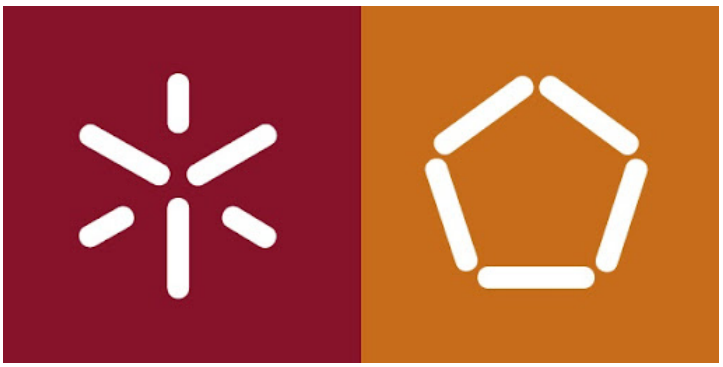


Universidade do Minho

Unidade Curricular:

Computação Gráfica

Relatório Fase 1- Projeto Prático

Elementos do grupo:

A93253: David Alexandre Ferreira Duarte
A89518: Ema Isabel Quintãos Dias
A94166: Samuel de Almeida Simões Lira

Introdução

Na primeira fase do projeto prático, é pedido à equipa que desenvolva um sistema *generator*/*engine* capaz de produzir as seguintes figuras geométricas: plano, cubo, esfera e plano. Desta forma, o *generator* deverá ser capaz de produzir um ficheiro com os pontos necessários à construção e o engine deverá ler esse mesmo ficheiro e produzir a figura em ambiente OpenGl.

Para tal, foram fornecidos ficheiros xml por parte da equipa docente, onde estão descritas as instruções da câmara e os ficheiros das figuras geométricas que se pretendem observar.

Generator

A equipa decidiu dividir o arquivo responsável pela implementação do *generator* em dois, com os respetivos *headers*, sendo que o *generator.cpp* invoca a implementação realizada no *generator_aux.cpp*.

Primeiramente, é necessário perceber qual a figura pedida para criar e, para isso, na função *main* é realizada uma divisão para cada uma das figuras geométricas, encaminhando para as funções implementadas no ficheiro auxiliar. Posto isto, sabendo qual é o modelo pedido, resta parametrizar os campos relativos às unidades de tamanho da figura, e sendo o caso, o número de *slices* e *stacks* pretendidas. Além disso, ainda é necessário saber para que ficheiro se pretende enviar os pontos calculados.

Generator_aux.cpp

Estruturas

Para desenvolver o projeto, decidiu-se criar duas importantes estruturas para agregar as informações relativas aos pontos calculados.

Primeiramente, criou-se a estrutura correspondente a um vértice, *Vertex*, como se sabe, um vértice contém três pontos (x,y e z), sendo cada um deles do tipo float. Assim, poder-se-á criar vértices que serão necessários para a construção dos triângulos essenciais à construção das figuras.

Como tal, também se criou a estrutura *Triangle*, capaz de armazenar três vértices do tipo *Vertex*.

Ficheiros .3d

Tal como referido anteriormente, o *generator* deve ser capaz de produzir ficheiros com os pontos, para mais tarde serem lidos pelo *engine*. Desta forma, após a criação dos triângulos, é escrito no ficheiro os vértices separados por vírgulas. Auxiliando-se em funções que convertem os vértices em strings, separando os três floats por “;”.

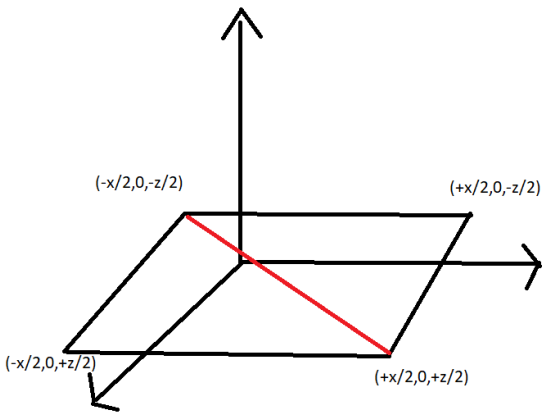
Assim, ao longo da implementação foram criadas as funções que calculam os pontos para cada uma das figuras.

Plano

Para criar o plano desenvolveu-se a função *createPlane*, que recebe como argumentos as unidades que o plano deverá ter como tamanho e o número de divisões que o mesmo deverá ter.

Um plano encontra-se em *xoz*, e como tal, necessita de realizar dois ciclos que percorrem o ponto mais negativo de x e o ponto mais negativo de z, somando-lhe iterativamente as divisões até chegar ao ponto mais positivo de ambos. O valor a somar iterativamente corresponde às unidades divididas pelo número de divisões. Toma-se como ponto mais negativo a metade do valor das unidades, ou seja, admitindo que a figura deve ser desenhada na origem, o comprimento e a largura do plano são divididos por dois, ficando no ponto mais negativo de x: -x/2 e no ponto mais positivo de x: x/2, onde x é o valor das unidades passado como argumento. O mesmo se aplica para z.

Por cada iteração são criados quatro vértices e dois triângulos, colocando no string* triangles (variável global) os mesmos, e no final dos ciclos enviando os pontos calculados para ficheiro inicialmente aberto que foi passado como argumento.

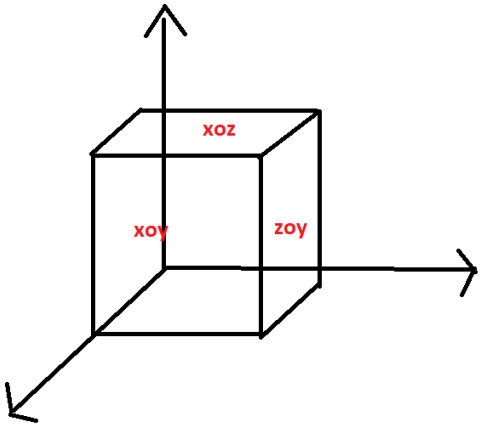


Tomemos como exemplo, o rascunho de um plano com uma única divisão.

Cubo

Para realizar a construção do cubo, é necessário utilizar como base os conhecimentos da construção do plano, já que sabemos que o cubo é constituído por seis faces, duas delas no plano xoy, duas no plano xoz e outras duas no plano zoy. Cada uma destas faces, necessita de ser iterada num ciclo em tudo comum com o ciclo do plano, considerando-se que na face xoy, o valor de z é constante, no plano xoz o valor de y é constante, e no plano zoy o valor de x é constante.

Portanto, são criados três ciclos aninhados, onde se constroem duas faces para cada um deles.



Vejamos o rascunho seguinte:
Tanto a face da frente do cubo como a de trás, encontram-se como planos xoy, ou seja os ciclos a iterar sobre as mesmas são aqueles que alteram o valor de x e de y iterando de acordo com as divisões pretendidas. As faces laterais, encontram-se no plano xoy e a de cima e a debaixo no plano xoz.

Posto isto, para cada um dos ciclos aninhados são construídos quatro triângulos, dois para cada face enviando a informação dos pontos calculados para o ficheiro.

Esfera

Para a escrita das coordenadas no ficheiro .3d, a função createSphere recebe o raio da esfera, o número de stacks e slices como argumentos. Para construção de uma esfera utilizamos dois ciclos aninhados, poderíamos ter optado por desenhar uma triangle strip ao invés de triângulos individuais, mas a fim de uniformizar o desenho das 4 formas na engine, optamos pela segunda opção. São utilizados apenas 4 vértices para construir os dois triângulos, embora haja uma sobreposição de vértices, que são escritos a cada iteração do ciclo interior, são calculados em função dos ângulos alfa, horizontal, e beta, vertical, um vértice é calculado da seguinte forma geral:

x=radius*cos((origem) + i*delta_beta)*sin(j*delta_alfa),
y=radius*sin((origem) + i*delta_beta),
z=radius*cos((origem) + i*delta_beta)*cos(j*delta_alfa)

O ângulo origem(beta) tendo o valor -pi sobre 2 sendo adicionado o delta_beta, pi sobre o número de stacks, vezes o número de stacks desenhadas, e o número de slices desenhadas, j, vezes delta_alfa, 2 vezes pi sobre o número de slices. Os triângulos são então escritos no ficheiro .3d.

Cone

No desenvolvimento do cone, que tem como parâmetros o raio, a altura, o número das slices e o número das stacks, a estratégia abordada para a criação dos vértices do mesmo foi a utilização das coordenadas polares. Como cada vértice é constituído por 3 coordenadas, ou seja, a coordenada no plano do eixo do x, a coordenada no plano do eixo do y e a coordenada no eixo do z. Para calcular as coordenadas polares utlizou-se as seguintes equações:

*alfa = (2 * M_PI) / slices*
*px = raio * sin(beta * slice)*
py = stack_height
*pz = raio * cos(omega * slice)*

(beta e omega são ângulos calculados através do alfa e do número do step na construção das slices, sendo o beta o ângulo seguinte e ômega o ângulo anterior. O step é utilizado como um contador nos ciclos de construção dos vértices, uma vez que a condição de paragem de cada ciclo é o número de stacks e de slices dada)
Isto foi feito no caso do step slices não ser logo o primeiro, ou seja ser 0, porque se fosse o primeiro, calcular-se-ia os vértices para fazer a base do cone.

Engine

No engine começa-se por ler o ficheiro xml pretendido, com recurso a tinyxml2. Na leitura do mesmo, encontramos os valores pretendidos para a câmera e os ficheiros dos pontos que se querem ler.

Para a câmara, criaram-se variáveis globais que são preenchidas conforme a leitura do xml, e atribuídas às funções do glut: gluLookAt e gluPerspective.

Em relação ao nome dos ficheiros .3d são guardados numa string* e enviados para a função responsável pela leitura dos mesmos.

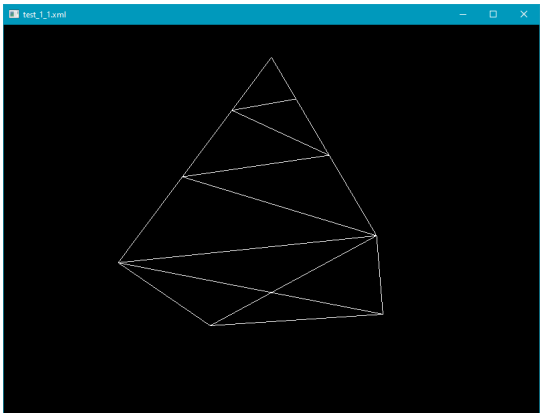
Essa função tem como nome *read3dfiles* e recebe como argumentos os nomes dos ficheiros e o número dos mesmos. É gerado um ciclo responsável de iterar todos os nomes dos ficheiros. Para cada um deles e, através da função *fscanf*, são retirados os pontos lidos, adicionados à estrutura Vertex e de seguida Triangle. Através da função *push_back,* os triângulos são guardados num vetor global, para que na função *RenderScene* se itere esse mesmo vetor e se desenhe os pontos correspondentes a cada um dos três vértices.

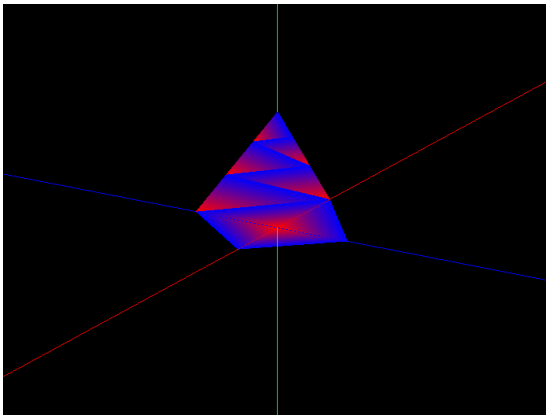
Resultados Finais:

Nesta secção serão apresentadas as imagens dos ficheiros fornecidos pela equipa docente, em comparação aos resultados que a equipa obteve. **temos de alterar as imagens com a esfera**

No ficheiro test_1_1.xml, pretendia-se obter a seguinte imagem do lado esquerdo, e a equipa obteve a do lado direito:

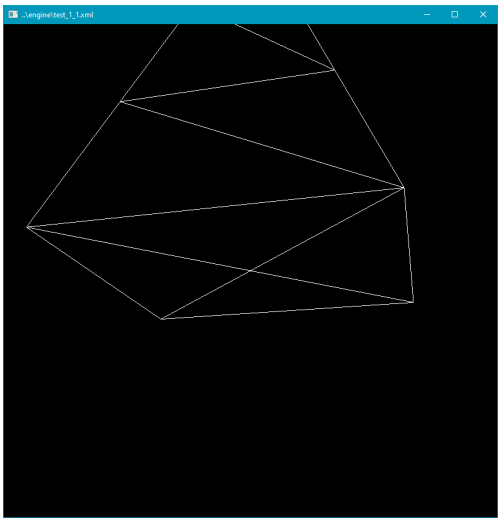
model: *cone 1 2 4 3 cone.3d*

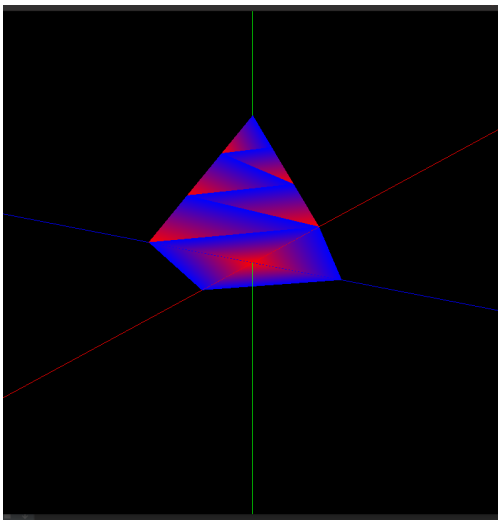




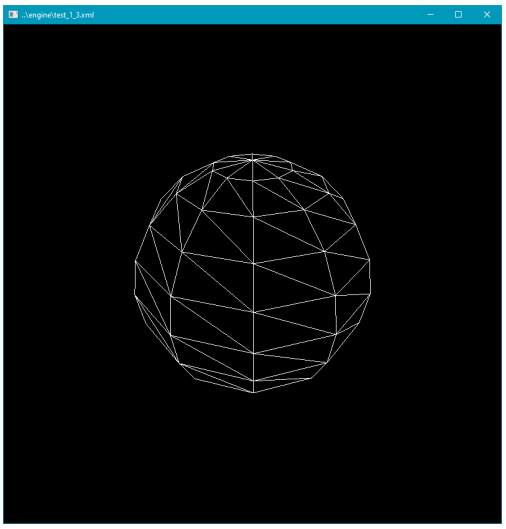
Da mesma forma, para o test_1_2.xml obteve-se:

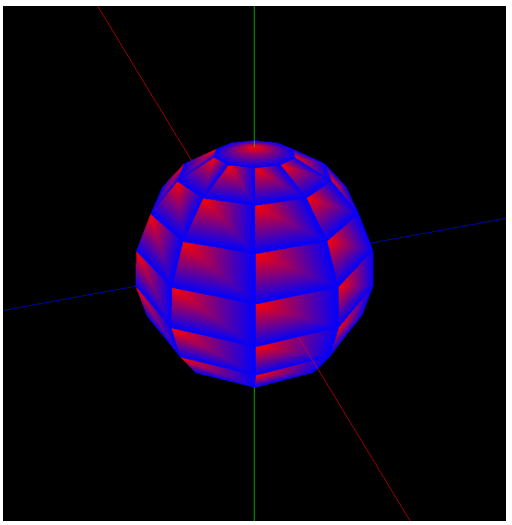
model: *cone 1 2 4 3 cone.3d*, mas para uma diferente posição de câmara



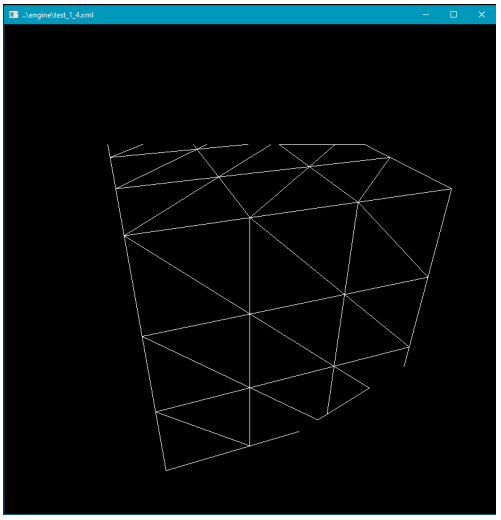


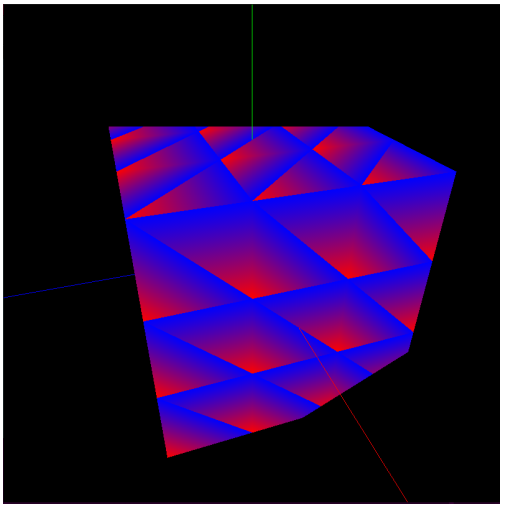
Para o test_1_3.xml, tem-se como model: *sphere 1 10 10 sphere.3d*





Já o test_1_4.xml, a equipa obteve o model: *box 2 3 box.3d*





Por fim o test_1_5.xml os model's pretendido era:

plane 3 plane .3d

sphere 1 10 10 sphere .3d