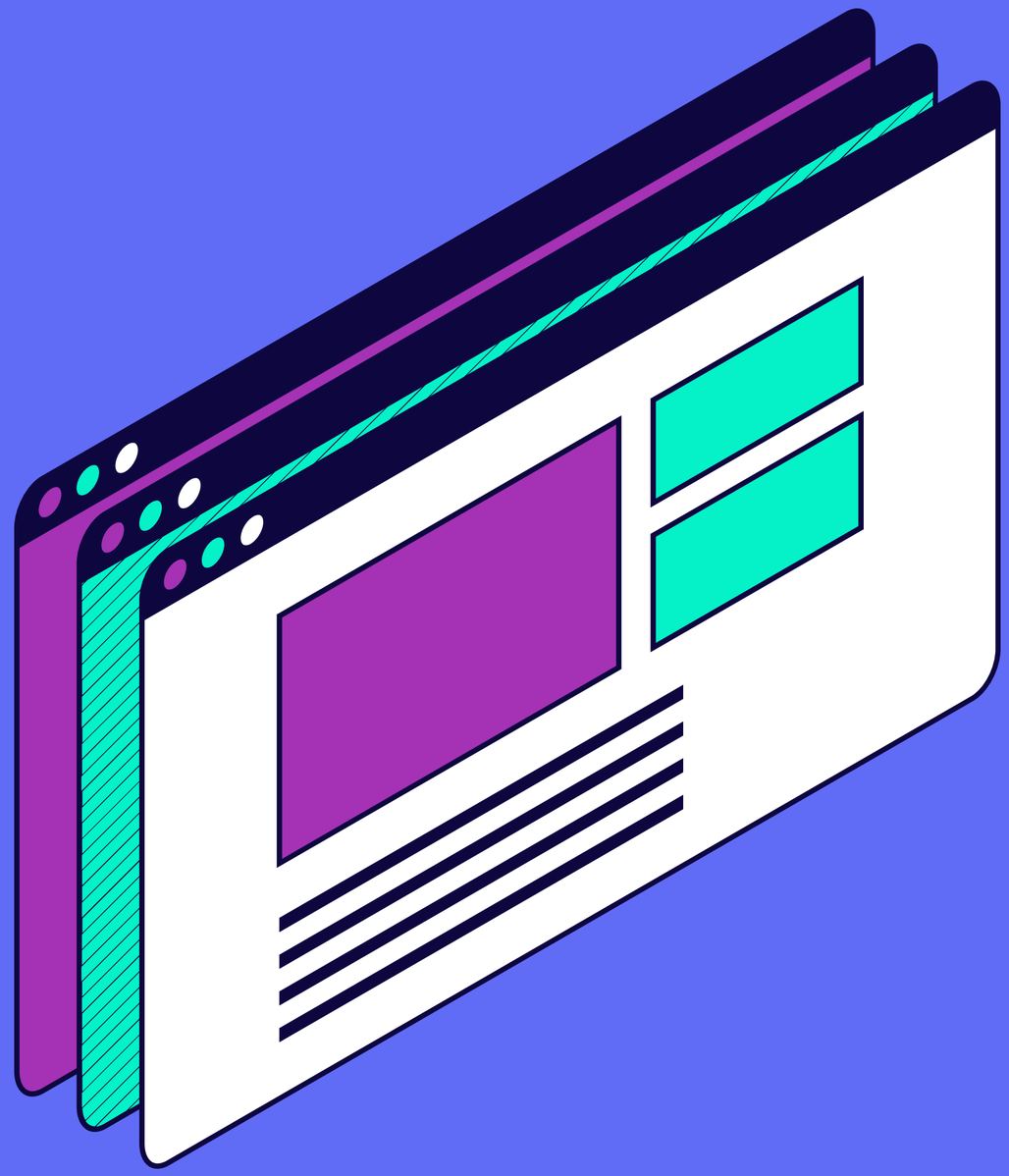
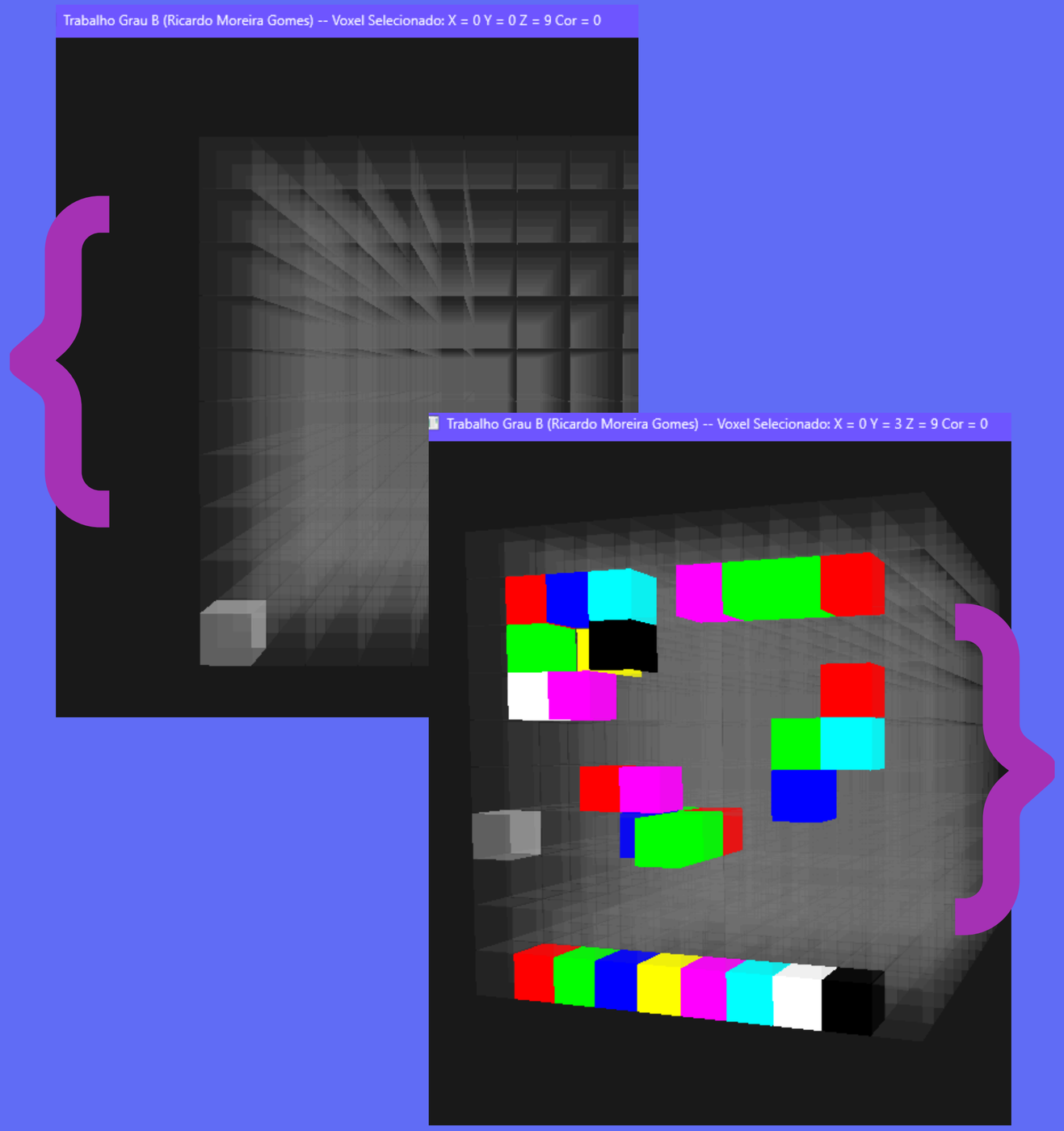
Fundamentos de
Computação Gráfica
2025/1

Trabalho Grau B

Ricardo Moreira Gomes

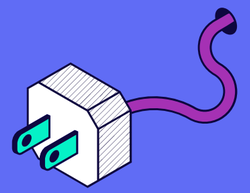
Índice





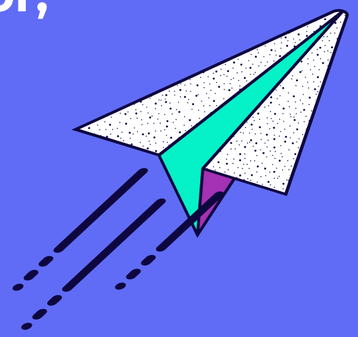
Demonstração & Funcionamento

- Conjunto semelhante ao construído ao longo das aulas;
- Movimentação via grid, câmera e zoom;
- Mostrar, esconder e trocar;
- Save/load de cenas;
- Efeitos sonoros especiais;
- UI via janela da aplicação.



Estrutura geral do código

- Voxel + Iteração sobre o que foi visto em aula;
- Novidades:
  - fstream, windows, mmsystem, winmm.lib;
  - variáveis de otimização para UI;
  - novas funções (salvarCena, carregarCena, atualizaTituloJanela, playSound);
  - alteração na mudança de cor;
  - playSound nas ações.



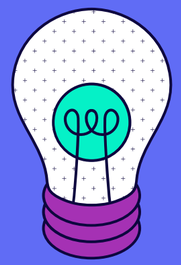
```
// adição pra salvar/carregar cenas txt
#include <fstream>

// configs arquivos de áudio pra windows
#include <windows.h>
#include <mmsystem.h> // playSound
#pragma comment(lib, "winmm.lib") // linka com a biblioteca de som

// função para tocar um som (.wav)
void playSound(const char* filename) {
    if (!PlaySoundA(filename, NULL, SND_FILENAME | SND_ASYNC)) {
        std::cerr << "Erro ao tocar: " << filename << std::endl;
        Beep(440, 100); // fallback se tudo der errado
    }
}
```

```
// muda a cor do voxel
bool mudouCor = false;
if (key == GLFW_KEY_C && action == GLFW_PRESS)
{
    playSound("change_color.wav");
    int corAtual = grid[selecaoY][selecaoX][selecaoZ].corPos;
    int totalCores = sizeof(colorList) / sizeof(glm::vec4);
    corAtual = (corAtual + 1) % totalCores;
    grid[selecaoY][selecaoX][selecaoZ].corPos = corAtual;
    mudouCor = true;
    printf("Troquei a cor para %d\n", corAtual);
}
```

Estrutura dos buffers e shaders



- Pipeline da OpenGL moderna;
- setupGeometry()
  - configura os vértices de um cubo;
  - criação e configuração de VBO/VAO.
- Vertex shader (transform, câmera, projeção) + Fragment shader (cor do voxel atual);
- setupShader() faz a compilação dos shaders;
- envio de model, view, proj e uColor → funções;
- desenho dos cubos via loop de renderização + glDrawArrays();

```
GLuint setupGeometry()
{
    GLfloat vertices[] = {
        // frente
        0.5, 0.5, 0.5, 0.5, -0.5, 0.5, -0.5, -0.5, 0.5,
        -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5, 0.5,
        // trás
        0.5, 0.5, -0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
        -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, -0.5, -0.5,
        // esquerda
        -0.5, -0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5, -0.5,
        -0.5, -0.5, -0.5, -0.5, 0.5, -0.5, -0.5, 0.5, 0.5,
        // direita
        0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5,
        0.5, -0.5, -0.5, 0.5, 0.5, -0.5, 0.5, 0.5, 0.5,
        // baixo
        -0.5, -0.5, 0.5, 0.5, -0.5, 0.5, 0.5, -0.5, -0.5,
        0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.5,
        // cima
        -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5,
        0.5, 0.5, -0.5, -0.5, 0.5, -0.5, -0.5, 0.5, 0.5};

    GLuint VBO, vao;
    glGenVertexArrays(1, &vao);
    glGenBuffers(1, &VBO);

    glBindVertexArray(vao);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (GLvoid *)0);
    glEnableVertexAttribArray(0);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);

    return vao;
```

Gerenciamento dos Voxels

- Grade tridimensional 10 x 10 x 10 (1000 voxels);
- Instâncias struct Voxel (pos, fatorEscala, visivel, selecionado, corPos);
- Inicialização → definição manual da posição;
- Transformações são individuais;
- Selecionado + Visibilidade (V / DELETE) + Troca Cor (C);
  - Cor → indíce no array global colorList[];
  - Movimentação via setinhas (x, y) e pgup/down (z);
- Possível salvar cena atual (K) ou dar load (L);

```
struct Voxel
{
    glm::vec3 pos;
    float fatorEscala;
    bool visivel = true, selecionado = false;
    int corPos;
};

int selecaoX, selecaoY, selecaoZ;
const int TAM = 10;
Voxel grid[TAM][TAM][TAM];

glm::vec4 colorList[] = {
    {0.5f, 0.5f, 0.5f, 0.1f}, // cinza     0
    {1.0f, 0.0f, 0.0f, 1.0f}, // vermelho  1
    {0.0f, 1.0f, 0.0f, 1.0f}, // verde     2
    {0.0f, 0.0f, 1.0f, 1.0f}, // azul      3
    {1.0f, 1.0f, 0.0f, 1.0f}, // amarelo   4
    {1.0f, 0.0f, 1.0f, 1.0f}, // magenta   5
    {0.0f, 1.0f, 1.0f, 1.0f}, // ciano     6
    {1.0f, 1.0f, 1.0f, 1.0f}, // branco    7
    {0.0f, 0.0f, 0.0f, 1.0f}, // preto     8
};
```

Inserção, deleção, movimentação e câmera

- Por padrão, todos voxels são instanciados na cor genérica cinza (0.1f transparente);
  - Apertar DELETE → não visível;
  - Apertar V → visível;
  - SFX específico para feedback da ação;
- Movimentação pelo grid é feita pelas setinhas + pgUp e pgDown no eixo z;
- Movimentação da câmera (15.0f) é feita via WASD e direção do mouse;
- Scroll do mouse → altera zoom da cena.

```
// troca a visibilidade de um voxel selecionado
if (key == GLFW_KEY_DELETE && action == GLFW_PRESS)
{
    grid[selecaoY][selecaoX][selecaoZ].visivel = false;
    playSound("show_hide.wav");
}
if (key == GLFW_KEY_V && action == GLFW_PRESS)
{
    grid[selecaoY][selecaoX][selecaoZ].visivel = true;
    playSound("show_hide.wav");
}
```

```
if (key == GLFW_KEY_RIGHT && action == GLFW_PRESS)
{
    if (selecaoX + 1 < TAM)
    {
        grid[selecaoY][selecaoX][selecaoZ].selecionado = false;
        selecaoX++;
        mudouSelecao = true;
        grid[selecaoY][selecaoX][selecaoZ].selecionado = true;
    }
}
if (key == GLFW_KEY_LEFT && action == GLFW_PRESS)
{
    if (selecaoX - 1 >= 0)
    {
        grid[selecaoY][selecaoX][selecaoZ].selecionado = false;
        selecaoX--;
        mudouSelecao = true;
        grid[selecaoY][selecaoX][selecaoZ].selecionado = true;
    }
}
```

Sistema de Save/Load de Cenas;

```cpp
void salvarCena(const std::string &nomeArquivo)
{
    std::ofstream out(nomeArquivo);
    if (!out.is_open())
    {
        std::cerr << "Erro ao abrir arquivo para salvar." << std::endl;
        return;
    }
    for (int x = 0; x < TAM; ++x)
    {
        for (int y = 0; y < TAM; ++y)
        {
            for (int z = 0; z < TAM; ++z)
            {
                const Voxel &v = grid[y][x][z];
                if (v.visivel)
                {
                    out << x << ' ' << y << ' ' << z << ' ' << v.corPos << '\n';
                }
            }
        }
    }
    out.close();
    std::cout << "Cena salva em: " << nomeArquivo << std::endl;
}
```

```cpp
void carregarCena(const std::string &nomeArquivo)
{
    std::ifstream in(nomeArquivo);
    if (!in.is_open())
    {
        std::cerr << "Erro ao abrir arquivo para carregar." << std::endl;
        return;
    }

    // Limpa grid antes de carregar
    for (int x = 0; x < TAM; ++x)
        for (int y = 0; y < TAM; ++y)
            for (int z = 0; z < TAM; ++z)
            {
                grid[y][x][z].visivel = false;
                grid[y][x][z].corPos = 0;
            }

    int x, y, z, cor;
    while (in >> x >> y >> z >> cor)
    {
        if (x >= 0 && x < TAM && y >= 0 && y < TAM && z >= 0 && z < TAM)
        {
            grid[y][x][z].visivel = true;
            grid[y][x][z].corPos = cor;
        }
    }
    in.close();
    std::cout << "Cena carregada de: " << nomeArquivo << std::endl;
}
```

Problemas Enfrentados e Futuras Atualizações

- Transparência do grid de voxels (necessários ajustes);
- Movimentação pelo grid pode ser confusa dependendo do ângulo da câmera (wasd x mouse x setas x pgup/down);
- Problema na rotação de cores;
- Dificuldade de usar a biblioteca de UI → topo da janela;
- Dificuldade de encontrar uma biblioteca simples para gerenciamento de áudio entre múltiplos SO → foco no windows;
- Futuras Atualizações:
  - Implementar UI via FreeType, se possível;
  - Colocar uma textura específica para cada cor da lista;
  - Permitir múltiplos salvamentos de cenas.

https://github.com/rikmgomes/fundcompgraf202501

Obrigado pela atenção!

Ricardo Moreira Gomes

{ Trabalho Grau B }

Fundamentos de Computação Gráfica 2025/1