Computação Gráfica
Projeto Prático: Fase 1

Licenciatura em Ciências da Computação

Grupo 3

Alef Keuffer (A91683), Alexandre Rodrigues Baldé (A70373)

Notas sobre Implementação

`Engine.cpp`

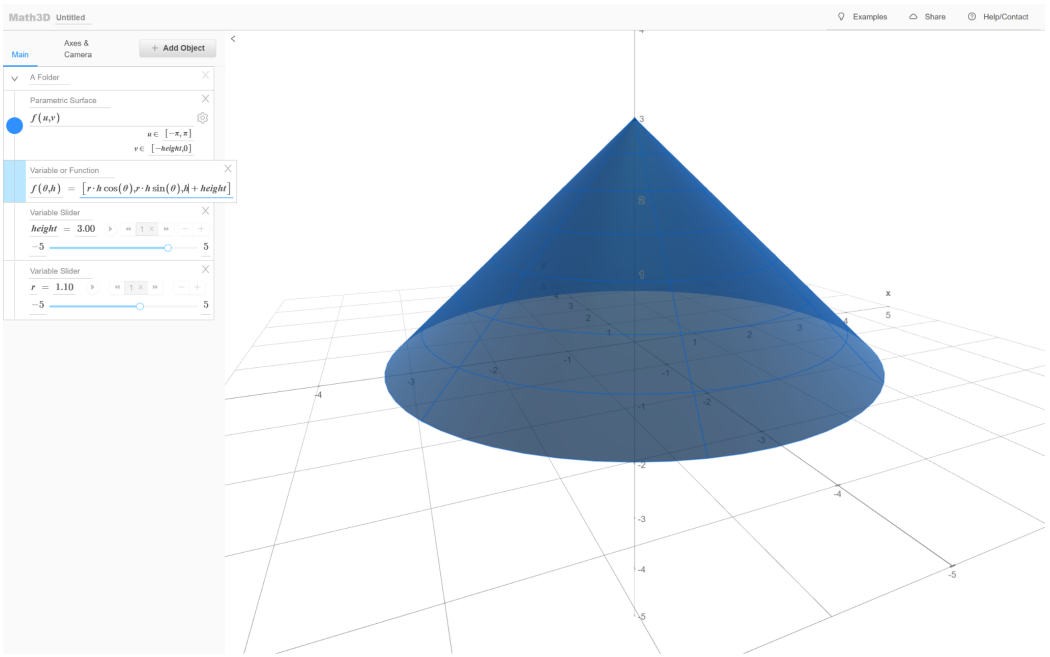
Geração de vértices para modelos

Esta seção do projeto não causou muita dificuldade: por sugestão do colega Alef, usou-se o serviço https://www.math3d.org para a visualização de superfícies em 3D, para posterior coleção das equações utilizadas no código C++ para depois se gerarem os vértices.

O serviço Math3D é interativo, permitindo a manipulação das figuras em tempo real para testar as hipóteses que se discutiram para implementar os algoritmos.

Cone

Veja-se, por exemplo, o caso do [cone]:



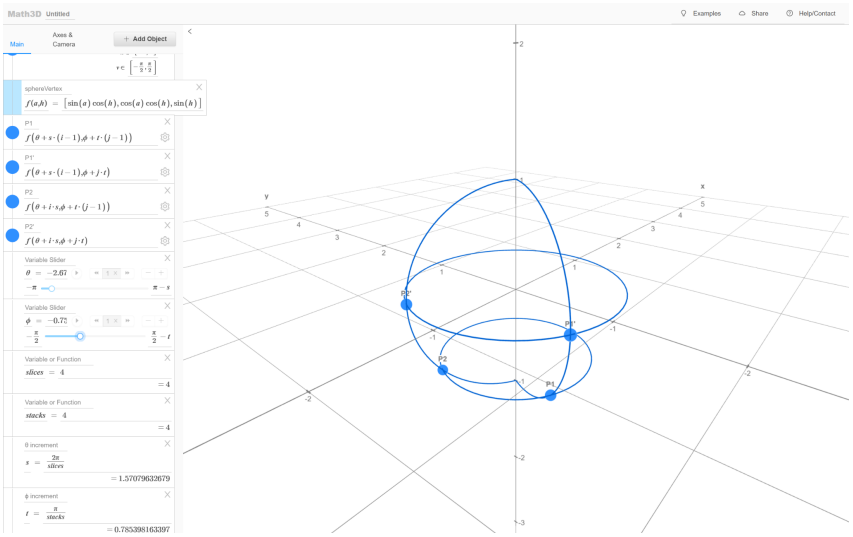
A imagem acima, que representa um modelo interativo, deu origem ao código

```
void drawCone(float r, float height, unsigned int slices, unsigned int
stacks) {
        // https://www.math3d.org/5gLCN9yBz

        float s = 2.0f * (float) M_PI / (float) slices;
        float t = height / (float) stacks;
        float theta = -M_PI;
        float h = -height;

        unsigned int nVertices = slices * stacks * 9;
        fwrite(&nVertices, sizeof(unsigned int), 1, globalFD);

        for (int m = 1; m <= slices; m++) {
```

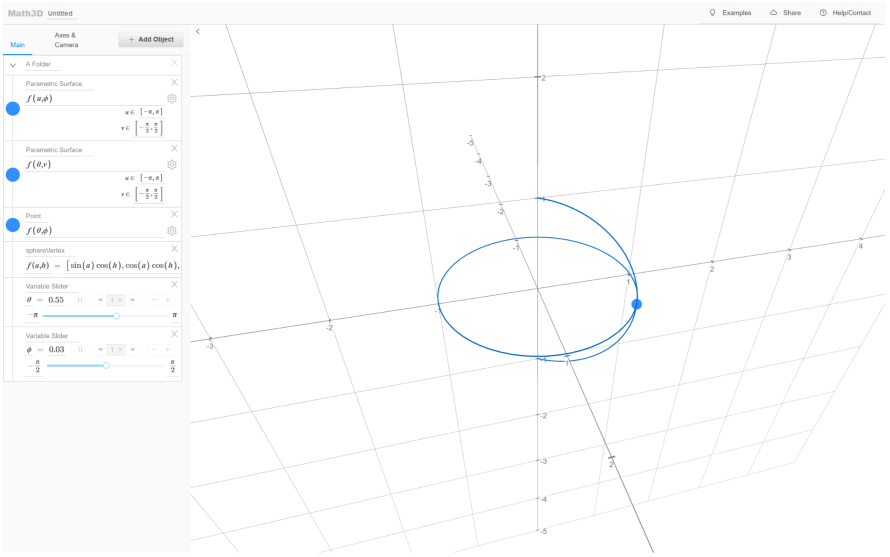
```
        for (int n = 1; n <= stacks; n++) {
        float i = (float) m;
        float j = (float) n;

        //base
        writeVertex(0, 0, 0); //O
        coneVertex(r, height, theta + s * (i - 1), h); //P1
        coneVertex(r, height, theta + s * i, h); //P2

        coneVertex(r, height, theta + s * i, h + t * (j - 1)); // P2
        coneVertex(r, height, theta + s * (i - 1), h + t * j); // P1'
        coneVertex(r, height, theta + s * i, h + t * j); //P2'

        coneVertex(r, height, theta + s * i, h + t * (j - 1)); // P2
        coneVertex(r, height, theta + s * (i - 1), h + t * (j - 1)); // P1
        coneVertex(r, height, theta + s * (i - 1), h + t * j); // P1'
        }
        }
}
```

Veja-se a tradução quase direta para C++.

Esfera



Veja-se que no exemplo acima, e no abaixo, a página apresenta “sliders” que permitem fazer variar o valor das coordenadas esféricas com se modelou a esfera, para se entender intuitivamente o conceito antes de o implementar.



A esfera deu origem a este código:

```
void drawSphere(float r, unsigned int slices, unsigned int stacks) {
        float s = 2.0f * (float) M_PI / (float) slices;
        float t = M_PI / (float) stacks;
        float theta = -M_PI;
        float phi = -M_PI / 2.0f;

        unsigned int nVertices = slices * stacks * 6;
        fwrite(&nVertices, sizeof(unsigned int), 1, globalFD);

        for (int m = 1; m <= slices; m++) {
        for (int n = 1; n <= stacks; n++) {
        float i = (float) m;
        float j = (float) n;

        sphereVertex(r, theta + s * (i - 1), phi + t * j); // P1'
        sphereVertex(r, theta + s * i, phi + t * (j - 1)); // P2
        sphereVertex(r, theta + s * i, phi + t * j); //P2'

        sphereVertex(r, theta + s * (i - 1), phi + t * (j - 1)); // P1
        sphereVertex(r, theta + s * i, phi + t * (j - 1)); // P2
        sphereVertex(r, theta + s * (i - 1), phi + t * j); // P1'
        }
        }
}
```

Note-se que tanto na esfera como no cone, as funções `coneVertex, sphereVertex` são "*helpers*" para, depois de efetuado o cálculo das coordenadas esféricas dos vértices das figuras, serem convertidas para coordenadas euclidianas usuais.

Plano, Cubo

Calcular as coordenadas dos vértices para os modelos do plano e do cubo foi simples.

Aqui vale apenas notar que no caso do plano, para ser visível de qualquer direção, os triângulos que compõem cada faixa são calculados duas vezes: uma com os seus pontos em ordem "*clockwise*", e outra vez pela ordem "*counter-clockwise*".

`Generator.cpp`

Como requerido no enunciado, a leitura do ficheiro XML é efetuada uma só vez.

Ficheiros `.3d`

Os ficheiros `.3d` que contêm cada modelo são também lidos uma só vez.

No início de cada tal ficheiro, estão 4 "*bytes*" que contém o número de coordenadas (ou seja, o triplo do número de vértices) nele contido.

No programa `` `generator` ``, guarda-se, para cada modelo, os seus vértices numa estrutura

```
typedef struct model {
    float *vertices;
    unsigned int nVertices;
} *Model;
```

Sendo que para a Fase 2, a leitura de modelos de XML vai ter que ser alterada na totalidade, para esta fase optou-se

pela simplicidade de guardar um vetor com um `Model` para cada modelo:

```
static std::vector<Model> globalModels;
```

Assim sendo, na função `renderScene()` chama-se `renderAllModels();`, onde

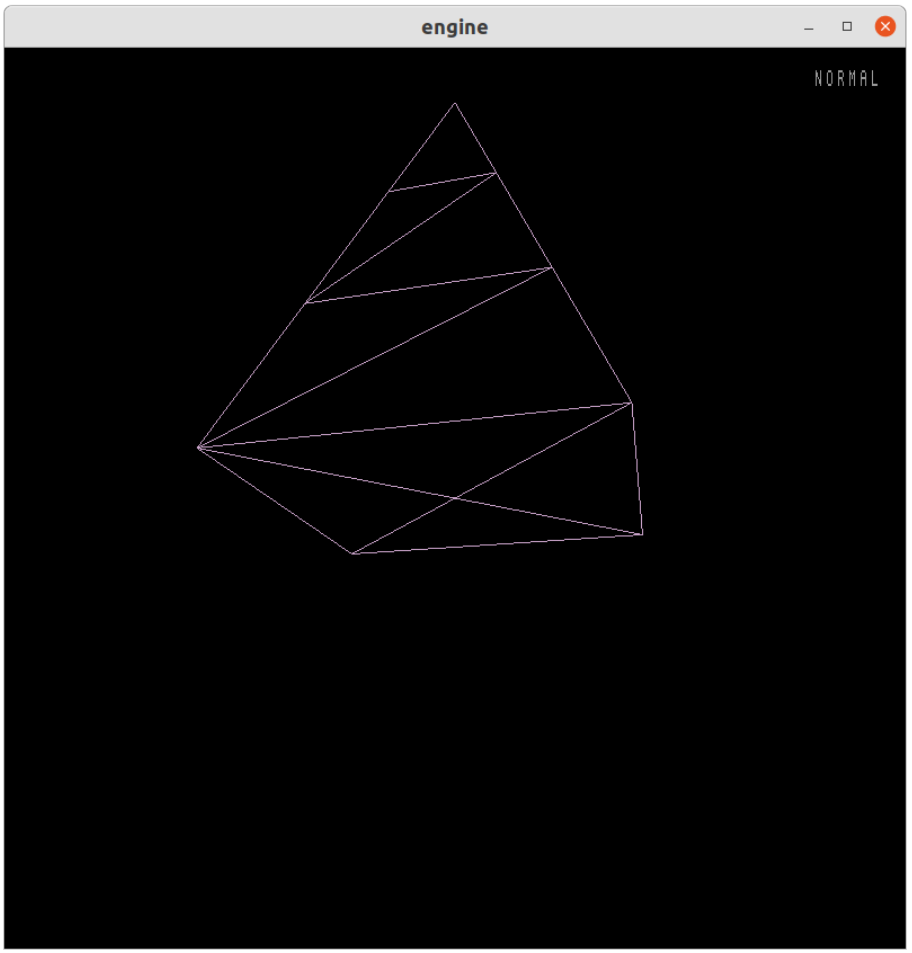
```
void renderAllModels() {

   for (Model m: globalModels) renderModel(m);

}
```

Para fazer "*parse*" dos ficheiros XML, utilizou-se a biblioteca de C++ TinyXML2, cuja sugestão pelo Professor Ramires se agradece.

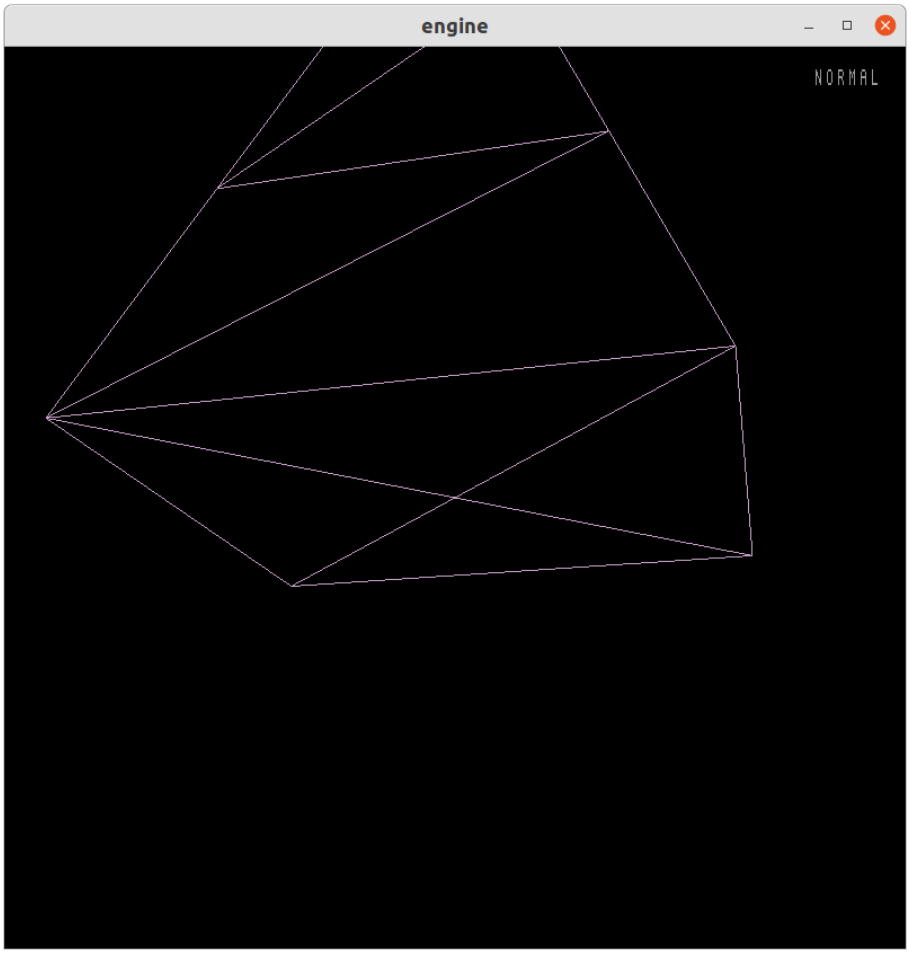
Exemplos de utilização

Apresentam-se agora os resultados de correr a aplicação produzida nos ficheiros de teste XML fornecidos, pela ordem que foram dados.

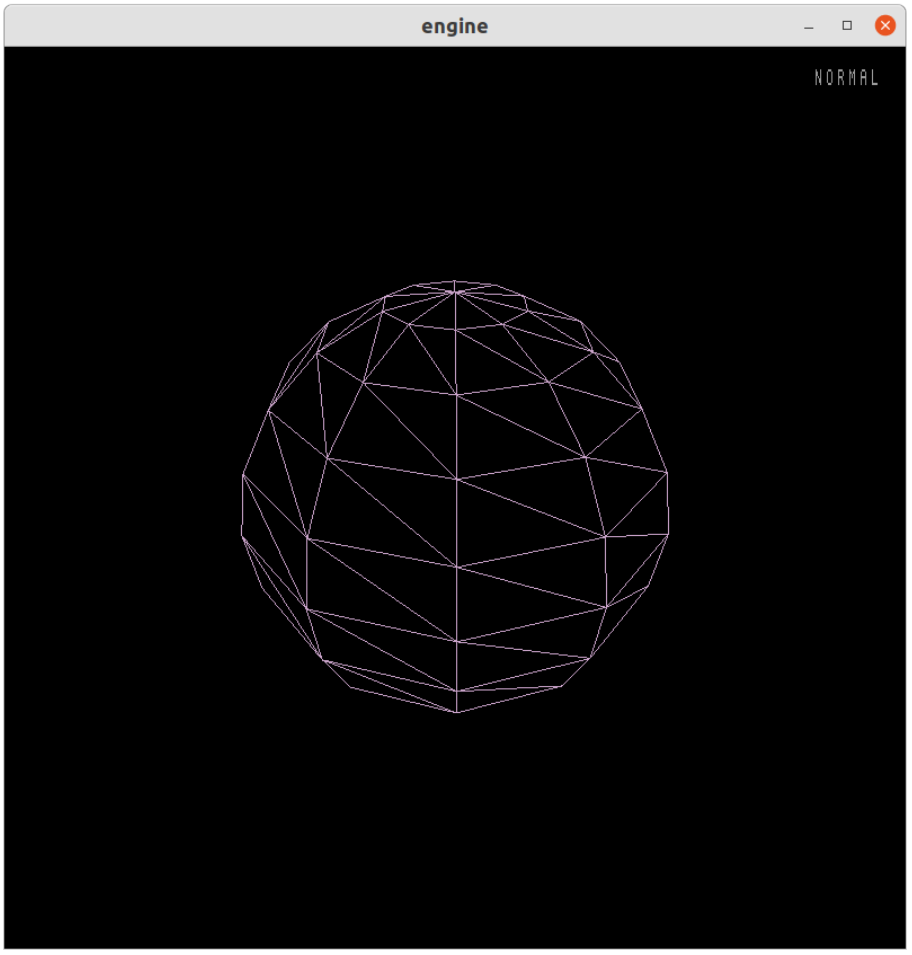
`test_files/test_1_1.xml`



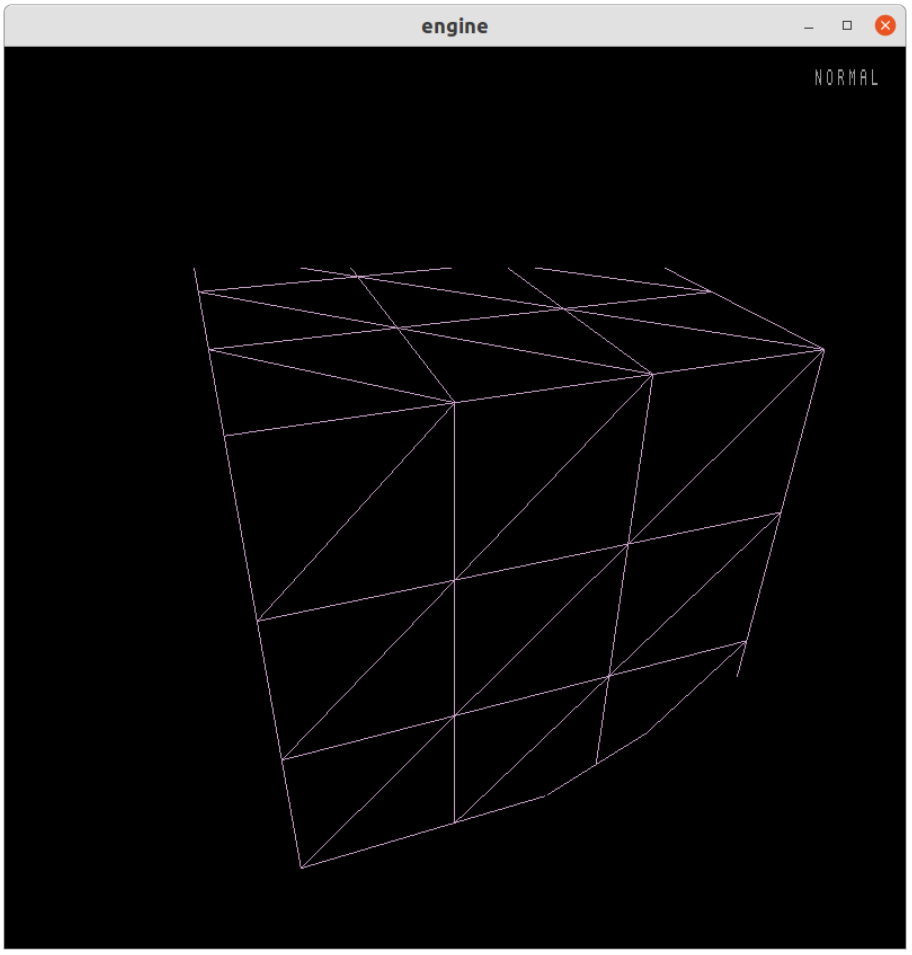
`test_files/test_1_2.xml`



`test_files/test_1_3.xml`



`test_files/test_1_4.xml`



`test_files/test_1_5.xml`