Relatório do Projeto de CG

João Pimentel (a80874) Rodolfo Silva (a81716)
Pedro Gonçalves (a82313)

Abril 2019

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Computação Gráfica
Grupo 25
3ª Fase
Curvas, Superfícies Cúbicas e VBOs

Conteúdo

Lista de Figuras

1 Introdução

O presente relatório consiste na descrição da terceira fase da implementação de um mini cenário gráfico 3D que representa um modelo dinâmico do sistema solar.

Nesta etapa, existem três grandes objetivos: o desenho dos modelos com base em *VBOs* (Vertex Buffer Objects), permitir que um modelo seja gerado através de um *patch* de Bezier e animar o, até então, sistema solar estático.

Gerar um modelo através de *patch* prende-se com a utilização de um conjunto de fórmulas, levando à construção de um *teapot*, no caso do projeto. Já a animação do sistema solar é baseada na utilização de curvas de Catmull-Rom e rotações baseadas em tempo.

Por fim, é pedido que seja, ainda, introduzido um cometa ao sistema, sendo este retratado por um *teapot*.

2 Estruturas de Dados e Classes

As alterações no formato dos ficheiros *XML* levaram a que fossem armazenados novos tipos de dados, nomeadamente tempos de translação e/ou rotação, bem como possíveis pontos de controlo, utilizados para retratar a trajetória de uma figura ao longo do tempo. Esta informação adicional levou a que as classes e estruturas definidas para as fases anteriores fossem aperfeiçoadas de modo a conseguir retratar o pedido.

Assim, a classe *figura* passou a possuir mais atributos, nomeadamente, um vetor que possui os pontos de controlo da trajetória ($vector<TPS>$ *translates*), o tempo de translação ($trTime$) e o tempo de rotação ($rTime$), como se vê na Figura 1.

Como não é obrigatório que uma figura possua transformações com base em tempo, foi definido que o valor temporal associado a estes casos seria zero e o vetor de pontos ficaria vazio, nestas situações.

```
class figura {
private:
    int filhos;

    int total;

    float trTime;
    std::vector<TPS> translates;
    float trX;
    float trY;
    float trZ;
    int T_order;

    float rTime;
    float rA;
    float rX;
    float rY;
    float rZ;
    int R_order;

    float sX;
    float sY;
    float sZ;
    int S_order;
```

Figura 1 - Classe *figura*.

3 VBOs

Os *VBOs* oferecem um aumento na performance em comparação com o modo de *rendering* imediato, devido ao facto de os dados serem armazenados na memória gráfica em vez de na memória do sistema, sendo diretamente renderizados pela *GPU*.

Extraída a informação relativa a uma figura, neste caso os seus vértices, os quais provêm de um ficheiro com extensão *3d*, esta é utilizada para inicializar e preencher o *buffer* relativo à figura em questão. Note-se que foi implementado um *VBO* por figura, pois estas são independentes umas das outras, na sua maioria.

Logo após a leitura do ficheiro com os pontos, estes ficam guardados num vetor. Este último é percorrido, inserindo os vértices num *array* que é utilizado no *VBO* da figura em questão. Este processo ocorre na função de *load* do objeto, sendo que, após a execução de carregamento de todas as figuras, o array de *VBOs* está finalizado e pronto a desenhar.

4 Patches de Bezier

De modo a conseguir desenhar um *teapot* a partir de um *patch* de Bezier, teve que ser definido um mecanismo que, recebendo um ficheiro, e um nível de tesselação, gerasse um ficheiro de pontos.

O primeiro passo consistiu em extrair a informação presente no ficheiro *patch*, tendo sido desenvolvido um *parser* que guarda os pontos e respetivos *patches* na forma de um *array* de índices. Ou seja, além de um *array* com todos os pontos existentes no ficheiro e o número total destes, existe, ainda, uma matriz de índices relativos aos *patches* a desenhar. Esta matriz é declarada como *indices*[n][ ], sendo que n é o número de patches indicados na primeira linha do ficheiro.

Em seguida, foram guardados os pontos de controlo num *array* de pontos (estrutura que possui coordenadas x,y e z).

Finzalizado o *parse* do ficheiro, começa o processo de gerar a figura em si. Tendo em conta as equações 1 e 2, o processo torna-se bastante simples, tendo sido apenas necessário codificar as mesmas.

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

$$M = \begin{bmatrix} -1.0f & 3.0 & -3.0 & 1.0 \\ 3.0 & -6.0 & 3.0 & 0.0 \\ -3.0 & 3.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix} \quad (2)$$

Assim, chamando, quando necessário, a função que devolve um ponto de Bezier, o processo de formação da figura torna-se algo iterativo. De notar que, quanto mais elevado o nível de tesselação (variável *divs*), maior será o detalhe da figura a ser desenhada, como se pode verificar pela Figura 2.

```
for (int i = 0; i < patches; i++) {
    for (int j = 0; j < 16; j++) {
        pv[j] = cpoints[indexes[i][j]];
    }
    for (int u = 0; u < divs; u++) {
        float p1[3];
        float p2[3];
        float p3[3];
        float p4[3];
        for (int v = 0; v < divs; v++) {
            get_ponto_bezier(u / (float)divs, v / (float)divs, pv, p1);
            get_ponto_bezier((u + 1) / (float)divs, v / (float)divs, pv, p2);
            get_ponto_bezier(u / (float)divs, (v + 1) / (float)divs, pv, p3);
            get_ponto_bezier((u + 1) / (float)divs, (v + 1) / (float)divs, pv, p4);

            pontos << p1[0] << ' ' << p1[1] << ' ' << p1[2] << '\n';
            pontos << p3[0] << ' ' << p3[1] << ' ' << p3[2] << '\n';
            pontos << p4[0] << ' ' << p4[1] << ' ' << p4[2] << '\n';

            pontos << p2[0] << ' ' << p2[1] << ' ' << p2[2] << '\n';
            pontos << p1[0] << ' ' << p1[1] << ' ' << p1[2] << '\n';
            pontos << p4[0] << ' ' << p4[1] << ' ' << p4[2] << '\n';

            linhas += 6;
        }
    }
}
```

Figura 2 - Escrita em ficheiro dos pontos de Bezier.

Dito isto, o resultado obtido para um nível de tesselação igual a 10, a partir do ficheiro fornecido pelo docente [1], está visível na Figura 3.

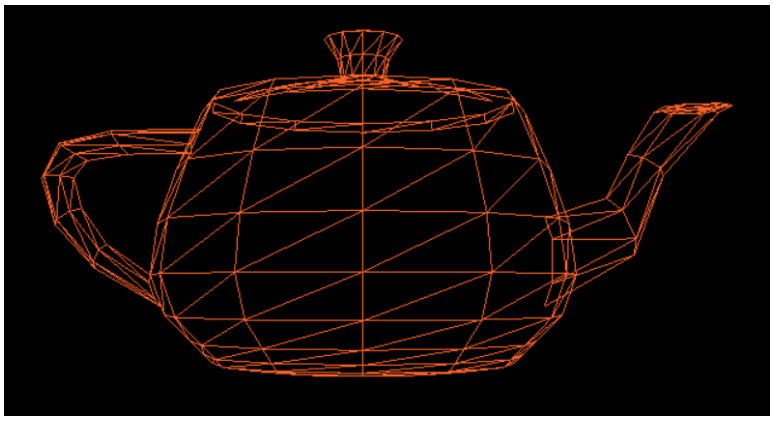

Figura 3 - *Teapot*.

5 Catmull-Rom

5.1 Curvas

Partindo da codificação das fórmulas fornecidas pela docente [1], o processo de codificação destas curvas tornou-se mais simples, sendo que a base deste processo reside na equação 3.

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

Assim, de modo a desenhar as trajetórias dos objetos, nomeadamente dos planetas, inicialmente, foram armazenados os pontos presentes no ficheiro XML. Através da utilização de um ciclo e da função *getGlobalCatmullRomPoint*, é possível obter, não só as coordenadas do próximo ponto, mas também definir uma trajetória baseadas nestes. Neste caso, optou-se por definir que uma curva teria 100 pontos, sendo este o valor do número de iterações do ciclo. Assim, aplicando um método que ligue estes pontos, a trajetória fica desenhada como pretendido.

Note-se, ainda, que, de modo a obter uma trajetória circular, foram utilizados 12 pontos de controlo por cada trajeto, baseados na Figura 4.

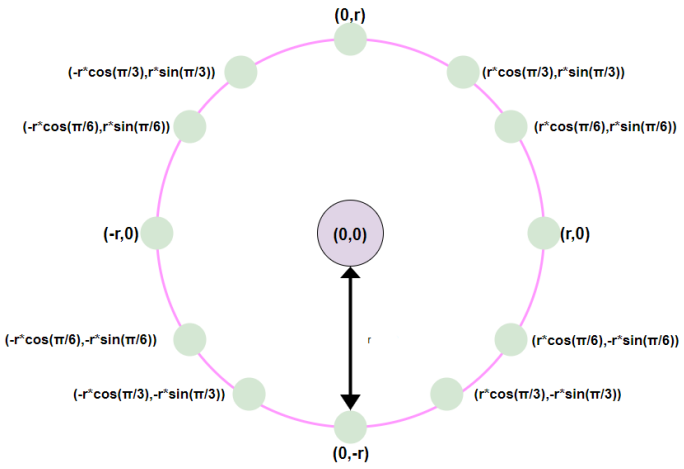

Figura 4 - Pontos de Controlo da Curva de Catmull-Rom.

5.2 Rotate

A variável temporal presente no ficheiro XML indica o tempo que a figura deve demorar a fazer uma rotação de $360^{\underline{o}}$ em torno de si mesma. Assim, a cada momento que passe o seu ângulo irá incrementar pela razão $360/(time * 1000)$, sendo *time* o valor lido no ficheiro (multiplicado por 1000 para ficar em milissegundos). Caso se aumente ao valor do tempo, a rotação ficará mais lenta e vice-versa.

5.3 Translate

Seguindo um pensamento semelhante ao utilizado para gerar as linhas das curvas, de modo a colocar uma figura num determinado ponto da trajetória, num dado momento, utilizou-se a função *getGlobalCatmullRomPoint*. Através desta foi possível obter o ponto na curva a efetuar a deslocação. Assim, a cada execução da função *renderScene*, o tempo decorrido até ao momento irá aumentar, levando a que o objeto se desloque ao longo da sua trajetória.

6 Sistema Solar Dinâmico

Tendo em conta que nesta fase o sistema solar teria que estar animado, o ficheiro XML teve que ver as suas transformações geométricas a depender do tempo. Este foi implementado de forma a ser próximo da realidade, possuindo pequenas alterações, de modo a permitir uma maior facilidade na visualização do projeto em execução. O mesmo ocorreu com as distâncias entre planetas e dimensões dos mesmos.

Houve, ainda, a preocupação de adicionar um cometa, representado pelo *teapot* obtido através do *patch* de Bezier, com uma órbita elíptica, como se vê nas Figuras 5 a 7.

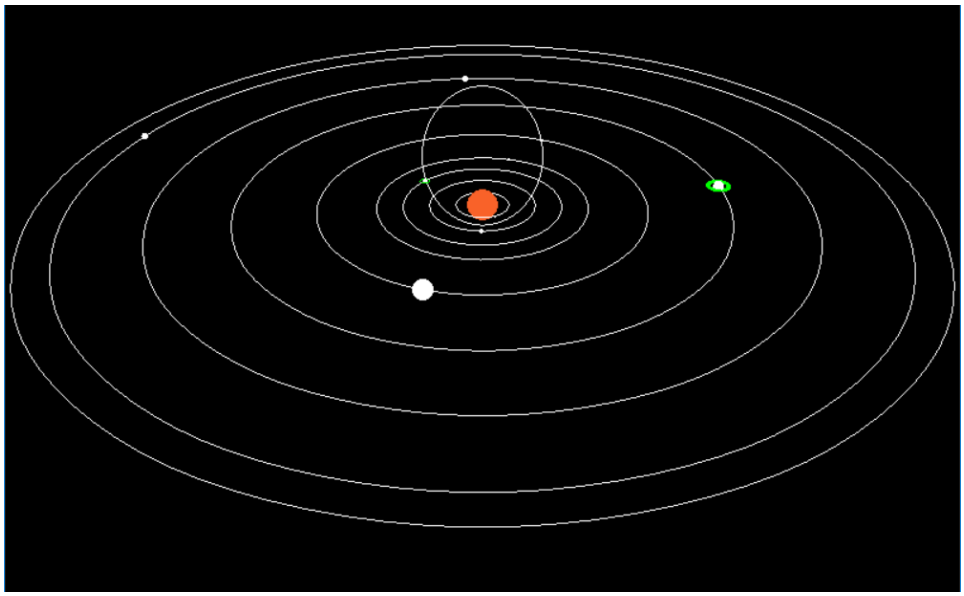

Figura 5 - Sistema Solar.

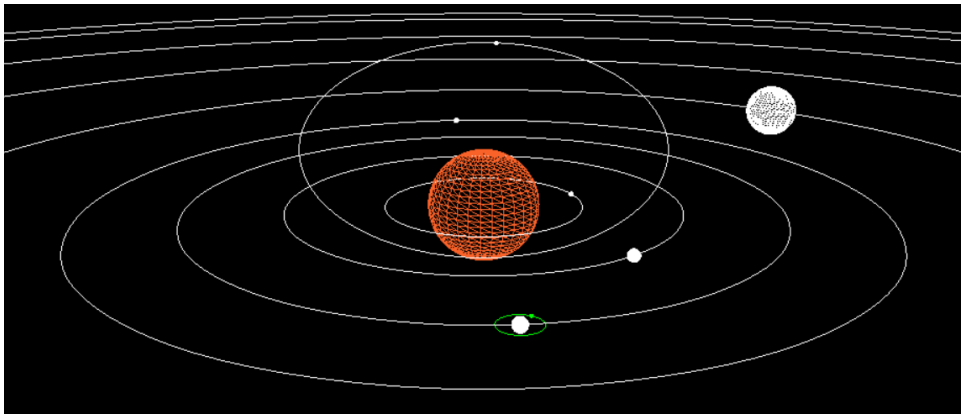

Figura 6 - Sistema Solar: Perspetiva Aproximada.

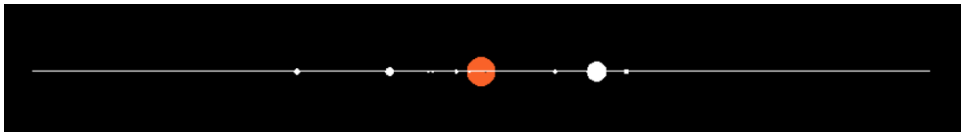

Figura 7 - Sistema Solar: Perspetiva Lateral.

7 Conclusões e Trabalho Futuro

Durante a conceção desta fase, o grupo deparou-se com algumas dificuldades, especialmente no que toca à elaboração de uma figura a partir de um *patch* de Bezier. Tal deveu-se ao facto de ser a primeira interação com este tipo de ficheiros, levando a que tivesse de ser efetuada uma pesquisa prévia à sua codificação. Apesar disto, após a pesquisa, o algoritmo parece bastante capaz de resolver os exemplos que lhe são passados, mostrando-se uma execução com sucesso.

No que toca às curvas de Catmull-Rom e aos *VBOs*, devido às interações desenvolvidas no decorrer das aulas, estas etapas mostraram-se mais simples, sendo mais rápida a sua implementação no projeto.

Assim, é credível que o resultado obtido cumpriu os requisitos apresentados no projeto, sendo possível desenvolver um modelo dinâmico do Sistema Solar, incluindo uma representação de um cometa gerada a partir de um *patch*. Dito isto, é esperado que a fase final deste projeto permita obter um resultado mais realista e visivelmente agradável, com a inserção de texturas e iluminação.

Referências

[1] António Ramires. *Curves and Surfaces: Notes for an Undergraduate Course in Computer Graphics*. University of Minho, 2019.