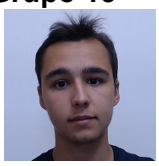
Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de
Computação Gráfica

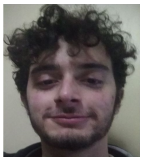
Ano Letivo de 2022/2023

Primitivas Gráficas
Fase 1

Grupo 15


a84696,Renato Gomes


a89486,Tomás Dias


a97223,João Castro

URL do Repositório
https://github.com/joaocasr/CG2022-23

March 10, 2023

Índice

Lista de Figuras

1. Introdução

Este relatório apresenta a primeira fase do nosso projeto, que envolve duas aplicações. A primeira aplicação é um gerador de ficheiros com informações de modelos e a segunda aplicação é o motor que lê um arquivo de configuração escrito em XML e exibe os modelos.

Para criar os arquivos de modelo, o ***generator*** receberá parâmetros como o tipo de primitiva gráfica, outros parâmetros necessários para a criação do modelo e o arquivo de destino onde as informações serão armazenados.

A segunda aplicação, o ***engine***, receberá um arquivo de configuração escrito em XML que conterá as configurações da câmera e a indicação de quais arquivos de modelos gerados pelo gerador devem ser carregados.

A metodologia aplicada ao longo desta fase foi reuniões frequentes com o grupo todo presente, dividindo tarefas de semana para semana.

2. Arquitetura do Sistema

O presente sistema encontra-se organizado em dois módulos, a engine e o generator. O módulo da engine possui um ficheiro engine.cpp e engine.h onde se encontram declarados os métodos e variáveis usadas no .cpp correspondente. Para além destes, estão presentes os ficheiros tinyxml2.cpp e tinyxml2.h, necessários para realizar a leitura e parsing dos ficheiros de configuração em formato XML. Por outro lado, foram adicionados dois folders, um com capturas de ecrã dos resultados dos testes realizados e outro com as capturas de ecrã das primitivas expectáveis.

O módulo generator possui o ficheiro generator.cpp e o generator.h. Consequentemente, após a geração das primitivas especificadas pelo utilizador, será criada uma pasta adicional, "models", com todos os ficheiros 3d, onde irão estar descritos todos os pontos do modelo correspondente.

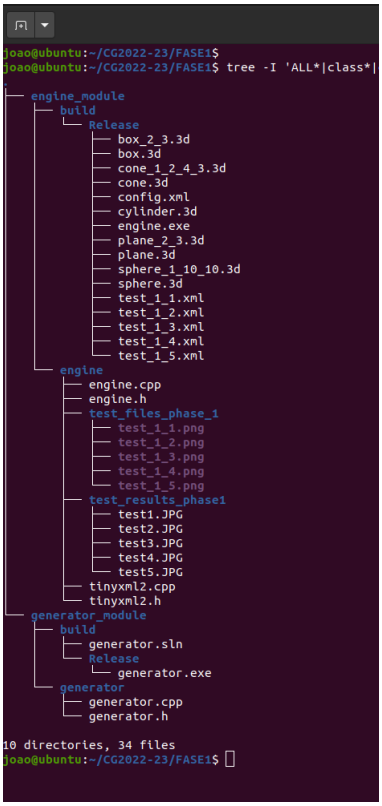


Figura 2.1: Arquitetura do sistema

3. Implementação

3.1 Generator

3.1.1 Primitivas

Plano

Para a construção do cubo procedeu-se inicialmente ao quociente entre o comprimento do lado do cubo (units) e o número de grades/grelhas (grid), que irão formar $grid^2$ quadrados. O valor desta divisão irá representar o *step* quando for necessário construir cada um destes triângulos. Os triângulos representados serão triângulos retângulos com catetos de igual comprimento, pelo que será possível obtermos um quadrado com 2 triângulos deste tipo. Logo, o valor do *step* será o valor de um desses lados. A partir desta informação e tendo noção que o Y é sempre igual a 0 para todos os pontos conseguimos então gerar todos os pontos necessários para criar os vértices do plano,

Para tal vamos utilizar dois ciclos for aninhados em que vamos calcular os pontos da seguinte forma:

$px1 = -finalx + (j * stepx);$
$px2 = -finalx + ((j+1) * stepx);$
$pz1 = -finalz + (i * stepz);$
$pz2 = -finalz + ((i+1) * stepz);$

sendo que:

$-finalz + i * stepz < finalz;$
$-finalx + j * stepx < finalx;$
para $i, j \in N$, $stepx = stepz = units/grid$

Cubo

Para a construção do cubo procedeu-se inicialmente ao quociente entre o comprimento do lado do cubo (units) e o número de grades/grelhas (grid), que irão formar $grid^2$ quadrados para cada uma das faces. O valor desta divisão irá representar o 'step' quando for necessário construír cada um destes triângulos. Os triângulos representados serão triângulos retângulos com catetos de igual comprimento, pelo que será possível obtermos um quadrado com 2 triângulos deste tipo. Logo, o valor do step será o valor de um desses lados. De seguida, procedeu-se à divisão por 2 do comprimento do lado do cubo (units), de forma a pudermos iterar dentro deste intervalo [-units/2,units/2], uma vez que o cubo tem de estar obrigatoriamente centrado na origem.

De seguida, procedeu-se à construção de planos paralelos, ou seja, em que apenas um dos componentes é sempre constante. Se considerarmos como exemplo a face inferior e a face superior do cubo (são paralelas entre si), todos os pontos de baixo terão um y<0 constante e os pontos de cima terão um y>0 (simétrico) constante. Logo, a partir dos valores do step e do intervalo mencionado em cima, vamos realizar ciclos for, de modo a irmos construindo cada um dos pontos dos triângulos de ambas as faces. Assim, um quadrado constituído por dois triângulos pertencentes ao plano perpendicular ao semi-eixo negativo Oy irão apresentar os seguintes pontos:
$p1(-finalz + ((i+1) * stepz), -finaly, -finalz + ((i+1) * stepz)$;
$p2(-finalx + (j * stepx), -finaly, -finalz + ((i+1) * stepz))$;
$p3(-finalx + (j * stepx), -finaly, -finalz + (i * stepz))$
$p4(-finalx + ((j+1) * stepx), -finaly, -finalz + (i * stepz))$

Sendo que:
$-finalz + i * stepz < finalz$ e $-finalx + j * stepx < finalx$, para $i, j \in N$,
$stepx = stepz = units/grid$ e $finalz = finaly = units/2$

Os triângulos do plano superior terão da mesma forma apenas a componente y simétrica. A construção das restantes faces do cubo seguem o mesmo raciocínio.

Esfera

Inicialmente, são definidos os valores de deltaAlpha e deltaBeta, que serão usados para gerar todas os pontos da esfera.

$deltaAlpha = 2 * \pi / slices$
$deltaBeta = \pi / stacks$

O algoritmo, então, começa a criar os triângulos que formarão a esfera, percorrendo todas as slices por cada uma das stacks. Procedeu-se à divisão de stacks por cada

semiesfera(desde -(stacks / 2) até (stacks / 2)), sendo que para cada uma é calculado o valor de beta e do próximo beta do nível da stack superior, enquanto que para cada slice, é calculado o valor de alpha e do próximo alfa em relação ao eixo Oz (o intervalo entre estes dois âmgulos corresponde ao ângulo da stack nessa slice). Isto é:

$beta = i * deltaBeta;$
$nextBeta = (i + 1) * deltaBeta$
$alpha = j * deltaAlpha$
$nextAlpha = (j + 1) * deltaAlpha$

Sendo que:

$-(stacks/2) < i < (stacks/2)$ e $0 < j < slices$, para $i, j \in N$

Para cada ponto da esfera, são calculadas as coordenadas x, y e z utilizando as fórmulas matemáticas apropriadas ($radius * cos(beta) * sin(alpha), radius * sin(beta), radius * cos(beta) * cos(alpha)$) com base no raio, no beta e no alpha. Em seguida, quatro pontos são criados a partir dos quais se irão formar dois triângulos.
Admitindo que, de Oz a OP' corresponde à largura da slice, e que, de OP' a OP corresponde à altura da stack, temos que:

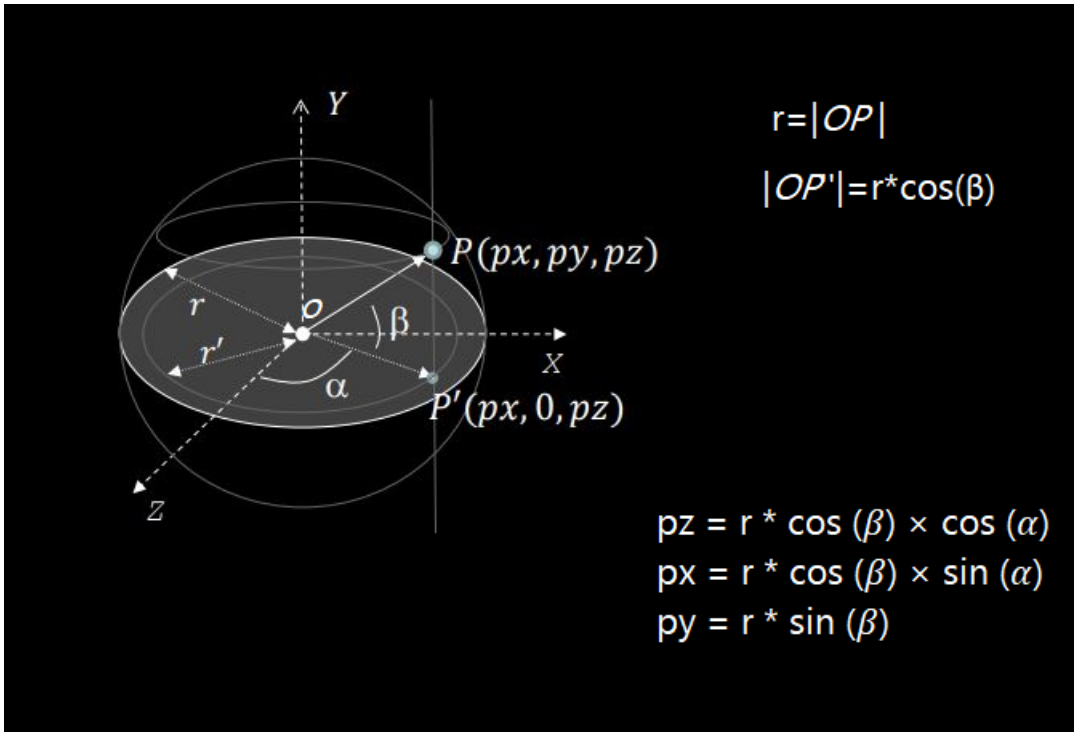


Figura 3.1: Coordenadas da esfera

Por fim, os dois triângulos são armazenados em um *vector* de triângulos. Após todos os pontos de todos os triângulos serem calculados escreve-se para o ficheiro .3d o número de pontos e de seguida os pontos propriamente ditos.

Cone

Inicialmente, cria-se a base do cone, que é um círculo formado por slices triângulares. O raio do círculo é definido pelo parâmetro radius, e o ângulo entre os triângulos é calculado pela fórmula (M_PI * 2) / slices (stepAlpha).

Depois, cria-se as fatias horizontais do cone, começando pela base e subindo até à penúltima *stack* do cone. Para cada *stack*, recalcula-se o raio da stack, que vai diminuindo à medida que o cone é construído.

Para isto as formulas utilizadas foram:

Point 1 = $(curRad * sin(alpha),\ curHeight,\ curRad * cos(alpha))$,
Point 2 = $(curRad * sin(alpha + step),\ curHeight,\ curRad * cos(alpha + step))$,
Point 3 = $((curRad - radStep) * sin(alpha + step),\ curHeight + stackStep,\ (curRad - radStep) * cos(alpha + step))$,
Point 4 = $((curRad - radStep) * sin(alpha),\ curHeight + stackStep,\ (curRad - radStep) * cos(alpha))$

Sendo que:

$curHeight = AlturaAtual$ & $curRad = RaioAtual$,
$stackStep = height \div stacks$ & $radStep = radius \div stacks$

Por fim, cria-se a parte superior do cone, que se trata de triângulos que ligam os pontos da última fatia horizontal à ponta do cone.

Cilindro & Prisma

Inicialmente, começa-se por criar os triângulos que formam as bases inferior e superior do cilindro. Para isso, percorre-se as fatias do cilindro, criando triângulos com vértices na borda da fatia e no centro da base, de acordo com o raio especifico (b_rad e t_rad), dependendo se esses raios são iguais ou não, teremos um cilindro ou um prisma.

Em seguida, inicia-se a construção da superfície lateral do cilindro. Percorre-se assim, novamente, as fatias do cilindro, mas desta vez para cada fatia ela cria um conjunto de triângulos que ligam as camadas horizontais do cilindro. O raio e a altura dos vértices são calculados para cada camada e fatia, sendo que no caso de um cilindro esses raios são sempre iguais.

Para isto as formulas utilizadas foram:

Point 1 = $(curRad * sin(alpha),\ curHeight,\ curRad * cos(alpha))$
Point 2 = $(curRad * sin(alpha + step),\ curHeight,\ curRad * cos(alpha + step))$

Point 3 = $((curRad - radStep) * sin(alpha + step), curHeight - stackStep, (curRad - radStep) * cos(alpha + step))$
Point 4 = $((curRad - radStep) * sin(alpha), curHeight - stackStep, (curRad - radStep) * cos(alpha))$

Sendo que:

$step = (\pi * 2) \div slices$
$stackStep = height \div stacks$
$radStep = abs(b\_rad - t\_rad) \div stacks$
$h\_height = height \div 2$

Torus

Inicialmente definimos algumas variáveis, como deltaAlpha (ângulo resultante entre slices) e deltaBeta (ângulo resultante entre stacks), que correspondem às diferenças dos ângulos de rotação em torno do eixo central e em torno da secção transversal, respetivamente. A construção do torus pode ser dividida em duas partes. Uma que corresponde à circunferência de raio R de uma stack e a outra que corresponde à circunferência do tubo de uma slice de raio r. Em seguida, temos esta ilustração que exemplifica o que foi referido anteriormente.

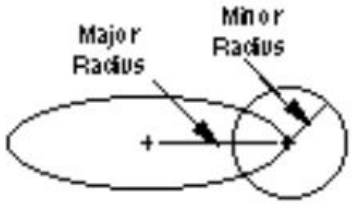


Figura 3.2: Construção da primitiva torus

Depois, entra em dois loops aninhados, um para cada ângulo de rotação. Para cada combinação de alpha e beta, o generator irá calcular as coordenadas dos quatro vértices do triângulo e os adiciona ao vetor de triângulos.

Assim, uma coordenada arbitrária poderá ser calculada da seguinte forma:
$x = (rad1 + rad2 * cos(teta1)) * cos(alfa)$
$y = rad2 * sin(teta1)$
$z = (rad1 + rad2 * cos(teta1)) * sin(alfa)$

sendo alfa um ângulo que varia de 0 a teta2 como é representativo da imagem seguinte (alfa vai sendo incrementado com um valor correspondente a deltaAlpha).

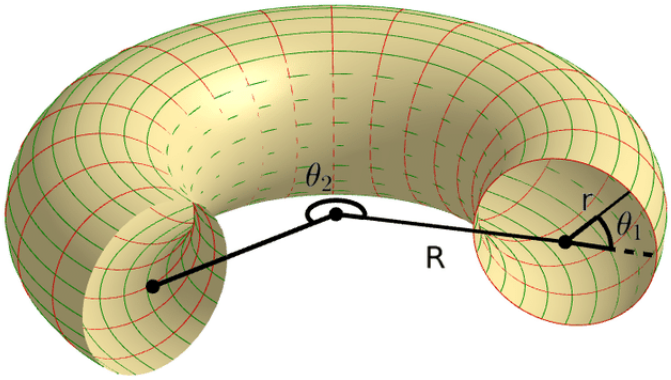


Figura 3.3: Coordenadas na primitiva torus

3.2 Engine

Ao executar a engine, inicialmente, é realizado o parse do ficheiro XML. A nossa equipa de trabalho optou por passar como argumento na execução da função main, o nome do ficheiro XML. Caso contrário, por default será considerado o ficheiro "config.xml". Neste ficheiro estarão algumas configurações da câmara, definições do display da janela e quais os modelos que devem ser apresentados.

A partir do tinyxml2 foi possivel armazenar em memória as configurações acima mencionadas. Para isso em primeiro lugar, foi declarado um apontador XMLDocument, e de seguida foi invocado o LoadFile com o path do ficheiro como argumento. A abertura do ficheiro será bem sucedida caso retorne 0. Posteriormente, foi possível fazer uma análise dos campos desejados recorrendo a apontadores XMLElement*. Quando uma tag A está aninhada numa tag B, realizou-se um A->FirstChildElement, de modo a obtermos um apontador XMLElement* da tag B. Caso as tags A e B estejam ao mesmo nível realizou-se um A->NextSiblingElement de modo a obtermos um apontador XMLElement* da tag B. De forma a obterem-se os valores dos componentes pretendidos realiza-se um Attribute do elemento XML desejado.

Após serem armazenados os modelos a desenhar num vector<string>, foi percorrida essa estrutura de dados e foram abertos os ficheiros 3d correspondentes. Na primeira linha encontra-se o número de linhas que o ficheiro possui, pelo que foi feito um ciclo

for em que a cada iteração foi realizado um getline e criado um objeto Point com os 3 componentes x, y e z. A cada iteração, estes pontos foram sendo armazenados num vector<Point>. De seguida, percorre-se essa estrutura de dados e desenha-se na renderscene pontos 3 a 3 com o auxílio de glVertex3f. Todos os pontos n0 vector<Point> vertex já se encontram ordenados de acordo com regra da mão direita.

Por outro lado, foi ainda definido algumas captações de eventos a partir do teclado (método 'processSpecialKeys'). Relativamente, ao modo de desenho (preenchimento) por default foi definido o glPolygonMode a GL_LINE. Ao pressionar a tecla especial $GLUT\_KEY\_END$ a(s) primitiva(s) ficam totalmente preenchidas (GL_FILL). Enquanto que ao pressionar a tecla $GLUT\_KEY\_PAGE\_DOWN$ a(s) primitiva(s) são representadas em pontos. Para voltar para modo linha, terá de ser pressionada a tecla $GLUT\_KEY\_PAGE\_UP$.

Para além desses eventos foram ainda implementados eventos de manipulação da posição da câmara (método 'processKeys'). Para as teclas 'q|Q' e 'e|E' foi manipulado o componente radius que corresponde ao raio do campo da câmara o que irá conduzir na sua aproximação ao modelo geométrico. As teclas 'a|A' e 'd|D' são responsáveis pelo incremento e decremento da variável G_alpha (ângulo da base esférica do campo da câmara que irá provocar o movimento para a esquerda e para a direita). Por sua vez as teclas 'w|W' e 's|S' manipulam G_beta que irá provocar uma subida e descida da câmara correspondentemente.

4. Manual de Utilização

De modo a gerar os modelos desejados, é necessário executar o generator com pelo menos 4 parâmetros. O primeiro é o nome do executável "generator", de seguida o nome da primitiva (sphere, box, cone, plane, cylinder ou prisma) e por fim o nome do ficheiro onde serão armazenados todos os pontos do modelo. Depois dependendo da primitiva desejada os parâmetros intermédios serão diferentes para cada um.
box: comprimento do lado, seguido do número de divisões da grelha
sphere: raio, seguido do número de fatias verticais(slices), e das fatias horizontais(stacks)
cone: o raio da base, a altura, as slices e por fim o número de stacks
plane: o comprimento dos lados, seguido do número de divisões da grelha
cylinder: raio da base e do topo, a altura, os slices e o número de stacks.
prisma: raio da base, raio do topo, a altura, os slices e o número de stacks.
torus: raio do circulo interior, raio do tubo, os slices e o número de stacks.

De forma a apresentarmos os modelos desejados temos que mencioná-los no ficheiro de configuração XML. Ficheiro este que será passado como argumento ao executar a engine. Uma limitação é que o ficheiro tem de estar ou na current working directory ou dentro de um folder na current working directory. Opcionalmente, poderá não ser passado nenhum ficheiro de configuração como argumento, sendo que por default será considerado o ficheiro "config.xml"apresentado de seguida.

```xml
<?xml version="1.0" encoding="utf-8"?>
<world>
    <window width="512" height="512" />
    <camera>
        <position x="8" y="4" z="4" />
        <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="plane.3d" />
            <model file="sphere.3d" />
        </models>
    </group>
</world>
```

Figura 4.1: Ficheiro de configuração XML por default

5. Testes

De modo a testar o nosso sistema foram executados os testes providenciados pelos docentes utilizando os vários ficheiros XML. Os resultado dos mesmos foram satisfatórios e de acordo como o desejável, como é possível averiguar pelas imagens seguintes. Para além das primitivas que foram enunciadas, complementámos também com a criação de um prisma, de um cilindro e com o torus (fig.5.6 , 5.7 e 5.8 correspondentemente).

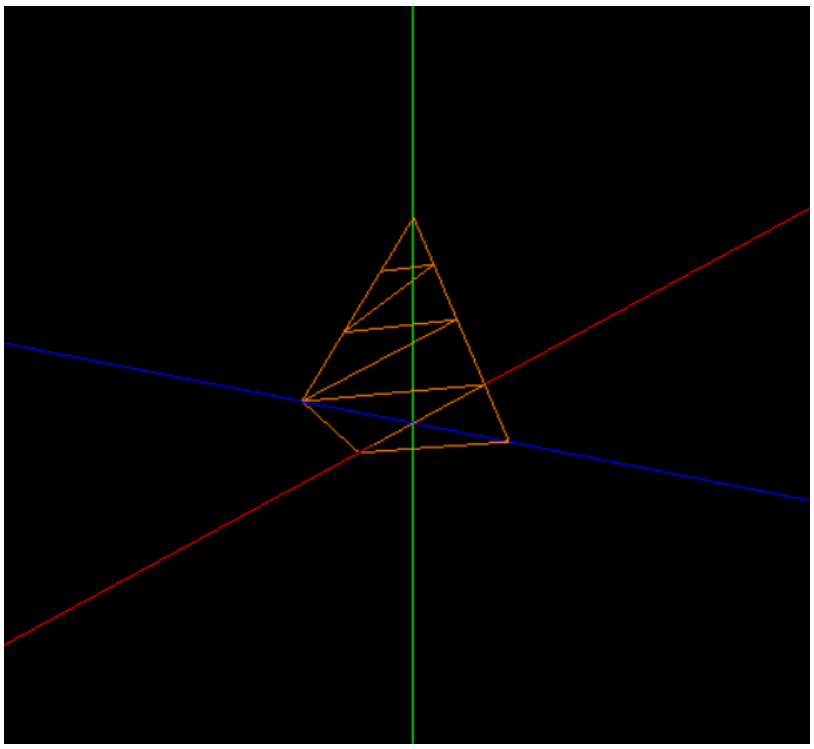


Figura 5.1: Resultado de test_1_1.xml

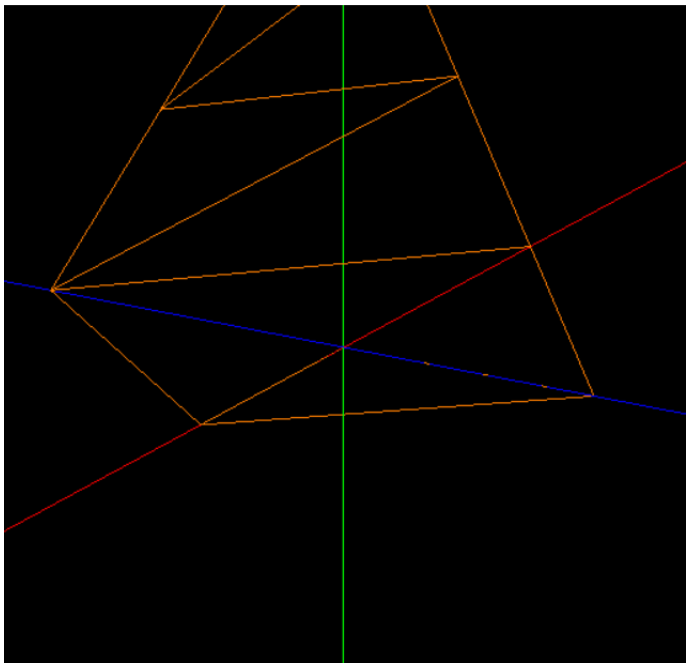


Figura 5.2: Resultado de test_1_2.xml

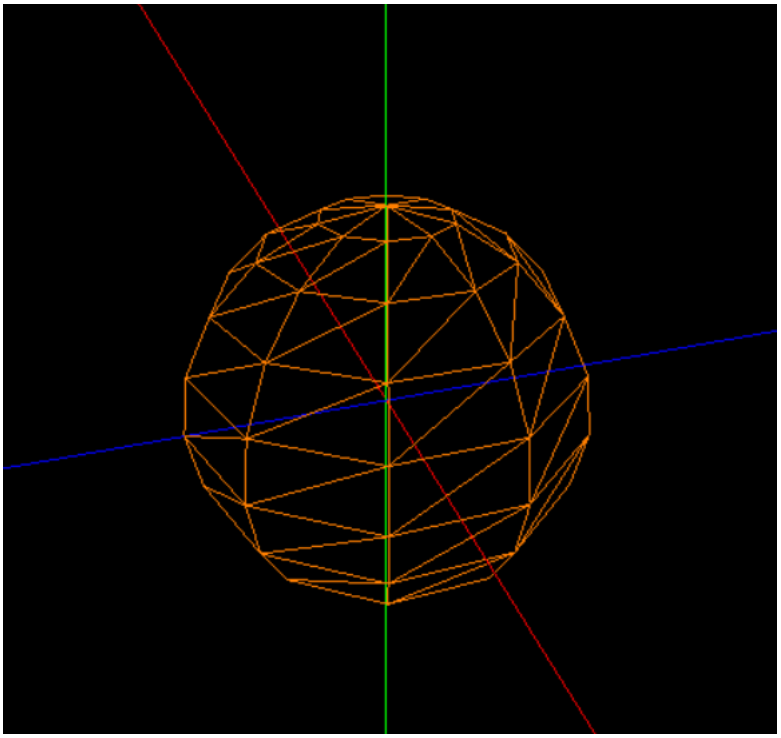


Figura 5.3: Resultado de test_1_3.xml

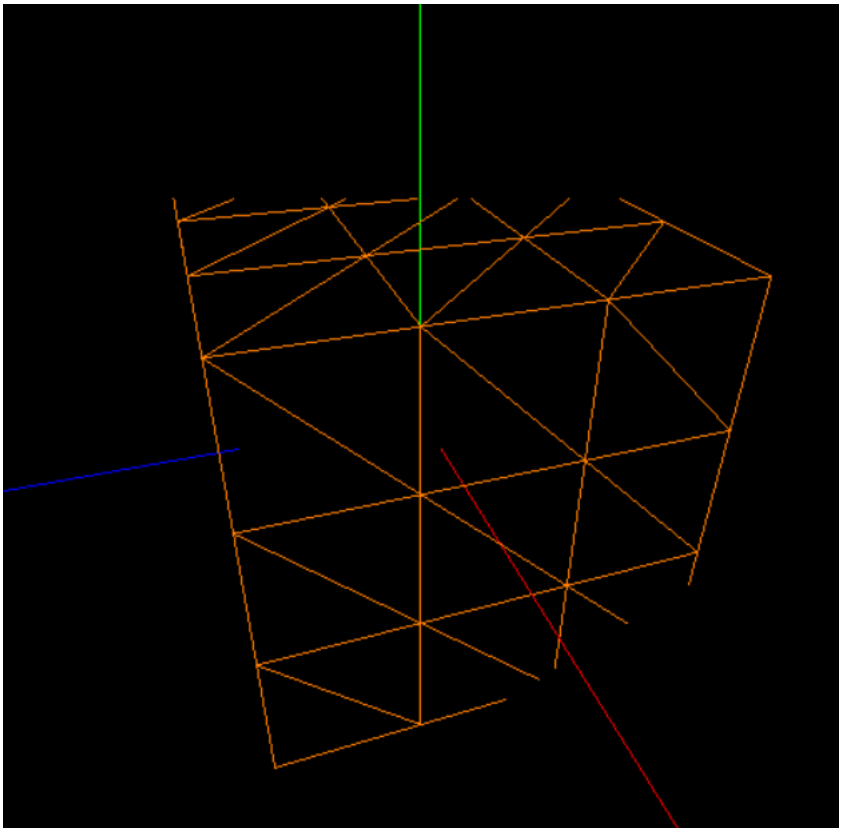


Figura 5.4: Resultado de test_1_4.xml

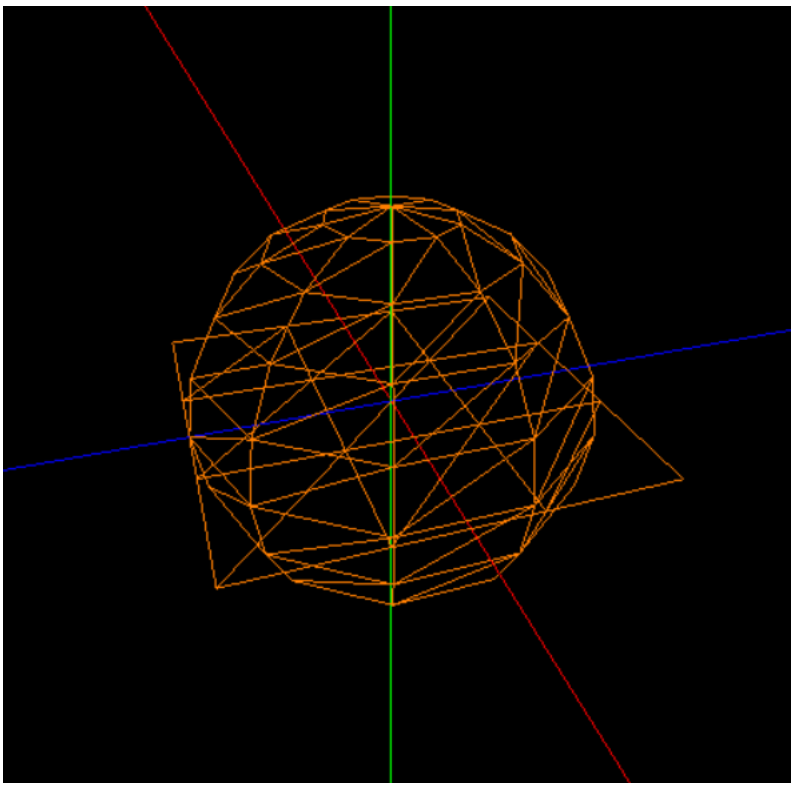


Figura 5.5: Resultado de test_1_5.xml

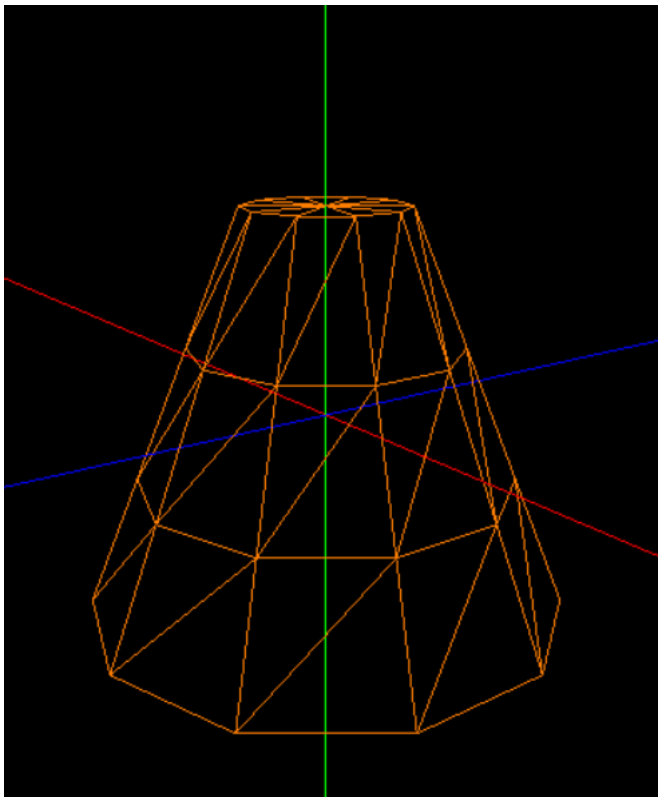


Figura 5.6: Prisma

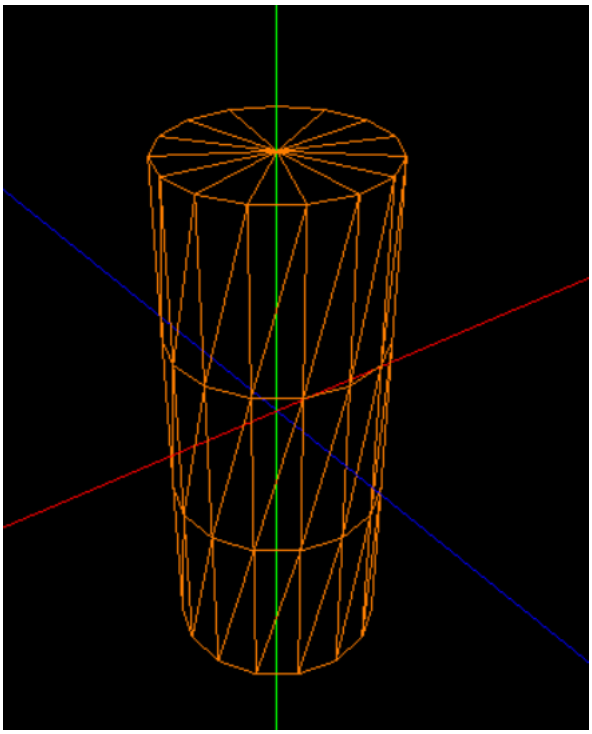


Figura 5.7: Cilindro

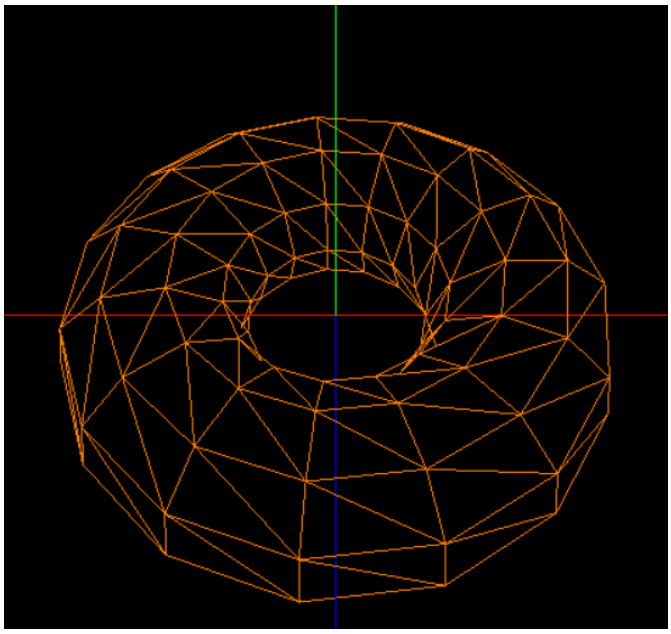


Figura 5.8: Torus

6. Conclusão

Durante a primeira fase do projeto, o grupo foi desafiado a usar os conhecimentos adquiridos nas primeiras semanas da UC para criar as formas geométricas desejadas pelos professores. Esta atividade permitiu que o grupo desenvolvesse competências na manipulação de vértices e triângulos para melhor criar formas geométricas.

Com a consolidação desses conceitos, o grupo sentiu-se mais confortável e confiante em relação às próximas fases do projeto. O uso de triângulos neste projeto tem um papel relevante, já que esta é a forma geométrica mais simples e fácil de computar e processar, o que a torna mais eficiente em relação às outras. Assim, com um entendimento mais sólido sobre a construção de primitivas, o grupo poderá concentrar-se em aprimorar as técnicas e na criação de formas mais complexas e detalhadas no futuro.

Além disso, o grupo também aprendeu a importância da colaboração e do trabalho em equipa na resolução de problemas e na criação de soluções eficazes.