|  |  |
| --- | --- |
| 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 генерации окружности и** **эллипса»**  **Дисциплина Компьютерная графика**  **Студент Кузин Антон**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Введение:**

Наиболее простыми подходами к построению окружности является использование канонического и параметрического уравнений, однако они требуют исследования большего количества точек. Поэтому появляется необходимость в более сложных алгоритмах, таких как алгоритм Брезенхема и средней точки.

**Цель:**

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

**Техническое задание:**

Реализовать алгоритмы построения окружности на основе

- Канонического уравнения X^2+Y^2=R^2

- Параметрического уравнения X=Rcost, Y=Rsint

- Алгоритма Брезенхема

- Алгоритма средней точки

- построение окружности с помощью библиотечной функции

Реализовать алгоритмы построения эллипса на основе

- Канонического уравнения X^2/a^2+Y^2/b^2=1

- Параметрического уравнения X=acost, Y=bsint

- Алгоритма Брезенхема (модифицировать самостоятельно)

- Алгоритма средней точки

- построение эллипса с помощью библиотечной функции

Пользователь выбирает из списка определённый алгоритм, задаёт центр, радиус или полуоси, цвет окружности или эллипса.

Сравнение визуальных характеристик при рисовании спектра концентрических окружностей или эллипсов.

Пользователь выбирает из списка алгоритм, задаёт цвет рисования, центр, изменение радиусов или полуосей.

Сравнить временные характеристики разных алгоритмов, построив графики в одном поле вывода.

**Практическая часть:**

При построении окружности, либо эллипса достаточно построить одну четверть, и отразить ее в 3 другие, для этого применяется функция:

void drawQuarters(const QPoint &center, int dx, int dy, canvas\_t &canvas)

{

canvas.image->setPixel(center.x() + dx, center.y() + dy, canvas.color->rgb());

canvas.image->setPixel(center.x() + dx, center.y() - dy, canvas.color->rgb());

canvas.image->setPixel(center.x() - dx, center.y() + dy, canvas.color->rgb());

canvas.image->setPixel(center.x() - dx, center.y() - dy, canvas.color->rgb());

}

1. **Каноническое уравнения**

Для окружности:

void canon(const QPoint &center, int r, canvas\_t &canvas)

{

float x;

for (float y = r; y > 0; y -= 1. / r)

{

x = sqrt(r \* r - y \* y);

drawQuarters(center, round(x), round(y), canvas);

}

}

Для эллипса:

void canonEl(const QPoint &center, int a, int b, canvas\_t &canvas)

{

int x, y;

int sqra = a \* a;

int sqrb = b \* b;

float ba = float(b) / a;

int dX = int(sqra / sqrt(sqra + sqrb));

for (x = 0; x <= dX; x++)

{

y = round(sqrt((sqra - x\*x)) \* ba);

drawQuarters(center, round(x), round(y), canvas);

}

float ab = float(a) / b;

int dY = round(sqrb / sqrt(sqra + sqrb));

for (y = 0; y <= dY; y++)

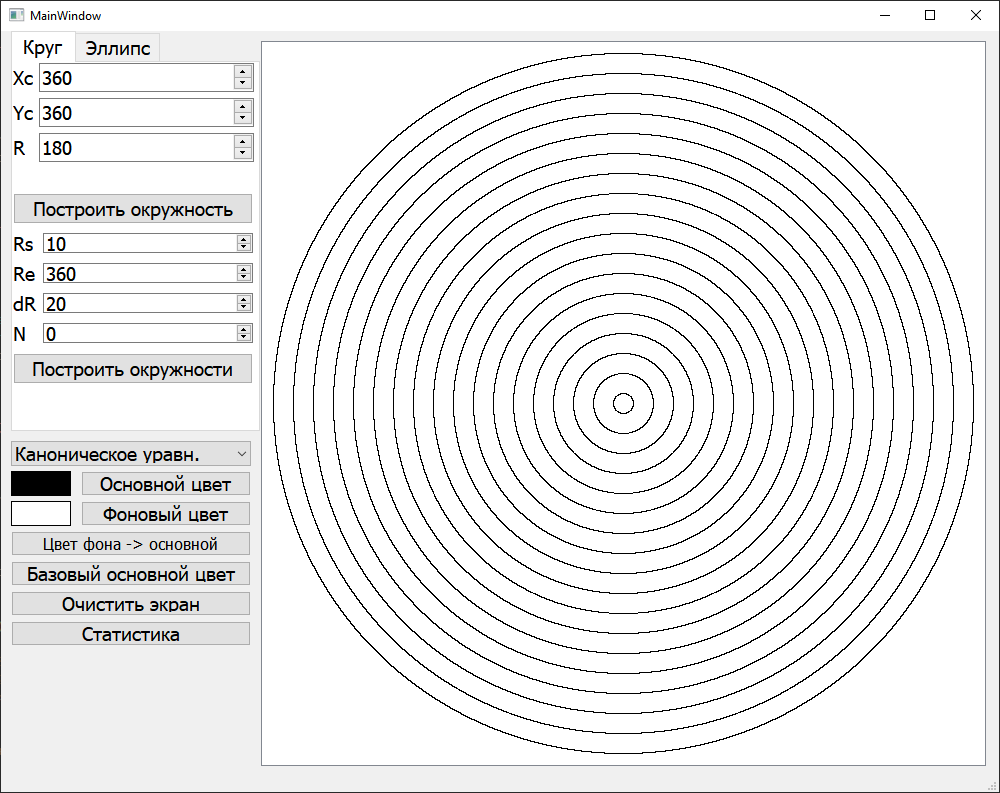
{

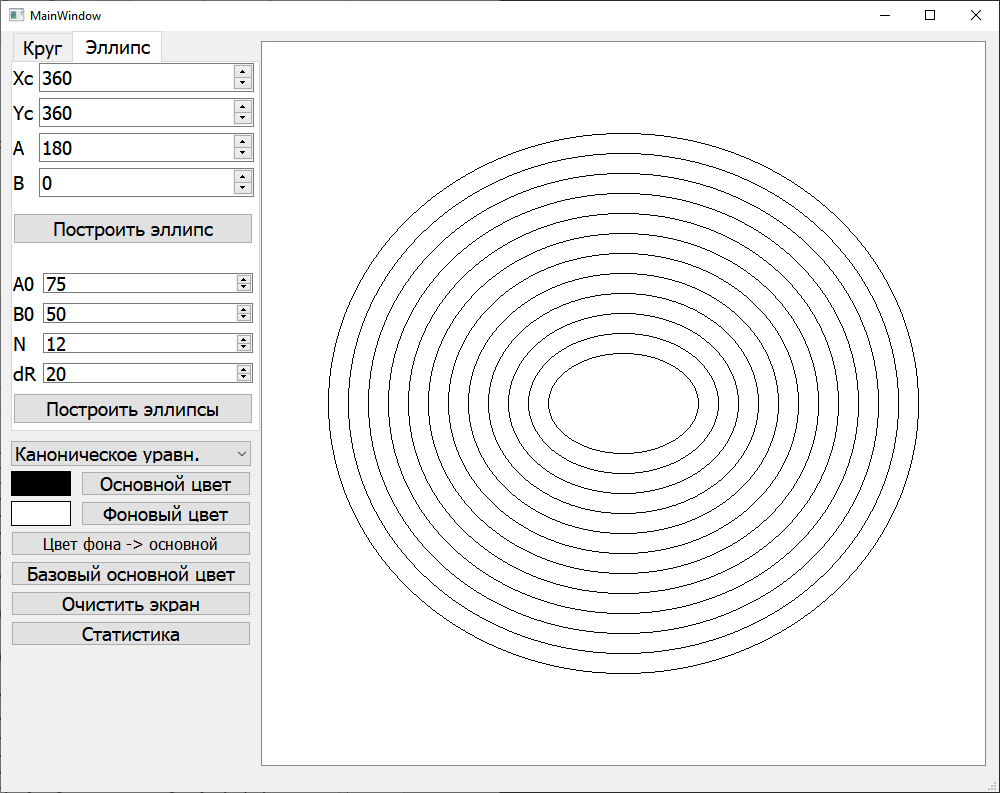
x = round(sqrt((sqrb - y\*y)) \* ab);

drawQuarters(center, round(x), round(y), canvas);

}

}





Построение окружности на основе канонического уравнения требует изменения значения параметра в зависимости от радиуса окружности, либо значений полуосей, в случае эллипса, также он требует взятия квадратного корня и округления, что значительно замедляет время работы.

1. **Параметрическое уравнение**

Для окружности:

void param(const QPoint &center, int r, canvas\_t &canvas)

{

float x, y;

for (float t = 0; t <= 2; t += 1. / r)

{

x = round(cos(t) \* r);

y = round(sin(t) \* r);

drawQuarters(center, x, y, canvas);

}

}

Для эллипса:

void paramEl(const QPoint &center, int a, int b, canvas\_t &canvas)

{

float dt = 1. / qMax(a, b);

int x, y;

for (float t = M\_PI / 2.; t > -dt / 2.; t -= dt)

{

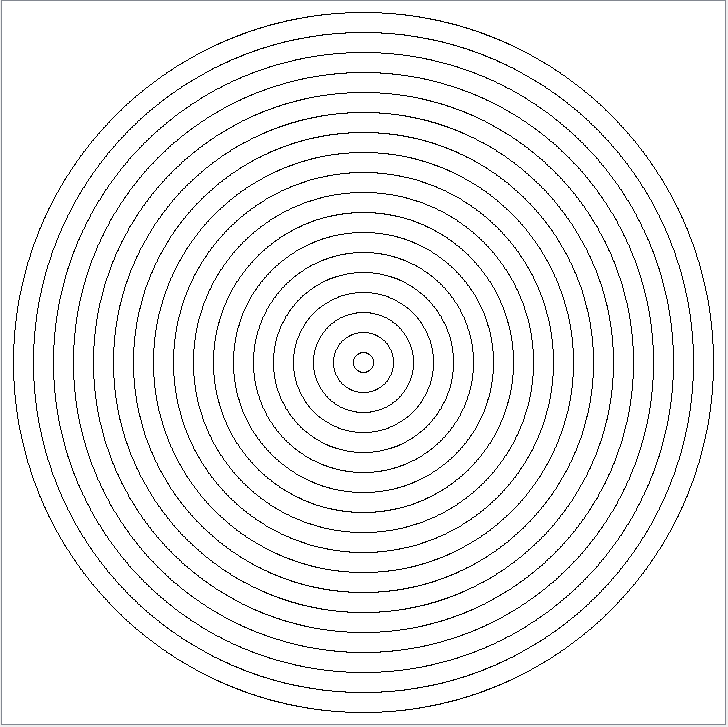
x = round(a \* cos(t));

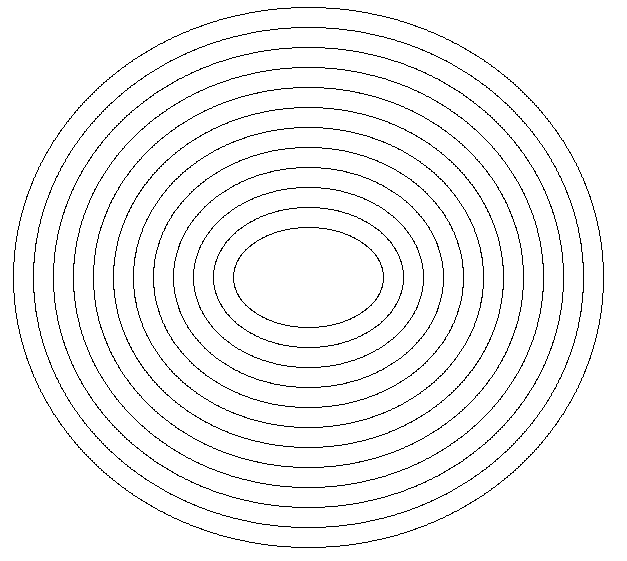
y = round(b \* sin(t));

drawQuarters(center, x, y, canvas);

}

}





У построения окружности, либо эллипса основываясь на параметрическом уравнении недостатки, аналогичные построению на основе канонического уравнения. На каждом шаге необходимо считать значение синуса, косинуса, округлять. Однако, этот метод является одним из наиболее алгоритмически простых, как для построения окружности, так и для построения эллипса.

1. **Брезенхем**

Для окружности:

void bresenhem(const QPoint &center, int r, canvas\_t &canvas)

{

int x = 0;

int y = r;

int d = 2 \* (1 - r), d1, d2;

int yk = 0;

while (y >= yk)

{

drawQuarters(center, x, y, canvas);

if (d < 0)

{

d1 = 2 \* (d + y) - 1;

x++;

if (d1 < 0)

d += 2 \* x + 1;

else

{

y--;

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

}

}

else if (d == 0)

{

x++;

y--;

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

}

else if (d > 0)

{

d2 = 2 \* d - 2 \* x - 1;

y--;

if (d2 < 0)

{

x++;

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

}

else

d -= 2 \* y - 1;

}

}

}

Для эллипса:

void bresenhemEl(const QPoint &center, int a, int b, canvas\_t &canvas)

{

int x = 0;

int y = b;

int sqra = a \* a;

int sqrb = b \* b;

int d2, d1;

int d = sqra + sqrb - 2 \* sqra \* y;

while (y >= 0)

{

drawQuarters(center, x, y, canvas);

if (d < 0)

{

d1 = 2 \* (d + sqra \* y) - sqra;

x++;

if (d1 < 0)

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

else

{

y--;

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

}

}

else if (d == 0)

{

x++;

y--;

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

}

else if (d > 0)

{

d2 = 2 \* (d - sqrb \* x) - sqrb;

y--;

if (d2 >= 0)

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

else

{

x++;

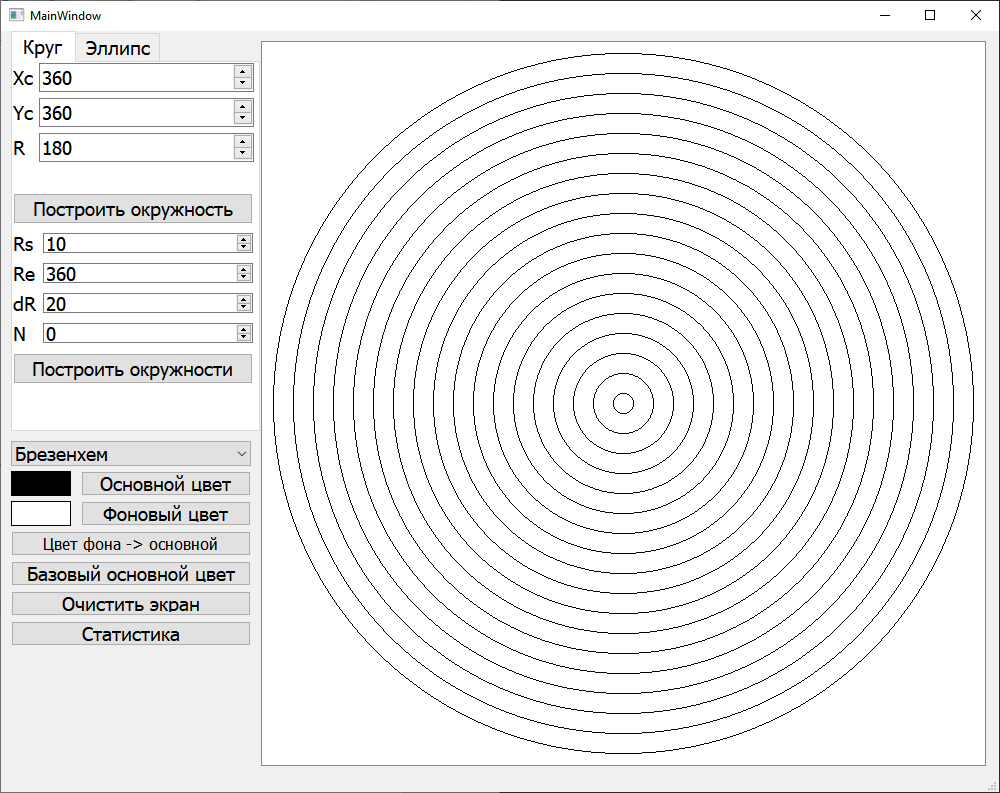
d += 2 \* (sqrb \* x - sqra \* y) + sqra + sqrb;

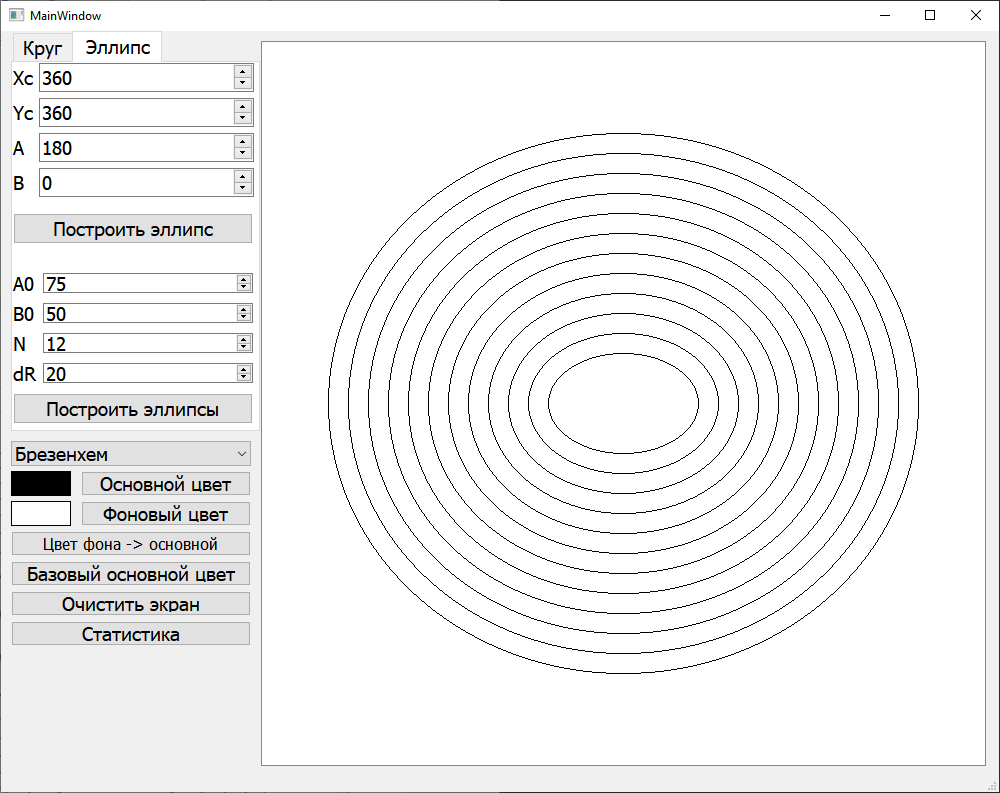
}

}

}

}





Алгоритм Брезенхема для построения окружности лишён недостатков предыдущих двух, так как он работает с целочисленными переменными, а также в меньшей степени зависит от изменения радиусов или длины полуосей, что будет видно в дальнейшем.

1. **Средней точки**

Для окружности:

void midPoint(const QPoint &center, int r, canvas\_t &canvas)

{

int x = 0;

int y = r;

int d = 1 - r;

while (y > x)

{

drawQuarters(center, x, y, canvas);

drawQuarters(center, y, x, canvas);

x++;

if (d < 0)

d += 2 \* x + 1;

else

{

y--;

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

}

}

}

Для эллипса:

void midPointEl(const QPoint &center, int a, int b, canvas\_t &canvas)

{

int x = 0;

int y = b;

int sqrb = a \* a;

int sqra = b \* b;

int f = sqra + sqrb \* (y - 0.5f) \* (y - 0.5) - (long long)(sqrb) \* sqra;

int dX = sqrb / sqrt(sqra + sqrb);

while (x <= dX)

{

drawQuarters(center, x, y, canvas);

x++;

if (f > 0)

{

y--;

f += -2 \* sqrb \* y;

}

f += sqra \* (2 \* x + 1);

}

f += 0.75f \* (sqrb - sqra) - (sqra \* x + sqrb \* y);

while (y >= 0)

{

drawQuarters(center, x, y, canvas);

y--;

if (f < 0)

{

x++;

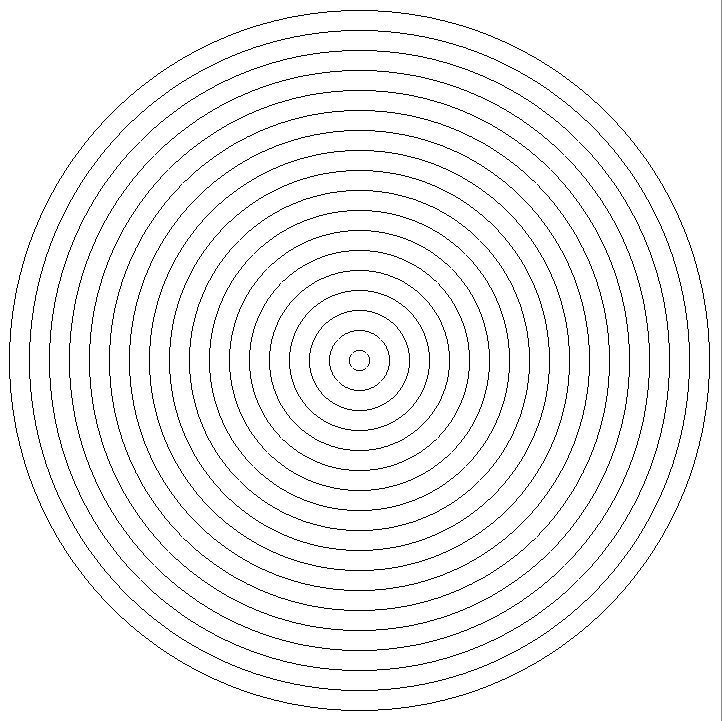
f += 2 \* sqra \* x;

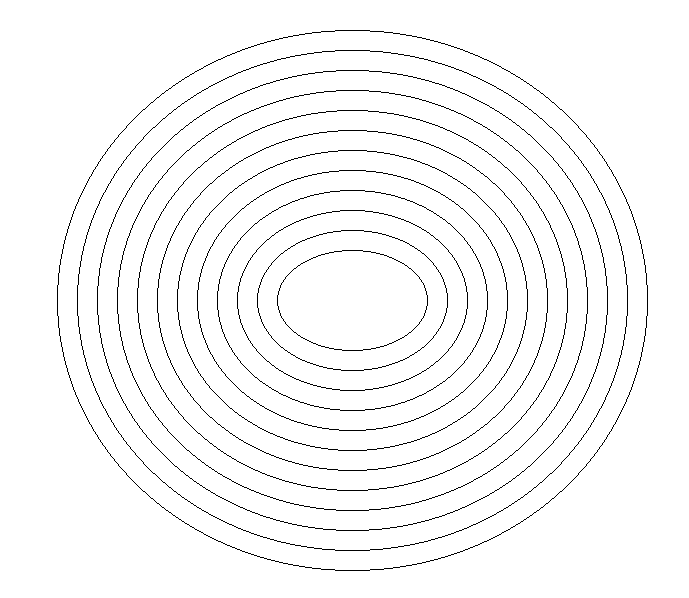
}

f += sqrb \* (1 - 2 \* y);

}

}





Метод средней точки выбирает те же точки, что и метод Брезенхема, однако он анализирует только одну восьмую окружности, что увеличивает его эффективность, при построении окружности, однако при построении эллипса он всё также анализирует целую четверть.

1. **Библиотечный**

void defaultQtDrawEl(const QPoint &center, int a, int b, QPainter &painter)

{

painter.drawEllipse(center, a, b);

}

void defaultQtEl(const QPoint &center, int a, int b, canvas\_t &canvas)

{

QPixmap pixmap = QPixmap::fromImage(\*canvas.image);

QPainter painter(&pixmap);

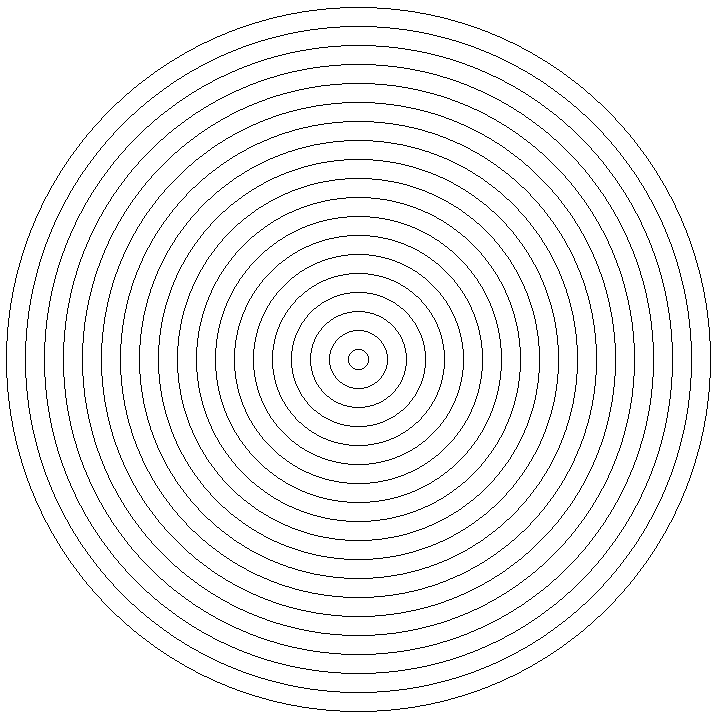
painter.setPen(\*canvas.color);

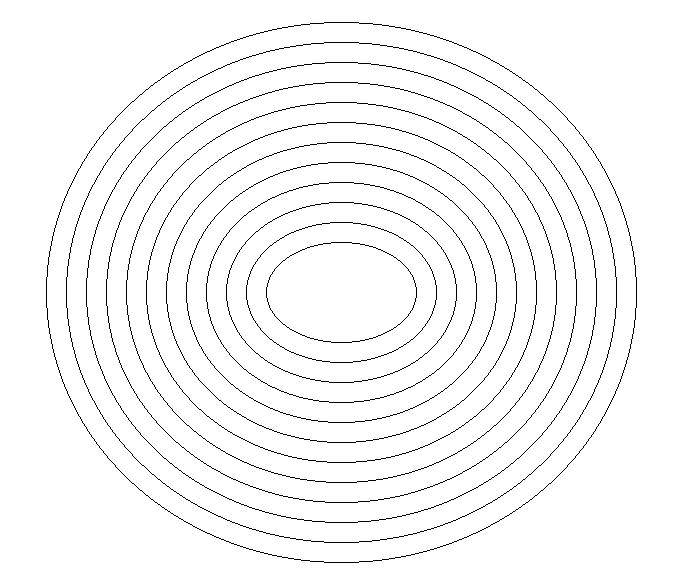
defaultQtDrawEl(center, a, b, painter);

painter.end();

\*canvas.image = pixmap.toImage();

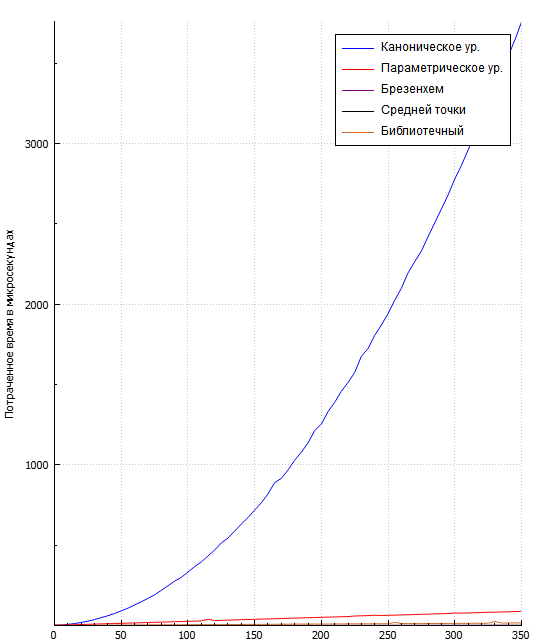
}

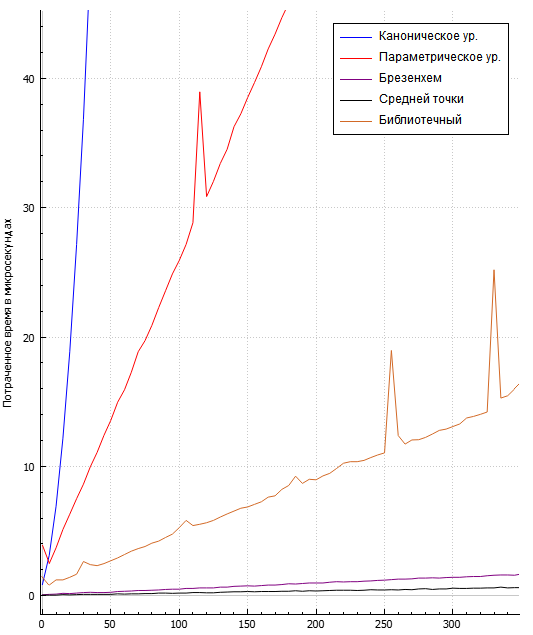




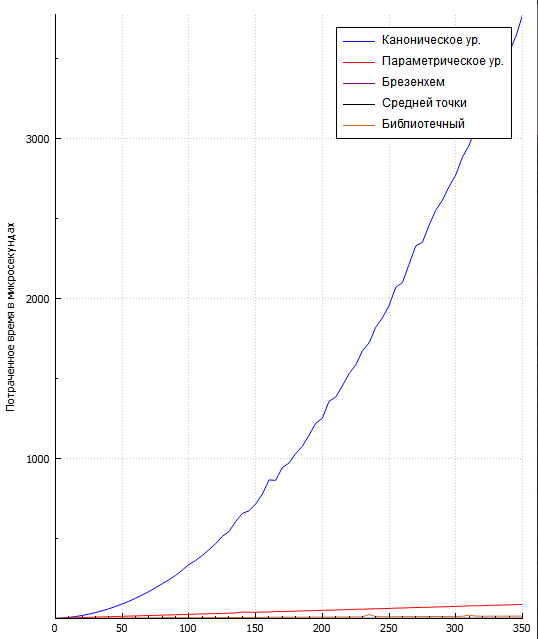
**Исследование временных характеристик**

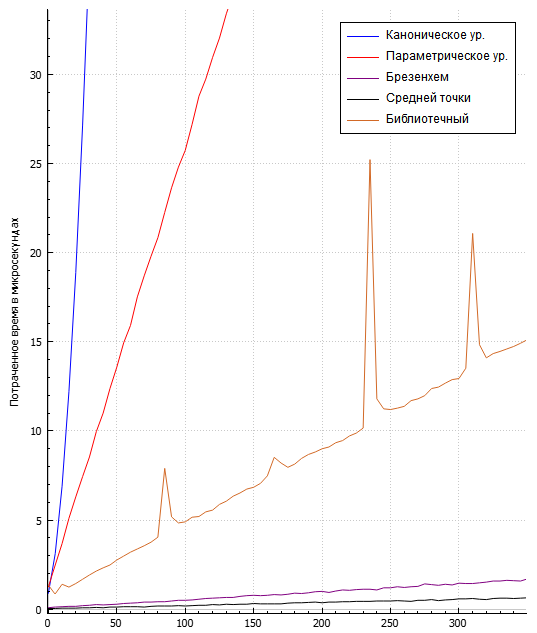
Для окружностей: измеряется при построении окружностей с радиусом от 5 до 360, с шагом изменения радиуса 5.





Для эллипсов: измеряется при построении эллипсов с полуосями 5 и 10, до 360 и 365, соответственно, с шагом изменения полуосей 5.





**Вывод**

Исследуя визуальные характеристики, становится ясно, что алгоритм Брезенхема, средней точки и библиотечный метод выбирают одни и те же точки. При исследовании временных характеристик видно, что построение окружности на основе канонического уравнения имеет наибольшую временную сложность, в зависимости от радиуса, в то время как алгоритмы Брезенхема и средней точки демонстрируют наилучшую эффективность по времени. Однако, следует заметить, что при использовании библиотечного метода невозможно полностью запретить отрисовку, поэтому он также является эффективным.