Trabalho de Computação Gráfica - Aplicação - Posto de Gasolina 3D.

Universidade Federal do Ceará -
Campus Russas.

Alunos: Douglas Lima - 552590
Thiago Henrique - 580851

Introdução

- Criação de um cenário 3D interativo de um posto de gasolina.
- O ambiente é composto por múltiplos elementos para criar uma cena:
  - Estrutura Principal: O posto de gasolina.
  - Entorno: Ruas de asfalto que cercam a área.
  - Objetos de Cena: Carros estacionados, uma garagem e lixeiras para detalhar o ambiente.
- O objetivo foi construir um pequeno mundo virtual, em primeira pessoa, aplicando os conceitos de computação gráfica.

Roteiro e Demonstração

- Demonstração ao vivo da aplicação.
- Análise técnica dos critérios implementados:
  - Modelagem e Texturização.
  - Câmera, Interação e Colisão.
  - Iluminação Dinâmica.
  - Música.

Ferramentas e Bibliotecas

- Tecnologias Utilizadas:
  - OpenGL (PyOpenGL): API gráfica principal para renderização 2D e 3D.
  - GLFW: Criação e gerenciamento da janela, contexto OpenGL e captura de eventos de entrada (teclado/mouse).
  - GLM (PyGLM): Biblioteca matemática para operações com vetores e matrizes, essencial para os cálculos gráficos.
  - Numpy: Estruturação e manipulação eficiente dos arrays de vértices enviados para a GPU.
  - PyWavefront: Carregamento e processamento de modelos 3D no formato .obj.
  - Pillow (PIL): Carregamento e manipulação dos arquivos de imagem para as texturas.
  - Pygame: Utilizada especificamente para a funcionalidade de áudio (música de fundo).

Arquitetura do Projeto

- Código modularizado com Programação Orientada a Objetos.
- Divisão de responsabilidades em classes principais:
  - Model: Carrega, renderiza e aplica texturas a modelos .obj.
  - Camera: Gerencia a visão em 1ª pessoa e a interação (mouse/teclado).
  - CollisionSystem: Implementa a lógica de detecção de colisão (AABB).
  - Skybox: Responsável pela renderização do mapa de ambiente (cubemap).

Modelagem e Cenário

- Cena composta por múltiplos modelos .obj independentes.
- Utilização da biblioteca pywavefront para carregar a geometria.
- Posicionamento dos objetos com matrizes de transformação da glm.

```
# Instância de um modelo
model_garagem = Model(obj_file_garagem, collision_system, texture_path=texture_file_garagem)

# Criação da matriz de transformação para posicionar, rotacionar e escalar
garagem_transform = glm.translate(glm.mat4(1.0), glm.vec3(50.5, -0.25, 2.5)) * \
                    glm.rotate(glm.mat4(1.0), glm.radians(180), glm.vec3(0, 1, 0)) * \
                    glm.scale(glm.mat4(1.0), glm.vec3(1.5, 1.5, 1.5))

# Renderização com a matriz aplicada
model_garagem.render(shader_program, model_matrix=garagem_transform)
```

Texturização e Skybox

- Carregamento de texturas (.jpg, .png) com a biblioteca Pillow.
- Mapeamento de texturas distintas para diferentes objetos.
- Implementação de Skybox para o céu com samplerCube.
- Manipulação de coordenadas de textura (TexCoords) no shader.

```
// ...
in vec2 TexCoords;
out vec4 FragColor;

uniform sampler2D texture_diffuse1; // Textura 2D normal
uniform samplerCube skybox;         // Textura do Skybox
uniform bool useTexture;

void main()
{
    vec3 baseColor = objectColor;
    if (useTexture) {
        // Busca a cor na textura 2D
        baseColor = texture(texture_diffuse1, TexCoords).rgb;
    }
    // ...
}
```

Câmera e Interação

- Câmera em primeira pessoa com visão controlada pelo mouse.
- Cálculo dos vetores de direção (yaw, pitch) para orientação.
- Movimentação no cenário com as teclas WASD.

```python
class Camera:
    def process_mouse(self, window, xpos, ypos):
        if self.first_mouse:
            self.last_x = xpos
            self.last_y = ypos
            self.first_mouse = False

        xoffset = xpos - self.last_x
        yoffset = self.last_y - ypos
        self.last_x = xpos
        self.last_y = ypos

        sensitivity = 0.1
        xoffset *= sensitivity
        yoffset *= sensitivity

        self.yaw += xoffset
        self.pitch += yoffset

        self.pitch = max(-89.0, min(89.0, self.pitch))

        direction = glm.vec3()
        direction.x = glm.cos(glm.radians(self.yaw)) * glm.cos(glm.radians(self.pitch))
        direction.y = glm.sin(glm.radians(self.pitch))
        direction.z = glm.sin(glm.radians(self.yaw)) * glm.cos(glm.radians(self.pitch))
        self.front = glm.normalize(direction)
```

Sistema de Colisão

- Sistema baseado em Axis-Aligned Bounding Boxes (AABB).
- Cada modelo possui uma ”caixa de colisão” invisível.
- A movimentação da câmera é impedida ao tentar entrar nessas caixas.
- Permite que objetos pareçam sólidos.

```python
class CollisionSystem:
    def _point_in_aabb_with_radius(self, point, aabb_min, aabb_max, radius):
        # Expande a caixa para considerar o "raio" do jogador.
        expanded_min = aabb_min - glm.vec3(radius, 0, radius)
        expanded_max = aabb_max + glm.vec3(radius, 0, radius)
        # Verifica se o ponto está dentro da caixa expandida.
        return (point.x >= expanded_min.x and point.x <= expanded_max.x and
                point.y >= aabb_min.y and point.y <= aabb_max.y and
                point.z >= expanded_min.z and point.z <= expanded_max.z)
```

Iluminação Dinâmica

- Modelo de iluminação de Phong implementado no Fragment Shader.
- Cálculo em tempo real de três componentes:
  - Ambiente: Luz geral na cena.
  - Difuso: Influência da luz na superfície do objeto.
  - Especular: Pontos de brilho e reflexos.
- Utilização de duas fontes de luz dinâmicas na cena.

```
// --- LUZ 1 ---
vec3 norm = normalize(Normal);
vec3 lightDir1 = normalize(lightPos - FragPos);
float diff1 = max(dot(norm, lightDir1), 0.0);
vec3 diffuse1 = diff1 * lightColor;

float specularStrength1 = 0.5;
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir1 = reflect(-lightDir1, norm);
float spec1 = pow(max(dot(viewDir, reflectDir1), 0.0), 32);
vec3 specular1 = specularStrength1 * spec1 * lightColor;

// --- LUZ 2 ---
vec3 lightDir2 = normalize(lightPos2 - FragPos);
float diff2 = max(dot(norm, lightDir2), 0.0);
vec3 diffuse2 = diff2 * lightColor2;

float specularStrength2 = 0.2;
vec3 reflectDir2 = reflect(-lightDir2, norm);
float spec2 = pow(max(dot(viewDir, reflectDir2), 0.0), 16);
vec3 specular2 = specularStrength2 * spec2 * lightColor2;

// Componentes de iluminação combinados.
vec3 lighting = ambient + diffuse1 + specular1 + diffuse2 + specular2;
vec3 result = lighting * baseColor;
FragColor = vec4(result, 1.0);
```

Elementos Secundários

- Ambiência Sonora.
  - Para aumentar a imersão do usuário no cenário, foi adicionada uma música de fundo.
  - A funcionalidade foi implementada utilizando a biblioteca *pygame.mixer*.
  - A música inicia automaticamente e toca em loop durante a execução.
- Trecho de Código (Implementação da Música):

```python
# Importação da biblioteca
import pygame

# ... na função main() ...

# Inicialização do Pygame e do mixer
pygame.init()
pygame.mixer.init()

# Carregamento e configuração do arquivo de música
pygame.mixer.music.load('músicas/sound1.mp3')
pygame.mixer.music.set_volume(0.2)

# ... no loop principal ...
if not music_started:
    pygame.mixer.music.play(-1) # O argumento -1 faz a música tocar em loop
    music_started = True
```

Conclusão

- Funcionalidades Implementadas:
  - Cena complexa com múltiplos modelos .obj.
  - Sistema de colisão AABB funcional.
  - Iluminação de Phong dinâmica com múltiplas fontes de luz.
  - Texturização avançada com Skybox.
  - Interação fluida via teclado e mouse.
  - Ambiência sonora.

OBRIGADO.