Computação Gráfica - TP3

Animações com Curvas, Superfícies Bézier e Otimização com VBOs

27.04.2025

André Carvalho a100818,

Flávio Sousa a100715,

Vicente de Carvalho Castro a91677,

Índice

1. Introdução

A Fase 3 do projeto tem como principal objetivo a introdução de conceitos avançados de modelação e animação no motor gráfico desenvolvido nas fases anteriores. Nesta etapa, foram implementadas superfícies paramétricas através de patches de Bézier, bem como transformações temporizadas utilizando curvas de Catmull-Rom. Para além destas funcionalidades, procedeu-se também à substituição do modo de renderização imediato (Immediate Mode) pela utilização de Vertex Buffer Objects (VBOs), com vista à optimização do desempenho gráfico.

Com estas adições, o motor gráfico passou a suportar animações complexas e uma representação mais realista e dinâmica das cenas, permitindo, por exemplo, a simulação do movimento orbital de um cometa ao longo de uma trajetória definida por uma curva cúbica, ou a criação de superfícies suaves e detalhadas a partir de um conjunto reduzido de pontos de controlo.

Este relatório descreve detalhadamente as abordagens seguidas para a implementação das novas funcionalidades, os algoritmos utilizados, bem como os desafios encontrados ao longo do desenvolvimento. Será ainda apresentada a cena de demonstração proposta – um sistema solar dinâmico com um cometa modelado com patches de Bézier – ilustrando o potencial das novas capacidades do motor.

2. Generator

2.1. Geração de Superfícies de Bézier e Curvas Catmull-Rom

Nesta fase do projeto, o `generator` foi ampliado para suportar a criação de superfícies paramétricas através de patches de Bézier e para interpretar curvas Catmull-Rom, utilizadas em transformações animadas. Estes mecanismos permitem uma representação mais fluida e detalhada da geometria e do movimento no motor gráfico.

2.2. Leitura do ficheiro `.patch`

Para gerar uma superfície de Bézier, o `generator` começa por ler um ficheiro com a extensão `.patch`. Este ficheiro contém, em primeiro lugar, o número de patches definidos. Seguem-se linhas com os índices dos pontos de controlo (16 por patch), que referenciam uma lista final com as coordenadas tridimensionais de todos os pontos de controlo utilizados.

A estrutura resultante associa a cada patch um conjunto de 16 pontos de controlo, necessários para calcular a respetiva superfície. Caso algum índice seja inválido ou as coordenadas estejam incompletas, o programa sinaliza o erro, garantindo a integridade dos dados lidos.

2.3. Avaliação da Superfície de Bézier

Uma superfície de Bézier é definida por uma grelha 4x4 de pontos de controlo. Para obter a posição de um ponto na superfície, utiliza-se uma fórmula baseada em multiplicações matriciais que combina os vetores paramétricos u e v com uma matriz de base de Bézier e os pontos de controlo.

Este método permite calcular qualquer ponto sobre a superfície com elevada precisão. A multiplicação é feita separadamente para as coordenadas x, y e z, o que permite reconstruir a posição completa do ponto no espaço tridimensional.

$$P(u,v) = U \cdot M \cdot P \cdot M^T \cdot V^T$$

com:

$$U = [u^3, u^2, u, 1]$$
$$V = [v^3, v^2, v, 1]$$
$$M = \text{Matriz de Bézier}$$

Matriz de Bézier:

$$M = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

2.4. Tesselação da Superfície

Com base num parâmetro de tesselação definido pelo utilizador, o espaço paramétrico da superfície é dividido uniformemente em pequenas células quadradas. Cada célula é convertida em dois triângulos que representam, de forma aproximada, a geometria local da superfície.

Este processo cria uma malha densa de triângulos, que pode ser renderizada com elevada eficiência utilizando VBOs.

2.5. Curvas Catmull-Rom

Para além das superfícies, foi também incluída a interpretação de curvas Catmull-Rom. Estas curvas são úteis para definir trajetórias suaves entre pontos de controlo, mantendo a continuidade e interpolando diretamente os pontos fornecidos.

A avaliação de uma curva Catmull-Rom baseia-se numa matriz própria, e permite determinar a posição e a derivada (ou direção) da curva num instante qualquer. Estas informações são essenciais, por exemplo, para animar objetos ao longo de um caminho e alinhá-los corretamente com a direção do movimento.

A implementação suporta o uso do tempo como parâmetro, possibilitando que o movimento ao longo da curva ocorra de forma fluida e natural ao longo da execução da aplicação.

$$P(t) = Tc \cdot M_{\{CR\}}c \cdot P_{\{seg\}}$$

$$M_{\text{CR}} = \begin{pmatrix} -0.5 & 1.5 & -1.5 & 0.5 \\ 1.0 & -2.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix}$$

3. Engine

Nesta secção descrevemos em detalhe a forma como a engine interpreta o XML de configuração, constrói o grafo de cena hierárquico, inicializa os buffers de vértices e procede à renderização, incluindo transformações estáticas e temporizadas.

3.1. Leitura do XML e construção do grafo da cena

A engine utiliza a biblioteca **tinyxml2** para percorrer o ficheiro XML definido pelo utilizador. A função central é a recursiva `processGroup`, que:

- Lê cada nó `<group>` do XML e cria uma instância de `GroupNode`.
- Extrai as transformações associadas (translate, rotate, scale), distinguindo entre modos estático e dependente de tempo/curva.
- Para `translate` com atributo `time`, converte os pontos `<point>` em trajectórias Catmull-Rom, armazenando tanto o tempo total como o flag `align`.
- Para `rotate` com atributo `time`, armazena a duração do ciclo completo de 360°.
- Carrega os modelos referenciados em `<model file="…"/>`, acumulando os vértices lidos num vetor.
- Processa recursivamente os subgrupos, criando uma árvore de `GroupNode` que reflecte a hierarquia definida no XML.

Esta abordagem garante que cada grupo herda apenas as transformações do nível acima, mantendo a separação lógica e simplificando animações hierárquicas.

3.2. Inicialização de VBOs

Para optimizar o envio de vértices à GPU, a engine substitui o modo imediato por **Vertex Buffer Objects (VBOs)**:

- A função `initVBOsRecursive` percorre toda a árvore de `GroupNode`.
- Para cada nó com vértices, gera um buffer OpenGL (`glGenBuffers`) e carrega os dados com `glBufferData(GL_ARRAY_BUFFER, …)`.
- Desta forma, cada grupo passa a ter um identificador `vboID`, permitindo chamadas rápidas a `glBindBuffer` e `glDrawArrays` durante o ciclo de renderização.

3.3. Renderização e aplicação de transformações

O algoritmo de renderização baseia-se em `renderGroup`, que para cada nó:

1. `glPushMatrix()` para isolar transformações.
2. Aplica, na ordem definida no XML, todas as transformações:
   - **Translate estático**: deslocamento simples com `glTranslatef`.
   - **Translate por curva**: calcula a posição e a derivada Catmull-Rom para o tempo corrente, desloca o objeto e, se `align=true`, alinha-o usando a normalizada derivada e um sistema de eixos ortonormais.
   - **Rotate estático**: rotação fixa com `glRotatef(ângulo, eixo)`.
   - **Rotate por tempo**: converte tempo em ângulo, permitindo rotações contínuas.
   - **Scale**: escalamento uniforme ou não uniforme.
3. Desenha a geometria do nó, se existir um VBO, configurando `glEnableClientState(GL_VERTEX_ARRAY)` e `glVertexPointer`, seguido de `glDrawArrays(GL_TRIANGLES, …)`.
4. Chama recursivamente `renderGroup` para cada filho, herdando a matriz de modelo actual.
5. `glPopMatrix()` para restaurar o estado de transformação anterior.

Adicionalmente, são desenhados, opcionalmente, eixos e órbitas (linhas de malha) para auxiliar o debugging e enfatizar o movimento.

3.4. Configuração da câmara e interacção do utilizador

A câmara é configurada a partir dos atributos em `<camera>`:

- `gluPerspective(fov, aspect, near, far)` ajusta o volume de visualização.
- `gluLookAt(camx, camy, camz, lookAtx, lookAty, lookAtz, upx, upy, upz)` posiciona e orienta a câmara.

O utilizador pode alterar a vista em tempo real através de callbacks de teclado:

- Teclas de cursor para rodar a cena ou aproximar/afastar.
- 'W', 'A', 'S', 'D', 'Q', 'E' para mover o ponto de observação.
- 'M' para alternar entre modo `wireframe` e preenchido.

3.5. Loop de renderização

O ciclo principal assenta em **GLUT**:

- `glutDisplayFunc(renderScene)` e `glutIdleFunc(renderScene)` garantem atualização contínua.
- `glutReshapeFunc(changeSize)` ajusta a janela e o frustum quando o utilizador redimensiona.
- `glutMainLoop()` entra no loop de eventos, processando input, actualizando transformações dependentes do tempo e redesenhando a cena.

3.6. Considerações de desempenho

- O uso de VBOs reduz a sobrecarga de CPU/GPU face ao modo imediato.
- A separação entre leitura de XML e inicialização de buffers permite pré-processar toda a geometria antes de iniciar o loop principal.
- A recursão em `renderGroup` facilita a aplicação de transformações hierárquicas sem necessidade de stacks manuais de matrizes.

Esta arquitectura modular torna a engine flexível e eficiente, suportando tanto cenas estáticas como animações baseadas em curvas e rotações temporizadas.

4. Cenas de Teste

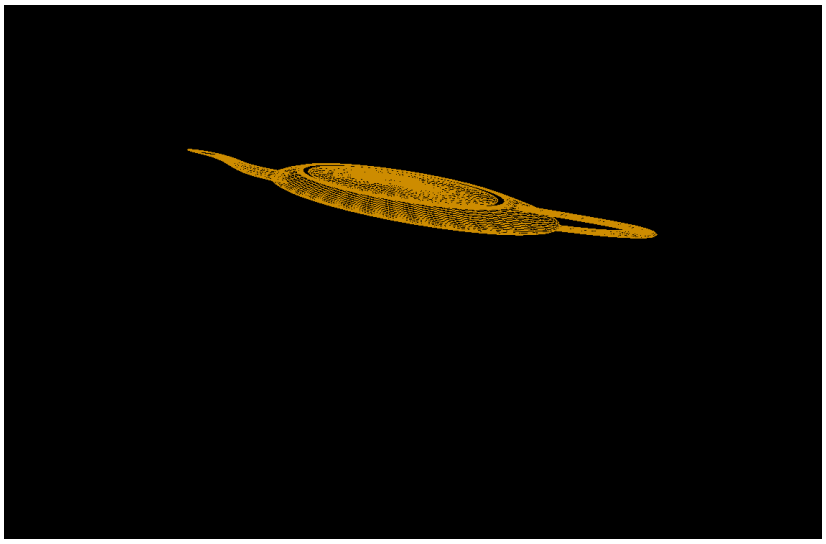

Figura 1: test_3_1.xml

Neste teste o objetivo não foi alcançado na sua totalidade uma vez que o bule segue a trajetória, contudo a trajetória não aparece materializada e a forma do bule aparece esticada ao longo do processo.

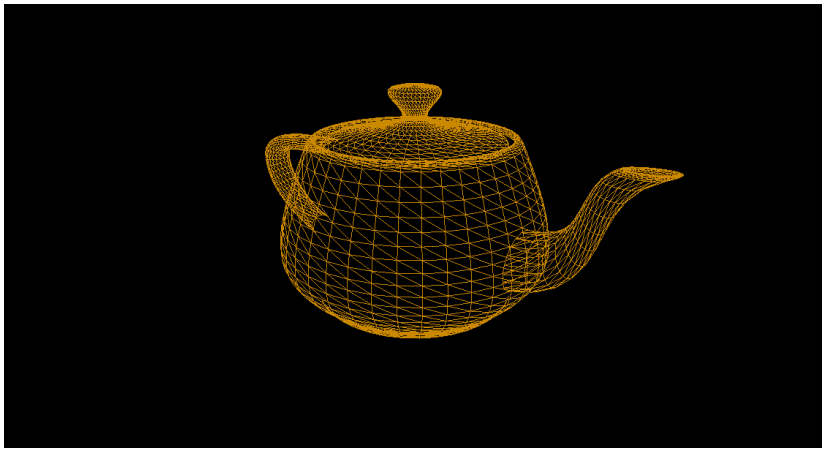

Figura 2: test_3_2.xml

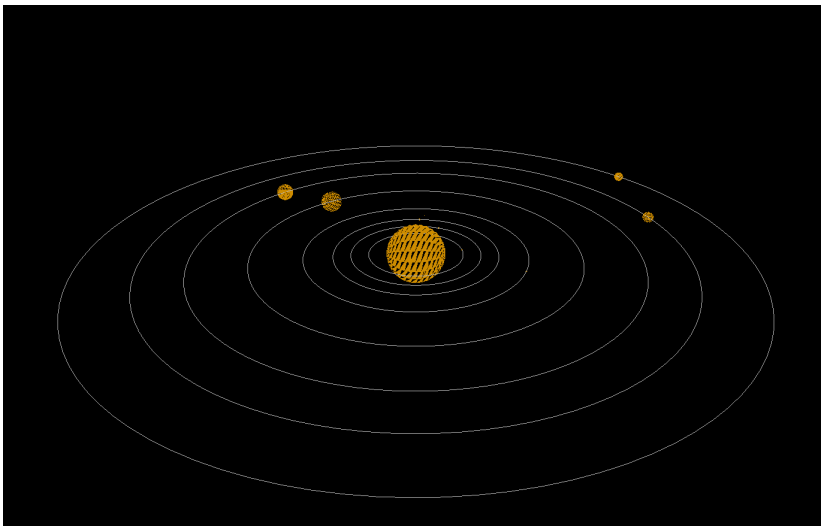

Figura 3: Sistema solar com curvas Catmull-Rom

5. Conclusão

A Fase 3 permitiu consolidar e expandir significativamente as capacidades do motor gráfico, ao integrar:

- **Superfícies de Bézier** geradas a partir de patches 4×4, através de leitura de ficheiros `.patch` e aplicação da fórmula matricial o que possibilitou criar geometria suave e detalhada com controlo mínimo de dados.
- **Curvas Catmull-Rom** para animações temporizadas, oferecendo trajectórias contínuas e a possibilidade de alinhar objetos ao longo da tangente, ideal para efeitos dinâmicos como o movimento de um cometa.
- **Vertex Buffer Objects (VBOs)**, garantindo desempenho superior face ao modo imediato, através da pré-alocação e envio eficiente de vértices para a GPU.

A combinação destas três componentes – geometria paramétrica, animação baseada em curvas e otimização de renderização – proporcionou uma engine flexível, capaz de suportar tanto cenas estáticas como dinâmicas, sem comprometer o desempenho. A implementação hierárquica de grupos e transformações recursivas simplificou a gestão de modelos e animações complexas.

Por fim, os resultados obtidos validam o design proposto: o motor produz superfícies suaves, animações fiáveis e rápida atualização em tempo real. Como próximos passos, a *Fase 4* focar-se-á na introdução de **normais** e **coordenadas de textura**, iluminação e texturização, elevando ainda mais o realismo visual das cenas.