

**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.

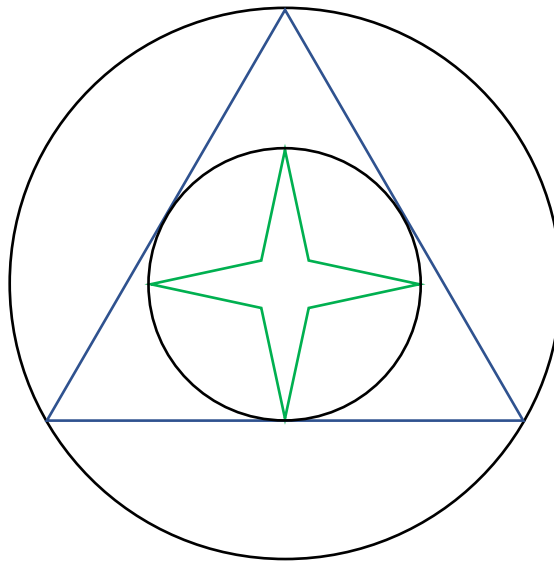
Лабораторная работа №2
Растровая заливка геометрических фигур
Вариант 8

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 W – ширина экрана, H – высота экрана. Диаметр описанной вокруг равностороннего треугольника окружности примем равным $R = 7/8 \min(W, H)$. Диаметр вписанной в треугольник окружности будет в два раза меньше описанного $r = R / 2$. Сторона треугольника будет равна $t = \frac{3R}{\sqrt{3}}$. Введём также центр экрана $C = (W / 2, H / 2)$. Вектор определяющие точки треугольника будут равны:

$$\begin{aligned} A_{\Delta} &= (C_x, C_y - R) \\ B_{\Delta} &= \left(C_x - \frac{t}{2}, C_y + \frac{R}{2}\right) \\ C_{\Delta} &= \left(C_x + \frac{t}{2}, C_y + \frac{R}{2}\right) \end{aligned}$$

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 $S_p = \frac{r}{12}$. Тогда звезда задаётся координатами

$$\begin{aligned} A_S &= (0, r), C_S = (S_p, S_p), D_S = (r, 0), E_S = (S_p, -S_p) \\ F_S &= (0, -r), G_S = (-S_p, -S_p), H_S = (-r, 0), I_S = (-S_p, S_p) \end{aligned}$$

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

$$\begin{aligned} V' &= M_{SRT} \cdot V \\ M_{SRT} &= M_S \cdot M_R \cdot M_T \end{aligned}$$

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

$$M_S = \begin{pmatrix} L_W & 0 & 0 \\ 0 & L_H & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

L_W – коэф.увел. по ширине, L_H – коэф.увел. по высоте

$$M_R = \begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

ϕ – угол поворота

$$M_T = \begin{pmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{pmatrix}$$

X, Y – смещение относительно центра

В нашем случае размер не изменяется ($L_W = L_H = 1$), а смещение относительно центра = C ($X = C_x; Y = C_y$). Для рисования варианта задания представлен текст программы на C++:

Frame.h

```
#ifndef FRAME_H
#define FRAME_H

#include <math.h>

// Структура для задания цвета
typedef struct tagCOLOR
{
    unsigned char RED;        // Компонента красного цвета
    unsigned char GREEN;      // Компонента зелёного цвета
    unsigned char BLUE;       // Компонента синего цвета
    unsigned char ALPHA;      // Прозрачность (альфа канал)

    tagCOLOR() : RED(0), GREEN(0), BLUE(0), ALPHA(255) { }

    tagCOLOR(int red, int green, int blue, int alpha = 255)
        : RED(red), GREEN(green), BLUE(blue), ALPHA(alpha)
    {
        if (red < 0) RED = 0;
        else if (red > 255) RED = 255;
        if (green < 0) GREEN = 0;
        else if (green > 255) GREEN = 255;
        if (blue < 0) BLUE = 0;
        else if (blue > 255) BLUE = 255;
        if (alpha < 0) ALPHA = 0;
        else if (alpha > 255) ALPHA = 255;
    }
} COLOR;

// Структура для задания цвета
typedef struct HSVCOLOR
{
    double H;        // Компонента красного цвета
    double S;        // Компонента зелёного цвета
    double V;        // Компонента синего цвета
    unsigned char ALPHA; // Прозрачность (альфа канал)

    HSVCOLOR() : H(0), S(0), V(0), ALPHA(255) { }

    HSVCOLOR(double hue, double saturation, double value, int alpha = 255)
        : H(hue), S(saturation), V(value), ALPHA(alpha)
    {
        if (hue < 0) H = 0;
        else if (hue > 360) H = 360;
        if (saturation < 0) S = 0;
        else if (saturation > 1) S = 1;
        if (value < 0) value = 0;
        else if (value > 1) V = 1;
        if (alpha < 0) ALPHA = 0;
        else if (alpha > 255) ALPHA = 255;
    }
}

COLOR convertToRgb() {
    int hi = int(floor(H / 60)) % 6;
    double f = H / 60 - floor(H / 60);
    int copyV = V * 255;
    int v = (int)(copyV);
    int p = (int)(copyV * (1 - S));
    int q = (int)(copyV * (1 - f * S));
    int t = (int)(copyV * (1 - (1 - f) * S));
    if (hi == 0)
        return { v, t, p, ALPHA };
    if (hi == 1)
```

```

        return { q, v, p, ALPHA };
    else if (hi == 2)
        return { p, v, t, ALPHA };
    else if (hi == 3)
        return { p, q, v, ALPHA };
    else if (hi == 4)
        return { t, p, v, ALPHA };
    return { v, p, q, ALPHA };
}

} HSVCOLOR;

template<typename TYPE> void swap(TYPE& a, TYPE& b)
{
    TYPE t = a;
    a = b;
    b = t;
}

// Буфер кадра
class Frame
{
    // Указатель на массив пикселей
    // Буфер кадра будет представлять собой матрицу, которая располагается в памяти в виде
    // непрерывного блока
    COLOR* pixels;

    // Указатели на строки пикселей буфера кадра
    COLOR** matrix;

public:

    // Размеры буфера кадра
    int width, height;

    Frame(int _width, int _height) : width(_width), height(_height)
    {
        int size = width * height;

        // Создание буфера кадра в виде непрерывной матрицы пикселей
        pixels = new COLOR[size];

        // Указатели на строки пикселей запишем в отдельный массив
        matrix = new COLOR * [height];

        // Инициализация массива указателей
        for (int i = 0; i < height; i++)
        {
            matrix[i] = pixels + (size_t)i * width;
        }

        // Задаёт цвет color пикселю с координатами (x, y)
        void SetPixel(int x, int y, COLOR color)
        {
            matrix[y][x] = color;
        }

        // Возвращает цвет пикселя с координатами (x, y)
        COLOR GetPixel(int x, int y)
        {
            return matrix[y][x];
        }

        void Triangle(float x0, float y0, float x1, float y1, float x2, float y2, COLOR color)
        {
            // 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)
        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)
            SetPixel(x, y, color);
        }

        rawX0 += (x2 - x1) / (y2 - y1);
        rawX1 += (x2 - x0) / (y2 - y0);
    }
}

template <class InterpolatorClass>
void Triangle(float x0, float y0, float x1, float y1, float x2, float y2, InterpolatorClass&
Interpolator)
{
    // Отсортируем точки таким образом, чтобы выполнялось 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;
}

}

~Frame(void)
{
    delete[] pixels;
    delete[] matrix;
}

};

#endif // FRAME_H

```

Painter.h

```

#ifndef PAINTER_H
#define PAINTER_H

#include "Frame.h"
#include "Matrices.h"
#include "RadialInterpolator.h"
#include "BarycentricInterpolator.h"
#include "SectorInterpolator.h"

// Установите 1 для отрисовки основного варианта, 0 - для отрисовки задания с защиты (сектор-
круг)
#define MAIN_TASK 1

// Угол поворота фигуры
float global_angle = 0;

// Координаты последнего пикселя, который выбрал пользователь
struct
{
    int X, Y;
} global_clicked_pixel = { -1, -1 };

enum DrawMode {
    SECTOR = 0,
    RADIAL = 1,
    BARYCENTRIC = 2
};

DrawMode bigCircleDrawMode = SECTOR;
DrawMode triangleDrawMode = BARYCENTRIC;

```

```

DrawMode smallCircleDrawMode = RADIAL;
DrawMode starDrawMode = BARYCENTRIC;

typedef struct
{
    float x;
    float y;
} coordinate;

class Painter
{
public:

    void Draw(Frame& frame)
    {
        // Шахматная текстура
        for (int y = 0; y < frame.height; y++)
            for (int x = 0; x < frame.width; x++)
            {
                if ((x + y) % 2 == 0)
                    frame.SetPixel(x, y, { 230, 255, 230 }); // Золотистый цвет
                //frame.SetPixel(x, y, { 217, 168, 14 });
                else
                    frame.SetPixel(x, y, { 200, 200, 200 }); // Чёрный цвет
                //frame.SetPixel(x, y, { 255, 255, 255 }); // Белый цвет
            }

        int W = frame.width, H = frame.height;
        // Размер рисунка возьмём меньше (7 / 8), чтобы он не касался границ экрана
        float a = 7.0f / 8 * ((W < H) ? W - 1 : H - 1);
        if (a < 1) return; // Если окно очень маленькое, то ничего не рисуем
        float angle = -global_angle; // Угол поворота
        a = a / 2;
        coordinate C = { W / 2, H / 2 };

        // Код для отрисовки основного задания.
        if (MAIN_TASK) {
            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, HSVCOLOR(255, 1, 1).convertToRgb(),
COLOR(255, 0, 0), 0);

    SectorInterpolator sectorInterpolator(C.x, C.y);

    RadialInterpolator radialInterpolator(x0, y0, x1, y1, COLOR(255, 0, 0), HSVCOLOR(311,
.1, .1).convertToRgb(), global_angle);

    BarycentricInterpolator triangleInterpolator(
        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),
        HSVCOLOR(51, 1, .5).convertToRgb(),
        COLOR(128, 128, 128));

    // Рисуем описанную окружность

    SectorInterpolator sector(C.x, C.y);
    if (bigCircleSelected)
        frame.Circle((int)C.x, (int)C.y, (int)a, selected);
    else if (bigCircleDrawMode == DrawMode::SECTOR)
        frame.Circle((int)C.x, (int)C.y, (int)a, sectorInterpolator);
    else if (bigCircleDrawMode == DrawMode::RADIAL)
        frame.Circle((int)C.x, (int)C.y, (int)a, radialInterpolator);

```

```

else if (bigCircleDrawMode == DrawMode::BARYCENTRIC)
    frame.Circle((int)C.x, (int)C.y, (int)a, triangleInterpolator);

//Рисуем треугольник

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 if (triangleDrawMode == DrawMode::SECTOR)
    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,
        sectorInterpolator);
else if (triangleDrawMode == DrawMode::RADIAL)
    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,
        radialInterpolator);
else if (triangleDrawMode == DrawMode::BARYCENTRIC)
    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,
        triangleInterpolator);

if (smallCircleSelected)
    frame.Circle((int)C.x, (int)C.y, (int)(a * 0.5), selected);
else if (smallCircleDrawMode == DrawMode::SECTOR)
    frame.Circle((int)C.x, (int)C.y, (int)(a * 0.5), sectorInterpolator);
else if (smallCircleDrawMode == DrawMode::RADIAL)
    frame.Circle((int)C.x, (int)C.y, (int)(a * 0.5), radialInterpolator);
else if (smallCircleDrawMode == DrawMode::BARYCENTRIC)
    frame.Circle((int)C.x, (int)C.y, (int)(a * 0.5), triangleInterpolator);

// 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 if (starDrawMode == DrawMode::SECTOR) {
    frame.Triangle(star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y,
sectorInterpolator);
    frame.Triangle(star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y,
sectorInterpolator);
    frame.Triangle(star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y,
sectorInterpolator);
    frame.Triangle(star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y,
sectorInterpolator);
}

```

```

        frame.Triangle(star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y,
sectorInterpolator);
        frame.Triangle(star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y,
sectorInterpolator);
    }
    else if (starDrawMode == DrawMode::RADIAL) {
        frame.Triangle(star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y,
radialInterpolator);
        frame.Triangle(star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y,
radialInterpolator);
        frame.Triangle(star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y,
radialInterpolator);
        frame.Triangle(star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y,
radialInterpolator);
        frame.Triangle(star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y,
radialInterpolator);
        frame.Triangle(star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y,
radialInterpolator);
    }
    else if (starDrawMode == DrawMode::BARYCENTRIC) {
        frame.Triangle(star[7].x, star[7].y, star[0].x, star[0].y, star[1].x, star[1].y,
triangleInterpolator);
        frame.Triangle(star[1].x, star[1].y, star[2].x, star[2].y, star[3].x, star[3].y,
triangleInterpolator);
        frame.Triangle(star[5].x, star[5].y, star[4].x, star[4].y, star[3].x, star[3].y,
triangleInterpolator);
        frame.Triangle(star[5].x, star[5].y, star[6].x, star[6].y, star[7].x, star[7].y,
triangleInterpolator);
        frame.Triangle(star[7].x, star[7].y, star[1].x, star[1].y, star[3].x, star[3].y,
triangleInterpolator);
        frame.Triangle(star[7].x, star[7].y, star[5].x, star[5].y, star[3].x, star[3].y,
triangleInterpolator);
    }
}
else {
    // Ожидаем после проверки
}
// Рисуем пиксель, на который кликнул пользователь
if (global_clicked_pixel.X >= 0 && global_clicked_pixel.X < W &&
    global_clicked_pixel.Y >= 0 && global_clicked_pixel.Y < H)
    frame.SetPixel(global_clicked_pixel.X, global_clicked_pixel.Y, { 34, 175, 60 }); //
Пиксель зелёного цвета
}
};

#endif // PAINTER_H

```

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 находиться 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);
    }
};

```

SectorInterpolator.h

```

#pragma once
#include "Frame.h"

// Класс для расчёта барицентрической интерполяции
class SectorInterpolator
{
    float c_x, c_y;
    COLOR C0, C1, C2;

    inline double getangle(int x, int y) {

```

```

        double x1 = 0;
        double y1 = 1;
        double x2 = x - c_x;
        double y2 = y - c_y;

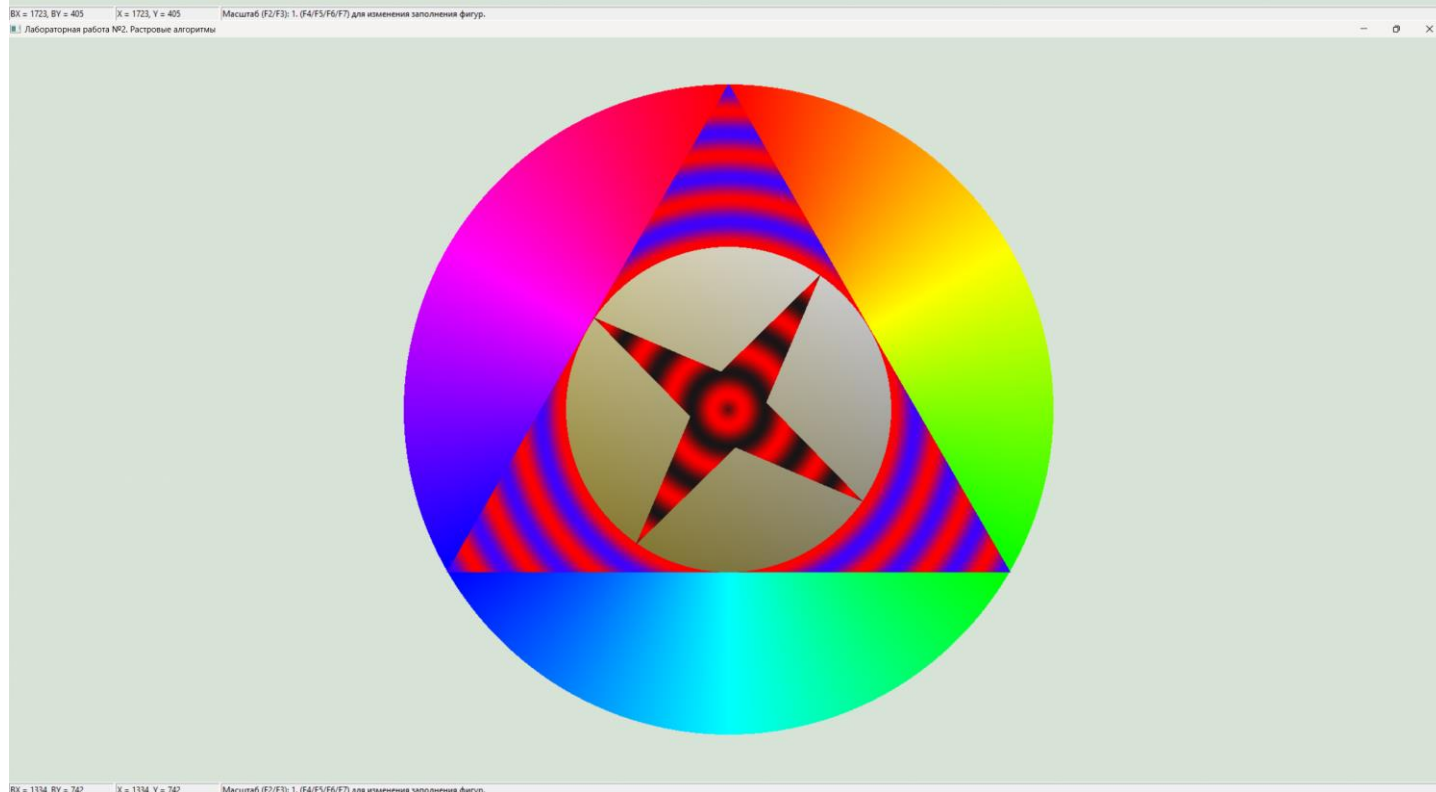
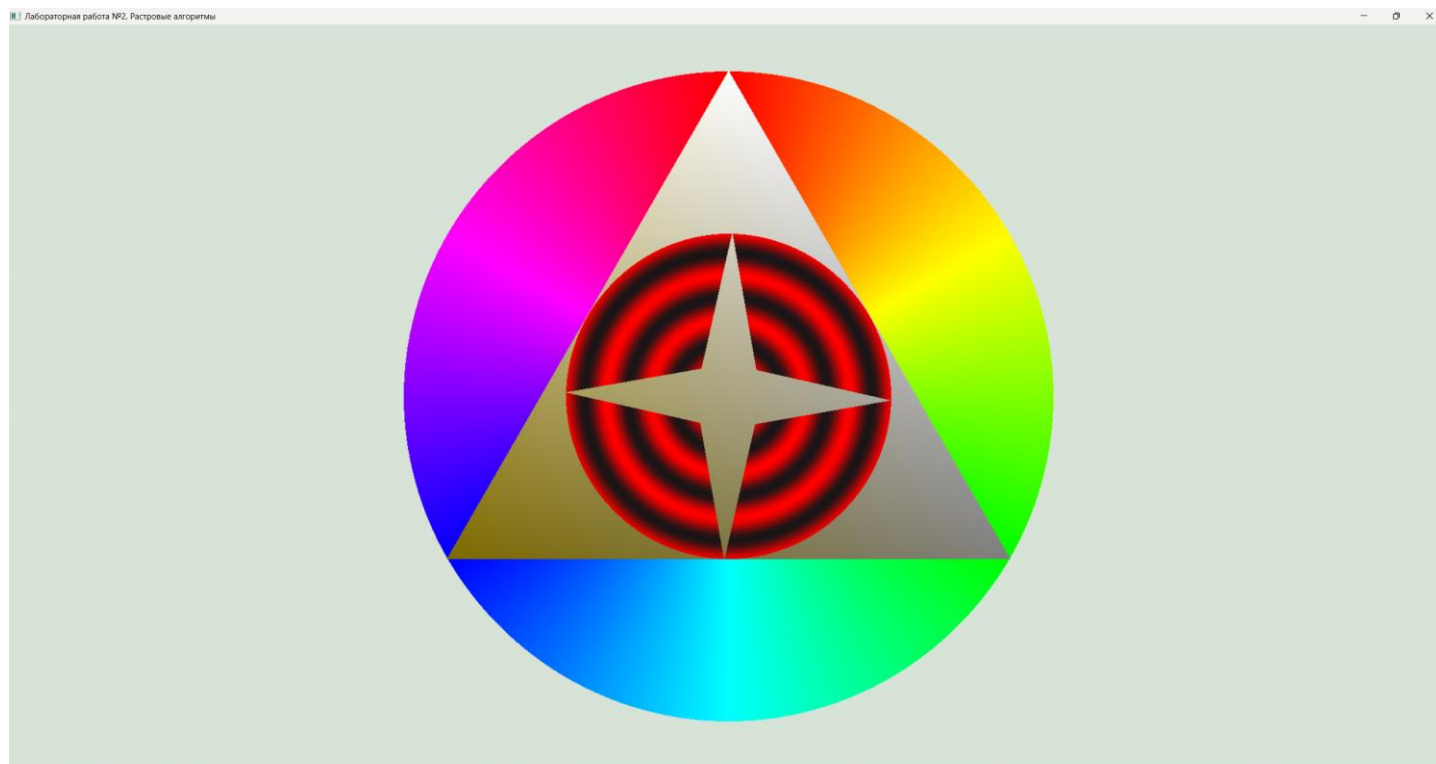
        double dot = x1 * x2 + y1 * y2;
        double det = x1 * y2 - y1 * x2;
        return atan2(det, dot);
    }
public:
    SectorInterpolator(float c_x, float c_y) :
        c_x(c_x), c_y(c_y)
    {
    }

    COLOR color(float x, float y)
    {
        return HSVCOLOR(180 + (getangle(x, y) * 180) / 3.14, 1, 1).convertToRgb();
    }
};

```

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