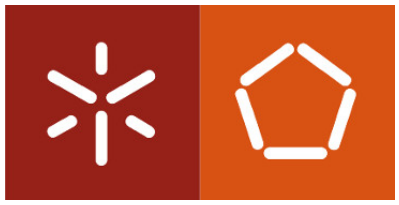

Universidade do Minho

Computação Gráfica

Fase 2 - Transformações Geométricas

Grupo 31

Mestrado Integrado em Engenharia Informática

A89466	A89511	A89550
		
João Amorim	Rodrigo Meira	João Sá

Março de 2020

Conteúdo

1 Introdução

Para a unidade currícular de Computação Gráfica foi nos proposto criar uma mini engine capaz de desenhar cenas com desenhos 3D que se encontra dividido em 4 fases.
Nesta segunda fase, foi proposta uma atualização relativa ao engine, que consiste na construção hierárquica de cenas utilizando transformações geométricas, tais como translação, rotação e escala. Iremos ao longo deste relatório, apresentar as decisões que o grupo tomou de forma a chegar à solução pretendida. Para além desta atualização relativamente ao engine, foi-nos também pedida a criação de uma cena do Sistema Solar.

2 Estrutura do Projeto

Testado em: Esta fase do trabalho prático de computação gráfica foi testada somente no sistema operativo windows.

Nesta fase foram efetuadas modificações à pasta engine, relativamente ao parser, pois foram pedidas informações novas e acrescentado um novo campo, mais especificamente o campo *transform*, que contém as transformações geométricas e a hierarquia já falada na introdução. Outra função que precisou de alterações foi a desenha pertencente a renderscene na main, pois além de ser necessário ler as transformações foi preciso aplicá-las, tendo em atenção a forma como estas são executadas, para isso acontecer foi necessário a utilização de pushs e pops de forma a que a aplicação das mesmas fosse bem efetuada. Para além destas alterações, modificamos também a função que gera o cone no programa generator, uma vez que nos apercebemos que o cone tinha de ser gerado com a base centrada na origem, e na versão anterior nós geravamos o cone centrado na origem.

Nota: A pasta toolkits é necessária para o gerador, mas não está contida no projeto uma vez que foram os docentes que a disponibilizaram.

Como biblioteca externa foi usado o *tinyxml2* (1) para *parsing* dos ficheiros xml.

3 Análise do XML

O objetivo desta fase foi essencialmente produzir um sistema que seja capaz de ler ficheiros xml e aplicar transformações geométricas de uma forma hierárquica de acordo com a semântica definida no enunciado.
Para obter uma solução, uma vez que as transformações geométricas de um grupo se mantêm pelos grupos "filhos" e só são desfeitas após sairmos desse grupo, decidimos realizar o *parsing* do ficheiro de forma recursiva.

A solução passa por criar uma função que se chame a si mesma, sempre que encontra um novo elemento "group", agindo de forma recursiva. Desta forma é fácil lidar com as transformações geométricas já que apenas temos de chamar a função push da matriz de transformações antes da chamada recursiva, e pop de seguida.

Pseudo-código:

```
Group readXMLGroup(XMLNode* group) {
        Group g = Group();
        XMLNode* next = group->FirstChild();
        while (next) {
                if (strcmp(next->Value(), "transform") == 0) {
                        g.addTransform(readXMLTransform(next));
                }
                else if (strcmp(next->Value(), "models") == 0) {
                        XMLElement* elem = next->
                        FirstChildElement("model");
                        while (elem) {
                                string s =
                                std :: string(elem->Attribute("file"));
                                g.addFile3d(s);
                                g.addVertices(lerFicheiro3d(s));
                                elem = elem->NextSiblingElement("model");
                        }
                }
                else if (strcmp(next->Value(), "group") == 0) {
                        g.addFilho(readXMLGroup(next));

                }
        next = next->NextSibling();
        }
        return g;
}
```

A função readXMLGroup é a função que permite a leitura das hierarquias especificadas no xml. Esta retorna um Group que não é nada mais nada menos que uma classe que tem como variáveis de instância um vetor de strings denominado como files3d que serve para guardar os ficheiros 3d que são gerados pelo generator e depois lidos na engine pela função lerficheiros3d. De seguida, temos também um vetor de transformações que armazena as transformações geométricas que serão aplicadas aos modelos, essas transformações são

adicionadas, através da função que addTransform que recebe como argumento a função readXMLTransform, que como nome indica faz a leitura do xml da parte das transformações.

Além disso, temos um vetor de vértices que guarda os vértices de um ficheiro. Como podemos ver na função acima demonstrada, o parâmetro g.addVertices(lerficheiro3d(s)) em que s é o nome do ficheiro 3d, à medida em que o ficheiro é lido os vértices desse mesmo ficheiro são adicionados a essse vetor de vértices, através da chamada do método que os adiciona.

Para finalizar, utilizamos um vetor de group denominado como filhos. E é aqui que entra a recursividade e que permite ter a noção de hierarquia pedida pelo corpo docente. Como podemos ver na imagem acima apresentada, o último else if verifica se o value do nodo de xml é um group, caso o seja e se tiver filhos então aplica a recursividade, como argumento da função addfilho, permitindo assim o armazenamento na variável filhos da classe group.
Variáveis de Instância da classe Group:

```
private:
    vector<string> files3d;
    vector<Transform> transformacoes;
    vector<Vertex> vertices;
    vector<Group> filhos;
```

Nota: Devido ao uso da biblioteca *tinyxml2* os campos dos elementos requerem o uso de aspas.

Exemplo:

Ainda a salientar, temos o exemplo de um ficheiro xml retirado da pasta de testes fornecida pelos docentes, que podia ser utilizado para esta fase, de forma a demonstrar aquilo que foi desenvolvido.

Ficheiro test_2_1.xml:

```
<world>

    <camera>
            <position x="10" y="3" z="10" />
            <lookAt x="0" y="0" z="0" />
            <up x="0" y="1" z="0" />
        <projection fov="60" near="1" far="1000" />
    </camera>

    <group>
        <transform>
                    <translate x="0" y="1" z="0" />
        </transform>
        <models>
            <model file="box.3d" />
        </models>
    </group>
</world>
```

3.1 Sistema Solar

Tal como pede o enunciado, o grupo desenvolviu um ficheiro xml no qual é modelado uma maquete do sistema solar com o Sol, os planetas, a suas órbitas e a lua (uma vez que o Sistema Solar possui 205 luas, optamos por desenhar apenas a da Terra). Os corpos celestes são desenhados como esferas após o uso de escalas para diferenciar os seus tamanhos. As órbitas são apenas circunfêrencias que os intersetam.

No que diz respeito às suas posições e tamanho na cena, tentamos fazer uma representação semelhante à real sublinhando que não foi definida nenhuma escala. O Sol é desenhado no centro da cena e os planetas são dispostos aplicando translações à origem, mediante a sua distância ao Sol. Apenas uma figura foge a esta regra e é aquela que representa a lua terrestre, pois para a desenhar aplicamos uma translação em relação ao referencial da Terra.

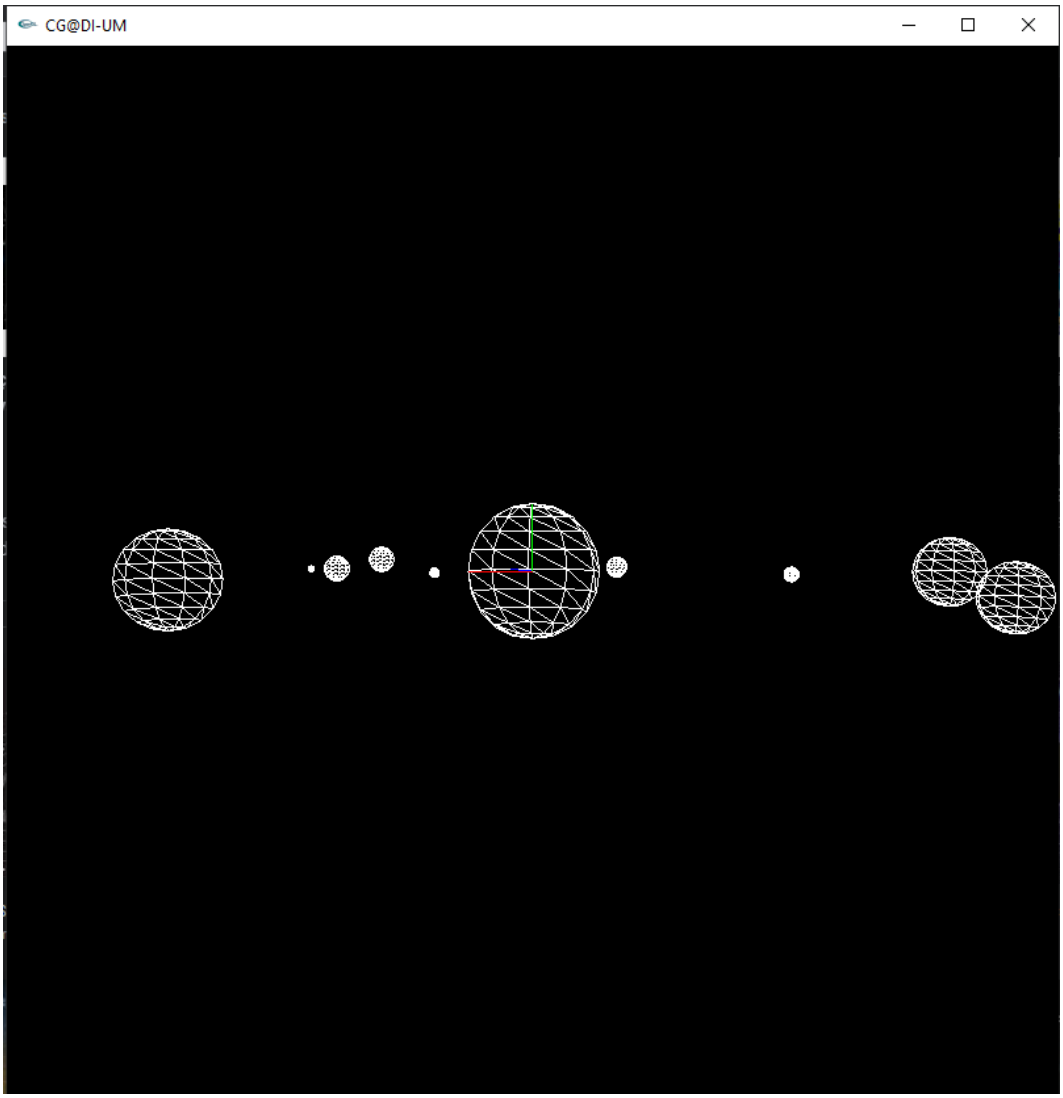

Figura 1: Modelo do sistema solar, a partir de um ficheiro xml, desenhado na engine.

4 Conclusão

Nesta 2ª Fase conseguimos realizar com sucesso as tarefas enunciadas, mas para além do pretendido pelo enunciado fizemos outra alteração no generator que foi na geração dos pontos que constituiem o cone. Com esta fase conseguimos consolidar melhor alguns conceitos abordados nas aulas ao nível da manipulação das transformações geométricas e da câmara.

Referências

[1] tinyxml2 - https://github.com/leethomason/tinyxml2.