3ª Ano

Computação Gráfica

Fase 3 – Curves, Cubic Surfaces and VBOS
LEI - 2023/2024

GRUPO 23

Ana Margarida Sousa Pimenta – A100830
Inês Gonzalez Perdigão Marques – A100606
Miguel Tomás Antunes Pinto – A100815
Pedro Miguel Costa Azevedo – A100557

Índice

1 Introdução

A terceira fase deste projeto marca um avanço significativo na aplicação geradora de modelos, introduzindo novos recursos e melhorias na renderização de cenas tridimensionais. Nesta etapa, o foco principal é a incorporação de modelos baseados em patches de Bezier, bem como a extensão das capacidades de transformação e rotação do motor gráfico.

Modelos Baseados em Patches de Bezier

O generator agora é capaz de criar modelos usando patches de Bezier. Estes modelos são definidos por pontos de controlo especificados num ficheiro e são renderizados com um nível de tesselação definido. O resultado final é um ficheiro .3d contendo uma lista de triângulos que compõem a superfície do modelo.

Transformações Avançadas

As capacidades de transformação foram expandidas para incluir curvas Catmull-Rom cúbicas para a tradução dos objetos. Os pontos que definem a curva, juntamente com a duração total da animação, são fornecidos como parâmetros. Além disso, foi introduzido um campo "align" para alinhar os objetos com a curva. Quanto à rotação, agora é possível especificar o tempo necessário para uma rotação completa de 360 graus em torno de um eixo específico, em vez do ângulo.

Renderização com VBOs

Uma mudança importante na renderização dos modelos é a transição do modo imediato para a utilização de Objectos de Buffer de Vértices (VBOs). Esta mudança melhora significativamente o desempenho e a eficiência da renderização, proporcionando uma experiência visual mais fluida.

Cena de Demonstração: Sistema Solar Dinâmico

Como parte da fase de demonstração, uma cena dinâmica de um sistema solar foi criada. Esta cena inclui um cometa com uma trajetória definida usando uma curva

Catmull-Rom. O cometa é construído utilizando patches de Bezier, por exemplo, com os pontos de controlo fornecidos para o bule de chá.

Em suma, esta fase representa um passo importante na evolução do projeto, introduzindo novas funcionalidades e aprimoramentos que enriquecem a experiência do usuário e expandem as possibilidades de criação de cenas tridimensionais realistas e dinâmicas.

2 Generator

Nesta 3ª fase do projeto tivemos de adicionar uma funcionalidade ao nosso generator responsável pela leitura de um ficheiro de patches (no caso o ficheiro teapot.patch) e passar o mesmo para a mesma estrutura das imagens das fase anteriores (esfera, cone, cubo e plano), no caso, teapot.3d. As funções implementadas para esta fase foram as seguintes:

readPatchesFile():

Esta função lê um ficheiro que contém informações sobre os patches de Bezier. Extrai o número de patches, os índices de controle de cada patch e os pontos de controlo. Cada patch é representado por uma lista de índices que referenciam os pontos de controlo. Os pontos de controlo são armazenados num vetor tridimensional.

multiplyMatrices():

Esta função realiza a multiplicação de duas matrizes. Ela é usada para multiplicar matrizes de transformação e pontos de controlo para calcular pontos na superfície do patch de Bezier.

surfacePoint(): Esta função calcula um ponto na superfície de um patch de Bezier dado os parâmetros 'u' e 'v', que variam de 0 a 1. Ela usa uma matriz de Bezier predefinida para calcular as coordenadas X, Y e Z do ponto.

buildPatches(): Esta função constrói os patches de Bezier. Ela lê os patches de um ficheiro usando a função readPatchesFile, e para cada patch, calcula os pontos na superfície usando a função surfacePoint. Depois, gera os triângulos necessários para

desenhar a superfície do patch com uma determinada tesselação e escreve-os num ficheiro de saída.

Em resumo, o código lê informações de patches de Bezier a partir de um ficheiro, calcula pontos na superfície desses patches e gera triângulos para desenhar a superfície.

3 Engine

Nesta fase, o engine, utilizando OpenGL e GLSL, implementa técnicas de animação baseadas em curvas de Catmull-Rom e rotações temporizadas, juntamente com uma gestão eficaz de modelos tridimensionais através do uso de VBOs (Vertex Buffer Objects) e VAOs (Vertex Array Objects).

A aplicação é estruturada de maneira a definir uma configuração de mundo, que inclui janelas, câmeras, e grupos de modelos, com cada grupo podendo conter transformações, tais como translação, rotação e escala. Utilizamos o formato XML para definir estas configurações, permitindo a flexibilidade e extensibilidade da aplicação.

Funções de Animação com Curvas de Catmull-Rom

interpolateCatmullRom(): Esta função é responsável por calcular a posição de um objeto num determinado instante t, baseando-se nos pontos de controlo da curva de Catmull-Rom. A função calcula primeiro em que segmento da curva o objeto deve estar, com base no valor de t e no número total de pontos. Depois, realiza a interpolação utilizando a fórmula matemática da curva de Catmull-Rom para determinar as coordenadas x, y e z precisas do objeto.

drawCatmullRomCurve(): Esta função desenha a curva de Catmull-Rom para visualização, chamando a interpolateCatmullRom para vários valores de t entre 0 e 1, e conectando os pontos resultantes com linhas. Isso ajuda a visualizar a trajetória que os objetos animados seguirão.

Funções de Rotação Baseada em Tempo

A rotação é implementada ajustando o ângulo de rotação com base no tempo decorrido desde o início da animação:

Dentro de **renderGroup**(): Esta função controla a transformação dos objetos, incluindo a sua rotação e translação. A rotação é ajustada com base no tempo, onde o ângulo é calculado como um múltiplo de 360 graus, proporcional ao tempo decorrido dividido pelo tempo total para uma rotação completa. Caso em vez do tempo seja fornecido o angle, acedemos a este a partir das estruturas definidas. Através desta função, é nos então possível gerar movimento nos objetos definidos, tendo em conta os seus parâmetros e definições.

Gestão de Modelos com VBOs e VAOs

initializeVBO(): Esta função cria um Vertex Buffer Object (VBO), que é uma memória na GPU onde armazenamos os vértices do modelo. Os vértices são carregados uma vez na GPU, aumentando a eficiência da renderização ao minimizar o tráfego de dados entre a CPU e a GPU.

readModel(): Esta função lê um ficheiro de modelo (contendo vértices), cria um novo modelo, e chama initializeVBO para transferir esses dados para a GPU. Esta função também inicializa um Vertex Array Object (VAO) que armazena a configuração de como os dados do VBO devem ser lidos durante a renderização.

Controle da Câmera

spherical2Cartesian(): Converte as coordenadas esféricas da câmera (raio, alfa, beta) para coordenadas cartesianas (x, y, z), permitindo que a posição da câmera seja atualizada dinamicamente com base na interação do usuário.

processKeys() e processSpecialKeys(): Estas funções manipulam a entrada do teclado para ajustar a posição e a orientação da câmera. Por exemplo, as teclas especiais permitem ao utilizador rotacionar a câmera ao redor do cenário ou aproximar-se e afastar-se, ajustando as variáveis esféricas alfa, beta e raio.

Reshape(): Esta função ajusta a projeção e a janela de visualização sempre que a janela da aplicação é redimensionada, garantindo que a perspectiva da câmera seja corretamente recalculada para manter a proporção e a escala corretas dos objetos na cena.

Cada uma destas funções desempenha um papel crucial na criação de uma experiência visual interativa e dinâmica, permitindo a animação suave dos objetos e o controlo intuitivo da câmera pelo utilizador.

4 Demonstração

4.1 Manual de utilização

De forma a tornarmos o nosso sistema mais iterativo adicionamos os seguintes comandos:

← rotação da câmara para a esquerda

→ rotação da câmara para a direita

↓ rotação da câmara para baixo

↑ rotação da câmara para cima

\+ zoom in

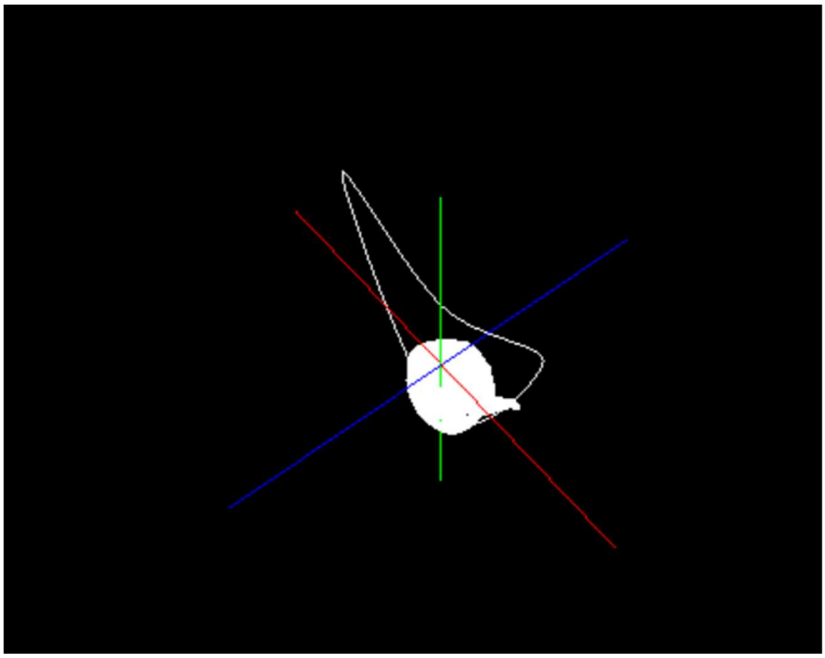
\- zoom out

q sair do programa

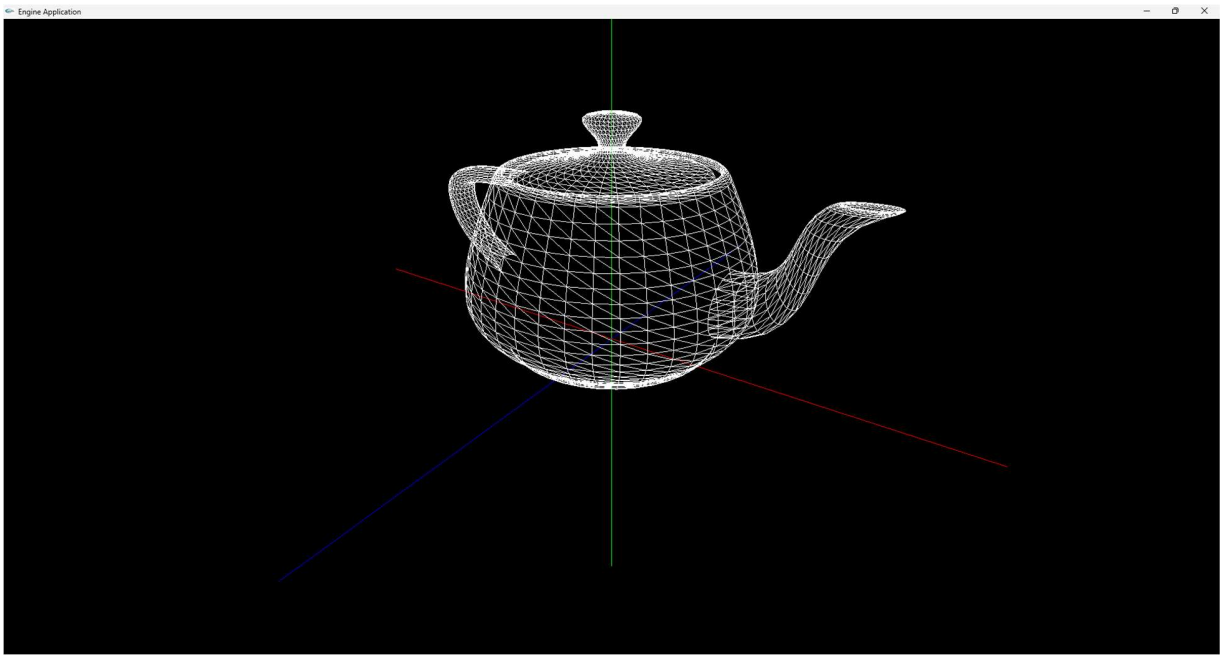
5 Resultado dos Testes

Neste ponto, demonstramos os resultados de todos os testes disponibilizados na BlackBoard.

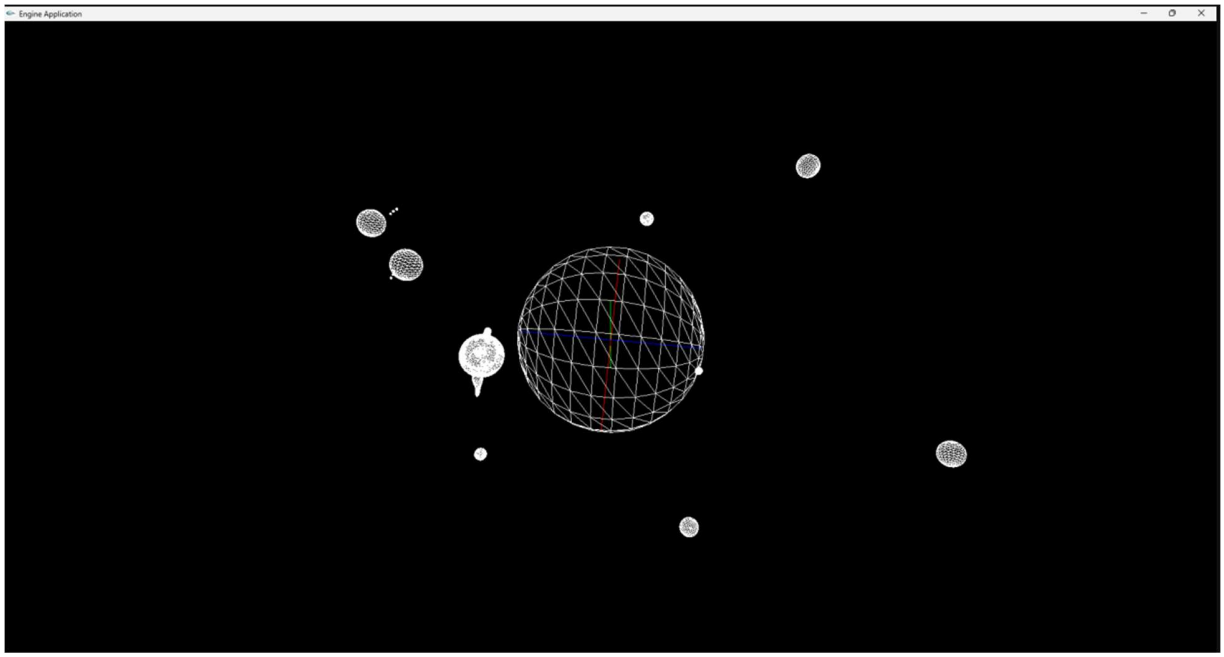
Teste 3.1



Teste 3.2



6 Sistema Solar



7 Conclusão

Nesta fase do projeto, alcançámos progressos significativos na capacidade de renderização e animação da nossa aplicação gráfica. A introdução de modelos baseados em patches de Bézier permitiu uma representação mais detalhada e suave de superfícies curvas, como evidenciado pelo modelo do cometa no sistema solar dinâmico. Esta funcionalidade foi implementada no *generator*, que agora aceita pontos de controlo de Bézier definidos num ficheiro externo e um nível de segmentação específico, resultando num ficheiro contendo uma lista de triângulos que descrevem a superfície.

Além disso, a extensão dos elementos de translação e rotação na *engine* permitiu uma maior flexibilidade e dinamismo nas animações. Com a implementação de curvas cúbicas de Catmull-Rom, foi possível definir trajetórias suaves e contínuas para os objetos, como demonstrado pela trajetória do cometa.

Importante destacar, a substituição do modo imediato pelo uso de VBOs (Vertex Buffer Objects) marcou uma evolução na otimização de desempenho. Esta abordagem moderna de gestão de dados de vértices não só melhorou a eficiência da renderização mas também alinhou a nossa aplicação com as práticas atuais de programação gráfica, resultando em animações mais fluidas e uma carga reduzida sobre o processamento da CPU.

Em conclusão, esta etapa do projeto cumpriu os objetivos propostos. Aplicamos conhecimentos padrão na área de computação gráfica. A execução desta fase prepara-nos adequadamente para as próximas etapas do trabalho.