Universidade do Minho
Escola de Engenharia

Universidade do Minho
Departamento de Informática
Computação Gráfica

Trabalho Prático Nº 3

Diogo Araujo (A89517)
António Silva (A89558)
Pedro Novais (A78211)
Gonçalo Soares (A84441)

30 de maio de 2021

Diogo Araújo

António Silva

Gonçalo Soares

Pedro Novais

Conteúdo

Fase 4 – Normais e Texturas

1.1 Introdução

Para esta fase do projeto, foi-nos pedido a continuação do desenvolvimento do sistema solar com a aplicação de iluminação e texturas.

O Generator deveria assim gerar coordenadas de textura e normais para cada vértice.

A aplicação de texturas e iluminaçáo aos modelos apresentados no sistema solar foi, á semelhança das fases anteriores, feita através da leitura de um ficheiro XML onde deveria ser possível definir as fontes de ilumínção. O presente relatório visa esclarecer, elucidar e demonstrar as soluções que o grupo encontrou para desenvolver a aplicação pedida.

1.2 Generator

O propósito do generator nesta fase é criar um ficheiro do tipo ".3d"pronto para ser lido pela *engine*. Nesta fase, além de gerar o que é pedido nas fases anteriores, temos de calcular as normais de todos os vértices do objeto. Para calcularmos as normais da esfera, "normalizamos"todos os pontos, isto é, dividimos cada ponto por o seu raio.

Quanto à geração das normais do bulé de chá, usamos as seguintes formulas:

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

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

Figura 1.1: Fórmula para calcular normais dos pontos de Bezier

O seguinte código gera o produto dos vetores de todos os pontos de um patch de bezier:

```
vector<vertice> createDUPatch(bezierPatch bp, vector<int> indices, int tesselation)
{
    vector<vector<float>> mMatrix = {{ -1.0f, 3.0f, -3.0f, 1.0f},
                                     { 3.0f, -6.0f, 3.0f, 0.0f},
                                     { -3.0f, 3.0f, 0.0f, 0.0f},
                                     { 1.0f, 0.0f, 0.0f, 0.0f}};
    vector<vector<float>> cpX = convertIndicesToMatrix(bp,indices,0);
    vector<vector<float>> cpY = convertIndicesToMatrix(bp,indices,1);
    vector<vector<float>> cpZ = convertIndicesToMatrix(bp,indices,2);

    vector<vertice> res;

    for(int i = 0; i <= tesselation; i++)
    {
        //pr pio ponto:
        //printf("%f %f %f", );
        for(int j = 0; j <= tesselation; j++)
        {

            float u = float (i) / float(tesselation);
            float v = float (j) / float(tesselation);
            vector<float> uMatrix = {u*u*u, u*u,u,1.0};
            vector<float> duMatrix = {3*u*u, 2*u,1.0,0.0};
            vector<float> vMatrix = {v*v*v, v*v,v,1.0};
            vector<float> dvMatrix = {3*v*v, 2*v,1.0,0.0};
            //dU plus M Matrix
            vector<float> dum_Matrix = multMatrix(duMatrix,mMatrix);
            //U plus M Matrix
            vector<float> um_Matrix = multMatrix(uMatrix,mMatrix);
            //dU plus M plus CpX/CpY/CpZ Matrix
            vector<float> dumcpX_Matrix = multMatrix(dum_Matrix,cpX);
            vector<float> dumcpY_Matrix = multMatrix(dum_Matrix,cpY);
            vector<float> dumcpZ_Matrix = multMatrix(dum_Matrix,cpZ);
            //U plus M plus CpX/CpY/CpZ Matrix
            vector<float> umcpX_Matrix = multMatrix(um_Matrix,cpX);
            vector<float> umcpY_Matrix = multMatrix(um_Matrix,cpY);
            vector<float> umcpZ_Matrix = multMatrix(um_Matrix,cpZ);

            //dU plus M plus CpX plus Mt Matrix
            vector<float> dumcpXm_Matrix = multMatrix(dumcpX_Matrix,mMatrix);
            vector<float> dumcpYm_Matrix = multMatrix(dumcpY_Matrix,mMatrix);
            vector<float> dumcpZm_Matrix = multMatrix(dumcpZ_Matrix,mMatrix);

            //U plus M plus CpX plus Mt Matrix
            vector<float> umcpXm_Matrix = multMatrix(umcpX_Matrix,mMatrix);
            vector<float> umcpYm_Matrix = multMatrix(umcpY_Matrix,mMatrix);
            vector<float> umcpZm_Matrix = multMatrix(umcpZ_Matrix,mMatrix);


            //dU plus M plus CpX plus Mt plux V Matrix
            vertice ver1, ver2;
            ver1.x = dumcpXm_Matrix[0]*vMatrix[0] + dumcpXm_Matrix[1]*vMatrix[1] +
```

```
                    dumcpXm_Matrix[2]*vMatrix[2] + dumcpXm_Matrix[3]*vMatrix[3];
            ver1.y = dumcpYm_Matrix[0]*vMatrix[0] + dumcpYm_Matrix[1]*vMatrix[1] +
                    dumcpYm_Matrix[2]*vMatrix[2] + dumcpYm_Matrix[3]*vMatrix[3];
            ver1.z = dumcpZm_Matrix[0]*vMatrix[0] + dumcpZm_Matrix[1]*vMatrix[1] +
                    dumcpZm_Matrix[2]*vMatrix[2] + dumcpZm_Matrix[3]*vMatrix[3];

            ver2.x = umcpXm_Matrix[0]*dvMatrix[0] + umcpXm_Matrix[1]*dvMatrix[1] +
                     umcpXm_Matrix[2]*dvMatrix[2] + umcpXm_Matrix[3]*dvMatrix[3];
            ver2.y = umcpYm_Matrix[0]*dvMatrix[0] + umcpYm_Matrix[1]*dvMatrix[1] +
                     umcpYm_Matrix[2]*dvMatrix[2] + umcpYm_Matrix[3]*dvMatrix[3];
            ver2.z = umcpZm_Matrix[0]*dvMatrix[0] + umcpZm_Matrix[1]*dvMatrix[1] +
                     umcpZm_Matrix[2]*dvMatrix[2] + umcpZm_Matrix[3]*dvMatrix[3];

            vertice vect = normalizeV(ver1, ver2);
            //printf("%f %f %f\n", vect.x, vect.y, vect.z);
            res.push_back(vect);
        }
    }
    return res;
}
```

1.3 Engine

Na engine verificamos algumas alterações, criamos uma nova estrutura luz para fazer parse e guardar a informação:

```
struct lights
{
    GLenum num;
    char* type;
    vertice pos;
};
```

Além disto inicializamos e corremos mais duas VBOs, uma para as normais e outra para as coordenadas das texturas, esta no entanto não se encontra completamente correta.

```
glGenBuffers(1,&(globalFigs->vbos[contador-1].normal));
glBindBuffer(GL_ARRAY_BUFFER, globalFigs->vbos[contador-1].normal);
glBufferData(GL_ARRAY_BUFFER, sizeof(float)*norm.size(), norm.data(),GL_STATIC_DRAW);
```

glBindBuffer(GL$_{A}RRAY_{B}UFFER, globalFigs- > vbos[i].normal); $glNormalPointer(GL_{F}LOAT, 0, 0)$;

1.4 Resultado obtido

O trabalho sofreu bastantes alterações desde a última fase pois passou de um sistema sem luz e planetas sem corpo para um sistema com luz e texturas. No entanto, as texturas não ficaram de todo como o desejado, tendo cada planeta apenas parte da textura como é visivel nas seguintes imagens:

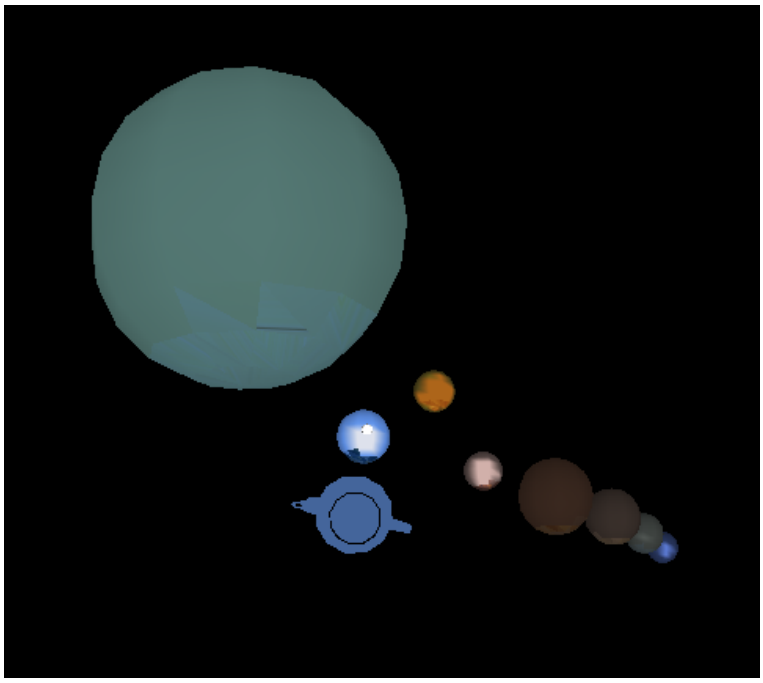

Figura 1.2: Exemplo de Sistema Solar gerado

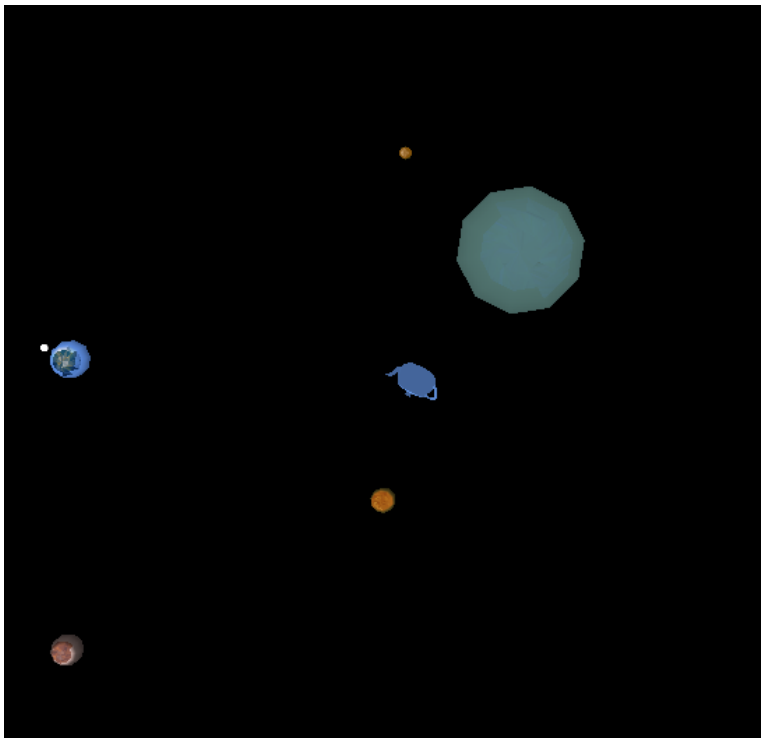

Figura 1.3: Exemplo de Sistema Solar gerado

1.5 Conclusão

Concluída esta quarta e última fase do trabalho prático, e tendo por base que tinha sido pedido a aplicação de texturas e iluminação aos modelos que compõe o sistema solar, o grupo apenas conseguiu dar resposta á iluminação, tendo ainda tentado desenvolver a parte das texturas. Este último requisito não se apresentada funcional pois o tempo revelou-se curto para o grupo. Tendo, as outras fases, sido bem sucedidas, chegamos a esta fase do projeto sem ter que alterar grande parte do código, o que se revela de todo um aspeto positivo.

A aplicação da iluminação dá, sem dúvida outro aspeto ao sistema desenvolvido tornando-o mais apelativo e o mais parecido ao sistema solar tal como o conhecemos.

Senfo esta a última fase da parte prática da UC Computação Gráfica, o grupo sente que o trabalho desenvolvido como um todo serviu para aprofundar todos os conhecimento relativos aos conceitos quer teóricos quer práticos que a disciplina nos disponibiliza.

Este trabalho introduziu dois termos teóricos lecionados nesta unidade curricular sendo estes as curvas de *Bezier* e *CatmullRom*. A maior dificuldade a superar foram as estruturas usadas nas fases anteriores onde se mostraram muito incompatíveis com as VBO's e a sua implementação, sendo então a *engine* a fase mais demorada deste projeto. No que toca às rotações a implementação foi bastante simples pois apenas alteramos o ângulo consoante o tempo o que levou a poucas alterações no código.

Em relação às translações, o processo foi mais demorado e complexo no entanto, o código está baseado no que foi feito nas aulas práticas onde se aplicou as curvas de *CatmullRom* sendo possível assim fazer translações amicas.