Computação Gráfica

3º Ano Licenciatura em Ciências da Computação

Grupo 3 - Fase 4 do Trabalho Prático

Modelo de Sistema Solar em OpenGL

Alef Keuffer	Alexandre Baldé	Pedro Paulo Costa Pereira
(A91683)	(A70373)	(A88062)

5 de junho de 2022

Resumo

Neste relatório explicam-se as decisões tomadas para completar a fase 4 do trabalho prática de Computação Gráfica.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este relatório contém a descrição do projeto realizado pelo Grupo 3 para a fase 4 do Trabalho Prático de *Computação Gráfica*, para o ano letivo de 2021/2022.

Estrutura do Relatório

A estrutura do relatório é a seguinte:

- No capítulo 2 faz-se uma análise das mudanças feitas aos executáveis `generator` e `engine`
- No capítulo 4 mostram-se alguns exemplos do modelo do sistema produzido.
- No capítulo 3 mostra-se algum trabalho adicional que não era pedido pelo enunciado.
- No capítulo 5 termina-se o relatório com as conclusões e o trabalho futuro.

Capítulo 2

Notas sobre Implementação

2.1 `Generator.cpp`

Começa-se por referir que a estrutura de dados usada para guardar cada modelo foi atualizada para conter as normais de superfície em cada vértice, e as respetivas coordenadas de textura:

```
struct baseModel {
    int nVertices;
    float *vertices;
    float *normals;
    float *texture_coordinates;
};
```

2.1.1 Geração de vetores normais e coordenadas de textura

Plano

- As normais do plano são simples: (0, 1, 0) ou (0, -1, 0) dependendo da face.
- As coordenadas de textura também são simples, e já foram abordadas nas aulas práticas:

```
for (unsigned int uidiv1 = 1; uidiv1 <= divisions; ++uidiv1) {
    for (unsigned int uidiv2 = 1; uidiv2 <= divisions; ++uidiv2) {
        auto const fdiv1 = (float) uidiv1;
        auto const fdiv2 = (float) uidiv2;
        auto const fdivisions = (float) divisions;

        vertices.emplace_back (... + fdiv1 ..., 0, ... + fdiv2 + ...);
        texture.emplace_back (fdiv1 / fdivisions, fdiv2 / fdivisions);
    }
}
```

'*Box*'

- As normais das caixas são como as do plano: (0, 1, 0), (0, -1, 0), (1, 0, 0), (-1, 0, 0), (0, 0, 1), (0, 0, -1), dependendo da face.
- as coordenadas de textura da caixa fazem o mesmo que as do plano, para cada face.

Esfera

- As normais da esfera são obtidas através das coordenadas esféricas obtidas para calcular o vértice: se um ponto tiver coordenadas esféricas (ρ, ϕ, θ), então as suas coordenadas cartesianas são $(\rho \cdot \sin \theta \cdot \cos \phi, \rho \cdot \sin \phi, \rho \cdot \cos \theta \cdot \cos \phi)$, e a normal da esfera nesse ponto é $(\sin \theta \cdot \cos \phi, \sin \phi, \cos \theta \cdot \cos \phi)$.
- As coordenadas de textura são obtidas de uma forma semelhante às do plano e cubo, em que se faz corresponder o ponto em que se está nas iterações às "*stacks*" e "*slices*" a um ponto no espaço textura:

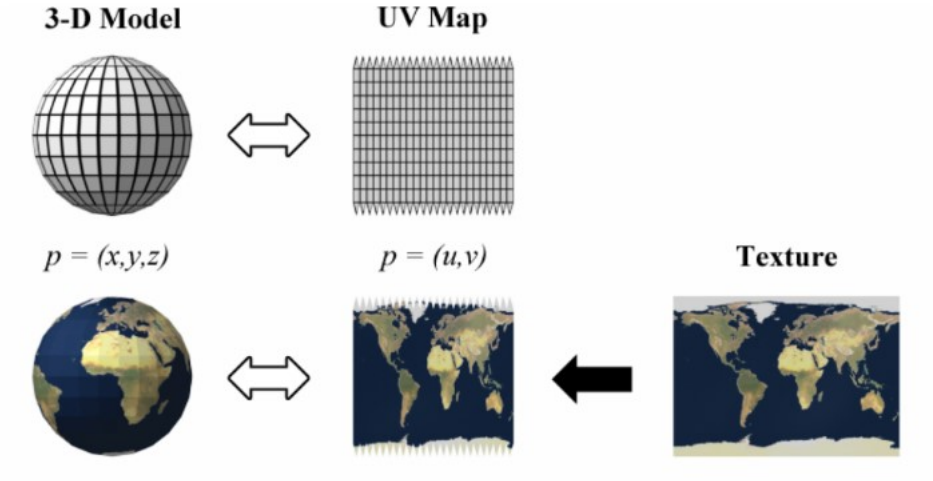

Figura 2.1: Normal para superfície curva do cone

```
for (unsigned int slice = 1; slice <= slices; ++slice) {
    for (unsigned int stack = 1; stack <= stacks; ++stack) {
        auto fslice = (float) slice;
        auto fstack = (float) stack;

        vertices.emplace_back (r, theta + s * fslice, phi + t * fstack);
        texture.emplace_back (fslice / fslices, fstack / fstacks);
    }
}
```

Cone

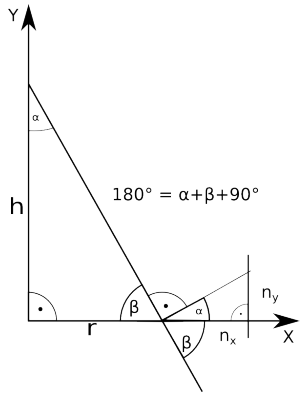

Figura 2.2: Normal para superfície curva do cone

- As normais do cone são calculadas de acordo com uma aproximação: para cada triângulo, calcula-se o seu vetor normal, que será dado a cada um dos seus vértices componentes.

  No entanto, os autores reconhecem que esta aproximação não se adequa para superfícies curvas como a do cone, servindo melhor para modelos que são intencionalmente poligonais. Devido à fórmula que se utilizou para o cálculo do cone (obtida, como referido em relatórios anteriores, através do serviço `https://www.math3d.org`), não foi possível adaptar facilmente a fórmula que se deriva da imagem 2.2, que seria:

$$(\cos(\beta) \cdot \sin(\alpha), \sin(\beta), \cos(\beta) \cdot \cos(\alpha))$$

```
beta = atan(radius / height);

for (int st = 0; st <= stacks; st++) {
    for (int sl = 0; sl <= slices; sl++) {
        alpha = sl * alpha_offset;

        curr_x = new_radius * sin(alpha);
        curr_y = st * stack_height;
        curr_z = new_radius * cos(alpha);

        vertices.emplace_back(curr_x, curr_y, curr_z);
        textures.emplace_back( sl / (float) slices, st / (float) stacks);
        normals.emplace_back(
            cos(beta) * sin(alpha),
            sin(beta),
            cos(beta) * cos(alpha));
    }
  ...
}
```

Superfície de Bezier

- As normais para as superfícies de Bezier foram simples, cortesia do Professor Ramires.

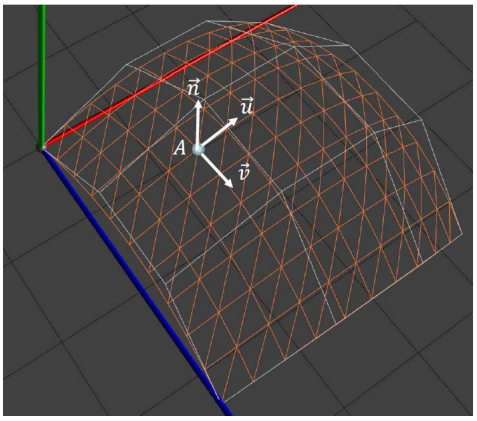

Figura 2.3: Normal para superfície de Bezier

$$\frac{\partial B(u,v)}{\partial u} = [3u^2 \quad 2u \quad 1 \quad 0]M \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T V^T$$

$$\frac{\partial B(u,v)}{\partial v} = UM \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} M^T \begin{bmatrix} 3v^2 \\ 2v \\ 1 \\ 0 \end{bmatrix}$$

Figura 2.4: Cálculo de normal para superfície de Bezier

Nota-se apenas que o cálculo em 2.4 foi relativamente simplificado por se ter usado a biblioteca `glm`[1].

- Para as texturas: considere-se o exemplo dado pelo Professor:

Figura 2.5: Exemplo de superfície de Bezier com textura

[1] código fonte em `https://github.com/g-truc/glm`

Para obter este efeito com repetições, fez-se o seguinte código:

```
auto float_tesselation = (float) int_tesselation;
const auto step = 1.0 / float_tesselation;

for (int v = 0; v < int_tesselation; ++v) {
    for (int u = 0; u < int_tesselation; ++u) {
        get_bezier_point_at (
          u * step,
          v * step,
          control_points,
          vertex,
          normal);

        auto fu = (float) u;
        auto fv = (float) v;
        vec3 vertex, normal;
        vertices.push_back (vertex);
        normals.push_back (normal);
        texture.emplace_back (-(u * step), -(v * step));
    }
}
```

ou seja, está-se a repetir a textura ao longo dos "*patches*" da superfície.

2.2 `Engine.cpp`

Para poder lidar com os novos campos XML com informação sobre luzes, propriedades dos materiais e ficheiros de textura, o código de "*parsing*" em `parsing.cpp` descrito nos relatórios anteriores teve que ser atualizado.

2.2.1 Valores por defeito para as luzes

Como não foi definido no enunciado que valores de RGB utilizar para as diferentes componentes de luz que o OpenGL suporta, escolheram-se os seguintes:

```
const auto RGB_MAX = 255.0;
struct model {
  GLsizei nVertices{};
  GLuint vbo{};
  GLuint normals{};
  struct {
/*
Source:
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/xhtml/glMaterial.xml
*/
    vec4 diffuse{200.0 / RGB_MAX, 200.0 / RGB_MAX, 200.0 / RGB_MAX, 1};
    vec4 ambient{50.0 / RGB_MAX, 50.0 / RGB_MAX, 50.0 / RGB_MAX, 1};
    vec4 specular{0, 0, 0, 1};
    vec4 emissive{0, 0, 0, 1};
    GLfloat shininess = 0;
  } material{};
/*
0 default value means it's optional with 0 meaning
it's not being used by a particular model.
*/
  GLuint tbo = 0; // texture buffer object
  GLuint tc = 0; // texture coordinates
};
```

Capítulo 3

Trabalho adicional

3.1 Extensão ao XML

De forma a poder executar os testes mais rapidamente, efetuaram-se mudanças no `engine` para poder aceitar ficheiros XML com as seguintes modificações:

```
<group>
    <models>
        <model file="plane.3d"> <!— generator plane 1 3 plane_nt.3d —>
            <generator argv="plane 2 1 plane.3d"/>
            <texture file="../test_files_phase_4/relva.jpg"/>
        </model>
    </models>
</group>
```

Através do uso do padrão `fork() / exec()`, e tornando o executável `engine` dependente do `generator` no `cmake`, simplifica-se a compilação e execução de um cenário de teste.

`./engine test_4_7.xml`, contendo toda a informação necessária para correr `generator`, gerará os vértices, normais e coordenadas de textura para cada modelo no ficheiro, e iniciará o programa com os ficheiros ".3d" necessários.
Veja-se o exemplo completo no Anexo A

3.2 Câmara FPS

Implementou-se, para além do modo explorador fornecido pelo Professor Ramires para as aulas práticas, um modo FPS com movimento através de periféricos (rato/teclado) para exploração do sistema solar.

É possível alterar os dois modos dinamicamente durante a execução do programa.

3.3 "*Skybox*" para sistema solar

Por sugestão do Professor Carlos Brito, para simular um ambiente visual mais rico durante a execução do `engine` com o modelo do sistema solar, acrescentou-se uma "*skybox*" com uma textura apropriada, retirada de Deep Star Maps 2020.

O método é o seguinte:

- Acrescenta-se uma esfera com tamanho apropriado no XML do sistema solar.
- Escolhe-se a textura que cobrirá o modelo da esfera
- Trata-se o modelo como todos os outros a respeito de textura, e "*rendering*".

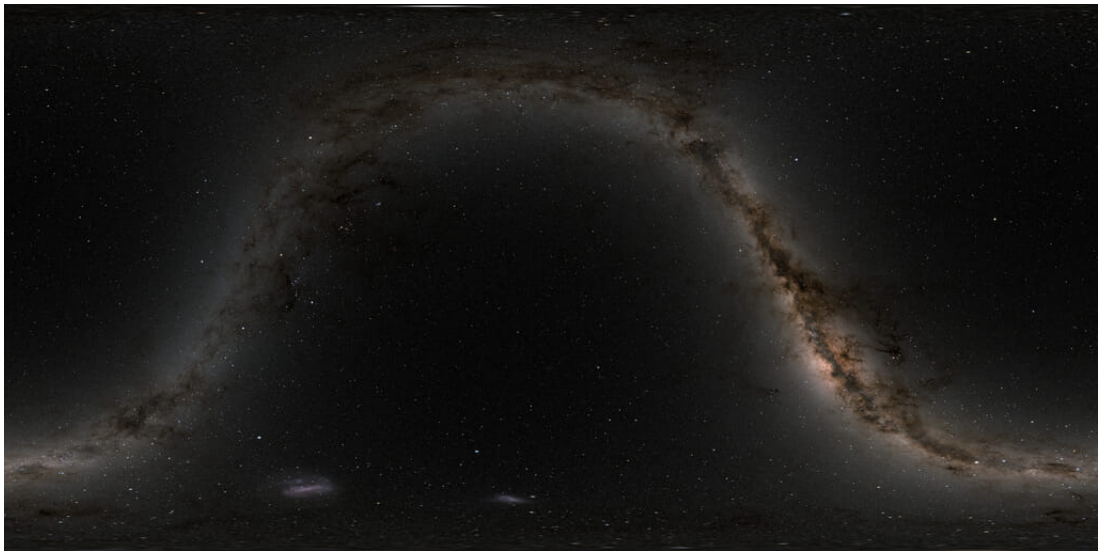

Figura 3.1: Imagem utilizada para pano de fundo

3.4 Movimento no espaço através de periférico

Através da função `glUnProject`, é possível, através de "*input*" predefinido (neste caso, o botão direito do rato), movimentar a posição da câmara no espaço global para a posição do cursor na janela.

Capítulo 4

Exemplo de utilização

Observe-se na imagem abaixo o sistema solar, com a "*skybox*" escolhida, assim como o "cometa" com trajetória numa curva de Catmull-Rom.

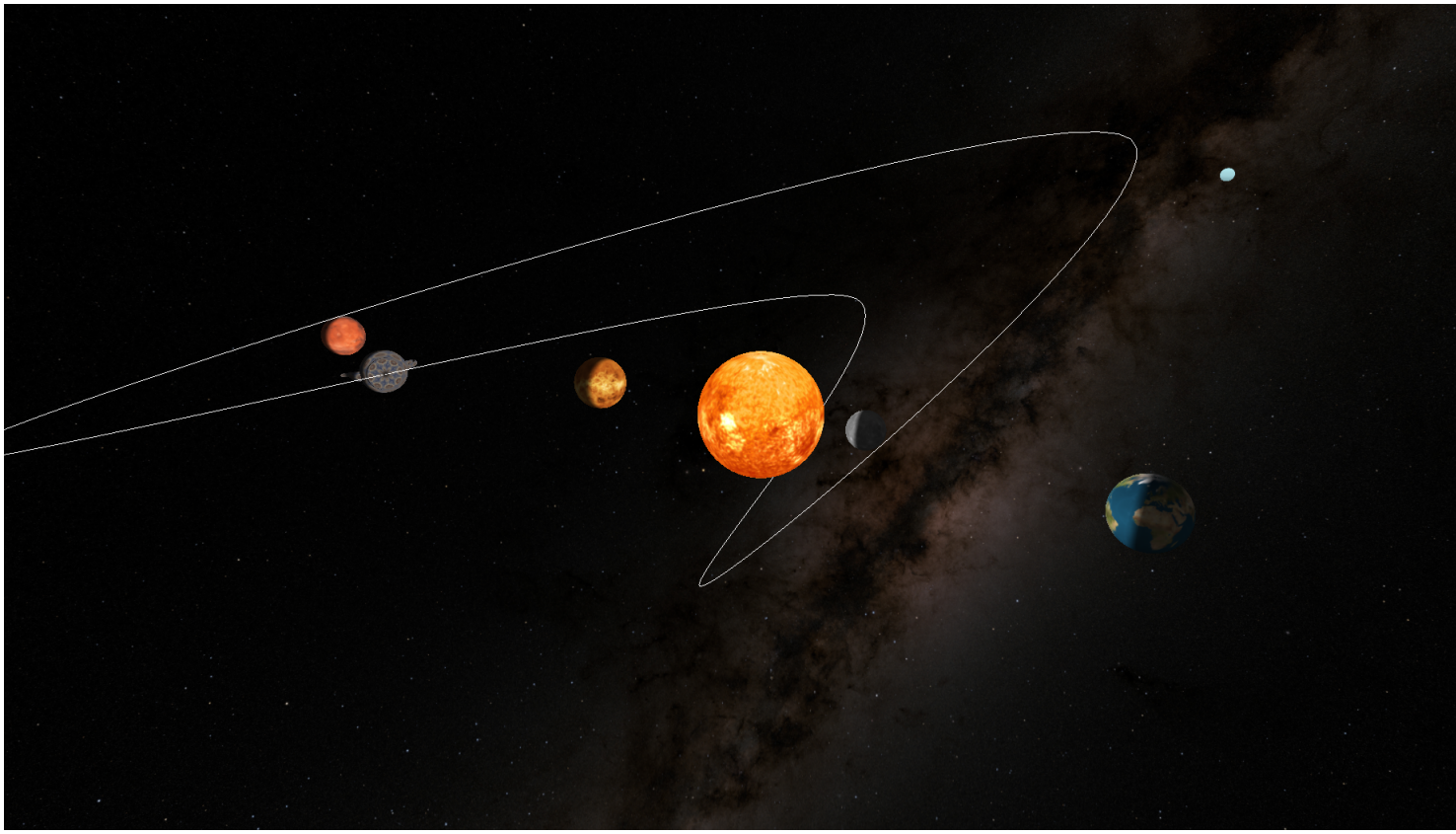

Figura 4.1: Sistema solar com órbita de ecometa e Via Lácter visíveis

Capítulo 5

Conclusão

Conclui-se desta forma a apresentação da fase 4 do Projeto Prático de *Computação Gráfica* do Grupo 3.

5.1 Comentários

A aproximação que se utilizou ao gerar as normais de superfície do cone, embora produza efeitos aceitáveis, não é ideal porque se o número de "*slices*" for baixo, terá precisão baixa quando comparada com a utilização da fórmula correta.

5.2 Trabalho Futuro

Este projeto foi escrito num contexto académico, e não terá qualquer uso futuro adicional.

Se tivesse, a primeira necessidade seria alterar a geração dos modelos de sólidos geométricos para facilitar o cálculo de normais e coordenadas de textura - embora o serviço Math3d tenha auxiliado na fase inicial para prototipagem, dificultou, no final, o cálculo dessa componentes para cada modelos.

Apêndice A

Excertos de Código Utilizado no Projeto

Listing A.1: Exemplo de XML com informação adicional de geração de modelos

```
<world>
<generator dir="../bin/generator"/>
<camera>
    <position x="-0.740200" y="1.182951" z="1.376347"/>
    <lookAt x="0" y="0" z="0"/>
    <up x="0" y="1" z="0"/>
    <projection fov="60" near="1" far="1000"/>
</camera>

<lights>
    <light type="point" posX="0" posY="0.2" posZ="0"/>
    <light type="point" posX="-2" posY="2" posZ="2"/>
    <!--        <light type="directional" dirX="-2" dirY="2" dirZ="2"/-->
</lights>

<group>
    <group>
        <models>
            <model file="plane.3d"> <!-- generator plane 1 3 plane_nt.3d -->
                <generator argv="plane␣2␣1␣plane.3d"/>
                <texture file="../test_files_phase_4/relva.jpg"/>
            </model>
        </models>
    </group>
    <group>
        <transform>
            <translate x="-0.5" y="0" z="-0.5"/>
            <scale x="0.2" y="0.2" z="0.2"/>
        </transform>
        <models>
            <model file="cone.3d"> <!-- generator cone 1 2 10 10 cone_nt.3d -->
                <generator argv="cone␣1␣2␣5000␣100␣cone.3d"/>
                <texture file="../test_files_phase_4/cone.jpg"/>
            </model>
        </models>
    </group>
    <group>
```

```
            <transform>
                <translate x="0.5" y="0.2" z="0.5"/>
                <scale x="0.2" y="0.2" z="0.2"/>
            </transform>
            <models>
                <model file="sphere.3d"> <!— generator sphere 1 32 32 sphere_nt.3d —>
                    <generator argv="sphere␣1␣100␣100␣sphere.3d"/>
                    <texture file="../test_files_phase_4/earth.jpg"/>
                </model>
            </models>
        </group>
        <group>
            <transform>
                <translate x="-0.5" y="0" z="0.5"/>
                <scale x="0.1" y="0.1" z="0.1"/>
                <rotate angle="-90" x="1" y="0" z="0"/>
            </transform>
            <models>
                <model file="teapot.3d"> <!— generator patch teapot.patch 10 bezier_nt.3d —>
                    <generator argv="bezier␣../test_files_phase_3/teapot.patch␣10␣teapot.3d"/>
                    <texture file="../test_files_phase_4/teapot.jpg"/>
                </model>
            </models>
        </group>
        <group>
            <transform>
                <scale x="0.2" y="0.2" z="0.2"/>
                <translate x="2.5" y="1.0" z="-2.5"/>
            </transform>
            <models>
                <model file="box.3d"> <!— generator box 2 3 box_nt.3d —>
                    <generator argv="box␣2␣3␣box.3d"/>
                    <texture file="../test_files_phase_4/box.jpg"/>
                </model>
            </models>
        </group>

    </group>
</world>
```