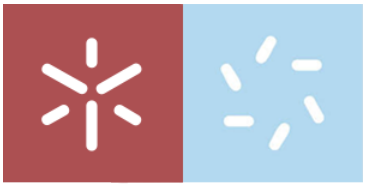
Universidade do Minho
Escola de Engenharia



Universidade do Minho
Escola de Ciências

Computação Gráfica

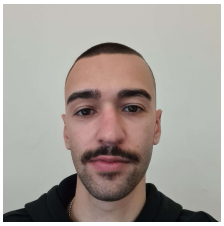
Guião 1

Grupo 16

Carlos André Machado Costa a94543
João Miguel Rodrigues da Cunha a96386
Rúben Gonçalo Araújo da Silva a94633

a94543

a96386



a94633

10 de março de 2023

Índice

1 Introdução

Na primeira fase deste projeto, no âmbito da disciplina de Computação Gráfica, foi proposto o desenvolvimento de duas aplicações. O primeiro é o gerador dos arquivos com as informações dos modelos e em segundo a engine que lê um arquivo de configuração, escrito em XML e exibe os respetivos modelos. O projeto é desenvolvido na linguagem de progamação C++ e recorren ao uso da biblioteca OpenGL.

2 Funcionalidades Implementadas

De momento temos implementadas as seguintes funcionalidades:

- Generator:
  - Primitivas:
    - plane;
    - box;
    - cylinder;
    - cone;
    - sphere;
    - torus.
- Engine:
  - Parser (TinyXML2);
  - Reader ficheiros ".3d";
  - Câmera:
    - Visão Esférica;
    - Primeira Pessoa.

3 Estrutura do Projeto

3.1 Ferramentas Utilizadas

Para demonstrar as funcionalidades do engine, utilizamos a ferramenta TinyXML2, que ajudou com o processo de parsing do cenário. Para além desta, tambem decidimos recorrer à biblioteca que nos ajudou com as funcionalidades gráficas do projeto, a OpenGL. A IDE utilizada foi o Visual Studio.

3.2 Aplicações

- **Generator:** Tem como função calcular o conjunto de vértices, que serão futuramente armazenados em ficheiros.
- **Engine:** Aqui lemos os ficheiros XML que tem as referências que foram criadas pelo gerador e os grupos que atualmente são só as primitivas gráficas, que recorrem ao uso da biblioteca OpenGL.

4 Generator

4.1 Definição

O generator é uma aplicação que recebe por parâmetros a primitiva gráfica, as características e o nome do ficheiro final ".3d". O generator cálcula todos os vértices das primitivas gráficas e exporta para o ficheiro final. Este contém inicialmente uma descrição breve para auxiliar a compreensão da primitiva em questão e por cada linha posteriormente, conterá 3 números floats referentes ao vértice em questão (cada 3 linhas representa um triângulo).

Para facilitar a visualização, os comandos dos vertices abaixo são do OpenGL, os nossos vértices são guardados em stringstream para posterior escrita no ficheiro ".3d".

4.2 Primitiva: plane

1. Calcular o tamanho de cada divisão:

$$divisions\_lenght = \frac{lenght}{divisions}$$

2. Calcular a translação para conseguirmos centrar o plano:

$$trans = \frac{lenght}{2}$$

   Este valor permite-nos posteriormente aplicar de modo subtrativo às coordenadas calculadas.

3. Gerar os triângulos seguindo a seguinte lógica:
   Para cada division(i):

   $z = (divisions\_lenght * i) - trans$
   $next\_z = (divisions\_lenght * (i + 1)) - trans$
   Para cada divisão (j):

   x e o $next\_x$ seguindo a mesma lógica do z, mas agora por cada divisão j;
   Desenhamos os triângulos necessários para gerar o plano:

   glVertex3f(x, 0, z)
   glVertex3f(x, 0, next_z)
   glVertex3f(next_x, 0, z)

   glVertex3f(next_x, 0, z)
   glVertex3f(x, 0, next_z)
   glVertex3f(next_x, 0, next_z)

4.3 Primitiva: box

Para explicar a box, apresentamos o cálculo da parte de cima e da parte de baixo, as laterais é exatamente a mesmo lógica, mas em vez de calcularmos o x, z, calculámos o y, z e o y, x. E o cálculo da translação fica sempre na variável que não é calculada.
Em suma, cálculamos duas faces de cada vez.

1. Calcular o tamanho de cada divisão:

$$divisions\_lenght = \frac{lenght}{divisions}$$

2. Calcular o a translação para conseguirmos centrar a box:

$$trans = \frac{lenght}{2}$$

   Este valor permite-nos posteriormente aplicar de modo subtrativo às coordenadas calculadas.

3. Gerar os triângulos seguindo a seguinte lógica:
   Para cada division(i):

   $z = (divisions\_lenght * i) - trans$
   $next\_z = (divisions\_lenght * (i + 1)) - trans$
   Para cada division(j):

   x e o $next\_x$ seguindo a mesma lógica do z, mas agora por cada divisão j;

   Desenhamos os triângulos necessários para gerar a lateral da box:

Parte superior da box:
glVertex3f(x, -trans, z)
glVertex3f(next_x, -trans, z)
glVertex3f(x, -trans, next_z)

glVertex3f(next_x, -trans, z)
glVertex3f(next_x, -trans, next_z)
glVertex3f(x, -trans, next_z)

Parte inferior da box:
glVertex3f(x, dimension - trans, z)
glVertex3f(x, dimension - trans, next_z);
glVertex3f(next_x, dimension - trans, z);

glVertex3f(next_x, dimension - trans, z);
glVertex3f(x, dimension - trans, next_z);
glVertex3f(next_x, dimension - trans, next_z);

4.4 Primitiva: cylinder

Para calcular o cylinder, é necessário desenha-lo por partes, bases e depois a lateral.

Bases:

1. Para calcular quantas slices do cylinder será necessário dividir por 360^{o} (em radianos) pelo número pretendido:

$$slice = \frac{2 * \pi}{slices}$$

2. Gerar os triângulos seguindo a seguinte lógica:
   Para cada slice(i):

   Ângulo atual e o próximo ângulo:
   $ang1 = slice * i$
   $ang2 = ang1 + slice$

   Coordenadas usando os ângulos anteriores:
   $x = (radius * sin(ang1))$
   $z = (radius * cos(ang1))$
   $next\_x = (radius * sin(ang2))$
   $next\_z = (radius * cos(ang2))$

   Desenhamos os triângulos necessários para gerar a base do cylinder:

   Base inferior:
   glVertex3f(0, 0, 0)
   glVertex3f(next_x, 0, next_z)
   glVertex3f(x, 0, z)

   Base superior:
   glVertex3f(0, height, 0)
   glVertex3f(x, height, z)
   glVertex3f(next_x, height, next_z)

Lateral:

1. Para calcular quantas stacks o cylinder terá na lateral, é necessário o seguinte cálculo:
$$height\_stack = \frac{height}{stacks}$$
2. Gerar os triângulos seguindo a seguinte lógica:
Para cada stack(i):

Para cada slice (calculado previamente)(j):

Ângulo atual e o próximo ângulo (igual às bases):
$ang1 = slice * i$
$ang2 = ang1 + slice$

Coordenadas do y usando os ângulos anteriores:
$y = (height\_stack * i)$
$next\_y = (height\_stack * (i + 1))$

Coordenadas (x, z) usando os ângulos anteriores:
$x = (radius * sin(ang1))$
$z = (radius * cos(ang1))$
$next\_x = (radius * sin(ang2))$
$next\_z = (radius * cos(ang2))$

Desenhamos os triângulos necessários para gerar a lateral do cylinder:

Triângulo da Esquerda:
glVertex3f(x, next_y, z)
glVertex3f(x, y, z)
glVertex3f(next_x, next_y, next_z)

Triângulo da Direita:
glVertex3f(x, y, z);
glVertex3f(next_x, y, next_z);
glVertex3fglVertex3f(next_x, next_y, next_z);

4.5 Primitiva: cone

A Matemática por de trás do cone é semelhante à do cylinder, mas temos de ter em atenção à amplitude.

Base:

1. Para calcular quantas slices a base do cone terá, será necessário dividir por $360^{\underline{o}}$ (em radianos) pelo número pretendido:

$$slice = \frac{2 * \pi}{slices}$$

2. Gerar os triângulos seguindo a seguinte lógica:
   Para cada slice(i):

   Ângulo atual e o próximo ângulo:
   $ang1 = slice * i$
   $ang2 = ang1 + slice$

   Coordenadas usando os ângulos anteriores:
   $x = (radius * sin(ang1))$
   $z = (radius * cos(ang1))$
   $next\_x = (radius * sin(ang2))$
   $next\_z = (radius * cos(ang2))$

   Desenhamos os triângulos necessários para gerar a base do cone:

   Base:
   glVertex3f(0, 0, 0)
   glVertex3f(next_x, 0, next_z)
   glVertex3f(x, 0, z)

   NOTA: Até este ponto, a lógica matemática é análoga à do cylinder, só com a diferença de ser uma única base que fica no plano XZ.

Lateral:

1. Para calcular quantas stacks o cone terá na lateral é necessário fazer o seguinte cálculo:
$$height\_stack = \frac{height}{stacks}$$
2. Gerar os triângulos seguindo a seguinte lógica:
Para cada stack(i):

Para cada slice (calculada previamente) (j):

Ângulo atual e o próximo ângulo (igual às bases):
$ang1 = slice * i$
$ang2 = ang1 + slice$

Coordenadas do y usando os ângulos anteriores:
$y = (height\_stack * i)$
$next\_y = (height\_stack * (i + 1))$

Coordenadas (x, z) inferiores:
$x1 = (radius * sin(ang1)) * (stacks - i)/stacks$
$z1 = (radius * cos(ang1)) * (stacks - i)/stacks$
$next\_x1 = (radius * sin(ang2)) * (stacks - i)/stacks$
$next\_z1 = (radius * cos(ang2)) * (stacks - i)/stacks$

Coordenadas (x, z) superiores:
$x2 = (radius * sin(ang1)) * (stacks - (i - 1))/stacks$
$z2 = (radius * cos(ang1)) * (stacks - (i - 1))/stacks$
$next\_x2 = (radius * sin(ang2)) * (stacks - (i - 1))/stacks$
$next\_z2 = (radius * cos(ang2)) * (stacks - (i - 1))/stacks$

Desenhamos os triângulos necessários para gerar a lateral do cone:

Triângulo da Esquerda:
glVertex3f(x2, next_y, z2)
glVertex3f(next_x1, y, next_z1)
glVertex3f(next_x2, next_y, next_z2)

Triângulo da Direita:
glVertex3f(x1, y, z1);
glVertex3f(next_x1, y, next_z1);
glVertex3fglVertex3f(x2, next_y, z2);

NOTA: A amplitude atual e a seguinte é calculada apartir desta fórmula:

$$(stacks - i)/stacks$$

Basicamente, com este cálculo, conseguimos aproximar o nosso ponto percentualmente ao eixo do Y (centro do cone).

4.6 Primitiva: sphere

Para calcular a sphere, seguimos as fórmulas das coordenadas esféricas. O β será representado pela *stack* e o α será representado pela *slice*.

1. Para calcular quantas slices (α) da sphere serão necessário dividir por $360^{\underline{o}}$ (em radianos) pelo número pretendido:

$$slice = \frac{2 * \pi}{slices}$$

2. Para calcular quantas stacks (β) da sphere serão necessário dividir por $180^{\underline{o}}$ (em radianos) pelo número pretendido:

$$stack = \frac{\pi}{stacks}$$

   NOTA: Pela lógica: $-90 < \beta < 90$ seria necessário começar em $-\frac{\pi}{2}$ e acabar em $\frac{\pi}{2}$. A nossa solução foi começar em $\frac{\pi}{2}$ e ir decrementando o ângulo de cada stack até chegar a $-\frac{\pi}{2}$.
   EM SUMA: A nossa esfera é gerada de cima para baixo.

3. Para cada slice(i):

   Para cada stack(j):

   Coordenadas do y:
   $y = radius * sin(\frac{\pi}{2} - (stack * j))$
   $next\_y = radius * sin(\frac{\pi}{2} - (stack * (j + 1)))$

   Coordenadas (x, z) superiores:
   $x1 = radius * cos(\frac{\pi}{2} - (stack * j)) * sin(slice * i)$
   $z1 = radius * cos(\frac{\pi}{2} - (stack * j)) * cos(slice * i)$
   $next\_x1 = radius * cos(\frac{\pi}{2} - (stack * j)) * sin(slice * (i + 1))$
   $next\_z1 = radius * cos(\frac{\pi}{2} - (stack * j)) * cos(slice * (i + 1))$

   Coordenadas (x, z) inferiores:
   $x2 = radius * cos(\frac{\pi}{2} - (stack * (j + 1))) * sin(slice * i)$
   $z2 = radius * cos(\frac{\pi}{2} - (stack * (j + 1))) * cos(slice * i)$
   $next\_x2 = radius * cos(\frac{\pi}{2} - (stack * (j+1))) * sin(slice * (i+1))$
   $next\_z2 = radius * cos(\frac{\pi}{2} - (stack * (j+1))) * cos(slice * (i+1))$

   NOTA 1: A variável $ang1$ e $ang2$ não foram escritas neste excerto para não acrescentar ruído vísual, pois teriamos de criar-las tanto para α o como para o β. Foram escritas dentro das funções *sin* e *cos* respetivamente;

Desenhamos os triângulos necessários para gerar a sphere:

Triângulo da Esquerda:
glVertex3f(x2, next_y, z2)
glVertex3f(next_x2,next_y, next_z2)
glVertex3f(next_x1, y, next_z1)

Triângulo da Direita:
glVertex3f(x1, y, z1);
glVertex3f(x2,next_y, z2);
glVertex3fglVertex3f(next_x1, y, next_z1);

4.7 Primitiva: torus

Para calcular o torus, seguimos as fórmulas de um website (link na bibliografia). EM SUMA: Para desenhar um torus é bastante semelhante a desenhar uma esfera, mas o ponto mais crucial é a translação do pontos calculados para o centro da "parte oca".

1. Para calcular quantos rings do torus será necessário dividir por 360º (em radianos) pelo número pretendido:

$$ring = \frac{2 * \pi}{rings}$$

2. Para calcular quantas sides do torus será necessário dividir por 360º (em radianos) pelo número pretendido:

$$side = \frac{2 * \pi}{sides}$$

3. Para cada ring(i):

   Para cada side(j):

   Coordenadas do y:
   $y = radius\_torus * sin(ring * i)$
   $next\_y = radius * sin(ring * (i + 1))$

   Coordenadas (x, z)**:**
   $x1 = CT + ((RT * cos(ring * i)) * cos(side * j))$
   $z1 = CT + ((RT * cos(ring * i)) * sin(side * j))$
   $next\_x1 = CT + ((RT * cos(ring * i)) * cos(side * (j + 1)))$
   $next\_z1 = CT + ((RT * cos(ring * i)) * sin(side * (j + 1))$

   Coordenadas (x, z)**:**
   $x2 = CT + ((RT * cos(ring * (i + 1))) * cos(side * j))$
   $z2 = CT + ((RT * cos(ring * (i + 1))) * sin(side * j))$
   $next\_x2 = CT + ((RT * cos(ring * (i + 1))) * cos(side * (j + 1)))$
   $next\_z2 = CT + ((RT * cos(ring * (i + 1))) * sin(side * (j + 1))$

   NOTA 1: A variável $ang1$ e $ang2$ não foram escritas neste excerto para não acrescentar ruído vísual, pois teriamos de criar-las tanto para α o como para o β. Foram escritas dentro das funções sin e cos respetivamente;

   NOTA 2: A Translação do ponto para o centro da "parte oca"é descrita no seguinte excerto:

$$CT + (.....)$$

Desenhamos os triângulos necessários para gerar o torus:

Triângulo da Esquerda:
glVertex3f(x1, y, z1)
glVertex3f(x2,next_y,z2)
glVertex3f(next_x1, y, next_z1)

Triângulo da Direita:
glVertex3f(x2, next_y, z2);
glVertex3f(next_x2,next_y, next_z2);
glVertex3fglVertex3f(next_x1, y, next_z1);

5 Engine

A engine é responsável por ler os ficheiros XML. Esta lê os ficheiros ".3d"gerados pelo generator e gera as figuras pretendidas. A nossa engine também conta com 2 tipos de camera motion, "Explorer Mode Camera"e "First Person Perspective Camera".
Atualmente, todos os dados necessários para as configurações como: posição da câmera, modos... foram todos guardados em variáveis globais.
Para ativar qualquer função por teclas ou rato, é necessário pressionar a tecla "P"para dizer à engine que queremos mexer na cena. Para mudar de câmera, basta clickar na tecla "M".

5.1 Read XML

1. Para não ser necessário criar código, usamos a library do "TinyXML2"adquirida do github deles (link na bibliografia).
2. Para estudarmos como usar essa ferramente, usamos uma documentação não oficial, mas bastante interessante com tudo o que precisamos (link na bibliografia).
3. Guardamos os valores do XML em variáveis globais.

5.2 Explorer Mode Camera

A posição da câmera é constantemente recalculada sempre que algum comando é chamado. Para este cálculo usamos as fórmulas das coordenadas da superfície esférica dada nas aulas práticas.

1. Conseguimos fazer zoomin e zoomout precionando a tecla + e -, respetivamente
2. Conseguimos mover a cena com o rato, desde que pressionemos previamente o botão esquerdo do rato

5.3 First Person Perspective Camera

O "lookUP"da câmera é constantemente recalculado sempre que algum comando é chamado. Para este cálculo usamos também as coordenadas da superfície esférica dada nas aulas práticas. NOTA: Temos de melhorar este modo de câmera, não está como pretendiamos.

1. Conseguimos andar para a frente ou para trás pressionando W e S, respetivamente
2. Conseguimos mover para onde estamos a olhar com o rato.

6 Modelos

6.1 plane

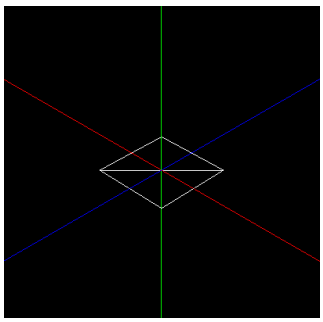


Figura 1: plane_1_1

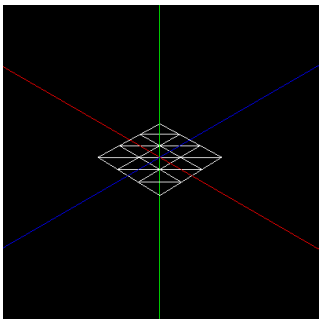


Figura 2: plane_1_3

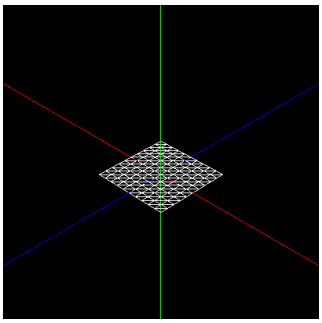


Figura 3: plane_1_10

6.2 box

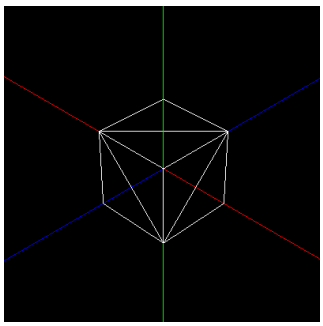


Figura 4: box_1_1

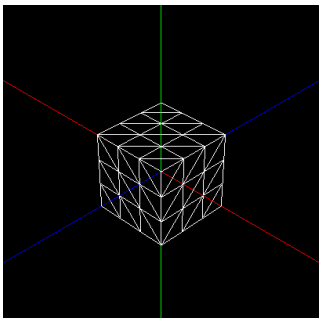


Figura 5: box_1_3

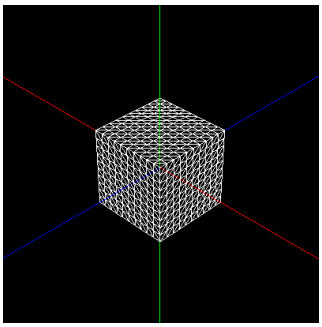


Figura 6: box_1_10

6.3 cylinder

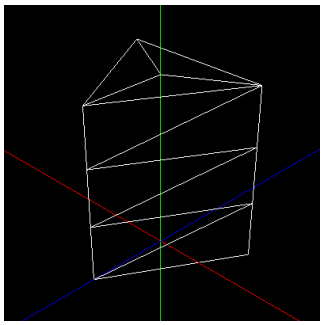


Figura 7: cylinder_1_2_3_3

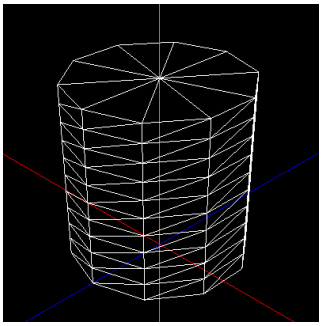


Figura 8: cylinder_1_2_10_10

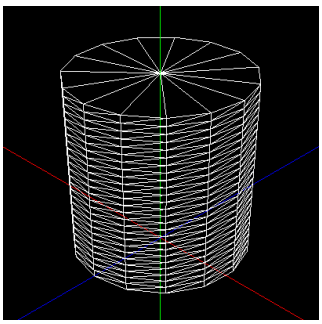


Figura 9: cylinder_1_2_15_27

6.4 cone

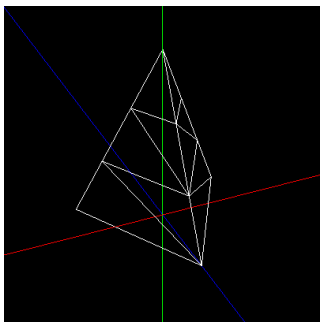


Figura 10: cone_1_2_3_3

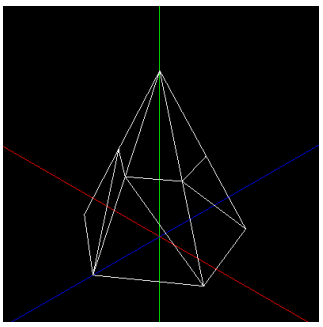


Figura 11: cone_1_2_5_2

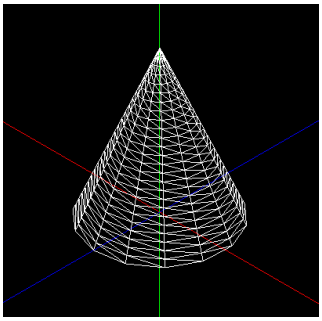


Figura 12: cone_1_2_15_25

6.5 sphere

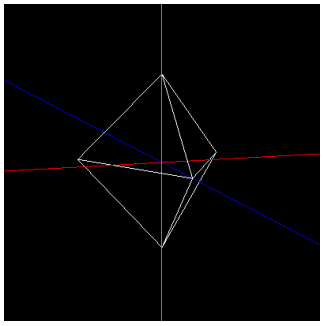


Figura 13: sphere_1_3_2

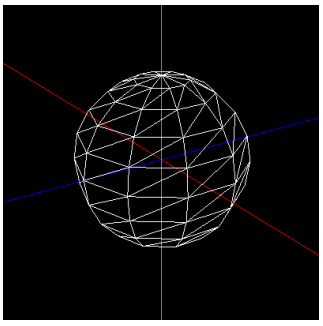


Figura 14: sphere_1_3_2

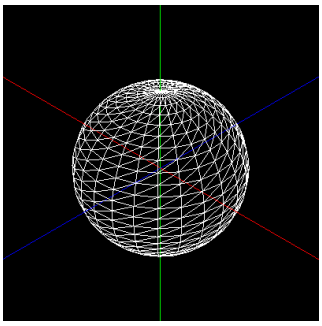


Figura 15: sphere_1_25_25

6.6 torus

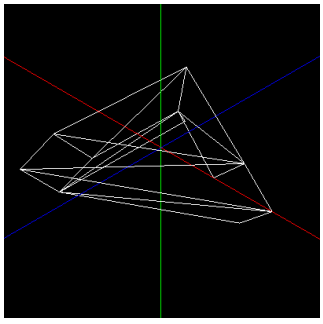


Figura 16: torus_1_2_3_3

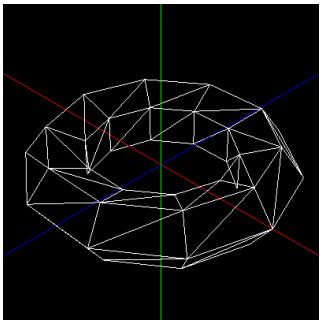


Figura 17: torus_1_2_5_10

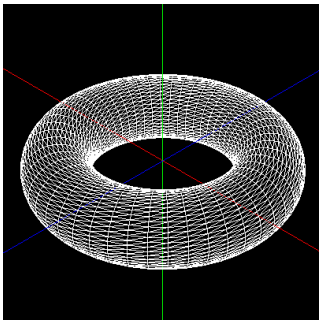


Figura 18: torus_1_2_50_50

6.7 extras

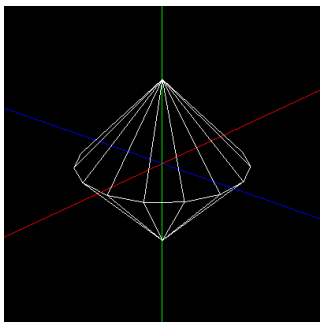


Figura 19: sphere_1_15_2

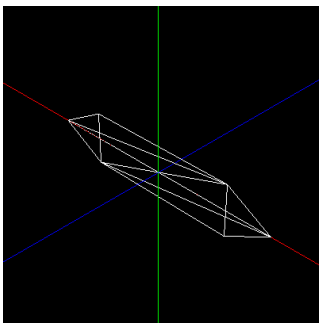


Figura 20: torus_1_2_2_3

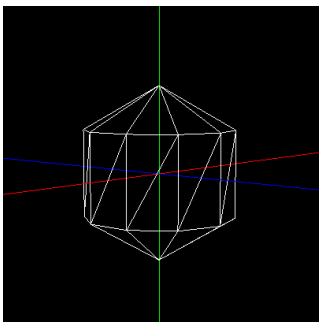


Figura 21: sphere_1_10_3

7 Demos

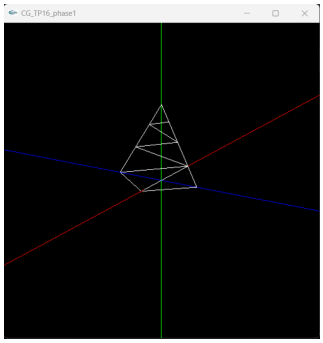


Figura 22: test_1_1

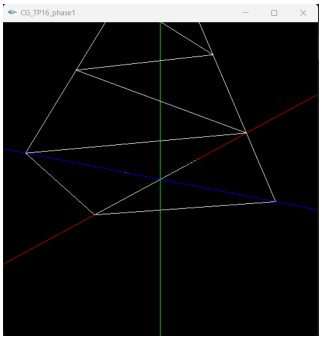


Figura 23: test_1_2

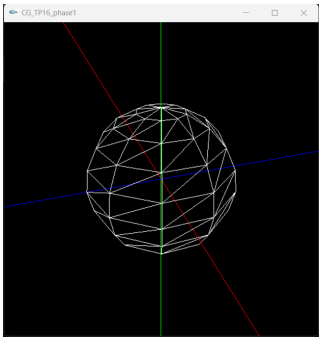


Figura 24: test_1_3

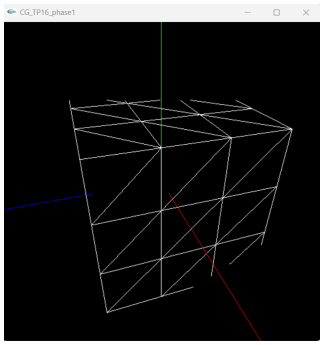


Figura 25: test_1_4

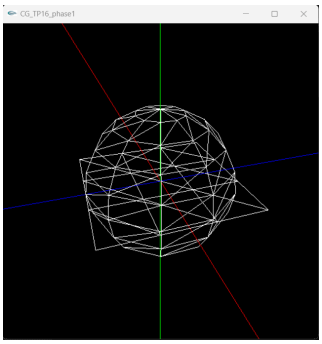


Figura 26: test_1_5

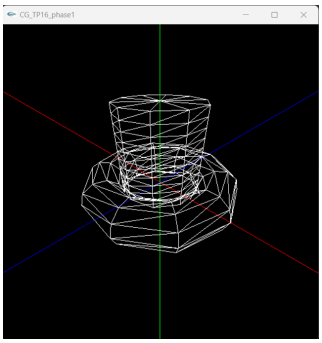


Figura 27: test_cilindro_torus

8 Conclusão

Durante esta primeira fase tivemos de lidar com situações difíceis que nunca nos deparamos durante as aulas. No entanto, olhamos para isto de forma positiva pois levou-nos a ter mais atenção a promenores em relação ao trabalho que fazíamos e tambem tivemos um maior empenho na realização deste trabalho.
A presente fase permitiu a consolidação da matéria lecionada nas aulas, especificamente sobre a utilização e domínio de ferramentas e conceitos relacionados com a UC, utilizado para isso OpenGL e, ainda, permitiu que explorassemos novos domínios como a leitura de ficheiros xml.

9 Bibliografia/Webgrafia

- Material da cadeira de Computação Gráfica 22/23
- Torus Fórmulas
- TinyXML2 Docs