Министерство науки и высшего образования

Российской Федерации

НГТУ

Лабораторная работа № 5

Закрашивание фигур

|  |  |
| --- | --- |
| Выполнил:  Жуков Иван  Группа: АТ-34 | Проверил:  Смагин С. М. |

Новосибирск

2024

Задание

1. Разработать программу, которая закрашивает произвольную фигуру (окружность с внутренними треугольниками и/или окружностями) по алгоритму заполнения с затравкой.

**Описание:**  
Flood Fill — это алгоритм, используемый для закрашивания области, ограниченной цветами. Он работает путем рекурсивного расширения от заданной точки (начальной координаты), пока не будет достигнут цвет границы (или другой критерий окончания).

### Алгоритм работы Flood Fill:

1. **Начальная проверка:**  
   Если начальная точка (x, y) находится за пределами сетки или цвет этой точки совпадает с целевым цветом, ничего не делаем.
2. **Изменение цвета текущей точки:**  
   Устанавливаем цвет начальной точки на новый цвет fillColor.
3. **Рекурсивный вызов для соседей:**  
   Проверяем все соседние точки (вверх, вниз, влево, вправо). Для каждой соседней точки выполняем:
   * Если она еще не окрашена в цвет fillColor и совпадает с базовым цветом baseColor, рекурсивно вызываем Flood Fill для этой точки.
4. **Отрисовка:**  
   После каждого шага перерисовываем экран с помощью метода render(), чтобы пользователь видел постепенное заполнение.
5. **Завершение:**  
   Алгоритм продолжается до тех пор, пока все возможные точки, соответствующие базовому цвету, не будут окрашены в новый цвет.

### Пример работы:

1. Пользователь указывает начальную точку и целевой цвет заливки.
2. Алгоритм начинает закрашивать с этой точки и постепенно переходит на соседние, пока все подходящие клетки не будут заполнены.

### Визуализация шагов:

1. Начальная точка закрашивается.
2. Закрашиваются все доступные соседние клетки.
3. Процесс продолжается до заполнения всей области.

Это можно представить как заполнение ведром краски области с одинаковым цветом на рисунке.

Скриншоты

Листинг программы

cat My\_grafcs.h

#include <SDL2/SDL.h>

class My\_graphics {

private:

int numSquares;

int squareSize;

SDL\_Window \*window;

SDL\_Renderer \*renderer;

SDL\_Color \*\* pixels;

void horisontal\_line(int x0, int x1, int y, SDL\_Color color);

inline void lineHelp(int x, int y, bool swap);

public:

My\_graphics(int numSquares, int squareSize);

~My\_graphics();

static SDL\_Color create\_color(int r, int g, int b, int a);

static bool IsSameColor(SDL\_Color c0, SDL\_Color c1);

void refresh\_screen();

void refresh\_screen(SDL\_Color color);

void render();

void Draw\_grid();

void Draw\_grid(SDL\_Color color);

void Draw\_point(int x, int y);

void Draw\_point(int x, int y, SDL\_Color color);

void Draw\_line\_digital\_differential\_analyzer(int x\_s, int y\_s, int x\_e,

int y\_e);

void Draw\_line\_digital\_differential\_analyzer(int x\_s, int y\_s, int x\_e,

int y\_e, SDL\_Color color);

void DrawlineBresenham(int x\_s, int y\_s, int x\_e, int y\_e);

void DrawlineBresenham(int x\_s, int y\_s, int x\_e, int y\_e, SDL\_Color color);

void DrawCircleBresenham(int centerX, int centerY, int radius);

void DrawCircleBresenham(int centerX, int centerY, int radius,

SDL\_Color color);

void DrawTriangle(int x0, int y0, int x1, int y1, int x2, int y2);

void DrawTriangle(int x0, int y0, int x1, int y1, int x2, int y2,

SDL\_Color color);

void DrawTriangleShape(int x0, int y0, int x1, int y1, int x2, int y2);

void DrawTriangleShape(int x0, int y0, int x1, int y1, int x2, int y2,

SDL\_Color color);

void FloodFill(int x, int y, SDL\_Color fillColor, SDL\_Color baseColor);

void DLB(int x\_s, int y\_s, int x\_e, int y\_e);

void DLB(int x\_s, int y\_s, int x\_e, int y\_e, SDL\_Color color);

};cat My\_grafcs.cpp

#include "My\_grafcs.h"

#include "iostream"

#include <SDL2/SDL.h>

#include <stack>

SDL\_Color My\_graphics ::create\_color(int r, int g, int b, int a) {

SDL\_Color color;

color.r = r;

color.g = g;

color.b = b;

color.a = a;

return color;

}

bool My\_graphics::IsSameColor(SDL\_Color c0, SDL\_Color c1) {

return c0.r == c1.r && c0.g == c1.g && c0.b == c1.b && c0.a == c1.a;

}

My\_graphics ::My\_graphics(int numSquares, int squareSize)

: numSquares(numSquares), squareSize(squareSize) {

if (SDL\_Init(SDL\_INIT\_VIDEO) != 0) {

throw "Ошибка инициализации SDL: ";

}

window = SDL\_CreateWindow("SDL Window", SDL\_WINDOWPOS\_CENTERED,

SDL\_WINDOWPOS\_CENTERED, numSquares \* squareSize + 1,

numSquares \* squareSize + 1, SDL\_WINDOW\_SHOWN);

if (!window) {

SDL\_Quit();

throw "Ошибка создания окна: ";

}

renderer = SDL\_CreateRenderer(window, -1, SDL\_RENDERER\_ACCELERATED);

if (!renderer) {

SDL\_DestroyWindow(window);

SDL\_Quit();

throw "Ошибка создания рендерера: ";

}

pixels = new SDL\_Color \*[numSquares];

for (int i = 0; i < numSquares; ++i) {

pixels[i] = new SDL\_Color[numSquares];

}

}

My\_graphics::~My\_graphics() {

// SDL\_FreeSurface(surface);

SDL\_DestroyRenderer(renderer);

SDL\_DestroyWindow(window);

SDL\_Quit();

for (int i = 0; i < numSquares; ++i) {

delete pixels[i];

}

delete pixels;

}

void My\_graphics ::Draw\_grid() { Draw\_grid(create\_color(255, 255, 255, 255)); }

void My\_graphics ::Draw\_grid(SDL\_Color color) {

SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

for (int i = 0; i <= numSquares; i++) {

SDL\_RenderDrawLine(renderer, i \* squareSize, 0, i \* squareSize,

numSquares \* squareSize);

SDL\_RenderDrawLine(renderer, 0, i \* squareSize, numSquares \* squareSize,

i \* squareSize);

}

}

void My\_graphics ::Draw\_point(int x, int y) {

Draw\_point(x, y, create\_color(0, 255, 0, 255));

}

void My\_graphics ::Draw\_point(int x, int y, SDL\_Color color) {

// SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

if (x >= 0 && y >= 0 && x < numSquares && y < numSquares) {

pixels[x][y] = color;

}

}

void My\_graphics ::Draw\_line\_digital\_differential\_analyzer(int x\_s, int y\_s,

int x\_e, int y\_e) {

Draw\_line\_digital\_differential\_analyzer(x\_s, y\_s, x\_e, y\_e,

create\_color(0, 255, 0, 255));

}

void My\_graphics ::Draw\_line\_digital\_differential\_analyzer(int x\_s, int y\_s,

int x\_e, int y\_e,

SDL\_Color color) {

SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

int dx = abs(x\_e - x\_s);

int dy = abs(y\_e - y\_s);

int steps = std::max(dx, dy);

float x\_inc = (float)dx / (float)steps;

float y\_inc = (float)dy / (float)steps;

float x = (float)x\_s;

float y = (float)y\_s;

for (int i = 0; i <= steps; i++) {

Draw\_point(static\_cast<int>(x), static\_cast<int>(y), color);

x += x\_inc;

y += y\_inc;

}

}

void My\_graphics::DrawlineBresenham(int x\_s, int y\_s, int x\_e, int y\_e) {

DrawlineBresenham(x\_s, y\_s, x\_e, y\_e, create\_color(0, 255, 0, 255));

}

void My\_graphics::DrawlineBresenham(int x\_s, int y\_s, int x\_e, int y\_e,

SDL\_Color color) {

SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

int dx = abs(x\_e - x\_s);

int dy = abs(y\_e - y\_s);

int dir\_x = (x\_s < x\_e) ? 1 : -1;

int dir\_y = (y\_s < y\_e) ? 1 : -1;

int err = dx - dy;

int x = x\_s;

int y = y\_s;

while (true) {

Draw\_point(x, y, color);

if (x == x\_e && y == y\_e) {

break;

}

int e2 = 2 \* err;

if (e2 < dx) {

err += dx;

y += dir\_y;

Draw\_point(x, y, color);

}

if (e2 > -dy) {

err -= dy;

x += dir\_x;

}

}

}

void My\_graphics::DrawCircleBresenham(int centerX, int centerY, int radius) {

DrawCircleBresenham(centerX, centerY, radius, create\_color(0, 255, 0, 255));

}

void My\_graphics::DrawCircleBresenham(int centerX, int centerY, int radius,

SDL\_Color color) {

int x = 0;

int y = radius;

int d = 3 - 2 \* radius;

while (y >= x) {

Draw\_point(centerX + x, centerY + y, color);

Draw\_point(centerX - x, centerY + y, color);

Draw\_point(centerX + x, centerY - y, color);

Draw\_point(centerX - x, centerY - y, color);

Draw\_point(centerX + y, centerY + x, color);

Draw\_point(centerX - y, centerY + x, color);

Draw\_point(centerX + y, centerY - x, color);

Draw\_point(centerX - y, centerY - x, color);

if (d <= 0) {

d = d + 4 \* x + 6;

} else {

d = d + 4 \* (x - y) + 10;

y--;

}

x++;

}

}

void My\_graphics::DrawTriangleShape(int x0, int y0, int x1, int y1, int x2, int y2) {

DrawTriangleShape(x0, y0, x1, y1, x2, y2, create\_color(255, 255, 255, 100));

}

void My\_graphics::DrawTriangleShape(int x0, int y0, int x1, int y1, int x2, int y2,

SDL\_Color color) {

DrawlineBresenham(x0, y0, x1, y1, create\_color(0, 0, 255, 255));

DrawlineBresenham(x1, y1, x2, y2, create\_color(0, 255, 0, 255));

DrawlineBresenham(x2, y2, x0, y0, create\_color(255, 0, 0, 255));

}

void My\_graphics::DrawTriangle(int x0, int y0, int x1, int y1, int x2, int y2) {

DrawTriangle(x0, y0, x1, y1, x2, y2, create\_color(255, 255, 255, 100));

}

void My\_graphics::DrawTriangle(int x0, int y0, int x1, int y1, int x2, int y2,

SDL\_Color color) {

DrawlineBresenham(x0, y0, x1, y1, create\_color(0, 0, 255, 255));

DrawlineBresenham(x1, y1, x2, y2, create\_color(0, 255, 0, 255));

DrawlineBresenham(x2, y2, x0, y0, create\_color(255, 0, 0, 255));

int y\_min = std::min(y0,std::min(y1,y2));

int y\_max = std::max(y0,std::max(y1,y2));

if (y1 == y\_min) {

std::swap(x0, x1);

std::swap(y0, y1);

} else if (y2 == y\_min) {

std::swap(x0, x2);

std::swap(y0, y2);

}

if (y0 == y\_max) {

std::swap(x2, x0);

std::swap(y2, y0);

} else if (y1 == y\_max) {

std::swap(x2, x1);

std::swap(y2, y1);

}

int line\_x0;

int line\_x1;

for (int i = y0; i < y1; ++i) {

line\_x0 = floor(x0 + (i - y0) \* (x2 - x0) / (y2 - y0));

line\_x1 = floor(x0 + (i - y0) \* (x1 - x0) / (y1 - y0));

horisontal\_line(line\_x0, line\_x1, i, color);

}

for (int i = y1; i < y2; ++i) {

line\_x0 = floor(x2 + (i - y2) \* (x1 - x2) / (y1 - y2));

line\_x1 = floor(x2 + (i - y2) \* (x0 - x2) / (y0 - y2));

horisontal\_line(line\_x0, line\_x1, i, color);

}

}

void My\_graphics::horisontal\_line(int x0, int x1, int y, SDL\_Color color) {

for (int i = std::min(x0, x1); i <= std::max(x0, x1); ++i) {

Draw\_point(i, y, color);

}

}

void My\_graphics::refresh\_screen() {

refresh\_screen(create\_color(0, 0, 0, 255));

}

void My\_graphics::refresh\_screen(SDL\_Color color) {

for (int x = 0; x < numSquares; ++x) {

for (int y = 0; y < numSquares; ++y) {

pixels[x][y] = color;

}

}

SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

SDL\_RenderClear(renderer);

SDL\_RenderPresent(renderer);

}

void My\_graphics::render() {

for (int x = 0; x < numSquares; ++x) {

for (int y = 0; y < numSquares; ++y) {

SDL\_SetRenderDrawColor(renderer, pixels[x][y].r, pixels[x][y].g,

pixels[x][y].b, pixels[x][y].a);

SDL\_Rect rect = {(x-1) \* squareSize,

numSquares \* squareSize - y \* squareSize,

squareSize, squareSize};

SDL\_RenderFillRect(renderer, &rect);

}

}

Draw\_grid();

SDL\_RenderPresent(renderer);

}

void My\_graphics::DLB(int x1, int y1, int x2, int y2) {

DLB(x1, y1, x2, y2, create\_color(255, 0, 0, 255));

}

void My\_graphics::DLB(int x1, int y1, int x2, int y2, SDL\_Color color) {

SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);

if (x1 == x2 && y1 == y2) {

Draw\_point(x1, y1);

return;

}

int dx = abs(x2 - x1);

int dy = abs(y2 - y1);

bool swap = false;

if (dy > dx) {

swap = true;

std::swap(x1, y1);

std::swap(x2, y2);

std::swap(dx, dy);

}

int dir\_x = x2 >= x1 ? 1 : -1;

int dir\_y = y2 >= y1 ? 1 : -1;

float t = (float)numSquares \* (float)dy / (float)dx;

float w = (float)numSquares - t;

float d = (float)numSquares / (float)2;

int i = dx + 1;

if (!t) {

while (i--) {

lineHelp(x1, y1, swap);

x1 += dir\_x;

}

return;

}

lineHelp(x1, y1, swap);

while (--i) {

if (d >= w) {

d -= w;

y1 += dir\_y;

lineHelp(x1, y1, swap);

x1 += dir\_x;

} else {

d += t;

x1 += dir\_x;

}

lineHelp(x1, y1, swap);

}

}

inline void My\_graphics::lineHelp(int x, int y, bool swap) {

if (swap)

Draw\_point(y, x);

else

Draw\_point(x, y);

}

void My\_graphics::FloodFill(int x, int y, SDL\_Color fillColor,

SDL\_Color baseColor) {

if (x < 0 || y < 0 || x >= numSquares || y >= numSquares) {

return;

}

SDL\_Color p = pixels[x][y];

if (IsSameColor( pixels[x+1][y], baseColor)){

Draw\_point(x+1,y, fillColor);

render();

SDL\_Delay(100);

FloodFill(x+1,y,fillColor,baseColor);

}

if (IsSameColor( pixels[x-1][y], baseColor)){

Draw\_point(x-1,y, fillColor);

render();

SDL\_Delay(100);

FloodFill(x-1,y,fillColor,baseColor);

}

if (IsSameColor( pixels[x][y+1], baseColor)){

Draw\_point(x,y+1, fillColor);

render();

SDL\_Delay(100);

FloodFill(x,y+1,fillColor,baseColor);

}

if (IsSameColor( pixels[x][y-1], baseColor)){

Draw\_point(x,y-1, fillColor);

render();

SDL\_Delay(100);

FloodFill(x,y-1,fillColor,baseColor);

}

}

cat main.cpp

#include "My\_grafcs.h"

#include <iostream>

int main() {

int numSquares; // Количество квадратов в сетке

int squareSize; // Размер одного квадрата

std::cout << "Enter number of squares in grid: ";

std::cin >> numSquares;

std::cout << "Enter the size of one square: ";

std::cin >> squareSize;

std::cout << "1-dot 2-line 3-circle 4-triangle: ";

int chouse;

std::cin >> chouse;

My\_graphics window(numSquares, squareSize);

window.refresh\_screen();

window.Draw\_grid();

switch (chouse) {

case 1: {

int pointX, pointY; // Координаты точки

std::cout << "Enter the coordinates of the point (X and Y separated by a "

"space): ";

std::cin >> pointX >> pointY;

window.Draw\_point(pointX, pointY);

break;

}

case 2: {

std::cout << "1 - Draw\_line\_digital\_differential\_analyzer\n2 - "

"DrawlineBresenham: ";

std::cin >> chouse;

int l\_x\_s, l\_x\_e, l\_y\_s, l\_y\_e;

std::cout << "Enter coordinate line like l\_x\_s, l\_y\_s, l\_x\_e, l\_y\_e: ";

std::cin >> l\_x\_s >> l\_y\_s >> l\_x\_e >> l\_y\_e;

switch (chouse) {

case 1: {

window.Draw\_line\_digital\_differential\_analyzer(l\_x\_s, l\_y\_s, l\_x\_e,

l\_y\_e);

break;

}

case 2: {

window.DrawlineBresenham(l\_x\_s, l\_y\_s, l\_x\_e, l\_y\_e);

break;

}

}

break;

}

case 3: {

std::cout << "Enter center circle x, y, and radius: ";

int centerX, centerY, radius;

std::cin >> centerX >> centerY >> radius;

window.DrawCircleBresenham(centerX, centerY, radius);

break;

}

case 4: {

std::cout << "Enter coordinate triangle like this:\nx0 y0 x1 y1 x2 y2:\n";

int x0, y0, x1, y1, x2, y2;

std::cin >> x0 >> y0 >> x1 >> y1 >> x2 >> y2;

window.DrawTriangle(x0, y0, x1, y1, x2, y2);

break;

}

case 5: {

window.DrawCircleBresenham(10,10,10, window.create\_color(255,0,0,255));

window.DrawTriangle(5, 5, 5, 13,15,13, window.create\_color(0,0,0,255));

break;

}

case 6: {

// int startX, startY;

// std::cout << "Enter starting point (x, y): ";

// std::cin >> startX >> startY;

//

// int r, g, b, a;

// std::cout << "Enter fill color (R G B A): ";

// std::cin >> r >> g >> b >> a;

// SDL\_Color fillColor = window.create\_color(r, g, b, a);

//

// std::cout << "Enter boundary color (R G B A): ";

// std::cin >> r >> g >> b >> a;

// SDL\_Color boundaryColor = window.create\_color(r, g, b, a);

window.refresh\_screen(window.create\_color(0, 0, 0, 255));

window.DrawCircleBresenham(10, 10, 5);

window.DrawCircleBresenham(13, 10, 5);

window.render();

SDL\_Delay(3000);

std::cout << "a\n";

window.FloodFill(10, 10, window.create\_color(255, 0, 0, 0),

window.create\_color(0, 0, 0, 255));

break;

}

case 7: {

window.DrawCircleBresenham(20, 20, 15);

window.DrawTriangleShape(5, 25, 35, 25, 20, 40);

window.FloodFill(20,20, window.create\_color(255,2,25,255), window.create\_color(0,0,0,255));

}

}

window.render();

int a;

std::cin >> a;

return 0;

}