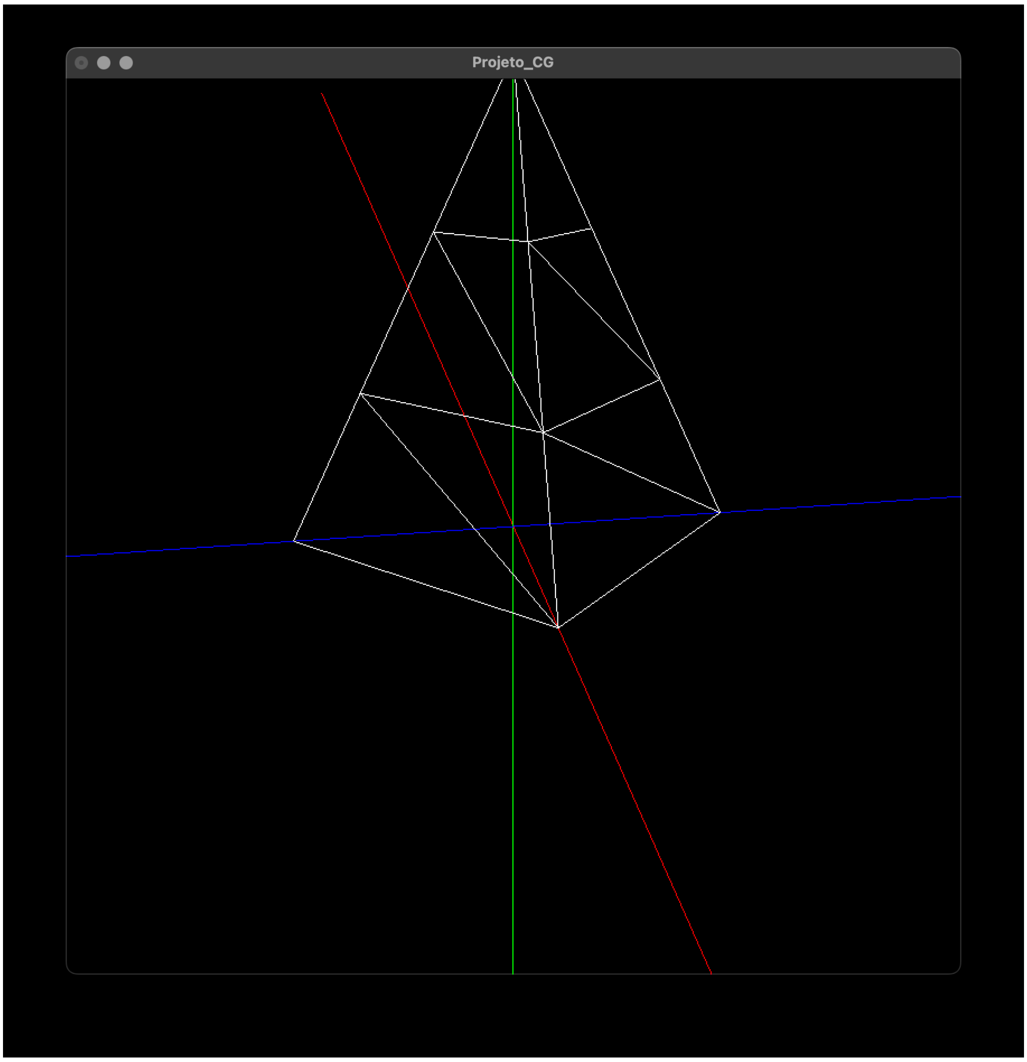

Figura 2: Cone zoom in - teste 1.2

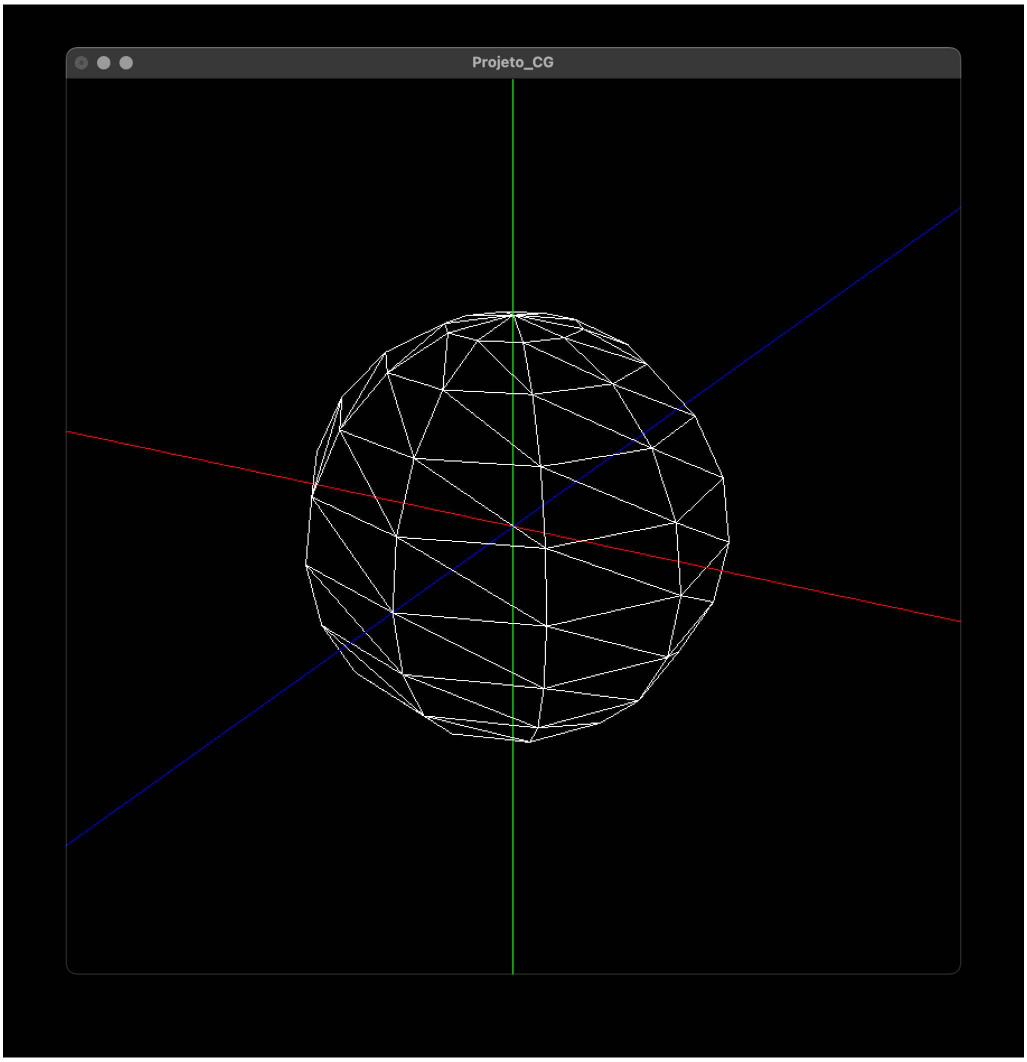

Figura 3: Esfera - teste 1.3

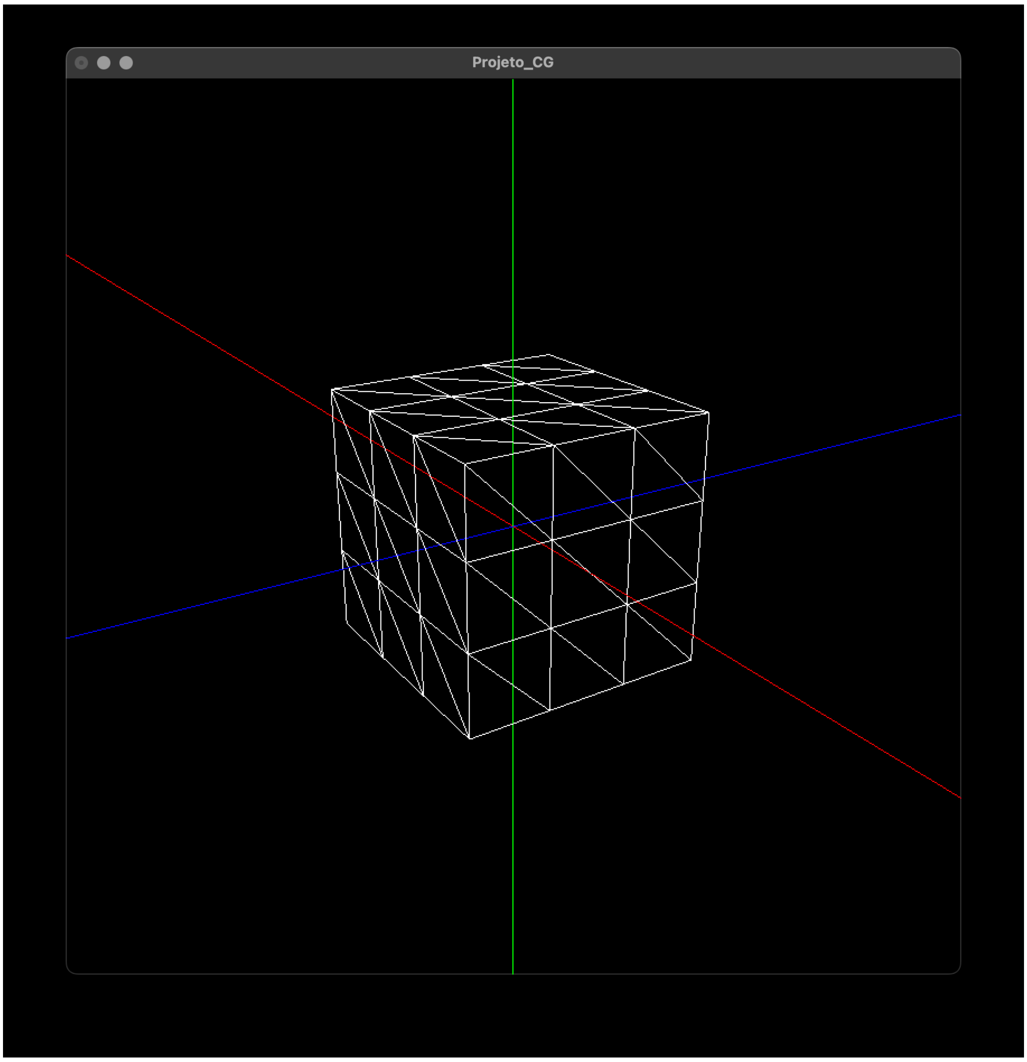

Figura 4: Caixa - teste 1.4

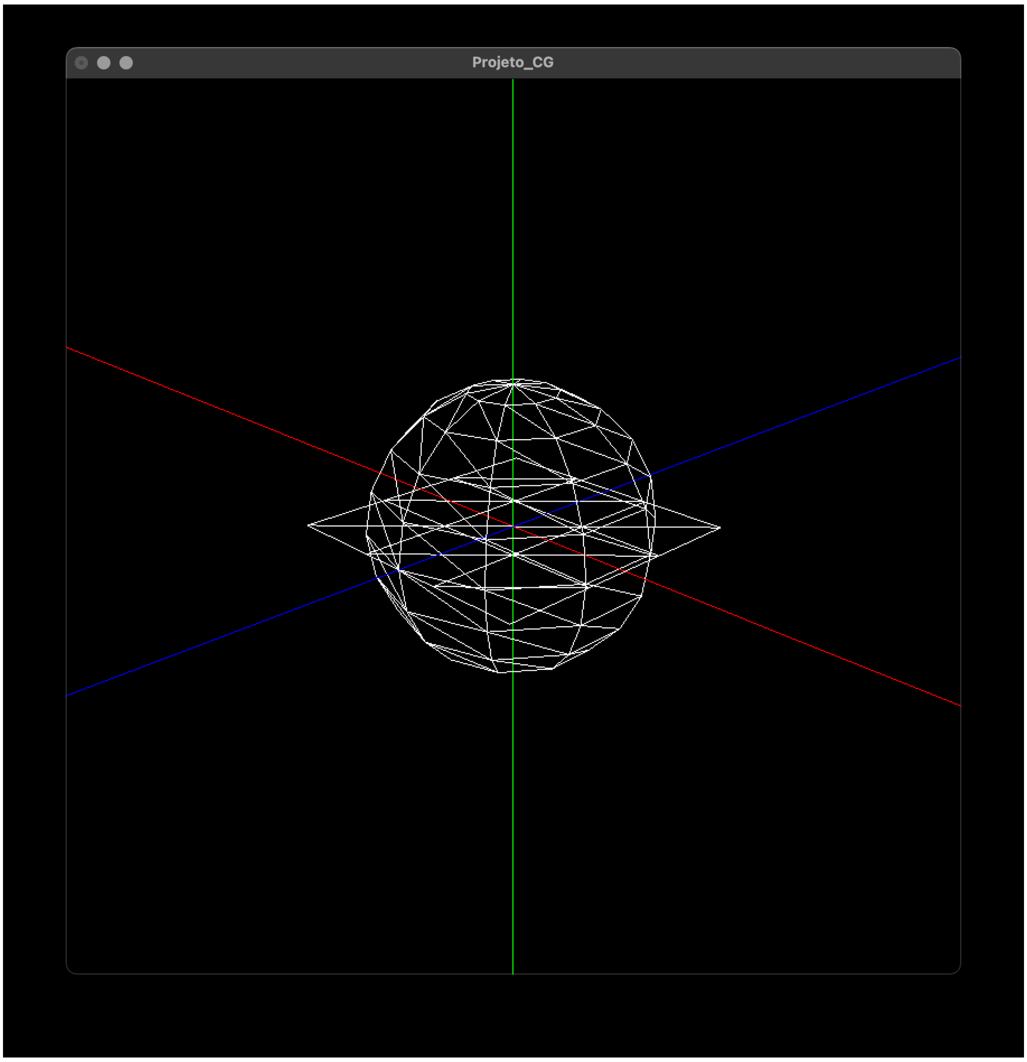

Figura 5: Mistura de plano e esfera - teste 1.5

5. Conclusões e trabalho futuro

A implementação deste projeto permitiu compreender os princípios fundamentais da geração e renderização de primitivas gráficas em computação gráfica. Através do desenvolvimento do gerador de modelos e da engine de visualização, conseguimos criar e manipular objetos tridimensionais de forma eficiente. Os resultados obtidos demonstram a eficácia das metodologias aplicadas, embora existam oportunidades para melhorias, como a otimização dos cálculos dos vértices. Para trabalho futuro, mais concretamente na segunda fase deste projeto, serão exploradas as transformações geométricas.