

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)**

Кафедра программного обеспечения вычислительной техники и автоматизированных  
систем



## **Лабораторная работа №2**

по дисциплине: Компьютерная графика

тема: «Растровая заливка геометрических фигур»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

Осипов Олег Васильевич

Белгород 2024 г.

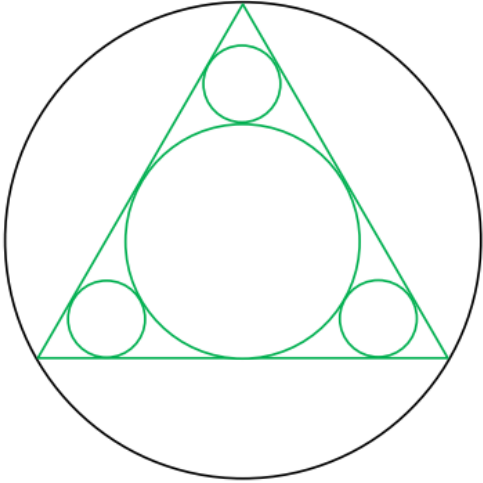
**Цель работы:** изучение алгоритмов растровой заливки основных геометрических фигур: кругов, многоугольников.

### **Вариант 3**

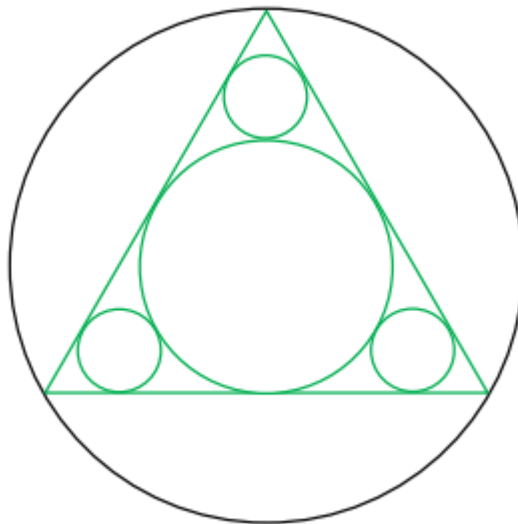
#### **Требования к программе**

1. Программа должна быть написана на языках Си или C++.
2. Фигуры, нарисованные в первой лабораторной работе, необходимо заполнить цветом. Реализовать следующие способы заливки: барицентрическая, радиальная, секторная. Изучить пример программы `lab_2_colored_square.vcxproj`. Реализовать возможность выбора пользователем (например, клавишами) различных способов заливки для каждой геометрической фигуры.
3. Программа должна реагировать на выделение пользователем фигур, когда он выбирает их с использованием кнопок мыши. Фигура должна подсвечиваться другой текстурой или цветом, когда она выбрана после наведения (клика) на неё курсора мыши.
4. В программе должна быть предусмотрена возможность изменять прозрачность фигур. Фигуры должны быть нарисованы в порядке убывания площади, чтобы большие фигуры не закрывали маленькие.
5. Изображение должно масштабироваться строго по центру окна с радиусом  $7/8$  относительно размера окна (см. пример проекта `lab_1_basics.vcxproj`).
6. Пользователь должен иметь возможность менять размер окна и изменять разрешение пикселей. См. пример проекта `lab_1_basics.vcxproj`, в котором разрешение изменяется клавишами F2/F3.
7. Если в задании указано, что требуется реализовать анимацию (например, вращение), то перерисовку изображения нужно выполнять по таймеру 30 раз в секунду.

8. Цвет примитивов выбрать по собственному усмотрению.

3,12,21		Реализовать вращение внутреннего зелёного треугольника и трёх кругов по часовой стрелке.
---------	---	--

### Вывод формул



Предположим, что треугольник вписанный в окружность равносторонний. Тогда сторона треугольника будет равна радиусу описанной окружности умноженной на корень из трех:

$$a = R_{\text{общ}} * \sqrt{3}$$

где  $a$  - сторона треугольника, а  $R_{\text{общ}}$  - радиус описанной окружности.

Зная сторону треугольника найдем радиус большей вписанной окружности.

Обозначим радиус вписанной окружности как -  $R$ . Площадь равностороннего треугольника можно найти по формуле -  $S = \frac{a^2\sqrt{3}}{4}$ .

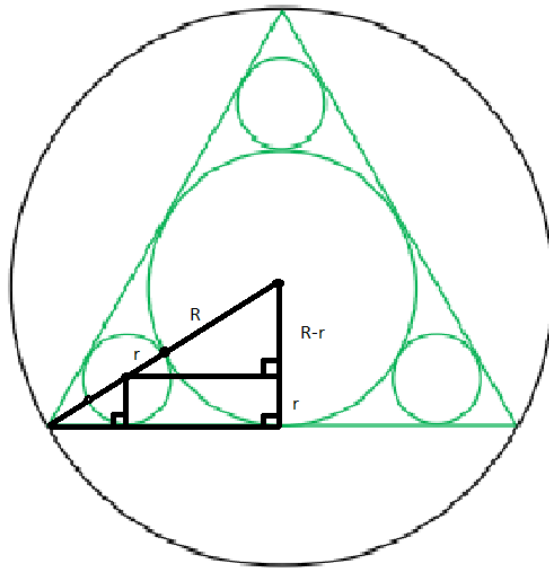
Периметр равностороннего треугольника равен -  $P = 3 * a$

Радиус вписанной окружности можно выразить через площадь и полупериметр:

$$R = \frac{S}{P/2}$$

подставляя в уравнение значения площади и периметра получим:

$$R = \frac{\frac{\sqrt{3}}{4}a^2}{3a/2} = \frac{\frac{\sqrt{3}}{4}a^2 * 2}{3a} = \frac{a\sqrt{3}}{6}$$



Зная радиус вписанной окружности, найдем радиусы трех окружностей, которые находятся в вершинах треугольника.

Опустим перпендикуляр из центра меньшей окружности на радиус большей окружности, проведенный в точку касания со стороной рассматриваемого угла. Получим прямоугольный треугольник с гипотенузой  $R + r$ , катетом  $R - r$  и углом в 30 градусов, противолежащим этому катету. Поэтому  $R + r = 2(R - r)$ .

Отсюда находим, что  $R = 3r \Rightarrow r = R / 3$ .

## Реализация заливок:

### Заливка треугольника:

```
template <class ShaderClass>
void Triangle(float x0, float y0, float x1, float y1, float x2, float y2, ShaderClass& shader, int alpha = 255)
{
    // Упорядочивание вершин по координате y для правильного рисования
    if (y1 < y0) { swap(y1, y0); swap(x1, x0); } // Если y1 меньше y0, меняем их местами
    if (y2 < y1) { swap(y2, y1); swap(x2, x1); } // Если y2 меньше y1, меняем их местами
    if (y1 < y0) { swap(y1, y0); swap(x1, x0); } // Повторная проверка для y0 и y1

    // Преобразование координат y в целые числа
    int Y0 = static_cast<int>(y0 + 0.5f);
    int Y1 = static_cast<int>(y1 + 0.5f);
    int Y2 = static_cast<int>(y2 + 0.5f);

    // Ограничение координат по высоте, чтобы не выйти за границы
    Y0 = max(0, min(Y0, height));
    Y1 = max(0, min(Y1, height));
    Y2 = max(0, min(Y2, height));

    // Вычисление приращений по x для каждой из сторон треугольника
    float dx0_1 = (x1 - x0) / (y1 - y0); // Изменение x от 0 до 1
    float dx0_2 = (x2 - x0) / (y2 - y0); // Изменение x от 0 до 2
    float dx1_2 = (x2 - x1) / (y2 - y1); // Изменение x от 1 до 2

    // Начальные значения для x координат
    float X0 = x0, X1 = x0;

    // Рисование нижней части треугольника
    for (float y = Y0 + 0.5f; y < Y1; y++)
    {
        int ix0 = static_cast<int>(X0 + 0.5f); // Округление x0
        int ix1 = static_cast<int>(X1 + 0.5f); // Округление x1
        if (ix0 > ix1) swap(ix0, ix1); // Убедимся, что ix0 меньше ix1

        // Ограничение по ширине
        ix0 = max(0, ix0); // Минимальное значение ix0
        ix1 = min(width, ix1); // Максимальное значение ix1

        // Рисование пикселей между ix0 и ix1 на текущей строке y
        for (int x = ix0; x < ix1; x++)
        {
            COLOR color = shader.color(x + 0.5f, y); // Получение цвета из шейдера
            color.ALPHA = alpha; // Установка альфа-канала
            SetPixel(x, y, color); // Установка пикселя
        }

        // Обновление x координат для следующей строки
        X0 += dx0_1; // Обновление X0
        X1 += dx0_2; // Обновление X1
    }

    // Рисование верхней части треугольника
    X0 = x1; // Начинаем с x1
    X1 = x0 + (Y1 - Y0) * dx0_2; // Обновление X1 для верхней части
    for (float y = Y1 + 0.5f; y < Y2; y++)
    {
        int ix0 = static_cast<int>(X0 + 0.5f); // Округление x0
        int ix1 = static_cast<int>(X1 + 0.5f); // Округление x1
        if (ix0 > ix1) swap(ix0, ix1); // Убедимся, что ix0 меньше ix1
    }
}
```

```
// Ограничение по ширине
ix0 = max(0, ix0); // Минимальное значение ix0
ix1 = min(width, ix1); // Максимальное значение ix1

// Рисование пикселей между ix0 и ix1 на текущей строке y
for (int x = ix0; x < ix1; x++)
{
    COLOR color = shader.color(x + 0.5f, y); // Получение цвета из шейдера
    color.ALPHA = alpha; // Установка альфа-канала
    SetPixel(x, y, color); // Установка пикселя
}

// Обновление x координат для следующей строки
X0 += dx1_2; // Обновление X0
X1 += dx0_2; // Обновление X1
}
}
```

## Заливка круга:

```
template <class ShaderClass>
void FillCircle(int x0, int y0, int radius, ShaderClass shader, int alpha)
{
    // Начальные значения для переменных, определяющих координаты и радиус
    //                                     круга

    int x = 0, y = radius;
    int d = 1 - radius; // Начальное значение для параметра окружности

    // Переменные для линий (в данном случае не используются)
    int lineX1 = 0, lineY1 = 0, lineX2 = 0, lineY2 = 0;
    bool noLine = (lineX1 == 0 && lineY1 == 0 && lineX2 == 0 && lineY2 ==
        0); // Проверка на отсутствие линий

    // Коэффициенты для уравнения линии
    int A = lineY2 - lineY1;
    int B = lineX1 - lineX2;
    int C = lineX2 * lineY1 - lineX1 * lineY2;

    // Основной цикл для рисования круга
    while (x <= y)
    {
        // Рисование горизонтальных линий на верхней и нижней частях круга
        DrawLine_with_SetPixel(x0 - x, y0 + y, x0 + x, y0 + y, A, B, C, noLine,
            shader, alpha);
        DrawLine_with_SetPixel(x0 - x, y0 - y, x0 + x, y0 - y, A, B, C, noLine,
            shader, alpha);

        // Проверка, чтобы избежать дублирования, если x и y равны
        if (y != x) {
            // Рисование вертикальных линий на левой и правой частях круга
            DrawLine_with_SetPixel(x0 - y, y0 + x, x0 + y, y0 + x, A, B, C,
                noLine, shader, alpha);

            if (x != 0) {
                // Рисование вертикальных линий на нижней части круга
                DrawLine_with_SetPixel(x0 - y, y0 - x, x0 + y, y0 - x, A, B,
                    C, noLine, shader, alpha);
            }
        }

        // Обновление параметра d и координаты y
        if (d < 0)
        {
            d += 3 + 2 * x; // Если d меньше 0, обновляем по формуле deltaE
        }
        else
        {
            d += 5 + 2 * (x - y); // Если d больше или равен 0, обновляем по
                формуле deltaSE
        }
        y--; // Уменьшаем y
    }
    x++; // Увеличиваем x
}
```

## Реализация шейдеров:

### Барицентрическая:

```
// Класс для расчёта барицентрической интерполяции
class BarycentricInterpolator {
float x0, y0, x1, y1, x2, y2, S;
COLOR C0, C1, C2;

// Вспомогательная функция для вычисления координат
void calculateCoordinates(float _x0, float _y0, float r) {
x1 = _x0 + r * cos(PI / 3);
y1 = _y0 + r * sin(PI / 3);
x2 = _x0 + r * cos(PI); // 180 degrees
y2 = _y0 + r * sin(PI);
float x3 = _x0 + r * cos(5 * PI / 3);
float y3 = _y0 + r * sin(5 * PI / 3);

// Сортировка координат по y
if (y2 < y1) {
std::swap(y2, y1);
std::swap(x2, x1);
}
if (y3 < y2) {
std::swap(y3, y2);
std::swap(x3, x2);
}
if (y2 < y1) {
std::swap(y2, y1);
std::swap(x2, x1);
}

// Присваиваем значения
this->x0 = x1;
this->y0 = y1;
this->x1 = x2;
this->y1 = y2;
this->x2 = x3;
this->y2 = y3;

// Вычисляем S (площадь треугольника)
S = (y1 - y2) * (x0 - x2) + (x2 - x1) * (y0 - y2);
}

public:
// Конструктор с ручным заданием координат
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), C0(A0), C1(A1), C2(A2) {

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

// Конструктор с радиусом
BarycentricInterpolator(float _x0, float _y0, float r, COLOR A0, COLOR A1, COLOR A2)
: C0(A0), C1(A1), C2(A2) {
calculateCoordinates(_x0, _y0, r);
}
```



```

// Метод получения цвета по координатам
COLOR color(float x, float y) {
float h0 = ((y1 - y2) * (x - x2) + (x2 - x1) * (y - y2)) / S;
float h1 = ((y2 - y0) * (x - x2) + (x0 - x2) * (y - y2)) / S;
float h2 = ((y0 - y1) * (x - x1) + (x1 - x0) * (y - y1)) / S;

// Проверка на выход за границы треугольника
if (h0 < -1E-6 || h1 < -1E-6 || h2 < -1E-6) {
return COLOR(0, 0, 0); // Ошибка алгоритма, возвращаем черный цвет
}

// Вычисления цветовых компонент
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;

r = max(min(r, 255), 0);
g = max(min(g, 255), 0);
b = max(min(b, 255), 0);

// Возврат цвета
return COLOR(r, g, b, a);
}
};

```

## Радиальная:

```

// Класс для расчёта радиальной заливки
class RadialBrush {
float cx, cy; // Центр прямоугольника
COLOR C0, C1; // Цвета радиальной заливки
float angle; // Начальный угол заливки

public:
// Конструктор для инициализации значения
RadialBrush(float _x, float _y, COLOR A0, COLOR A1, float _angle)
: cx(_x), cy(_y), C0(A0), C1(A1), angle(_angle) {}

// Метод для получения цвета по координатам
COLOR color(float x, float y) {
double dx = static_cast<double>(x) - cx; // Разница по X
double dy = static_cast<double>(y) - cy; // Разница по Y
double radiusSquared = dx * dx + dy * dy; // Избегаем вычисления квадратного корня

// Нормируем радиус, можем использовать радиус вместо реального радиуса
float normRadius = sqrt(radiusSquared);

// Вычисление перехода цветов по синусоиде
float h0 = (sin(normRadius / 4 + angle) + 1.0f) * 0.5f;
float h1 = 1.0f - h0;

// Вычисление окончательного цвета
return COLOR(
h0 * C0.RED + h1 * C1.RED,
h0 * C0.GREEN + h1 * C1.GREEN,
h0 * C0.BLUE + h1 * C1.BLUE
);
}
};

```

## Секторная:

```
class SectorFill {
float cx, cy;      // Центр сектора
COLOR C0, C1;     // Цвета (не используются в текущем примере, можно удалить)
int radius;       // Радиус сектора
float startAngle;  // Начальный угол заливки в градусах

public:
// Конструктор для инициализации значений
SectorFill(float _x, float _y, float _radius, float angle, float mult)
: cx(_x), cy(_y), radius(static_cast<int>(_radius)),
startAngle(angle * mult) { }

// Преобразование из HSV в RGB
COLOR ColorFromHSV(double hue, double saturation, double value) {
COLOR color;
int hi = static_cast<int>(hue / 60) % 6;
double f = hue / 60 - static_cast<int>(hue / 60);
int v = static_cast<int>(value * 255);
int p = static_cast<int>(v * (1 - saturation));
int q = static_cast<int>(v * (1 - f * saturation));
int t = static_cast<int>(v * (1 - (1 - f) * saturation));

switch (hi) {
case 0: color = COLOR(v, t, p); break;
case 1: color = COLOR(q, v, p); break;
case 2: color = COLOR(p, v, t); break;
case 3: color = COLOR(p, q, v); break;
case 4: color = COLOR(t, p, v); break;
default: color = COLOR(v, p, q); break;
}

return color;
}

// Метод для получения цвета по координатам
COLOR color(float x, float y) {
double dx = static_cast<double>(x) - cx;
double dy = static_cast<double>(y) - cy;
double distanceSquared = dx * dx + dy * dy; // Избегаем вычисления sqrt
double distance = sqrt(distanceSquared);

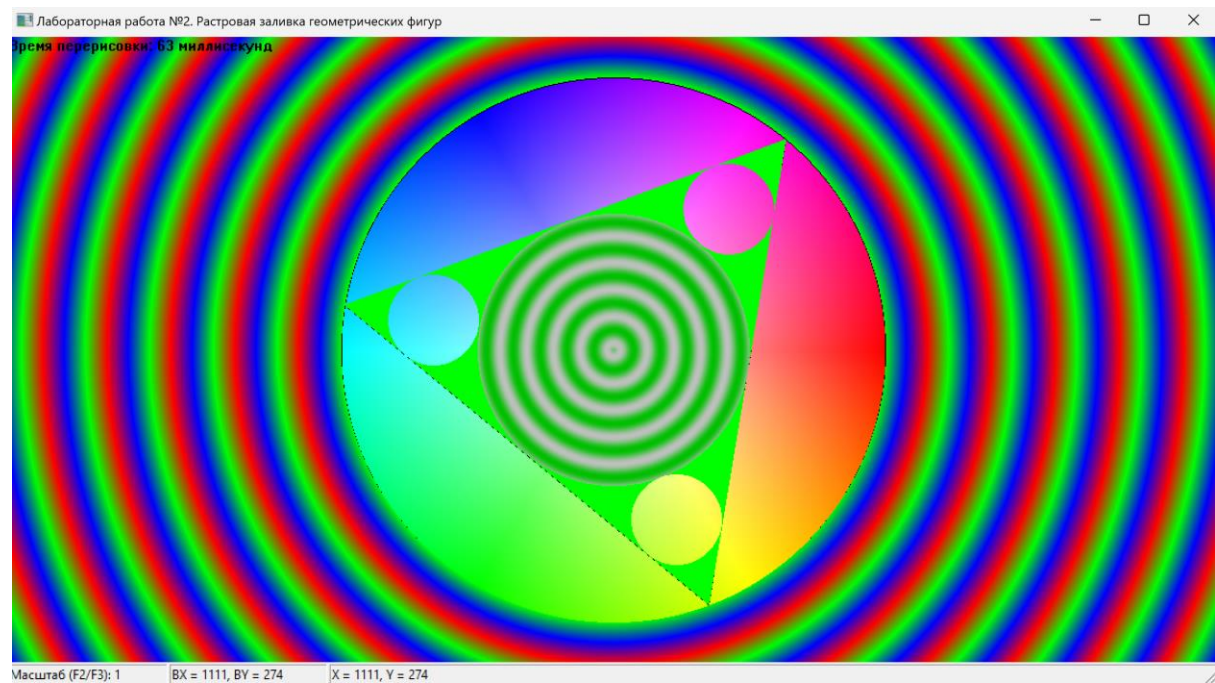
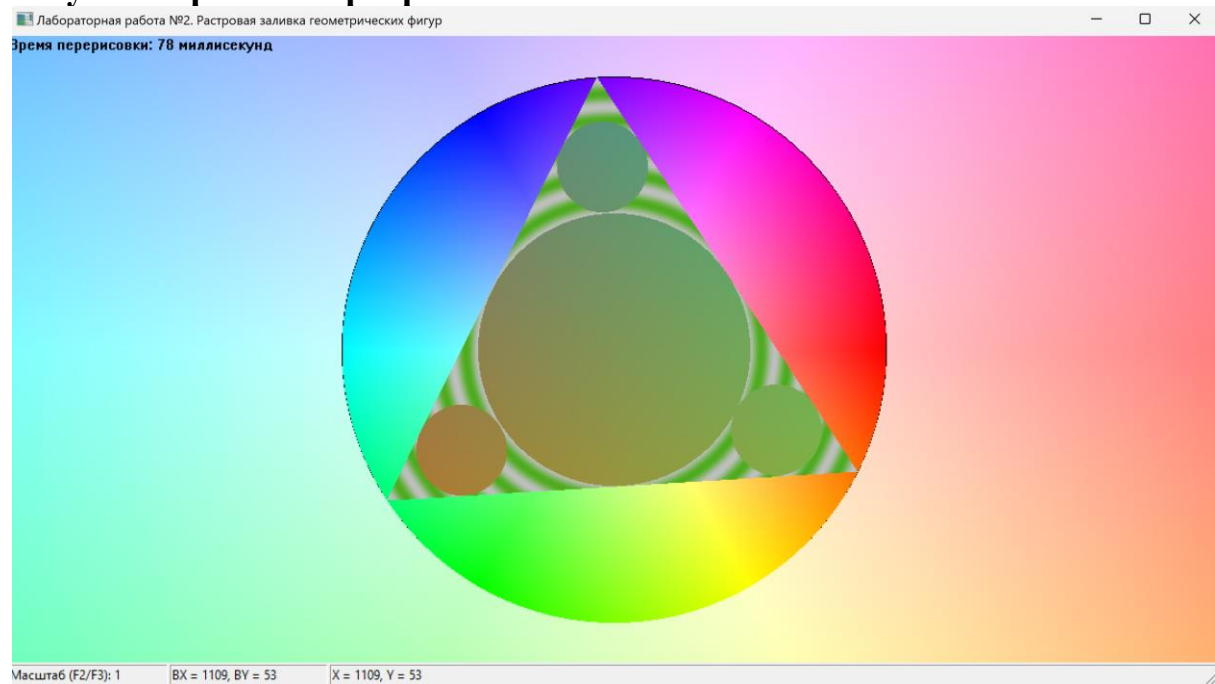
if (distance > radius) {
return COLOR(0, 0, 0); // Черный цвет вне сектора
}

double angle = atan2(dy, dx) * 180.0 / PI; // Используем M_PI для точности
angle = fmod(angle + startAngle, 360.0);
if (angle < 0) angle += 360.0;

double saturation = distance / radius;
double value = 1.0;

return ColorFromHSV(angle, saturation, value);
}
```

## Результат работы программы:



**Вывод:** В ходе выполнения лабораторной работы мы изучили алгоритмы растровой заливки основных геометрических фигур: кругов, многоугольников