Universidade do Minho
Escola de Ciências

Computação Gráfica

Ciências da Computação
Fase 2

Bruno Fernandes (A95972) | Tiago Silva (A97450) | Tomás Pereira (A97402)

14 de abril de 2023

Índice

Capítulo 1

Introdução

Este relatório foi elaborado no âmbito da segunda fase do trabalho prático da unidade curricular de Computação Gráfica, no qual foi proposto atualizar o *engine* de forma a implementar transformações geométricas com hierarquia. Para além disso, foi-nos proposto criar uma cena com um modelo estático do sistema solar.

Capítulo 2

Generator

As primitivas criadas na primeira fase não foram suficientes para criar o sistema solar, mais precisamente o anel de Saturno. Deste modo, tivemos que implementar a primitiva gráfica do *torus* ao *generator*.

2.1 Primitivas Gráficas

2.1.1 Torus

Para o *torus* utilizamos o raio exterior, raio interior, número de *slices* e número de *stacks* e geramos as coordenadas de cada ponto dos triângulos da seguinte forma:

```
void drawtorus(int rf, int rd, int slices, int stacks, std::string name) {
    std::stringstream s;
    float arch_alpha = (2 * M_PI) / stacks, arch_beta = (2 * M_PI) / slices;
    float x1, x2, x3, x4, y1, y2, y3, y4, z1, z2, z3, z4;

    rf = (rf + rd) / 2;
    rd = rf - rd;


    for (int i = 0; i < stacks; i++) {
        for (int j = 0; j < slices; j++) {
            x1 = (rf + rd * cos(arch_alpha * i)) * cos(arch_beta * j);
            x2 = (rf + rd * cos(arch_alpha * (i+1))) * cos(arch_beta * j);
            x3 = (rf + rd * cos(arch_alpha * (i+1))) * cos(arch_beta * (j+1));
            x4 = (rf + rd * cos(arch_alpha * i)) * cos(arch_beta * (j+1));

            y1 = rd * sin(arch_alpha * i);
            y2 = rd * sin(arch_alpha * (i+1));
            y3 = rd * sin(arch_alpha * (i+1));
            y4 = rd * sin(arch_alpha * i);

            z1 = (rf + rd * cos(arch_alpha * i)) * sin(arch_beta * j);
            z2 = (rf + rd * cos(arch_alpha * (i+1))) * sin(arch_beta * j);
            z3 = (rf + rd * cos(arch_alpha * (i+1))) * sin(arch_beta * (j+1));
            z4 = (rf + rd * cos(arch_alpha * i)) * sin(arch_beta * (j+1));

            s << x1 << ' '  << y1 << ' '  << z1 << '\n';
            s << x2 << ' '  << y2 << ' '  << z2 << '\n';
            s << x4 << ' '  << y4 << ' '  << z4 << '\n';

            s << x2 << ' '  << y2 << ' '  << z2 << '\n';
            s << x3 << ' '  << y3 << ' '  << z3 << '\n';
            s << x4 << ' '  << y4 << ' '  << z4 << '\n';

        }
    }
    wrtfile(s.str(),name);
}
```

Figura 2.1: `Função que calcula as coordenadas dos pontos do torus`

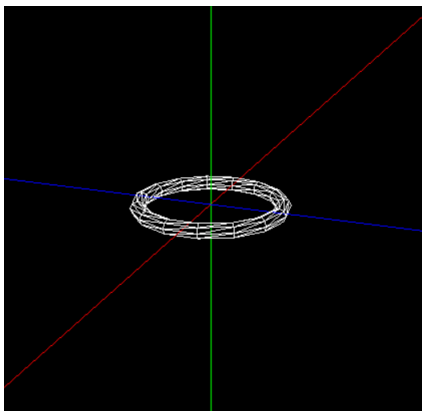

Figura 2.2: Resultado do comando *generator torus 11 9 15 8 torus_11_9_15_8.3d*

Capítulo 3

Engine

3.1 Arquitetura do Código

Para esta fase acrescentamos a possibilidade de fazer transformações. Para isso criamos uma *struct* "objeto" que contém um vetor com as transformações (que são guardadas numa *struct*) e um vetor com os pontos para desenhar o modelo. Desta forma conseguimos agrupar as transformações aplicadas a cada modelo.

Armazenamos nas variáveis globais a seguinte informação:

- Janela:
  - Largura (*width*)
  - Altura (*height*)
- Câmara:
  - Posição da câmara (*position*)
  - Ponto para onde a câmara está a olhar (*lookAt*)
  - Inclinação da câmara (*up*)
  - Projeção (*projection*)
- Transformações:
  - Translações (*translate*)
  - Escalas (*scale*)
  - Rotações (*rotate*)
- Modelos:
  - Nome do(s) ficheiro(s) *.3d*

À *engine* da fase anterior acrescentamos uma função *readgroup* que percorre as transformações e os modelos dos documentos *XML*, guardando a informação num vetor (para cada objeto guardamos os seus pontos e as suas transformações). Para além disso, foi necessário atualizar a função *renderScene* que agora percorre o vetor com os objetos, realizando as transformações geométricas necessárias antes de os desenhar.

Acrescentamos, também, a possibilidade de movimentar a câmara com *input* do teclado, através dos

seguintes comandos:

- Teclas a e d: movimenta uma unidade para a esquerda e para a direita no eixo do x, respetivamente
- Teclas w e s: movimenta uma unidade para cima e para a baixo no eixo do y, respetivamente
- Teclas z e x: movimenta uma unidade para trás e para a frente no eixo do z, respetivamente
- Teclas m e n: aproxima e afasta a câmara, respetivamente

3.2 Resultados Gerados

Para facilitar a utilização da nossa *engine*, decidimos criar um menu para o utilizador escolher a opção que prentende executar:

```
MENU
Escolha o ficheiro que pretende testar:
Fase 1:                 Fase 2:
1 - teste_1_1.xml       6 - test_2_1.xml
2 - teste_1_2.xml       7 - test_2_2.xml
3 - teste_1_3.xml       8 - test_2_3.xml
4 - teste_1_4.xml       9 - test_2_4.xml
5 - teste_1_5.xml      10 - sistema_solar.xml
Opcao:
```

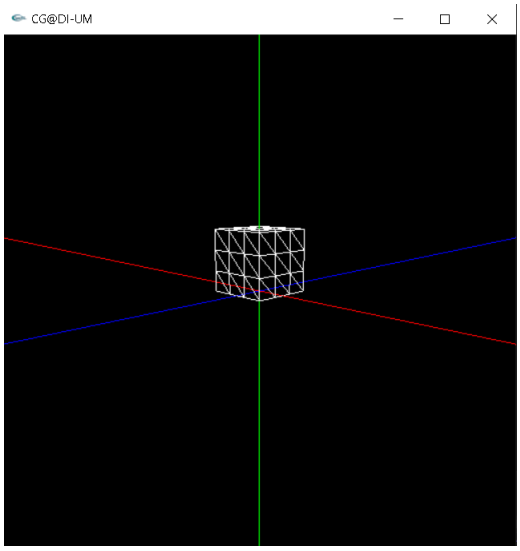

Figura 3.1: `Resultado do` *`test_2_1.xml`*

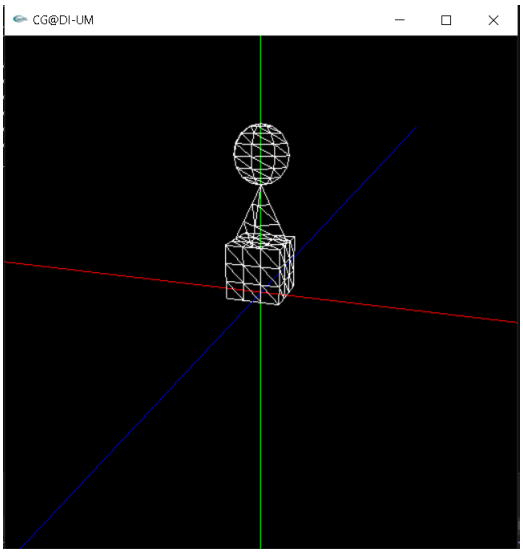

Figura 3.2: `Resultado do` *`test_2_2.xml`*

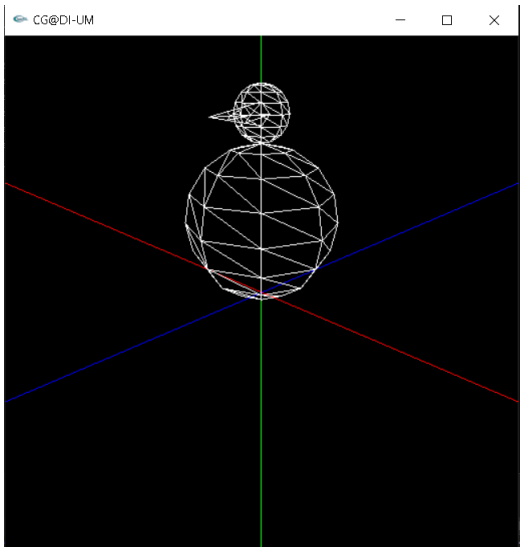

Figura 3.3: `Resultado do` *`test_2_3.xml`*

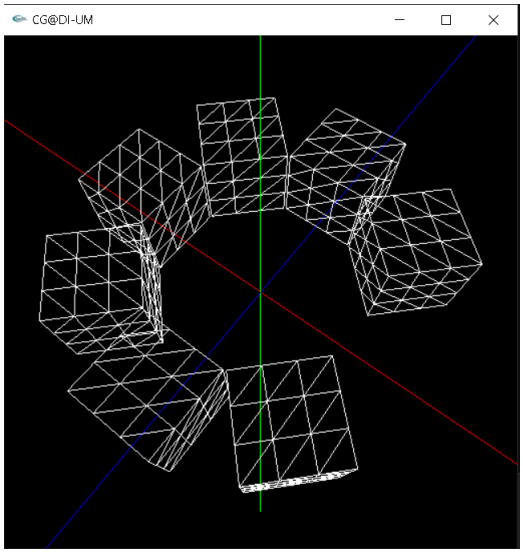

Figura 3.4: `Resultado do` *`test_2_4.xml`*

3.3 Criação do Modelo do Sistema Solar

Para a criação do sistema solar utilizamos o modelo da esfera criado anteriormente para os planetas e para as luas. Para o anel de Saturno utilizamos um *torus*. Desenvolvemos um modelo em *XML* com as transformações necessárias para cada planeta: translação sobre o eixo do x e escala. Não utilizamos escalas muito rigorosas de modo a tornar a visualização mais fácil. Para as luas da Terra e de Júpiter usamos a mesma metodologia dos planetas, aplicando a propriedade de hierarquia para fazer a translação num eixo centrado no planeta.

```xml
<group>
    <group>                                                    <!-- SOL -->
        <transform>
            <scale x="20" y="20" z="20" />
        </transform>
        <colors>
            <color r="1" g="1" b="0" />
        </colors>
        <models>
            <model file="sphere_1_8_8.3d" /> <!-- generator sphere 1 8 8 sphere_1_8_8.3d -->
        </models>
    </group>
    <group>                                                    <!-- MERCÚRIO -->
        <transform>
            <translate x="35" y="0" z="0" />
            <scale x="0.5" y="0.5" z="0.5" />
        </transform>
        <colors>
            <color r="0" g="0" b="1" />
        </colors>
        <models>
            <model file="sphere_1_8_8.3d" /> <!-- generator sphere 1 8 8 sphere_1_8_8.3d -->
        </models>
    </group>
    <group>                                                    <!-- VÉNUS -->
        <transform>
            <translate x="45" y="0" z="0" />
            <scale x="1.5" y="1.5" z="1.5" />
        </transform>
        <models>
            <model file="sphere_1_8_8.3d" /> <!-- generator sphere 1 8 8 sphere_1_8_8.3d -->
        </models>
    </group>
    <group>                                                    <!-- TERRA -->
        <transform>
            <translate x="55" y="0" z="0" />
            <scale x="2" y="2" z="2" />
        </transform>
        <models>
            <model file="sphere_1_8_8.3d" /> <!-- generator sphere 1 8 8 sphere_1_8_8.3d -->
        </models>
        <group>                                                    <!-- TERRA - LUA -->
            <transform>
                <scale x="0.5" y="0.5" z="0.5" />
                <translate x="3.5" y="3.5" z="3.5" />
                <scale x="0.4" y="0.4" z="0.4" />
            </transform>
            <models>
                <model file="sphere_1_8_8.3d" /> <!-- generator sphere 1 8 8 sphere_1_8_8.3d -->
            </models>
        </group>
    </group>
```

Figura 3.5: Excerto do documento *XML* para a geração do sistema solar

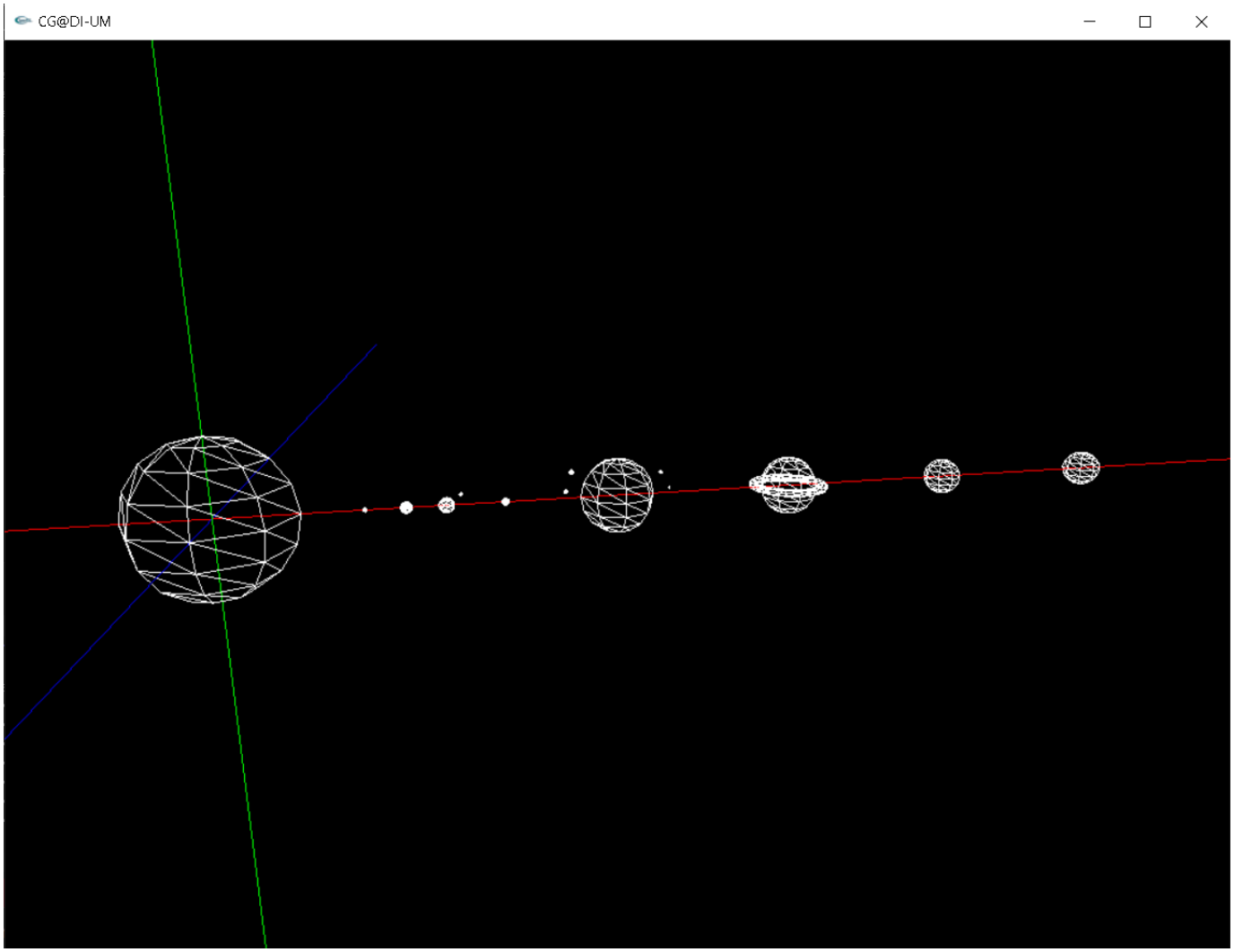

Figura 3.6: `Modelo do sistema solar`

Capítulo 4

Conclusão

Nesta fase o maior desafio foi perceber qual a melhor maneira para armazenar a informação de forma a que fosse preservada uma hierarquia entre os níveis da árvore do *parsing* do *XML*. Consideramos que resolvemos este problema de forma eficaz utilizando vetores de *structs* e uma função recursiva. Em relação à fase anterior, sentimo-nos mais familiarizados com o *OpenGL* e tivemos mais facilidade na criação da nova primitiva gráfica.

Em suma, pensamos ter concluído esta fase com sucesso pois conseguimos gerar o modelo do sistema solar pedido com os pormenores pretendidos.