TP3-CG-MIEI

João Bernardo Freitas a74814

3 Maio 2020

Contents

1 Introdução

Neste relatório irei apresentar e discutir o trabalho realizado pelo grupo, no âmbito da Unidade Curricular de Computação Gráfica, do 4º ano do Mestrado Integrado em Engenharia Informática.
Nesta última fase tive como objectivo actualizar a cena criada na fase anterior, adicionando luzes e texturas. Mais uma vez, todos os modelos foram desenhados com recurso a **VBOs**.
Como tal, neste relatório, irei apresentar ao detalhe a forma como cumpri esses objetivos.

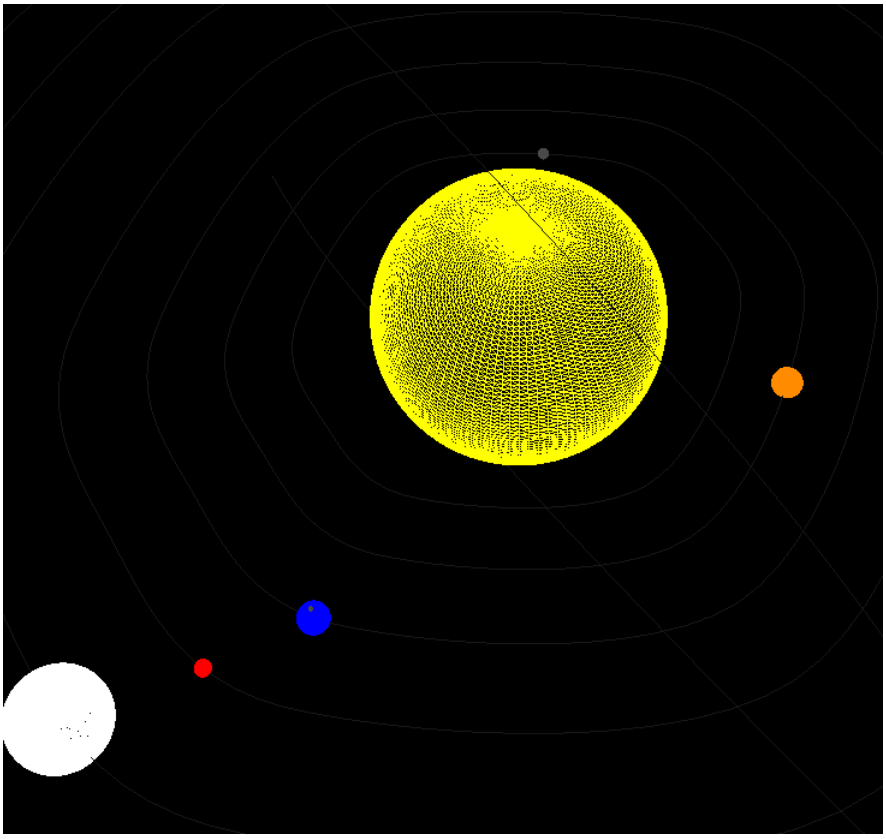

Figure 1: Resultado Fase 3

2 Generator

Para poder gerar os pontos para a textura das esferas é necessário calcular os pontos onde a textura é aplicada.
Para tal, usei duas variáveis,**t** e **s**, para a geração das coordenadas da textura para a esfera.

```
for (int i = 0; i < stacks; i++) {
    alfa = 0;
    t = (stacks - i) * deltaT;
    for (int j = 0; j < slices; j++) {
        s = j * deltaS;
```

Figure 2: Variáveis t e s

Estas coordenadas, juntamente com as coordenadas do ponto e as suas normais irão ser escritas no ficheiro *sphere.3d*, onde a primeira linha é o vértice em si, a segunda linha é o vértice que representa a sua normal, que é usada para aplicar a luz no modelo, e por fim o vértice da textura.

```
ficheiro << ponto1x << " " << ponto1y << " " << ponto1z << "\n";
ficheiro << sin(beta) * sin(alfa) << " " << cos(beta) << " " << sin(beta) * cos(alfa) << "\n";
ficheiro << s << " " << t << " " << -1 << "\n";

ficheiro << ponto4x << " " << ponto4y << " " << ponto4z << "\n";
ficheiro << sin(beta + deltaBeta) * sin(alfa + deltaAlfa) << " " << cos(beta + deltaBeta) << " " << sin(beta + deltaBeta) * cos(alfa + deltaAlfa) << "\n";
ficheiro << (s + deltaS) << " " << (t - deltaT) << " " << -1 << "\n";

ficheiro << ponto3x << " " << ponto3y << " " << ponto3z << "\n";
ficheiro << sin(beta) * sin(alfa + deltaAlfa) << " " << cos(beta) << " " << sin(beta) * cos(alfa + deltaAlfa) << "\n";
ficheiro << (s + deltaS) << " " << t << " " << -1 << "\n";


ficheiro << ponto1x << " " << ponto1y << " " << ponto1z << "\n";
ficheiro << sin(beta) * sin(alfa) << " " << cos(beta) << " " << sin(beta) * cos(alfa) << "\n";
ficheiro << s << " " << t << " " << -1 << "\n";

ficheiro << ponto2x << " " << ponto2y << " " << ponto2z << "\n";
ficheiro << sin(beta + deltaBeta) * sin(alfa) << " " << cos(beta + deltaBeta) << " " << sin(beta + deltaBeta) * cos(alfa) << "\n";
ficheiro << s << " " << (t - deltaT) << " " << -1 << "\n";

ficheiro << ponto4x << " " << ponto4y << " " << ponto4z << "\n";
ficheiro << sin(beta + deltaBeta) * sin(alfa + deltaAlfa) << " " << cos(beta + deltaBeta) << " " << sin(beta + deltaBeta) * cos(alfa + deltaAlfa) << "\n";
ficheiro << (s + deltaS) << " " << (t - deltaT) << " " << -1 << "\n";
```

Figure 3: Criação de uma esfera

3 Parser

O parser foi novamente alterado de forma a poder ler estes novos parâmetros.
Para guardar o tipo e parâmetros da luz no sistema a seguinte estrutura foi criada.

```
typedef struct luz {
    char* type;
    double params[3];
}Luz;
```

Figure 4: Estrutura da luz

A mesma lógica foi utilizada para guardar o tipo e os parâmetros para a cor.

```
typedef struct cor {
    char* type;
    double colors[3];
}Cor;
```

Figure 5: Estrutura da cor

O que levou a uma nova alteração da estrutura dos **Ficheiros**.

```
typedef struct ficheiros {
    char* file;
    int numpontos;
    char* textura;
    Cor* cor;
    vector<Transforma*> transfs;
} Ficheiros;
```

Figure 6: Estrutura ficheiros

4 Extracção

Visto que o ficheiro **sphere.3d** irá agora conter tanto as coordenadas dos vértices mais as texturas e suas normais foi necessário alterar a função que extraía as coordenadas.

```
if (nlines % 3 == 1) {
    //cout << "VERTICE->LINHA->" << nlines << " X->" << p.x << " Y->" << p.y << " Z->" << p.z << endl;
    vertices.push_back(p);
    ficheiro->numpontos++;
    (*nVertices)++;
}
else if(nlines%3==2){
    //cout << "NORMAL->LINHA->" << nlines << " X->" << p.x << " Y->" << p.y << " Z->" << p.z << endl;
    normals.push_back(p);
}
else if(nlines%3==0){
    //cout << "TEXTURA->LINHA->" << nlines << " X->" << p.x << " Y->" << p.y << " Z->" << p.z << endl;
    texturas.push_back(p);
}
```

Figure 7: Extracção de coordenadas

Mais uma vez é as coordenadas (2020,2020,2020) são adicionadas para indicar o fim de um modelo.

5 Luz e Cor

A luz é adicionada pela função **adicionaLuz()**, onde, através da componente type da estrutura **luz**, se verifica o tipo de luz que é necessário aplicar ao sistema solar.

```
void adicionaLuz() {
    GLfloat pos[4] = { 0.0, 0.0, 1.0, 0.0 };
    GLfloat amb[4] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat diff[4] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat spotDir[3] = { 0.0, 0.0, -1.0 };

    Luz* l = luz.front();

    pos[0] = static_cast<GLfloat>(l->params[0]);
    pos[1] = static_cast<GLfloat>(l->params[1]);
    pos[2] = static_cast<GLfloat>(l->params[2]);
    pos[3] = 1;

    if (strcmp(l->type, "POINT") == 0) {

        glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_POSITION, pos);
        glLightfv(GL_LIGHT0, GL_AMBIENT, amb);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, diff);

    }
```

Figure 8: Adiciona Luz

A mesma lógica é aplicada para adicionar cor, que trata das luzes individuais de cada modelo.

```
void adicionaCor(Cor* cor) {
    GLfloat amb[4] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat dif[4] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat spec[4] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat emis[4] = { 0.0, 0.0, 0.0, 1.0 };

    if (strcmp(cor->type, "diffuse") == 0) {
        dif[0] = static_cast<GLfloat>(cor->colors[0]);
        dif[1] = static_cast<GLfloat>(cor->colors[1]);
        dif[2] = static_cast<GLfloat>(cor->colors[2]);
        dif[3] = 1;

        glMaterialfv(GL_FRONT, GL_DIFFUSE, dif);
    }

    if (strcmp(cor->type, "specular") == 0) {
        spec[0] = static_cast<GLfloat>(cor->colors[0]);
        spec[1] = static_cast<GLfloat>(cor->colors[1]);
        spec[2] = static_cast<GLfloat>(cor->colors[2]);
        spec[3] = 1;

        glMaterialfv(GL_FRONT, GL_SPECULAR, spec);
    }

    if (strcmp(cor->type, "emissive") == 0) {
        emis[0] = static_cast<GLfloat>(cor->colors[0]);
        emis[1] = static_cast<GLfloat>(cor->colors[1]);
        emis[2] = static_cast<GLfloat>(cor->colors[2]);
        emis[3] = 1;

        glMaterialfv(GL_FRONT, GL_EMISSION, emis);
    }

    if (strcmp(cor->type, "ambient") == 0) {
        amb[0] = static_cast<GLfloat>(cor->colors[0]);
        amb[1] = static_cast<GLfloat>(cor->colors[1]);
        amb[2] = static_cast<GLfloat>(cor->colors[2]);
        amb[3] = 1;

        glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
    }
}
```

Figure 9: Adiciona cor

Obtendo então luz dinâmica nos planetas.

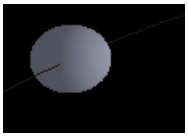

Figure 10: Luz num planeta

6 Resultado final

Por fim temos a representação do sistema solar, contendo o sol e os 8 planetas e suas orbitas e a luz proveniente do sol.

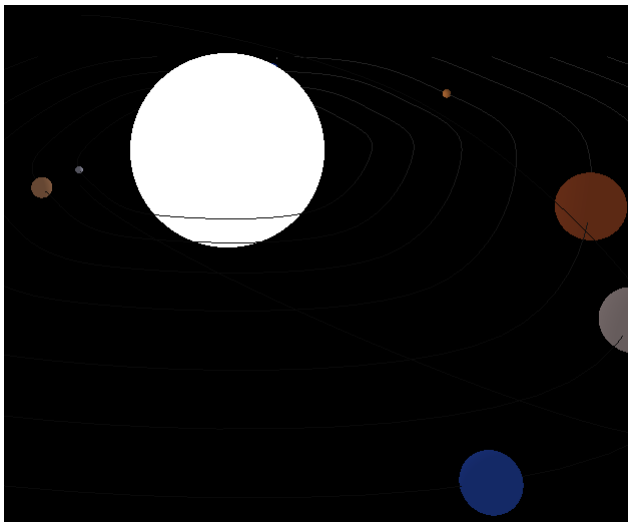

Figure 11: Sistema solar final

Infelizmente não consegui colocar texturas nos planetas, porém vemos que dependendo da posição dos planetas, a luz incidente é alterada, logo a luz é dinâmica.

7 Conclusão

Uma vez que não consegui implementar texturas aos planetas o resultado final ficou insatisfatório, o que acaba por ser a minha maior falha neste projeto inteiro.
Apesar disso, aprendi diversos conceitos sobre *OpenGL* o que me prepara melhor para os testes e para o futuro.
De uma forma geral o trabalho realizado no semestre inteiro foi satisfatório, uma vez que conseguimos identificar com facilidade os planetas com as suas órbitas e luzes.