Relatório de trabalho

Universidade da Beira Interior

“Loading, Moving and Rendering Star Wars’ Object Meshes in R^3”

Realizado por: Gonçalo Caixeiro, nº 49758

Orientador: Professor Doutor Abel Gomes

Engenharia Informática

Covilhã

23 de novembro de 2024

Introdução

Objetivos

Este trabalho tem como objetivo explorar o movimento de objetos em um ambiente iluminado pela técnica de iluminação *Phong*. Este exercício nos propõe compreender de que maneira a iluminação impacta a atmosfera do ambiente.

Solução

Ficheiros .OBJ

O que é um OBJ?

Este formato é utilizado para a representação de modelos tridimensionais, contendo informações sobre vértices, normais e coordenadas UV para texturas. Esses componentes são essenciais, pois permitem que, por meio do código, possamos efetivamente representar esses modelos 3D.

Etapas do Carregamento de Arquivos OBJ

Para iniciar, é necessário ler os dados do nosso modelo a partir do arquivo OBJ. Para isso, utilizaremos bibliotecas adicionais. Além disso, será fundamental triangular nosso modelo, de modo a que ele seja composto exclusivamente por triângulos, facilitando assim a sua leitura.

Representação do Modelo

Após a leitura e armazenamento em vetores, os dados serão transmitidos para os *buffers*, a fim de serem enviados à *GPU*. Em seguida, procederemos à renderização dos modelos. Para isso, calcularemos a matriz MVP, na qual já realizamos algumas transformações nos objetos, como a inversão de um deles ao longo do eixo das abcissas e a redução das suas dimensões.

Por fim, enviaremos as matrizes e as variáveis de iluminação para os *shaders*, ativaremos os *buffers* e realizaremos o desenho dos modelos.

Movimentação dos Modelos

É estabelecida uma posição inicial para os dois hangares e para a nave, além de definirmos o *deltatime* para a transição de *frames*. Como nosso objetivo é mover a nave de um hangar para o outro, será designada uma posição inicial que corresponda à localização de um dos hangares. Para

realizar a movimentação da nave, utilizaremos as teclas W, A, S e D, afetando apenas os eixos X e Z da posição da nave, uma vez que não consideraremos a opção de subida ou descida da nave.

Iluminação

A iluminação *Phong*, iluminação utilizada, é dividida em 3 componentes:

- Luz Ambiente: A luz está presente em todo o lado e não depende dá localização da luz ou do observador.
- Luz Difusa: Representa a interação da luz com a superfície e irá depender do vetor da luz e da normal da superfície.
- Luz Especular: Representa os reflexos da superfície do objeto.

No código necessitamos de fornecer algumas informações aos *shaders* para que estes possam realizar a iluminação na cena.

Parâmetros para o *Shader*

Estes parâmetros serão:

lightPos: Posiciona a fonte de luz.

viewPos: Representa a posição da câmera, necessária para os cálculos de reflexão especular.

lightColor: Define a intensidade e cor da luz.

objectColor: Cor do material do objeto.

shininess: Controla o grau de reflexividade do material.

A iluminação *Phong* irá ser calculada no nosso *fragment shader.*

Aplicação na cena

A nave e os hangares recebem a iluminação *Phong*, que é afetada pela posição da luz definida em lightPos.

A posição e orientação dos objetos na cena são manipuladas pela matriz do modelo (*ModelMatrix*), enquanto a iluminação calcula os reflexos com base na normal da superfície.

Funções do código

loadOBJ()

Cabeçalho: bool loadOBJ(const char * path, std::vector<glm::vec3> & out_vertices, std::vector<glm::vec2> & out_uvs,std::vector<glm::vec3> & out_normals)

Input: Path do ficheiro, vértices, normais e UVs do modelo.

Output: Um booleano que diz respeito ao sucesso da operação.

Tarefa: A função loadOBJ desempenha um papel crucial na importação de modelos 3D no formato OBJ, preparando os dados para serem renderizados em tempo real. Sua implementação é direta, mas poderosa o suficiente para manipular modelos básicos em ambientes gráficos modernos.

glGetUniformLocation()

Cabeçalho: GLint glGetUniformLocation(GLuint program, const char *name);

Input: O identificador da variável uniforme que será atualizada, indica quantos vec3 serão enviados e o ponteiro para os dados a serem enviados.

Output: Não possui.

Tarefa: glUniform3fv envia valores para variáveis vec3 uniformes nos shaders GLSL.

É útil para trabalhar com luz, câmera, cores ou vetores no espaço 3D.

computeMatricesFromInputs()

Cabeçalho: void computeMatricesFromInputs();

Input: Informação sobre a cena, mais especificamente o movimento realizado pelo utilizador que irá influenciar a cena.

Output: Atualização da matriz de projeção.

Tarefa: Calcula a posição e orientação da câmera no espaço 3D com base no *input* do rato e teclado.

Atualiza as matrizes *ViewMatrix* e *ProjectionMatrix*: Estas matrizes são usadas no *pipeline* gráfico para transformar objetos 3D em coordenadas de tela.

Permite movimento (WASD ou teclas direcionais) e rotação da visão, simulando uma câmera controlada pelo jogador.

Conclusões

Com a realização deste trabalho, pude concluir que a utilização de matrizes não é a única abordagem para criar objetos em uma cena. Além disso, compreendi como a iluminação influencia toda a composição visual e os cálculos necessários para que esse fenômeno ocorra.

Webgrafia

https://chatgpt.com

https://www.di.ubi.pt/~agomes/cg/

https://learnopengl.com