FACULDADE ENGENHARIA
Departamento de Informática

Jogo de Labirinto 3D com OpenGL

Bruno Correia (51741), Henrique Laia (51667)

Relatório de Projeto de Computação Gráfica
Engenharia Informática

Professores: Abel Gomes, João Dias

Dezembro de 2025

Resumo

Este projeto consiste no desenvolvimento de um jogo de labirinto 3D com recurso a OpenGL como API gráfica principal. O jogo apresenta geração procedural de labirintos através do algoritmo de Kruskal, um sistema de iluminação dinâmica com efeito de lanterna, comunicação em rede entre dois jogadores (host e cliente) via sockets TCP, e renderização de texto para interface com o utilizador. O projeto demonstra a aplicação prática de conceitos fundamentais de computação gráfica, que incluem transformações 3D, iluminação, texturas, shaders GLSL, e gestão de cena.

Palavras-chave

OpenGL, Computação Gráfica, Labirinto, Geração Procedural, Kruskal, GLFW, Shaders, Networking

Índice

Resumo **iii**

Índice **v**

Lista de Figuras **ix**

Lista de Tabelas **xi**

1 Introdução **1**

1.1 História e Enquadramento 1

1.2 Motivação 1

1.3 Objetivos 1

1.4 Estrutura do Relatório 2

2 Tecnologias Utilizadas **3**

2.1 OpenGL 3

2.2 GLFW 3

2.3 GLM 4

2.4 GLAD 4

2.5 FreeType 4

2.6 STB Image 5

2.7 CMake 5

2.8 Sockets TCP 5

2.9 Resumo de Bibliotecas 5

3 Desenvolvimento e Implementação **7**

3.1 Gestão do Projeto 7

3.1.1 Estrutura de Diretorias 7

3.1.2 Sistema de Compilação 7

3.1.3 Carregamento de Assets Portável 8

3.2 Descrição do Código 8
3.2.1 Classe Game 8
3.2.2 Classe Maze 9
3.2.3 Classe Camera 9
3.2.4 Classe Shader 10
3.2.5 Geração de Labirinto: Algoritmo de Kruskal 10
3.2.6 Classe TextRenderer 10
3.2.7 Classe Mesh 11
3.2.8 Módulo Network 11
3.3 Funcionalidades Implementadas 11
3.3.1 Geração Procedural de Labirintos 11
3.3.2 Sistema de Iluminação Dinâmica 12
3.3.3 Networking Host/Cliente 12
3.3.4 Sistema de Diálogo Introdutório 13
3.3.5 Ambiente Exterior 13
3.3.6 Deteção de Colisões 14
3.3.7 Sistema de Pausa 14
3.3.8 Transição de Cor Progressiva 15
3.3.9 Shader de Metal Líquido (Portal) 16
3.3.10 Texturas e Materiais 16
3.3.11 Minimapa (HUD) 17
3.4 Documentação com Doxygen 18

4 Conclusão e Trabalho Futuro **19**

4.1 Conclusão 19
4.1.1 Trabalho Realizado 19
4.2 Limitações e Trabalho Não Realizado 20
4.3 Trabalho Futuro 20
4.4 Considerações Finais 21

Bibliografia **23**

Lista de Figuras

3.1 Efeito de lanterna (Spotlight) e atenuação da luz nas paredes do labirinto. . 12

3.2 Menu introdutório renderizado com FreeType sobre a cena 3D. 14

3.3 Ambiente exterior mostrando as árvores procedurais e o portal ao fundo. . . 15

3.4 Overlay de Pausa renderizado quando o jogador pressiona ESC. 16

3.5 Renderização do Portal com efeito "Mercúrio Líquido" e iluminação Fresnel. 17

3.6 Interface de jogo com o Minimapa no canto superior direito, mostrando a posição do jogador (vermelho) e as paredes (preto). 18

Lista de Tabelas

2.1 Bibliotecas externas utilizadas no projeto. 5

Capítulo 1

Introdução

[1]

1.1 História e Enquadramento

O jogador encontra-se num labirinto, com uma lanterna que ilumina o caminho e uma citação pouco esclarecedora. Percebe que a mesma pessoa que vai sair dali não é a mesma que entrou.

1.2 Motivação

A escolha de desenvolver um jogo de labirinto 3D foi motivada por diversos fatores:

- **Geração Procedural**: Implementar algoritmos de geração procedural, nomeadamente o algoritmo de Kruskal, permite criar labirintos únicos e imprevisíveis em cada execução.
- **Aplicação de Computação Gráfica**: O projeto requer a aplicação prática de transformações 3D, sistemas de iluminação, mapeamento de texturas, e programação de shaders GLSL.
- **Interatividade**: A implementação de controlos de câmara em primeira pessoa e deteção de colisões proporciona uma experiência imersiva.
- **Networking**: A inclusão de funcionalidades de rede (modo host/cliente) adiciona complexidade técnica e demonstra conhecimentos de programação de sistemas.
- **Desafios Técnicos**: O projeto apresenta desafios interessantes em áreas como otimização de renderização, gestão de recursos, e arquitetura de software.

1.3 Objetivos

Os principais objetivos deste projeto são:

[1] Este relatório foi produzido com recurso ao template LaTeX para teses e dissertações da Universidade da Beira Interior [1], uma versão não oficial mantida pela comunidade.

- Desenvolver um motor de renderização 3D funcional utilizando OpenGL.
- Implementar geração procedural de labirintos através do algoritmo de Kruskal.
- Criar um sistema de iluminação dinâmica com efeito de lanterna (*flashlight/spotlight*).
- Implementar comunicação em rede entre dois jogadores (modo host e cliente).
- Desenvolver sistema de câmara em primeira pessoa com controlos intuitivos.
- Adicionar elementos visuais como ambiente exterior, árvores e portal de saída.
- Implementar renderização de texto para interface com o utilizador.
- Aplicar texturas e materiais realistas aos elementos do jogo.

1.4 Estrutura do Relatório

Este relatório está organizado da seguinte forma:

- O **Capítulo 2** descreve as tecnologias e bibliotecas utilizadas no desenvolvimento do projeto.
- O **Capítulo 3** detalha o desenvolvimento e implementação, incluindo a gestão do projeto, descrição das classes e funções principais.
- O **Capítulo 4** apresenta as conclusões, trabalho realizado, limitações e sugestões para trabalho futuro.

Capítulo 2

Tecnologias Utilizadas

Este capítulo descreve as tecnologias, bibliotecas e ferramentas utilizadas no desenvolvimento do jogo de labirinto 3D.

2.1 OpenGL

O **OpenGL** (Open Graphics Library) [2] é a API gráfica principal utilizada neste projeto. Trata-se de uma API multiplataforma para renderização de gráficos 2D e 3D, amplamente utilizada na indústria de jogos e aplicações gráficas.

O projeto utiliza OpenGL moderno (versão 3.3+) com um *pipeline* programável, permitindo o uso de shaders personalizados para controlo total sobre a renderização. As principais funcionalidades do OpenGL utilizadas incluem:

- **Vertex Buffer Objects (VBO)**: Para armazenamento eficiente de dados de vértices na GPU.
- **Vertex Array Objects (VAO)**: Para gestão de estado de atributos de vértices.
- **Shaders GLSL**: Programas executados na GPU para processamento de vértices e fragmentos.
- **Texturas**: Mapeamento de imagens em superfícies 3D para realismo visual.
- **Depth Testing**: Para renderização correta de objetos com base na profundidade.

2.2 GLFW

A biblioteca **GLFW** [3] é utilizada para gestão de janelas, contextos OpenGL e processamento de entrada (teclado e rato). GLFW é uma biblioteca leve e portável que simplifica a criação de aplicações OpenGL multiplataforma.

Principais funcionalidades utilizadas:

- Criação e gestão de janela de renderização
- Inicialização de contexto OpenGL

- Gestão de eventos de entrada (teclado, rato, movimento de rato)
- Controlo de cursor (normal/escondido para modo de jogo)

2.3 GLM

A biblioteca **GLM** (OpenGL Mathematics) [4] fornece funcionalidades matemáticas essenciais para programação gráfica 3D. GLM implementa tipos e operações compatíveis com GLSL, o que facilita a transferência de dados entre CPU e GPU.

Funcionalidades utilizadas:

- Vetores 3D (`glm::vec3`) para posições, direções e cores
- Matrizes 4x4 (`glm::mat4`) para transformações
- Funções de transformação: `translate`, `rotate`, `scale`
- Projeção perspectiva (`glm::perspective`)
- Cálculos trigonométricos e normalização de vetores

2.4 GLAD

O **GLAD** é um carregador de extensões OpenGL que gere a obtenção de ponteiros para funções OpenGL. É necessário porque as funções OpenGL modernas não estão disponíveis diretamente no sistema operativo, uma vez que devem ser carregadas em tempo de execução.

2.5 FreeType

A biblioteca **FreeType** [5] é utilizada para renderização de texto na interface do jogo. FreeType permite o carregamento de fontes TrueType e a geração de texturas de caracteres para renderização em OpenGL.

No projeto, FreeType é utilizado para:

- Renderização do diálogo introdutório
- Mensagens de estado do jogo
- Interface de utilizador

2.6 STB Image

A biblioteca **STB Image** [6] é uma biblioteca de cabeçalho único (*header-only*) para carregamento de imagens em diversos formatos (PNG, JPEG, etc.). É utilizada para carregar as texturas aplicadas às paredes, chão e ambiente exterior do labirinto.

2.7 CMake

O sistema de construção **CMake** [7] é utilizado para gestão da compilação do projeto. CMake permite definir dependências, configurar opções de compilação e gerar ficheiros de projeto para diferentes sistemas operativos e IDEs.

O projeto está configurado para gerar dois executáveis distintos:

- `maze_host.exe`: Versão host/servidor
- `maze_client.exe`: Versão cliente

2.8 Sockets TCP

Para a comunicação em rede entre o host e o cliente, o projeto utiliza **sockets TCP** POSIX. Esta escolha permite comunicação fiável entre processos através de uma interface de programação padrão disponível em sistemas Unix-like.

2.9 Resumo de Bibliotecas

A Tabela 2.1 resume todas as bibliotecas externas utilizadas e os seus propósitos.

Tabela 2.1: Bibliotecas externas utilizadas no projeto.

Biblioteca	**Propósito**
OpenGL	API gráfica para renderização 3D
GLFW	Gestão de janela e entrada
GLM	Matemática 3D (vetores, matrizes, transformações)
GLAD	Carregamento de extensões OpenGL
FreeType	Renderização de texto
STB Image	Carregamento de texturas (PNG, JPEG)
CMake	Sistema de construção multiplataforma
Sockets TCP	Comunicação em rede host/cliente

Capítulo 3

Desenvolvimento e Implementação

Este capítulo descreve a gestão do projeto, a arquitetura do código, e as principais funcionalidades implementadas.

3.1 Gestão do Projeto

3.1.1 Estrutura de Diretorias

O projeto está organizado da seguinte forma:

- `src/`: Código fonte C++ (.cpp)
- `include/`: Ficheiros de cabeçalho (.h, .hpp)
- `shaders/`: Shaders GLSL (vertex e fragment shaders)
- `assets/`: Recursos (texturas, fontes)
- `build/`: Diretoria de compilação (gerado pelo CMake)
- `docs/`: Documentação do projeto

3.1.2 Sistema de Compilação

O projeto utiliza CMake para gestão da compilação. O ficheiro `CMakeLists.txt` define:

- Dependências externas (GLFW, FreeType, OpenGL)
- Dois executáveis distintos: `maze_host` e `maze_client`
- Ficheiros fonte comuns partilhados entre ambos os executáveis
- Comandos personalizados para cópia de shaders e assets para o Diretoria de compilação

3.1.3 Carregamento de Assets Portável

Para garantir que o jogo funciona corretamente em qualquer máquina, foi implementado um sistema de carregamento de assets portável.

- O ficheiro `configuration/root_directory.h.in` define uma variável `logl_root` que aponta para o Diretoria de assets.
- Esta variável foi configurada para `"."` (Diretoria atual), o que permite que o jogo procure assets relativamente ao executável.
- O CMake copia automaticamente as pastas `assets` e `shaders` para junto do executável na pasta `build`.
- A classe `FileSystem` utiliza este caminho relativo para resolver todos os caminhos de ficheiros em tempo de execução.

A separação em dois executáveis permite executar o jogo em modo host e cliente simultaneamente para testar funcionalidades de rede.

3.2 Descrição do Código

Nota de Atribuição: As classes base para gestão de Shaders, Câmara, Mesh e sistema de ficheiros foram adaptadas a partir dos tutoriais e código disponibilizado pelo portal *LearnOpenGL* [8]. Estas classes fornecem uma abstração robusta sobre as primitivas do OpenGL, o que permitiu focar o desenvolvimento na lógica específica do jogo e na geração procedural.

3.2.1 Classe Game

A classe `Game` é o núcleo da aplicação, responsável pela gestão do ciclo de vida do jogo. Localizada em `src/Game.cpp` e `include/Game.h`, implementa os seguintes métodos principais:

- `Init()`: Inicialização de recursos (shaders, geometria, texturas, rede)
- `ProcessInput(float dt)`: Processamento de entrada do teclado e rato
- `Update(float dt)`: Lógica do jogo (networking, deteção de proximidade ao portal)
- `Render()`: Renderização da cena 3D
- `ProcessMouseMovement()`: Controlo de câmara com o rato

- `CheckPortalProximity()`: Verificação de proximidade ao portal de saída

A classe gere também o estado do jogo incluindo:

- Modo de jogo (HOST ou CLIENT)
- Estado de pausa
- Estado do diálogo introdutório
- Bloqueio de movimento (para modo cliente)
- Sockets de rede para comunicação

3.2.2 Classe Maze

A classe `Maze` (`src/Maze.cpp`, `include/Maze.h`) é responsável pela representação e renderização do labirinto. Principais funcionalidades:

- `Generate(int width, int height)`: Gera um novo labirinto com recurso ao algoritmo de Kruskal
- `Draw(Shader &shader)`: Renderiza paredes e chão do labirinto
- `IsWall(float x, float z)`: Deteção de colisões com paredes
- Armazena a grelha do labirinto (matriz 2D de inteiros)
- Gere os *meshes* de parede e chão

3.2.3 Classe Camera

A classe `Camera` (`include/learnopengl/camera.h`) implementa uma câmara em primeira pessoa com os seguintes recursos:

- Movimento WASD para navegação
- Rotação da câmara com o rato
- Cálculo de matriz de visualização (*view matrix*)
- Vetores direcionais (frente, direita, cima)
- Campo de visão ajustável (*zoom*)

3.2.4 Classe Shader

A classe `Shader` (`include/Shader.h`) encapsula a gestão de programas shader GLSL:

- Carregamento e compilação de vertex e fragment shaders
- Ligação (*linking*) de programa shader
- Métodos utilitários para definição de uniformes (matrizes, vetores, valores escalares)
- Validação de erros de compilação

3.2.5 Geração de Labirinto: Algoritmo de Kruskal

A classe `maze::kruskal` (`include/kruksal/kruksal.cpp`) implementa o algoritmo de Kruskal para geração procedural de labirintos perfeitos (sem ciclos).

Nota de Atribuição: A implementação do algoritmo de Kruskal foi adaptada do repositório de Ferenc Nemeth [9], que fornece implementações de referência de vários algoritmos de geração de labirintos em C++.

Funcionamento do algoritmo:

1. Inicializa cada célula como um conjunto disjunto
2. Cria lista de todas as paredes possíveis entre células adjacentes
3. Baralha aleatoriamente a lista de paredes
4. Para cada parede:
    - Se as células adjacentes pertencem a conjuntos diferentes
    - Remove a parede
    - Une os conjuntos
5. Continua até que todas as células pertençam ao mesmo conjunto

Este método garante que existe sempre um caminho único entre quaisquer dois pontos do labirinto.

3.2.6 Classe TextRenderer

A classe `TextRenderer` (`src/TextRenderer.cpp`) utiliza FreeType para renderização de texto:

- `Load(std::string font, unsigned int fontSize)`: Carrega fonte TrueType

- `RenderText()`: Renderiza texto 2D sobre a cena 3D
- Gera texturas de caracteres dinamicamente
- Utiliza projeção ortográfica para renderização 2D

3.2.7 Classe Mesh

A classe `Mesh` (`include/Mesh.hpp`) encapsula geometria 3D:

- Armazena vértices (posição, normal, coordenadas de textura)
- Gere VAO, VBO para renderização eficiente
- Suporta múltiplas texturas (difusa, normal, rugosidade)
- Método `Draw()` para renderização

3.2.8 Módulo Network

As funções de networking (`src/network.cpp`, `include/Network.h`) implementam comunicação TCP:

- `createSocket()`: Cria socket TCP
- `bindAndListen()`: Inicia servidor (modo host)
- `acceptConnection()`: Aceita ligação de cliente
- `connectToServer()`: Liga ao servidor (modo cliente)
- `sendData()`, `receiveData()`: Transferência de dados
- `closeSocket()`: Encerramento de ligação

3.3 Funcionalidades Implementadas

3.3.1 Geração Procedural de Labirintos

Cada execução do jogo gera um labirinto único através do algoritmo de Kruskal. O tamanho do labirinto é configurável (atualmente 15x15 células).

3.3.2 Sistema de Iluminação Dinâmica

O jogo implementa um sistema de iluminação tipo lanterna (*flashlight/spotlight*) através de shaders GLSL personalizados:

- **Spotlight**: A luz segue a posição e direção da câmara
- **Atenuação**: Diminuição gradual da intensidade luminosa com a distância
- **Ângulos de corte**: Controlo do cone de luz (ângulo interno e externo)
- **Fog Effect**: Névoa atmosférica que esconde geometria distante

O vertex shader (`shaders/blinn_phong.vert`) transforma vértices e calcula posições para iluminação. O fragment shader (`shaders/blinn_phong.frag`) implementa o modelo de iluminação de **Blinn-Phong**, que utiliza o vetor *halfway* para reflexões especulares mais realistas e eficientes, integrando ainda características de spotlight.

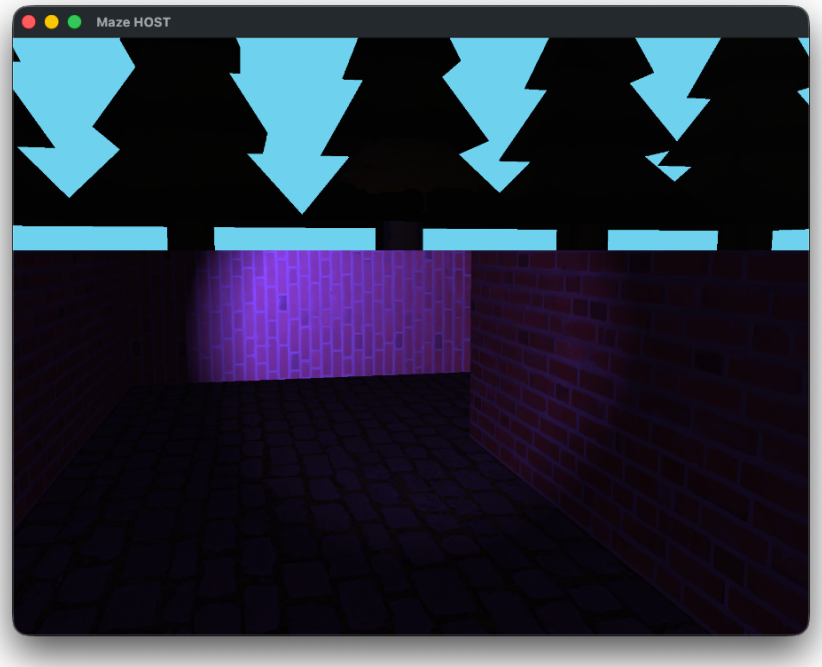

Figura 3.1: Efeito de lanterna (Spotlight) e atenuação da luz nas paredes do labirinto.

3.3.3 Networking Host/Cliente

O jogo suporta dois modos de execução:

Modo HOST:

- Inicia servidor TCP na porta 8080
- Aguarda ligação de cliente
- Ao atingir o portal, envia mensagem "UNLOCK" ao cliente
- Permite movimento livre desde o início

Modo CLIENT:

- Liga ao servidor host (127.0.0.1:8080)
- Movimento bloqueado até receber mensagem "UNLOCK"
- Após desbloqueio, pode navegar até ao próprio portal
- Apresenta sequência de fim ao atingir portal

3.3.4 Sistema de Diálogo Introdutório

Ao iniciar o jogo, é apresentado um diálogo com:

- Mensagem de boas-vindas
- Indicação do modo de jogo (HOST/CLIENT)
- Objetivo do jogo
- Controlos disponíveis
- Instrução para pressionar ENTER e iniciar

Durante a apresentação do diálogo, a entrada de movimento é ignorada.

3.3.5 Ambiente Exterior

Para além do labirinto, o jogo inclui:

- **Plano de relva**: Superfície texturizada que estende 10 células além do labirinto
- **Árvores procedurais**: Árvores simples compostas por tronco (cilindro) e copa (pirâmide)
- **Portal de saída**: Esfera flutuante com efeito visual de "Mercúrio Líquido/Prata" (*Liquid Silver*), gerado proceduralmente via shaders GLSL, que pulsa e brilha, e atua como o objetivo final do jogo.

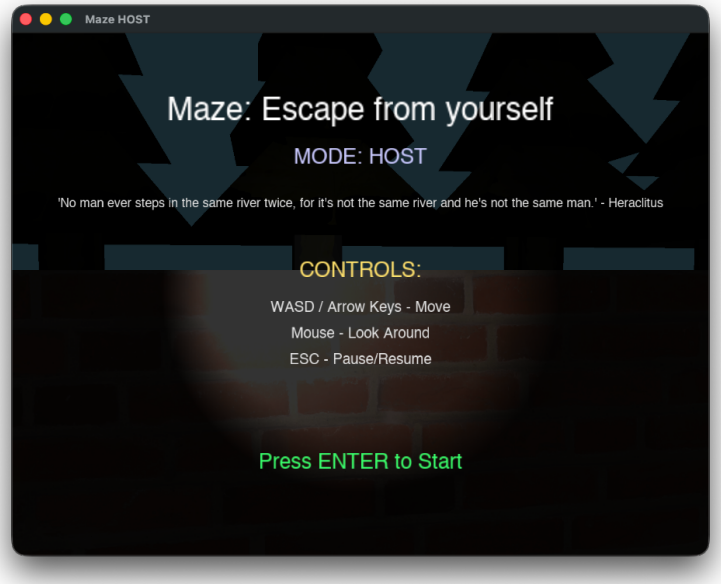

Figura 3.2: Menu introdutório renderizado com FreeType sobre a cena 3D.

3.3.6 Deteção de Colisões

O sistema de colisões verifica:

- Interseção do jogador com paredes do labirinto
- Movimento é bloqueado em direções que causariam colisão
- Permite movimento deslizante ao longo de paredes
- **Prevenção de Clipping**: Implementado um raio de colisão (`playerRadius = 0.3f`) em torno da câmara para impedir que o plano de visualização atravesse as esquinas das paredes.

3.3.7 Sistema de Pausa

Pressionando a tecla ESC:

- O jogo pausa/despausa
- Cursor do rato torna-se visível (modo pausa)
- Movimento e rotação da câmara são bloqueados

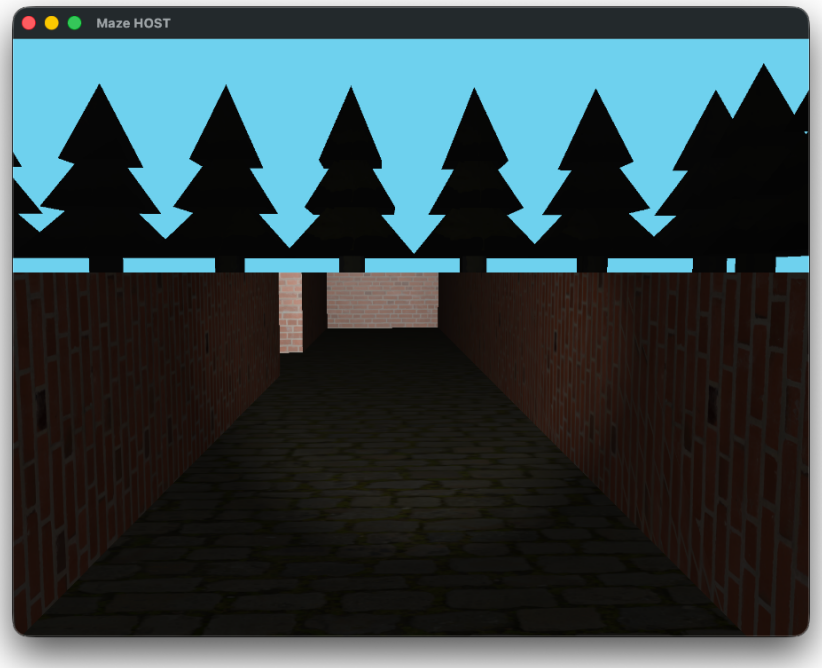

Figura 3.3: Ambiente exterior mostrando as árvores procedurais e o portal ao fundo.

3.3.8 Transição de Cor Progressiva

Uma funcionalidade adicional implementada é a transição gradual da cor do ambiente à medida que o jogador se aproxima do portal. Este efeito visual cria feedback atmosférico e ajuda o jogador a perceber a proximidade ao objetivo.

Implementação:

- Cálculo da distância entre câmara e portal
- Normalização da distância (0-1) em relação à largura total do labirinto
- Aplicação de *smoothstep* para interpolação suave
- Transição de branco (normal) para roxo/místico (próximo do portal)
- Aplicação do *tint* no fragment shader antes do efeito de névoa

A função `GetEnvironmentTint()` calcula o fator de interpolação usando a fórmula:

$$t = t^2(3 - 2t) \quad (3.1)$$

Este *smoothstep* garante uma transição visual suave, sem mudanças abruptas de cor. Antes era feita a formula à mão, mas agora usamos a função `glm::smoothstep`.

Figura 3.4: Overlay de Pausa renderizado quando o jogador pressiona ESC.

3.3.9 Shader de Metal Líquido (Portal)

Para criar o efeito do portal "Mercúrio Líquido", foi implementado um shader procedural em GLSL que substitui a renderização de texturas padrão quando a flag `isPortal` está ativa.

Características do efeito:

- **Animação Procedural**: Combinação de três ondas sinusoidais (`sin/cos`) com frequências e fases diferentes, animadas pelo tempo (`uniform float time`), o que cria um padrão fluido e orgânico.
- **Efeito Fresnel**: As bordas da esfera são iluminadas intensamente (efeito de borda/rim light) baseado no ângulo de visão (`dot(viewDir, normal)`), o que simula um material metálico ou energético.
- **Cores**: Mistura de uma base prateada azulada `(0.7, 0.75, 0.8)` com tons de branco puro nos picos das ondas e um leve brilho ciano nas bordas.
- **Pulsação**: A intensidade global da cor oscila com o tempo, o que dá a impressão de um objeto "vivo".

3.3.10 Texturas e Materiais

O jogo utiliza texturas PBR (Physically-Based Rendering) de alta qualidade (4K):

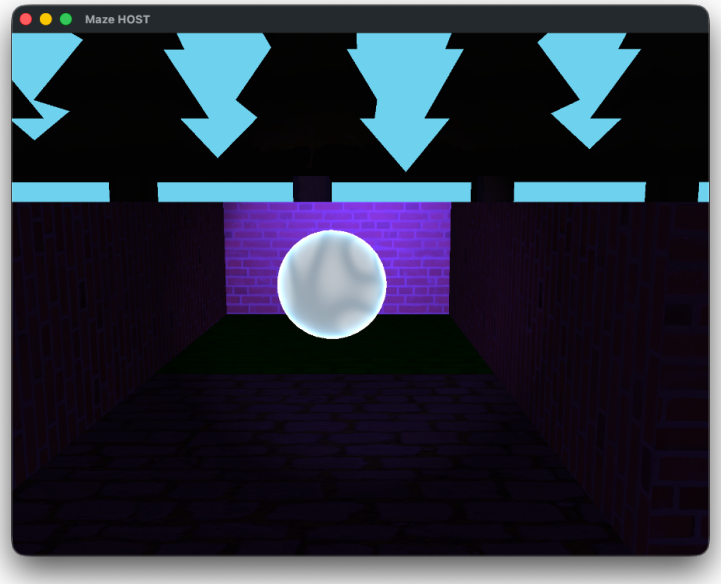

Figura 3.5: Renderização do Portal com efeito "Mercúrio Líquido" e iluminação Fresnel.

- **Paredes**: Textura de tijolos (Bricks101) com mapas de cor, normal e rugosidade
- **Chão**: Textura de pedras pavimentadas (PavingStones138)
- **Exterior**: Textura de relva (Grass005)

3.3.11 Minimapa (HUD)

Foi implementado um minimapa 2D no canto superior direito do ecrã para auxiliar a navegação do jogador.

Implementação Técnica:

- **Renderização**: Utiliza um shader simples (`simpleShader`) que desenha apenas cores sólidas, sem texturas ou iluminação.
- **Projeção**: Usa uma matriz de projeção ortográfica (`glm::ortho`) para mapear coordenadas de ecrã (pixels) diretamente, ignorando-se a perspetiva 3D.
- **Conteúdo**:
  - **Fundo**: Retângulo cinzento escuro semitransparente.
  - **Labirinto**: Itera sobre a estrutura de dados `grid` do labirinto e desenha quadrados pretos para cada parede.
  - **Jogador**: Um quadrado vermelho representa a posição atual da câmara, atualizada em tempo real.

O minimapa ignora o *Depth Test* durante a sua renderização para garantir que aparece sempre sobreposto à cena 3D (HUD).

Figura 3.6: Interface de jogo com o Minimapa no canto superior direito, mostrando a posição do jogador (vermelho) e as paredes (preto).

3.4 Documentação com Doxygen

Todas as classes e ficheiros de cabeçalho do projeto foram documentados utilizando o formato Doxygen. Isto inclui descrições detalhadas de:

- Classes principais (`Game`, `Maze`, `Camera`)
- Utilitários (`Objloader`, `Texture`, `TextRenderer`)
- Estruturas de dados e definições de tipos

A documentação HTML pode ser gerada executando o comando:

```
doxygen Doxyfile
```

Esta documentação técnica facilita a manutenção futura e a compreensão da arquitetura por novos programadores.

Capítulo 4

Conclusão e Trabalho Futuro

4.1 Conclusão

Este projeto atingiu com sucesso os seus objetivos principais, o que resultou numa aplicação 3D interativa funcional que demonstra a aplicação prática de conceitos fundamentais de computação gráfica.

4.1.1 Trabalho Realizado

As seguintes funcionalidades foram implementadas com sucesso:

- **Motor de Renderização 3D**: Sistema completo baseado em OpenGL com *pipeline* programável e shaders GLSL customizados.
- **Geração Procedural**: Implementação do algoritmo de Kruskal para geração de labirintos perfeitos, permitindo experiências únicas em cada execução.
- **Sistema de Iluminação**: Iluminação dinâmica tipo lanterna com atenuação realista e efeito de névoa atmosférica, o que cria uma atmosfera imersiva.
- **Networking Multiplayer**: Sistema de comunicação TCP entre host e cliente com mecânica de desbloqueio progressivo.
- **Câmara FPS**: Controlos de primeira pessoa intuitivos com movimento WASD e rotação com rato.
- **Ambiente 3D Completo**: Para além do labirinto, inclusão de ambiente exterior com árvores procedurais, portal arquitetónico e texturas realistas.
- **Interface de Utilizador**: Sistema de renderização de texto para diálogos e feedback ao jogador.
- **Física Básica**: Deteção de colisões e impedimento de atravessamento de paredes.
- **Texturas PBR**: Utilização de texturas de alta qualidade com mapas de normal e rugosidade para realismo visual.

- **Transição de Cor Progressiva**: Sistema de mudança gradual da cor do ambiente baseado na proximidade ao portal, proporcionando feedback visual atmosférico ao jogador.

O desenvolvimento deste projeto proporcionou aprendizagem valiosa em áreas como:

- Programação gráfica com OpenGL moderno
- Arquitetura de software para aplicações em tempo real
- Álgebra linear aplicada (transformações 3D, projeções)
- Programação de shaders GLSL
- Gestão de recursos (texturas, *meshes*, shaders)
- Networking com sockets TCP
- Algoritmos de geração procedural

4.2 Limitações e Trabalho Não Realizado

Apesar do sucesso geral do projeto, algumas limitações foram identificadas:

- **Otimização**: Não foram implementadas técnicas avançadas de otimização como *frustum culling* ou *occlusion culling*.
- **Sincronização de Rede**: O networking é assíncrono básico sem sincronização completa de posições entre jogadores.
- **Áudio**: Não foi implementado sistema de som ou música.

4.3 Trabalho Futuro

O projeto tem margem significativa para evolução. Sugestões para versões futuras incluem:

- **Níveis Múltiplos**: Sistema de progressão com múltiplos labirintos de dificuldade crescente.
- **Algoritmos Alternativos**: Implementação de outros algoritmos de geração (Prim, Recursive Backtracking, Eller) com seleção pelo utilizador.
- **Sistema de Áudio**: Música de fundo, efeitos sonoros de passos, sons ambientes.

- **Power-ups e Obstáculos**: Elementos de *gameplay* adicionais como chaves, portas trancadas, armadilhas.
- **IA de Inimigos**: Adicionar entidades controladas por IA que perseguem o jogador.
- **Opção de solução**: Como há no mesmo repositório de onde tiramos o algoritmo de Kruskal vários algoritmos de solução de labirintos, podiamos implementar uma opção que permitisse ao jogador resolver o labirinto automaticamente.
- **Interface de Utilizador**: Implementação de uma interface de utilizador mais agradável e intuitiva.

4.4 Considerações Finais

Este projeto demonstrou com sucesso a aplicação de conceitos de computação gráfica numa aplicação prática e interativa. A combinação de geração procedural, renderização 3D, iluminação dinâmica e networking resultou numa experiência de jogo funcional e tecnicamente interessante.

O desenvolvimento reforçou a importância de uma arquitetura de software bem estruturada, da gestão cuidadosa de recursos gráficos, e da compreensão profunda dos conceitos matemáticos subjacentes à computação gráfica 3D.

Bibliografia

[1] Comunidade UBI, "Template LaTeX UBI - template não oficial para teses e dissertações da universidade da beira interior," https://github.com/manoelcampos/template-ubi-latex, accessed: 2025-12-22. 1

[2] Khronos Group, "OpenGL - the industry standard for high performance graphics," https://www.opengl.org/, accessed: 2025-12-22. 3

[3] GLFW Contributors, "GLFW - an opengl library," https://www.glfw.org/, accessed: 2025-12-22. 3

[4] G-Truc Creation, "OpenGL Mathematics (GLM)," https://github.com/g-truc/glm, accessed: 2025-12-22. 4

[5] The FreeType Project, "FreeType - a free, high-quality, and portable font engine," https://www.freetype.org/, accessed: 2025-12-22. 4

[6] S. Barrett, "stb - single-file public domain libraries for c/c++," https://github.com/nothings/stb, accessed: 2025-12-22. 5

[7] Kitware, "CMake - cross-platform make," https://cmake.org/, accessed: 2025-12-22. 5

[8] J. de Vries, "LearnOpenGL - learn modern opengl graphics programming," https://learnopengl.com/, accessed: 2025-12-22. 8

[9] F. Nemeth, "Maze generation algorithms," https://github.com/ferenc-nemeth/maze-generation-algorithms, accessed: 2025-12-22. 10