Министерство науки и высшего образования

Российской Федерации

НГТУ

Лабораторная работа № 4

РИСОВАНИЕ ТРЕУГОЛЬНИКА.

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

Новосибирск

2024

Задание

1. Разработать программу, которая будет рисовать треугольник на сетке с заданными параметрами (как в Лабораторной работе №1).

## Описание функций

### void DrawTriangle(int x1, int y1, int x2, int y2, int x3, int y3, SDL\_Color color)

Рисует закрашенный треугольник методом горизонтальных линий.

* **Параметры:**
  + x1, y1, x2, y2, x3, y3 — координаты трёх вершин треугольника.
  + color — цвет треугольника.
* **Алгоритм:**
  + Сортирует вершины по координате Y.
  + Использует интерполяцию для вычисления границ треугольника на каждом горизонтальном уровне.
  + Для каждой строки рисует горизонтальный отрезок между границами.

1. **Взаимодействие с пользователем:**
   * Предлагается выбрать тип графического объекта: точка, линия или окружность.
   * Пользователь вводит параметры объекта (координаты, радиус, алгоритм).
2. **Рисование:**
   * В зависимости от выбора пользователя, вызывается одна из функций:
     + Draw\_point — рисует точку.
     + Draw\_line\_digital\_differential\_analyzer или DrawlineBresenham — рисуют линии.
     + DrawCircleBresenham — рисует окружность.
     + Для треугольников: вызывается DrawTriangle.
   * В процессе работы обновляется сетка и рендерятся графические элементы.
3. **Вывод результата:**
   * Используется render, чтобы отобразить изменения на экране.
   * Результат остаётся на экране до завершения программы.

Листинг программы

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;  
 }  
 window.render();  
 int a;  
 std::cin >> a;  
  
 return 0;  
}

My\_grafcs.h:

#include <SDL2/SDL.h>  
  
class My\_graphics {  
private:  
 int numSquares{};  
 int squareSize{};  
 SDL\_Window \*window;  
 SDL\_Renderer \*renderer;  
  
 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();  
  
 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 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);  
  
 SDL\_Color create\_color(int r, int g, int b, int a);  
};

My\_grafcs.cpp:

#include "My\_grafcs.h"  
#include "iostream"  
#include <SDL2/SDL.h>  
  
  
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;  
}  
  
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: ";  
 }  
}  
  
My\_graphics::~My\_graphics() {  
 SDL\_DestroyRenderer(renderer);  
 SDL\_DestroyWindow(window);  
 SDL\_Quit();  
}  
  
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);  
 x--;  
 y--;  
 SDL\_Rect rect = {x \* squareSize,  
 numSquares \* squareSize - (y + 1) \* squareSize, squareSize,  
 squareSize};  
 SDL\_RenderFillRect(renderer, &rect);  
  
}  
  
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::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) {  
  
 int y\_min = std::min(y0,std::min(y1,y2));  
 int y\_max = std::max(y0,std::max(y1,y2));  
 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));  
  
  
 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((float)x0 + ((float)i - (float)y0) \* ((float)x2 - (float)x0) / ((float)y2 - (float)y0));  
 line\_x1 = floor((float)x0 + ((float)i - (float)y0) \* ((float)x1 - (float)x0) / ((float)y1 - (float)y0));  
 horisontal\_line(line\_x0, line\_x1, i, color);  
 render();  
 SDL\_Delay(100);  
 }  
 for (int i = y1; i < y2; ++i) {  
 line\_x0 = floor((float)x2 + ((float)i - (float)y2) \* ((float)x1 - (float)x2) / ((float)y1 - (float)y2));  
 line\_x1 = floor((float)x2 + ((float)i - (float)y2) \* ((float)x0 - (float)x2) / ((float)y0 - (float)y2));  
 horisontal\_line(line\_x0, line\_x1, i, color);  
 render();  
 SDL\_Delay(100);  
 }  
  
  
}  
  
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) {  
 SDL\_SetRenderDrawColor(renderer, color.r, color.g, color.b, color.a);  
 SDL\_RenderClear(renderer);  
}  
  
void My\_graphics::render() { 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);  
}

Скриншоты:

