GOVERNO DO ESTADO DO AMAZONAS
UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA

CURSO DE ENGENHARIA DE COMPUTAÇÃO

ALEXANDRO PANTOJA DOS SANTOS
2115080001

Computação Gráfica e PDI
Framework Freeglut

Trabalho realizado para obtenção de nota na disciplina de Computação Gráfica e PDI

Prof. Ricardo da Silva Barboza

MANAUS - AM
2025

1. Introdução

Este relatório tem como objetivo analisar a utilização de cores e transparências em computação gráfica utilizando a biblioteca OpenGL em conjunto com o framework FreeGLUT. FreeGLUT serve como uma camada de abstração para a criação de janelas e gerenciamento de eventos, permitindo que o desenvolvedor foque na renderização com OpenGL.

Serão utilizados dois códigos-fonte: colors.cpp, que demonstra o uso do padrão de cores RGB para criar objetos opacos, e transparent.cpp, que explora o padrão RGBA para aplicar efeitos de transparência.

2. Desenvolvimento

2.1 Código 1

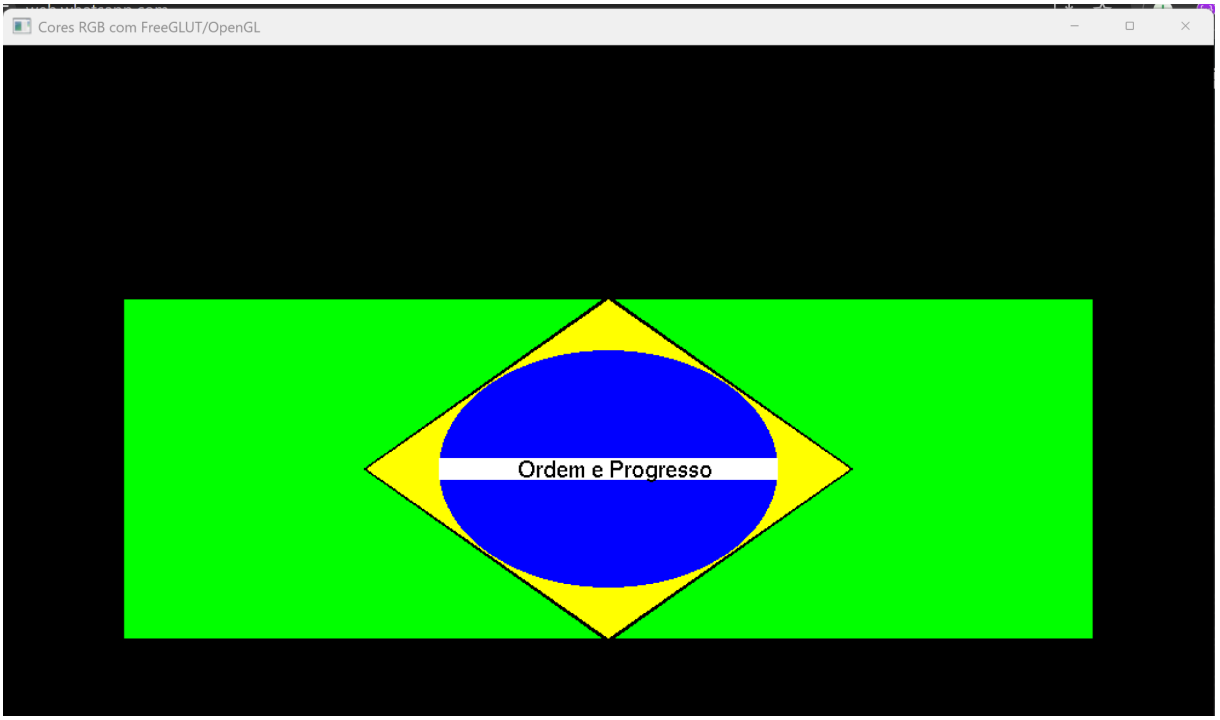

imagem 1: Bandeira do Brasil, criada no arquivo colors.cpp

```
#include <GL/freeglut.h>
#include <iostream>
#include <cmath>

// Função de desenho
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    float vx_inicial = 0.8f;
    float vx_final = 0.01f;
    float altura = 0.4f;

    // Esquerda
    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_QUAD_STRIP);
        glVertex2f(-vx_inicial, -altura);
        glVertex2f(-vx_final, -altura);
        glVertex2f(-vx_final, altura);
        glVertex2f(-vx_inicial, altura);
```

```
    glEnd();

    // Direita
    glColor3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_TRIANGLE_STRIP);
        glVertex2f(vx_final, -altura);
        glVertex2f(vx_inicial, -altura);
        glVertex2f(vx_inicial, altura);
        glVertex2f(vx_final, altura);
    glEnd();

    // Losango amarelo
    glColor3f(1.0f, 1.0f, 0.0f);
    glBegin(GL_POLYGON);
        glVertex2f(-0.4f, 0.0f);
        glVertex2f(0.0f, 0.4f);
        glVertex2f(0.4f, 0.0f);
        glVertex2f(0.0f, -0.4f);
    glEnd();

    // Círculo azul
    glColor3f(0.0f, 0.0f, 1.0f);
    float cx = 0.0f, cy = 0.0f, r = 0.28f;
    int num_segments = 100;
    glBegin(GL_TRIANGLE_FAN);
        glVertex2f(cx, cy); // centro
        for (int i = 0; i <= num_segments; ++i) {
            float theta = 2.0f * 3.1415926f * float(i) /
float(num_segments);
            float x = r * cosf(theta);
            float y = r * sinf(theta);
            glVertex2f(cx + x, cy + y);
        }
    glEnd();

    // Faixa branca horizontal Branca
    float faixaWidth = 0.56f;
    float faixaHeight = 0.05f;
    float faixaCenterY = 0.0f;

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
        glVertex2f(-faixaWidth / 2, faixaCenterY + faixaHeight / 2);
```

```
        glVertex2f(faixaWidth / 2, faixaCenterY + faixaHeight / 2);
        glVertex2f(faixaWidth / 2, faixaCenterY - faixaHeight / 2);
        glVertex2f(-faixaWidth / 2, faixaCenterY - faixaHeight / 2);
    glEnd();

    // Texto "Ordem e Progresso"
    glColor3f(0.0f, 0.0f, 0.0f);
    void *font = GLUT_BITMAP_HELVETICA_18;
    const char* texto = "Ordem e Progresso";

    // Calcula largura do texto
    int textoLen = strlen(texto);
    int textoWidth = 0;
    for (int i = 0; i < textoLen; ++i) {
        textoWidth += glutBitmapWidth(font, texto[i]);
    }

    // Posição inicial à esquerda da faixa branca
    float windowWidth = 1000.0f;
    float faixaLeftX = -faixaWidth / 2.0f; // coordenada normalizada da
borda esquerda da faixa
    float margem = 0.13f; // margem interna à esquerda

    glRasterPos2f(faixaLeftX + margem, faixaCenterY - 0.02f);
    for (int i = 0; i < textoLen; ++i) {
        glutBitmapCharacter(font, texto[i]);
    }

    glutSwapBuffers();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(1000, 700);
    glutCreateWindow("Cores RGB com FreeGLUT/OpenGL");

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

O arquivo colors.cpp tem como finalidade demonstrar a criação de uma cena complexa utilizando apenas cores opacas, definidas pelo padrão RGB (Red, Green, Blue).

- Função: O código renderiza uma representação da bandeira do Brasil. Ele utiliza primitivas geométricas do OpenGL como **GL_QUAD_STRIP** (para o fundo verde), **GL_POLYGON** (para o losango amarelo), **GL_TRIANGLE_FAN** (para o círculo azul) e **GL_QUADS** (para a faixa branca).
- Definição de Cor: A cor de cada objeto é definida pela função **glColor3f**(r, g, b), onde r, g, e b são valores de ponto flutuante entre 0.0 e 1.0. Por exemplo, **glColor3f**(0.0f, 1.0f, 0.0f) define a cor atual como verde puro. Como não há um canal alfa (transparência), todos os objetos são desenhados de forma opaca, sobrepondo-se completamente aos que foram desenhados anteriormente.

2.2 Código 2

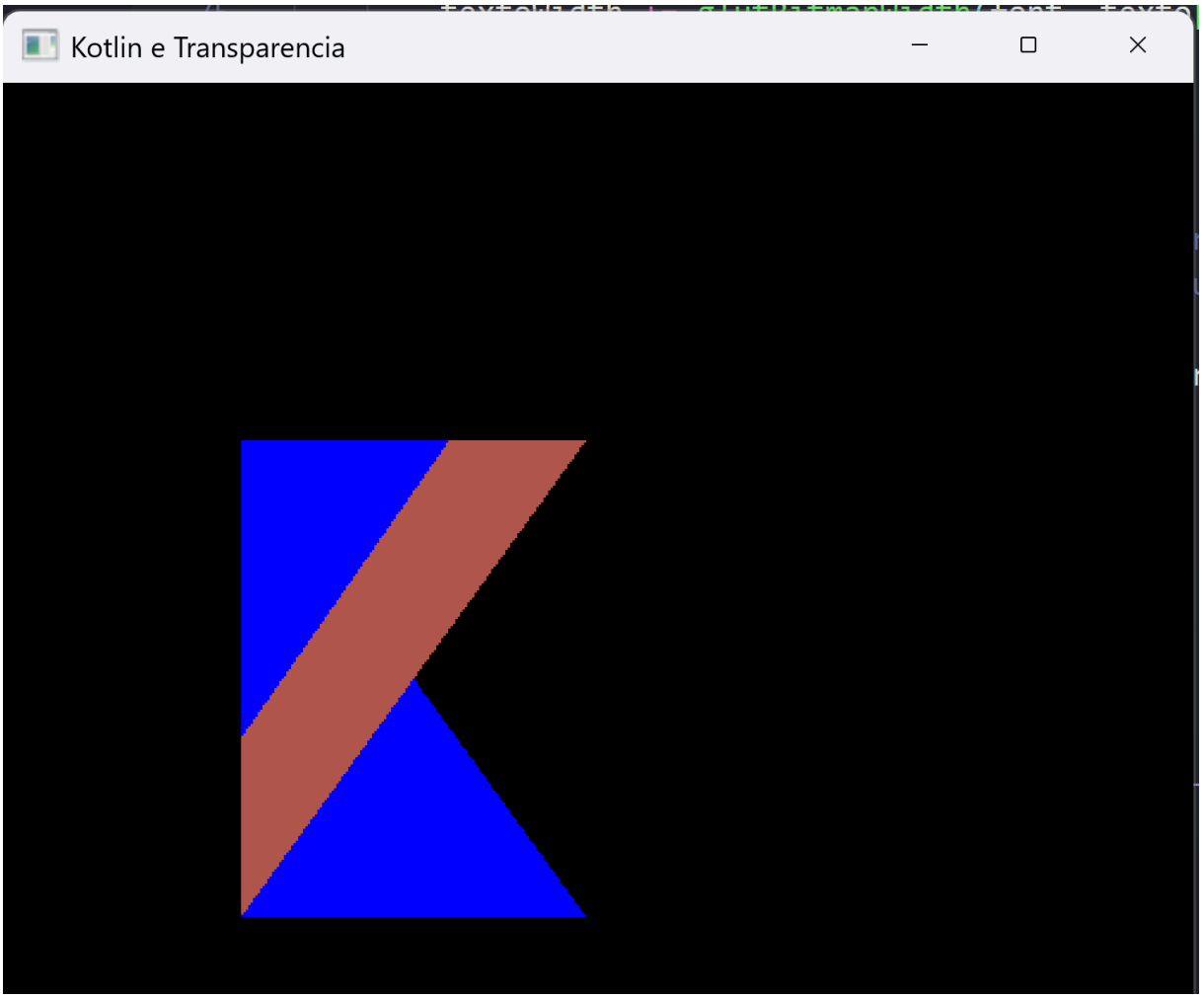

Imagem 2: Símbolo Koltin, criada no arquivo transparent.cpp

```cpp
#include <GL/freeglut.h>
#include <cmath>

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    float vx_inicial = 0.6f;
    float vx_final = 0.02f;
    float altura = 0.4f;
    glColor3f(0.0f, 0.0f, 1.0f); // Azul
    glBegin(GL_QUAD_STRIP);
        glVertex2f(-vx_inicial, -altura);
        glVertex2f(-vx_final, -altura);
```

```
        glVertex2f(-vx_final, altura);
        glVertex2f(-vx_inicial, altura);
    glEnd();

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glColor4f(1.0f, 0.5f, 0.0f, 0.7f);
    glBegin(GL_QUADS);
        glVertex2f(-0.6f, -0.1f);
        glVertex2f(-0.25f, 0.4f);
        glVertex2f(-0.02f, 0.4f);
        glVertex2f(-0.6f, -0.4f);
    glEnd();

    glDisable(GL_BLEND);

    glutSwapBuffers();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Kotlin e Transparencia");

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

O arquivo transparent.cpp foca em demonstrar como a transparência é implementada no OpenGL.

- Função: O código primeiro desenha uma forma azul opaca. Em seguida, desenha uma forma laranja semi-transparente por cima, permitindo que a forma azul seja parcialmente visível através dela.
- Definição de Cor e Transparência: Para alcançar o efeito de transparência, são necessários três passos:
- Habilitar o Blending: A função glEnable(**GL_BLEND**) ativa o "blending" (mistura de cores) da placa de vídeo.
- Definir a Função de Blending: glBlendFunc(**GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**) é a configuração mais comum. Ela instrui o

OpenGL a misturar a cor do novo fragmento (fonte) com a cor do fragmento já existente no buffer (destino) com base no valor alfa da fonte.

- Definir a Cor com Alpha: A cor da forma transparente é definida com **glColor4f**(r, g, b, a). O quarto parâmetro, a (alpha), controla a opacidade. Um valor de 0.7 (como no código) significa que o objeto é 70% opaco.

3. Resultados Obtidos

A análise dos códigos permitiu observar na prática a diferença entre os modelos de cores RGB e RGBA. Com colors.cpp, foi possível construir uma imagem estática complexa, onde a ordem de desenho é crucial para a sobreposição correta dos elementos opacos.

Com transparent.cpp, foi verificado que a simples definição de uma cor com canal alfa não é suficiente para gerar transparência. É mandatório habilitar e configurar o GL_BLEND para que o hardware realize a mistura de cores corretamente. O resultado foi a renderização bem-sucedida de um objeto semi-transparente, através do qual o objeto de fundo permaneceu visível, validando o processo de blending.

Conclui-se que FreeGLUT e OpenGL oferecem controle de baixo nível e alta performance sobre cores e transparência, sendo ferramentas poderosas para a criação de qualquer tipo de cena gráfica 2D ou 3D.