|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

|  |  |
| --- | --- |
| **Тема**  Реализация и исследование алгоритмов  построения окружностей и эллипсов  **Студент** Козлов М. А.  **Группа ИУ 7-45**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Научиться выполнять построение окружностей и эллипсов различными алгоритмами и проанализировать их.

**Техническое задание:**

1. Построение отдельных окружностей и эллипсов и сравнение их визуальных характеристик
   1. Каноническое уравнение
   2. Параметрическое уравнение
   3. Алгоритм Брезенхэма
   4. Алгоритм средней точки
   5. Библиотечный алгоритм
2. Исследование визуальных характеристик окружностей и эллипсов
3. Сравнение временных характеристик (результат оформить в виде графика)

**Теоретический материал:**

Все алгоритмы имеют пошаговый характер – на очередном шаге высвечиваем пиксель, и производим вычисления, используемые в следующем шаге.

**Алгоритм с использованием параметрического уравнения окружности.**

void Parametric::DrawCicrle(QImage& canvas, int x, int y, int r, unsigned int color)

{

float step = 1 / (float)r;

int i, j;

for (float t = 0; t < M\_PI\_4; t += step)

{

i = roundf(r \* cosf(t));

j = roundf(r \* sinf(t));

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

canvas.setPixel(x + j, y + i, color);

canvas.setPixel(x + j, y - i, color);

canvas.setPixel(x - j, y + i, color);

canvas.setPixel(x - j, y - i, color);

}

// t = pi/4

i = roundf(r \* M\_SQRT1\_2);

j = i;

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

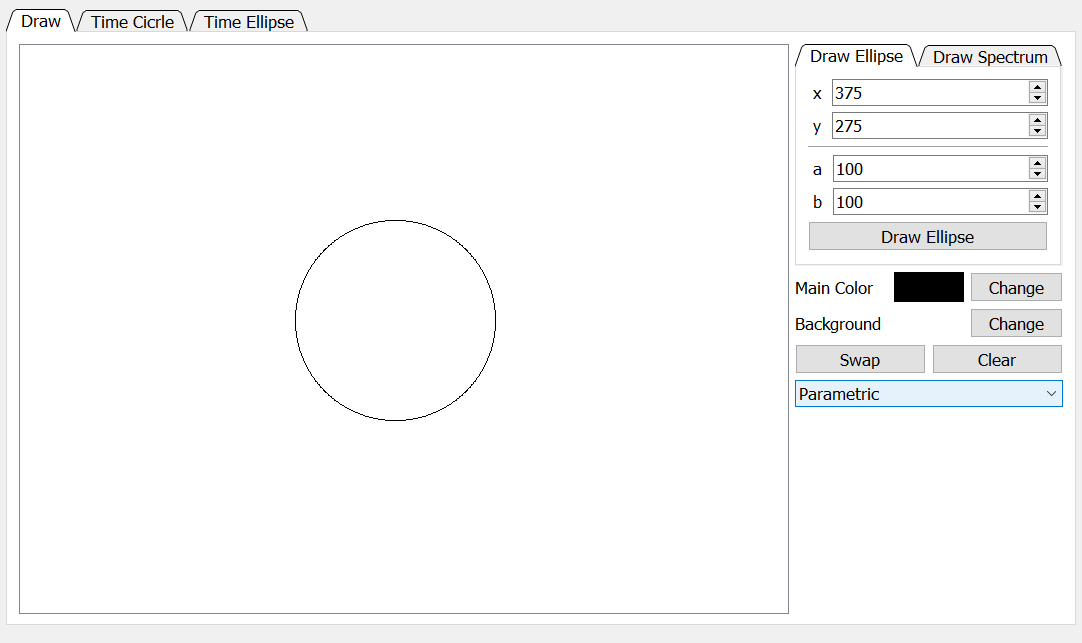
canvas.setPixel(x - i, y + j, color);

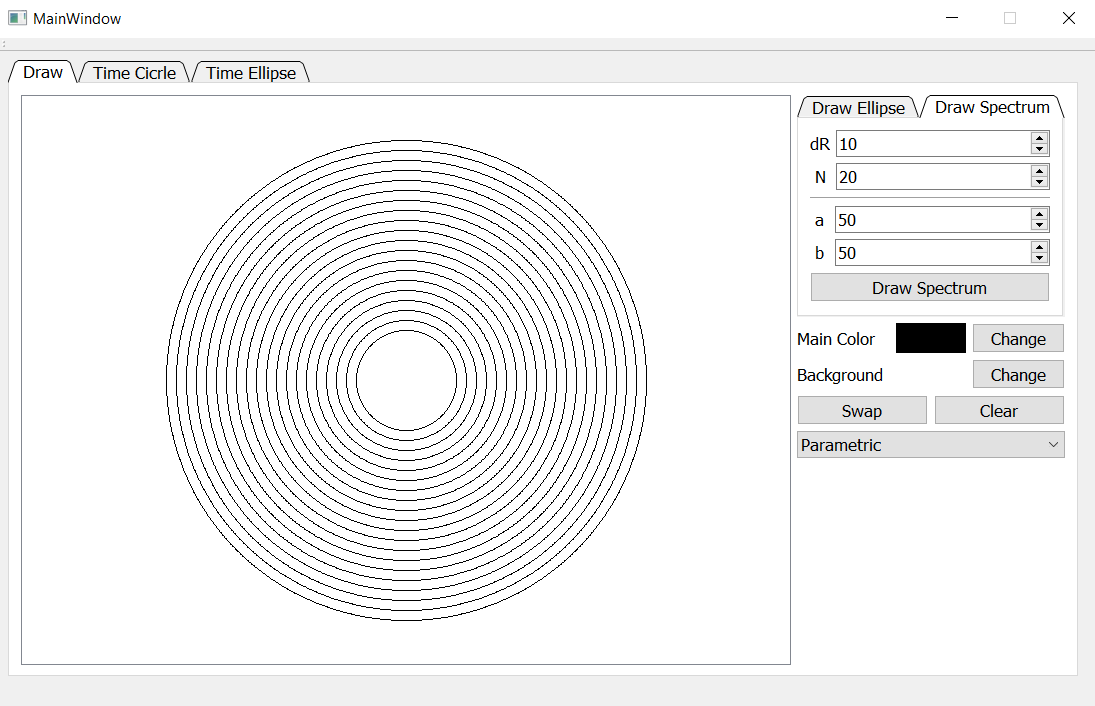
canvas.setPixel(x - i, y - j, color);

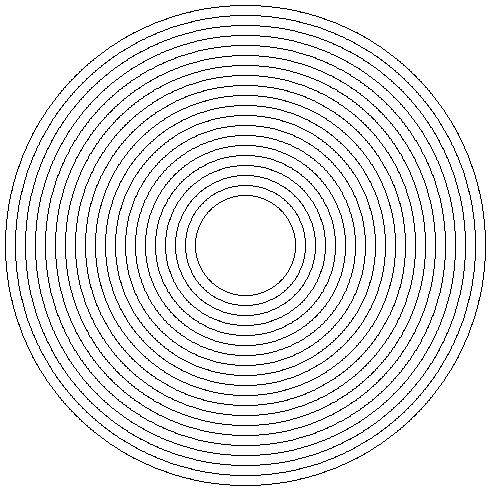
}

Проблемы данного алгоритма

1. Вычисление sin и cos (долгая и не точная операция)
2. Округление



\*работа алгоритмов далее будет показываться на спектре



**Алгоритм с использованием канонического уравнения окружности.**

void Canonical::DrawCicrle(QImage& canvas, int x, int y, int r, unsigned int color)

{

int r2 = r \* r;

int i = 0, j = r;

while (i < j)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

canvas.setPixel(x + j, y + i, color);

canvas.setPixel(x + j, y - i, color);

canvas.setPixel(x - j, y + i, color);

canvas.setPixel(x - j, y - i, color);

i++;

j = roundf(sqrtf(r2 - i \* i));

}

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

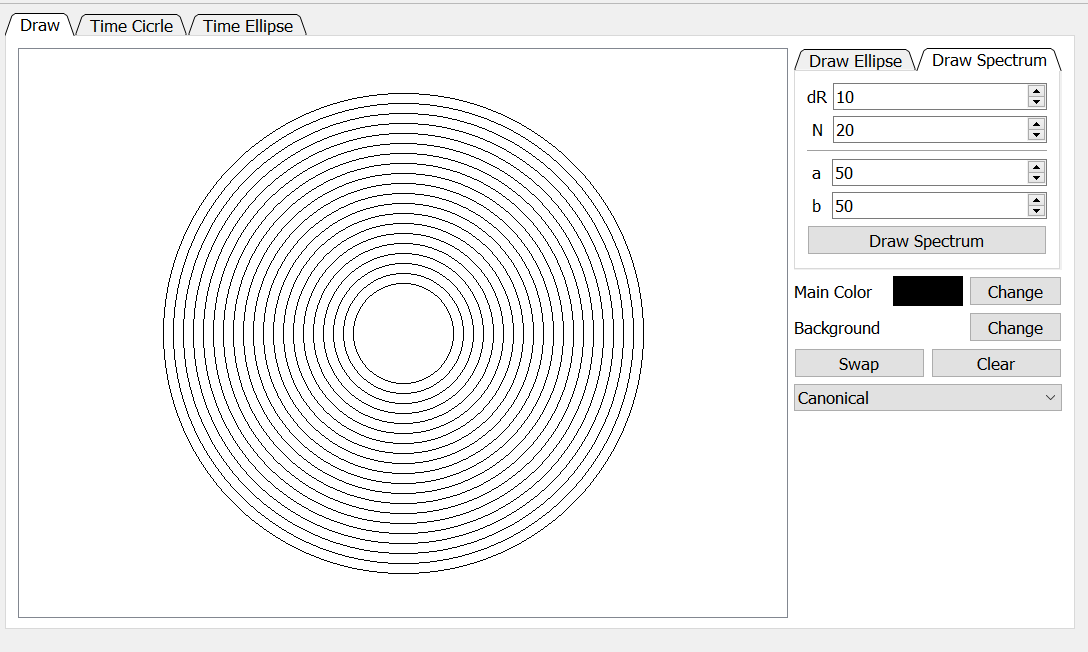
canvas.setPixel(x - i, y + j, color);

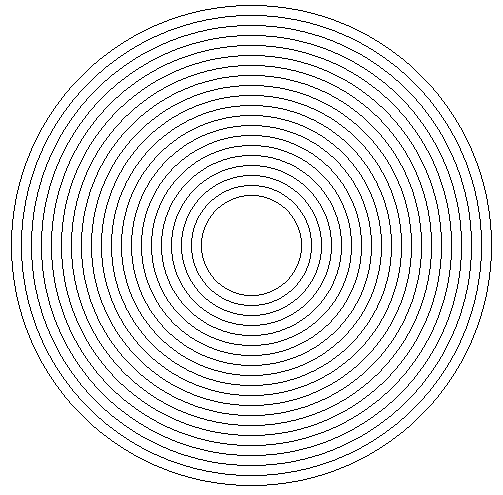
canvas.setPixel(x - i, y - j, color);

}

Проблемы данного алгоритма

1. Округление
2. Вычисление квадрата и квадратного корня





**Алгоритм Брезенхэма для окружности**

inline void DiagonalStepCicrle(int& x, int& y, int& d)

{

x++;

y--;

d += 2 \* (x - y + 1);

}

inline void HorizontalStepCicrle(int& x, int& y, int& d)

{

x++;

d += 2 \* x + 1;

}

inline void VerticalStepCicrle(int& x, int& y, int& d)

{

y--;

d += -2 \* y + 1;

}

void Breshenham::DrawCicrle(QImage& canvas, int x, int y, int r, unsigned int color)

{

int r2 = r \* r;

int d = 2 \* (1 - r); // первоначальная ошибка

int i = 0, j = r;

while (i < j)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

canvas.setPixel(x + j, y + i, color);

canvas.setPixel(x + j, y - i, color);

canvas.setPixel(x - j, y + i, color);

canvas.setPixel(x - j, y - i, color);

if (d == 0) // диагональная точка лежит на окружности

DiagonalStepCicrle(i, j, d);

else if (d < 0) // диагональная точка внутри окружности

{

// раст. до гор пикселя раст. до диаг. пикселя

// |(xi+1)^2 + yi^2 - r^2| - |(xi+1)^2 + (yi-1)^2 - r^2|

int delta1 = 2 \* (d + j) - 1;

if (delta1 > 0)

DiagonalStepCicrle(i, j, d);

else

HorizontalStepCicrle(i, j, d);

}

else // диагональная точка вне окружности

{

// раст. до диаг. пикселя раст. до верт пикселя

// |(xi+1)^2 + (yi-1)^2 - r^2| - |xi^2 + (yi-1)^2 - r^2|

int delta2 = 2 \* (d - i) - 1;

if (delta2 < 0)

DiagonalStepCicrle(i, j, d);

else

VerticalStepCicrle(i, j, d);

}

}

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

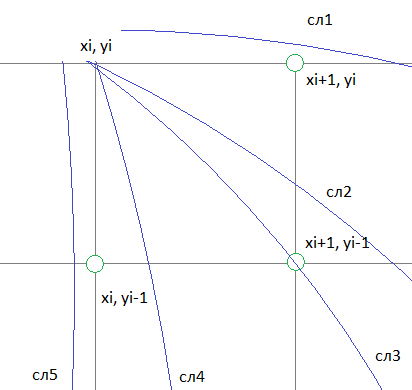
canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

}

Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

Окружность – симметричная фигура, можно построить одну восьмую и отражать.



– в первой четверти монотонно убывающая.

На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

Критерий

Если

Возможны случаи 1 и 2.

Для того чтобы решить какой пиксель высвечивать введём

Случай 1: первый модуль <=0, второй <0

Случай 2: первый модуль >=0, второй <0

Возможны случаи 4 и 5.

Для того чтобы решить какой пиксель высвечивать введём

Случай 4: первый модуль > 0, второй < 0

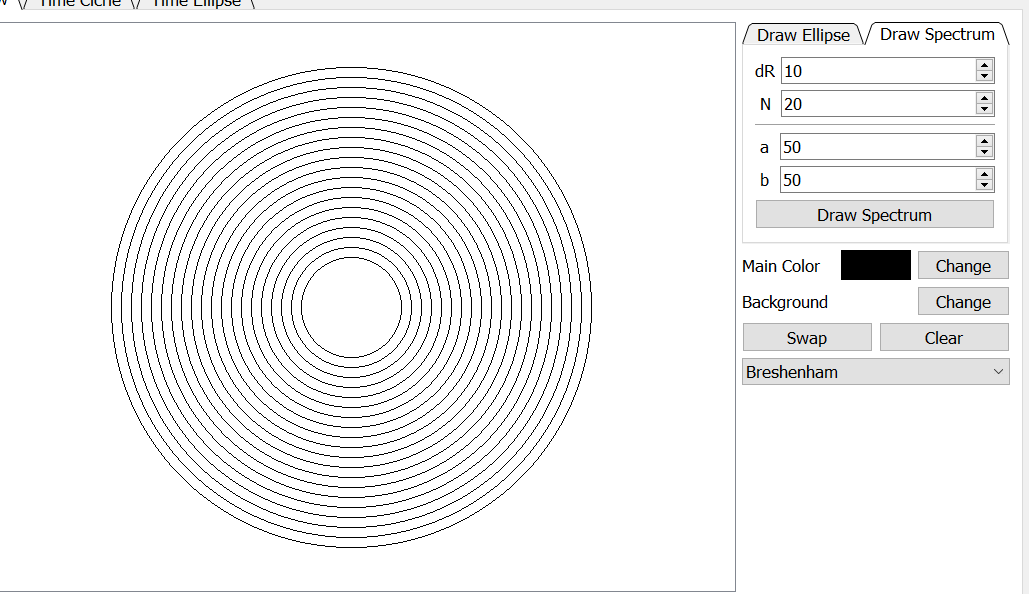
Случай 5: первый модуль > 0, второй > 0

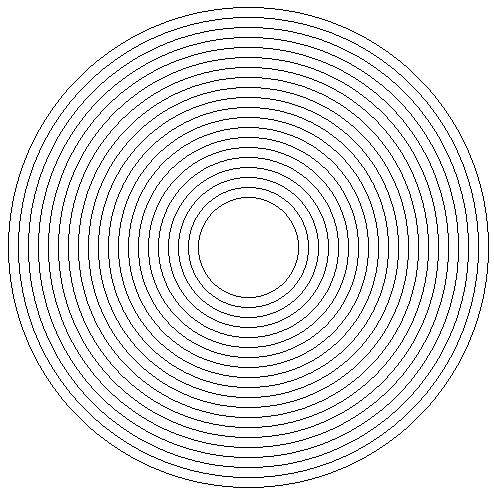
Начальное значение

Найдём при горизонтальном шаге

Найдём при диагональном шаге

Найдём при вертикальном шаге





**Алгоритм Средней точки для окружности**

void MiddleDot::DrawCicrle(QImage& canvas, int x, int y, int r, unsigned int color)

{

int i = 0, j = r;

// t = x^2 + y^2 - r^2

// t(1, r-1/2.)

// t = 5 / 4 - r

// 0,25 можно отбросить

// начинаем с целого числа и каждый раз прибавляем целое число

// невозможна ситуация целое число + 0,25 получить ноль

int t = 1 - r;

// t += 2 \* i + 3; при горизонтальном шаге

// t += 2 \* (i - j) + 5 при диагональном шаге

int incrA = 3; // i = 0

int incrB = 5 - 2 \* r; // i = 0, j = r

while(i < j)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

canvas.setPixel(x + j, y + i, color);

canvas.setPixel(x + j, y - i, color);

canvas.setPixel(x - j, y + i, color);

canvas.setPixel(x - j, y - i, color);

// если диагональный пиксель ближе

if (t >= 0)

{

// скорректировать пробную функцию

t += incrB;

incrB += 4;

j--;

}

else

{

t += incrA;

incrB += 2;

}

incrA += 2;

i++;

}

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

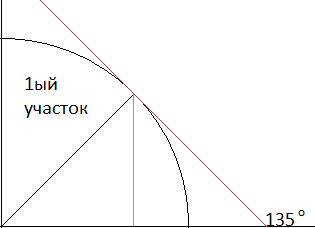
canvas.setPixel(x - i, y - j, color);

}

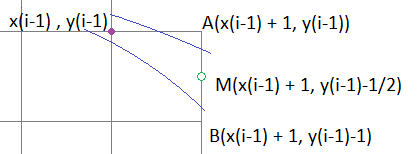
Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

Окружность – симметричная фигура, можно построить одну восьмую и отражать.

Достаточно построить 1ый участок остальные получить отражением относительно осей OX, OY и прямой y=x



На этом участке возможен выбор из двух пикселей горизонтального и диагонального.



Для того, чтобы решить какой пиксель высвечиваем введём пробную функцию

Найдём изменение пробной функции при горизонтальном шаге

Найдём изменение пробной функции при диагональном шаге

Найдём начальное значение пробной функции

Заметим, начальное значение равно целому числу сдвинутому на 0,25 вверх, но изменение пробной функции есть число целое и при выборе высвечиваемого пикселя, нам важен лишь знак пробной функции, поэтому величину 0,25 можно отбросить.

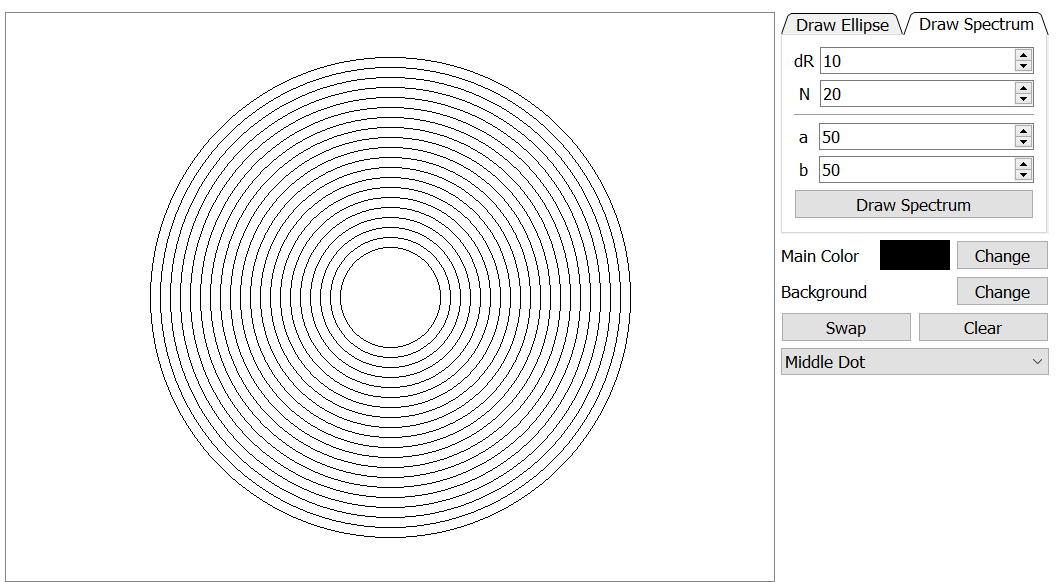
Избавимся от умножения на x, y в цикле.

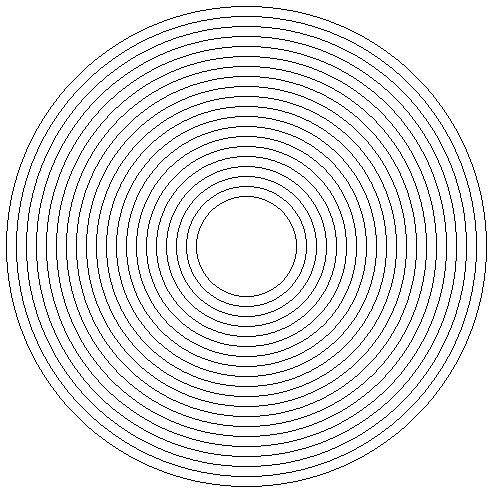
Для этого введём инкремент incrA – величина изменения пробной функции при горизонтальном шаге, инкремент incrB – величина изменения пробной функции при диагональном шаге.

Вычислим начальное значение инкрементов

Найдём изменение инкрементов при горизонтальном шаге.

Найдём изменение инкрементов при диагональном шаге.





**Алгоритм с использованием параметрического уравнения эллипса**

void Parametric::DrawEllipse(QImage& canvas, int x, int y, int a, int b, unsigned int color)

{

// кривизна кривой 1/ R = K = lim da/ds

// ds - смещение вдоль дуги кривой

// da - изменение угла наклона касательной

// K = |x'\*y'' - y'\*x''| / (x'^2 + y'^2)^(3/2)

// x = cos(t)

// y = sin(t)

// K = ab/(a^2\*sin^2(t) + b^2\*cos^2(t))^(3/2)

// в вершинах t = 0 K = a/b^2

// t = pi/2 K = b/a^2

float step = a > b ? (float)b / (a \* a) : (float)a / (b \* b);

int i, j;

for (float t = 0; t < M\_PI\_2 + step / 2; t += step)

{

i = roundf(a \* cosf(t));

j = roundf(b \* sinf(t));

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

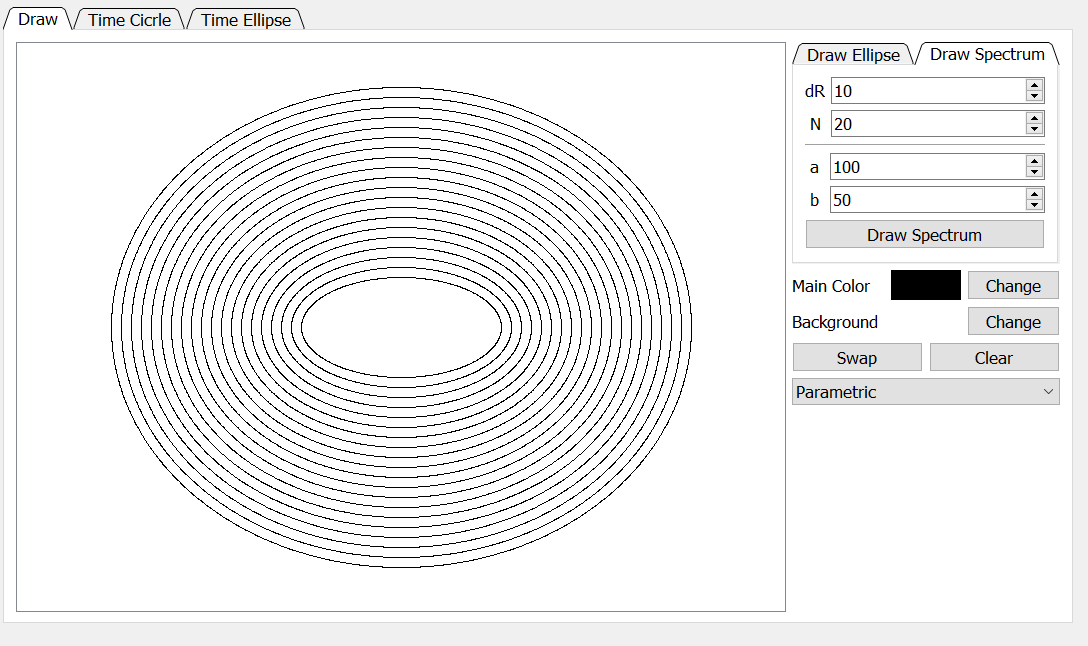
canvas.setPixel(x - i, y - j, color);

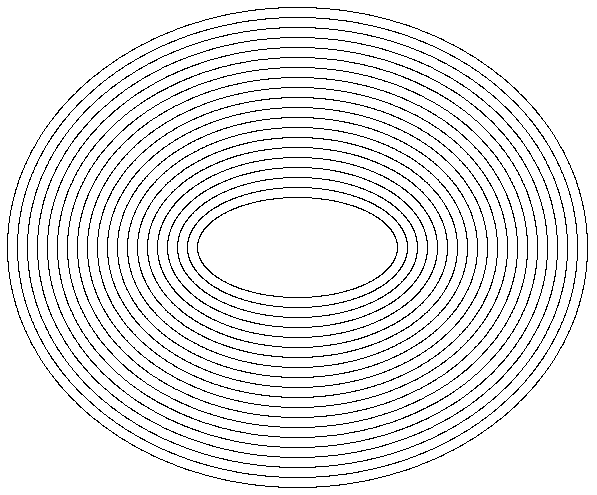
}

}

Проблемы данного алгоритма

1. Вычисление sin и cos
2. Округление





**Алгоритм с использованием канонического уравнения эллипса**

void Canonical::DrawEllipse(QImage& canvas, int x, int y, int a, int b, unsigned int color)

{

int a2 = a \* a;

int b2 = b \* b;

// Производная при y'= -1 , является границей для оптимального рисования

// y'= -b^2/a^2 \* x/y

// b^2 \* x = a^2 \* y

int i = 0, j = b;

// dx = 1

while (b2 \* i < a2 \* j)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

i++;

j = roundf(b / (float)a \* sqrtf(a2 - i \* i));

}

// dy = -1

while (j >= 0)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

j--;

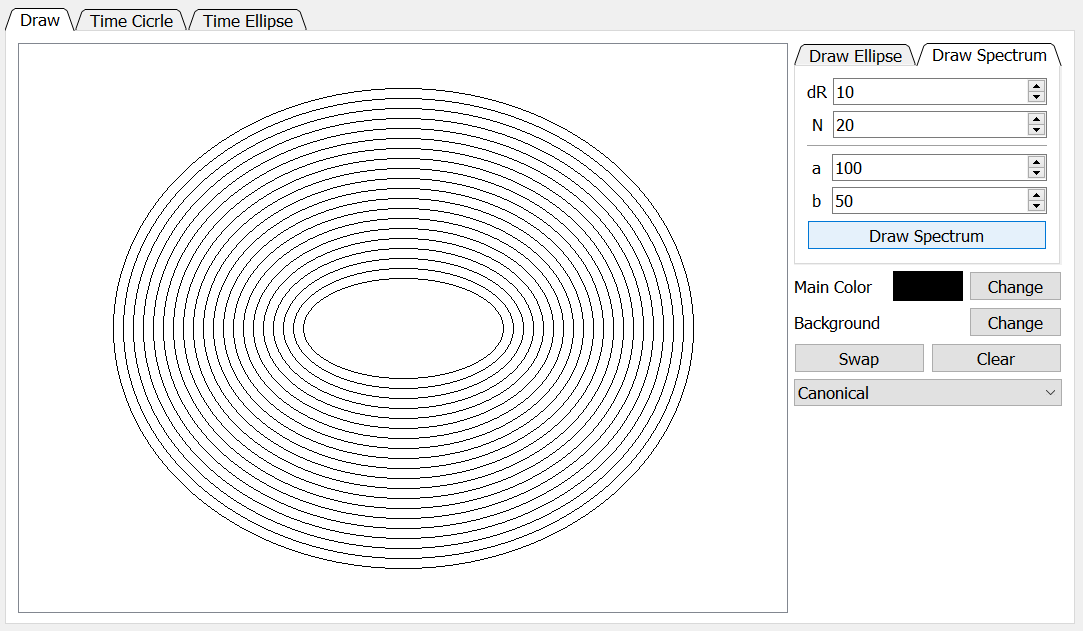
i = roundf(a / (float)b \* sqrtf(b2 - j \* j));

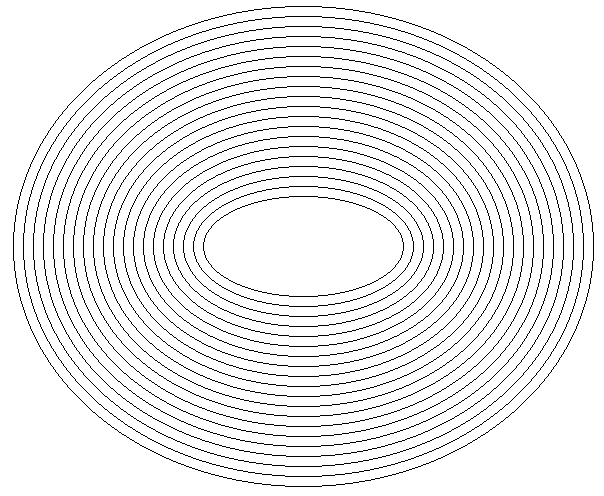
}

}

Проблемы данного алгоритма

1. Округление
2. Вычисление квадрата и квадратного корня





**Алгоритм Брезенхэма для эллипса**

inline void DiagonalStepEllipse(int& x, int& y, int a2, int b2, int& d)

{

x++;

y--;

d += b2 \* (2 \* x + 1) + a2 \* (1 - 2 \* y);

}

inline void HorizontalStepEllipse(int& x, int& y, int a2, int b2, int& d)

{

x++;

d += b2 \* (2 \* x + 1);

}

inline void VerticalStepEllipse(int& x, int& y, int a2, int b2, int& d)

{

y--;

d += a2 \* (1 - 2 \* y);

}

void Breshenham::DrawEllipse(QImage& canvas, int x, int y, int a, int b, unsigned int color)

{

int a2 = a \* a;

int b2 = b \* b;

int d = a2 + b2 - 2 \* a2 \* b;

int i = 0, j = b;

while (j > 0)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

if (d == 0) // диагональная точка лежит на эллипсе

DiagonalStepEllipse(i, j, a2, b2, d);

else if (d < 0) // диагональная точка внутри эллипса

{

// раст. до гор пикселя раст. до диаг. пикселя

// |b^2(xi+1)^2 + a^2yi^2 - a^2b^2| - |b^2(xi+1)^2 + a^2(yi-1)^2 - a^2b^2|

int delta1 = 2 \* (d + a2 \* j) - a2;

if (delta1 > 0)

DiagonalStepEllipse(i, j, a2, b2, d);

else

HorizontalStepEllipse(i, j, a2, b2, d);

}

else // диагональная точка вне эллипса

{

// раст. до диаг. пикселя раст. до верт пикселя

// |b^2(xi+1)^2 + a^2(yi-1)^2 - a^2b^2| - |b^2xi^2 + a^2(yi-1)^2 - a^2b^2|

int delta2 = 2 \* (d - b2 \* i) - b2;

if (delta2 < 0)

DiagonalStepEllipse(i, j, a2, b2, d);

else

VerticalStepEllipse(i, j, a2, b2, d);

}

}

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

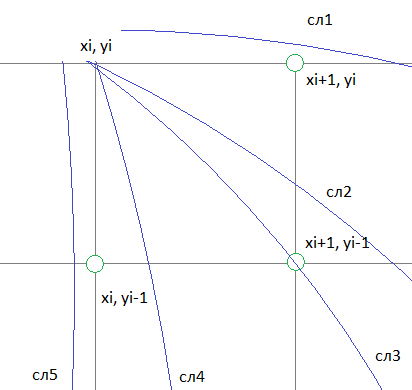
canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

}

Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

Эллипс – симметричная фигура, можно построить одну четверть и отразить относительно OX и OY



– в первой четверти монотонно убывающая.

На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

Критерием выбора высвечиваемого пикселя является величина

Если

Возможны случаи 1 и 2.

Для того чтобы решить какой пиксель высвечивать введём

Случай 1: первый модуль <=0, второй <0

Случай 2: первый модуль >=0, второй <0

Возможны случаи 4 и 5.

Для того чтобы решить какой пиксель высвечивать введём

Случай 4: первый модуль > 0, второй < 0

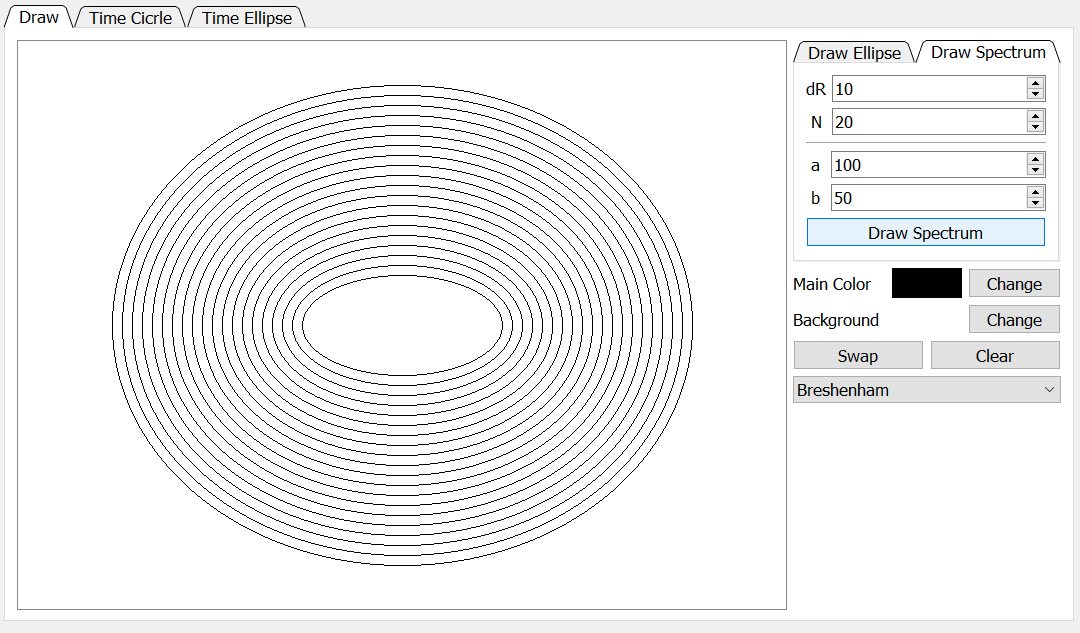
Случай 5: первый модуль > 0, второй > 0

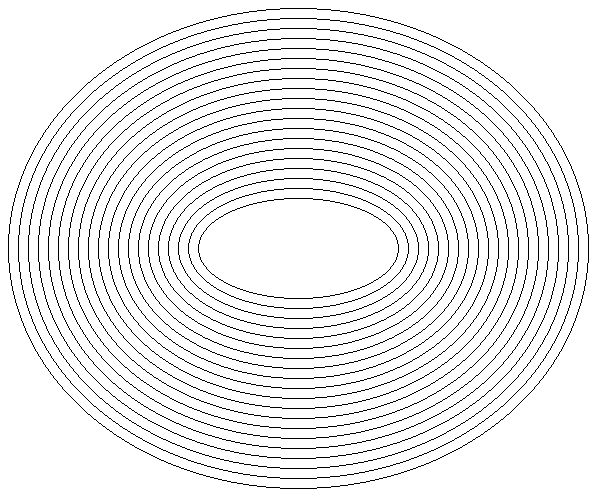
Начальное значение

Найдём при горизонтальном шаге

Найдём при диагональном шаге

Найдём при вертикальном шаге





**Алгоритм средней точки для эллипса**

void MiddleDot::DrawEllipse(QImage& canvas, int x, int y, int a, int b, unsigned int color)

{

int a2 = a \* a;

int b2 = b \* b;

int ad = 2 \* a2;

int bd = 2 \* b2;

int i = 0, j = b;

// tпр = b^2\*x^2 + a^2\*y^2 - a^2 \* b^2;

// начальное значение средней точки M(1, b-1/2.)

float t = a2 / 4. - a2 \* b + b2;

// Производная при y'= -1 , является границей для оптимального рисования

// y'= -b^2/a^2 \* x/y

while (b2 \* i < a2 \* j)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

// M вне эллипса, ближе диагональный пиксель

if (t >= 0)

{

j--;

t -= ad \* j;

}

i++;

t += bd \* i + b2;

}

t += 3 / 4. \* (a2 - b2) - (b2 \* i + a2 \* j);

while (j > 0)

{

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

canvas.setPixel(x - i, y - j, color);

// M внутри эллипса, ближе диагональный пиксель

if (t <= 0)

{

i++;

t += bd \* i;

}

j--;

t += -ad \* j + a2;

}

canvas.setPixel(x + i, y + j, color);

canvas.setPixel(x + i, y - j, color);

canvas.setPixel(x - i, y + j, color);

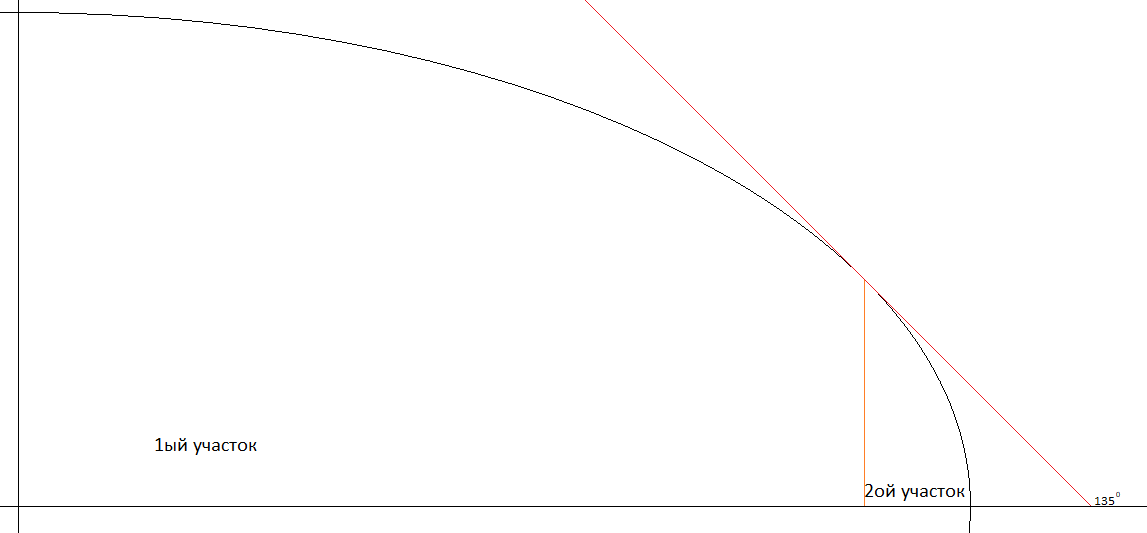
canvas.setPixel(x - i, y - j, color);

}

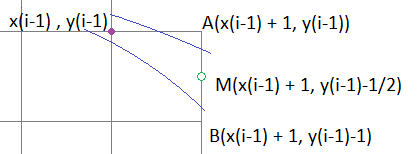
Будем считать, что центр находится в начале координат, хс=ус=0. У направлена вверх, Х вправо.

Эллипс – симметричная фигура, можно построить одну четвертую и отразить относительно осей OX и OY

Достаточно построить 1ый участок остальные получить отражением относительно осей OX, OY и прямой y=x



На первом участке возможен выбор из двух пикселей горизонтального и диагонального.

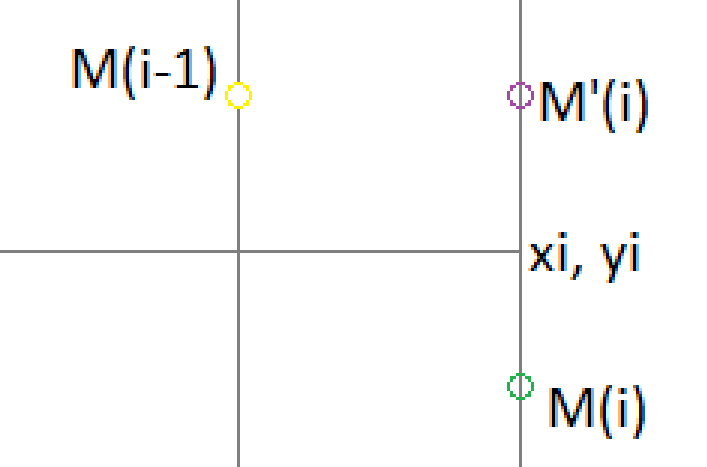


Для того, чтобы решить какой пиксель высвечиваем введём пробную функцию

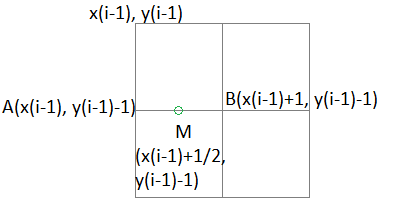
Найдём начальное значение пробной функции

Найдём изменение пробной функции при горизонтальном шаге

Скорректируем значение пробной функции при диагональном шаге



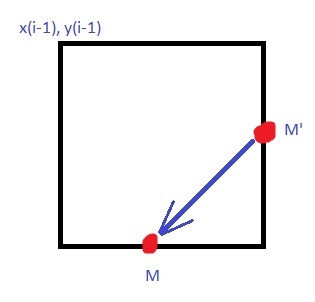
После 1ого участка переходим к обработке 2ого.

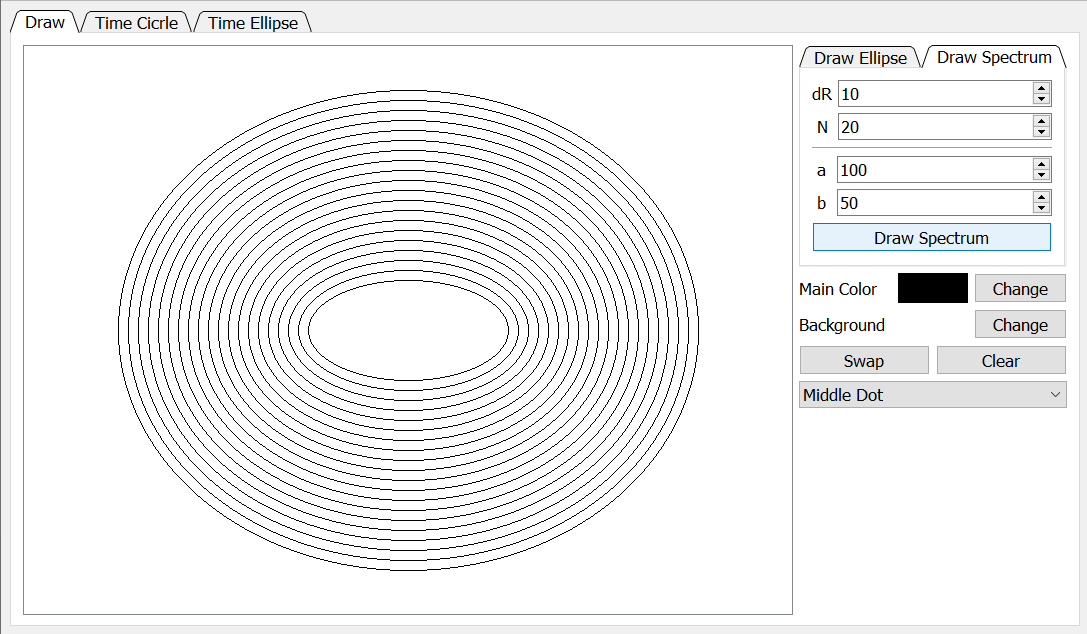


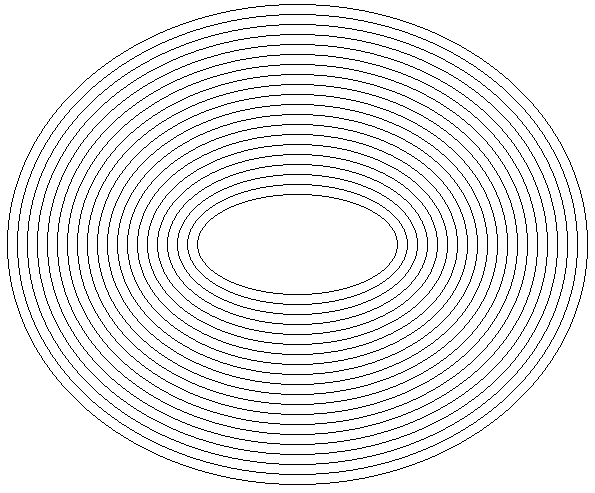
Найдём изменение пробной функции при вертикальном шаге

Скорректируем значение пробной функции при диагональном шаге

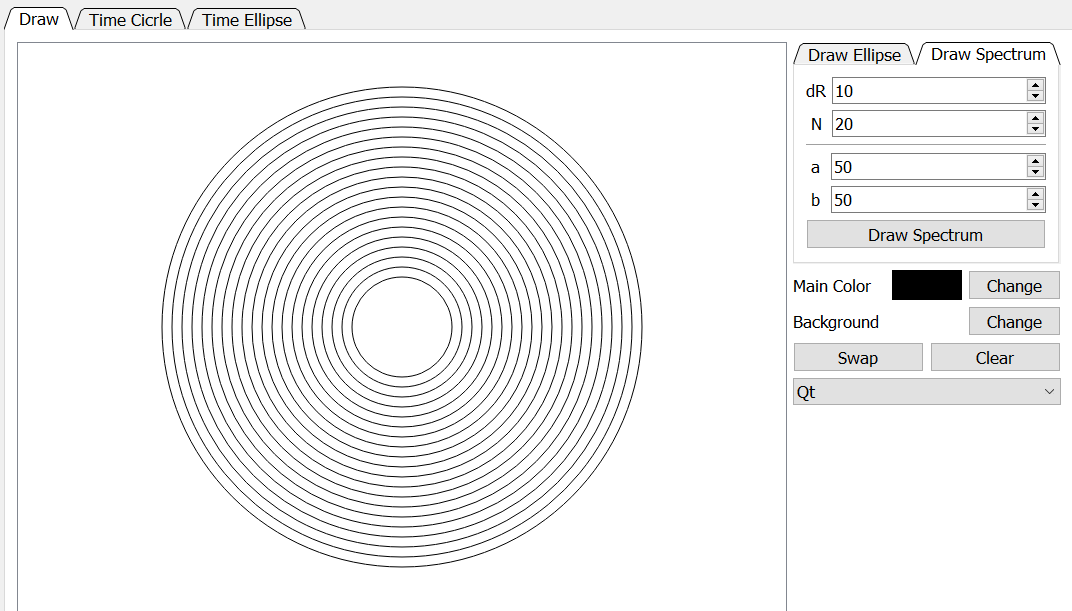
При переходе от 1ого интервала ко 2ому необходимо однократно скорректировать ранее вычисленное значение пробной функции.

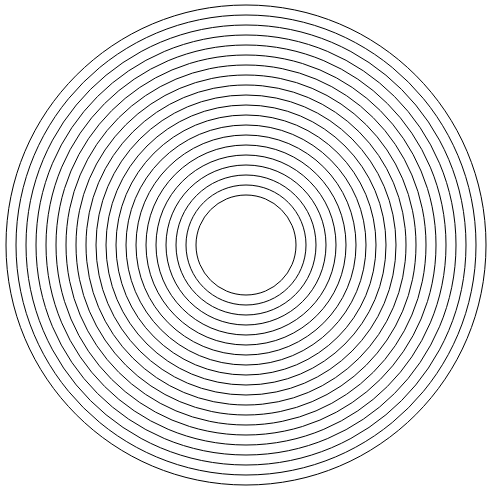


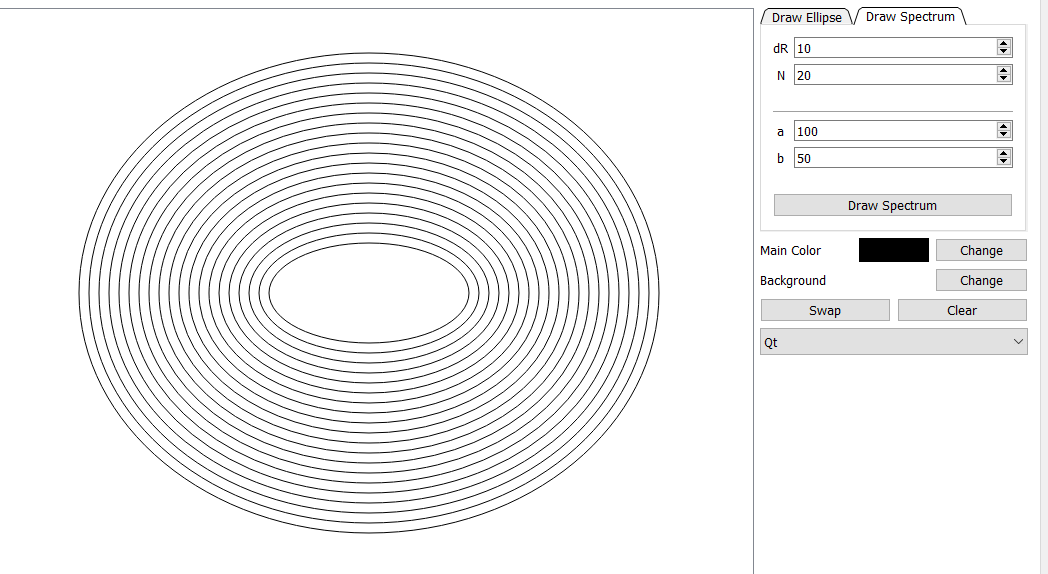


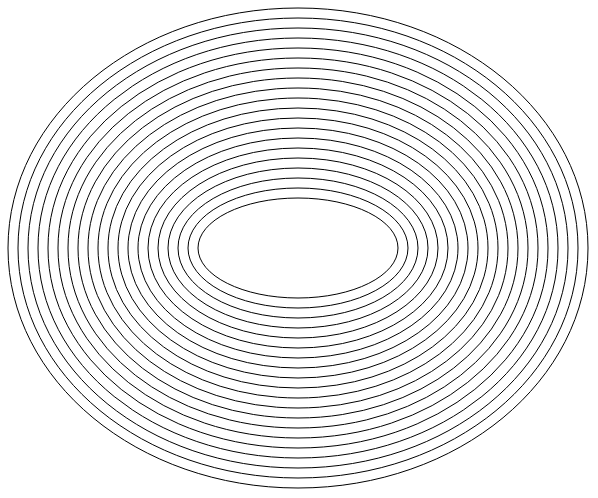


**Библиотечный алгоритм**



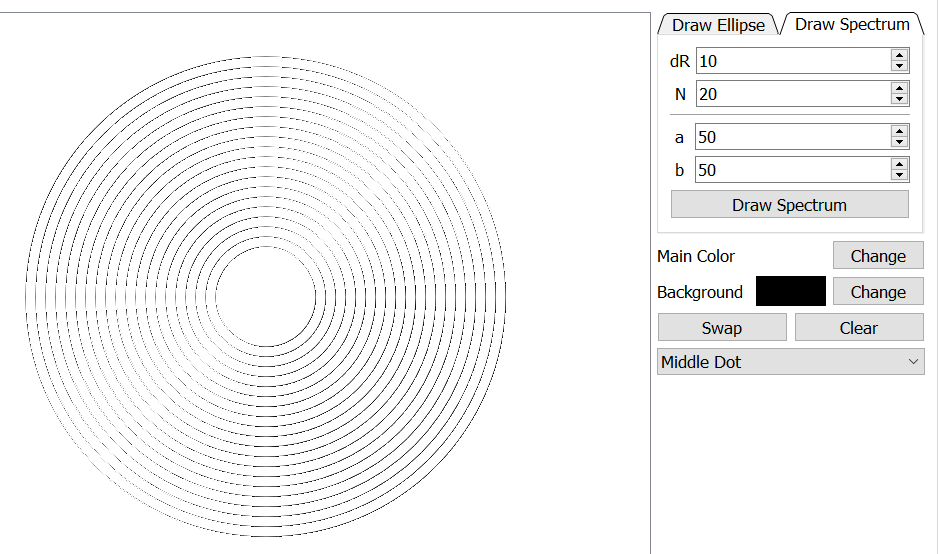






Библиотечный алгоритм применяет один из алгоритмов устранения ступенчатости.

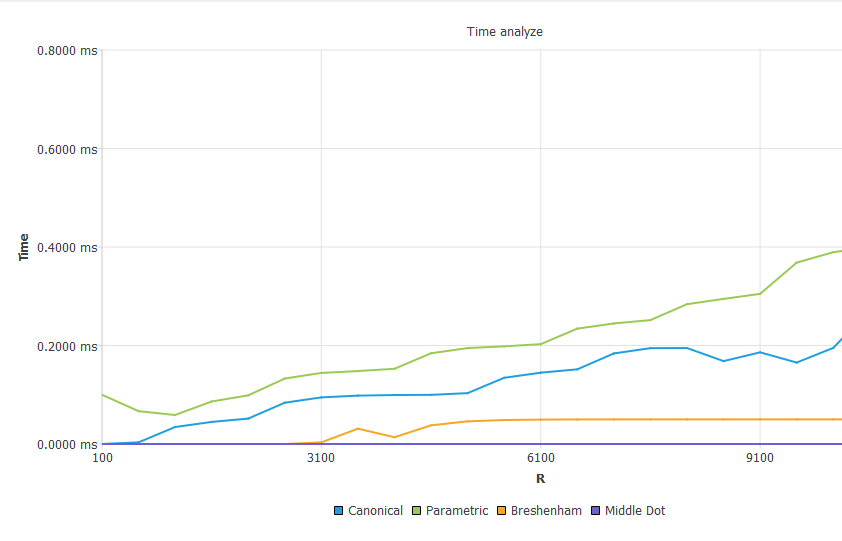
Результат наложения алгоритма средней точки на библиотечный алгоритм.



Можно заметить, что библиотечный алгоритм рисует окружность двойной толщины. Возможно библиотечный метод рисования окружностей / эллипсов является модифицированным алгоритмом Ву для отрезков.

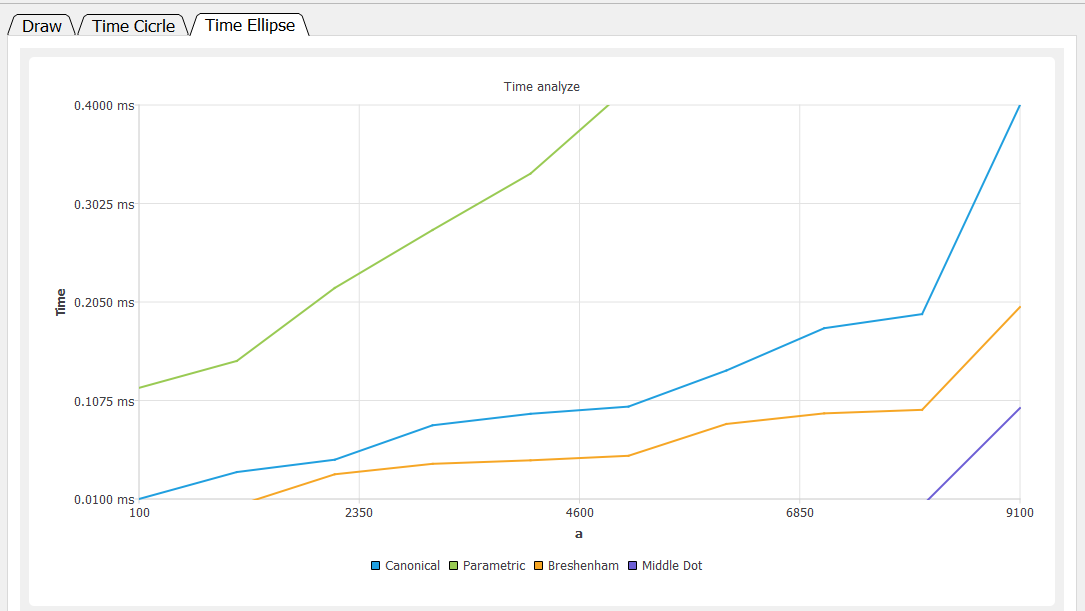
**Сравнение временных характеристик**

Зависимость времени работы алгоритмов построения окружности от её радиуса.



\*для каждого r проводятся 10 измерений и вычисляется среднее значение

Зависимость времени работы алгоритмов построения эллипса от её радиуса.



\*для каждого a и b проводятся 10 измерений и вычисляется среднее значение