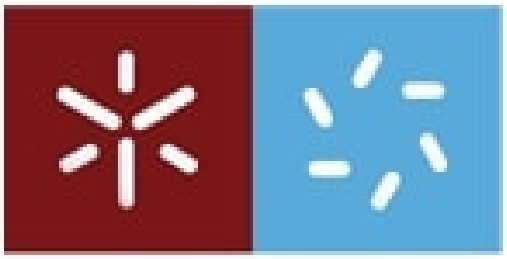

Universidade do Minho
Escola de Ciências

Computação Gráfica

Jorge Daniel Pereira Silva (a80931)
José Pedro Silva Ferreira (a96798)
Rui Alexandre Dias Neto (a80433)

8 de março de 2024

Conteúdo

1 Introdução

O objetivo proposto nesta primeira fase do trabalho prático é o desenvolvimento de um programa de criação e desenho de algumas primitivas gráficas como um plano, uma box, uma esfera e um cone. Para isso, foram necessárias duas aplicações o **Generator** e o **Engine**.

2 Generator

Nesta aplicação o objetivo é gerar pontos e organizar esses mesmos pontos de formar a criar triângulos para que depois seja possível visualizar as primitivas gráficas,através de um ficheiro *.3d*.Começamos por criar uma **main** onde verificamos se os argumentos escritos no terminal corresponde ao que é pretendido.Após isso, decidimos que iríamos necessitar de duas classes,a **Point** e a **Index**, de modo a facilitar a geração e organização dos pontos.Finalizando essas classes passamos à geração e organização dos pontos por primitiva gráfica.

2.1 Classes

2.1.1 Point

A classe *Point* foi concebida para representar pontos tridimensionais, onde cada ponto é definido pelas coordenadas *x*, *y* e *z*.Decidimos também implementar esta classe de forma como se fez na cadeira de Programação Orientada a Objetos,onde possuímos métodos *get* e *set* e também um método que converte pontos em string para escrever no ficheiro.

2.1.2 Index

A classe *Index* foi desenvolvida para fornecer uma estrutura eficiente de armazenamento e manipulação de índices associados aos pontos.Esta classe foi desenvolvida como a classe *Point*,ou seja com métodos *get*, *set* e um método que converte índices em string.A grande diferença é que os índices são guardados num tuplo.

2.2 Figures

2.2.1 Plano

Para a criação de um plano é necessário dois parâmetros. O primeiro parâmetro é o **length** que corresponde ao tamanho de cada lado e **divisions** que corresponde a quantas divisões vai ter o plano. Começamos por criar uma função com o nome *getplanepoints* onde calculamos as coordenadas necessárias para gerar pontos de forma a corresponder com os parâmetros passados. Para isso começamos por ver as coordenadas do ponto A (ver figura 1), que o *x=-length/2* e o *z=-length/2*. Para calcular o ponto B o z continua exatamente igual mas o *x=-length/2 + 1*(length/divisions)* até chegar ao fim dessa linha.Todos os pontos desta função são guardados num *vector<Point> points*.Após chegar ao ponto D avança de linha e o raciocínio é o mesmo só que *x* e *z* com valores diferentes. O valor do *y* é sempre zero. Para organizar os pontos em triângulos criamos a função *getplaneindex* onde começamos para calcular como vamos buscar os quatro vertices de cada triângulo(Pontos A,B,E,F da figura 1) e usamos a regra da mão direita para coloca-los por ordem.Todos os pontos desta função são guardados num *vector<Index> Index*.Após isto temos uma função com o nome *plane* onde invocamos essas duas funções anteriores e uma chamada *create3d* que vai criar o ficheiro com os pontos que geramos e os índices dos vértices organizados.

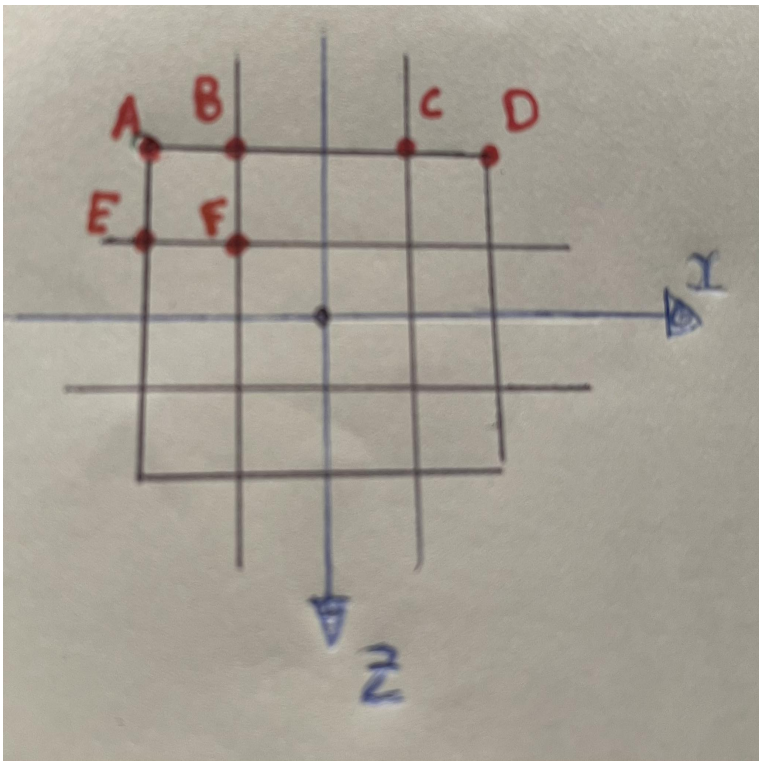

Figura 1: Exemplo de um plano

2.2.2 Box

Para gerar e organizar os pontos da Box, foi necessário desenvolver funções análogas às utilizadas para a criação do plano. As modificações na função de geração de pontos levaram em consideração não apenas a determinação da face em questão, mas também a orientação do referencial, o qual foi centrado no meio da caixa.No que diz respeito à organização dos pontos para formar triângulos, seguimos a regra da mão direita, garantindo que a orientação dos triângulos fosse consistente e respeitasse as convenções de orientação na representação tridimensional.

2.2.3 Cone

Para gerar os pontos do cone começamos por calcular que altura vai ter cada stack e qual o ângulo de cada slice.Após isso adicionamos o ponto superior do cone e vamos começar a calcular pontos de cima para baixo.Em seguida calculamos o seu raio e a sua altura que vão ser necessárias para atribuir pontos e percorremos cada slice já sabendo o ângulo e a altura e através das coordenadas polares conseguimos calcular o 1,2,3,4 da figura 3.

Para organizar os índices para formar triângulos foi feito por 3 diferentes fases a superior,a inferior e a do meio. Na superior foi só unir ao ponto de cima e meter os triângulos visíveis segundo a regra da mão direita(como demonstramos figura 2 e figura 3).

```
//top of cone
for (int i = 1; i <= slices; i++) {
  int index3 = (i % slices) + 1;

index.push_back(Index(0, i, index3));
}
```

Figura 2: Organizar parte superior

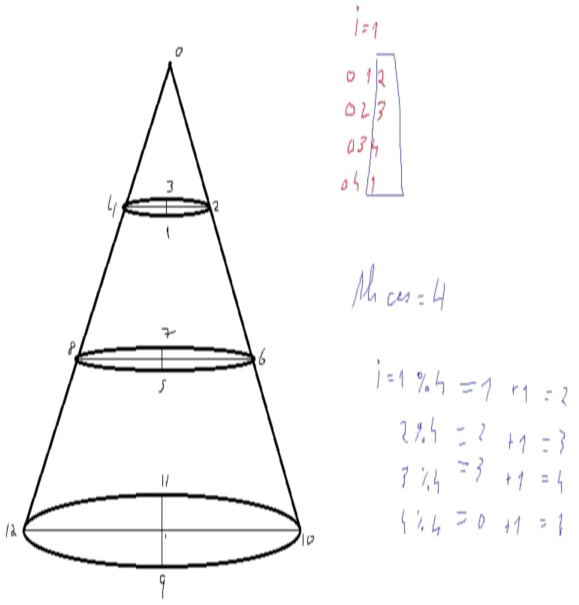

Figura 3: Forma Pensamento

Na parte inferior foi unir o ponto central a outros dois,como podemos verificar pela figura 4 e figura 5.

```
//  bottom
 for (int i = 1; i <= slices; i++) {
   int index2 = nPoints - i;
   int index3 = nPoints - (i % slices) - 1;

   index.push_back(Index(nPoints, index2, index3));
 }
```

Figura 4: Organizar parte inferior

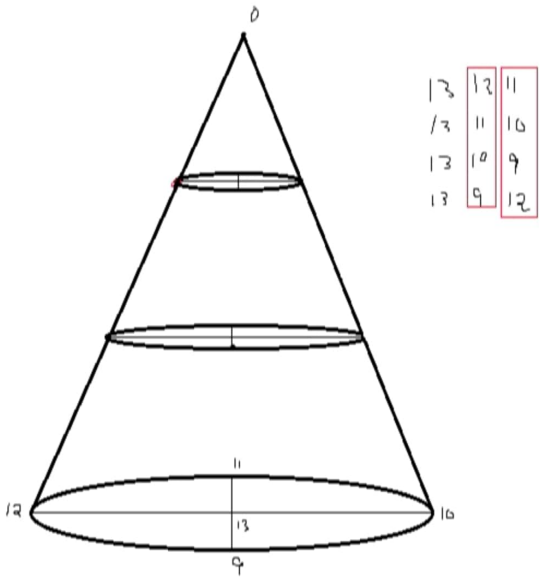

Figura 5: Forma Pensamento

Na parte do meio a organização dos pontos foi feito de acordo com a figura 8, onde a forma de pensamento do código foi feito baseado na figura 7.

```
for (int i = 1; i < nPoints - slices; i += slices) {
  for (int j = 0; j < slices; j++) {
    int t1_i1 = i + j;
    int t1_i2 = t1_i1 + slices;
    int t1_i3 = ((j + 1) % slices) + slices + i;
    index.push_back(Index(t1_i1, t1_i2, t1_i3));

    int t2_i3 = ((j + 1) % slices) + i;
    index.push_back(Index(t1_i1, t1_i3, t2_i3));
  }
}
```

Figura 6: Organizar parte meio

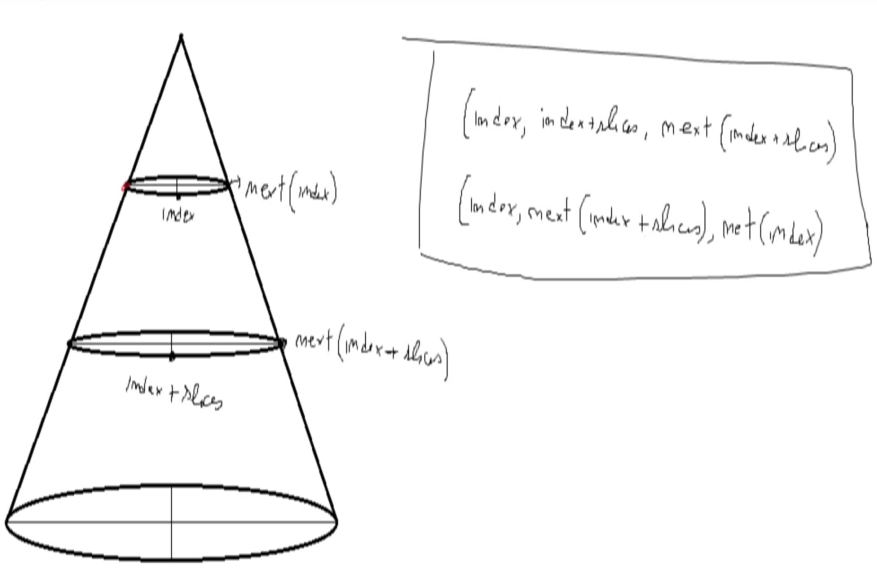

Figura 7: Forma Pensamento

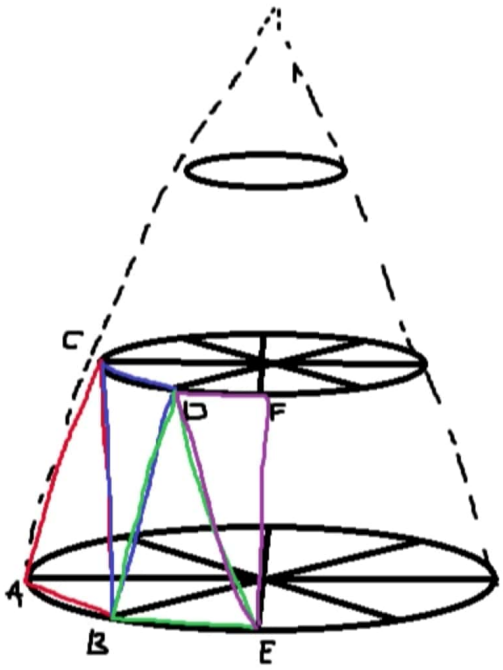

Figura 8: Forma Pensamento

2.2.4 Esfera

Para gerar os pontos da esfera começamos por calcular o ângulo de cada slice e o ângulo de cada stack. Depois utilizamos a figura 9 para calcular as coordenadas dos pontos percorrendo cada stack e calculando-os através de cada slice.

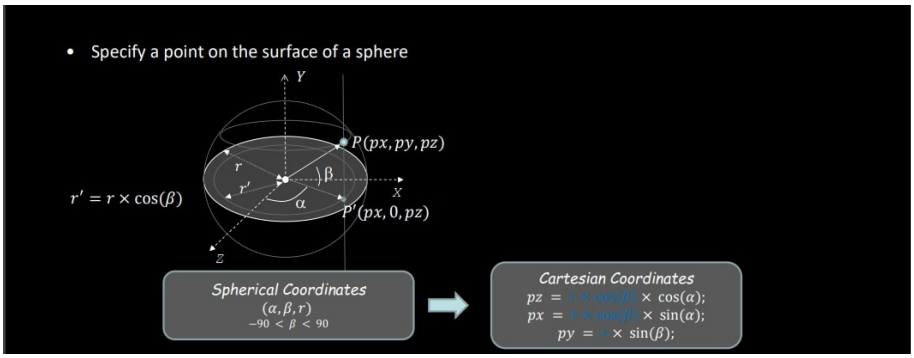

Figura 9: Gerar pontos

```
void get_sphere_points(float radius, int stacks, int slices,vector<Point> &points) {
  //calculates the angle between the top and bottom points of the sphere
  float stack_angle = ((M_PI) / (float)stacks) - (M_PI/2);
  float slice_angle = (2 * M_PI) / (float)slices;

  points.push_back(Point(0, -radius, 0));

  for (int i = 1; i < stacks; i++) {
    float beta = (i * stack_angle);
    for (int j = 0; j < slices; j++) {
      float alpha = j * slice_angle;

      float px = radius * cos(beta) * sin(alpha);
      float py =  radius * sin(beta);
      float pz = radius * cos(beta) * cos(alpha);

      points.push_back(Point(px, py, pz));
    }
  }

  points.push_back(Point(0, radius, 0));
}
```

Figura 10: Código Gerar pontos

Para organizar os pontos foi usada exatamente a mesma lógica do cone.

3 Engine

Esta aplicação está responsável pela leitura dos ficheiros XML.Para leitura e escrita de ficheiros XML recorreu-se à ferramenta **Tinyxml2** e criamos classes para guardar a informação que contem cada parte do ficheiro XML.

3.1 Classes

3.1.1 Window

Nesta classe o objetivo é guardar os valores que os xml contêm sobre a window.Ao construir esta classe fizemos como em Programação Orientada a Objetos. Criamos o método onde só percorre os elementos da window e guardamos em duas variáveis *Width* e *Height*.Atualizando na função main esses valores com os métodos *get* sobre o objeto.

3.1.2 Camera

Nesta classe temos 4 vectores para guardar os valores necessários para a Camera. Temos como em todas as outras classes os métodos *get* e *set*.Criamos o método para dar parse à câmara no XML e verificamos qual é o primeiro filho e dependendo de qual seja guarda os valores no seu vetor apropriado dando *set* a esses valores.Com a ajuda do *tinnyxml2* usamos funções como FloatAttribute.Na *main* onde temos a função *renderScene* atualizamos lá os valores da câmara do XML lido, assim como no função *changeSize* atualiza lá os valores da perspetiva.

3.1.3 Model

Aqui nesta classe ,tal como nas anteriores, criamos os métodos habituais juntamente com o método para dar parse ao model. Nesta classe tivemos que adicionar um novo método chamado *readPointsFromFile* onde vai ler os pontos e a ordem com que devem ser lidos do ficheiro lido no xml e chamamos esta função no parse para conseguir desenhar como pretendido.

3.1.4 Group

Nesta Classe voltamos a criar os métodos básicos e o parse onde no parse verificamos se o nome do filho é model e então se for percorre os seus filhos.

3.2 Main

Neste ficheiro temos um pouco do que foi feito nas aulas práticas onde temos uma função chamada *Draw-Models*(figura 11) onde percorremos os models dentro da group,onde pode ter mais que um model como acontece no teste 5 desta fase, e onde vai percorrer os índices de como têm que ser desenhados os triângulos e após ter os índices vai buscar o ponto daquele índice e desenha.Criamos também a função *ProcessNormalKeys* onde quando abre a janela podemos fechá-la clicando no *ESC*. E possui ainda a função *readXML* que chama cada objeto de cada classe para conseguir ler o ficheiro XML.

```
for (Model model : group.get_models()) {
    for (vector<int> triangleIndices : model.get_index()) {
        int index1 = triangleIndices[0];
        int index2 = triangleIndices[1];
        int index3 = triangleIndices[2];


        vector<float> point1 = model.get_points()[index1];
        vector<float> point2 = model.get_points()[index2];
        vector<float> point3 = model.get_points()[index3];

        glVertex3f(point1[0], point1[1], point1[2]);
        glVertex3f(point2[0], point2[1], point2[2]);
        glVertex3f(point3[0], point3[1], point3[2]);
    }
}
```

Figura 11: Função DrawModels

3.3 Resultados

Demonstramos aqui os ficheiros XML lidos e o seu resultado:

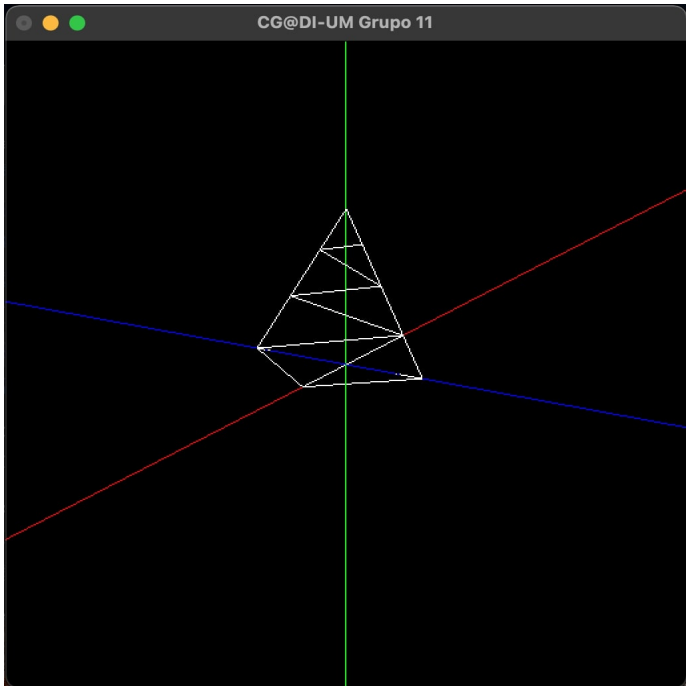

Figura 12: Teste1

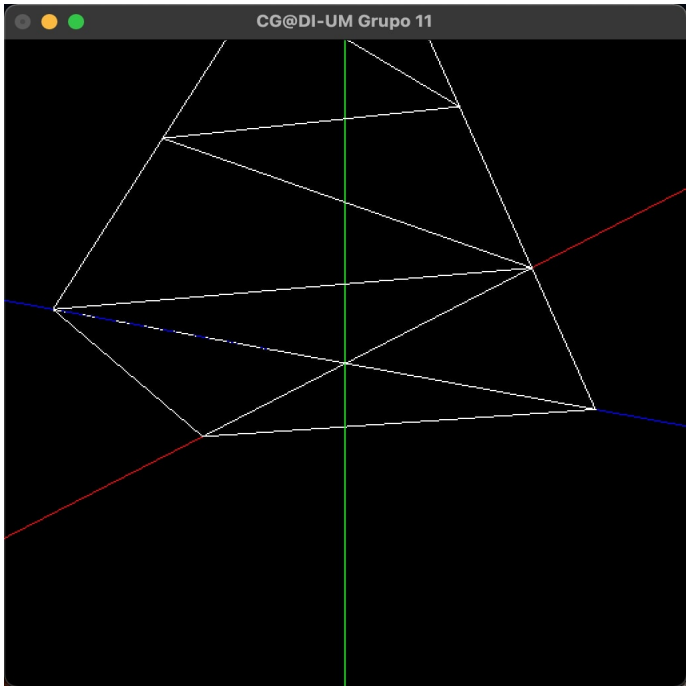

Figura 13: Teste2

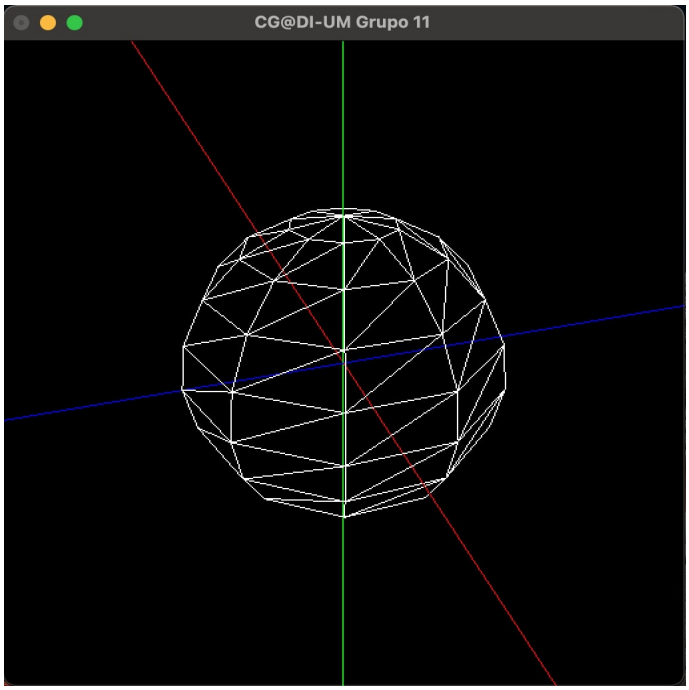

Figura 14: Teste3

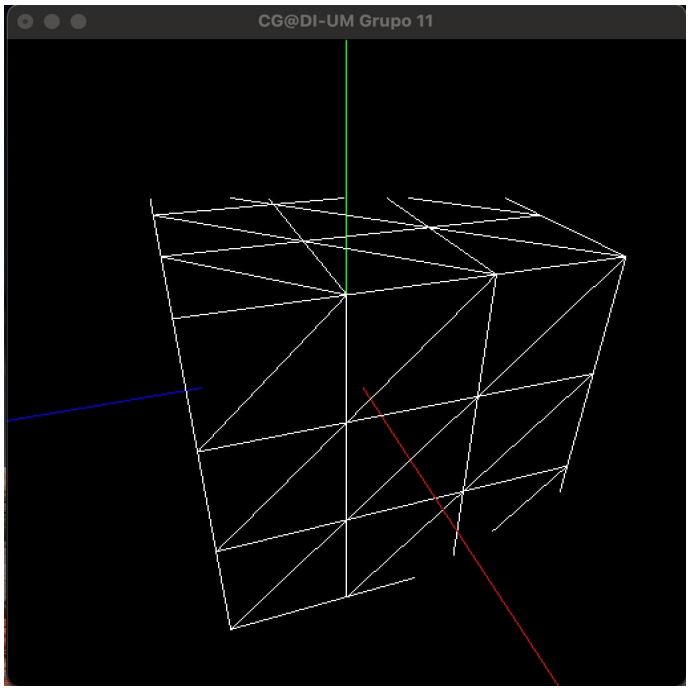

Figura 15: Teste4

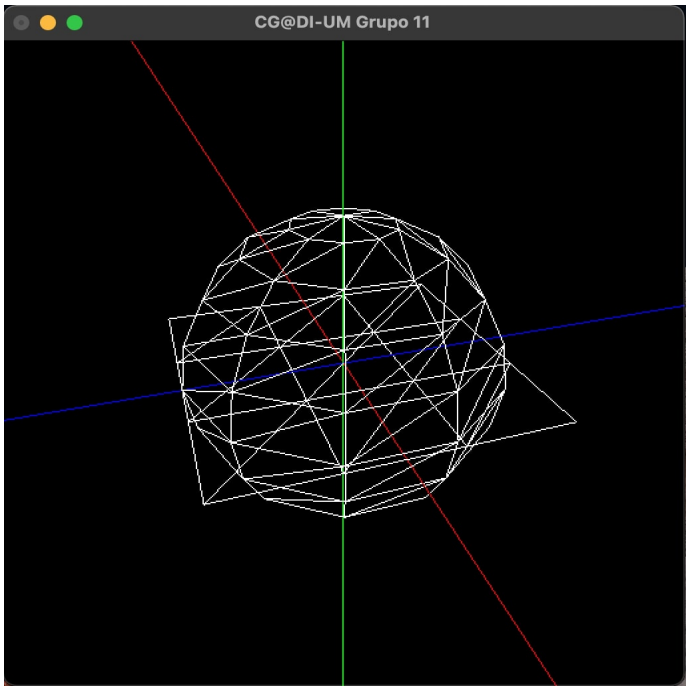

Figura 16: Teste5

4 Conclusão

Nesta fase inicial do projeto, conseguimos adquirir experiência com ferramentas essenciais para a disciplina. Embora tenhamos enfrentado algumas dificuldades na criação do cone e da esfera,o processo de resolução desses obstáculos permitiu-nos consolidar ainda mais o nosso conhecimento.No final,alcançamos os resultados desejados sucesso. Em suma, consideramos que cumprimos os objetivos estabelecidos para esta etapa do projeto.