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 3º ano do Mestrado Integrado em Engenharia Informática.
Nesta fase tive como objectivo criar uma cena dinâmica envolvendo os planetas do sistema solar e as suas orbitas á volta do sol através de modelos criados pelo gerador que foi desenvolvido na 1ª fase aplicando transformações geométricas, entre as quais, translações, rotações e escalas.
Para gerar o modelo dinâmico foi necessário implementar curvas de catmull-rom e bezier patches.
Ao contrário das fases anteriores os modelos têm de ser desenhados com recurso a **VBOs** o que levou a uma adaptação do código já existente.

2 Parser

Mais uma vez o parser do XML foi a primeiro a sofrer alterações, sendo estas devido á necessidade de implementar orbitas. Estas orbitas são definidas no ficheiro de configuração através de 8 pontos no espaço.
Como tal foi necessário implementar uma estrutura que permita guardar o tempo das transformações e os pontos que constituem a sua orbita.
Obviamente que tanto a estrutura **Grupo** e **Transforma** tiveram de ser editados de forma a guardarem esta estrutura.

```
typedef struct orbitas {
    double tempo;
    vector<Point> pontos;
} Orbitas;

typedef struct grupo {
    char* transf;
    double params[4];
    Orbitas* orbita;
    struct grupo* next;
} Grupo;

typedef struct transforma {
    char* transf;
    double params[4];
    Orbitas* orbita;
} Transforma;
```

Figure 1: Estruturas de dados modificadas

Através desta estrutura tenho uma forma de armazenar os dados facilmente, sendo agora necessário preenche-la.
Sempre que o Parser encontra um atributo *"time"* significa que se trata de uma transformação dinâmica, orbita ou rotação, logo todos os pontos disponibilizados irão ser percorridos e guardados um a um no seu espaço. No fim iremos ter uma estrutura onde estão armazenados todos os modelos, suas transformações, orbitas e rotações.

No ficheiro de configuração irá estar presente o atributo *"time"*, que indica o tempo, em segundos, que o modelo irá demorar a percorrer uma curva de Catmull-Rom e os pontos que constituem uma órbita.

```
<group>
    <translate time="1" >
        <point X="0" Y="0" Z="155" /> -- top
        <point X="81" Y="0" Z="132" />
        <point X="155" Y="0" Z="0" /> -- right
        <point X="81" Y="0" Z="-132" />
        <point X="0" Y="0" Z="-155" /> -- down
        <point X="-81" Y="0" Z="-132" />
        <point X="-155" Y="0" Z="0" /> -- left
        <point X="-81" Y="0" Z="132" />
    </translate>
    <scale X="8" Y="8" Z="8"/>
    <colour R="1" G="0.55" B="0"/>
    <models>
        <model file="sphere.3d"/>  --venus
    </models>
</group>
```

Figure 2: Ficheiro de configuração

Estes pontos irão ser, juntamente com o tempo, guardados na estrutura **Orbitas**.

```
if (group->Attribute("time")) {
    char* time = (char*)group->Attribute("time");
    tempo = atof(time);
    for (const XMLElement* translate = group->FirstChildElement(); translate; translate = translate->NextSiblingElement()) {
        transX = (char*)translate->Attribute("X");
        if (transX != NULL) x = atof(transX);
        transY = (char*)translate->Attribute("Y");
        if (transY != NULL) y = atof(transY);
        transZ = (char*)translate->Attribute("Z");
        if (transZ != NULL) z = atof(transZ);
        adicionaP(x, y, z, pontos);
    }
    orbita = new Orbitas();
    orbita->tempo = tempo;
    orbita->pontos = pontos;
```

Figure 3: Algoritmo de leitura de translações dinámicas

3 VBOs

Os VBOs (Vertex Buffer Object) é uma característica do OpenGL que permite um aumento de performance substancial comparado com o método utilizado nas duas primeiras fases. Isto deve-se ao facto de a informação ser agora armazenada na placa gráfica e não na memória do sistema.
Como tal, é necessário preencher os VBOs com os triângulos que constituem os diversos modelos do sistema.

```
while (iPonto != vertices.end()) {
    Ficheiros* fich = *iFicheiros;
    vbo[x] = (double*)malloc(sizeof(double) * fich->numpontos * 3);
    f = true;
    pontos = 0;
    while (iPonto != vertices.end() && f!=false) {
        Point prim = *iPonto;
        iPonto++;
        Point sec = *iPonto;
        iPonto++;
        Point tre = *iPonto;
        iPonto++;
        if (!ver(prim, sec, tre)) {
            vbo[x][pontos++] = prim.x;
            vbo[x][pontos++] = prim.y;
            vbo[x][pontos++] = prim.z;
            vbo[x][pontos++] = sec.x;
            vbo[x][pontos++] = sec.y;
            vbo[x][pontos++] = sec.z;
            vbo[x][pontos++] = tre.x;
            vbo[x][pontos++] = tre.y;
            vbo[x][pontos++] = tre.z;
        }
```

Figure 4: Algoritmo de preenchimento de vbo

Após este processo irei ter uma matriz com todos os pontos a desenhar, onde cada linha da matriz corresponde a um modelo diferente. Por fim é necessário desenhar as VBOs aplicando as transformações, podendo estas ser dinâmicas ou estáticas.

4 Transformações

Para verificar se uma transformação é estática ou dinâmica basta verificar se existe a estrutura de dados "Orbitas", se existir quer dizer que se trata de uma transformação dinâmica, se não, sabemos que se trata de uma transformação estática.

```
if (strcmp((trans->transf), "translate") == 0) {
    if (trans->orbita == nullptr)glTranslatef(trans->params[0], trans->params[1], trans->params[2]);
    else renderCurve(x, trans);
}
```

Figure 5: Translação estática ou dinâmica

5 Curvas de Catmull-Rom

Para construir as órbitas dos planetas foram utilizadas as curvas de Catmull-Rom.

Para tal, extraí a fórmula de Catmull-Rom presente no formulário disponibilizado pelo docente.

$$[t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1 & -2.5 & 2 & -0.5 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -0.5t^3 + t^2 - 0.5t \\ 1.5t^3 - 2.5t^2 + 1 \\ -1.5t^3 + 2t^2 + 0.5t \\ 0.5t^3 - 0.5t^2 \end{bmatrix}^T \tag{24}$$

Figure 6: Fórmula de Catmull-Rom

Através das seguinte matrizes e aplicando a metodologia de cálculo de Catmull-Rom é possivel gerar as orbitas.

```
double matrixT[1][4] = { {powf(t,3),powf(t,2),t,1} };
// Matrix Catmull-Rom
double m[4][4] = { {-0.5f,  1.5f, -1.5f,  0.5f},
                   { 1.0f, -2.5f,  2.0f, -0.5f},
                   {-0.5f,  0.0f,  0.5f,  0.0f},
                   { 0.0f,  1.0f,  0.0f,  0.0f} };
```

Figure 7: Matriz T e Matriz Catmull-Rom

```
void getCatmullRomPoint(float t,
                        float *p0, float *p1, float *p2, float *p3,
                        float *pos, float *deriv) {

    // catmull-rom matrix
    float m[4][4] = {   {-0.5f,  1.5f, -1.5f,  0.5f},
                        { 1.0f, -2.5f,  2.0f, -0.5f},
                        {-0.5f,  0.0f,  0.5f,  0.0f},
                        { 0.0f,  1.0f,  0.0f,  0.0f}};

    // Compute A = M * P
    // for component x P is the vector (p0[0], p1[0], p2[0],p3[0]
    // Compute pos = T * A
    // compute deriv = T' * A
    // ...
}
```

Figure 8: Algoritmo de Catmull-Rom

6 Orbitas

Para a definição das órbitas dos planetas usei como base as distâncias utilizadas na fase anterior, definindo então 8 pontos distintos de forma a formar uma órbita circular à volta do sol. Em todos os pontos a coordenada Y é sempre igual a 0 sendo que as coordenadas X irão variar entre (0,r,-r*cos(45) e r*cos(45)) e Z entre (0,r,-r*sin(45) e r*sin(45)), onde r é a distância de um planeta ao sol (0,0,0).
Nesta fase as órbitas foram desenhadas com a ajuda das curvas Catmull-Rom.

```
glBegin(GL_LINE_LOOP);
for (int i = 0; i < npontos; i++) {
    getGlobalCatmullRomPoint((double)i / npontos, position, derivada, curva, tamanho);
    glColor3f(0.1, 0.1, 0.1);
    glVertex3f(position[0], position[1], position[2]);
}
glEnd();
```

Figure 9: Ciclo das orbitas

7 Cometa

Infelizmente não consegui implementar os Patches de Bezier, como tal usei o modelo "box" criado na 1ª fase. A este cometa estão associados 4 pontos que formam a sua trajétoria, sendo que foi também implementado um movimento de rotação dinâmico.

```
<group>
    <translate time="2" >
        <point X="400" Y="0" Z="0" />
        <point X="-200" Y="-120" Z="40" />
        <point X="-400" Y="0" Z="0" />
        <point X="200" Y="120" Z="0" />
    </translate>
    <rotate time="0.5" X="10" Y="10" Z="10"/> -- Rotate with time
    <scale X="0.8" Y="0.8" Z="0.8" />
    <color R="1" G="0.6" B="0.3" />
    <models>
        <model file="box.3d" /> -- Comet
    </models>
</group>
```

Figure 10: Configuração do cometa

É então possível verificar a orbita do cometa, infelizmente é impossível demonstrar a rotação aplicada ao cometa numa imagem.

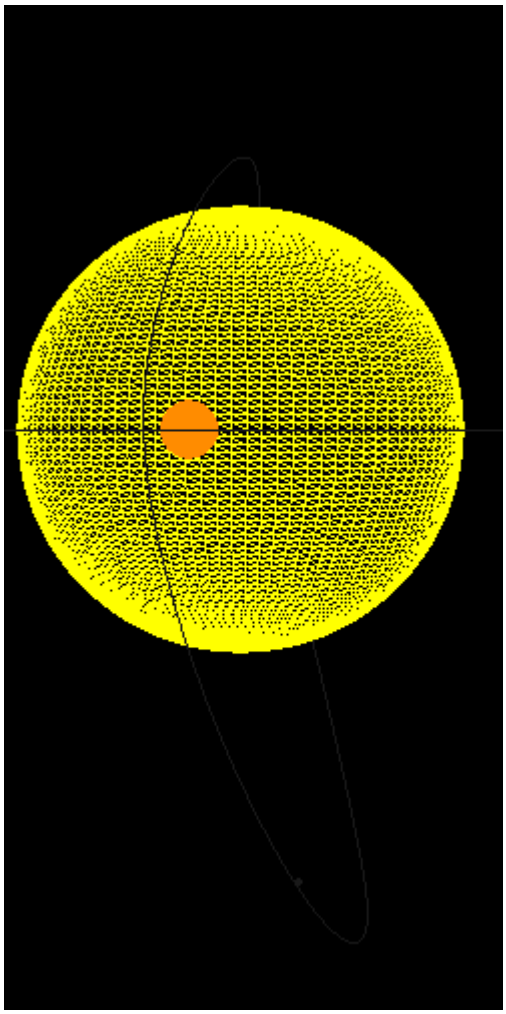

Figure 11: Órbita do cometa

8 Resultado final

Por fim temos a representação do sistema solar, contendo o sol e os 8 planetas e suas orbitas, mais uma vez não está á escala e as órbitas não são elípticas como na vida real.

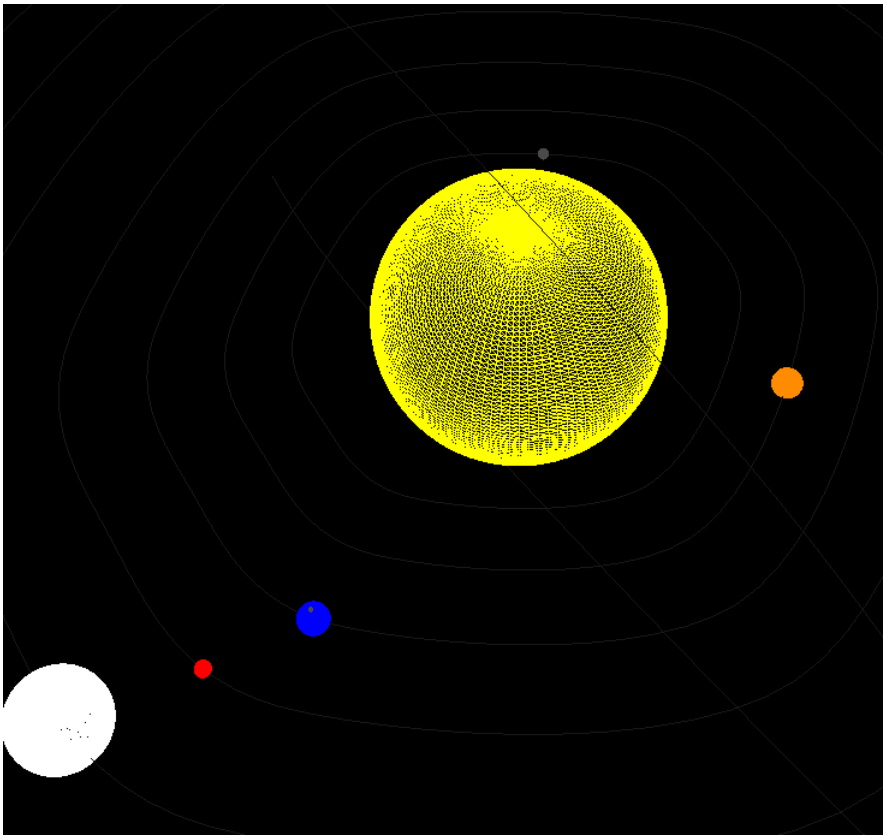

Figure 12: Orbitas implementadas no sistema solar

9 Conclusão

Esta terceira fase do trabalho prático de Computação Gráfica foi consideravelmente mais demorosa, tendo sentido maior dificuldade devido á quantidade de objetivos a completar.
Nesta fase consegui implementar VBOs e transformações dinámicas. Infelizmente não consegui implementar os Bezier Patches, um ponto negativo a ter em conta.

Pondo isto, podemos dizer que a minha prática em C++ foi um pouco melhorada com este projeto até então, bem como os conhecimentos em OpenGL e que dou o objetivo principal como cumprido.