Computação Gráfica

3ª Fase – Curvas, Superfícies e *VBO's*

André Geraldes 67673

Patrícia Barros 67665

Sandra Ferreira 67709

Conteúdo

Índice de Figuras

Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Computação Gráfica, pertencente ao plano de estudos do 3º ano da licenciatura em Engenharia Informática.

Este projeto será constituído por 4 fases distintas com o objetivo final de criar um *motor 3D*. Nesta terceira fase foi-nos proposto fazer algumas alterações no trabalho desenvolvido anteriormente, nomeadamente a implementação de translações definidas por pontos de uma curva e por tempo, a adaptação da rotação para ser também associada ao tempo e ainda a utilização de *VBOs*.

Desenvolvimento

Superfícies de *Bézier*

As superfícies de *Bézier* são definidas por um conjunto de pontos de controlo na direção dos quais são esticadas como se houvesse uma força atrativa.

Para a implementação desta funcionalidade foi necessário alterar o nosso Gerador para que este seja capaz de definir e criar listas de triângulos correspondentes às mesmas. Para isto o gerador recebe um ficheiro com informação relativa aos pontos de controlo e também o grau de tesselação pretendido e gera, à semelhança do que acontece nas outras primitivas, um ficheiro com a lista de triângulos correspondentes.

Foi então necessário criar a função `void readPatch(string path)` para fazer a leitura do ficheiro *.patch*. Esta função guarda os *patches* num objeto da classe **Patch** que tem um vetor de inteiros, e os pontos de controlo num vetor de objetos da classe **Pontos** que guarda os valores das três coordenadas de um ponto.

De seguida criámos a função `Ponto calcular(float t, float *p1, float *p2, float *p3, float *p4)` que dado os pontos de controlo de uma superfície de *Bézier* calcula as coordenadas de um ponto, utilizando a seguinte fórmula:

$$\mathbf{B}(t) = (1-t)^3\mathbf{B}_0 + 3t(1-t)^2\mathbf{B}_1 + 3t^2(1-t)\mathbf{B}_2 + t^3\mathbf{B}_3 \ , \ t \in [0,1].$$

Figura 1 - Equação para cálculo de curvas de Bézier

A função `Ponto bezier(float u, float v, vector<int> pat)` que percorre a lista de 16 inteiros (*patch*) e de 4 em 4 utilizando a função descrita acima calcula um ponto. Depois de calculados desta forma os 4 pontos é calculado o ponto final do *patch*.

Foi ainda necessária a criação da função `void patchBezier(int tess, int ip, ofstream& file)` que para cada incremento num ciclo que vai desde 0 até ao grau de tesselação obtém 4 pontos que formam 2 triângulos que de seguida são escritos no ficheiro passado como argumento.

A função `void initSupBezier(int tess, string nameFile)` dá início a todo o processo descrito acima, chamando a função *patchBezier* para o ficheiro e grau de tesselação passado como argumento.

Desta forma e utilizando o ficheiro *.patch* fornecido pelo professor foi possível criar um *Teapot* formado por superfícies de *Bézier* que servirá para desenhar um cometa em forma de bule de chá no nosso sistema solar.

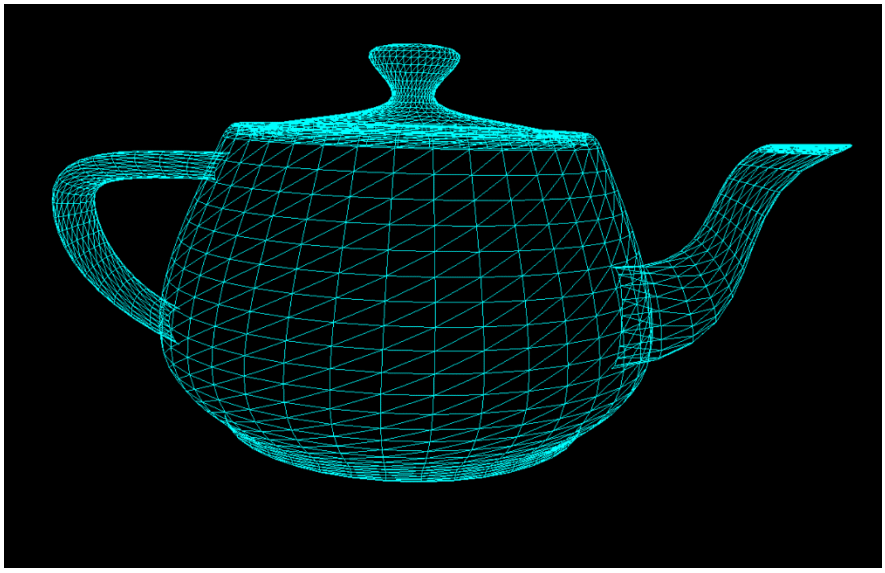

Figura 2 - *Teapot* gerado através de curvas de *Bézier*

Rotações e Translações

Para a implementação da nova forma de translação foram necessárias algumas alterações no Motor3D e também nas classes **Rotação** e **Translação** criadas na fase anterior. Na classe **Rotação** a única alteração foi retirar a variável relativa ao ângulo e acrescentar uma variável relativa ao tempo.

```
class Rotacao{
       float time;
       float eixoX;
       float eixoY;
       float eixoZ;
```

Para as rotações a única alteração no Motor3D foi na parte relativa à rotação na função *renderScene:*

```
float r = glutGet(GLUT_ELAPSED_TIME) % (int)(t.getRotacao().getTime() * 1000);
float gr = (r * 360) / (t.getRotacao().getTime() * 1000);
glRotatef(gr,       t.getRotacao().geteixoX(),       t.getRotacao().geteixoY(),
t.getRotacao().geteixoZ());
```

Já a classe **Translação** agora guarda para além do tempo, um vetor de pontos referentes aos pontos de controlo da curva e um vetor referente aos pontos finais da translação (depois de calculada a curva).

```
class Translacao{
       float time;
       int tam;
       vector<Ponto> pontosTrans;
       vector<Ponto> pontosCurva;
```

Para além disso, na classe **Translação** foram definidos métodos relativos às curvas *Catmull-Rom*, baseados na informação que o professor disponibilizou pelo assunto nos apontamentos da disciplina.

VBOs

Os *VBOs* (***Vertex Buffer Objects***) visam melhorar a performance do *OpenGL* e consistem em armazenar os triângulos e as suas informações e que depois são passadas à memória da placa para que esta depois os desenhe, libertando assim o processador de carga desnecessária. Através deste método só carregamos os pontos para o *Buffer* uma vez para serem posteriormente desenhados, sem que haja mais algum custo para o *OpenGL*. De notar que os pontos tem de ser adicionados ao *Buffer* seguindo a sua ordem especifica, caso contrário os pontos associaram-se de forma diferente da que desejamos e formariam triângulos que não os pretendidos.

Posto isto resta explicar como foi implementada esta tecnologia no nosso trabalho. As principais alterações aconteceram na classe **Primitiva** que possuí agora três novos métodos:

- O método **construir**, que através do GL_TRIANGLES constrói todos os triângulos correspondentes aos pontos que se encontram no vetor desse objeto de modo imediato;
- O método **preparar**, que cria e preenche um *array* com os pontos dos triângulos e de seguida cria um *Buffer* e preenche-o de forma ordenada com os dados do *array* criado previamente;
- O método **desenhar**, que indo buscar a informação ao *Buffer* desenha os triângulos guardados.

```
public:
	Primitiva();
	Primitiva(string n, vector<Primitiva>, vector<Ponto>,
Transformacao);
	string getNome(){ return nome; }
	vector<Ponto> getPontos() { return pontos; }
	vector<Primitiva> getFilhos(){ return filhos; }
	Transformacao getTransformacao(){ return transformacao; }
	void setNome(string n){ nome = n; }
	void setFilhos(vector<Primitiva> f){ filhos = f; }
	void setPontos(vector<Ponto> p){ pontos = p; }
	void setTransformacao(Transformacao t){ transformacao = t; }
	void preparar();
	void desenhar();
	void construir();
	virtual ~Primitiva() {};
};
```

No Motor3D, na função *renderScene* são então chamados os métodos acima descritos, e ficam assim implementados os *VBOs* no nosso projeto.

Análise de Resultados

Passamos então a mostrar os resultados que obtivemos com o nosso Motor3D para o ficheiro XML que criamos com o Sistema Solar. Este ficheiro foi melhorado relativamente ao realizado na segunda fase, e incluí o Sol, oito planetas, a nossa lua, as luas de Saturno e um cometa em forma de bule de chá construído à custa de *patches* de *Bézier*.

As escalas e rotações dos planetas foram calculadas tomando como referência os valores reais, aproximadamente.

As translações relativas às órbitas dos planetas e do cometa foram geradas utilizando coordenadas polares.

```
ofstream file("pontos.txt");
double ang = 0;
for (int xq = 0; xq < 16; xq++){
    file << "\t\t\t\t\t" << "<ponto X=\"" << 35 * cos(ang) << "\" Y=\"" << 0 << "\" Z=\"" << 35 * sin(ang) << "\" />" << endl;
    ang += M_PI / 8;
}
```

Figura 3 - Função utilizada para calcular os pontos das órbitas dos planetas

```
ofstream file("pontos.txt");
double ang = 0;
for (int xq = 0; xq < 16; xq++){
    file << "<ponto X=\"" << 50 * cos(ang) << "\" Y=\"" << 0 << "\" Z=\"" << 250 * sin(ang)-230 << "\" />" << endl;
    ang += M_PI / 8;
}
```

Figura 4 - Função utilizada para calcular os pontos da órbita do cometa

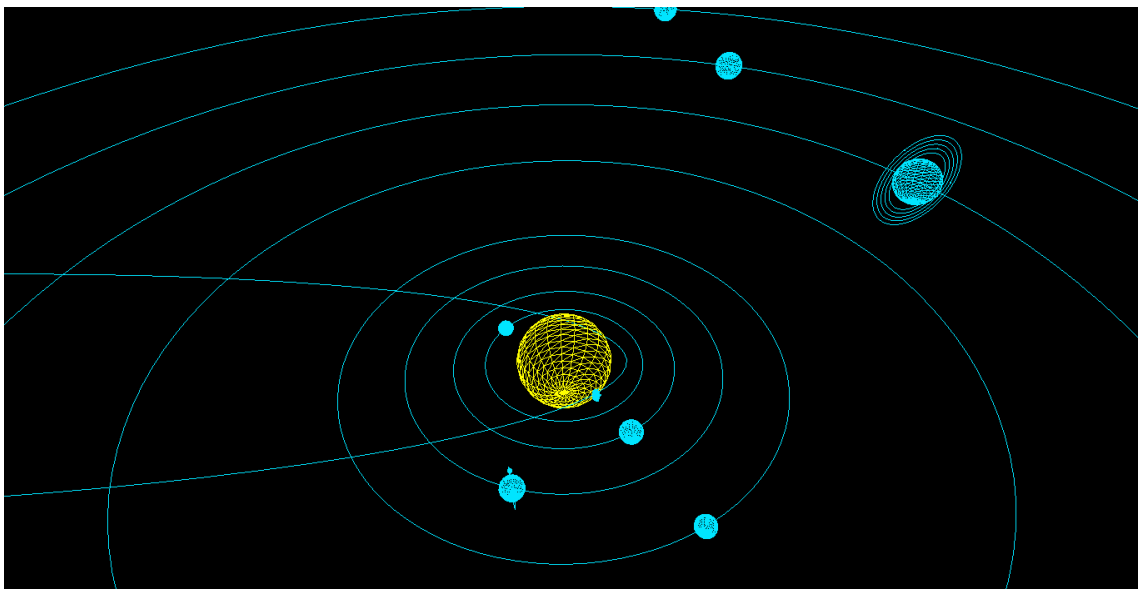

Figura 5 - Perspetiva do Sistema Solar visto "de cima"

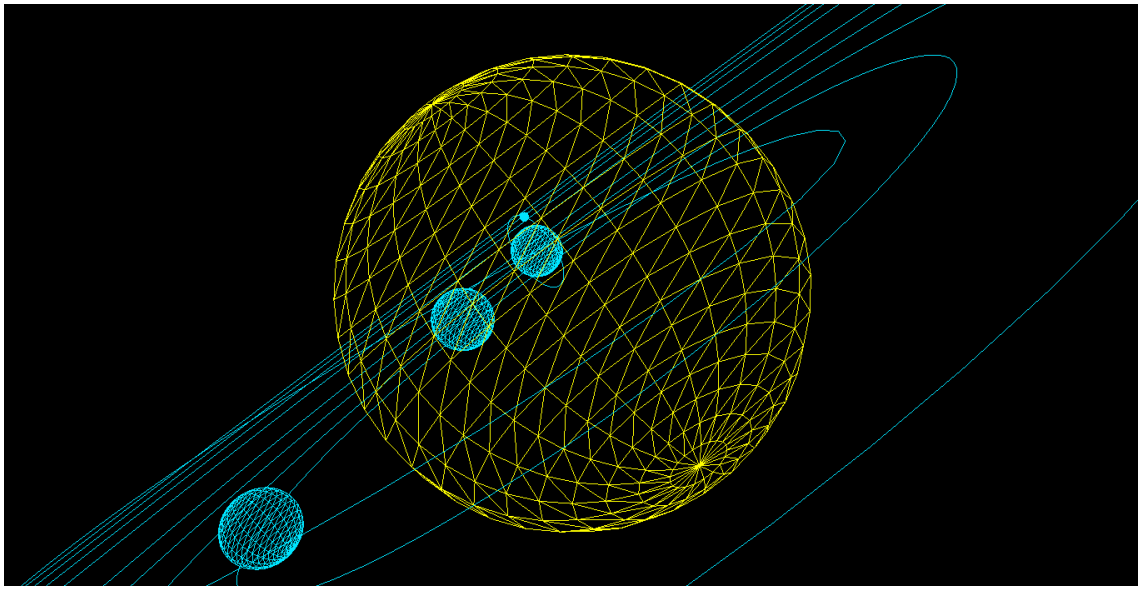

Figura 6 - Perspetiva do Sistema solar visto de perto do Sol

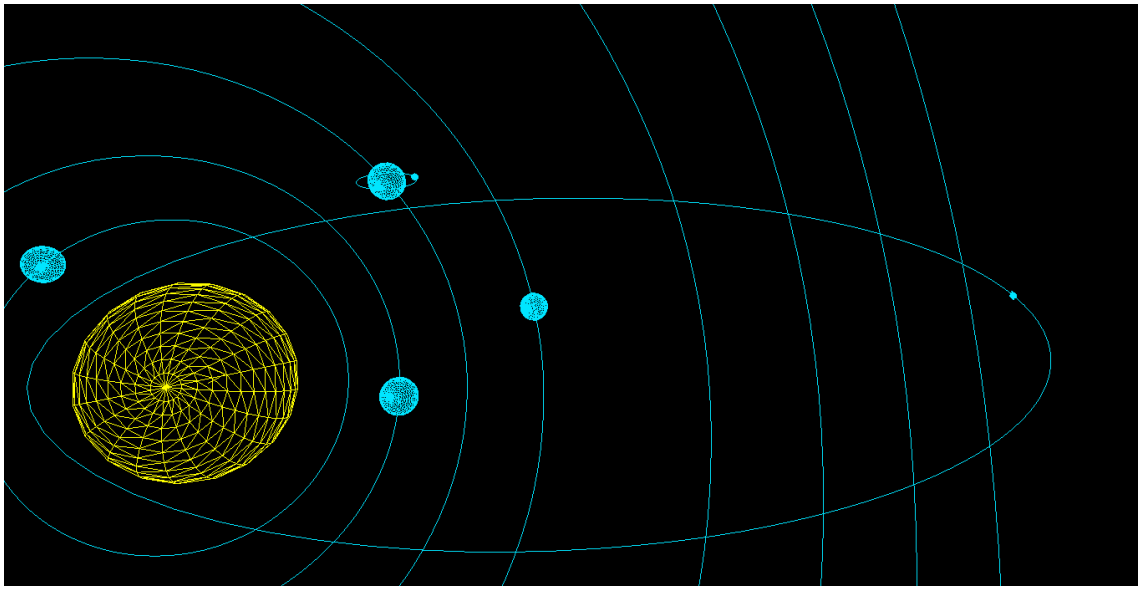

Figura 7 - Perspetiva do Sistema Solar estando visível a órbita do cometa *Teapot*

Conclusão

Esta fase do projeto foi muito útil para consolidar os conhecimentos sobre *VBOs* que não tínhamos conseguido aplicar com sucesso nas aulas e agora, graças ao projeto já percebemos e conseguimos implementar.

Para além disso foi interessante ver agora o Sistema Solar já em movimento e com os planetas a realizarem os seus movimentos naturais de rotação e translação.

A parte de construir o *Teapot* através de curvas de *Bézier* foi onde tivemos mais dificuldades.