TP2-CG-MIEI

João Bernardo Freitas a74814

23 Março 2020

Contents

1 Introdução

Neste relatório iremos apresentar e discutir o trabalho realizado pelo grupo, no âmbito da Unidade Curricular de Computação Gráfica, do 3º ano do Mestrado Integrado em Engenharia Informática.
Nesta fase tivemos como objectivo gerar um sistema solar estático através de modelos criados pelo gerador que foi desenvolvido na 1ª fase aplicando transformações geométricas, entre as quais, translações, rotações e escalas.
O trabalho do grupo focou-se na adaptação do código desenvolvido na 1ª fase para responder a este novo paradigma.

2 Parser

Tendo em conta os novos requisitos, nomeadamente as transformações geométricas, é necessário adaptar o código desenvolvido na 1ª fase para ler as transformações geométricas e aplica-las aos modelos, sendo que estas transformações poderiam estar no próprio grupo ou no grupo "pai".
Visto que é necessário guardar um grande número de dados, referentes ao tipo de transformação e o ficheiro onde as transformações serão aplicadas, decidimos implementamar as seguintes estruturas de dados.

```
typedef struct grupo {
    char* transf;
    double params[4];
    struct grupo* next;
} Grupo;

typedef struct transforma {
    char* transf;
    double params[4];
} Transforma;

typedef struct ficheiros {
    char* file;
    vector<Transforma*> transfs;
} Ficheiros;
```

Figure 1: Estruturas de dados

NOTA: Os dados irão ficar guardados num vetor de estruturas Ficheiros ***vector<Ficheiros*> ficheiros****.*

2.1 Ficheiros

Esta estrutura irá conter o nome do ficheiro bem como todas as transformações a aplicar a esse ficheiro, sendo que essas transformações têm a sua própria estrutura.

2.2 Transforma

Nesta estrutura são guardadas todas as transformações aplicadas a um certo modelo.
O campo **tranfs** contém o nome da transformação, *scale*,*translate* ou *rotate*, bem como os seus parâmetros que dependendo do tipo de transformação irá necessitar de 3 (X,Y,Z) ou 4 (Angle,X,Y,Z) parâmetros.

2.3 Grupo

Esta estrutura auxiliar surgiu da necessidade de aplicar as transformações do grupo "pai" ao grupo filho logo temos os mesmos campos que a estrutura *Transforma* e um apontador para a próxima transformação.

2.4 Exemplo

Neste capitulo segue um pequeno exemplo de como é que as transformações são lidas para as estruturas.

```
<group>
    <translate Z="120" />
    <scale X="8" Y="8" Z="8"/>
    <colour R="1" G="0.55" B="0"/>
    <models>
        <model file="sphere.3d"/>  --venus
    </models>
</group>
```

Figure 2: Exemplo da utilização da tag

Assumindo que é encontrada a tag "translate" no ficheiro **config.xml**, guardamos todas as informações relevantes, neste caso os valores X,Y e Z, nas respectivas variáveis que são enviadas para a função **muda**, que é responsável por inserir as variáveis na estrutura de dados *Grupo*. Quando é encontrada a tag **file** sabemos que não existem mais transformações, por isso copiamos o conteúdo da estrutura *Grupo* para a estrutura *Transforma*.

```
if (strcmp((char*)group->Value(), "translate") == 0) {
    char translate[15];
    strcpy(translate, (char*)group->Value());
    transX = (char*)group->Attribute("X");
    transY = (char*)group->Attribute("Y");
    transZ = (char*)group->Attribute("Z");
    x = 0;
    y = 0;
    z = 0;
    if (transX != NULL) x = atof(transX);
    if (transY != NULL) y = atof(transY);
    if (transZ != NULL) z = atof(transZ);
    muda(g, i, translate, x, y, z, 0);
}
```

Figure 3: Parser

Para testar a partilha de transformações geométricas entre grupos decidimos criar o planeta Terra e Marte com os respetivos satélites, *Lua, Deimos e Phobos.*

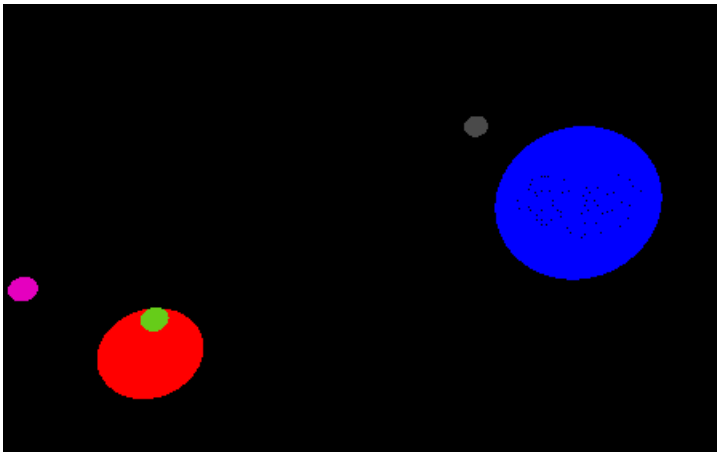

Figure 4: Planeta Terra e Marte

Após os testes podemos ver que o resultado foi bastante aceitável.

3 Extração das coordenadas

A extração das coordenadas é feita através da função *ReadFile* que é semelhante ao da fase anterior, que remove as coordenadas do ficheiro.3d e insere na estrutura **Point**.

Para sabermos quando um modelo acaba e começa outro foi criado um triângulo imaginário com 3 pontos com as coordenadas (2020,2020,2020), sendo que este triângulo não irá ser desenhado.
Com todas as coordenadas inseridas na estrutura de dados passamos á ultima fase, o desenho dos modelos.

4 Desenho do modelo

Como foi mencionado no tópico anterior, temos todas as coordenadas num vetor, logo é necessário agrupar as coordenadas em grupos de 3 de forma a criar triângulos, criando uma figura em 3 dimensões.

Porém, antes de desenhar estes triângulos, temos de aplicar todas as transformações guardadas no parser. Assim, percorremos o vector onde estão guardadas as transformações geométricas de um modelo e, consoante o nome da mesma, aplica-se a função correspondente predefinida do OpenGL.
Essas transformações são as previamente mencionadas **rotate,translate,scale** com a adição de **colour** que adicionamos para poder dar cor aos modelos.
Estas operações são salvaguardadas pelas funções *glPushMatrix()* e*glPopMatrix()*, de forma a evitar que as transformações sejam aplicadas a todos os modelos seguintes.

```
while (iPonto != vertices.end()){
    f = true;
    Ficheiros* fich = *iFicheiros;

    iTransforma = fich->transfs.begin();

    glPushMatrix();
    while (iTransforma != fich->transfs.end()) {
        Transforma* trans = *iTransforma;
        if (strcmp((trans->transf), "scale") == 0) glScalef(trans->params[0], trans->params[1], trans->params[2]);
        if (strcmp((trans->transf), "rotate") == 0) glRotatef(trans->params[3], trans->params[0], trans->params[1], trans->params[2]);
        if (strcmp((trans->transf), "translate") == 0)  glTranslatef(trans->params[0], trans->params[1], trans->params[2]);
        if (strcmp((trans->transf), "colour") == 0) glColor3f(trans->params[0], trans->params[1], trans->params[2]);
        iTransforma++;
    }
    while (iPonto != vertices.end() && f != false) {
        Point prim = *iPonto;
        iPonto++;
        Point sec = *iPonto;
        iPonto++;
        Point tre = *iPonto;
        iPonto++;
        if (!ver(prim, sec, tre)) {
            glBegin(GL_TRIANGLES);
            glVertex3f(prim.x, prim.y, prim.z);
            glVertex3f(sec.x, sec.y, sec.z);
            glVertex3f(tre.x, tre.y, tre.z);
            glEnd();
        }
        else {
            f = false;
        }
    }
```

Figure 5: drawVertices

NOTA: A função ver é a função que verifica se o modelo chegou ao fim ou não

5 Resultado final

Por fim temos a representação do sistema solar, contendo o sol e os 8 planetas mais as luas da Terra e de Marte. Como é óbvio não está desenhado á escala.

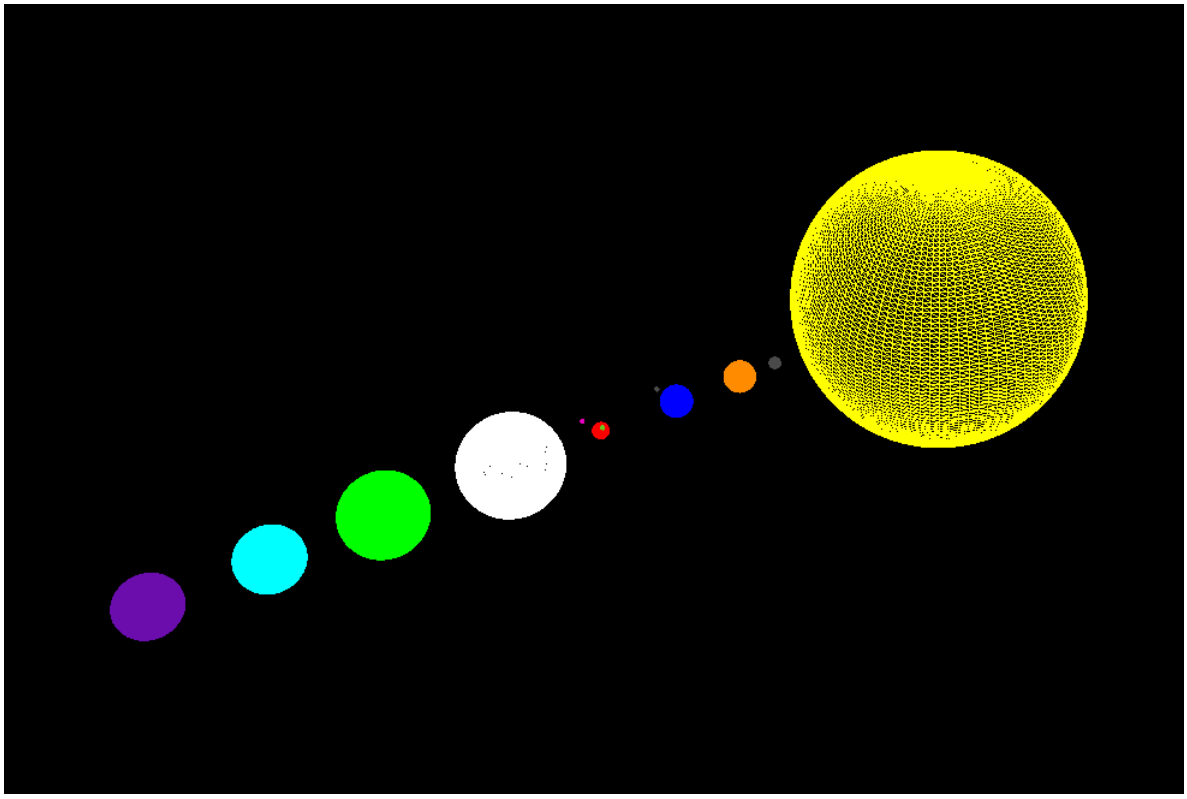

Figure 6: Sistema solar

6 Conclusão

Esta segunda fase do trabalho prático de Computação Gráfica foi bastante mais complexa que a primeira, devido principalmente às mudanças ao Parser do XML para poder ler as transformações. Sendo que guardar os dados nas respectivas estruturas foi o maior desafio desta fase.
Pondo isto, podemos dizer que a nossa prática em C++ foi um pouco melhorada com este projeto até então, bem como os conhecimentos em OpenGL e que damos o objetivo principal como cumprido.