Universidade do Minho
Escola de Ciências

Computação Gráfica

Ano letivo 21/22

- Fase 2- Transformações Geométricas

Grupo 10:

- Hugo Costeira - A87976
- João Silva - A87939
- João Goulart - A82643
- Pedro Martins - A87964

Índice

1. Introdução

Tendo por base o trabalho já efetuado para a primeira fase do trabalho, esta segunda fase propõe a alteração da fase anterior através da introdução de transformações geométricas como translações, rotações e escalas. Estas transformações permitir-nos-ão representar adequadamente as primitivas da primeira fase e, nesta segunda fase, a nova primitiva “Torus”.

Relativamente ao ficheiro XML, este terá um modelo hierárquico (em árvore), no qual cada nodo terá um conjunto de transformações bem como os modelos a serem carregados. Devido ao facto de a representação do nosso XML utilizar o modelo hierárquico, toda a transformação de um nodo afetará os seus nodos filho. A cena que esta fase propõe que representemos corresponde a um modelo estático do sistema solar (formado pelo sol, planetas e algumas luas definidos hierarquicamente).

2. Generator

O Generator é responsável por criar ficheiros 3D que contém os vértices necessários para a criação de cada uma das primitivas gráficas indicadas. Estas primitivas correspondem à esfera que já foi criada na primeira fase e que utilizaremos para representar o sol, os planetas e as luas (4 luas de Júpiter, 4 luas de Saturno, 2 luas de Urano e 2 luas de Neptuno) bem como a nova primitiva definida “Torus” que utilizaremos para representar os anéis de Saturno e Urano.

2.1. Torus

```
void torus(float innerRadius, float outterRadius, int sides, int rings, char* filename) {
    FILE* f;
    f = fopen(filename, "w");

    float angle_alfa = 2 * M_PI / sides;  //w -> rotação ◆ volta do referencial principal
    float angle_beta = 2 * M_PI / rings;  //v -> rotação ◆ volta do corpo do torus
    float v = 0.0f;
    float w = 0.0f;

    if (f) {
        fprintf(f, "%d \n", 3 * 6 * rings * sides); // num de coordenadas

        for (int i = 0; i < sides; i++) {
            for (int j = 0; j < rings; j++) {
                float x = (outterRadius + innerRadius * cos(v)) * cos(w);
                float x1 = (outterRadius + innerRadius * cos(v)) * cos(w + angle_alfa);
                float x2 = (outterRadius + innerRadius * cos(v + angle_beta)) * cos(w + angle_alfa);
                float x3 = (outterRadius + innerRadius * cos(v + angle_beta)) * cos(w);
                float y = (outterRadius + innerRadius * cos(v)) * sin(w);
                float y1 = (outterRadius + innerRadius * cos(v)) * sin(w + angle_alfa);
                float y2 = (outterRadius + innerRadius * cos(v + angle_beta)) * sin(w + angle_alfa);
                float y3 = (outterRadius + innerRadius * cos(v + angle_beta)) * sin(w);
                float z = innerRadius * sin(v);
                float z1 = innerRadius * sin(v + angle_beta);

                fprintf(f, "%f %f %f \n", x, y, z);
                fprintf(f, "%f %f %f \n", x1, y1, z);
                fprintf(f, "%f %f %f \n", x2, y2, z1);

                fprintf(f, "%f %f %f \n", x2, y2, z1);
                fprintf(f, "%f %f %f \n", x3, y3, z1);
                fprintf(f, "%f %f %f \n", x, y, z);

                v = angle_beta * (j + 1);
            }
            w = angle_alfa * (i + 1);
        }
        printf("Ficheiro .3d criado com sucesso.\n");
    }
    else {
        printf("Erro ao criar o ficheiro.\nTente novamente.\n");
    }
    fclose(f);
}
```

De modo a representar o torus, definimos a função void torus(float r, float R, int sides, int rings, char* filename), que recebe como parâmetros o raio interior innerRadius, o raio exterior outerRadius, o

número de sides, o número de rings e, por último, o ficheiro onde serão guardados os vértices.
Tal como tínhamos por exemplo com a esfera na primeira fase, no torus as sides representam o número de divisões do mesmo ao longo do eixo do x e os rings o número de divisões ao longo do eixo do y.

3. Engine

O Engine refere-se à parte do projeto que utiliza os ficheiros 3d gerados pelo Generator e as transformações geométricas do ficheiro XML para representar a cena pretendida. Nesta fase apenas aproveitamos a implementação feita anteriormente e melhoramos o código para corresponder aos requisitos pedidos. De seguida, só apresentaremos as partes do código que são diferentes da fase anterior.

3.1. Estruturas de dados

```
//VB0s
typedef struct buffer {
    GLuint buffers[1];
    int verticesCount;
    struct buffer* next;
    int pos;
}*Buffer;



typedef struct caracteristicas {

    float* translate;
    float* rotate;
    float* scale;
    int* ordem;

    Buffer b;
    struct caracteristicas** luas;
    int nLuas;
}*Caracteristicas;
```

A estrutura *buffer* permite-nos guardar informação acerca dos VBOs de cada primitiva que teremos de representar através da variável buffers, o número de vértices de cada primitiva com o inteiro verticesCount, a posição em que a primitiva se encontra no sistema solar e, por fim, temos um apontador que nos permite ir para a primitiva seguinte. Através desta estrutura definimos a variável b na

struct características, que é um array com todas as estruturas que vão ser precisas e que vão ser desenhadas com VBOs.

A estrutura *caracteristicas* serve para guardarmos informação acerca de cada transformação que é realizada no Sistema Solar, armazenando desta maneira, as informações relativas às suas transformações (translate, rotate e scale, sendo futuramente adicionado a cor), assim como as informações das suas luas.

3.2. ReadXML

O ficheiro XML apresentado pelo grupo visa a criação do sistema solar requerido constituído pelo sol, planetas, as 4 luas principais de Júpiter e Saturno, 2 luas principais de Urano e Neptuno e a única lua da Terra .Apresenta também os anéis de Saturno e Urano. Nele é guardado a informação sobre as cores e as transformações a aplicar para a melhor simulação do sistema pedido. O grupo tentou ainda que o sistema ficasse o mais idêntico possível com o sistema solar tal como o conhecemos, atribuindo o tamanho correto (em escala) (sol e lua também).

Exemplo de XML utilizado para representar Urano:

```
<group>
        <transform>
            <translate x="37.25" y="0" z="0"/>
            <rotate angle="97.7" x="0" y ="0" z = "1" />
            <scale x="0.35" y="0.35" z="0.35"/>
        </transform>
        <models>
            <model file="sphere.3d"/>
        </models>
        <group>
            <transform>
                <rotate angle="-97.7" x="0" y ="0" z = "1" />
                <translate x="1.04" y="0" z="0"/>  <!-- Lua Urano 1 Titania-->
                <scale x="0.03" y="0.03" z="0.03"/>
            </transform>
            <models>
                <model file="sphere.3d"/>
            </models>
        </group>
        <group>
            <transform>
                <rotate angle="-97.7" x="0" y ="0" z = "1" />
```

```
            <translate x="1.05" y="0" z="0"/>  <!-- Lua Urano 2 Oberon-->
            <scale x="0.03" y="0.03" z="0.03"/>
        </transform>
        <models>
            <model file="sphere.3d"/>
        </models>
    </group>
    <group>
        <transform>
        <rotate angle="97.7" x="1" y="0" z="0"/>
        <scale x="0.35" y="0.35" z="0.35"/>
        </transform>
        <!-- Anel Urano-->
        <models>
            <model file="torus.3d"/>
        </models>
    </group>
</group>
```

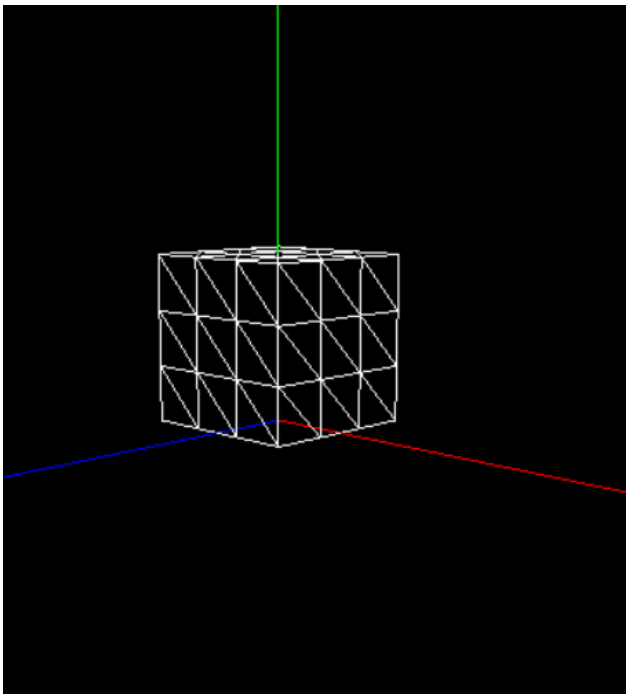
3.3. Transform

```
void Transform(Caracteristicas c, int flag) {

    glTranslatef(init_translate[0], init_translate[1], init_translate[2]);

    if (c->translate != NULL)
        glTranslatef(c->translate[0], c->translate[1], c->translate[2]);

    if (c->rotate != NULL)
        glRotatef(c->rotate[0], c->rotate[1], c->rotate[2], c->rotate[3]);


    if (c->scale != NULL && flag)
        glScalef(c->scale[0], c->scale[1], c->scale[2]);
}
```

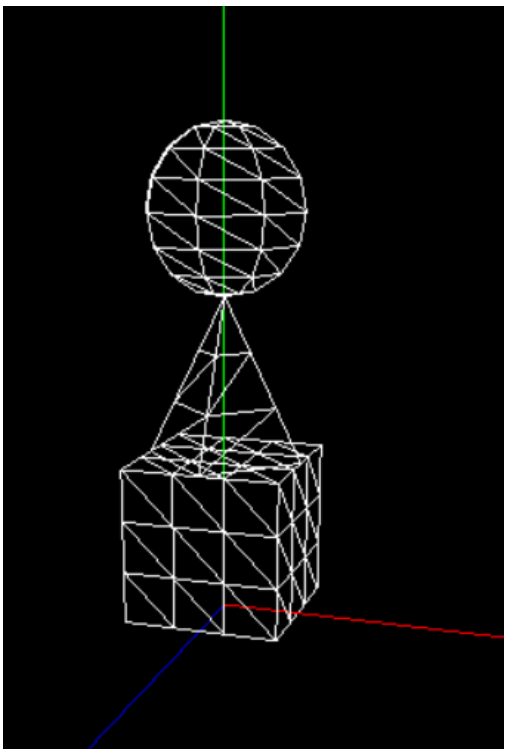
Esta função foi uma função criada para aplicar as transformações presentes no ficheiro xml fornecido.

3.4. Test Files

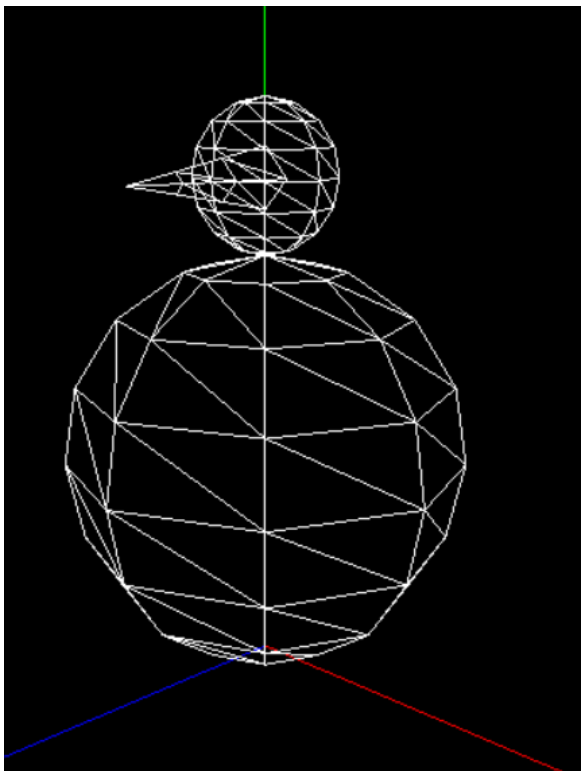
Em seguida mostramos os resultados dos testes dados pelo professor:



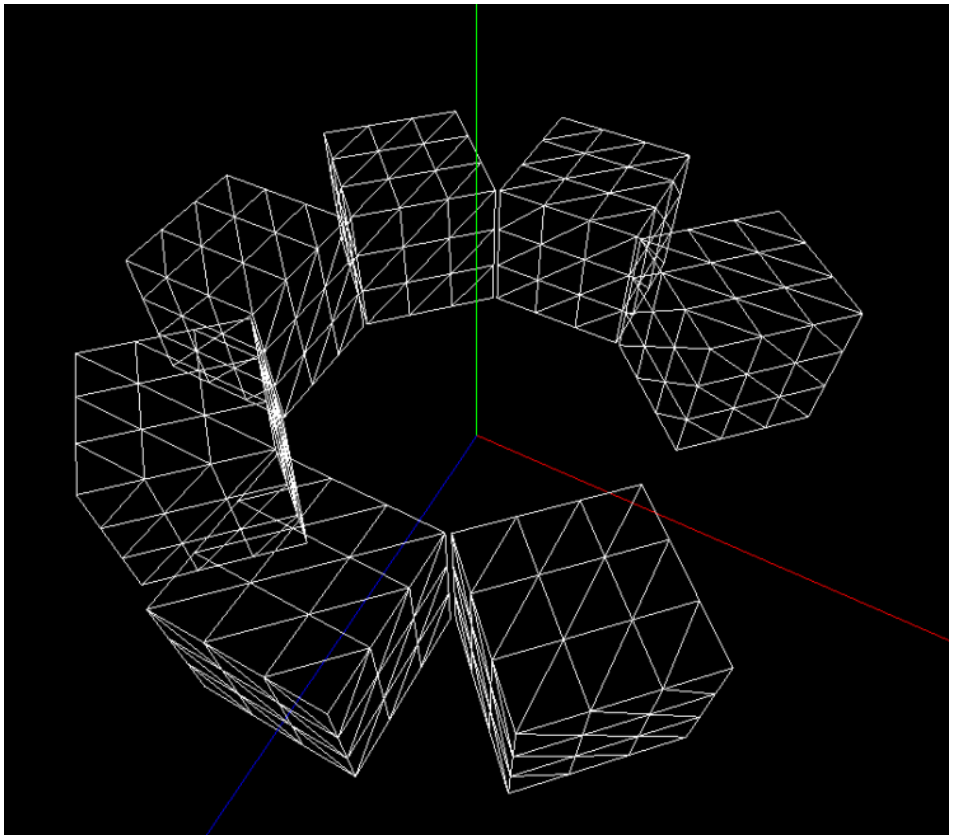
Test_2_1.xml



Test_2_2.xml



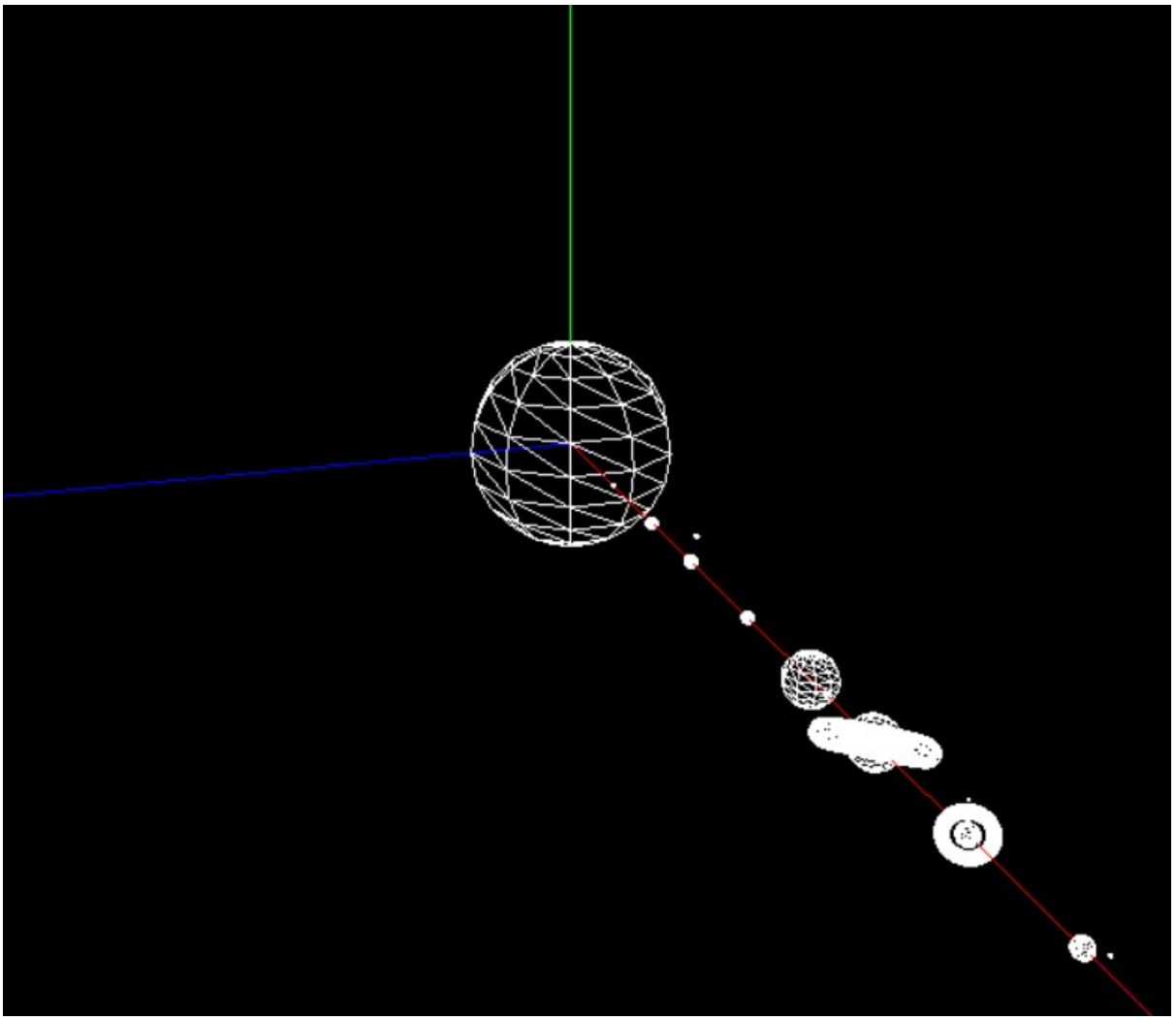
Test_2_3.xml



Test_2_4.xml

4. Resultado Final

De seguida apresentamos uma imagem que demonstra o resultado final do sistema solar:



5. Conclusão

Nesta fase do projeto, houve duas etapas essenciais que consistiram na elaboração do ficheiro XML e de seguida a sua leitura e elaboração das figuras.

A elaboração do XML está estritamente ligada com a construção de estruturas para guardarmos apropriadamente a informação lida.

De seguida, com os dados nas estruturas recorremos ao desenho das figuras sendo que recorremos aos algoritmos utilizados anteriormente mas tendo sempre em consideração o uso das funções rotate e translate que nos permitiu colocar as figuras com diferentes perspectivas em diferentes referenciais.

Em suma, este trabalho foi importante para aprofundar e cimentar tudo o que foi dado que consistia em desenho de figuras recorrendo a triângulos, até a elaboração de várias figuras em diferentes planos, recorrendo a estruturas para guardar o conteúdo lido de XML.