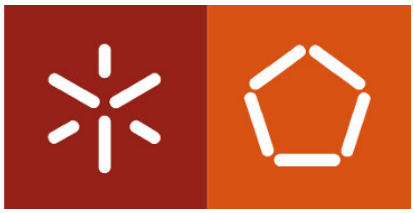

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Computação Gráfica - Trabalho Prático - Fase 2
Ano Letivo 2021/2022
Grupo 6

Gonçalo Braz (a93178) Simão Cunha (a93262)
Tiago Silva (a93277) Gonçalo Pereira (a93168)

5 de junho de 2022

Resumo

Nesta fase do projeto, teremos de alterar o nosso motor de forma a que consiga ler hierarquias de transformações geométricas (`translate`, `rotate` e `scale`) e de modelos de forma a renderizar uma dada cena (como por exemplo, a *demo* do sistema solar requisitada).

Conteúdo

1 Introdução

Este relatório surge no âmbito da segunda fase do trabalho prático da UC de Computação Gráfica, que consiste em adaptar o nosso motor (*engine*) de forma a ser capaz de aplicar transformações geométricas através de uma hierarquia de operações no ficheiro de configuração.

Dada a complexidade desta organização, alteramos o modo de escrita dos vértices, onde utilizamos o VBO (*Vertex Buffer Object*) - esta funcionalidade só será requerida na terceira fase. Em consequência disto, removemos a classe *triangles*. Também decidimos mudar algumas estruturas de dados, tal como se pode observar na adoção de *vectors* em detrimento dos *arrays*, dado que a primeira permite alocação de memória de forma dinâmica [1].

Além disso, foi criada uma *demo* do sistema solar de forma a aplicarmos os conhecimentos das diversas transformações geométricas e da colocação da câmera.

1.1 Correções da fase anterior

Reparámos que o cone estava desenhado de forma errada através da execução dos testes fornecidos pela equipa docente para esta fase. Esta figura geométrica possuia a sua base desenhada em `y = -altura/2` e agora o `y` toma o valor 0. Consequentemente, o vértice do cone foi alterado de `y = altura / 2` para `y = altura`.

2 Arquitetura da solução

2.1 Transformações geométricas

As três transformações geométricas em causa são: translação, rotação e escala. Para as aplicarmos, utilizaremos as seguintes funções do `GLUT` para as implementarmos nos objetos: `glTranslatef`, `glRotatef` e `glScalef`.

2.2 `Transformations`

A classe `transformations` é responsável por aplicar todas as transformações que serão guardadas em `vectors`.

```
1  class transformations {
2  public:
3      vector<transformation*> trs;
4      transformations() {}
5
6      void add_transformation(transformation* tr);
7
8      void transform() {
9          glPushMatrix();
10         for (vector<transformation*>::iterator it = this->trs.begin();
11              it != this->trs.end();
12              ++it) {
13             (*it)->transform();
14         }
15     }
16
17     void destransform() {
18         glPopMatrix();
19     }
20 };
```

Esta classe possui dois métodos: `transform` e `destransform`. A primeira aplica `glPushMatrix()` e, de seguida, aplica todas as transformações existentes no `vector` da variável de instância . Já a segunda limita-se a executar `glPopMatrix()`.

2.2.1 `Transformation`

De forma a englobarmos as três transformações, criamos a classe `transformation` que aplica cada operação que vai ser descrita nas próximas sub-secções.

```
class transformation {
public:
    float angle = 0;
    float x = 0; float y = 0; float z = 0;

    virtual void transform() = 0;
};
```

Esta classe é abstrata [2], pois as suas classes hereditárias serão responsáveis por implementar graficamente cada uma das transformações. Possui variáveis de instância que serão herdadas da classe progenitora.

2.2.2 Translação (`translation`)

A translação é uma operação que consiste em mover um ponto (ou um conjunto de pontos) numa dada direção e comprimento. No exemplo abaixo, `P'` foi obtido através da translação do ponto `P` pelo vetor $\vec{v}$.

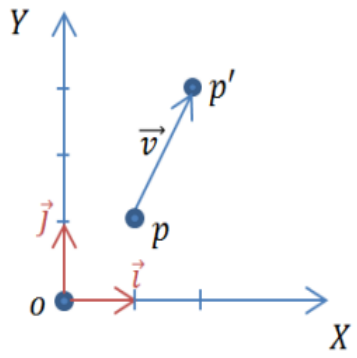

Figura 1: Exemplificação da translação

Assim, a classe `translation` herda atributos da classe `transformation`. Possui variáveis de instância que representam os valores da operação.

```
class translation : public transformation {
public:
    translation(float x, float y, float z) {
        this->x = x;
        this->y = y;
        this->z = z;
    }
    void transform() {
        glTranslatef(this->x, this->y, this->z);
    }
};
```

Esta classe possui ainda um método `transform`, aplicando a função `glTranslatef(x,y,z)` aos vértices guardados nos ficheiros `.3d`.

2.2.3 Rotação (`rotation`)

A rotação é uma transformação geométrica que consiste em rodar um ponto em torno de um eixo.

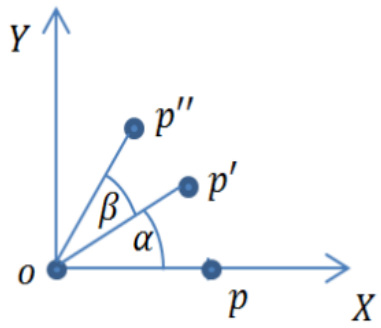

Figura 2: Exemplificação da rotação

Assim, a classe `rotation` herda atributos da classe `transformation`. No exemplo abaixo, o ponto `P''` foi obtido através de uma rotação de $\alpha + \beta$ graus do ponto `P` e `P'` foi obtido através de uma rotação de α graus do ponto `P`.

```
class rotation : public transformation {
public:
    rotation(float angle, float x, float y, float z) {
        this->x = x;
        this->y = y;
        this->z = z;
        this->angle = angle;
    }
    void transform() {
        glRotatef(this->angle, this->x, this->y, this->z);
    }
};
```

Esta classe possui ainda um método `transform`, aplicando a função `glRotatef(angle,x,y,z)` aos vértices guardados nos ficheiros `.3d`.

2.2.4 Escala (`scale`)

A escala é uma transformação geométrica que consiste na multiplicação de um valor escalar (uma constante) às coordenadas de um objeto de forma a aumentá-lo ou diminuí-lo.

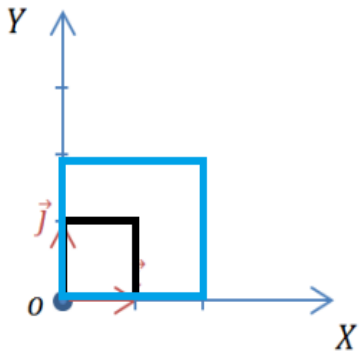

Figura 3: Exemplificação da escala

Assim, a classe `scaling` herda atributos da classe `transformation`. Possui variáveis de instância que representam os valores da operação que nos permite satisfazer a necessidade de podermos aplicar esta operação no nosso motor.

```
1  class scaling : public transformation{
2  public:
3      scaling(float x, float y, float z) {
4          this->x = x;
5          this->y = y;
6          this->z = z;
7      }
8      void transform() {
9          glScalef(this->x, this->y, this->z);
10     }
11 };
```

Esta classe possui ainda um método `transform`, aplicando a função `glScalef(x,y,z)` aos vértices guardados nos ficheiros `.3d`.

2.3 Processamento do ficheiro `XML`

2.3.1 `Group`

A classe `group` surgiu com o intuito de ser capaz de incorporar a hierarquia das transformações geométricas. Esta possui um `vector` de objetos `group`, uma variável de instância de `transformations` e outra `models`. A função que lê um objeto `group` do ficheiro `XML`, quando encontra um objeto desta natureza, chama-se a si mesma de forma recursiva (group xml_group(XMLElement* group_e)).

```
class group {
public:
    vector<group> gs;
    models ms;
    transformations trs;

    group();

    void add_group(group g);
    void add_models(models ms);
    void add_transformations(transformations trs);

    void prepare_data(); //prepara os buffers para desenhar em VBO

    void render();//aplica as tranformações e desenha os modelos
};
```

2.4 *Demo scenes*

2.4.1 Sistema solar

Para esta segunda fase, foi requerida pela equipa docente uma demonstração do nosso projeto através da criação de uma *demo* estática do Sistema Solar.

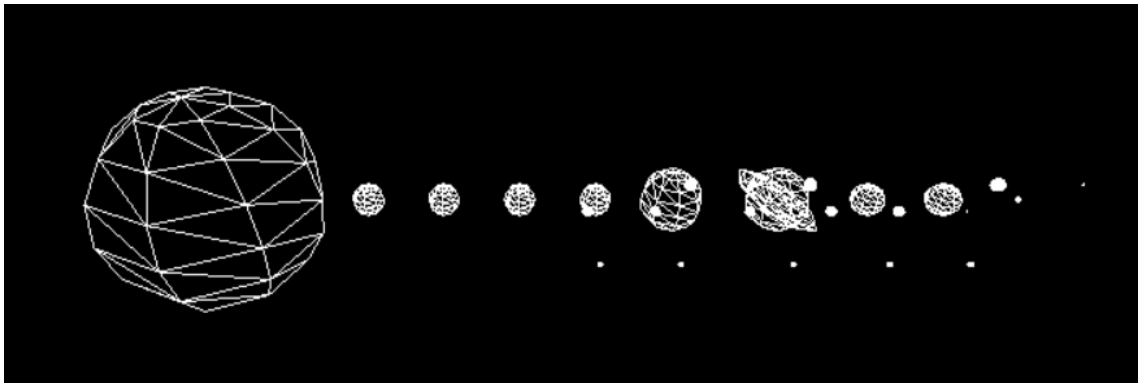

Figura 4: Imagem da *demo* do Sistema Solar

De forma a obtermos a imagem acima, recorremos apenas à primitiva `sphere` (esfera). Esta foi obtida através do *generator* da seguinte forma:

```
$generator sphere 1 8 8 sphere.3d
```

Em primeiro lugar, tivemos que estruturar o ficheiro `XML` que contém a configuração da cena: existirão tantos elementos `group` quantos sol e planetas existirem e dentro de cada figura desenhada irá conter tantos elementos `group` quantas luas tiverem. Decidimos estruturar desta forma, uma vez que torna-se relativamente mais fácil de desenhar os satélites naturais pois as translações a efetuar serão em relação ao centro do seu planeta e não relativamente ao centro do sol (que se encontra na origem do referencial).

Em segundo lugar, tivemos que escolher as coordenadas para a origem do referencial: esta toma o valor (0,0,0).

De seguida, passamos para as coordenadas dos planetas. Desenhamos os mesmos pela ordem correta, mas as distâncias entre eles não estão à escala com os valores reais. Além disso, devido à imensa quantidade de satélites naturais que, por exemplo, Júpiter e Saturno têm, decidimos desenhar 3 das suas luas (que também não estão à escala).

Depois, de forma a simular as órbitas dos planetas, decidimos desenhá-los aplicando uma rotação de 45^{o} em torno do eixo do x - não aplicamos esta operação em Nepturno, Urano e Plutão de forma a podermos vê-los no ecrã.

Por último, colocamos a câmera numa posição *default* de (30,0,-30) de forma a podermos a observar a cena "de cima". No entanto, tornamos possível ao utilizador observar a cena da forma que preferir devido à câmera dinâmica que criamos através do rato do computador ou de algumas teclas do teclado.

NOTA: O anel de Saturno foi desenhado com recurso da primitiva `sphere`, onde foi aplicada a operação de `scale` de forma a que tivesse a forma achatada (diminuímos ao valor de y).

2.4.2 Boneco de neve

Como extra, criamos uma cena de um boneco de neve mais detalhado inspirado no exemplo da equipa docente:

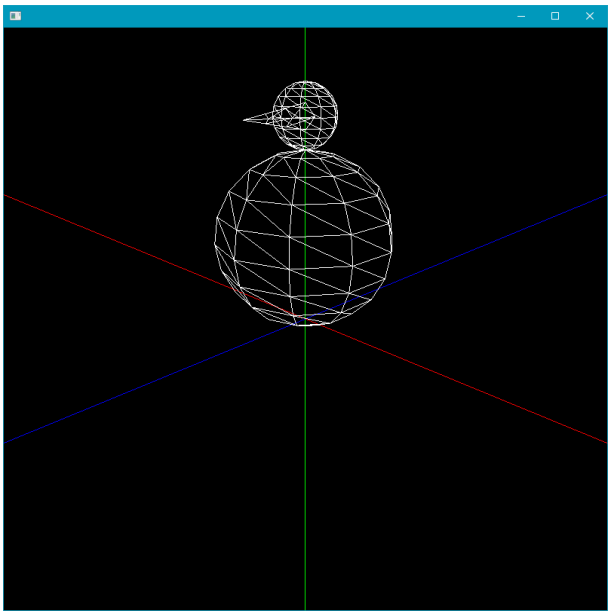

Figura 5: Imagem da *demo* do boneco de neve da equipa docente

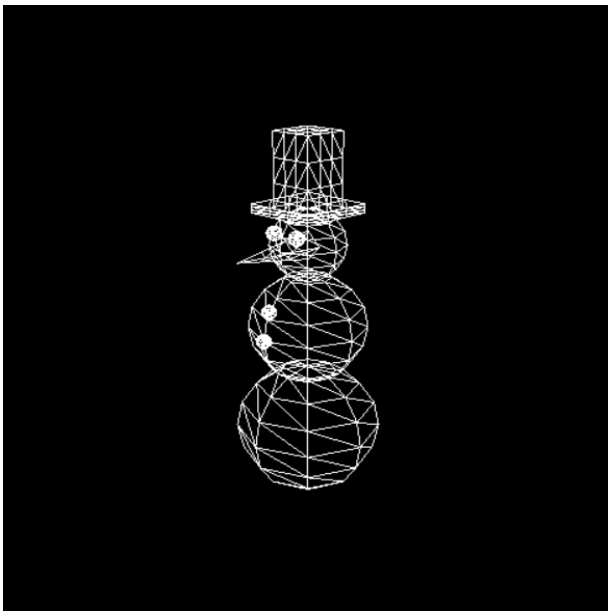

Figura 6: Imagem da *demo* do boneco de neve do nosso grupo

2.5 Implementações extras

De forma a podermos observar de forma mais dinâmica as diversas *demos*, criámos diversas funcionalidades para a câmara:

- Se o utilizador clicar na tecla `R`, mostrar-se-á o referencial adotado (a vermelho o eixo do `x`, a verde o eixo `y` e a azul o eixo `z`);
- Se o utilizador clicar na tecla `P`, será ativado o *pointer*, ou seja, cria uma esfera rosa nas coordenadas do `lookAt` da câmera - foi usada para *debugging* da câmera, mas decidimos manter por ser interessante;
- Calculamos os FPS (*Frames Per Second*) e mostramos o valor no título da janela de visualização;
- Permitimos ao utilizador manobrar na cena através do rato do seu computador, onde o *Mouse1* (botão esquerdo) permite girar o modelo e, com o *Mouse2* (botão direito), faz *zoom-in* ou *zoom-out*, consoante o movimento ascendente ou descendente do seu rato. Além disso, permitimos as tradicionais *keys*: `W`, `A`, `S` e `D`, bem como `E` e `Q` para alterar as coordenadas do `lookAt`.

3 Conclusão

Nesta fase, dedicamo-nos ao processamento de ficheiros `XML` com hierarquização de transformações geométricas como a translação, a rotação e a escala. Conseguimos alterar alguns aspetos (e adicionar alguns não requisitados pela equipa docente), nomeadamente ao nível da colocação da câmara e da estruturação do código - com a escrita de vértices com recurso a `VBO` (por exemplo) - aplicando os conhecimentos obtidos nas aulas teóricas e práticas da UC de Computação Gráfica.

Em suma, consideramos que satisfazemos os requisitos definidos pelos professores para esta fase do trabalho prático.

Referências

[1] https://www.geeksforgeeks.org/advantages-of-vector-over-array-in-c/

[2] https://www.geeksforgeeks.org/pure-virtual-functions-and-abstract-classes/