实验报告：GraphicsLibrary

董奕柳

2024 年 12 月 24 日

1 项目概述

这是一个简单的图形函数库，用于绘制基本的几何图形和文本显示，包含以下功能：

- 绘制直线段
- 绘制圆弧
- 绘制椭圆弧
- 多边形区域填充
- 显示名字

该项目采用 C++ 编写，并使用 CMake 构建工具。

2 功能设计

2.1 绘制直线段

直线段的绘制基于 Bresenham 算法，其核心思想是通过整数计算近似实现直线段的绘制，避免使用浮点运算，从而提高效率。算法的数学公式及伪代码如下：

2.1.1 数学公式

假设起始点为 (x_1, y_1)，终点为 (x_2, y_2)，直线段的斜率为 $k = \frac{\Delta y}{\Delta x}$，其中：

$$\Delta x = x_2 - x_1, \quad \Delta y = y_2 - y_1$$

Bresenham 算法通过计算误差项 e 来决定当前像素的位置更新：

$$e = 2\Delta y - \Delta x$$

根据误差更新规则：- 若 $e > 0$，则表示需要调整纵坐标 y：

$$e = e - 2\Delta x$$

- 否则，只调整横坐标 x：

$$e = e + 2\Delta y$$

2.1.2 算法描述

Algorithm 1 Bresenham 绘制直线段算法

Require: 起始点 (x_1, y_1)，终点 (x_2, y_2)

Ensure: 绘制从 (x_1, y_1) 到 (x_2, y_2) 的直线段

计算 $\Delta x = |x_2 - x_1|$, $\Delta y = |y_2 - y_1|$
初始化误差项 $e = 2\Delta y - \Delta x$
设置步长 $y_{\text{step}} = 1$ 若 $y_2 > y_1$，否则 $y_{\text{step}} = -1$
初始化 $y = y_1$
for $x = x_1$ to x_2 **do**
    绘制点 (x, y)
    if $e > 0$ **then**
        $y \leftarrow y + y_{\text{step}}$
        $e \leftarrow e - 2\Delta x$
    end if
    $e \leftarrow e + 2\Delta y$
end for

2.2 绘制椭圆弧

椭圆弧的绘制基于参数方程，其核心思想是使用角度增量逐点计算椭圆弧上的像素点。特别地，当椭圆的长轴和短轴相等时，椭圆弧就退化为圆弧。

2.2.1 数学公式

椭圆弧的参数方程为：

$$(x, y) = (cx + a\cos\theta, cy + b\sin\theta), \quad \theta \in [\text{start}, \text{end}]$$

其中：- (cx, cy) 为椭圆的中心坐标；- a 为椭圆的长轴半径；- b 为椭圆的短轴半径；- θ 为椭圆弧的角度。

当 $a = b$ 时，椭圆弧退化为圆弧，其参数方程变为：

$$(x, y) = (cx + r\cos\theta, cy + r\sin\theta), \quad \theta \in [\text{start}, \text{end}]$$

其中 $r = a = b$ 为圆的半径。

2.2.2 算法描述

Algorithm 2 椭圆弧绘制算法

Require: 椭圆中心 (cx, cy)，长轴 a，短轴 b，起始角度 start，终止角度 end

Ensure: 绘制从起始角度到终止角度的椭圆弧

设置步长 step 用于角度增量
for $\theta =$ start to end step step **do**
    $x \leftarrow cx + a\cos\theta$
    $y \leftarrow cy + b\sin\theta$
    绘制点 (x, y)
end for

2.3 多边形填充

多边形填充基于扫描线算法，其核心思想是逐行扫描像素并填充多边形内部区域。

2.3.1 数学公式

设多边形的顶点集合为 $(v_1, v_2, \ldots, v_n)$，对于扫描线 $y = k$，交点的 x-坐标可以通过多边形的边方程计算：

$$x = x_1 + \frac{(k - y_1)(x_2 - x_1)}{y_2 - y_1}, \quad y_1 \leq k < y_2$$

2.3.2 算法描述

Algorithm 3 扫描线多边形填充算法

Require: 多边形顶点集合 $(v_1, v_2, \ldots, v_n)$

Ensure: 填充多边形内部

计算多边形的最小 y-坐标 $y_{\min}$ 和最大 y-坐标 $y_{\max}$
for $y = y_{\min}$ to $y_{\max}$ **do**
    找到扫描线与多边形边的交点集合
    按 x-坐标对交点排序
    for 每对交点 $(x_{\text{left}}, x_{\text{right}})$ **do**
        填充从 x_{left} 到 x_{right} 之间的像素
    end for
end for

3 结果展示

图 1 展示了通过本图形库绘制的不同图形及其效果，包括以下几个部分：

- **红色直线段**：从点 $(50, 50)$ 到点 $(400, 50)$，在窗口的顶部水平排列。

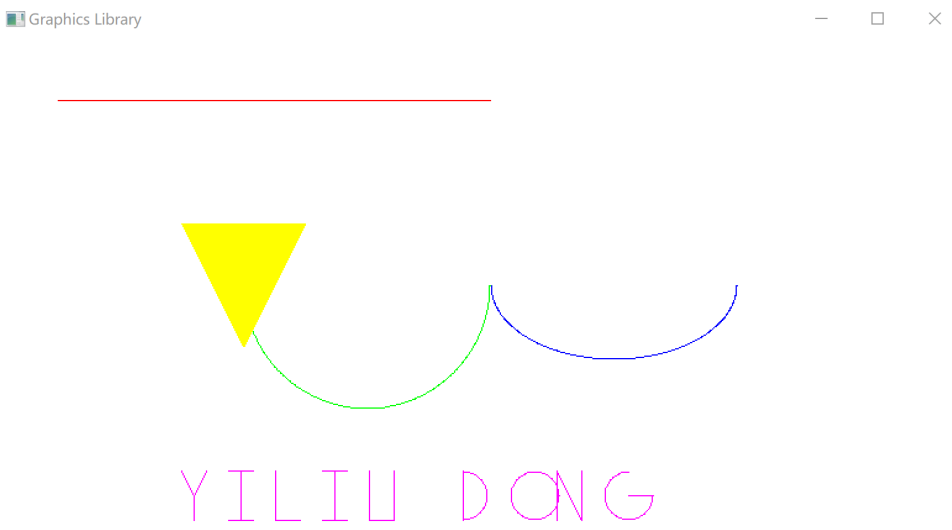

图 1: 通过图形库绘制的图形效果展示

- **绿色半圆弧**：圆心位于 $(300, 200)$，半径为 100，起始角度为 0 度，终止角度为 180 度，形成一个半圆。
- **蓝色椭圆弧**：圆心位于 $(500, 200)$，长轴为 100，短轴为 60，起始角度为 0 度，终止角度为 180 度，形成一个椭圆弧。
- **黄色三角形**：顶点分别位于 $(150, 150)$、$(250, 150)$ 和 $(200, 250)$，并进行了填充。
- **紫色文本**：在位置 $(150, 350)$ 绘制的 “YILIU DONG” 文本，展示了如何在图形界面中插入文本。

4 总结

通过本图形库，我们能够方便地绘制直线段、圆弧、椭圆弧和多边形填充，并实现文本显示功能。

A 附录

A.1 README.md

```
# GraphicsLibrary

## 项目概述
这是一个简单的图形函数库，有绘制直线段、圆弧、椭圆弧、填充多边形区域以及显示名字等功能。该项目采用 C++ 编写，并使用 CMake 构建工具。
```

```
5
6  ## 功能列表
7  - 绘制直线段
8  - 绘制圆弧
9  - 绘制椭圆弧
10 - 多边形区域填充
11 - 显示名字
12
13 ## 编译与运行
14
15 ## 依赖
16
17 - Windows 系统
18 - CMake
19 - MinGW 或其他支持 Windows 的 g++ 编译器
20
21 ## 步骤
22
23 1. 安装 CMake 和 MinGW。确保 cmake 和 g++ 已正确配置到系统路径。
24 2. 使用以下命令编译并运行项目:
25
26 ```shell
27 .\run.bat
28 ```
29
30 ## 项目结构
31
32 ```css
33 GraphicsLibrary/
34 |   README.md
35     CMakeLists.txt
36     run.bat
37     src/
38         main.cpp
39         graphics.cpp
40         graphics.h
41 ```
```

A.2 CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.10)
2
3  # 项目名称
4  project(GraphicsLibrary)
5
6  # 设置 C++ 标准
7  set(CMAKE_CXX_STANDARD 11)
8
```

```
# 设置编译器标志以支持 Unicode
if(MINGW)
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DUNICODE -D_UNICODE")
else()
    set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /DUNICODE /D_UNICODE")
endif()

# 设置源文件
set(SOURCE_FILES src/main.cpp src/graphics.cpp)

# 创建可执行文件
add_executable(GraphicsLibrary ${SOURCE_FILES})

# 链接 Windows API 库 (Gdi32 用于图形)
target_link_libraries(GraphicsLibrary Gdi32)

# 设置目标为 Windows GUI 程序
set_target_properties(GraphicsLibrary PROPERTIES
    WIN32_EXECUTABLE YES  # 这会告诉 CMake 该项目是一个 Windows GUI 程序
)
```

A.3 run.bat

```
@echo off
if exist build rd /s /q build
mkdir build
cd build
cmake -G "MinGW Makefiles" ..
mingw32-make
if exist GraphicsLibrary.exe (
    GraphicsLibrary.exe
)
```

A.4 main.cpp

```
#include <windows.h>
#include "graphics.h"

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
    switch (uMsg) {
        case WM_PAINT: {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hwnd, &ps);

            // 绘制红色的直线
            DrawLine(hdc, 50, 50, 400, 50, RGB(255, 0, 0));

```

```
            // 绘制绿色的半圆弧
            DrawArc(hdc, 300, 200, 100, 0, 180, RGB(0, 255, 0));

            // 绘制蓝色的椭圆弧
            DrawEllipseArc(hdc, 500, 200, 100, 60, 0, 180, RGB(0, 0, 255));

            // 绘制黄色的三角形
            POINT points[] = {{150, 150}, {250, 150}, {200, 250}};
            FillPolygon(hdc, points, 3, RGB(255, 255, 0));

            // 绘制紫色的名字
            DrawName(hdc, L"YILIU DONG", 150, 350, RGB(255, 0, 255));

            EndPaint(hwnd, &ps);
        } break;

        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        default:
            return DefWindowProc(hwnd, uMsg, wParam, lParam);
    }
    return 0;
}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
    int nCmdShow) {
    const wchar_t CLASS_NAME[] = L"GraphicsWindowClass";

    WNDCLASS wc = {};
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = CLASS_NAME;

    RegisterClass(&wc);

    HWND hwnd = CreateWindowEx(0, CLASS_NAME, L"Graphics Library",
        WS_OVERLAPPEDWINDOW,
                               CW_USEDEFAULT, CW_USEDEFAULT, 800, 600,
                               NULL, NULL, hInstance, NULL);

    if (hwnd == NULL) {
        return 0;
    }

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

```

```
    MSG msg = {};
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```

A.5 graphics.cpp

```
#include "graphics.h"
#include <cmath>
#include <vector>
#include <algorithm>
#include <wchar.h>

// 绘制直线的函数
void DrawLine(HDC hdc, int x1, int y1, int x2, int y2, COLORREF color) {
    int dx = abs(x2 - x1), dy = abs(y2 - y1);
    int sx = (x1 < x2) ? 1 : -1, sy = (y1 < y2) ? 1 : -1;
    int err = dx - dy;

    while (true) {
        SetPixel(hdc, x1, y1, color);
        if (x1 == x2 && y1 == y2) break;
        int e2 = err * 2;
        if (e2 > -dy) { err -= dy; x1 += sx; }
        if (e2 < dx) { err += dx; y1 += sy; }
    }
}

// 绘制圆弧的函数
void DrawArc(HDC hdc, int cx, int cy, int radius, int startAngle, int endAngle,
    COLORREF color) {
    float startRad = startAngle * 3.14159f / 180;
    float endRad = endAngle * 3.14159f / 180;
    float step = 1.0f / radius; // 根据半径动态调整步长

    for (float angle = startRad; angle <= endRad; angle += step) {
        int x = cx + static_cast<int>(radius * cos(angle));
        int y = cy + static_cast<int>(radius * sin(angle));
        SetPixel(hdc, x, y, color);
    }
}

// 绘制椭圆弧的函数
void DrawEllipseArc(HDC hdc, int cx, int cy, int a, int b, int startAngle, int
```

```
    endAngle, COLORREF color) {
    float startRad = startAngle * 3.14159f / 180;
    float endRad = endAngle * 3.14159f / 180;
    float step = 1.0f / std::max(a, b);

    for (float angle = startRad; angle <= endRad; angle += step) {
        int x = cx + static_cast<int>(a * cos(angle));
        int y = cy + static_cast<int>(b * sin(angle));
        SetPixel(hdc, x, y, color);
    }
}

// 填充多边形的函数
void FillPolygon(HDC hdc, POINT *points, int n, COLORREF color) {
    LONG yMin = points[0].y, yMax = points[0].y;
    for (int i = 1; i < n; i++) {
        yMin = std::min(yMin, points[i].y);
        yMax = std::max(yMax, points[i].y);
    }

    for (LONG y = yMin; y <= yMax; y++) {
        std::vector<int> intersections;
        for (int i = 0; i < n; i++) {
            int j = (i + 1) % n;
            if ((points[i].y <= y && points[j].y > y) || (points[i].y > y && points[
                j].y <= y)) {
                int x = points[i].x + (y - points[i].y) * (points[j].x - points[i].x
                    ) / (points[j].y - points[i].y);
                intersections.push_back(x);
            }
        }

        std::sort(intersections.begin(), intersections.end());
        for (size_t i = 0; i + 1 < intersections.size(); i += 2) { // 防止越界
            for (int x = intersections[i]; x <= intersections[i + 1]; x++) {
                SetPixel(hdc, x, y, color);
            }
        }
    }
}

// 绘制单个字母 'Y' 的函数
void DrawLetterY(HDC hdc, int x, int y, COLORREF color) {
    DrawLine(hdc, x, y, x + 10, y + 20, color);    // 左上到中间
    DrawLine(hdc, x + 20, y, x + 10, y + 20, color); // 右上到中间
    DrawLine(hdc, x + 10, y + 20, x + 10, y + 40, color);  // 中间到下
}

// 绘制字母 'I' 的函数
```

```
83  void DrawLetterI(HDC hdc, int x, int y, COLORREF color) {
84      DrawLine(hdc, x + 10, y, x + 10, y + 40, color);  // 垂直线
85      DrawLine(hdc, x, y, x + 20, y, color);            // 顶部横线
86      DrawLine(hdc, x, y + 40, x + 20, y + 40, color);  // 底部横线
87  }
88
89  // 绘制字母 'L' 的函数
90  void DrawLetterL(HDC hdc, int x, int y, COLORREF color) {
91      DrawLine(hdc, x, y, x, y + 40, color);            // 垂直线
92      DrawLine(hdc, x, y + 40, x + 20, y + 40, color); // 底部横线
93  }
94
95  // 绘制字母 'U' 的函数
96  void DrawLetterU(HDC hdc, int x, int y, COLORREF color) {
97      DrawLine(hdc, x, y, x, y + 40, color);            // 左垂直线
98      DrawLine(hdc, x + 20, y, x + 20, y + 40, color); // 右垂直线
99      DrawLine(hdc, x, y + 40, x + 20, y + 40, color); // 底部横线
100 }
101
102 // 绘制字母 'D' 的函数
103 void DrawLetterD(HDC hdc, int x, int y, COLORREF color) {
104     DrawLine(hdc, x, y, x, y + 40, color);            // 垂直线
105     DrawArc(hdc, x, y + 20, 20, 270, 450, color); // 半圆弧（右边）
106 }
107
108 // 绘制字母 'O' 的函数
109 void DrawLetterO(HDC hdc, int x, int y, COLORREF color) {
110     DrawArc(hdc, x + 20, y + 20, 20, 0, 360, color); // 完整圆形
111 }
112
113 // 绘制字母 'N' 的函数
114 void DrawLetterN(HDC hdc, int x, int y, COLORREF color) {
115     DrawLine(hdc, x, y, x, y + 40, color);           // 左垂直线
116     DrawLine(hdc, x + 20, y, x + 20, y + 40, color); // 右垂直线
117     DrawLine(hdc, x, y, x + 20, y + 40, color);      // 斜线
118 }
119
120 // 绘制字母 'G' 的函数
121 void DrawLetterG(HDC hdc, int x, int y, COLORREF color) {
122     DrawArc(hdc, x + 20, y + 20, 20, 0, 270, color); // 弧形（3/4圆）
123     DrawLine(hdc, x + 20, y + 20, x + 40, y + 20, color); // 底部横线
124 }
125
126 // 绘制名字的函数
127 void DrawName(HDC hdc, const wchar_t *name, int x, int y, COLORREF color) {
128     for (int i = 0; i < wcslen(name); ++i) {
129         wchar_t letter = name[i];
130         switch (letter) {
131             case L'Y': DrawLetterY(hdc, x, y, color); break;
```

```
            case L'I': DrawLetterI(hdc, x, y, color); break;
            case L'L': DrawLetterL(hdc, x, y, color); break;
            case L'U': DrawLetterU(hdc, x, y, color); break;
            case L'D': DrawLetterD(hdc, x, y, color); break;
            case L'O': DrawLetterO(hdc, x, y, color); break;
            case L'N': DrawLetterN(hdc, x, y, color); break;
            case L'G': DrawLetterG(hdc, x, y, color); break;
            default:
                // 如果字母不在case中，直接绘制字母
                SetTextColor(hdc, color);
                SetBkMode(hdc, TRANSPARENT); // 透明背景
                TextOutW(hdc, x, y, &letter, 1);
                break;
        }
        x += 38; // 设置字母之间的间距
    }
}
```

A.6 graphics.h

```
#ifndef GRAPHICS_H
#define GRAPHICS_H

#include <windows.h>

// 绘制直线
void DrawLine(HDC hdc, int x1, int y1, int x2, int y2, COLORREF color);

// 绘制圆弧
void DrawArc(HDC hdc, int cx, int cy, int radius, int startAngle, int endAngle,
    COLORREF color);

// 绘制椭圆弧
void DrawEllipseArc(HDC hdc, int cx, int cy, int a, int b, int startAngle, int
    endAngle, COLORREF color);

// 填充多边形
void FillPolygon(HDC hdc, POINT *points, int n, COLORREF color);

// 绘制名字
void DrawName(HDC hdc, const wchar_t *name, int x, int y, COLORREF color);

#endif // GRAPHICS_H
```