**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №2**

по дисциплине: Компьютерная графика

тема: «Растровая заливка геометрических фигур»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

ст. пр. Осипов Олег Васильевич

Белгород 2024 г.

**Лабораторная работа №2  
Растровая заливка геометрических фигур  
Вариант 8**

**Цель работы:** изучение алгоритмов растровой заливки основных

геометрических фигур: кругов, многоугольников

**Задания для выполнения к работе:**

1. Изучить растровые алгоритмы заливки геометрических фигур.
2. Разработать алгоритм и составить программу для построения на экране изображения в соответствии с номером варианта (по журналу старосты). В качестве исходных данных взять указанные в таблице №1 лаб. работы №1.

**Задание:**

Реализовать вращение 4-конечной звезды против часовой стрелки.

Пусть – ширина экрана, – высота экрана. Диаметр описанной вокруг равностороннего треугольника окружности примем равным . Диаметр вписанной в треугольник окружности будет в два раза меньше описанного r = R / 2. Сторона треугольника будет равна . Введём также центр экрана . Вектор определяющие точки треугольника будут равны:

С окружностями с радиусом R и r и центром C получим равносторонний треугольник с описанной и вписанной окружностью.

Высота и ширина четырёхконечной звезды будет равна радиусу вписанной окружности, однако пока что обозначим её центр в точке (0, 0). Также введём некоторый отступ от центра для частей, где соединяются выступы звезды. Он будет равен . Тогда звезда задаётся координатами

Для выполнения трансформаций над звездой будем использовать SRT-матрицу. Для преобразования координат нужно переменожить каждую из них на SRT матрицу:

Где соответственно отвечают за размер, вращение и перемещение вектора. Для двухмерного пространства достаточно будет использовать трёхмерные матрицы. Матрица составляется следующим образом:

В нашем случае размер не изменяется (, а смещение относительно центра = С (. Для рисования варианта задания представлен текст программы на C++:

**Frame.h**

template <class InterpolatorClass>

void Triangle(float x0, float y0, float x1, float y1, float x2, float y2, InterpolatorClass& Interpolator)

{

    // Отсортируем точки таким образом, чтобы выполнилось условие: y0 < y1 < y2

    if (y1 < y0)

    {

        swap(y1, y0);

        swap(x1, x0);

    }

    if (y2 < y1)

    {

        swap(y2, y1);

        swap(x2, x1);

    }

    if (y1 < y0)

    {

        swap(y1, y0);

        swap(x1, x0);

    }

    // Определяем номера строк пикселей, в которых располагаются точки треугольника

    int Y0 = (int)(y0 + 0.5f);

    int Y1 = (int)(y1 + 0.5f);

    int Y2 = (int)(y2 + 0.5f);

    // Отсечение невидимой части треугольника

    if (Y0 < 0) Y0 = 0;

    else if (Y0 >= height) Y0 = height;

    if (Y1 < 0) Y1 = 0;

    else if (Y1 >= height) Y1 = height;

    if (Y2 < 0) Y2 = 0;

    else if (Y2 >= height) Y2 = height;

    double rawX0 = (Y0 + 0.5f - y0) / (y1 - y0) \* (x1 - x0) + x0;

    double rawX1 = (Y0 + 0.5f - y0) / (y2 - y0) \* (x2 - x0) + x0;

    bool should\_swap = rawX0 > rawX1;

    int X0, X1;

    // Рисование верхней части треугольника

    for (float y = Y0 + 0.5f; y < Y1; y++)

    {

        X0 = rawX0;

        X1 = rawX1;

        if (should\_swap) swap(X0, X1);

        if (X0 < 0) X0 = 0;

        if (X1 > width) X1 = width;

        for (int x = X0; x <= X1; x++)

        {

            // f(x + 0.5, y)

            COLOR color = Interpolator.color(x + 0.5f, y);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[(int)y][x]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x, y, color);

        }

        rawX0 += (x1 - x0) / (y1 - y0);

        rawX1 += (x2 - x0) / (y2 - y0);

    }

    rawX0 = (Y1 + 0.5f - y1) / (y2 - y1) \* (x2 - x1) + x1;

    rawX1 = (Y1 + 0.5f - y0) / (y2 - y0) \* (x2 - x0) + x0;

    should\_swap = rawX0 > rawX1;

    // Рисование нижней части треугольника

    for (float y = Y1 + 0.5f; y < Y2; y++)

    {

        X0 = rawX0;

        X1 = rawX1;

        if (should\_swap) swap(X0, X1);

        if (X0 < 0) X0 = 0;

        if (X1 > width) X1 = width;

        for (int x = X0; x <= X1; x++)

        {

            // f(x + 0.5, y)

            COLOR color = Interpolator.color(x + 0.5f, y);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[(int)y][x]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x, y, color);

        }

        rawX0 += (x2 - x1) / (y2 - y1);

        rawX1 += (x2 - x0) / (y2 - y0);

    }

}

bool IsPointInCircle(int x0, int y0, int radius, int point\_x, int point\_y)

{

    return (x0 - point\_x) \* (x0 - point\_x) + (y0 - point\_y) \* (y0 - point\_y) < radius \* radius;

}

bool IsPointInTriangle(float x0, float y0, float x1, float y1, float x2, float y2, float point\_x, float point\_y)

{

    float S = (y1 - y2) \* (x0 - x2) + (x2 - x1) \* (y0 - y2);

    float h0 = ((y1 - y2) \* (point\_x - x2) + (x2 - x1) \* (point\_y - y2)) / S;

    float h1 = ((y2 - y0) \* (point\_x - x2) + (x0 - x2) \* (point\_y - y2)) / S;

    float h2 = 1 - h0 - h1;

    return h0 >= 0 && h1 >= 0 && h2 >= 0;

}

template <class InterpolatorClass>

void Circle(int x0, int y0, int radius, InterpolatorClass& Interpolator)

{

    int x = 0, y = radius;

    int DSUM = 2 \* x \* x + 2 \* y \* y - 2 \* radius \* radius - 2 \* y + 1;

    while (x < y)

    {

        // Если ближе точка (x, y - 1), то смещаемся к ней

        if (DSUM > 0) {

            DSUM -= 4 \* y - 4;

            y--;

        }

        // Перенос и отражение вычисленных координат на все октанты окружности

        for (int X0 = -x; X0 <= x; X0++) {

            COLOR color = Interpolator.color(x0 + X0 + 0.5f, y0 + y);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[y0 + y][x0 + X0]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x0 + X0, y0 + y, color);

            color = Interpolator.color(x0 + X0 + 0.5f, y0 - y);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[y0 - y][x0 + X0]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x0 + X0, y0 - y, color);

        }

        for (int X0 = -y; X0 <= y; X0++) {

            COLOR color = Interpolator.color(x0 + X0 + 0.5f, y0 + x);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[y0 + x][x0 + X0]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x0 + X0, y0 + x, color);

            color = Interpolator.color(x0 + X0 + 0.5f, y0 - x);

            // Для рисования полупрозрачных фигур будем использовать альфа-смешивание

            if (color.ALPHA < 255)

            {

                COLOR written = matrix[y0 - x][x0 + X0]; // Уже записанное в буфере кадра значение цвета, т.е. цвет фона

                float a = color.ALPHA / 255.0f, b = 1 - a;

                color.RED = color.RED \* a + written.RED \* b;

                color.GREEN = color.GREEN \* a + written.GREEN \* b;

                color.BLUE = color.BLUE \* a + written.BLUE \* b;

            }

            SetPixel(x0 + X0, y0 - x, color);

        }

        x++;

        DSUM -= -4 \* x - 2;

    }

}

**Painter.h**

double t = (3 \* a) / sqrt(3);

coordinate triangleA = { C.x, C.y - a };

coordinate triangleB = { C.x - t / 2, C.y + a / 2 };

coordinate triangleC = { C.x + t / 2, C.y + a / 2 };

Matrix S = { 1, 0, 0,

                0, 1, 0,

                0, 0, 1 };

Matrix R = { cos(angle), -sin(angle),  0,

                sin(angle),  cos(angle),  0,

                        0,           0,  1 };

Matrix T = { 1, 0, W / 2.0,

                0, 1, H / 2.0,

                0, 0,       1 };

Matrix SRT = (T.multiply(R)).multiply(S);

double starOffset = a / 12;

coordinate star[8] = {

    { 0, a / 2 },

    { starOffset, starOffset },

    { a / 2, 0 },

    { starOffset, -starOffset },

    { 0, -a / 2 },

    { -starOffset, -starOffset },

    { -a / 2, 0 },

    { -starOffset, starOffset } };

for (int i = 0; i < 8; i++)

{

    Vector pointVector = { star[i].x, star[i].y, 1 };

    pointVector = SRT.multiply(pointVector);

    star[i].x = pointVector.vector[0];

    star[i].y = pointVector.vector[1];

}

bool starSelected = frame.IsPointInTriangle(

    star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y,

    global\_clicked\_pixel.X, global\_clicked\_pixel.Y) ||

    frame.IsPointInTriangle(

        star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y) ||

    frame.IsPointInTriangle(

        star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y) ||

    frame.IsPointInTriangle(

        star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y) ||

    frame.IsPointInTriangle(

        star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y) ||

    frame.IsPointInTriangle(

        star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y);

bool smallCircleSelected = !starSelected &&

    frame.IsPointInCircle((int)C.x, (int)C.y, (int)(a \* 0.5),

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y);

bool triangleSelected = !smallCircleSelected && !starSelected && frame.IsPointInTriangle(

    triangleA.x, triangleA.y,

    triangleB.x, triangleB.y,

    triangleC.x, triangleC.y,

    global\_clicked\_pixel.X, global\_clicked\_pixel.Y);

bool bigCircleSelected = !triangleSelected && !smallCircleSelected && !starSelected &&

    frame.IsPointInCircle((int)C.x, (int)C.y, (int)a,

        global\_clicked\_pixel.X, global\_clicked\_pixel.Y);

float x0 = 0, y0 = 0, x1 = frame.width, y1 = frame.height;

RadialInterpolator selected(x0, y0, x1, y1, COLOR(255, 0, 0), COLOR(255, 0, 0), 0);

RadialInterpolator radialInterpolator(x0, y0, x1, y1, COLOR(255, 0, 0), COLOR(100, 20, 0), global\_angle);

RadialInterpolator radialInterpolator2(x0, y0, x1, y1, COLOR(255, 255, 0), COLOR(20, 255, 255), global\_angle);

// Рисуем описанную окружность

frame.Circle((int)C.x, (int)C.y, (int)a, bigCircleSelected ? selected : radialInterpolator);

//Рисуем треугольник

BarycentricInterpolator triangle(

    triangleA.x + 0.5, triangleA.y + 0.5,

    triangleB.x + 0.5, triangleB.y + 0.5,

    triangleC.x + 0.5, triangleC.y + 0.5, COLOR(255, 255, 0), COLOR(20, 255, 142), COLOR(31, 173, 142, 31));

BarycentricInterpolator triangleMetallic(

    triangleA.x + 0.5, triangleA.y + 0.5,

    triangleB.x + 0.5, triangleB.y + 0.5,

    triangleC.x + 0.5, triangleC.y + 0.5, COLOR(255, 255, 255), COLOR(51, 51, 51), COLOR(128, 128, 128));

if (triangleSelected) {

    frame.Triangle(

        triangleA.x + 0.5, triangleA.y + 0.5,

        triangleB.x + 0.5, triangleB.y + 0.5,

        triangleC.x + 0.5, triangleC.y + 0.5,

        selected);

}

else {

    frame.Triangle(

        triangleA.x + 0.5, triangleA.y + 0.5,

        triangleB.x + 0.5, triangleB.y + 0.5,

        triangleC.x + 0.5, triangleC.y + 0.5,

        triangle);

}

// Рисуем вписанную окружность

frame.Circle((int)C.x, (int)C.y, (int)(a \* 0.5), smallCircleSelected ? selected : radialInterpolator2);

// Добавим заливку для звезды в центре

if (starSelected) {

    frame.Triangle(star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y, selected);

    frame.Triangle(star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y, selected);

    frame.Triangle(star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y, selected);

    frame.Triangle(star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y, selected);

    frame.Triangle(star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y, selected);

    frame.Triangle(star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y, selected);

}

else {

    frame.Triangle(star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y, triangleMetallic);

    frame.Triangle(star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y, triangleMetallic);

    frame.Triangle(star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y, triangleMetallic);

    frame.Triangle(star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y, triangleMetallic);

    frame.Triangle(star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y, triangleMetallic);

    frame.Triangle(star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y, triangleMetallic);

}

**BarycentricInterpolator.h**

#pragma once

#include "Frame.h"

// Класс для расчёта барицентрической интерполяции

class BarycentricInterpolator

{

    float x0, y0, x1, y1, x2, y2, S;

    COLOR C0, C1, C2;

public:

    BarycentricInterpolator(float \_x0, float \_y0, float \_x1, float \_y1, float \_x2, float \_y2, COLOR A0, COLOR A1, COLOR A2) :

        x0(\_x0), y0(\_y0), x1(\_x1), y1(\_y1), x2(\_x2), y2(\_y2),

        S((\_y1 - \_y2)\* (\_x0 - \_x2) + (\_x2 - \_x1) \* (\_y0 - \_y2)), C0(A0), C1(A1), C2(A2)

    {

    }

    COLOR color(float x, float y)

    {

        // Барицентрическая интерполяция

        float h0 = ((y1 - y2) \* (x - x2) + (x2 - x1) \* (y - y2)) / S;

        float h1 = ((y2 - y0) \* (x - x2) + (x0 - x2) \* (y - y2)) / S;

        float h2 = 1 - h0 - h1;

        float r = h0 \* C0.RED + h1 \* C1.RED + h2 \* C2.RED;

        float g = h0 \* C0.GREEN + h1 \* C1.GREEN + h2 \* C2.GREEN;

        float b = h0 \* C0.BLUE + h1 \* C1.BLUE + h2 \* C2.BLUE;

        float a = h0 \* C0.ALPHA + h1 \* C1.ALPHA + h2 \* C2.ALPHA;

        // Из-за погрешности аппроксимации треугольника учитываем, что центр закрашиваемого пикселя может находится вне треугольника.

        // По этой причине значения r, g, b могут выйти за пределы диапазона [0, 255].

        return COLOR(r, g, b, a);

    }

};

**RadialInterpolator.h**

#pragma once

#include "Frame.h"

// Класс для расчёта радиальной интерполяции

class RadialInterpolator

{

    float cx, cy; // Центр прямоугольника

    COLOR C0, C1; // Цвета радиальной заливки

    float angle;  // Начальный угол заливки

public:

    RadialInterpolator(float \_x0, float \_y0, float \_x1, float \_y1, COLOR A0, COLOR A1, float \_angle) :

        cx((\_x0 + \_x1) / 2.0f), cy((\_y0 + \_y1) / 2.0f),

        C0(A0), C1(A1), angle(\_angle)

    {

    }

    COLOR color(float x, float y)

    {

        double dx = (double)x - cx, dy = (double)y - cy;

        double radius = sqrt(dx \* dx + dy \* dy);

        float h0 = (sin(radius / 10 + angle) + 1.0f) / 2;

        float h1 = 1 - h0;

        float r = h0 \* C0.RED + h1 \* C1.RED;

        float g = h0 \* C0.GREEN + h1 \* C1.GREEN;

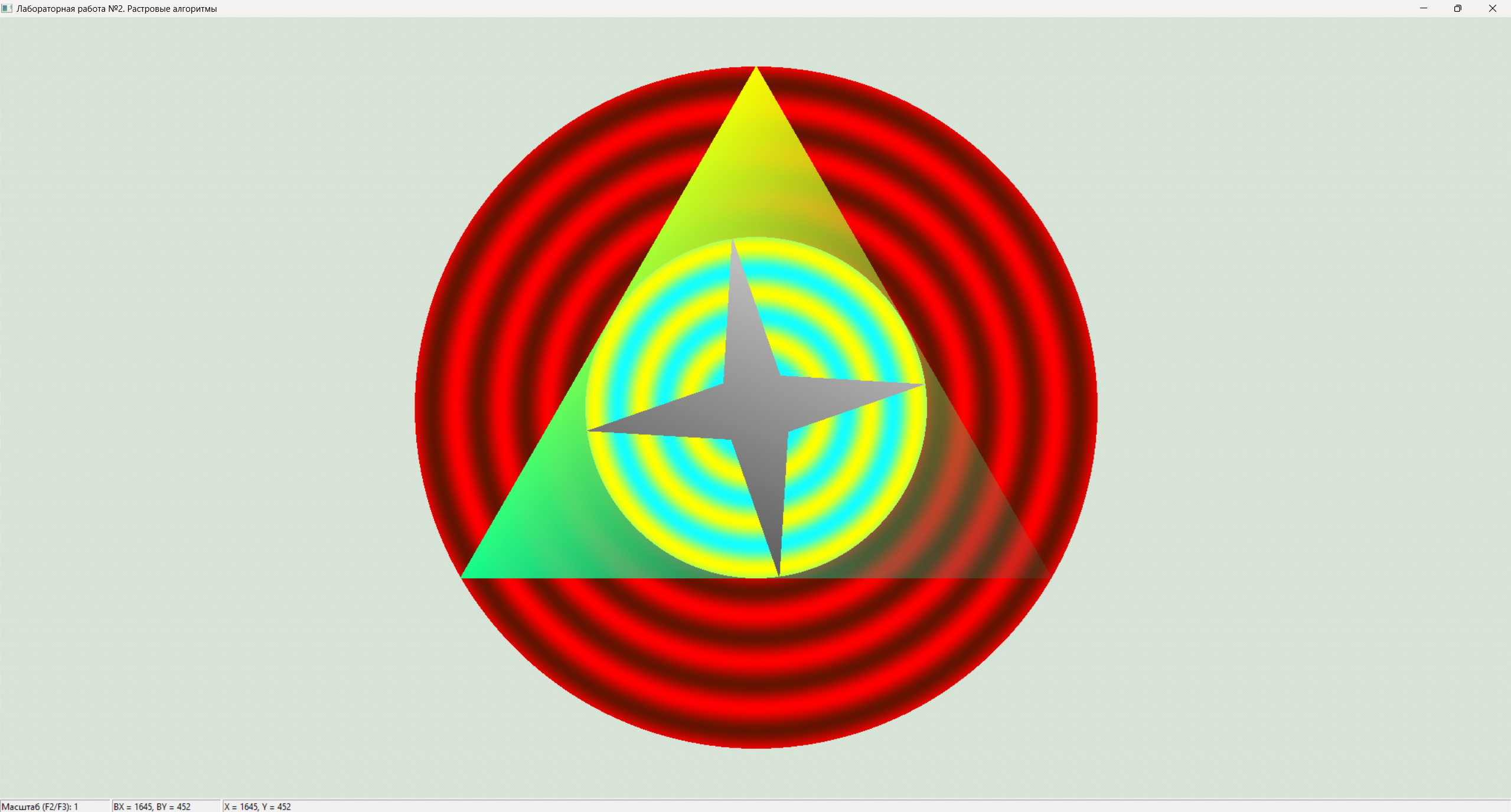
        float b = h0 \* C0.BLUE + h1 \* C1.BLUE;

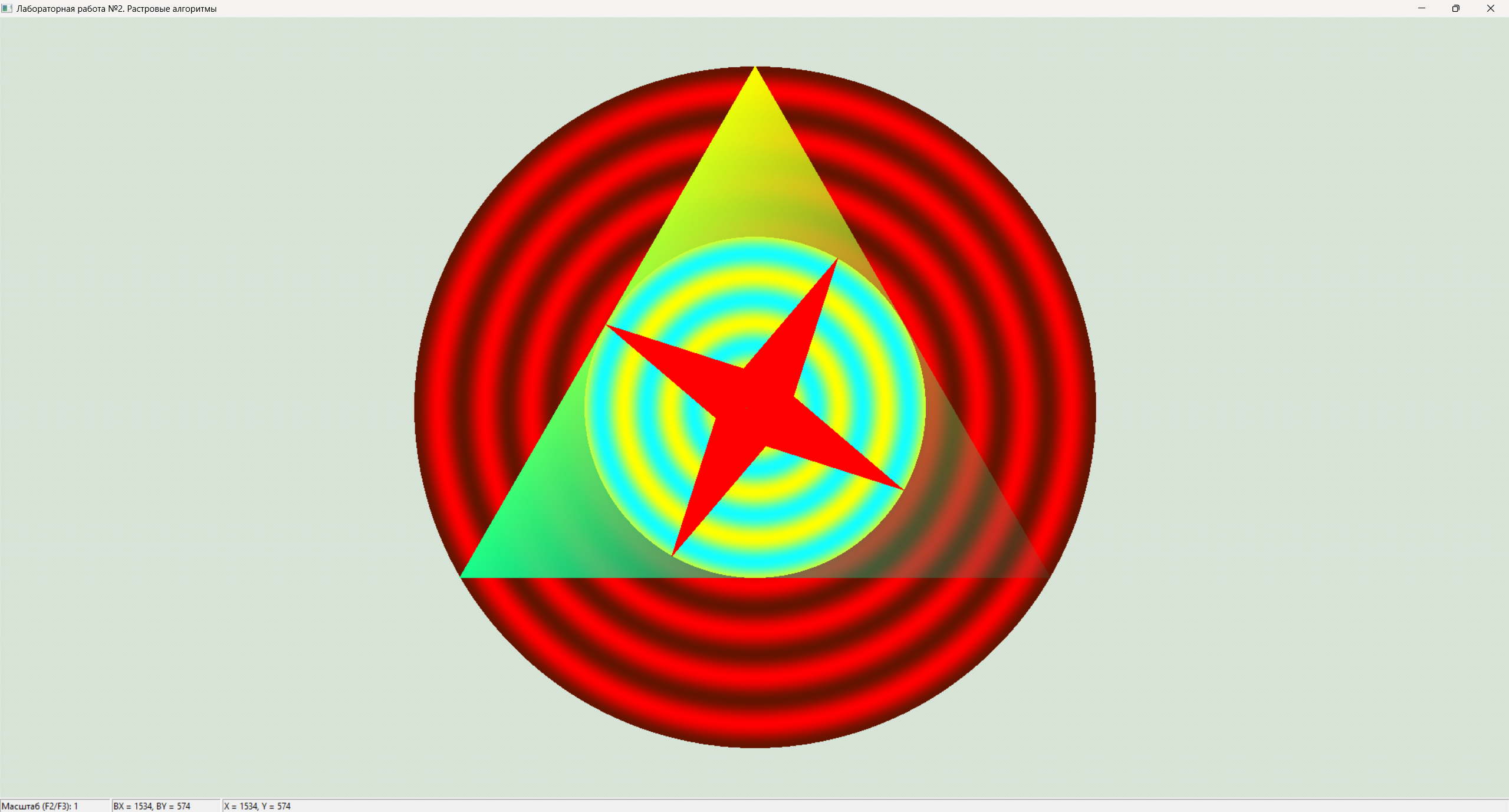
        return COLOR(r, g, b);

    }

};

Ссылка на репозиторий: <https://github.com/IAmProgrammist/comp_graphics/tree/main/lab_2_colored_square>





**Вывод:** в ходе лабораторной работы изучены алгоритмы растровой заливки основных

геометрических фигур: кругов, многоугольников