|  |  |
| --- | --- |
| **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 окружностей  **Студент:** Блохин Д.М.  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А. В.** |  |

Москва.

2020 г.

# Цель работы:

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

# Техническое задание:

1. Вывод на экран окружности, построенной с помощью одного из нижеперечисленных алгоритмов. Предоставить пользователю возможность выбрать любой из них:

- Канонического уравнения ;

- Параметрического уравнения ;

- Алгоритма Брезенхема;

- Алгоритма средней точки;

- Библиотечный алгоритм.

1. Вывод на экран эллипса, построенного с помощью одного из нижеперечисленных алгоритмов. Предоставить пользователю возможность выбрать любой из них:

- Канонического уравнения ;

- Параметрического уравнения ;

- Алгоритма Брезенхема (модифицировать самостоятельно);

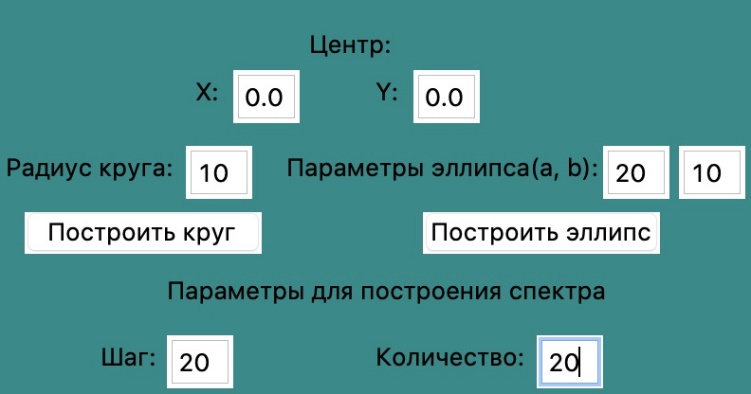
- Алгоритма средней точки;

- Библиотечный алгоритм;

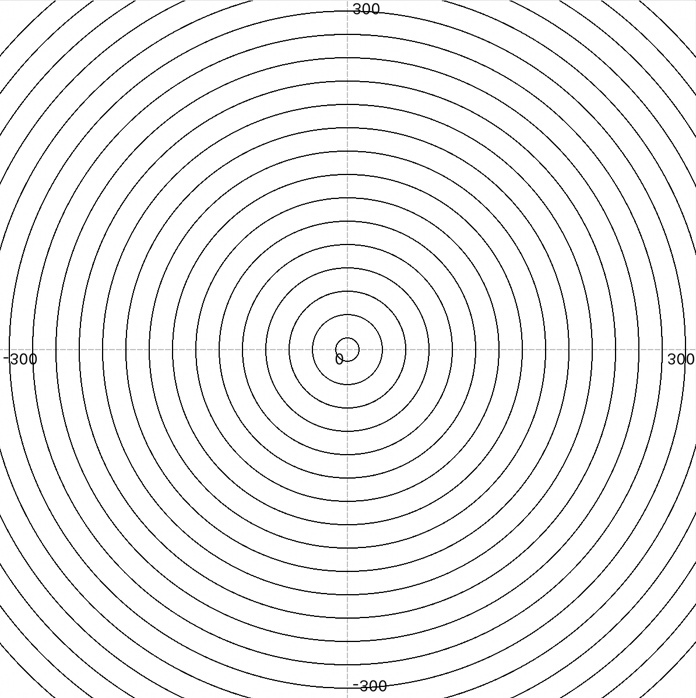
1. Исследование визуальных характеристик при рисовании спектра концентрических окружностей.
2. Исследование визуальных характеристик при рисовании спектра концентрических эллипсов.
3. Исследование временных характеристик алгоритмов построения окружностей (результат вывести в виде графика).
4. Исследование временных характеристик алгоритмов построения эллипсов (результат вывести в виде графика).

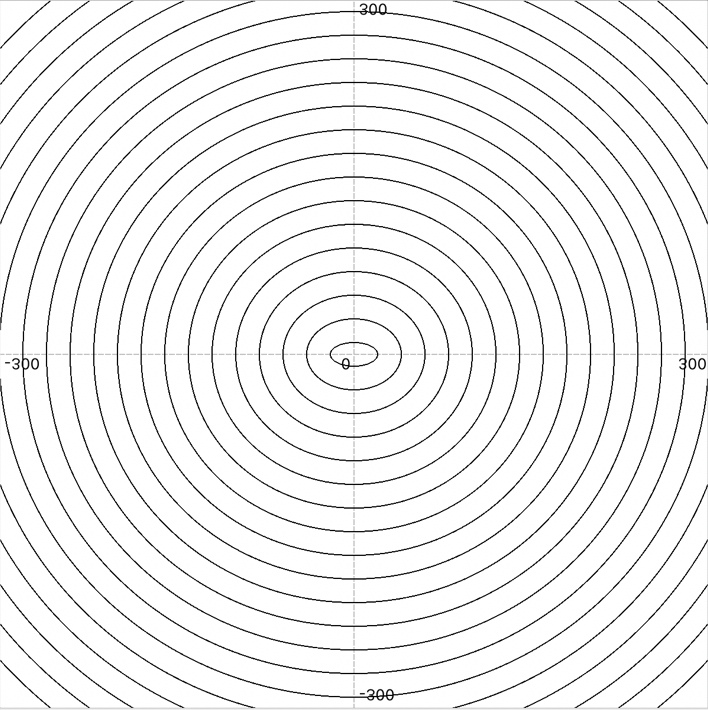
**Примеры работы программы**

Для примеров будут использоваться следующие входные данные:



# Библиотечный алгоритм





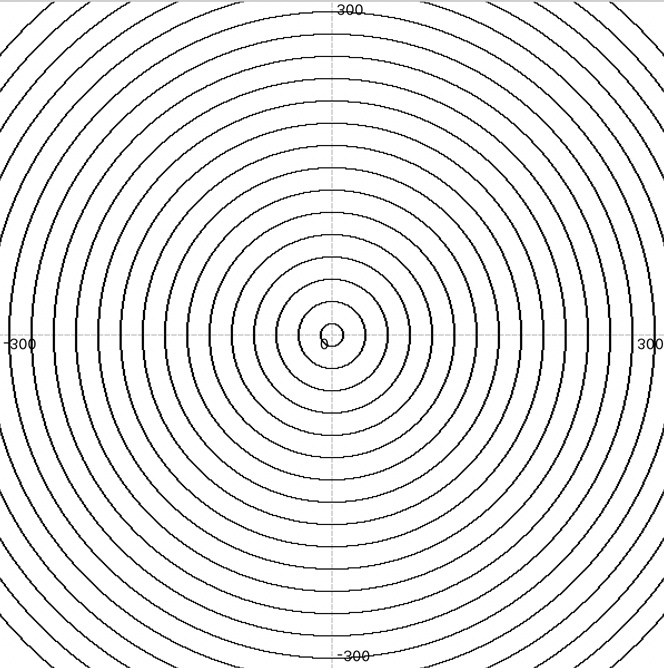
# Каноническое уравнение

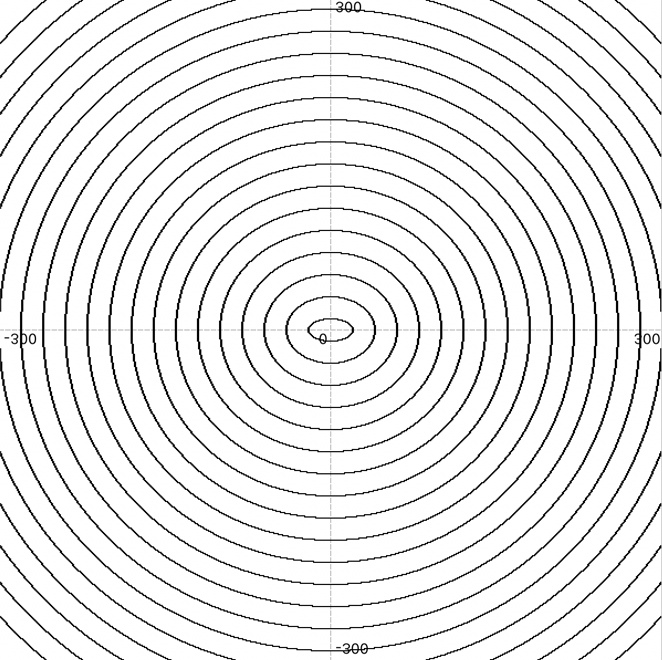
Для окружности:

Для эллипса:

Нужно рассчитать все точки окружности.

Данный способ довольно таки простой, но требует большого количества вычислений.

****



**Код алгоритма:**

**void canon\_circle(int x\_c, int y\_c, int r, graphics\_scene \*canvas)**

**{**

**int x = 0, y = 0;**

**double r2 = r \* r;**

**while (x <= r)**

**{**

**y = (int)round(sqrt(r2 - x \* x));**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**x += 1**

**}**

**while (y <= r)**

**{**

**x = (int)round(sqrt(r2 - y \* y));**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**y += 1**

**}**

**}**

**void canon\_ellipse(int x\_c, int y\_c, int a, int b, graphics\_scene \*canvas)**

**{**

**int x = 0, y = 0;**

**double a2 = a \* a;**

**double b2 = b \* b;**

**while (x <= a)**

**{**

**y = (int)round(b \* sqrt(1.0 - x \* x / a2));**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**}**

**while (x <= b)**

**{**

**x = (int)round(a \* sqrt(1.0 - y \* y / b2));**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**}**

**}**

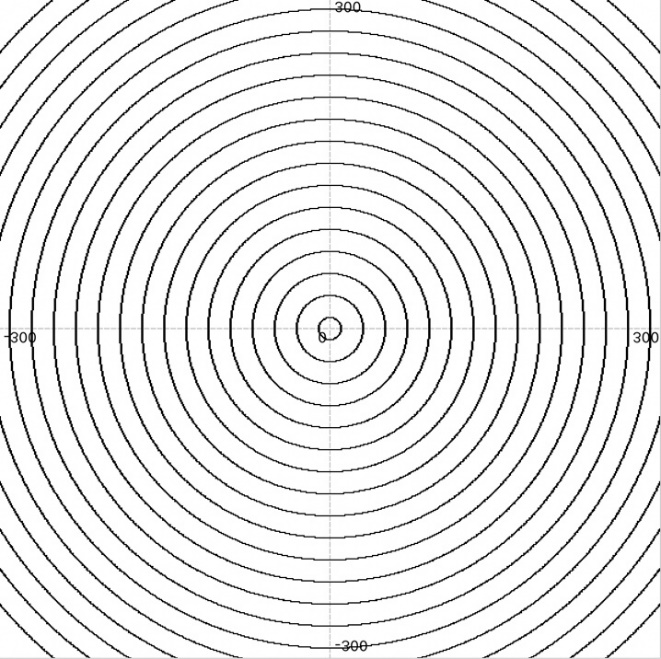
# Параметрическое уравнение

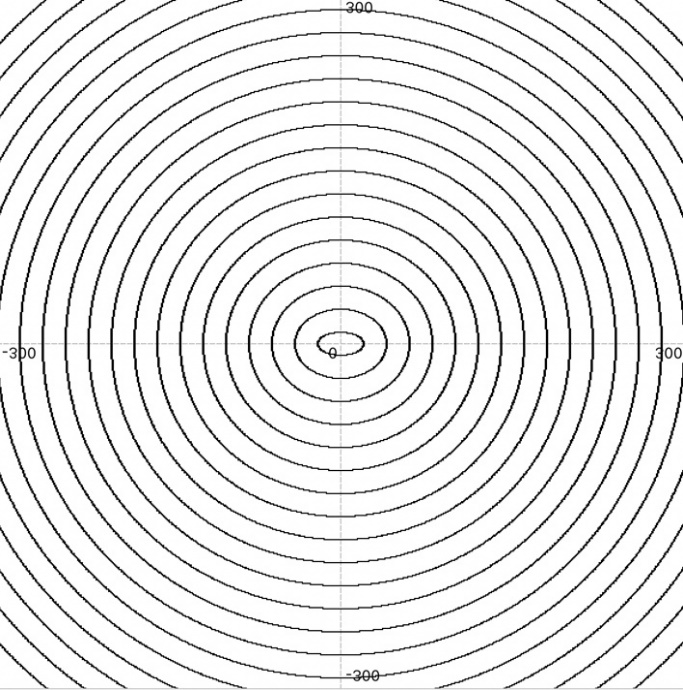
Для окружности:

Для эллипса:

Нужно рассчитать для каждого значения параметра ) значения координат соответствующих точек окружности и соединить их затем отрезками прямых

Также достаточно простой способ, но требует большого количества вычислений.





**Код алгоритма:**

**void param\_circle(int x\_c, int y\_c, int r, graphics\_scene \*canvas)**

**{**

**double d = 1.0 / (double)r;**

**int x, y;**

**double tmp = 0;**

**while (tmp <= M\_PI\_2 + d)**

**{**

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

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

**draw\_pix (x\_c + x, y\_c + y, canvas);**

**draw\_pix (x\_c - x, y\_c + y, canvas);**

**draw\_pix (x\_c + x, y\_c - y, canvas);**

**draw\_pix (x\_c - x, y\_c - y, canvas);**

**tmp += d;**

**}**

**}**

**void param\_ellipse(int x\_c, int y\_c, int a, int b, graphics\_scene \*canvas)**

**{**

**int max\_r = (a > b) ? a : b;**

**double d = 1.0 / (double) max\_r;**

**int x, y;**

**double tmp = 0;**

**while (tmp <= M\_PI\_2 + d)**

**{**

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

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

**draw\_pixel(x\_c + x, y\_c + y, canvas);**

**draw\_pixel(x\_c - x, y\_c + y, canvas);**

**draw\_pixel(x\_c + x, y\_c - y, canvas);**

**draw\_pixel(x\_c - x, y\_c - y, canvas);**

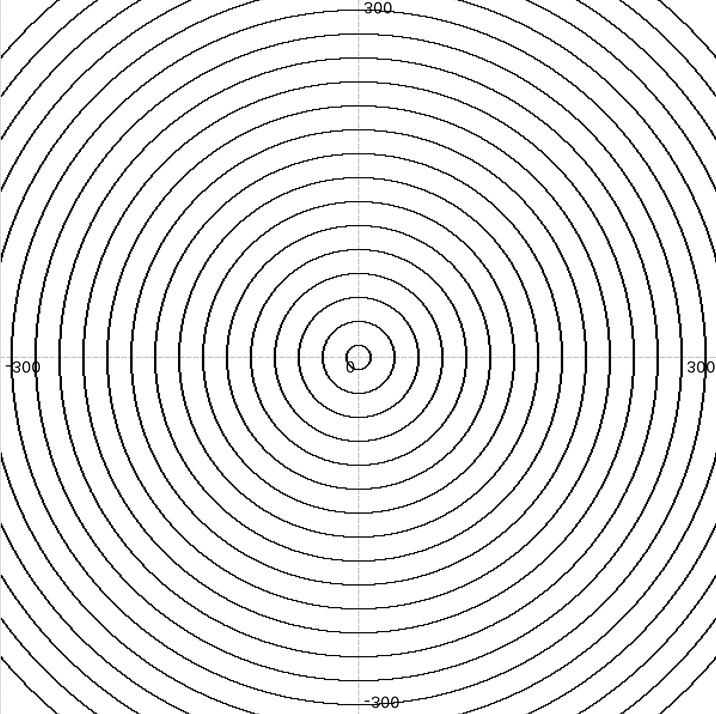
**tmp += d;**

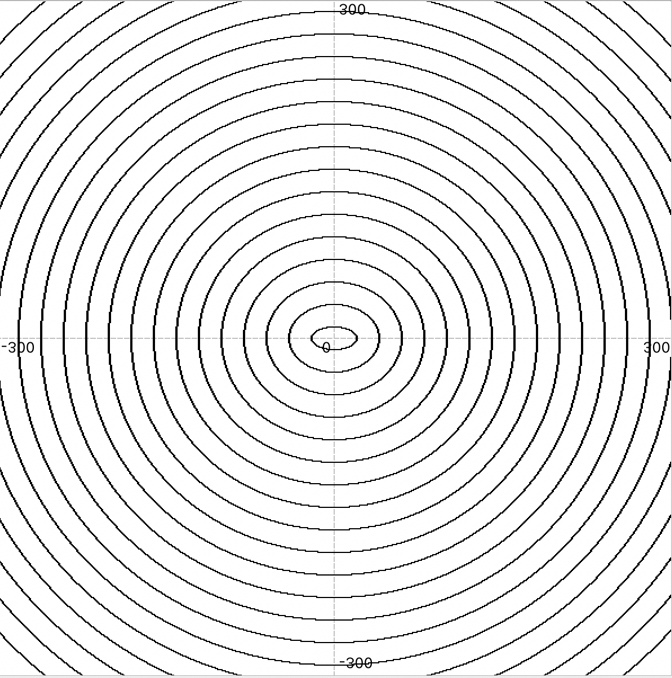
**}**

**}**

# Алгоритм Брезенхема

Алгоритм Брезенхема эффективнее предыдущих, так как для построения достаточно генерировать часть окружности. Остальные части получаются путем симметричного отражения относительно определенной прямой.





**Код алгоритма:**

**void brez\_circle(int x\_c, int y\_c, int r, graphics\_scene \*canvas)**

**{**

**int x = 0, y = r;**

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

**int d1 = 0, d2 = 0;**

**int y\_end = 0;**

**while (y >= y\_end)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (d < 0)**

**{**

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

**x += 1;**

**if (d1 < 0)**

**d = d + 2 \* x + 1;**

**else**

**{**

**y -= 1;**

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

**}**

**}**

**else if (d == 0)**

**{**

**x += 1;**

**y -= 1;**

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

**}**

**else**

**{**

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

**y -= 1;**

**if (d2 < 0)**

**{**

**x += 1;**

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

**}**

**else**

**d = d - 2 \* y + 1;**

**}**

**}**

**}**

**void brez\_ellipse(int x\_c, int y\_c, int a, int b, graphics\_scene \*canvas)**

**{**

**int x = 0;**

**int y = b; // при b = 0 и a != 0 получится точка**

**int a2 = a \* a;**

**int b2 = b \* b;**

**int da2 = 2 \* a2;**

**int db2 = 2 \* b2;**

**int d = b2 - da2 \* b + a2;**

**int d1 = 0, d2 = 0;**

**int y\_end = 0;**

**while (y >= y\_end)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (d < 0)**

**{**

**d1 = 2 \* d + da2 \* y - a2;**

**x += 1;**

**if (d1 < 0)**

**d = d + db2 \* x + b2;**

**else**

**{**

**y -= 1;**

**d = d + db2 \* x - da2 \* y + a2 + b2;**

**}**

**}**

**else if (d == 0)**

**{**

**x += 1;**

**y -= 1;**

**d = d + db2 \* x - da2 \* y + a2 + b2;**

**}**

**else**

**{**

**d2 = 2 \* d - db2 \* x - b2;**

**y -= 1;**

**if (d2 < 0)**

**{**

**x += 1;**

**d = d + db2 \* x - da2 \* y + a2 + b2;**

**}**

**else**

**d = d - da2 \* y + a2;**

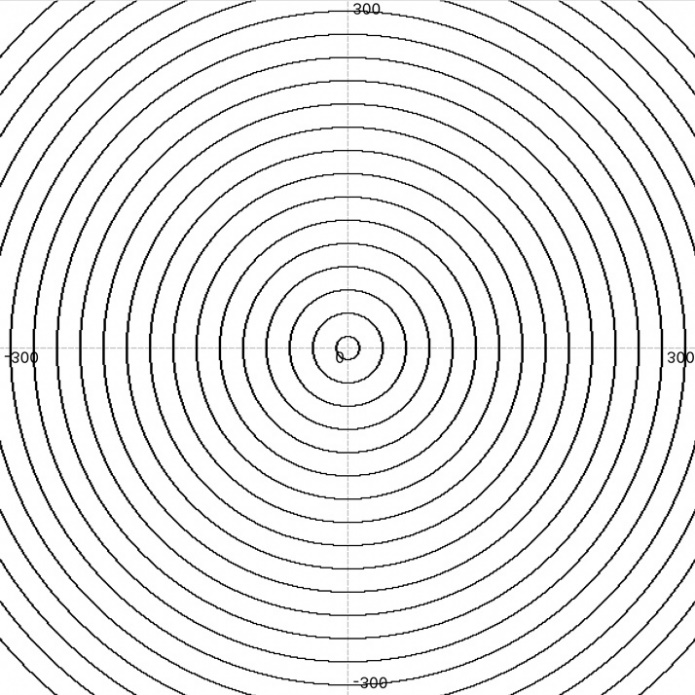
**}**

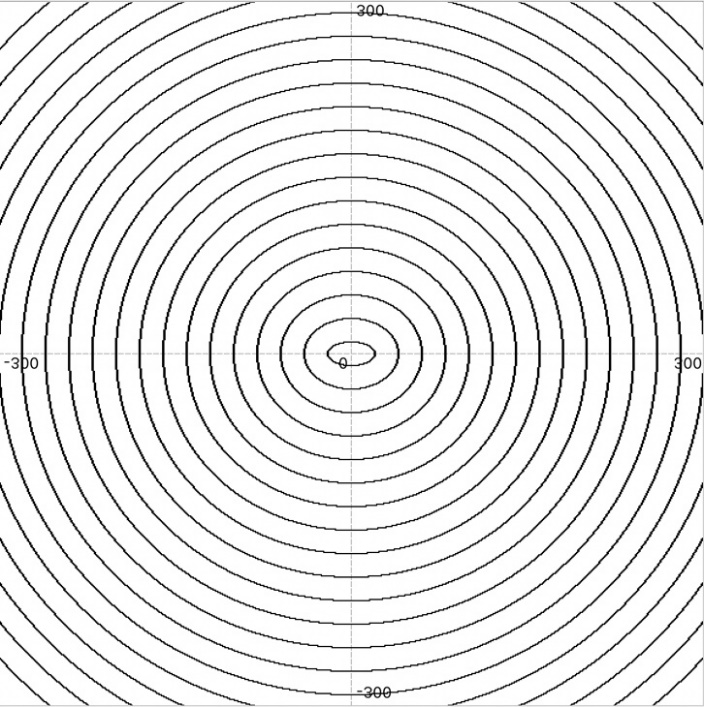
**}**

**}**

# Алгоритм средней точки

По эффективности алгоритм близок к алгоритму Брезенхема. В алгоритме средней точки происходит выборка для средних положений между пикселями вблизи заданной окружности





**Код алгоритма:**

**void midpoint\_circle(int x\_c, int y\_c, int r, graphics\_scene \*canvas)**

**{**

**int x = 0;**

**int y = r;**

**int df = 0;**

**int delta = -2 \* y;**

**int x\_bound = r / sqrt(2);**

**int f = 1.25 - r;**

**for (; x <= x\_bound; x++)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (f >= 0)**

**{**

**y -= 1;**

**delta += 2;**

**f += delta;**

**}**

**df += 2;**

**f += df + 1;**

**}**

**delta = 2 \* x;**

**df = -2 \* y;**

**f += -x - y;**

**for (; y >= 0; y--)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (f < 0)**

**{**

**x += 1;**

**delta += 2;**

**f += delta;**

**}**

**df += 2;**

**f += 1 + df;**

**}**

**}**

**void midpoint\_ellipse(int x\_c, int y\_c, int a, int b, graphics\_scene \*canvas)**

**{**

**int a2 = a \* a;**

**int b2 = b \* b;**

**int da2 = 2 \* a2;**

**int db2 = 2 \* b2;**

**int x = 0;**

**int y = b;**

**int df = 0;**

**int delta = -da2 \* y;**

**int x\_bound = a2 / sqrt(a2 + b2);**

**int f = b2 - a2 \* b + 0.25 \* a2;**

**if (b == 0)**

**f = -1;**

**for (; x <= x\_bound; x++)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (f >= 0)**

**{**

**y -= 1;**

**delta += da2;**

**f += delta;**

**}**

**df += db2;**

**f += df + b2;**

**}**

**if (a == 0)**

**x = 0;**

**delta = db2 \* x;**

**df = -da2 \* y;**

**f += 0.75 \* (a2 - b2) - a2 \* y - b2 \* x;**

**for (; y >= 0; y--)**

**{**

**draw\_pix(x\_c + x, y\_c + y, canvas);**

**draw\_pix(x\_c - x, y\_c + y, canvas);**

**draw\_pix(x\_c + x, y\_c - y, canvas);**

**draw\_pix(x\_c - x, y\_c - y, canvas);**

**if (f < 0)**

**{**

**x += 1;**

**delta += db2;**

**f += delta;**

**}**

**df += da2;**

**f += a2 + df;**

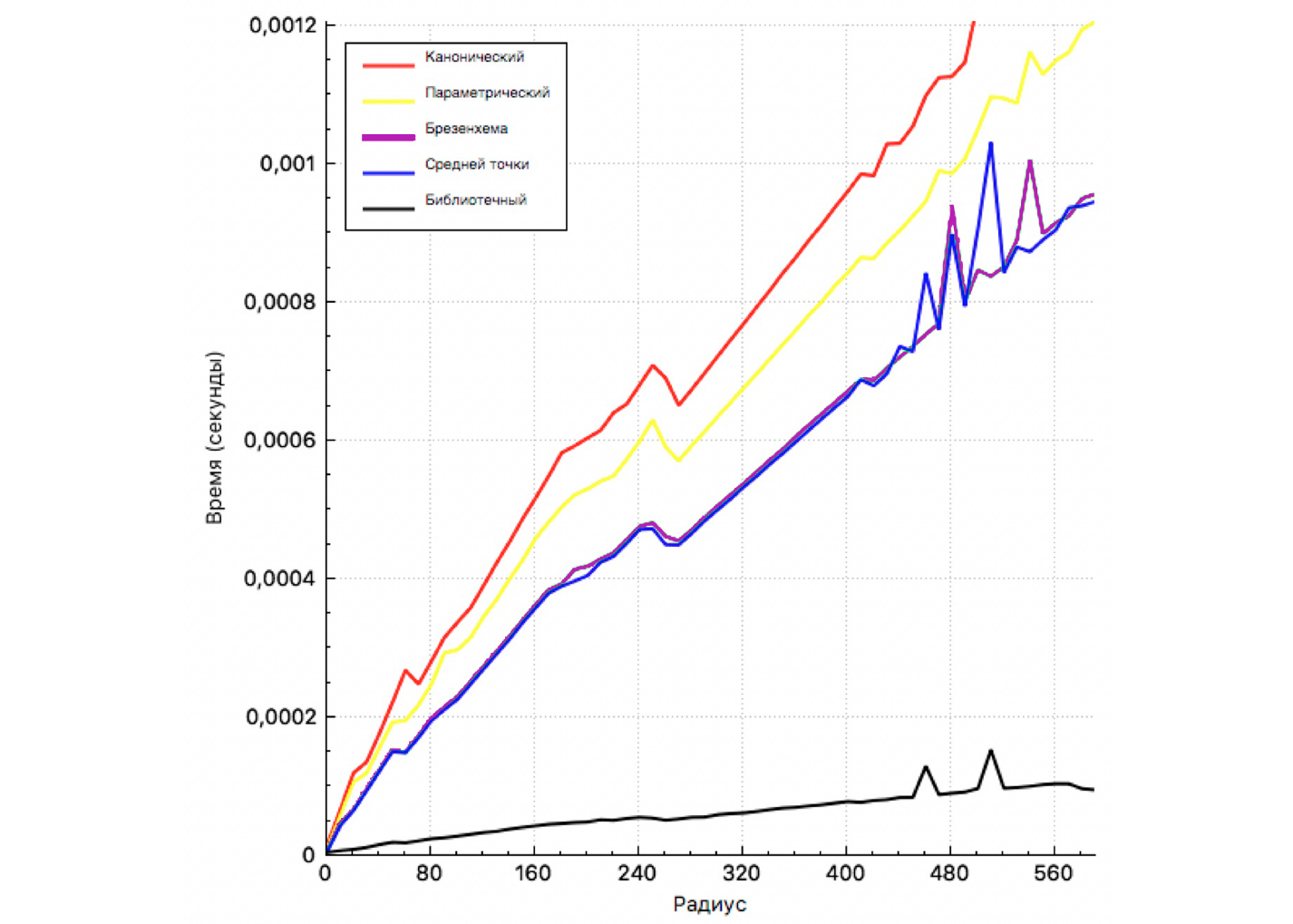
**}**

**}**

# Анализ времени выполнения алгоритмов:

Для окружностей:

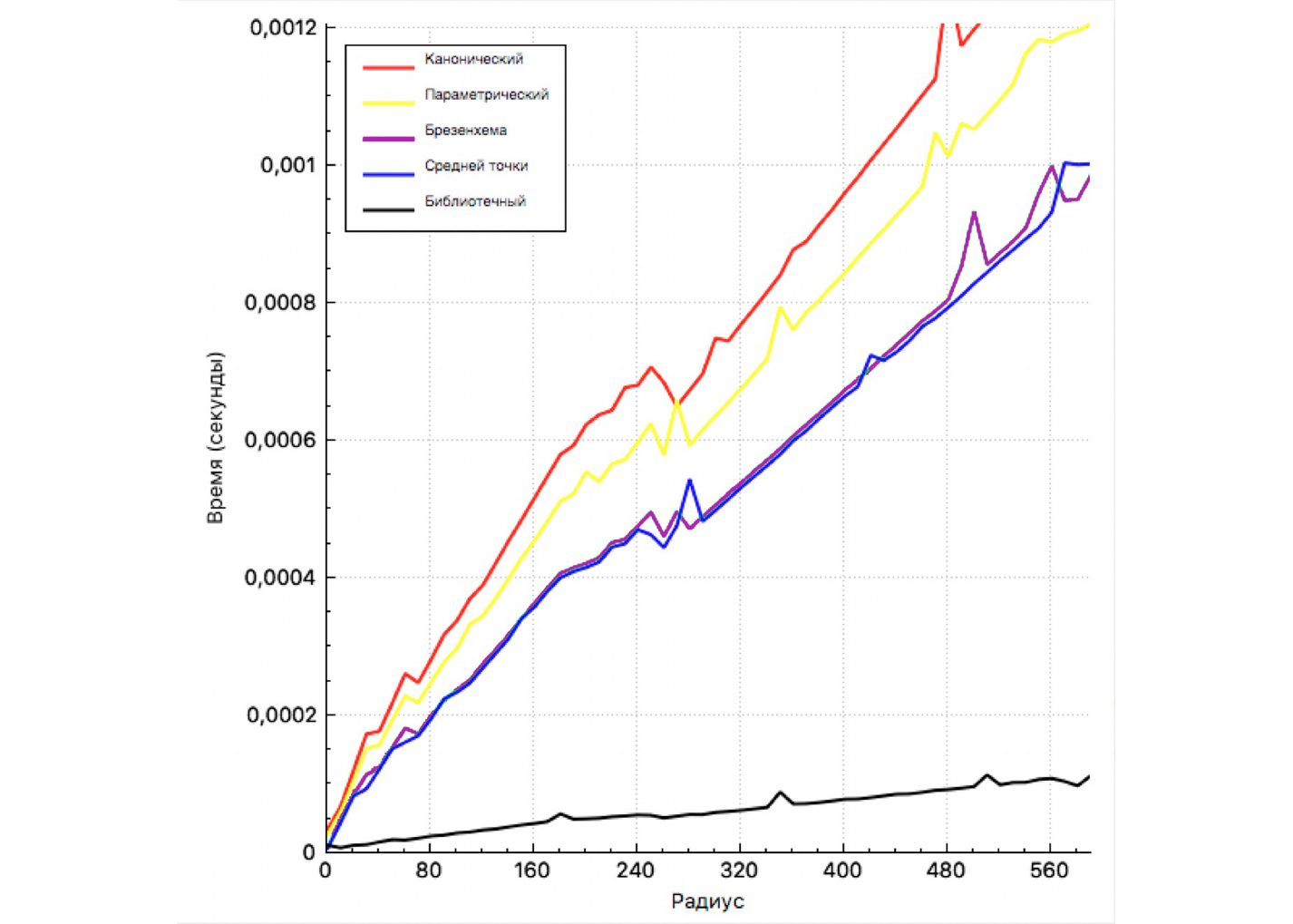
Радиус изменяется от 10 до 620 с шагом 10.



Рисование окружности с помощью уравнения занимает больше всего времени. Время работы алгоритма Брезенхема и алгоритма средней точки практически совпадают.

Для эллипса:

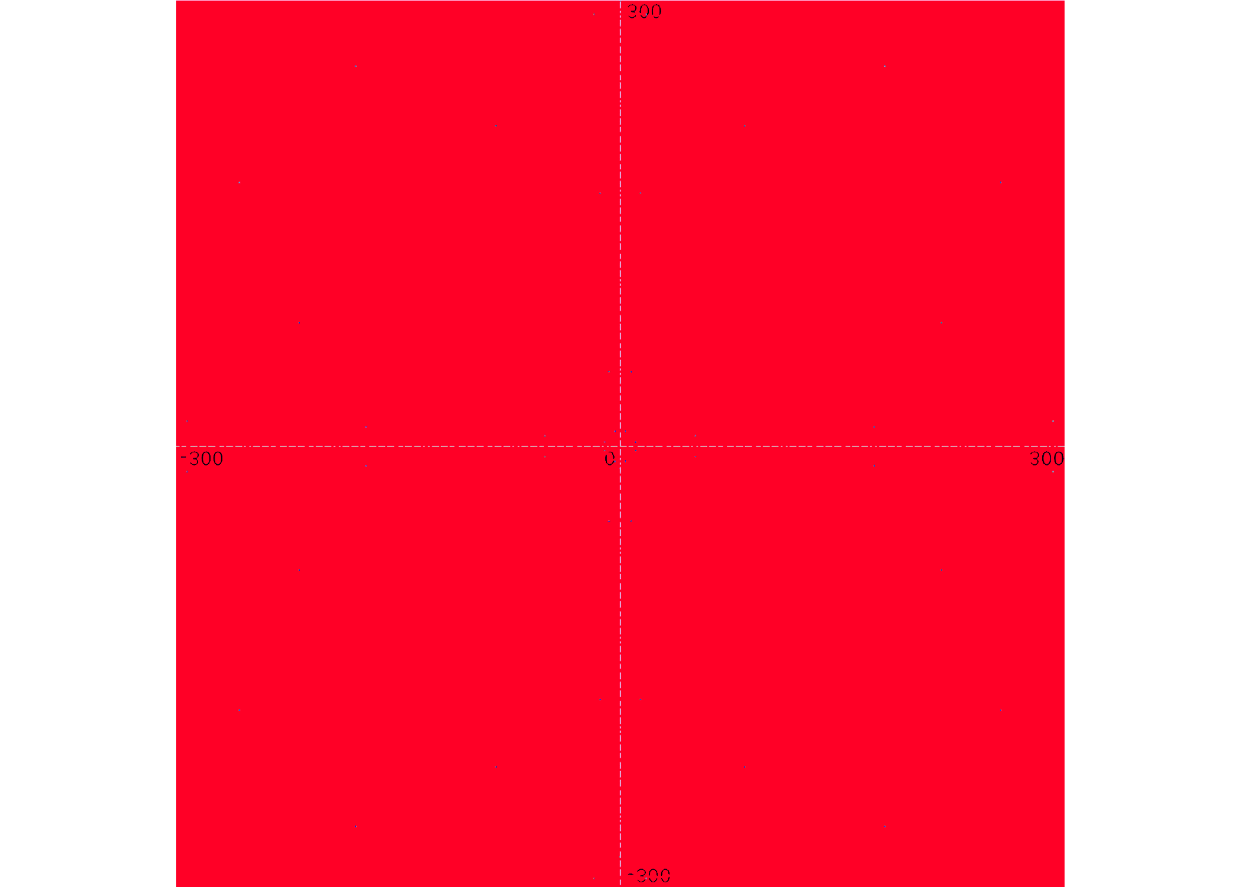
Параметр А изменяется от 20 до 620 c шагом 10, а параметр B – от 20 до 1200 с шагом 20.



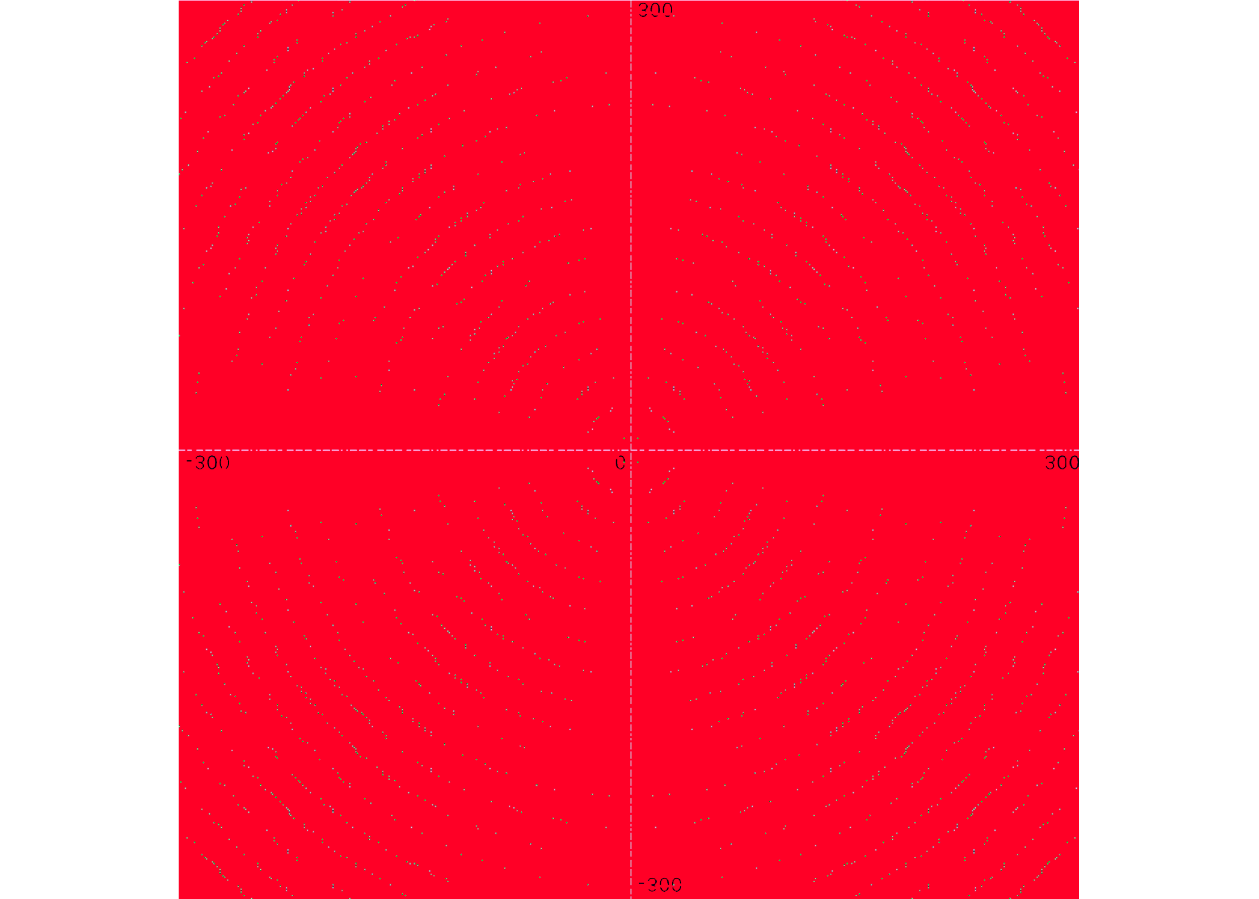
Результаты похожи

# Исследование визуальных характеристик

Если построить спектр окружностей сначала алгоритмом средней точки сигим цветом, затем сверху наложить такой же спектр красного цвета фона, построенный с помощью алгоритма Брезенхема, то он практически полностью перекроет синий спектр. Можно заметить редкие пикселы синего цвета



Если проделать аналогичные действия со спектрами, построенным с помощью канонического и параметрического уравнений, то несовпадений будет больше



# Интерфейс программы

