Universidade do Minho
Mestrado Integrado em Engenharia Informática

Computação Gráfica

Practical Assignment CG - 2019/20
Phase 2: Geometric Transforms

Março 2020


Angélica Freitas
(A83761)


Joana Afonso Gomes
(A84912)


Rodrigo Pimentel
(A83765)

Conteúdo

1 Introdução

No âmbito da UC de Computação Gráfica foi-nos proposto o desenvolvimento de um cenário com gráficos 3D. O objetivo da segunda fase deste trabalho prático é o desenvolvimento de cenas hierárquicas, que pode ser divido essencialmente em duas partes:

- A primeira consiste no aperfeiçoar do programa que permite a leitura de ficheiros XML, para que seja conseguido um processamento correto e ideal dos dados;
- A segunda fase caracteriza-se pela elaboração de um ficheiro a partir do qual será gerado o sistema solar, sendo que o programa deverá gerar o modelo sugerido.

No presente relatório incluímos a apresentação pormenorizada das várias partes acima apresentadas, com o apoio de equações, gráficos, esquemas e figuras para auxiliar a compreensão do processo de elaboração.

2 Notas Importantes

Antes de começar a correr código, é de extrema importância ler e acompanhar as instruções para melhor aproveitamento do trabalho (*README*).

Os ficheiros SolarSystem.xml e sol.3d encontram-se na diretoria Generator/Debug/.

3 Engine

De modo a facilitar a leitura de ficheiros XML com hierarquia, decidimos criar as classe ***Model***, ***TransformationState*** e ***Models***.

3.1 Classe TransformationState

Como há a possibilidade de um grupo conter um ou mais grupos, as transformações geométricas realizadas no grupo atual são passadas para os grupos nele contido (hierarquia). Assim, para saber quais as transformações que teremos de realizar, criamos uma classe *Transformation state*, como referido anteriormente.

Assim, sempre que são lidas as informações das transformações geométricas de um grupo, estas vão ser adicionadas às transformações atuais, sendo previamente inicializadas:

```
class TransformationState{
public:
    std::vector < float > translation;
    std::vector < float > rotation;
    std::vector < float > scale;

    TransformationState() {
        translation = { 0,0,0 };
        rotation = { 0,0,0,0 };
        scale = { 1,1,1 };
    }
```

3.2 Classe Model

Para cada modelo, estabelecemos criar um classe *Model* em que a mesma continha os pontos para poder ser desenhada, assim como possíveis transformações geométricas relacionadas com o mesmo modelo, que se traduzem em:

```
class Model {
public:
    std::vector < std::vector<float>> points;
    std::vector < float > translation;
    std::vector < float > rotation;
    std::vector < float > scale;
```

3.3 Classe Models

Sendo possível a utilização de um mesmo ficheiro XML (contendo os pontos a gerar), decidimos criar a classe *Models* que, para cada ficheiro, terá uma lista de Model. A mesma possui também a lista *files*, para percorrermos os diferentes modelos através da posição. Resultando:

```
class Models {
public:
    std::unordered_map<std::string, std::vector<Model*>> data;
    std::vector<std::string> files;
    int pos;
    int len;
```

4 Leitura de ficheiro xml hierárquico

Para a leitura do ficheiro XML utilizamos a biblioteca *tinyxml2*. Assim, caso um ficheiro não exista ou o esteja mal formatado, é impresso do mesmo. Visto que cada grupo filho vai ter as mesmas transformações geométricas que o grupo pai, optamos por ler o ficheiro de maneira recursiva. Isto é, dentro de um grupo, verifica se existem transformações (guardando essas informações no *TransformationState*) e adicionando ao *map* de *Model* (*data* na classe *Models*) o *model file*. Caso este encontre um novo grupo, chama de novo a função com esse elemento xml, como se pode ver pela seta na figura abaixo:

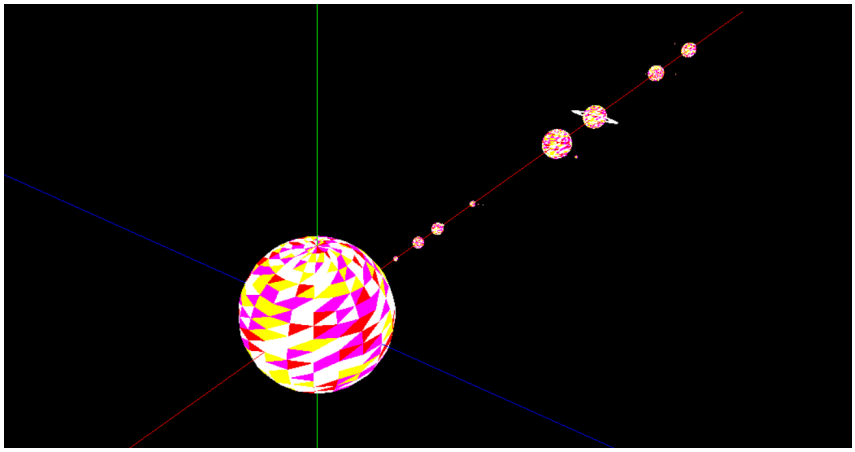
```
void auxReadFile(XMLElement * elem, TransformationState ts) {
    for (XMLElement* child = elem->FirstChildElement(); child != NULL; child = child->NextSiblingElement()) {

        if (strcmp(child->Name(), "translate") == 0) {
            ts.translate(child->FindAttribute("X") ? atof(child->FindAttribute("X")->Value()) + ts.translation[0] : ts.translation[0], chil

        }
        else if (strcmp(child->Name(), "rotate") == 0) {
            ts.rotate(child->FindAttribute("angle") ? atof(child->FindAttribute("angle")->Value()) + ts.rotation[0] : ts.rotation[0], child
        }
        else if (strcmp(child->Name(), "scale") == 0) {
            ts.setScale(child->FindAttribute("X") ? atof(child->FindAttribute("X")->Value())  : ts.scale[0], child->FindAttribute("Y") ? at
        }
        else if (strcmp(child->Name(), "models") == 0) {
            for (XMLElement* childModels = child->FirstChildElement(); childModels != NULL; childModels = childModels->NextSiblingElement()
                std::string filename = childModels->Attribute("file");
                models->addFile(filename, ts.translation, ts.rotation, ts.scale);

            }
        }
        else if (strcmp(child->Name(), "group") == 0) {
            auxReadFile(child, ts.clone());

        }

        else {
            std::cout << "What command is this? -> " << child->Name() << std::endl;
            exit(0);
        }
    }
}
```

5 Modelo Sistema Solar

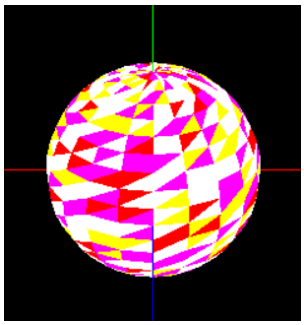
Tirando partido de existirem grupos intrínsecos a outros, recorremos ao mesmo ficheiro de esfera com 1 de raio para desenhar não só os planetas mas também as suas luas e o sol.

Para tal, recorremos inicialemnte a *ratios* reais para obter distâncias e tamanhos do sistema solar. Porém, depois de desonvolvermos a ferramenta de leitura de ficheiros xml hierárquicos e visualizarmos a cena, chegamos à conclusão que não podiamos seguir um modelo tão realista.

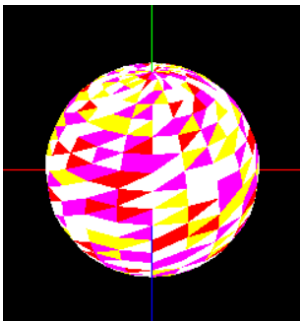
Procurando ter em conta os tamanhos e distâncias, obtemos a seguinte cena:



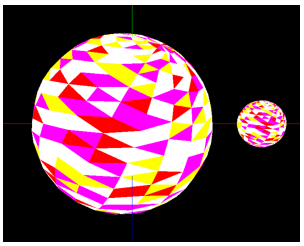
Sistema Solar



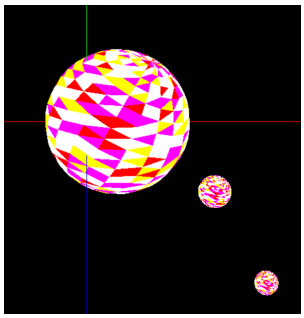
Mercúrio



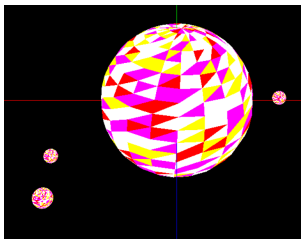
Vénus



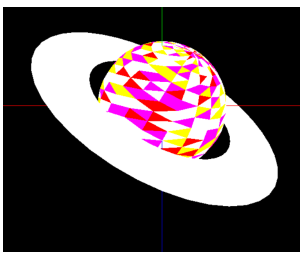
Terra
lua



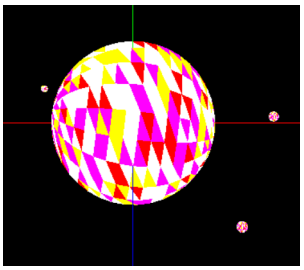
Marte
Phobos (meio)
Deimos (direita)



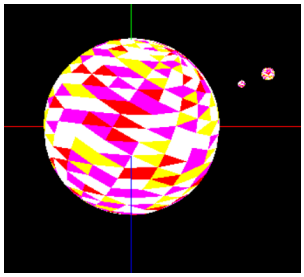
Júpiter
IO(direita)
Ganymede(esquerda)
Europa(esquerda,cima)



Saturno



Urano
Titânia(direita)
Oberon(frente)
Umbriel(esquerda)



Neptuno
Proteu(meio)
Triton(direita)