Atividade 7: Implementação de Labirinto 3D e Cubo Animado com Pygame

Discente: Nicolas Expedito Lana Mendes
Matrícula: 22.1.4028
Docente: Rafael Alves Bonfim de Queiroz
Universidade Federal de Ouro Preto
Departamento de Computação
BCC327 - Computação Gráfica

Março de 2025

1 Introdução

Nesta atividade, foram desenvolvidas duas aplicações gráficas interativas utilizando exclusivamente a biblioteca **Pygame**. O objetivo principal é demonstrar a implementação de conceitos fundamentais de computação gráfica, sem o uso de OpenGL, através de:

1. **Exercício 7.1: Labirinto 3D com Raycasting**
   Simulação de um ambiente 3D utilizando a técnica de **raycasting**. O usuário navega em primeira pessoa pelo labirinto, controlando a posição e o ângulo da câmera com as teclas de movimentação, enquanto a lógica de colisão impede a travessia de paredes.

2. **Exercício 7.2: Cubo Animado com Projeção Alternada**
   Renderização de um cubo animado, que rotaciona e varia seu tamanho (efeito pulsante), com a possibilidade de alternar entre projeção perspectiva e ortogonal. A renderização é realizada em wireframe e as transformações são calculadas manualmente.

Este relatório detalha a abordagem adotada, as adaptações realizadas para substituir funções OpenGL por métodos do Pygame e os resultados alcançados.

2 Descrição da Implementação

2.1 Exercício 7.1: Labirinto 3D com Raycasting

Representação do ambiente:
O labirinto é modelado como uma matriz bidimensional, onde:

- `1` representa uma parede.
- `0` representa um espaço livre para a movimentação.

Técnica de Raycasting:
A renderização 3D é simulada utilizando a técnica de **raycasting**, que consiste em lançar raios a partir da posição do jogador dentro de um intervalo definido pelo *campo de visão* (FOV). Para cada raio, é calculada a distância até encontrar uma parede. Com essa distância, calcula-se a altura da parede a ser desenhada, criando a sensação de profundidade. A correção do efeito *fish-eye* é aplicada multiplicando a distância pelo cosseno da diferença entre o ângulo do jogador e o ângulo do raio.

Movimentação e Colisão:
O jogador (câmera) é controlado por:

- `W/S` para avançar e recuar;
- `A/D` para movimento lateral (strafing);
- `Setas Esquerda/Direita` para girar a câmera.

Antes de atualizar a posição, o novo ponto é verificado na matriz para assegurar que não corresponde a uma parede, evitando a travessia.

2.2 Exercício 7.2: Cubo Animado com Projeção Alternada

Modelagem do Cubo:
O cubo é definido por 8 vértices e 12 arestas que conectam esses pontos. As transformações são aplicadas manualmente:

- **Rotação:** A rotação em torno do eixo Y é realizada pela função `rotate_y()`, que utiliza funções trigonométricas para transformar cada vértice.
- **Escala:** Um fator de escala (com efeito pulsante) é aplicado multiplicando as coordenadas dos vértices.

Projeção:
Duas projeções foram implementadas:

- **Perspectiva:** Simulada dividindo as coordenadas x e y por um fator que depende da distância z do ponto, criando a sensação de profundidade.
- **Ortogonal:** Despreza o componente z, mantendo as proporções dos objetos sem perspectiva.

O usuário alterna entre as projeções pressionando `p` (perspectiva) e `o` (ortogonal).

3 Adaptações Realizadas

Como o código original utilizava funções OpenGL, foram necessárias as seguintes adaptações para uma implementação apenas com Pygame:

- **Substituição de Transformações:** Em OpenGL, funções como `glTranslatef()`, `glRotatef()` e `glScalef()` facilitam as transformações 3D. Neste trabalho, essas transformações são calculadas manualmente usando funções matemáticas. Por exemplo, a rotação em torno do eixo Y é implementada na função `rotate_y()`.

- **Implementação de Projeções:** As funções `gluPerspective()` e `glOrtho()` foram substituídas por funções customizadas (`project_point()` e `project()`) que realizam os cálculos necessários para simular as projeções perspectiva e ortogonal.
- **Renderização via Raycasting:** Para o labirinto 3D, em vez de usar `glutSolidCube()` para desenhar paredes, foi utilizada a técnica de raycasting para desenhar retângulos verticais que variam em altura conforme a distância, simulando a visão 3D.
- **Controle de Movimentação:** Em vez de utilizar uma câmera virtual com `gluLookAt()`, a posição do jogador é representada por coordenadas na matriz, e a movimentação é controlada diretamente por atualizações dessas coordenadas com verificação de colisão.

4 Instruções de Execução

1. Instale o Python e a biblioteca `pygame` (ex.: `pip install pygame`).
2. Salve o código da implementação como `atividade7.py`.
3. Execute o script com `python atividade7.py`.
4. Para alternar entre os modos:
   - Pressione a tecla `1` para entrar no modo Labirinto 3D.
   - Pressione a tecla `2` para o modo Cubo Animado.
5. No modo Labirinto, utilize:
   - `W/S`: avançar/recuar;
   - `A/D`: movimento lateral;
   - `Setas Esquerda/Direita`: girar a câmera.
6. No modo Cubo, pressione:
   - `p` para ativar a projeção perspectiva;
   - `o` para a projeção ortogonal.

5 Resultados e Discussão

Labirinto 3D:
A aplicação permite a navegação por um labirinto renderizado com raycasting. As paredes são desenhadas como barras verticais cujo tamanho varia com a distância, proporcionando uma sensação de profundidade. A colisão com as paredes é efetivamente verificada, impedindo que o jogador atravesse os limites definidos.

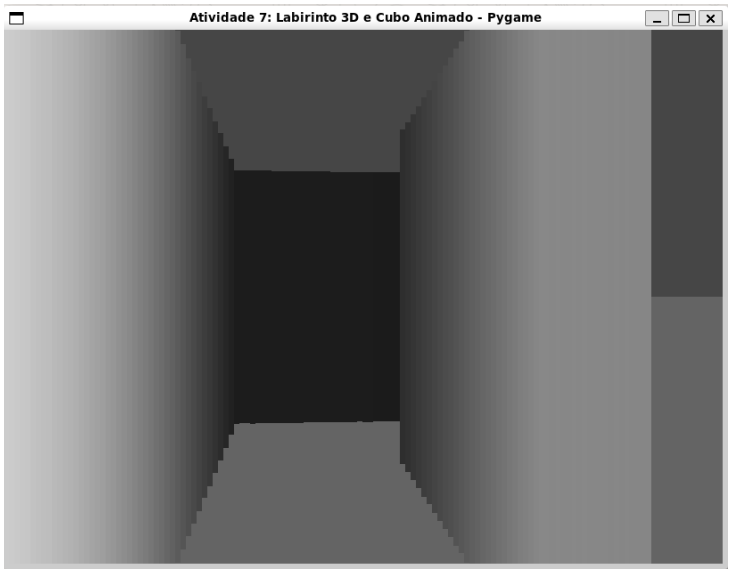

Figure 1: Exercício 7.1: Visualização do labirinto 3D com raycasting. Parte 1

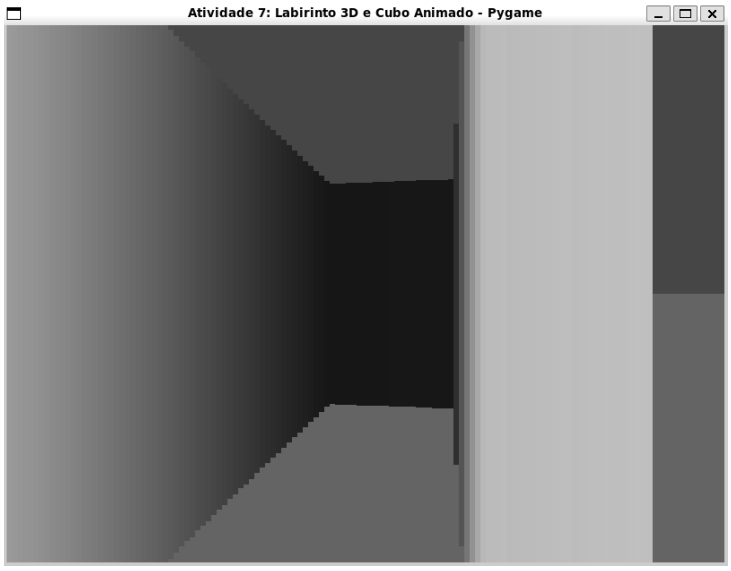

Figure 2: Exercício 7.1: Visualização do labirinto 3D com raycasting. Parte 2

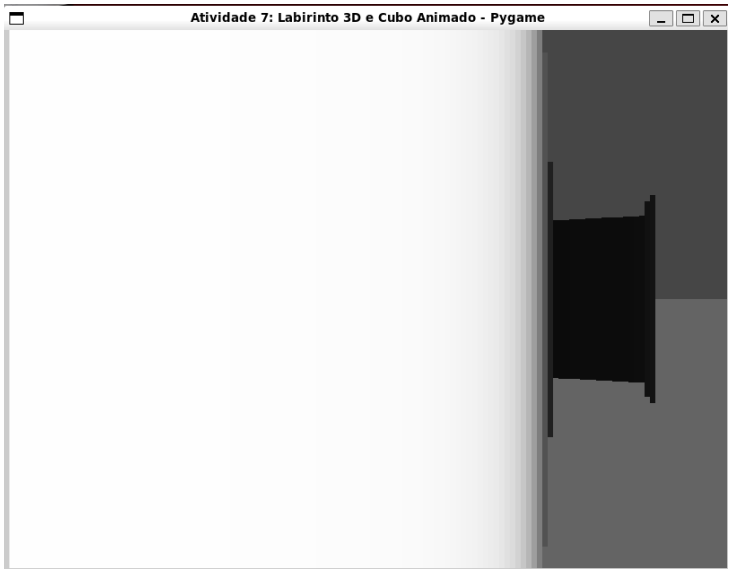

Figure 3: Exercício 7.1: Visualização do labirinto 3D com raycasting. Colisão

Cubo Animado:
O cubo, renderizado em wireframe, apresenta uma rotação contínua e varia seu tamanho de forma pulsante. A alternância entre projeção perspectiva e ortogonal é perceptível, evidenciando a diferença na forma como os vértices são projetados na tela. Esta implementação demonstra de maneira clara como os cálculos manuais podem substituir as funções nativas do OpenGL.

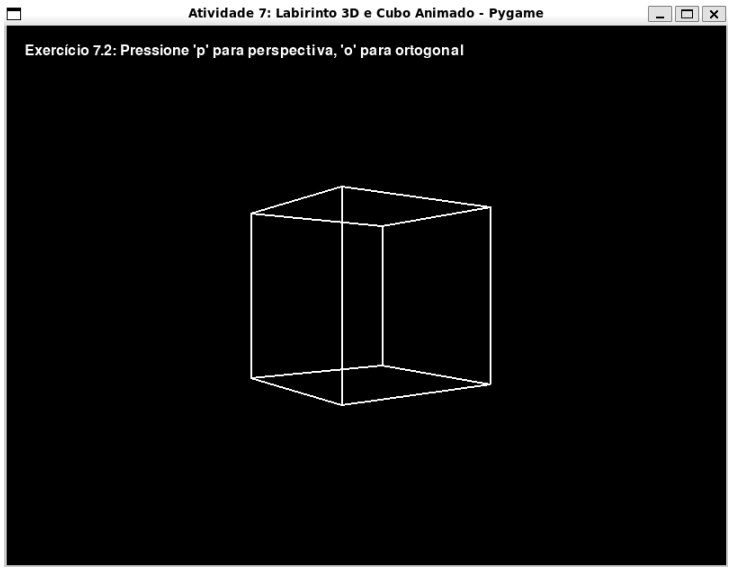

Figure 4: Exercício 7.2: Cubo animado com projeção alternada. Perspectiva

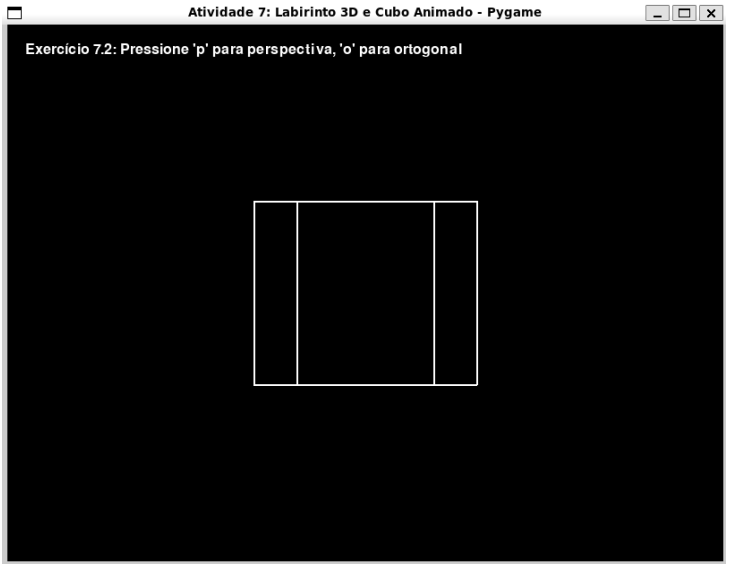

Figure 5: Exercício 7.2: Cubo animado com projeção alternada. Projeção Ortogonal

6 Conclusão

A implementação dos exercícios 7.1 e 7.2 utilizando apenas Pygame comprovou ser viável simular conceitos de renderização 3D, movimentação de câmera, projeção e transformações. As adaptações realizadas para substituir funções OpenGL por cálculos matemáticos demonstraram uma compreensão aprofundada dos fundamentos de computação gráfica. Em resumo:

- O labirinto 3D com raycasting permite a exploração interativa com verificação de colisão, simulando um ambiente 3D.
- O cubo animado ilustra a aplicação de rotação, escala e projeção, com alternância dinâmica entre os modos perspectiva e ortogonal.

Estas implementações fornecem uma base robusta para projetos mais avançados em computação gráfica utilizando apenas bibliotecas 2D, como o Pygame.