Computação Gráfica (3º ano de LCC)

Trabalho Prático (Fase 2) — Grupo 1

Relatório de Desenvolvimento

Bruno Dias da Gião (A96544) Bruno Miguel Fernandes Araújo (A97509)
João Luís da Cruz Pereira (A95375)

5 de abril de 2024

Conteúdo

1 Introdução **2**

1.1 Estrutura do Relatório 2

2 *Engine* **2**

2.1 Classes de Transformações 2

2.2 Leitura do ficheiro de configuração 3

2.2.1 Agrupamento do Desenho de modelos e Aplicação de transformações 3

3 Desenvolvimento da Demo **4**

4 Conclusão **5**

4.1 Trabalho Futuro 5

5 Resultados **6**

Lista de Figuras

1 Teste 1 6

2 Teste 2 7

3 Teste 3 7

4 Teste 4 8

5 Sistema Solar 8

1 Introdução

Este relatório serve como um apoio ao trabalho ,tem explicações sobre os raciocínios usados na segunda fase do projeto da unidade curricular Computação Gráfica.

Nesta fase apenas o engine foi modificado, sendo que foi necessário implementar o uso de transfomações sobre as figuras.

Tivemos de desenvolver um ficheiro *XML* que através da leitura pelo engine , representará o Sistema Solar (ficheiro Demo).

1.1 Estrutura do Relatório

- §2 - Nesta fase, passaremos a detalhar as classes de transformações geométricas(2.1), a leitura do ficheiro XML(2.2) e as gerações agrupadas de transformações e modelos(2.2.1).
- §3 - Explicação do desenvolvimento do ficheiro XML Demo.
- §4 - Conclusão e trabalho futuro.
- §5 - Imagens tiradas dos modelos gerados a partir dos ficheiros *XML* dos testes e da demo do Sistema Solar.

2 *Engine*

Passaremos a explicar o trabalho relativo ao **Motor de Gráficos** concebido na fase anterior:

2.1 Classes de Transformações

Tirando proveito do principio de polimorfismo e herança de Programação Orientada a Objetos temos uma superclasse ***transform***. que encapsula as classes ***rotate***, ***scale***,***translate***, identificadas por um selecionador do tipo, para debug, com valores `TRANS_ROT`, `TRANS_SCA` e `TRANS_TRA`, respetivamente.
Ora, cada subclasse contém variáveis privadas de transformação, construtores e um método que se sobrepõe a um método da superclasse que invoca as funções respetivas de transformação da API do OpenGL, como podemos ver nas seguintes definições:

```
1  transform::transform(int t,float xx, float yy, float zz)
2  {this->a=0.f,this->type = t,this->x=xx,this->y=yy,this->z=zz;}
3  transform::transform(int t,float aa,float xx, float yy, float zz)
4  {this->a=aa,this->type = t,this->x=xx,this->y=yy,this->z=zz;}
5  void transform::do_transformation(){}
6  int transform::get_type(){return this->type;}
7  float transform::get_angle(){return this->a;}
8  float transform::get_x(){return this->x;}
9  float transform::get_y(){return this->y;}
10 float transform::get_z(){return this->z;}
11
12 rotate::rotate(float a,float xx, float yy, float zz)
13   : transform(TRANS_ROT, a, xx, yy, zz){}
14 void rotate::do_transformation()
15 {glRotatef(this->get_angle(),this->get_x(),this->get_y(),this->get_z());}
16
17 scale::scale(float xx, float yy, float zz)
18   :transform(TRANS_SCA, xx, yy, zz){}
19 void scale::do_transformation()
20 {glScalef(this->get_x(),this->get_y(),this->get_z());}
21
22 translate::translate(float xx, float yy, float zz)
```

```
  : transform(TRANS_TRA, xx, yy, zz){}
void translate::do_transformation()
{glTranslatef(this->get_x(),this->get_y(),this->get_z());}
```

2.2 Leitura do ficheiro de configuração

A estrutura do ficheiro de configuração é uma versão estendida do ficheiro de configuração da primeira fase, tendo adicionado a interpretação de elementos `transform` que pode conter $n \in \mathbb{N}$ transformações de tipo:

$$t \in \{\texttt{TRANS\_ROT}, \texttt{TRANS\_SCA}, \texttt{TRANS\_TRA}\}$$

. Sendo assim, a sua leitura, ao contrário da primeira fase, passa a ser feita de forma recursiva seguindo a seguinte implementação:

```
void group_read(int cur_parent, int cur_g, XMLElement*gr,
                bool reading = false,
                int i=0)
{
  XMLElement*elem = !reading ? gr->FirstChildElement():
    gr->NextSiblingElement();
  if (cur_g == -1 || !elem)
    return;
    if (cur_parent > -1 && !reading) {
    for (int i=0; i<world.transformations.size(); i++) {
            struct trans copia;
            if (world.transformations[i].group == cur_parent) {
                copia.group = cur_g;
                copia.t = world.transformations[i].t;
                world.transformations.push_back(copia);
            }
        }
  }
  if (cur_g >= global)
    global = cur_g+1;
  if (!strcmp(elem->Name(),"models"))
    group_read_models(cur_parent, cur_g, elem);
  else if (!strcmp(elem->Name(), "transform"))
    group_read_transform(cur_parent, cur_g, elem);
  else if (!strcmp(elem->Name(), "group"))
    group_read(cur_g, global, elem,  false, i);
  group_read(cur_parent, cur_g, elem, true, i);
}
```

Todas as outras funcões invocadas por esta seguem o mesmo padrão recursivo de guardar o grupo "pai" e o grupo atual nos argumentos e "iterar" sobre o grupo sabendo essa informação, na sua essência, sabendo se a "iteração" atual é a primeira ou não procuramos o primeiro filho ou o próximo irmão, respetivamente, acabando quando este proximo elemento for nulo ou chegarmos a uma chamada recursiva onde a raiz em si do ficheiro XML é o grupo atual.

2.2.1 Agrupamento do Desenho de modelos e Aplicação de transformações

De notar o segmento de código que itera sobre `world.transformations` quando o pai atual não é a raiz do ficheiro de configuração, este permite uma funcionalidade de extrema importância: a herança de transformação dos grupos pais para os seus filhos.
Já o desenho das figuras tem uma pequena alteração, isto pois, com transformações o desenho de figuras terá de ser segregado em grupos, sendo assim, iterando sobre o número de grupos, que é computado de acordo com o maior valor atual do grupo na leitura recursiva, conseguimos comparar o nome associado a um conjunto de coordenadas com o

nome associado à primitiva em si, o que permite verificar se pertence ao grupo a ser trabalhado, como podemos ver no seguinte segmento de código:

```
void drawfigs(void)
{
  int i, j, k, l, g;
    for (g=0; g<global; g++) {
    glPushMatrix();
    for (l=0;l<world.transformations.size();l++) /* trans*/
      if (world.transformations[l].group == g) {
        world.transformations[l].t->do_transformation();
            }
        for (k=0; k<world.primitives.size(); k++) {
            if (world.primitives[k].group == g) {
                for (i = 0; i<prims.size(); i++) {
                    if (!strcmp(prims[i].name, world.primitives[k].name)) {
                    glBegin(GL_TRIANGLES);
                        for (j=0; j<prims[i].coords.size();j++) {
                            glVertex3f(prims[i].coords[j].x,
                                    prims[i].coords[j].y,
                                    prims[i].coords[j].z);
                        }
                    glEnd();
                    }
                }
            }
        }
    glPopMatrix();
  }
}
```

De notar a forma como o código é completamente agnóstico ao tipo de transformações e ao tipo de primitiva, verificando exclusivamente se pertence ao grupo estudado e quais sãos os valores das coordenadas.

3 Desenvolvimento da Demo

Para a representação do sistema solar, decidimos colocar cada planeta num grupo , cujo este terá como subgrupos as suas luas ou no caso de Júpiter, também o seu anel.

Usamos o mesmo ficheiro modelo da esfera para representar os planetas e as luas de forma a evitar ter de gerar e de ler diferentes ficheiros, assim apenas é necessário gerar um que será lido uma vez no engine.

Demos uso da figura extra que desenvolvemos na fase anterior , o torus, para criarmos o anel de Júpiter.

Para as diferenças de tamanho e de distância ao Sol dos planetas, usamos as transformações scale e translate , respetivamente.

Veremos então como exemplo o caso da Terra com a Lua:

```
<group>
    <transform>
        <translate x="80" y="0" z="80" />
        <scale x="0.4" y="0.4" z="0.4" />
    </transform>
            <models>
                <model file="sphere_10_20_20.3d" /> <!-- Terra -->
            </models>
            <group>
                <transform>
                    <translate x="24" y="24" z="24" />
```

```
12                          <scale x="0.1" y="0.1" z="0.1" />
13                    </transform>
14                          <models>
15                                <model file="sphere_10_20_20.3d" /> <!-- Lua -->
16                          </models>
17              </group>
18 </group>
```

Como nesta fase foi nos indicado fazer uma representação inicial, não definimos nenhuma escala apropriada, nem foram colocadas todas as luas dos planetas.Apesar disto tivemos em conta os seus tamanhos para irmos de encontro com a realidade, ou seja se um planeta é mais pequeno que outro, este também o vai ser no nosso modelo.

4 Conclusão

Efetuamos as alterações necessárias no nosso **motor** de gráficos para o mesmo ser capaz de, interpretar um ficheiro XML com diversos grupos, transformações e modelos, como pedido no enunciado do projeto. Elaboramos também um ficheiro XML para modelar uma simulação do sistema solar.

4.1 Trabalho Futuro

Para além do conteúdo requisito das fases ainda por realizar, temos certos objetivos pessoais que gostavamos de realizar ao longo da elaboração do projeto, nomeadamente:

- A implementação de 3 tipos de câmaras que podem ser predefinidas no input XML e alteradas ao longo da execução do engine, particularmente, uma câmara explorador, uma câmara primeira pessoa e uma câmara terceira pessoa[1].
- Otimização do **engine** através do desenho das primitivas usando VBOs e VBOs com indexação.
- Criação de mais tipos de primitivas como, por exemplo, o teapot ou prismas.

[1] ver Eve Online - `https://www.eveonline.com/`

5 Resultados

Os resultados obtidos para cada teste fornecido e para um exemplo do torus.

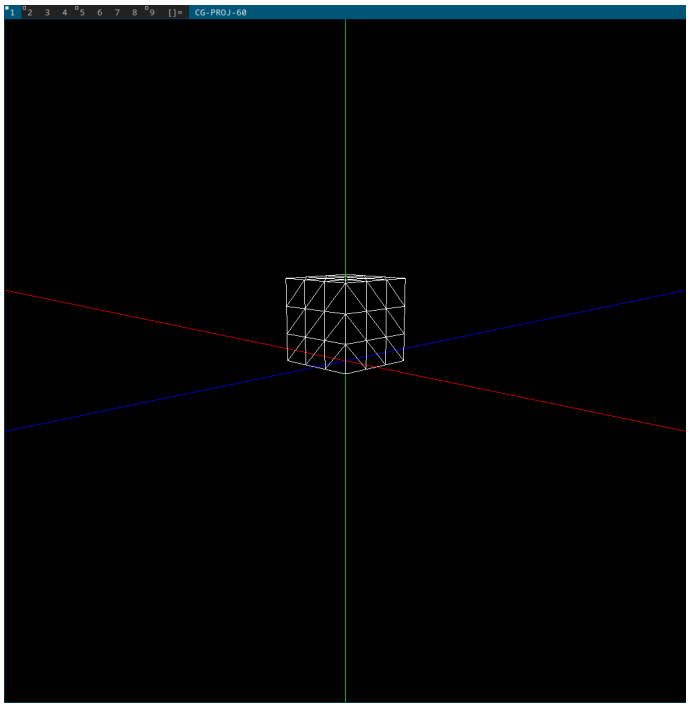

Figura 1: Teste 1

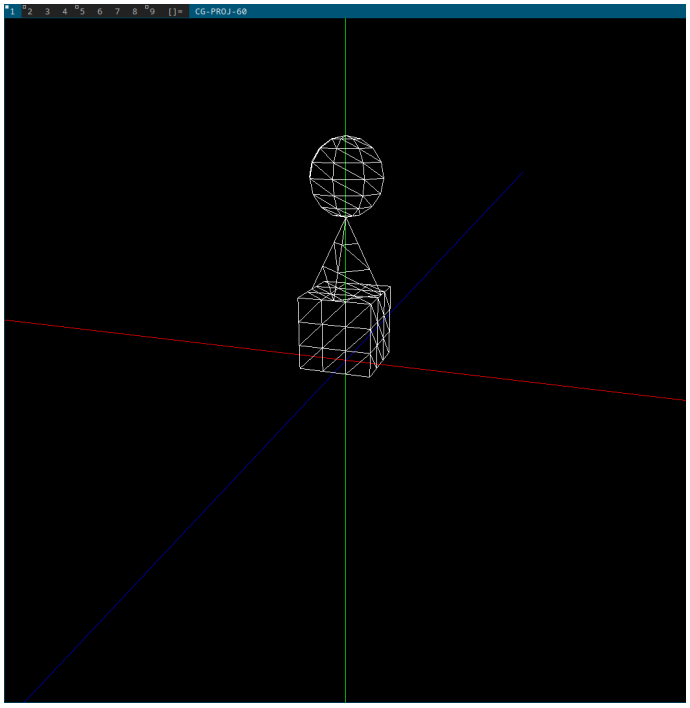

Figura 2: Teste 2

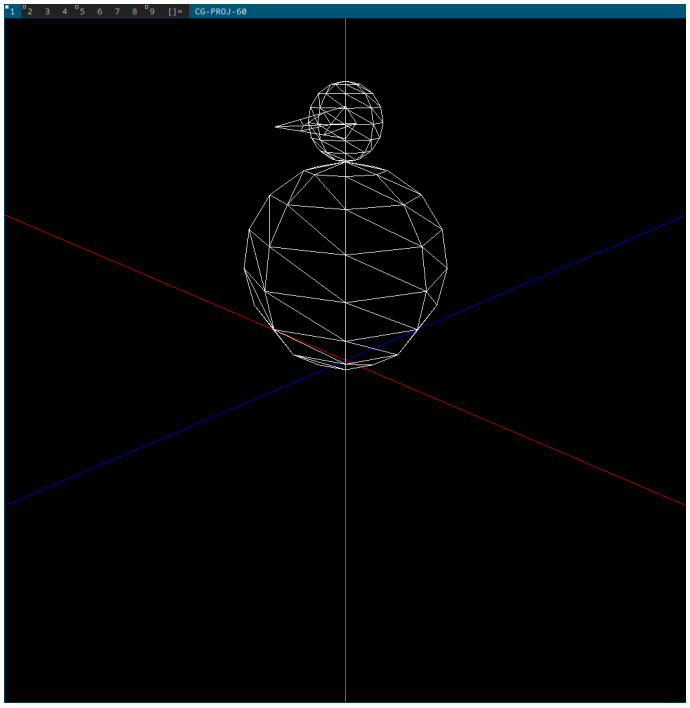

Figura 3: Teste 3

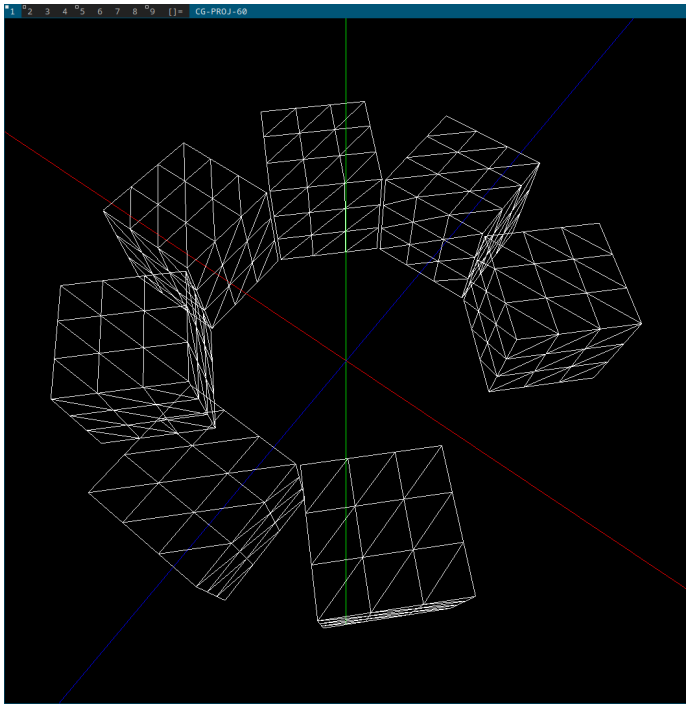

Figura 4: Teste 4

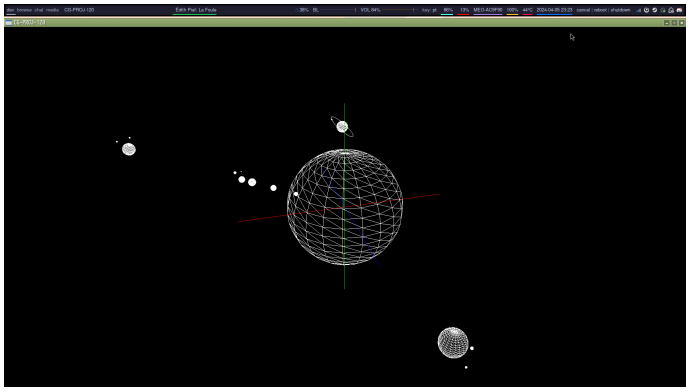

Figura 5: Sistema Solar