

Computação Gráfica  
**Fase 2/4**  
*”Transformações geométricas”*

Hugo Sousa, A76257

Matias Capitão, A82726

27 de Março 2020

# Conteúdo

|          |                                 |           |
|----------|---------------------------------|-----------|
| <b>1</b> | <b>Introdução</b>               | <b>3</b>  |
| <b>2</b> | <b>Análise e Especificação</b>  | <b>4</b>  |
| 2.1      | Estruturas . . . . .            | 4         |
| 2.2      | TinyXML . . . . .               | 5         |
| <b>3</b> | <b>Processo de renderização</b> | <b>7</b>  |
| <b>4</b> | <b>Resultados</b>               | <b>8</b>  |
| <b>5</b> | <b>Bibliografia</b>             | <b>10</b> |
| <b>6</b> | <b>Conclusão</b>                | <b>11</b> |

# Lista de Figuras

|     |   |   |
|-----|---|---|
| 2.1 | Ficheiro XML exemplo . . . . .                                    | 5 |
| 4.1 | Ficheiro XML para desenhar uma box sem transformações . . . . .   | 8 |
| 4.2 | Resultado de utilizar o ficheiro XML na figura anterior . . . . . | 8 |
| 4.3 | Ficheiro XML para desenhar uma box transformada . . . . .         | 9 |
| 4.4 | Resultado de utilizar o ficheiro XML na figura anterior . . . . . | 9 |
| 4.5 | Sistema Solar . . . . .   | 9 |

# Capítulo 1

## Introdução

A segunda fase do trabalho vai consistir em aplicar as transformações às figuras criadas na primeira fase de modo a construir o sistema solar. Estas vão ser aplicadas depois de ler um ficheiro xml, que contém as informações necessárias, com recurso á ferramenta "tinymce".

## Capítulo 2

# Análise e Especificação

Em relação á primeira fase do trabalho, as alterações a ser feitas são no Motor do programa, a começar pelo parsing do ficheiro com a ferramenta "tinysql".

### 2.1 Estruturas

Estrutura que representa um ponto:

```
struct Point
{
    float x;
    float y;
    float z;
};
```

Estrutura que representa o tipo de uma transformação geométrica:

```
enum type
{
    ROTATE,
    TRANSLATE,
    SCALE,
};
```

Estrutura que representa uma transformação geométrica:

```
struct gt
{
    struct Point p;
    enum type gt_type;
    float r_angle;
};
```

Estrutura que representa um grupo:

```
struct group
{
    vector<struct gt> gt;
    vector<vector<struct Point>> models;
    vector<struct group> child;
};
```

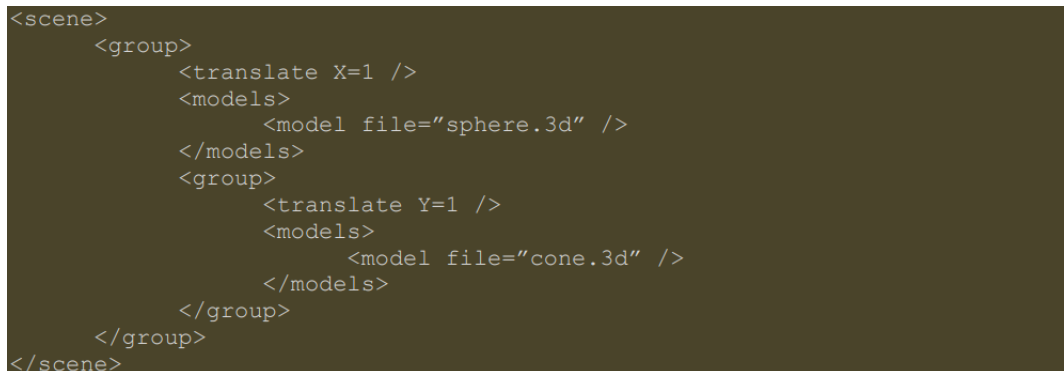
Estrutura que representa uma scene:

```
struct scene
{
    vector<struct group> groups;
    int nModels;
};
```

## 2.2 TinyXML

Esta ferramenta irá permitir fazer o parsing do ficheiro XML que é recebido como argumento, e guardá-lo nas estruturas internas do programa.

Basicamente, vamos detetar a presença de "group", "models", "model file", "translate", "rotate" e "scale", ao oposto da primeira fase que só era necessário encontrar "model file".



```
<scene>
  <group>
    <translate X=1 />
    <models>
      <model file="sphere.3d" />
    </models>
  </group>
  <group>
    <translate Y=1 />
    <models>
      <model file="cone.3d" />
    </models>
  </group>
</scene>
```

Figura 2.1: Ficheiro XML exemplo

Começamos por definir uma estrutura global chamada "scene" que irá guardar um vetor de estruturas group e também irá guardar o número total de modelos lidos do ficheiro .xml.

Depois de aberto o ficheiro, a sua raiz será, naturalmente "scene", e para cada tag "group" encontrada é chamada a função process\_groups() que funciona da seguinte forma:

- É criado uma estrutura group que vai guardar os dados de um determinado grupo.
- Caso o grupo esteja vazio a função retorna os dados lidos até agora.
- Se este grupo tiver grupos filhos a função é chamada recursivamente para esse filho.
- Caso seja encontrada a tag "translate", é inicializada uma struct gt com a função init\_translate(), que inicializa a translação no ponto(0,0,0) e define o seu tipo como TRANSLATE, e de seguida é

alterada conforme os dados recebidos do ficheiro, por fim, esta transformação é adicionada ao vetor de transformações da estrutura group.

- Caso seja "rotate", é usado o mesmo processo indicado no ponto anterior, mas agora inicializando a estrutura com a função `init_rotate()` que inicia o eixo de rotação como o  $(0,0,0)$ , o ângulo de rotação a  $0^\circ$  e define o seu tipo como ROTATE.
- Caso seja "scale", o processo é o mesmo mas inicializando a transformação geometrica com `init_scale()` e definimos o "type" como SCALE.

## Capítulo 3

# Processo de renderização

A renderização é feita a partir de uma chamada à função `draw_scene()`, que recebe como parâmetro a lista dos grupos da "scene", e desenha esses mesmos grupos da seguinte forma:

- `glPushMatrix()` – Duplica a matriz `modelView` atual e adiciona-a ao topo da stack de matrizes que representam as transformações geométricas que vão ser aplicadas aos modelos.
- De seguida, é chamada a função `draw_gt()` que recebe como parâmetro o grupo atual que está a ser desenhado, e que desenha todas as transformações geométricas associadas a esse mesmo grupo
- Neste ponto, é chamada a função `draw_scene` recursivamente para os filhos deste grupo.
- Por fim é chamada uma função `draw_vbo()` que vai desenhar os modelos como vbo's previamente criados. (Aqui também temos a possibilidade de desenhar os modelos pelo método "`glBegin glVertex3f ... glEnd`" basta alterar a função `draw_vbo` pela função `draw_models()` que recebe como parâmetro o grupo atual.
- Por fim, para que as transformações geométricas deste grupo não sejam aplicadas a nenhum outro grupo a ser desenhado usa-se a função `glPopMatrix()`, que elimina a matriz que se encontra no topo da stack de matrizes.

```
void draw_scene(vector<struct group> groups)
{
    for (int i = 0; i < groups.size(); i++)
    {
        struct group group = groups[i];
        glPushMatrix();
        {
            draw_gt(group);
            draw_scene(group.child);
            draw_vbo();
            //draw_models(group);
        }
        glPopMatrix();
    }
}
```



## Capítulo 4

# Resultados

```
<scene>  
  <group>  
    <models>  
      <model file="box.3d" />  
    </models>  
  </group>  
</scene>
```

Figura 4.1: Ficheiro XML para desenhar uma box sem transformações

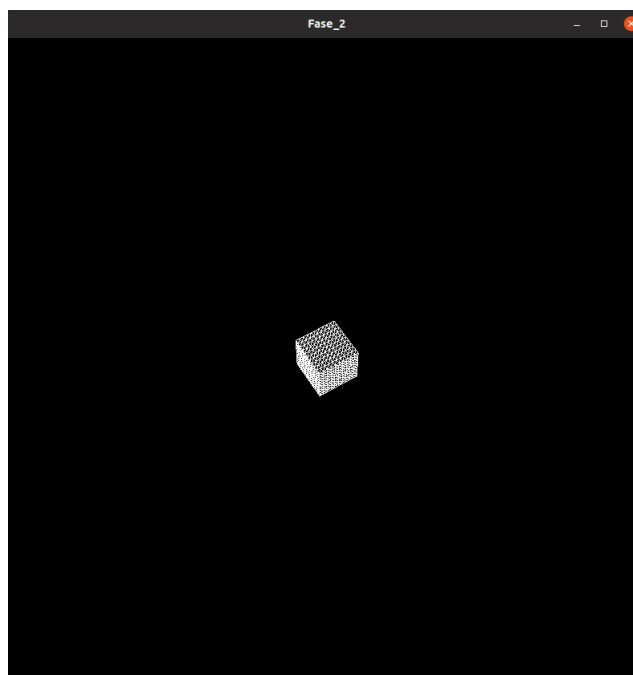


Figura 4.2: Resultado de utilizar o ficheiro XML na figura anterior

```

<scene>
  <group>
    <translate Z="10" />
    <scale X="2" Y="2" Z="2" />
    <models>
      <model file="box.3d" />
    </models>
  </group>
</scene>

```

Figura 4.3: Ficheiro XML para desenhar uma box transformada

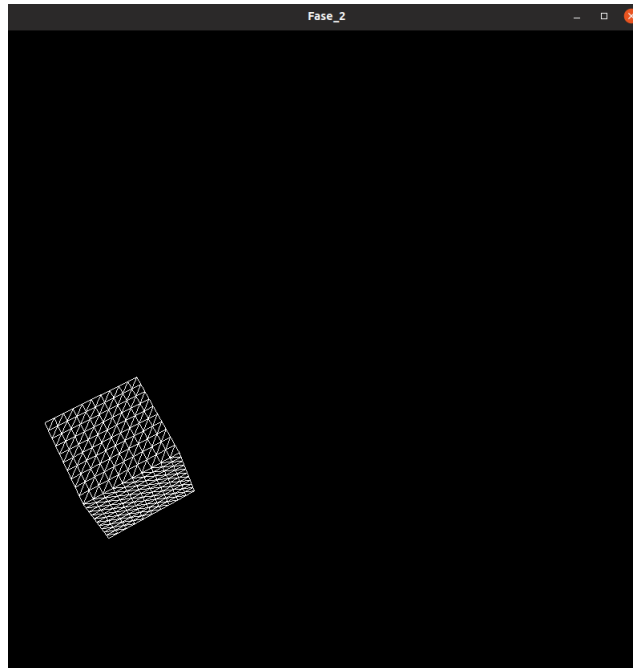


Figura 4.4: Resultado de utilizar o ficheiro XML na figura anterior

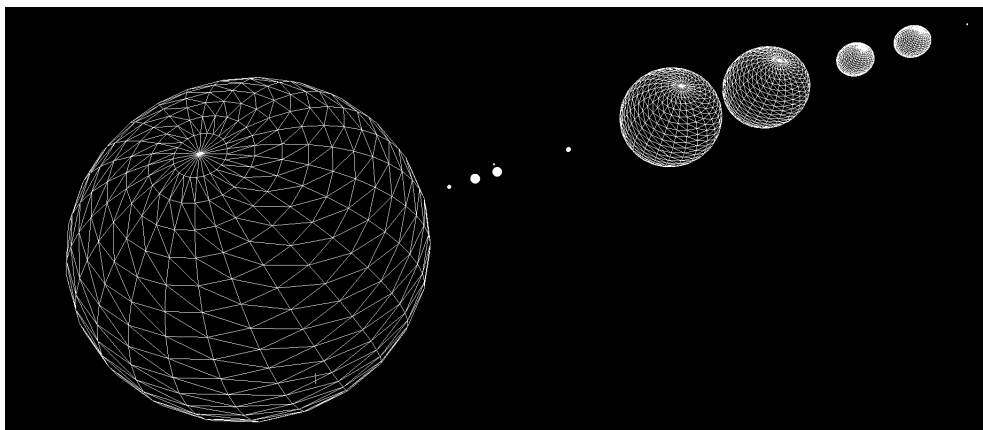


Figura 4.5: Sistema Solar

## Capítulo 5

# Bibliografia

<http://www.grinninglizard.com/tinyxml/>

## Capítulo 6

# Conclusão

Com a elaboração desta fase do projeto concluímos a importância da organização de scenes em hierarquia de groups. Algo que acabamos por incluir no trabalho, já a pensar nas próximas fases do trabalho, foram VBO's, visto que, traz melhorias significativas de performance. Em suma, acreditamos que fomos de encontro ao resultado esperado.