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 3 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

Este código é parte de uma aplicação de engine gráfica que permite renderizar cenas tridimensionais definidas em ficheiros XML. A implementação atual visa criar uma hierarquia de cenas utilizando transformações geométricas, como translação, rotação e escala. Vamos analisar cada parte do código de acordo com o que foi solicitado no enunciado.

Estruturas de Dados:

São definidas várias estruturas de dados para representar elementos da cena, como janelas, câmeras, modelos, transformações e grupos.

Função spherical2Cartesian():

Esta função calcula as coordenadas cartesianas da câmera a partir das coordenadas esféricas (raio, alfa e beta).

Função renderModel():

Renderiza um modelo definido por uma lista de vértices. A renderização é feita duas vezes: uma vez em modo sólido e outra em modo wireframe.

Função renderGroup():

Renderiza um grupo, aplicando as transformações geométricas definidas para esse grupo e, em seguida, renderiza os modelos e subgrupos pertencentes a ele.

Função display():

Define a função de exibição da cena. Configura a projeção e a câmera, em seguida, renderiza todos os grupos da cena.

Função initialize():

Inicializa o ambiente gráfico OpenGL, configurando o teste de profundidade e a cor de fundo. Também calcula as coordenadas cartesianas iniciais da câmera.

Função readModel():

Lê um ficheiro de modelo (.3d) e retorna uma estrutura contendo os vértices desse modelo.

Função parseXML():

Analisa um ficheiro XML que descreve a cena, incluindo informações sobre a janela, câmera, grupos e modelos. Essas informações são armazenadas na estrutura de dados **World**.

Funções processKeys() e processSpecialKeys():

Lidam com a interação do usuário, como alterar a posição e a direção da câmera, bem como ajustar o zoom.

Função reshape():

Redefine a matriz de projeção quando a janela é redimensionada.

Função main():

Inicia a aplicação, inicializa o OpenGL, lê o ficheiro XML da cena, configura a janela e as funções de callback, e inicia o loop principal do OpenGL.

Conclusão:

Este código implementa a estrutura básica para renderização de uma cena 3D hierárquica, permitindo a aplicação de transformações geométricas em grupos de modelos. Essas transformações são especificadas num ficheiro XML, juntamente com outras informações da cena. A implementação atual é um passo importante para o desenvolvimento de uma engine gráfica mais completa e funcional.

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.

6 Sistema Solar

Para a representação do sistema solar neste trabalho optamos por fazer dois tipos de xml diferentes para 2 tipos de representação diferentes.

Primeiramente corremos um ficheiro XML, onde usamos a função translate em relação ao sol para todos os planetas. Usamos ainda a mesma função para as respetivas luas de cada planeta, porém não em relação ao sol, mas sim o seu planeta correspondente.

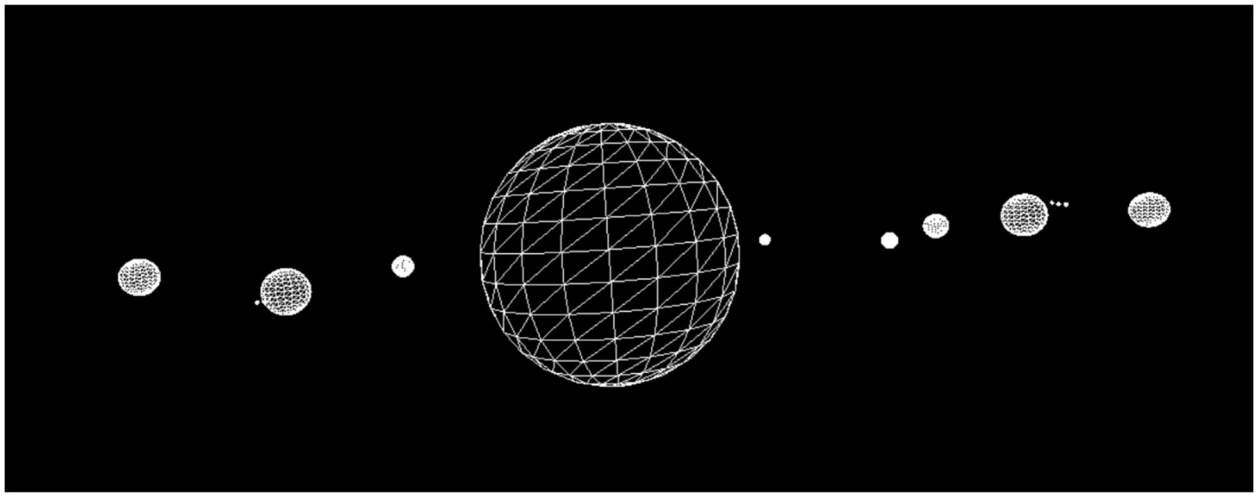

Imagem do sistema solar, através do SolarSystem.xml

Para o segundo XML, já usamos dimensões mais realistas para os planetas e para o sol. Mudamos também ainda a forma como os planetas são desenhados, sendo que, ao contrário do primeiro xml (SolarSystem.xml) a forma como estes são desenhados é aplicada a função translate relativamente ao planeta anterior. Neste xml metemos ainda os planetas alinhados no eixo dos X.

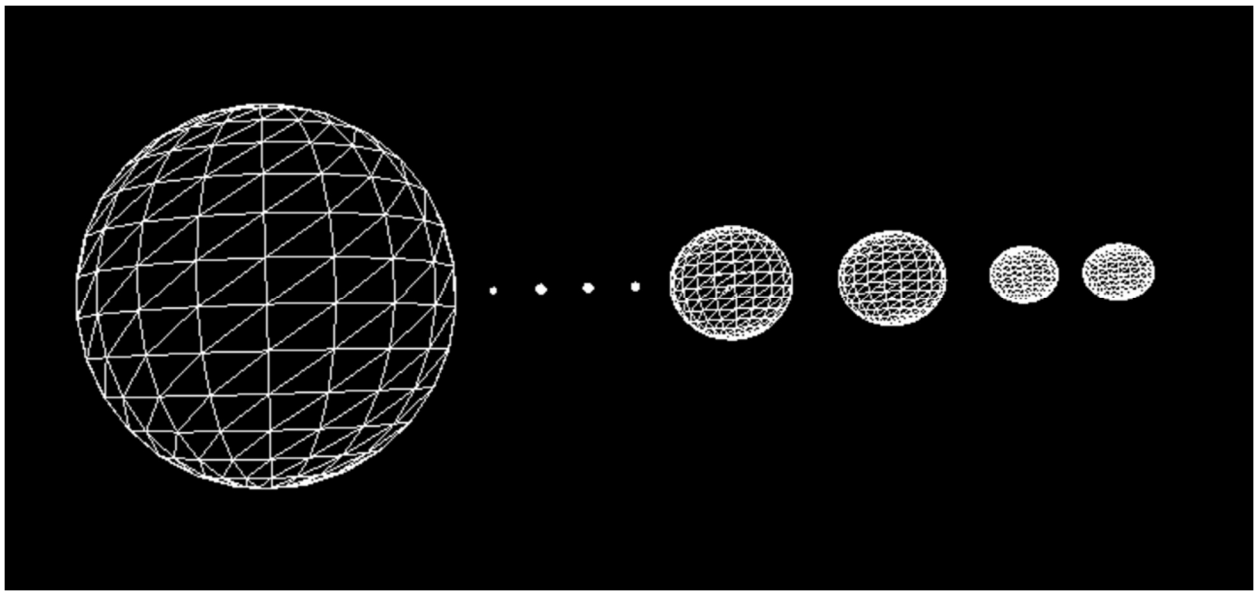

Imagem do sistema solar, através do SolarSystemAlligned.xml

7 Conclusão