Computação Gráfica: Etapa#1

Bernardo Rodrigues
a99999@alunos.uminho.pt

César Silva
a77518@alunos.uminho.pt

Pedro Faria
a82725@alunos.uminho.pt

Rui Silva
a99999@alunos.uminho.pt

Universidade do Minho — 25 de Fevereiro de 2019

Resumo

A **Computação Gráfica**, uma das muitas áreas da informática, que assume um papel fulcral em interações homem-máquina e na visualização de dados. o seu espectro de aplicações varia desde geração de imagens até a simulação do mundo real. O presente relatório refere-se à primeira fase de entrega da componente prática da unidade curricular (UC) de **Computação Gráfica** enquadrada no curso de Ciências da Computação da Universidade do Minho.

Conteúdo

1 Introdução

Nesta fase, foram desenvolvidas duas aplicações. Um gerador de modelos, que aceite argumentos a partir do terminal, que nesta fase deveria escrever para um documento os pontos de uma primitiva gráfica desejada pelo utilizador. E a última, um motor capaz de gerar os modelos a partir de um ficheiro de configuração. Com base nos tópicos enunciados nas aulas teóricas e ferramentas exploradas nas aulas práticas. Implementamos o gerador e o motor na linguagem **C++[]**, em particular esta última faz uso de biblioteca **TinyXML2** para que a leitura dos documentos que lhe são dados com input seja feita de forma simples e consistente. E por fim, utilizamos a API fornecida pelo **OpenGl** para dar vida aos nossos modelos e confirmar a consistência dos nossos programas.

1.1 Análise do Enunciado

Talvez remover esta ainda vamos ver.

1.2 Estrutura do Relatório

bla bla

2 Gerador

2.1 Caixa

2.2 Cone

2.3 Esfera

2.4 Plano

3 Motor

O motor é a parte do nosso trabalho que faz o linking de todas as outras partes que foram realizadas. O motor, lê um ficheiro xml (conf.xml). Este ficheiro contém uma estrutura bastante simples, dividida em scenes.

conf.xml

```
<scene>
        <model file="esfera.txt" />
        <model file="cone.txt" />
        <model file="caixa.txt" />
</scene>
```

Cada ficheiro que está referenciado nas tags **<model>** é um ficheiro de texto criado pelo nosso gerador e contém todos os pontos das figuras que queremos representar. Como o motor lê os ficheiros de pontos a partir dum ficheiro **XML** utilizamos um parser xml para **C++** chamado **tinyxml-2**.

main.cpp

```
...
tinyxml2::XMLDocument doc;

doc.LoadFile("./conf.xml");

tinyxml2::XMLNode *scene = doc.FirstChild();

tinyxml2::XMLElement* model;

while(scene) {
        for(model = scene->FirstChildElement();
        model != NULL;
        model = model->NextSiblingElement()) {
                const char * file;
                file = model->Attribute("file");
                guardaPontos(file);
        }
        scene = scene->NextSiblingElement();
}
...
```

Com este snippet de código, criamos um objeto do tipo XMLDocument. De seguida, com a função **LoadFile**, abrimos o ficheiro de configuração **conf.xml**, e começamos a manipular o seu conteúdo. Criamos um objeto do tipo **XMLNode** e associamos-lhe a primeira **tag** do ficheiro conf.xml que é a raiz da estrutura do nosso ficheiro. No ciclo while, percorremos todos o ChildElements de scene, que são as tags que guardam os nossos ficheiros das figuras. Cada ficheiro de figura tirado dos **models** é passado à função **guardaPontos**, que será explicada de seguida.

main.cpp

```
...
struct Pontos {
    float a;
    float b;
    float c;
};

std::vector<Pontos> pontos;

void guardaPontos(std::string ficheiro) {
        std::ifstream file;
        std::string s = "./";
        s.append(ficheiro.c_str());
        file.open(s.c_str());
        float a,b,c;
        while(file >> a >> b >> c) {
                Pontos aux;
                aux.a = a;
                aux.b = b;
                aux.c = c;
                pontos.push_back(aux);
        }
}
...
```

Criamos uma estrutura **Pontos** que tem como campos 3 *floats*, que servem para registar as coordenadas destes. De seguida usamos um vector que utiliza a estrutura enunciada para os guardar. À função **guardaPontos** são passados nomes de ficheiros. A função abre os ficheiros e lê pontos linha a linha, guardando cada coordenada *x, y e z* num **Ponto** e de seguida inserindo-o no vector.

Info: A instrução:

```
file >> a >> b >> c
```

associa a cada uma das variaveis *a, b* e *c*, as coordenadas da linha que está a ser lida em cada iteração do ciclo.

Por ultimo temos a função **printPontos**:

main.cpp

```
...
void printPontos(std::vector<Pontos> pontos) {
  for(int i = 0; i < pontos.size(); i++) {
                glVertex3f(pontos[i].a,
                        pontos[i].b,
                        pontos[i].c);
        }
}
...
```

Esta função é chamada na renderScene e geras todos os pontos percorrendo o vector.

4 Conclusão

De um modo geral achamos que grupo, apesar das dificuldades, correspondeu às exigências propostas do enunciado. Tentamos implementar uma solução que suportasse requisitos futuros.

A Coisas que se pode fazer com este template

Question 1

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

(a) Do this.

(b) Do that.

(c) Do something else.

Algorithm 1: `FastTwoSum`

Input: (a, b), two floating-point numbers
Result: (c, d), such that $a + b = c + d$

if $|b| > |a|$ **then**
  exchange a and b ;
end
$c \leftarrow a + b$;
$z \leftarrow c - a$;
$d \leftarrow b - z$;
return (c, d) ;

Question 2 *(with optional title)*

In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.

$$I = \int_a^b f(x)\, \mathrm{d}x. \tag{1}$$

Info: This is an interesting piece of information, to which the reader should pay special attention. Fusce varius orci ac magna dapibus porttitor. In tempor leo a neque bibendum sollicitudin. Nulla pretium fermentum nisi, eget sodales magna facilisis eu. Praesent aliquet nulla ut bibendum lacinia. Donec vel mauris vulputate, commodo ligula ut, egestas orci. Suspendisse commodo odio sed hendrerit lobortis. Donec finibus eros erat, vel ornare enim mattis et.

hello.py

```
#! /usr/bin/python

import sys
sys.stdout.write("Hello World!\n")
```

Command Line

```
$ chmod +x hello.py
$ ./hello.py

Hello World!
```

Notice: In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.