**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Лабораторная работа № 4

**Дисциплина** Компьютерная графика

**Тема** Реализация и исследование алгоритмов генерации окружности и эллипса

**Студент** Хетагуров П.К.

**Группа ИУ7-45**

**Оценка (баллы) _____**

**Преподаватель** Куров А. В

Москва.
2020 г.

**Цель работы:**

Реализация алгоритмов построения окружности, исследование и сравнение визуальных и временных характеристик алгоритмов

**Задание:**

1) Рисование отдельных окружностей и сравнение их визуальных характеристик с помощью:
    1) Канонического уравнения X^2+Y^2=R^2
    2) Параметрического уравнения X=Rcost, Y=Rsint
    3) Алгоритма Брезенхема
    4) Алгоритма средней точки
    5) Построение окружности с помощью библиотечной функции
2) Рисование отдельных окружностей и сравнение их визуальных характеристик с помощью:
    1) Канонического уравнения X^2/a^2+Y^2/b^2=1
    2) Параметрического уравнения X=acost, Y=bsint
    3) Алгоритма Брезенхема (модифицировать самостоятельно)
    4) Алгоритма средней точки
    5) Построение эллипса с помощью библиотечной функции

3) Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических окружностей.

4) Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических эллипсов.

Дополнительно:

5) Сравнение временных характеристик алгоритмов

**Теоретическая часть:**

Генерацию окружности можно провести в одном октанте и отразить в другие, для уменьшения кол-ва вычислений.

Так как эллипс имеет только две оси симметрии, то генерацию эллипса необходимо произвести в одной четверти и отразить в три оставшиеся.

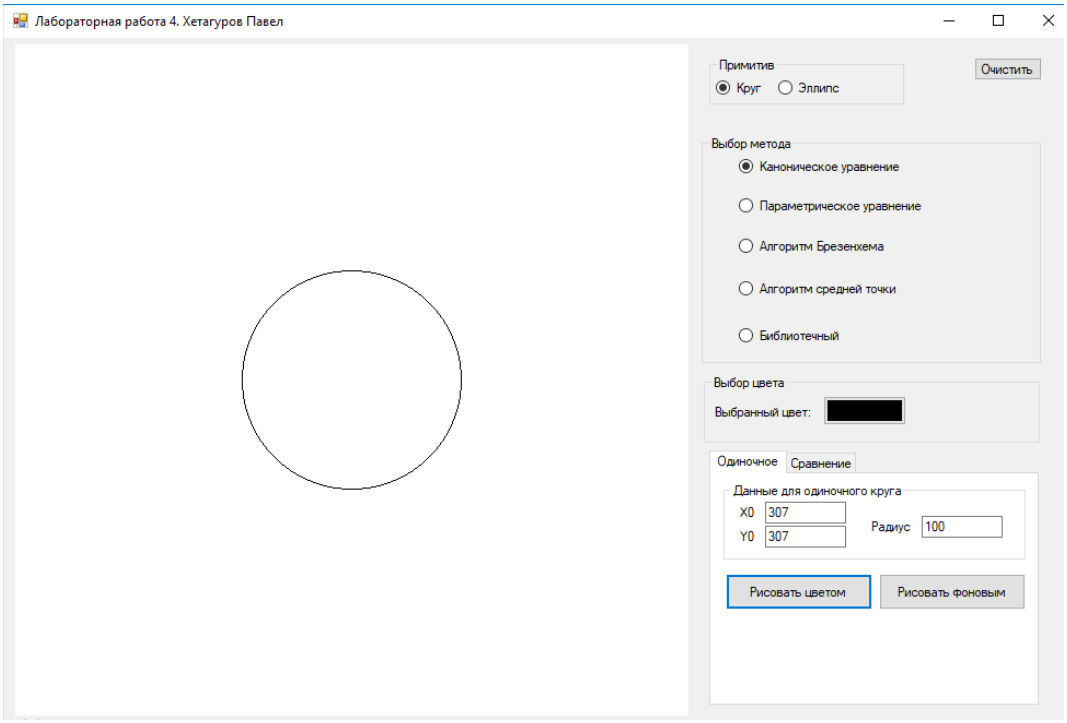
Алгоритмы должны работать быстро.

Алгоритмы генерации “в лоб” (на основе параметрического или канонического уравнения) требуют большое количество вычислений и не являются пошаговыми. Алгоритмы Брезенхема и средней точки являются пошаговыми и требуют меньшее кол-во вычислений.

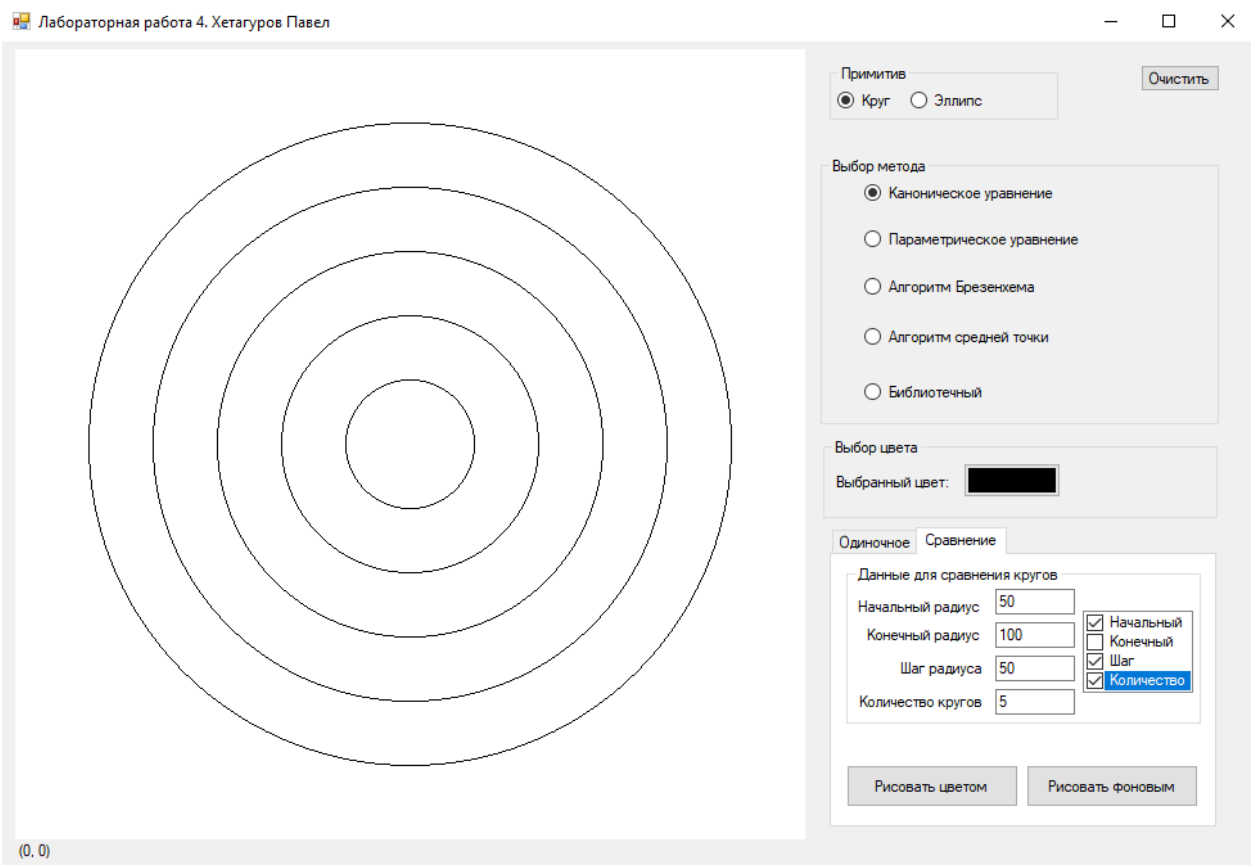
**Практическая часть**

**Интерфейс программы:**

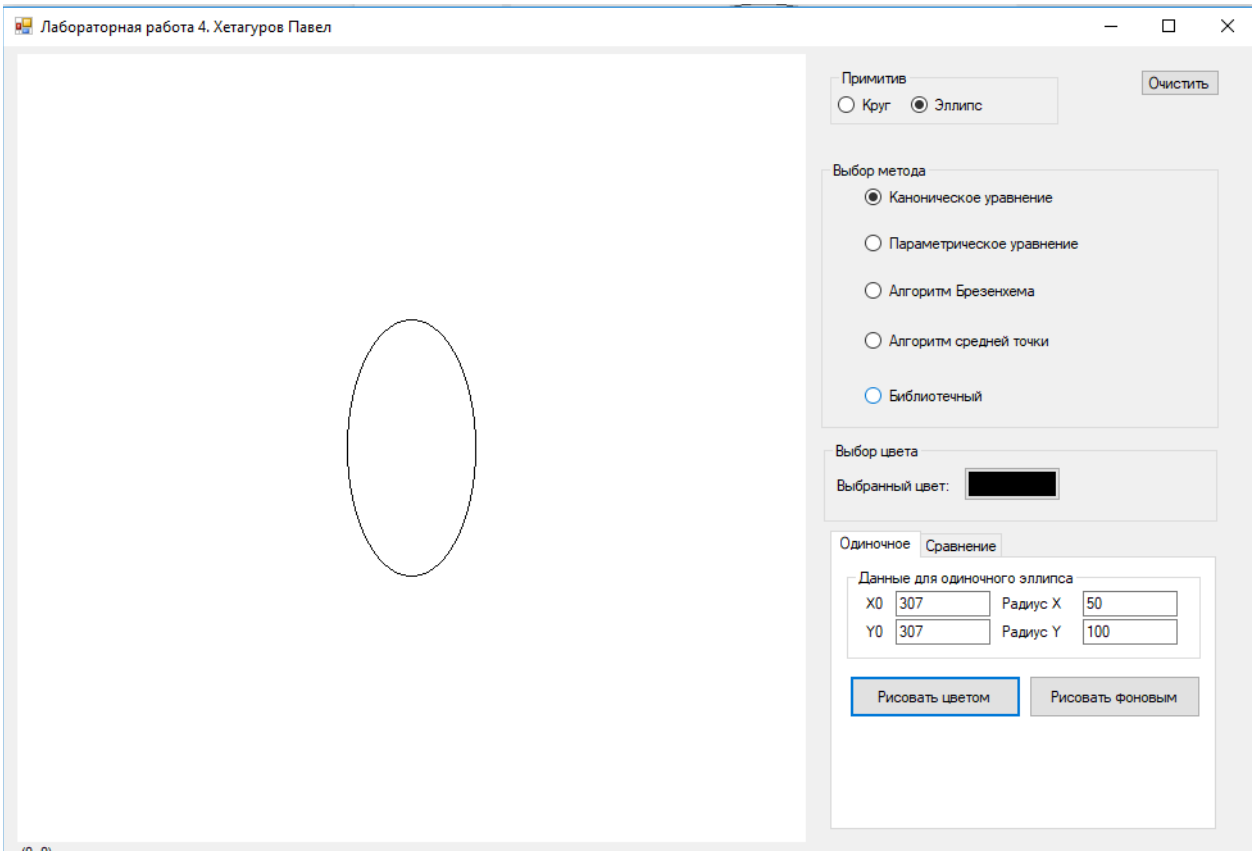
Одиночный круг:



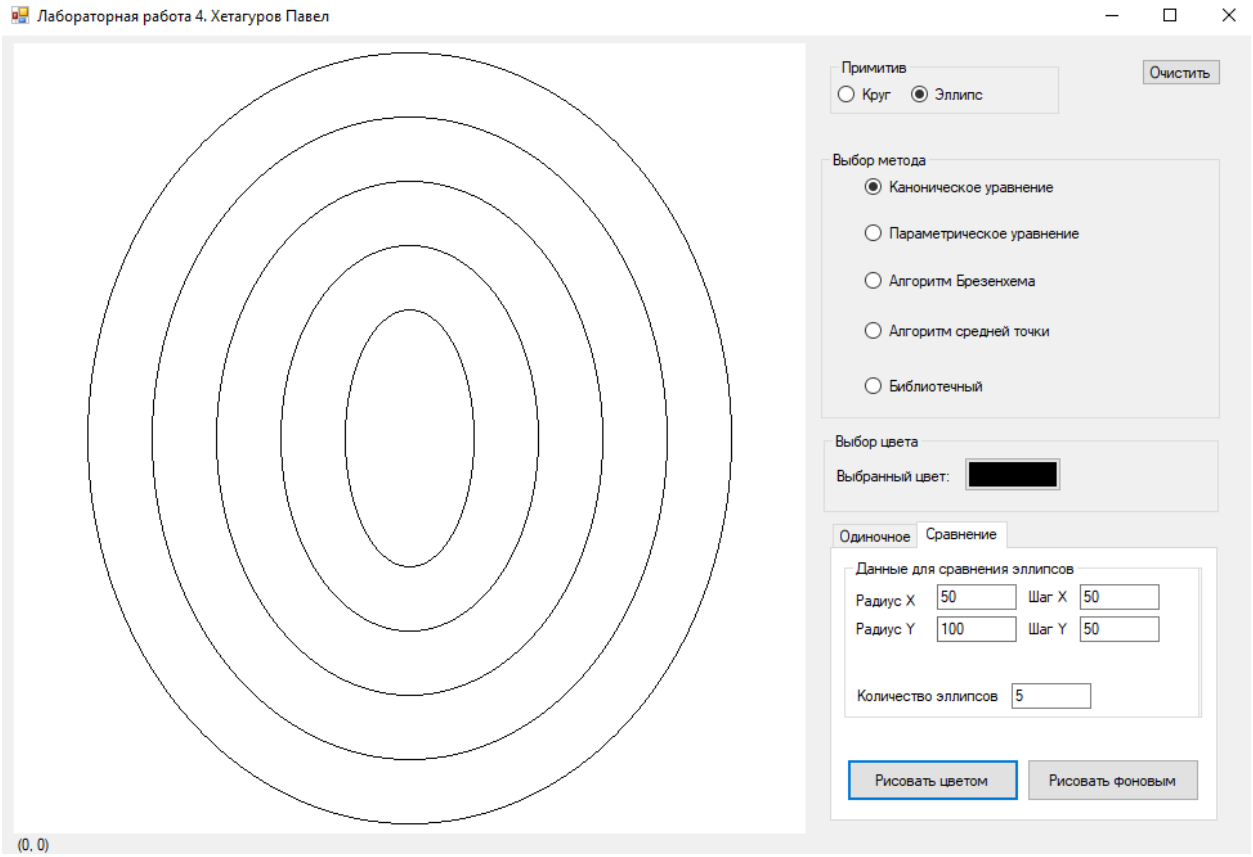

Сравнение:




Одиночный эллипс:




Сравнение эллипсов:




В дальнейшем, при демонстрации работы алгоритма буду прикладывать только скриншоты генерируемой окружности или эллипса.

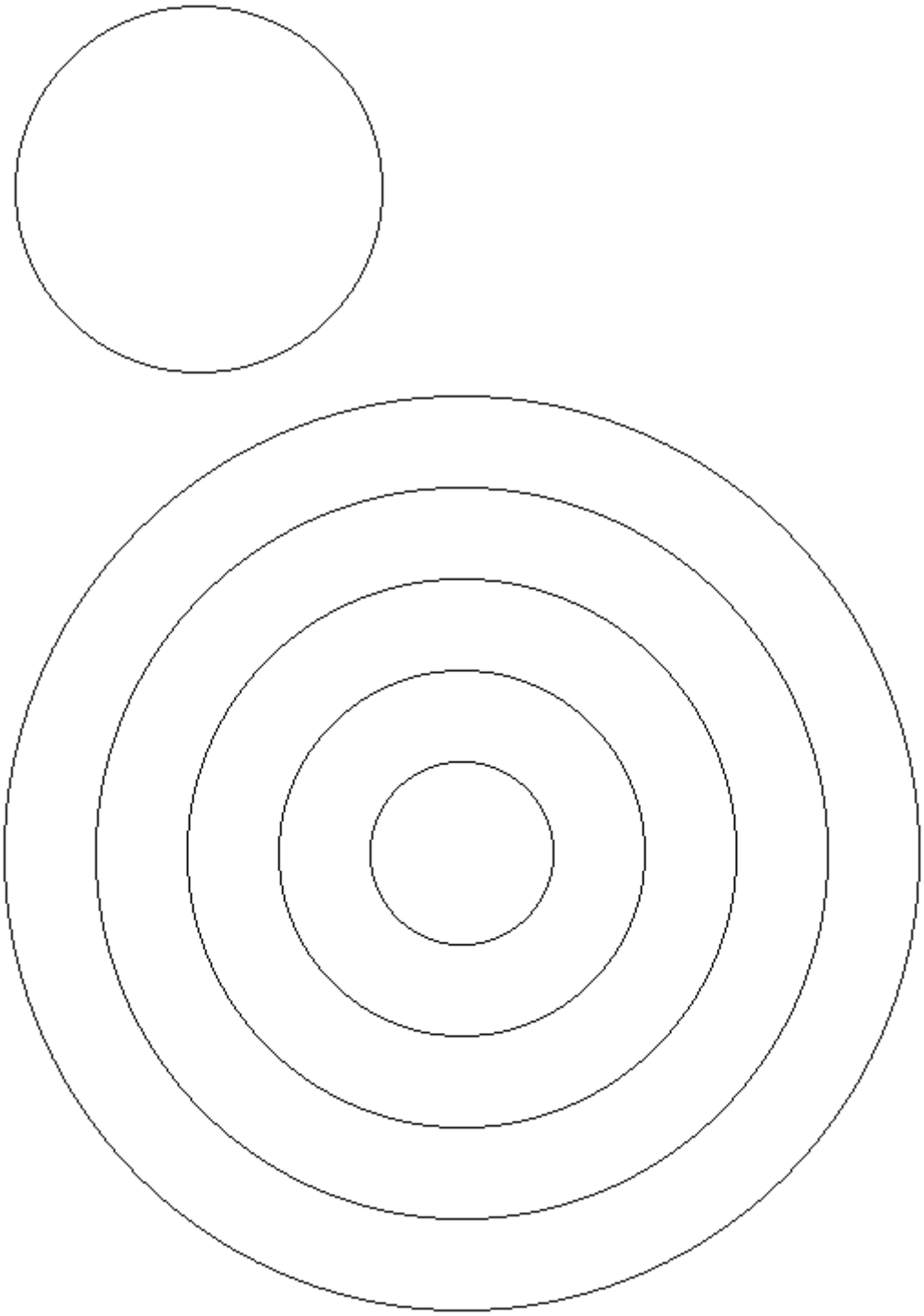
1. **Генерация окружностей**

1. **На основе канонического уравнения X^2+Y^2=R^2**

```
public static void DrawCircleCanonical(Bitmap workBitmap, Color workColor, Point
center, int radius)
    {
        int x = 0;
        int y = 1;

        int radiusSqr = radius * radius;

        for (x = 0; x < y; x++)
        {
            y = Convert.ToInt32(Math.Sqrt(radiusSqr - x * x));

            DrawSymmetric(workBitmap, workColor, center, x, y);
            DrawSymmetric(workBitmap, workColor, center, y, x);
        }
    }
```

Алгоритм медленный, так как происходит взятие корня и округление при каждой итерации

Результат выполнения:



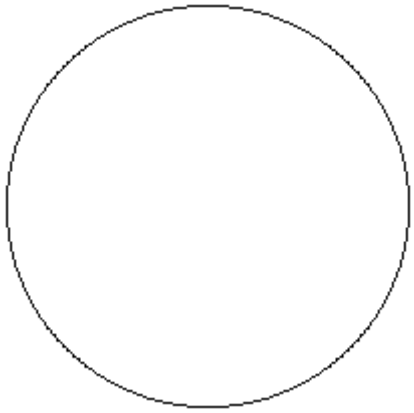
2. **Параметрическое уравнение X=Rcost, Y=Rsint:**

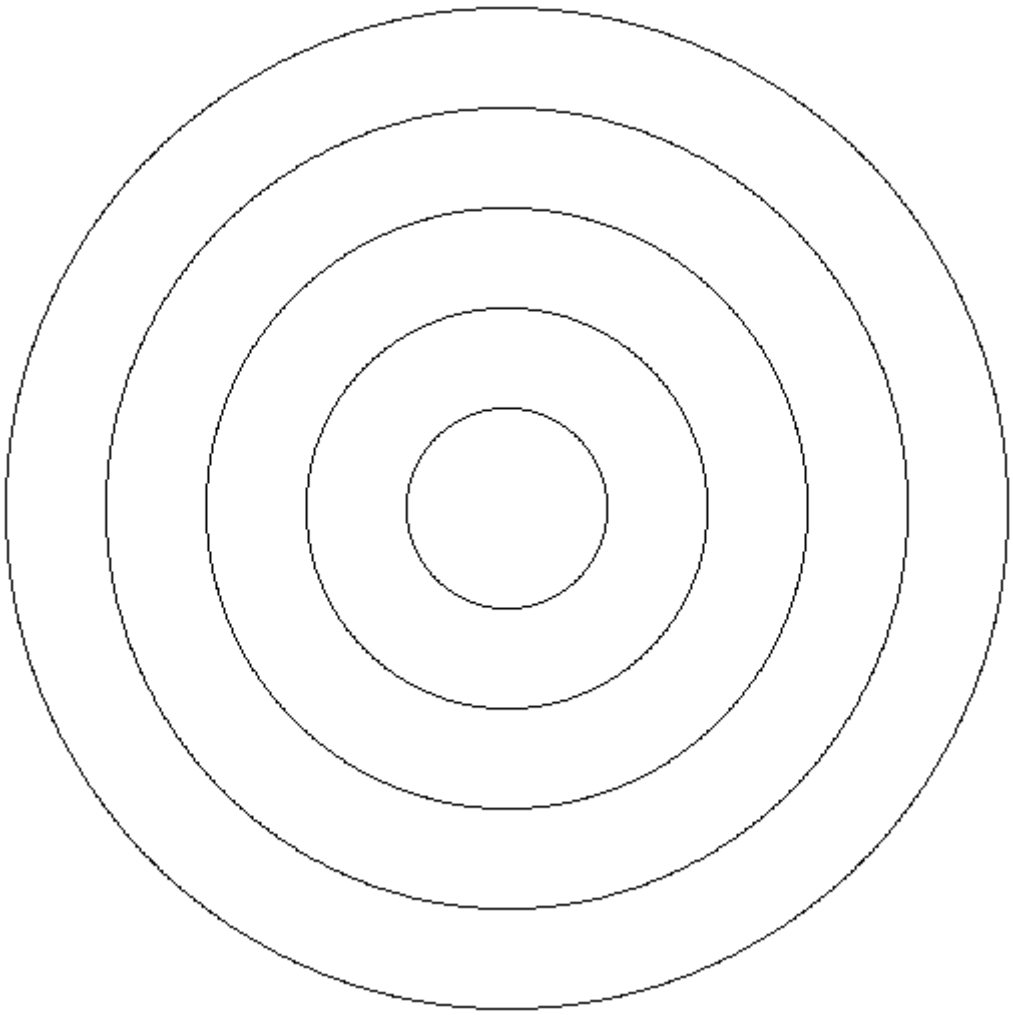
```
public static void DrawCircleParam(Bitmap workBitmap, Color workColor, Point
center, int radius)
```

```
    {
        double angle = 0;
        double delta = 1.0 / radius;

        while (angle < Math.PI / 4 + delta)
        {
            double x = (radius * Math.Cos(angle));
            double y = (radius * Math.Sin(angle));

            DrawSymmetric(workBitmap, workColor, center, (int)Math.Round(x),
(int)Math.Round(y));
            DrawSymmetric(workBitmap, workColor, center, (int)Math.Round(y),
(int)Math.Round(x));


            angle += delta;
        }
    }
```

Алгоритм неэффективен, так как требует вычисления косинуса и синуса, а также округления на каждом шаге.

Результат работы:





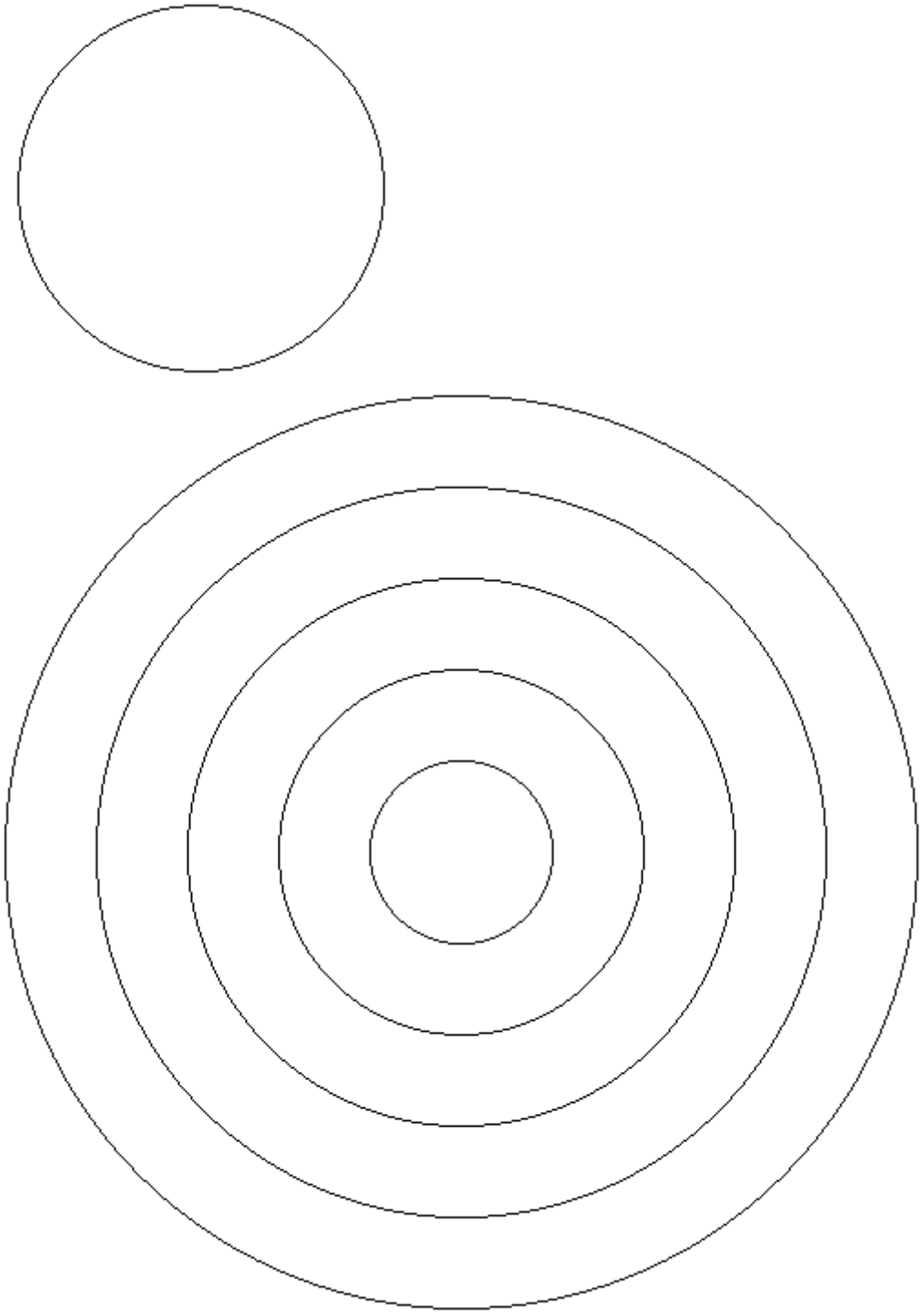
### 3. Алгоритм Брезенхема

```
        public static void DrawCircleBresenham(Bitmap workBitmap, Color
workColor, Point center, int radius)
{
    int x = 0;
    int y = radius;

    // error = (x + 1)^2 + (y -1)^2 - R^2 =
    int error = 2 * (1 - radius); // ошибка

    DrawSymmetric(workBitmap, workColor, center, x, y);
    while (x < y)
    {
        if (error == 0)   // Пиксель лежит на окружности
        {
            DiagonalStep(ref x, ref y, ref error);
        }
        else if (error < 0)   // Пиксель лежит внутри окружности
```

```
        {
            if (2 * error + 2 * y - 1 > 0) // Если расстояние до горизонтального пиксела больше, чем до диагонального, то выбираем диагональный. Иначе - горизонтальный

                DiagonalStep(ref x, ref y, ref error);
            else
                HorizontalStep(ref x, ref error);
        }
        else  // Пиксель лежит вне окружности
        {
            if (2 * error - 2 * x - 1 < 0) // Если расстояние до вертикального пиксела больше, чем до диагонального, то выбираем диагональный. Иначе - вертикальный
                DiagonalStep(ref x, ref y, ref error);
            else
                VerticalStep(ref y, ref error);
        }

        DrawSymmetric(workBitmap, workColor, center, x, y);
        DrawSymmetric(workBitmap, workColor, center, y, x);
    }
}

private static void DiagonalStep(ref int x, ref int y, ref int error)
{
    x++;
    y--;
    error += 2 * (x - y + 1);
}
private static void HorizontalStep(ref int x, ref int error)
{
    x++;
    error += 2 * x + 1;
}
private static void VerticalStep(ref int y, ref int error)
{
    y--;
    error += -2 * y + 1;
}
```

Данный алгоритм эффективен, так как не требует на каждом шаге округления, взятия корня или вычисления синуса или косинуса.

Результат работы:



4. **Алгоритм средней точки:**

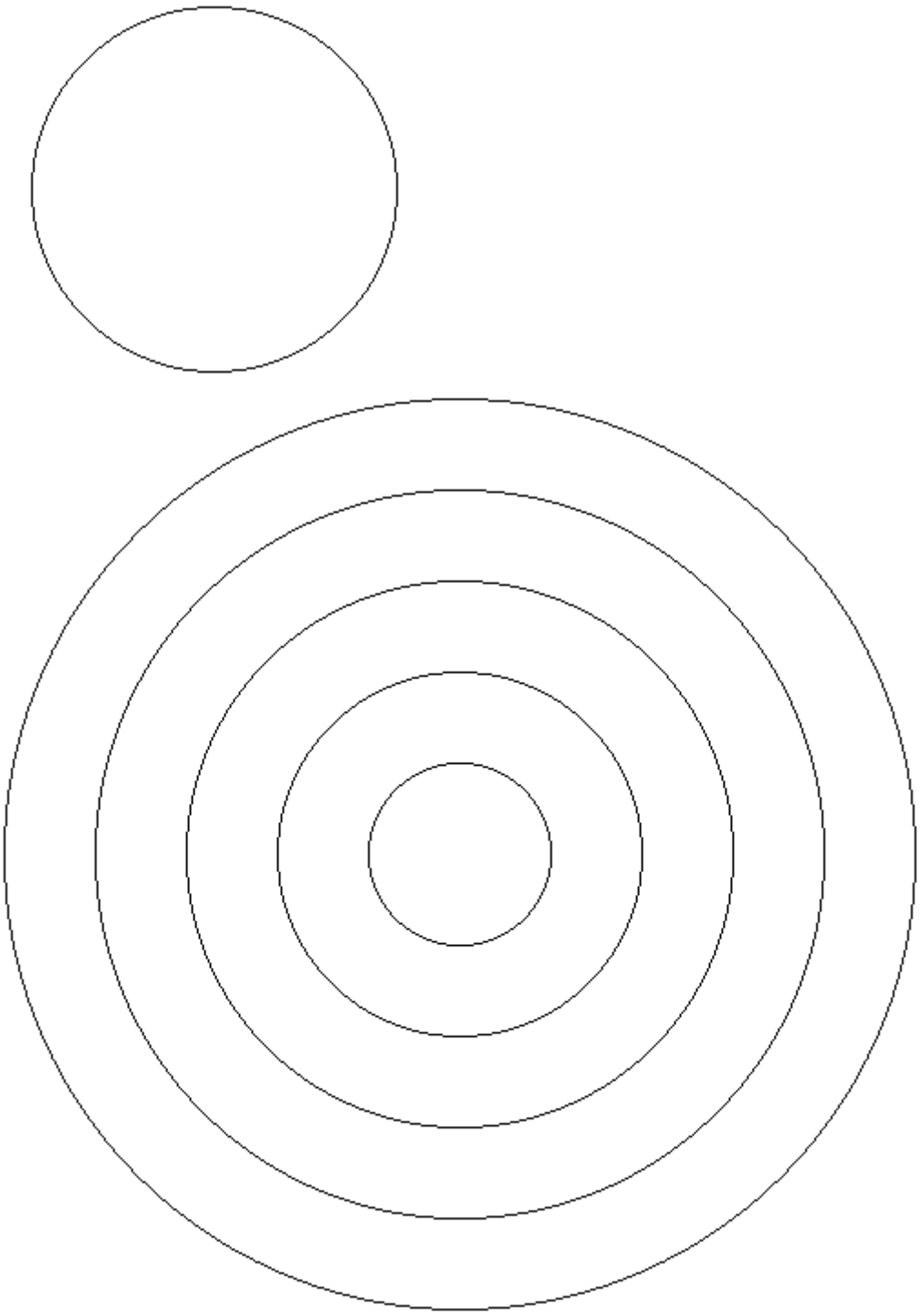
```
public static void DrawCircleMiddle(Bitmap workBitmap, Color workColor, Point
center, int radius)
    {
        int x = 0;
        int y = radius;
        // b^2 - a^2 * b + (a^2 / 4)... a = b = radius =>
```

```
        double p = 5 / 4.0 - radius;

        DrawSymmetric(workBitmap, workColor, center, x, y);
        DrawSymmetric(workBitmap, workColor, center, y, x);

        while (x < y)
        {
            x++;
            if (p < 0)
            {
                p += 2 * x + 1;
            }
            else
            {
                y--;
                p += 2 * (x - y) + 1;
            }

            DrawSymmetric(workBitmap, workColor, center, x, y);
            DrawSymmetric(workBitmap, workColor, center, y, x);
        }
    }
```

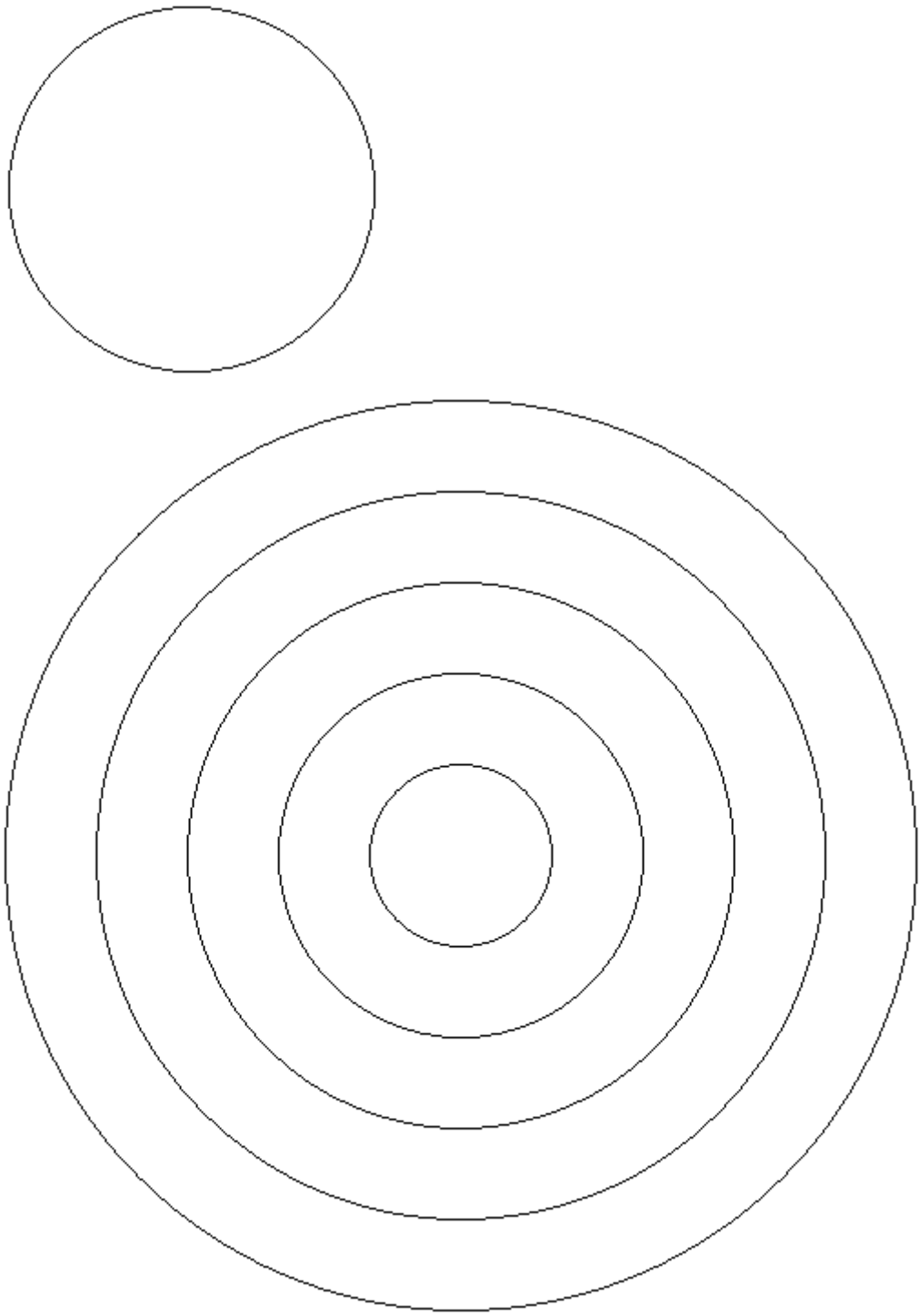
Требуется меньше всего действий. Результат совпадает с алгоритмом Брезенхема и алгоритмом на основе канонического уравнения.

Если округлить начальное значение ошибки до (1 - radius) , то результат, на проведенных мною тестах, не изменится, что позволяет перейти в полностью целые числа.

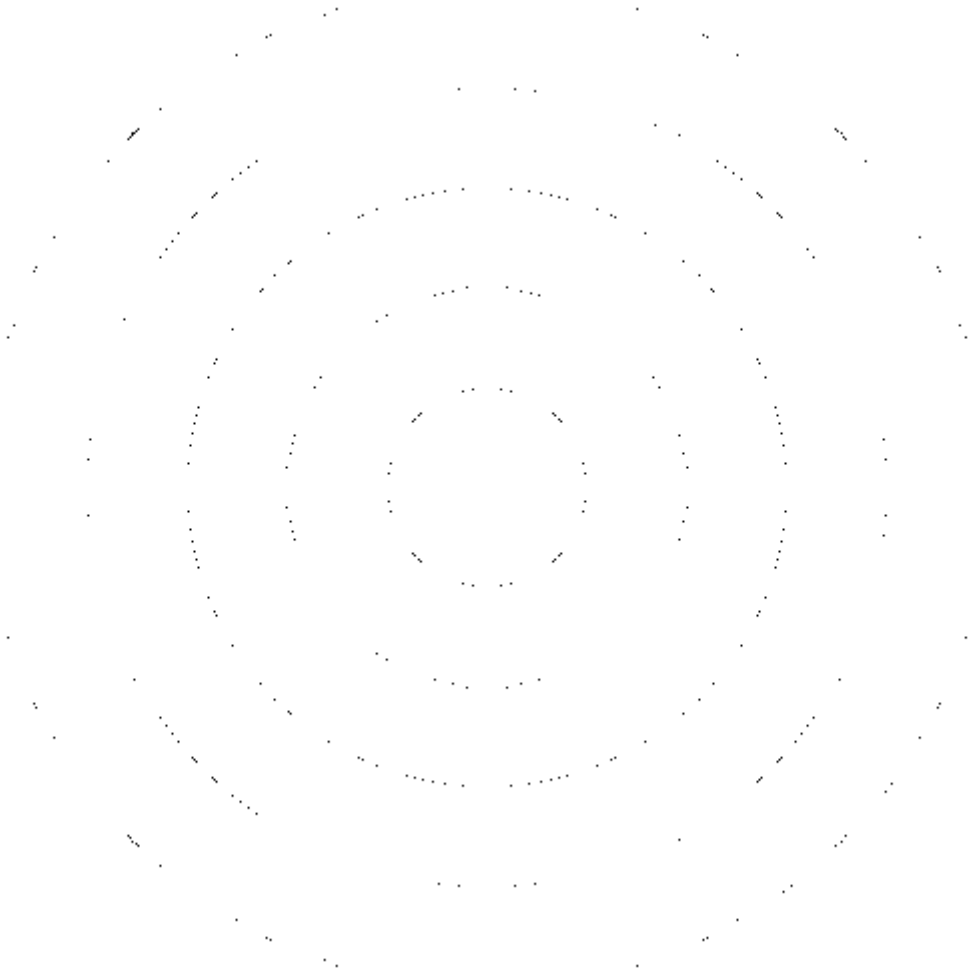
Результат работы:



**5. Библиотечный**



Наложение библиотечного и Брезенхема:



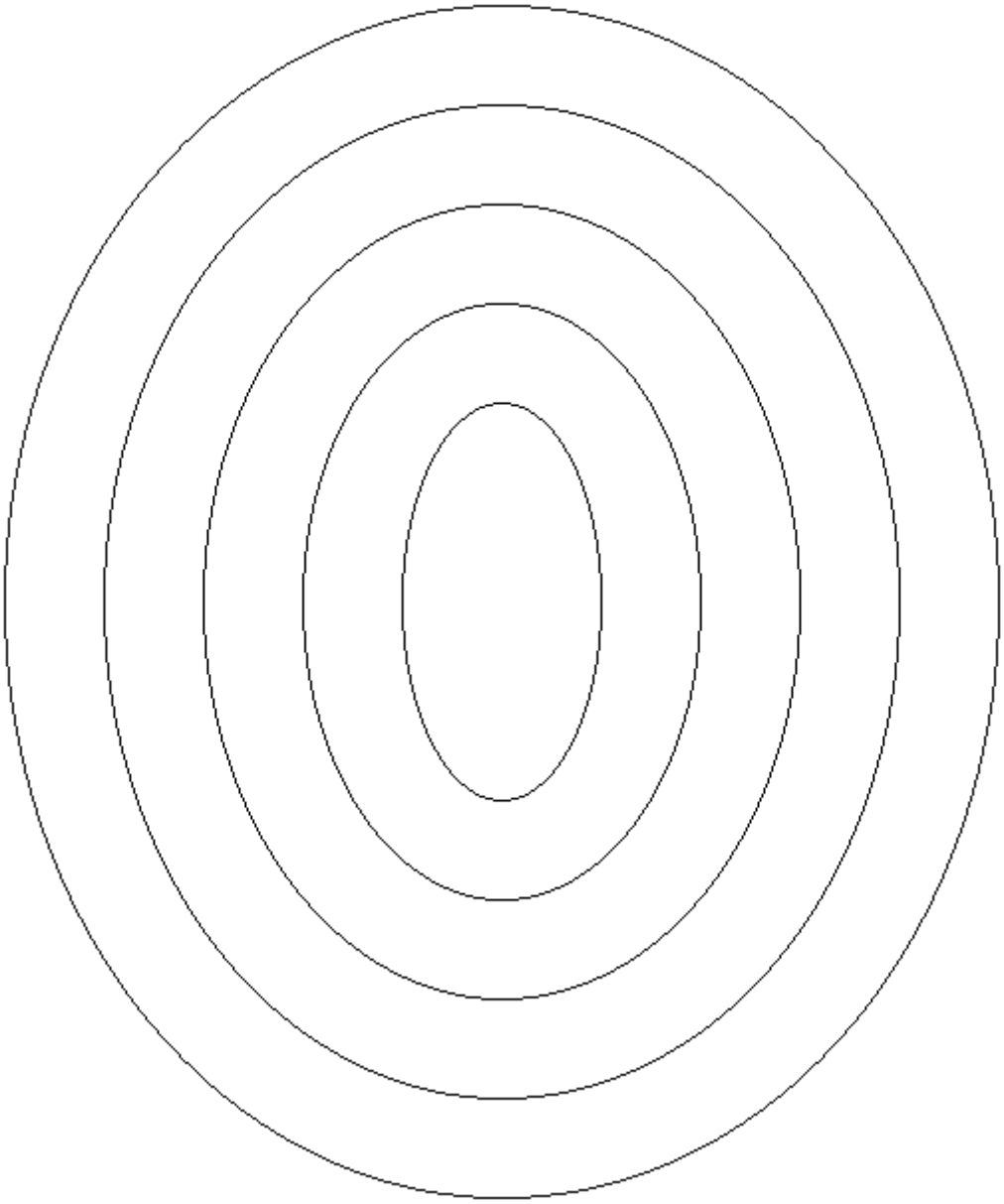
## 2. Генерация эллипсов

1. **На основе канонического уравнения X^2/a^2+Y^2/b^2=1**

```
public static void DrawEllipseCanonical(Bitmap workBitmap, Color
workColor, Point center, int radiusX, int radiusY)
    {
        int aSqr = radiusX * radiusX;
        int bSqr = radiusY * radiusY;

        // y' = -1.
        int oprimalX = (int)Math.Round(aSqr / Math.Sqrt(aSqr + bSqr));

        int x = 0;
        int y = 0;

        double m = (double)radiusY / radiusX; // b/a
        for (x = 0; x <= oprimalX; x++)
        {
```

```
            y = (int)Math.Round(Math.Sqrt(aSqr - x * x) * m);  // y = b/a
* sqrt(a^2 - x^2)
            DrawSymmetric(workBitmap, workColor, center, x, y);
        }

        // то же, но для Y
        int optimalY = (int)Math.Round(bSqr / Math.Sqrt(aSqr + bSqr));
        m = 1 / m;

        for (y = 0; y <= optimalY; y++)
        {
            x = (int)Math.Round(Math.Sqrt(bSqr - y * y) * m);
            DrawSymmetric(workBitmap, workColor, center, x, y);
        }
    }
```

Алгоритм требует вычислений корня и округления на каждом шаге. Неэффективно.

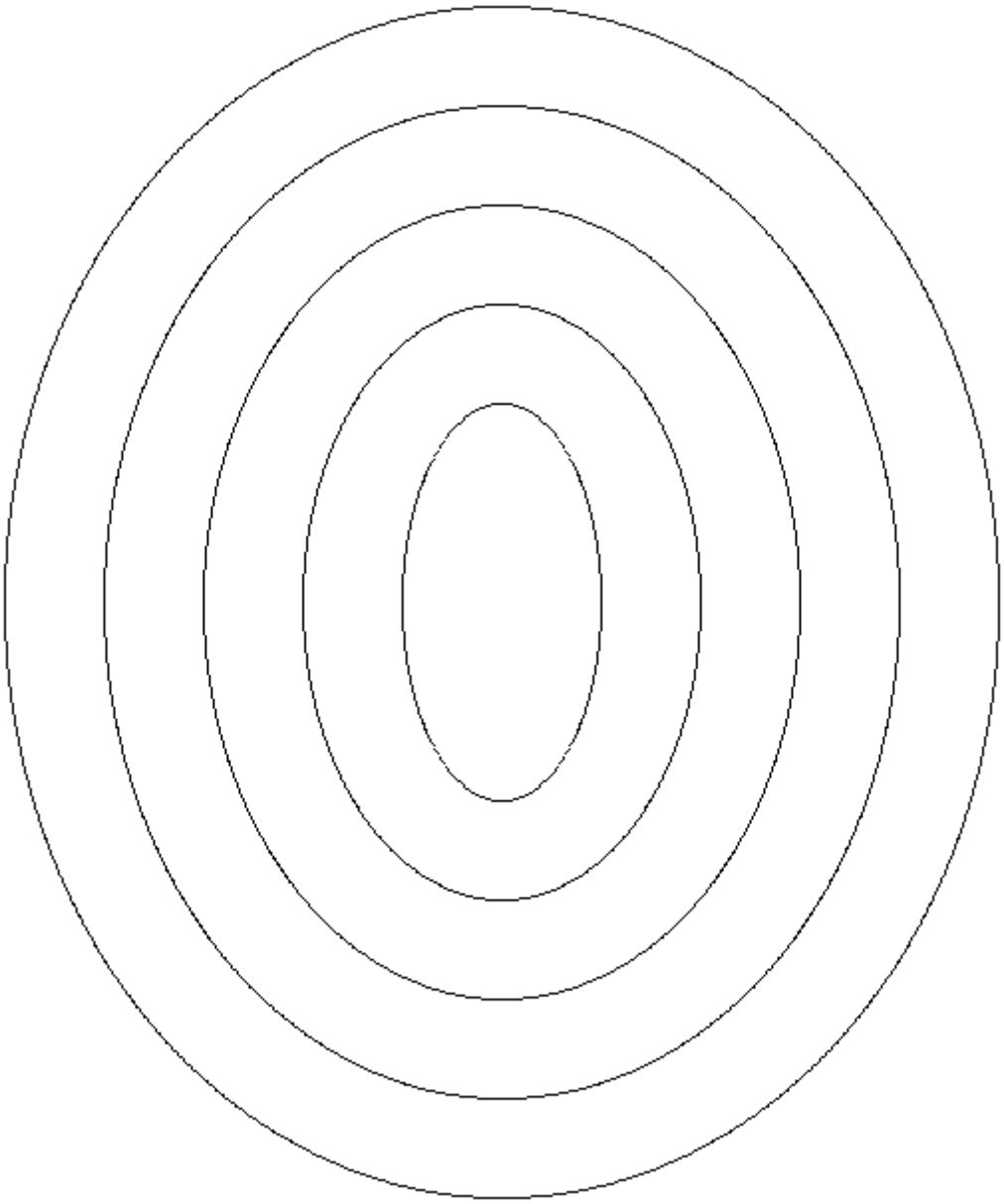


2. **На основе Параметрического уравнения X=acost, Y=bsint**

```
public static void DrawEllipseParam(Bitmap workBitmap, Color workColor,
Point center, int radiusX, int radiusY)
    {
        double angle = Math.PI / 2;

        int x = 0, y = radiusY;

        double step = 1 / (double)radiusX;
        while (angle > Math.PI / 4)
        {
            x = Convert.ToInt32(radiusX * Math.Cos(angle));
            y = Convert.ToInt32(radiusY * Math.Sin(angle));

            DrawSymmetric(workBitmap, workColor, center, x, y);
```

```
            angle -= step;
        }


        step = 1 / (double)radiusY;
        while (angle > -1e9)
        {
            x = Convert.ToInt32(radiusX * Math.Cos(angle));
            y = Convert.ToInt32(radiusY * Math.Sin(angle));

            DrawSymmetric(workBitmap, workColor, center, x, y);

            angle -= step;
        }
    }
```

Такие же минусы, как и у алгоритма на основе параметрического уравнения для круга (неэффективность из-за округления и вычисления косинуса и синуса на каждой итерации)



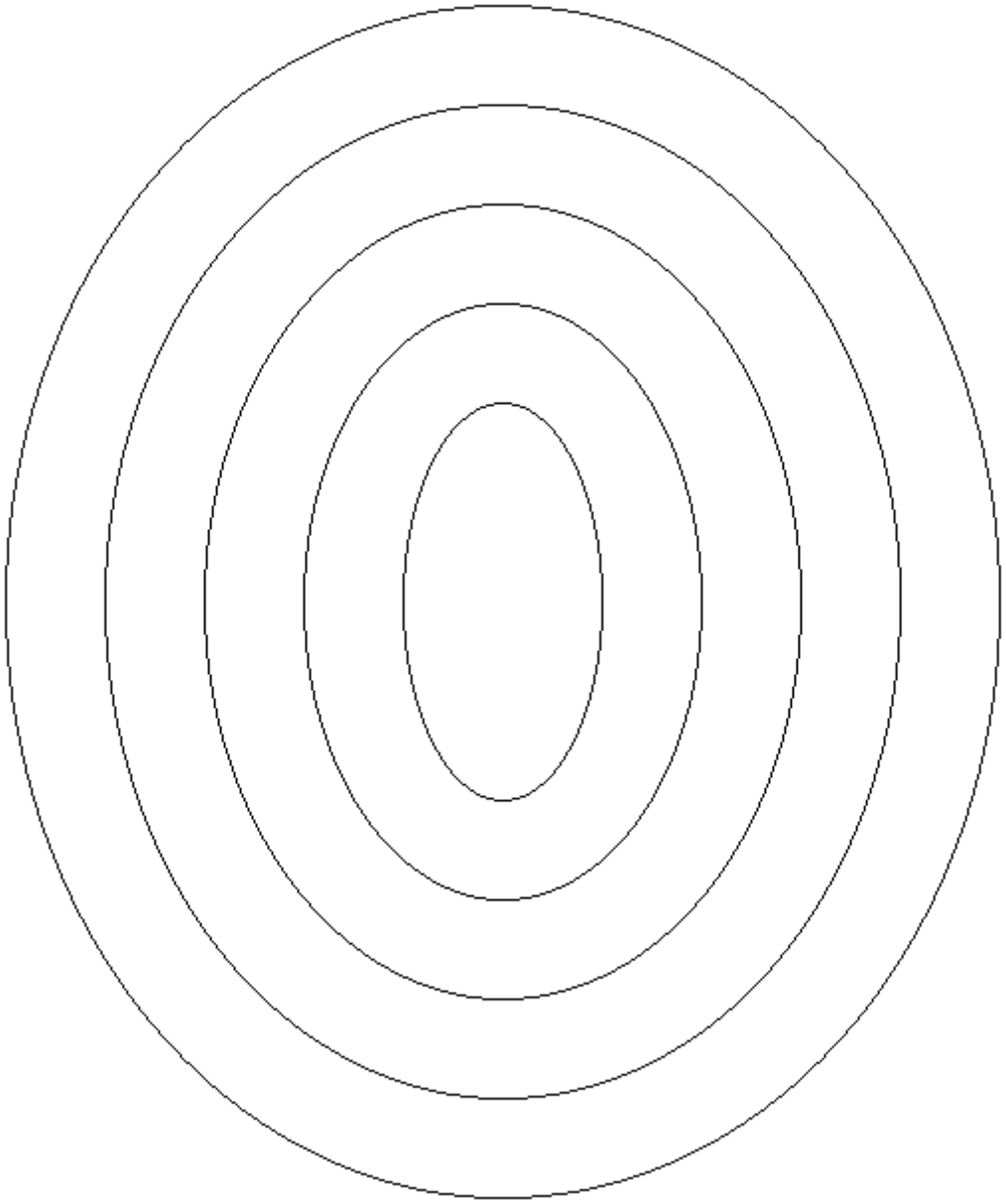
### 3. Алгоритм Брезенхема (модифицированный)

```
public static void DrawEllipseBresenham(Bitmap workBitmap, Color
workColor, Point center, int radiusX, int radiusY)
    {
        int x = 0;
        int y = radiusY;
        long aSqr = radiusX * radiusX;
        long bSqr = y * y;

        long aSqrTwice = aSqr * 2;
        long bSqrTwice = bSqr * 2;

        //error = b^2 * (x+1)^2 + a^2 * (y-1)^2-a^2 * b^2=
        long error = bSqr - aSqrTwice * radiusY + aSqr;

        long dx = 0;
```

```
        long dy = aSqrTwice * y;

        DrawSymmetric(workBitmap, workColor, center, x, y);

        while (y > 0)
        {
            if (error < 0)
            {
                if (2 * error + dy - aSqr > 0)
                {
                    DiagonalStep(ref x, ref y, ref error, ref aSqr, ref
aSqrTwice, ref bSqr, ref bSqrTwice, ref dx, ref dy);
                }
                else
                {
                    HorizontalStep(ref x, ref y, ref error, ref bSqr, ref
bSqrTwice, ref dx);
                }
            }
            else if (error > 0)
            {
                if (2 * error - dx - bSqr > 0)
                {
                    VerticalStep(ref x, ref y, ref error, ref aSqr, ref
aSqrTwice, ref dy);
                }
                else
                {
                    DiagonalStep(ref x, ref y, ref error, ref aSqr, ref
aSqrTwice, ref bSqr, ref bSqrTwice, ref dx, ref dy);
                }
            }
            else
            {
                DiagonalStep(ref x, ref y, ref error, ref aSqr, ref
aSqrTwice, ref bSqr, ref bSqrTwice, ref dx, ref dy);
            }

            DrawSymmetric(workBitmap, workColor, center, x, y);
        }
    }
```

```
private static void DiagonalStep(ref int x, ref int y, ref long error, ref long aSqr, ref long aSqrTwice, ref long bSqr, ref long bSqrTwice, ref long dx, ref long dy)
    {
        x++;
        y--;
        dx += bSqrTwice;
        dy -= aSqrTwice;
        error += dx - dy + aSqr + bSqr;
    }

private static void HorizontalStep(ref int x, ref int y, ref long error, ref long bSqr, ref long bSqrTwice, ref long dx)
    {
        x++;
        dx += bSqrTwice;
        error += dx + bSqr;
    }

private static void VerticalStep(ref int x, ref int y, ref long error, ref long aSqr, ref long aSqrTwice, ref long dy)
    {
        y--;
        dy -= aSqrTwice;
        error += -dy + aSqr;
    }
```

Так как алгоритм Брезенхема делает выбор сразу между тремя альтернативами, то корректировка ошибки при dY/dX = -1 не требуется (в отличии от алгоритма средней точки)

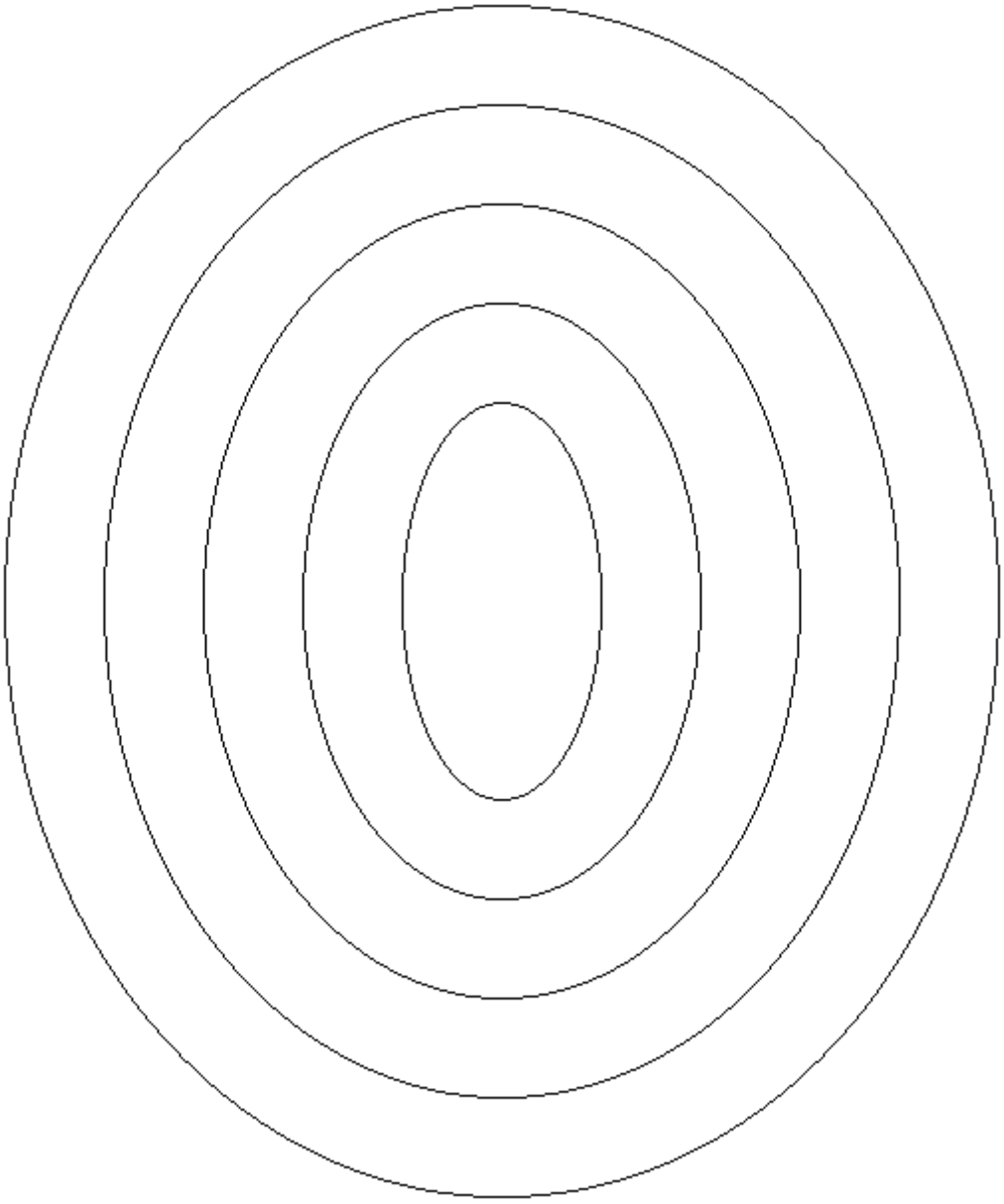


4. **Алгоритм средней точки**

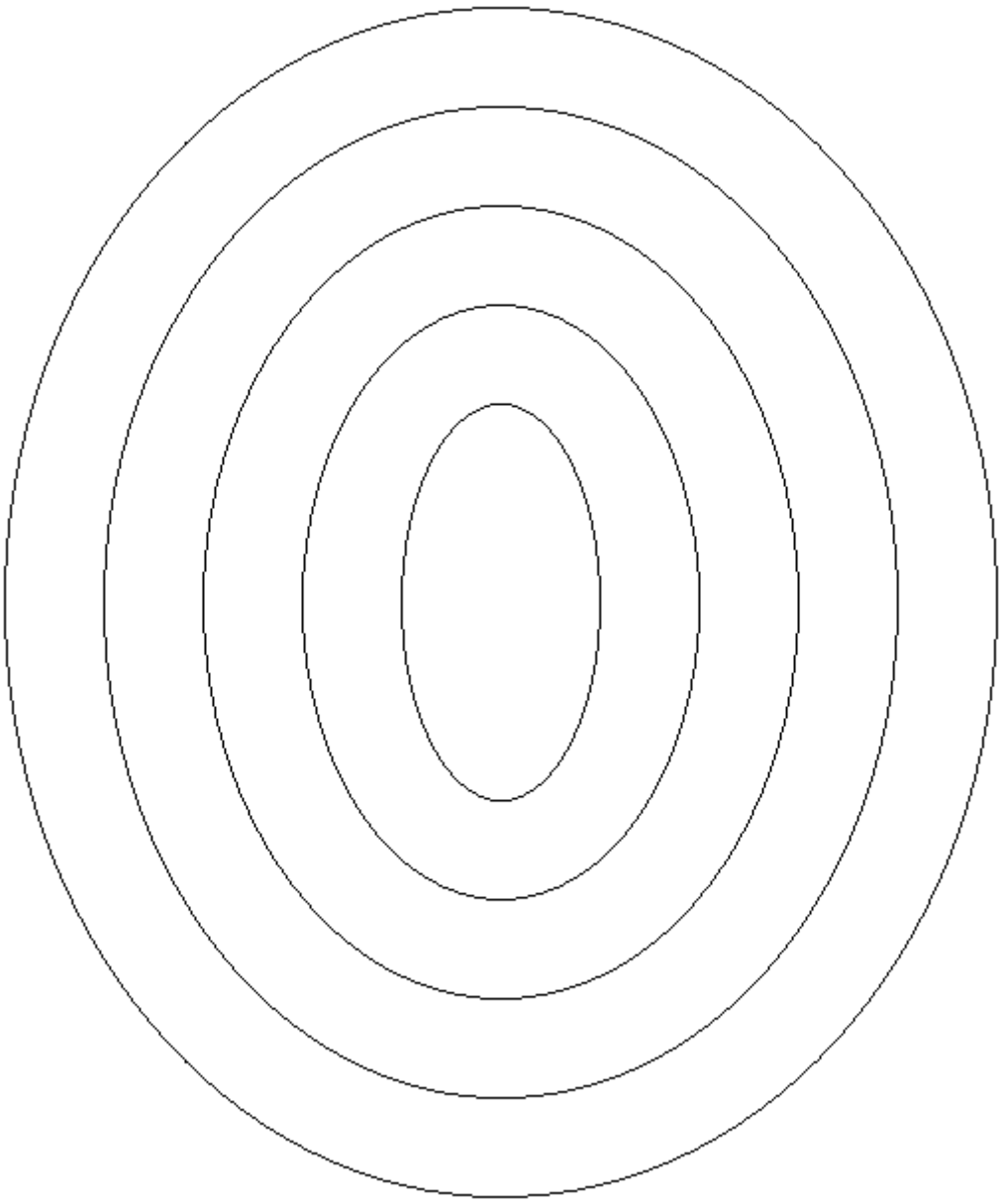
```
public static void DrawEllipseMiddle(Bitmap workBitmap, Color workColor,
Point center, int radiusX, int radiusY)
    {
        int x = 0;
        int y = radiusY;
        long aSqr = radiusX * radiusX;
        long bSqr = y * y;

        long aSqrTwice = aSqr * 2;
        long bSqrTwice = bSqr * 2;

        // Начальная ошибка
        long p = (long)Math.Round(bSqr - aSqr * radiusY + (aSqr * 0.25));

        DrawSymmetric(workBitmap, workColor, center, x, y);
```

```
        long dx = 0;
        long dy = aSqrTwice * y;
        while (dx < dy)
        {
            x++;
            dx += bSqrTwice;
            if (p < 0)
            {
                p += dx + bSqr;
            }
            else
            {
                y--;
                dy -= aSqrTwice;
                p += dx - dy + bSqr;
            }

            DrawSymmetric(workBitmap, workColor, center, x, y);
        }

        // Корректировка ошибки (смена выбора с вертикальных на горизонт.)
        p += (long)Math.Round((aSqr - bSqr) * 0.75) - (dx + dy) / 2;

        while (y > 0)
        {
            y--;
            dy -= aSqrTwice;
            if (p > 0)
            {
                p += -dy + aSqr;
            }
            else
            {
                x++;
                dx += bSqrTwice;
                p += dx - dy + aSqr;
            }
            DrawSymmetric(workBitmap, workColor, center, x, y);
        }
        /**/
    }
```

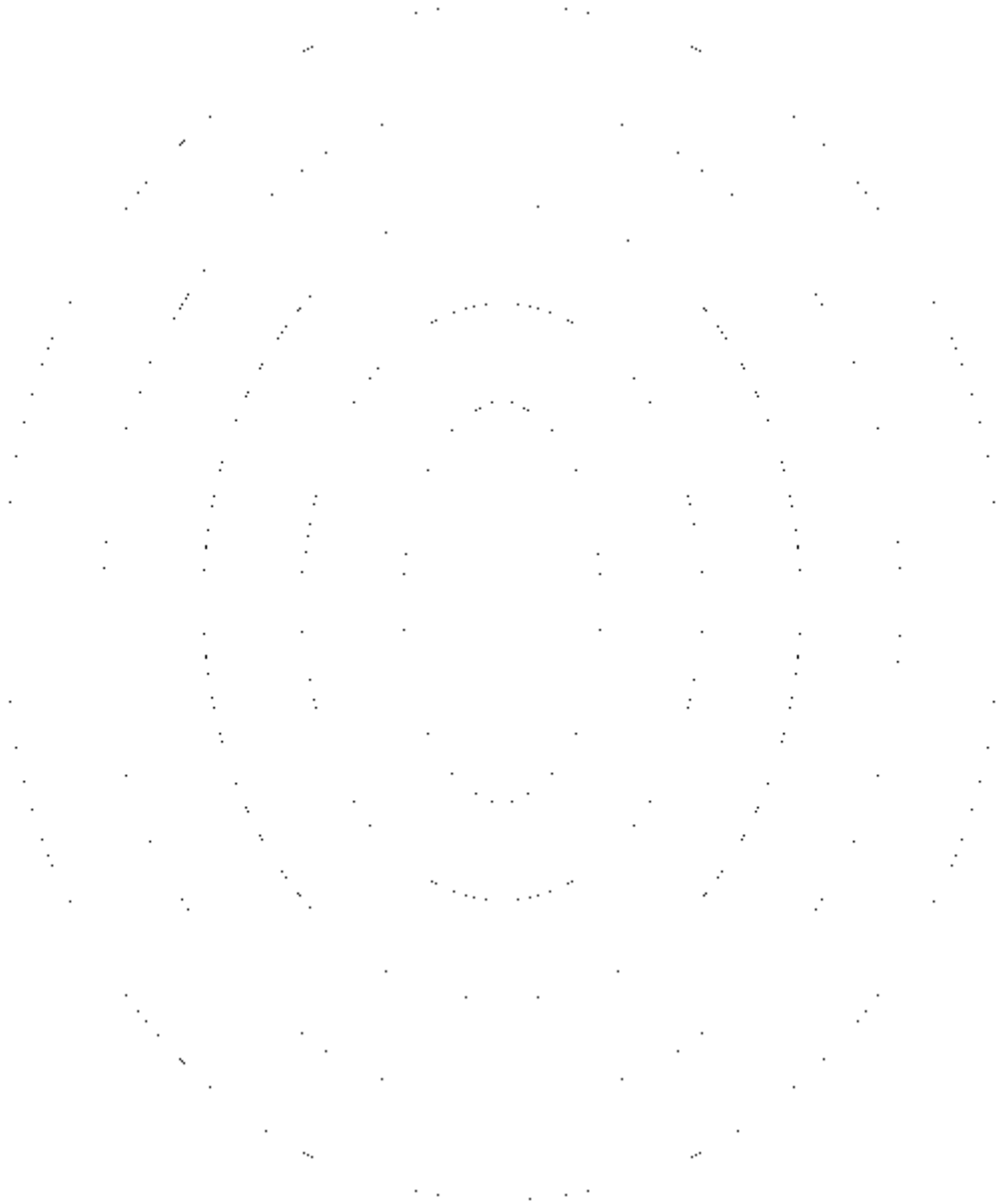
Требует больше памяти для хранения промежуточных переменных. В 32х битных целых при радиусах больше 200 у меня получалось переполнение, пришлось перейти на 64х битные целые.



**5. Библиотечная функция**



Наложение библиотечной и модифицированного Брезенхема:



Библиотечный алгоритм, скорее всего, реализован на основе алгоритма Брезенхема