|  |  |
| --- | --- |
| **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** Челядинов Илья  **Группа ИУ 7-43**  **Оценка (баллы)**  **Преподаватель Куров А. В.** |  |

Москва.

2020 г.

**Цель работы:**

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

**Техническое задание:**

1. Реализовать алгоритмы построения окружности на основе:

1. Канонического уравнения.

2. Параметрического уравнения.

3. Алгоритма Брезенхема.

4. Алгоритма средней точки.

5. Библиотечным методом.

1. Реализовать алгоритмы построения эллипса на основе:

1. Канонического уравнения.

2. Параметрического уравнения.

3. Алгоритма Брезенхема.

4. Алгоритма средней точки.

5. Библиотечным методом.

1. Сравнение визульных характеристик разных алгоритмов путем рисования спектра концентрических окружностей и эллипсов.
2. Исследование временных характеристик алгоритмов.

**Теоретический материал:**

Эллипс можно описать с помощью канонического уравнения (1):

*x2/a2 + y2/b2 = 1.*

Так же эллипс можно описать с помощью параметрических уравнений (2):

*x = a \* cos(t)*

*y = b \* cos(t)*

(Данные уравнения приведены для эллипса с центром в (0; 0). Для переноса центра в уравнении 1 нужно заменить *x → (x - x0),* а в системе уравнений 2 - прибавлять координаты центра (*a \* cos(t) → x0 + a \* cos(t)))*

Окружность — это эллипс, полуоси которого равны. Таким образом, если домножить уравнение 1 на полуось (которая теперь одна (ну или 2 одинаковые)), то можно получить уравнение окружности:

*x2 + y2 = R2*

C параметрическими уравнениями окружности поступаем так же (полуось теперь одна, поэтому исходя из уравнений эллипса (2) имеем:)

*x = R \* cos(t)*

*y = R \* cos(t)*

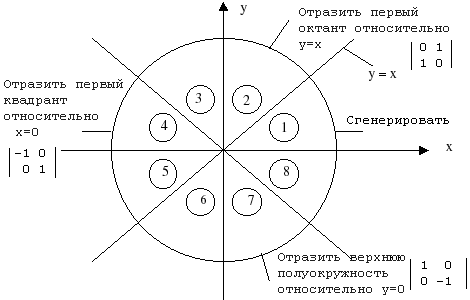
(Для переноса центра окружности в произвольную точку требуется сделать преобразования, аналогичные преобразованиям в эллипсе)

Однако в (1) случае, чтобы выразить одну из координат, нам потребуется искать квадратный корень (целочисленной величины), что, учитывая, что чаще всего он ищется путем разложения в ряд, довольно-таки затратно в плане вычислительных ресурсов. А во (2) случае искать sin и cos тоже не очень быстро, в связи с чем были придуманы отдельные алгоритмы (подробнее о каждом — ниже).

**1. Алгоритм построения окружности на основе канонического уравнения.**

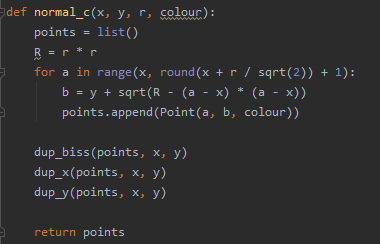
Идея данного алгоритма проста: имея каноническое уравнение окружности можно выразить одну из координат (то есть получить *у = …)*, и далее, устанавливая приращение аргумента = 1, находить в каждой точке значение функции. Пусть для определенности мы выражаем *у.* Тогда, учитывая, что у должен иметь меньшее приращение (чтобы шаг по х, равный единице, не приводил к прерываниям функции), нам следует чертить только ту часть окружности, которая имеет наклон <= 45o (по модулю).

С данной проблемой можно легко справиться, потому что у окружности есть одно большое **преимущество** **(которое мы везде будем использовать)**:

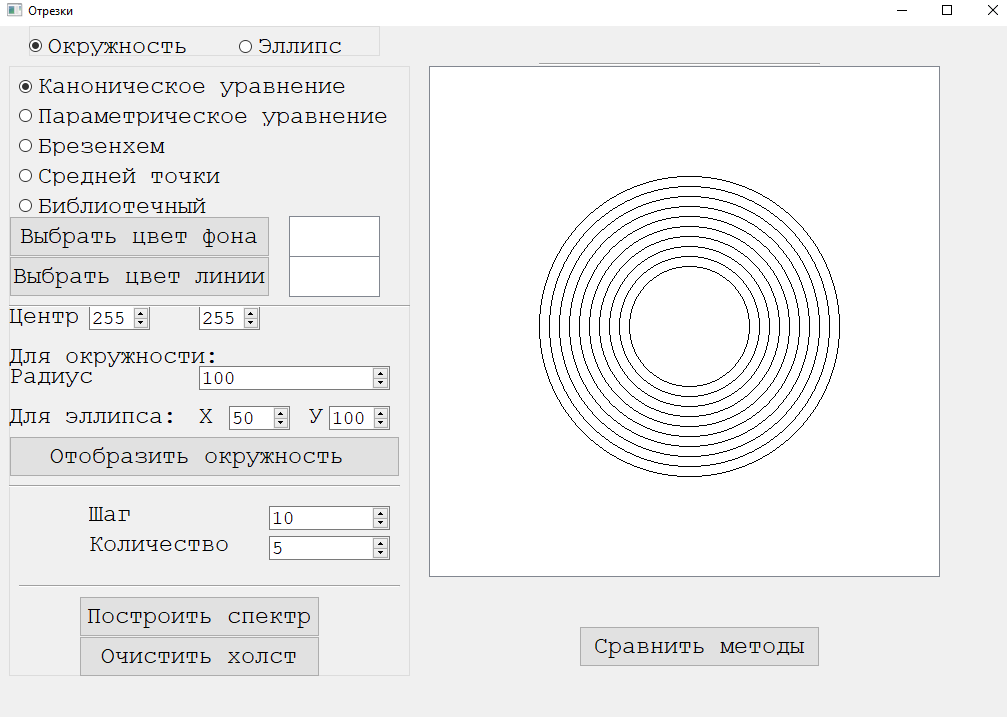
Нам достаточно найти только 1/8 часть окружности, а всю окружность целиком можно получить путем 3 отражений: вокруг биссектрисы 1 четверти, вокруг оси Х и вокруг оси У (данные отражения указаны для окружности, лежащей в начале координат; если требуется отразить не в начале координат, то биссектриса и оси сдвинутся на величину сдвига центра фигуры). Ниже приведена картинка, как можно генерировать окружность.

В моем алгоритме я выражал *у*, а учитывая ограничения, накладываемые на приращение, я выбрал (2) кусок окружности (см. рисунок выше) для генерации, а все остальные получал путем трекратного отражения.

Мой код:



Пример работы:

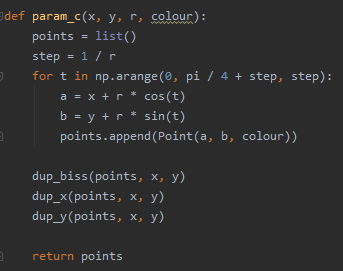


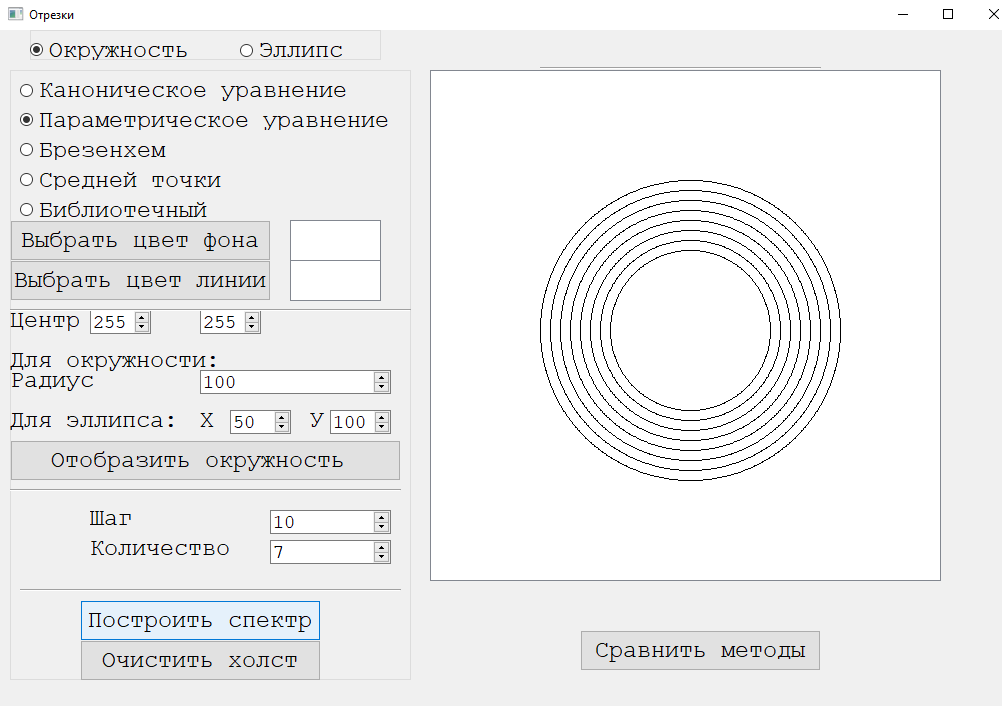
**2. Алгоритм построения окружности на основе системы параметрических уравнений.**

Идея данного алгоритма в следующем: мы выбираем некоторый шаг, равный 1/R, для переменной t, и скаждым новым вычислением увеличиваем t на эту величину. Данная величина была выбрана неслучайно: по 1 замечательному пределу *x / sin(x) = 1*  и получается, что расстояние между рисуемыми пикселями пропорционально углу между ними (с вершиной в центре окружности). Так, при окружности радиусом 1, шаг угла будет = 1 радиан, расстояние между пикселями = 1 ширина пикселя; при увеличении радиуса в R раз, угол должен быть в R раза меньше, так расстояние между пикселями будет расти пропорционально этому радиусу (см. определения угла в 1 радиан).

В данном случае я снова чертил 1/8 окружности и далее отражал, для экономии времени (то есть получается угол будет меняться от 0 до pi / 4 с шагом 1 / R):

Мой код:



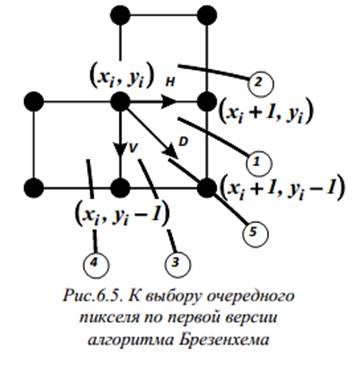


Пример работы:

**3. Алгоритм Брезенхема построения окружности.**

В данном алгоритме окружность строится путем рассмотрения возможных ситуаций прохода прямой и, исходя из этого положения, рассмотрения расстояния до ближайших пикселей.

Рассмотрим алгоритм подробнее:

В данном алгоритме, мы имеем некоторый текущий пиксел (xi; yi). Перед нами стоит задача выбора следующего пиксела, а возможные варианты прохода прямой, исходя из выбранного на данный момент пиксела указаны на рисунке (заметим также, что рассматриваются только 3 пиксела; на рисунке даны описания возможных направлений движения: h(orizontal), v(ertical), d(iagonal))

В первую очередь Брезенхем ввел величину Δi = (xi + 1)2 + (yi - 1)2 - R2.

Если посмотреть на эту величину повнимательнее, то можно заметить, что *(xI  + 1)*2 *+ (yi - 1)*2 *—* есть квадрат расстояния от вершины окружности до диагонального пикселя (из рассматриваемых, при учете, что рассматривается i-ый), а R2 — квадрат расстояния до нашей окружности. Таким образом, если данная величина отрицательная, то диагональный пиксел вне окружности и это случаи 3, 4. Если величина — положительная, то это случаи 1, 2. Если = 0, то случай 5.

Разделим рассмотрение групп случаев:

1) случаи 1, 2. Так как в данных случаях можно не рассматривать вертикальный пиксел, вводится следующая величина:

d1 = |(xi + 1)2 + yi 2 - R2| - |(xi + 1)2 + (yi - 1)2 - R2|

Присмотревшись получше к данным выражениям можно заметить, что 1 модуль — расстояние между горизонтальным пикселем и нашей окружность, а 2ой модуль — расстояние между диагональным пикселем и окружностью. Таким образом если величина отрицательная, то расстояние до горизонтального пиксела меньше, иначе — до диагонального меньше (соответственно, если величина = 0, то расстояния одинаковые и можно брать любой).

Однако работа с модулями и возведение большого количества величин в квадрат сильно усложняет задачу, поэтому для 2 случаев рассматривается приведенная данная величина (без модулей (раскрываем исходя из положения нашей окружности относительно пикселей) и без возможности сокращения). Таким образом будем иметь:

Случай 1: d1 = 2 (Δi + yi) - 1 (выбирается пиксел исходя из описания выше)

Случай 2: d1 = 2 yi + 1 < 0 (выбирается горизонтальный пиксел, см. описание выше)

Примечательно, что случай 2 можно не рассматривать отдельно (на практике, в коде), потому что при расположении пикселей как в случае 2, выражение для случая 1 всегда будет < 0, то есть как и выражение для случая 1 (Однако здесь мы были вынуждены проверить данный случай, так как была вероятность того, что случай 1 не обработал бы случай 2, поэтому в качестве доказательства мы провели все вычисления и убедились в этом).

2) случаи 3, 4. Так как в данном случае можно не рассматривать горизонтальный пиксел вводится следующая величина:

d2 = |(xi + 1)2 + (yi - 1)2 - R2| - |xi2 + (yi - 1)2 - R2|

Опять же видно, что здесь первый модуль — расстояние до диагонального пиксела, 2ой модуль — расстояние до вертикального пиксела (горизональный можно не рассматривать).

Таким образом если величина отрицательная, то расстояние до диагонального пиксела меньше, иначе — до вертикального меньше (соответственно, если величина = 0, то расстояния одинаковые и можно брать любой).

Работа с модулями повышает вычислительные расходы, соответсвенно делаем то же, что в пункте (1):

случай 3: d2 = 2 (Δi + xi) - 1

случай 4: d1 = 2 xi + 1 > 0

Рассмотрение случая 4 так же можно включить в случай 3 (как в пункте (1)). Мы рассматривали этот случай отдельно по тем же причинам.

3) Случай 5. Его можно вынести отдельно (выбор очевиден, диагональный пиксел лежит на окружности), однако лучше будет проверить, действительно ли в такой конфигурации пункты (1) и (2) будут выбирать диагональный пиксел. Тогда этот случай можно будет рассматривать вместе с пунктами (1) и (2) и он не будет исключительным.

Проверяем:

d1 = |(xi + 1)2 + yi 2 - R2| > 0 — величина положительная, диагональный пиксел

d2 = - |xi2 + (yi - 1)2 - R2| < 0 — величина отрицательная, диагональный пиксел

Таким образом все сводится к проверке сначала Δi , а за ней d1 или d2, исходя из значения Δi .

Единственное, что остается сделать — проверить, какой будет Δi+1, исходя из значений Δi, xi+1, yi+1.

Считать значение Δi+1 каждый раз с нуля неудобно и долго, поэтому можно просто найти ее приращение при горизонтальном, вертикальном и диагональном шагах ( Δi+1 (xi+1, yi+1) - Δi (xi, yi) ). Это делается легко путем явной подстановки величин xi+1, yi+1, выраженных через xi, yi (это тоже выразить легко, потому что известно направление шага).

Горизонтальный шаг:

xi+1 = xi + 1; yi+1 = yi ;

di+1 = di + 2 xi+1 + 1

Вертикальный шаг:

xi+1 = xi ; yi+1 = yi + 1;

di+1 = di - 2 yi+1 + 1

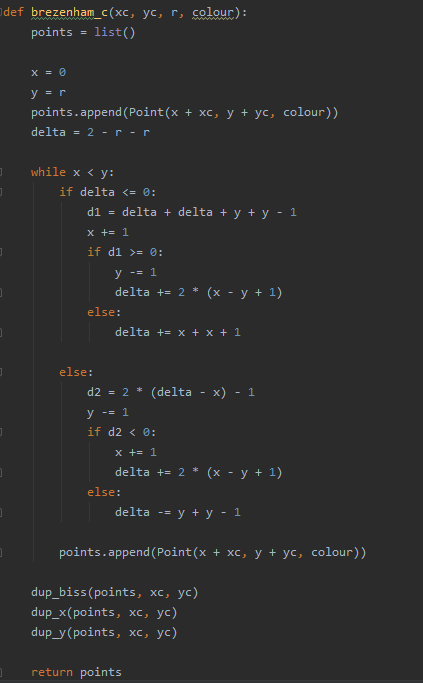
Диагональный шаг:

xi+1 = xi + 1; yi+1 = yi + 1;

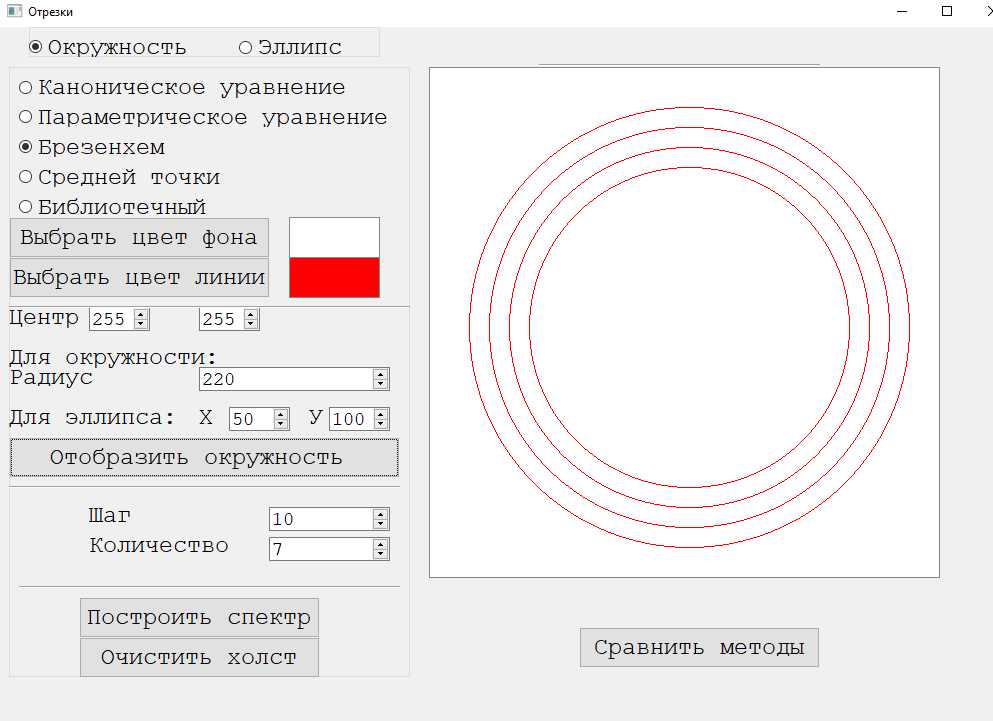
di+1 = di + 2 xi+1 - 2 yi+1 + 2.

Таким образом, мы сделали достаточно дополнительных мат. преобразований, чтобы избавиться от операций возведения в квадрат и взятия модуля. Описанных выше преобразований достаточно для написания кода.

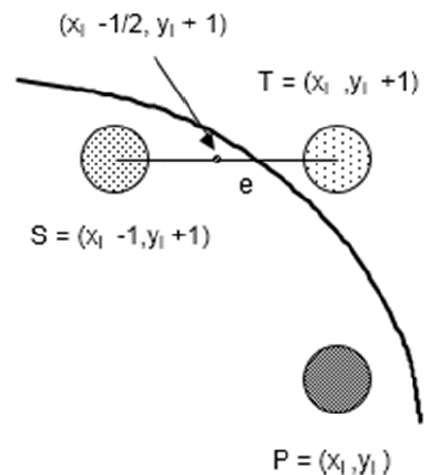
Код:



Пример работы программы:



**4. Алгоритм средней точки построения окружности.**

В моей реализации алгоритма средней точки построение начиналось с точки (R; 0) и шло ровно половину 1ой четверти (1/8 окружности). Ситуация выбора следующей точки нарисована на картинке: Шаг по у = 1, будет ли шаг по х или нет — определяется положением средней точки: если средняя точка лежит внутри окружности (как на рисунке), то истинной окружности ближе вертикальный пиксел и шага по х нет, иначе — ближе диагональный пиксел и шаг по х = -1.

Очевидно, что для корректного построения *непрерывной* окружности требуется, чтобы приращения у было больше х — у меня данное условие в 1ой половине 1ой четверти выполняется.

В данном алгоритме вводится следующая функция:  
Pk = (xk  - 0.5)2+ (yk  + 1)2  - R2

где хk, yk — координаты пиксела, выбранного на последнем шаге. Если присмотреться, то можно увидеть, что данная функция содержит разность квадрата расстояния от центра окружности до «средней точки» рассматриваемого на текущий момент пиксела и квадрата расстояния до идеальной окружности. Получается, что если расстояние до средней точки будет больше, то функция принимает положительное значение и выбирается диагональный пиксел (средняя точка вне окружности).

Получаем следующее условие:

Pk < 0 → верхний, иначе — диагональный (знак может быть и <, и <=)

расчет каждый раз величины данной функции очень емок в плане ресурсов, поэтому аналогично алгоритму брезенхема была найдена величина «приращения» данной функции:

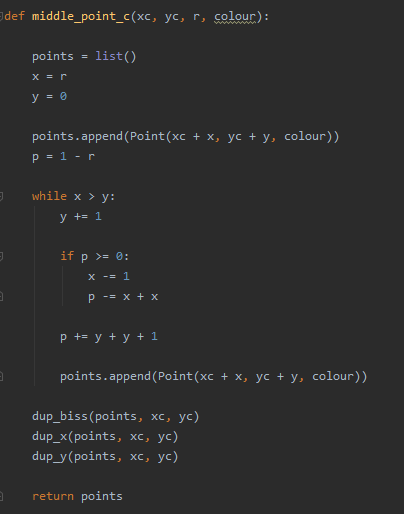
Δ = Pk+1  - Pk = 2 (yk + 1) + 1 — при вертикальном шаге;

Δ = Pk+1  - Pk = 2 (yk + 1) + 1 - 2 (хk - 1) — при диагональном шаге.

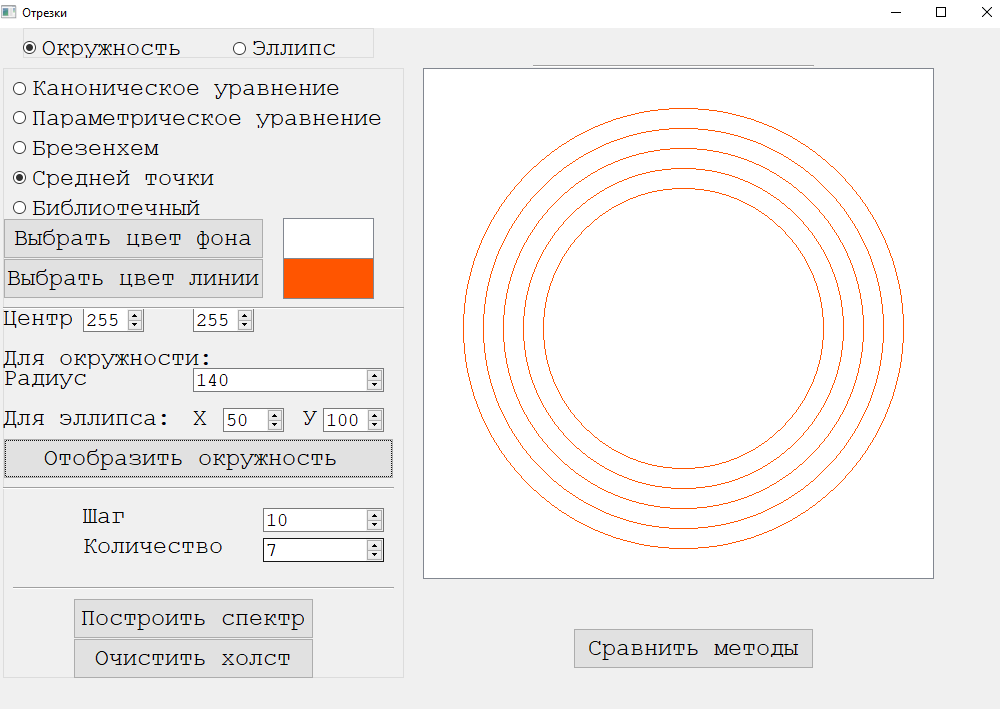
Таким образом: Pk+1  = Pk + Δ

Данный алгоритм быстрее, чем алгоритм брезенхема, в связи с тем, что анализирует меньше ситуаций, однако за счет расширенного анализа алгоритм Брезенхема может строить четверть окружности, а алгоритм средней точки — лишь 1/8 (однако в случае с окружностью нужно строить 1/8, поэтому средняя точка лучше; да и с эллипсом не так сложно построить эллипс за 2 этапа, при этом на каждом шаге анализируя на один пиксел меньше, поэтому и там средняя точка работает лучше).

Код:



Пример работы:



**1. Алгоритм построения эллипса на основе канонического уравнения.**

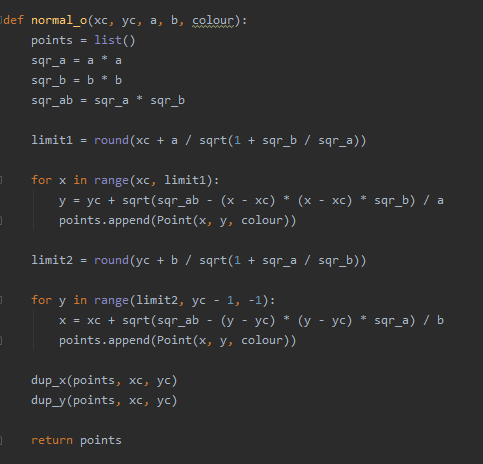
Алгоритм построения эллипса на основе канонического уравнения похож на аналогичный алгоритм для окружности, однако есть **важные** различия:

1. При построении эллипса требуется строить ¼, а не 1/8, так как эллипс не имеет симметрии относительно биссектрисы 1 четверти (в отличие от окружности).

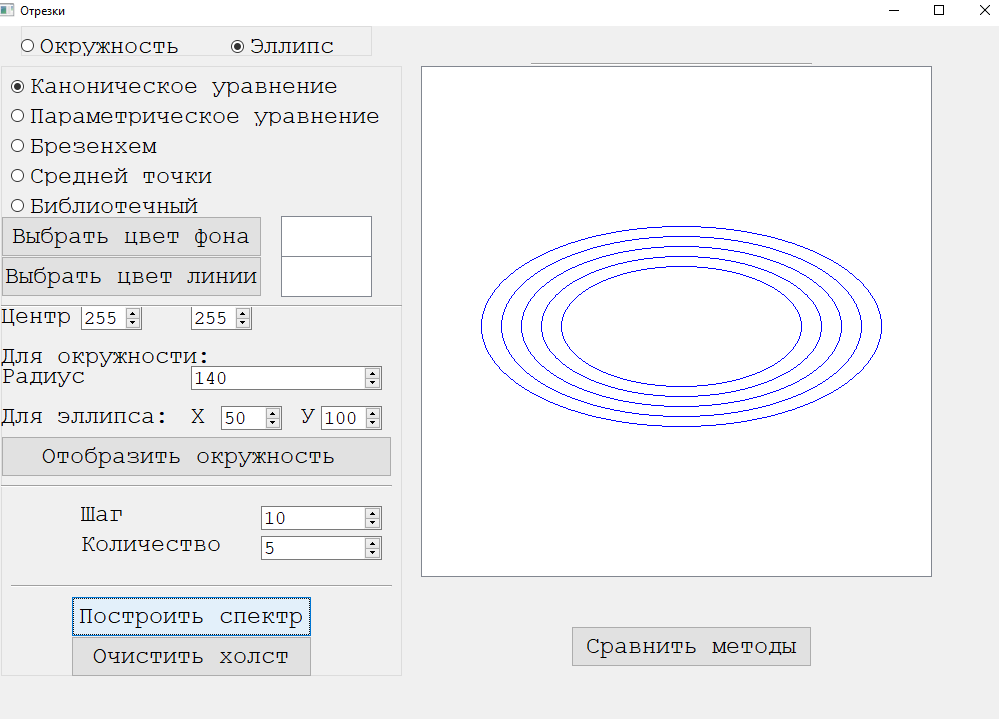
2. В связи с надобностью строить ¼, появляется необходимость находить точку, в которой наклон касательной к эллипсу становится меньше 45о. Мы вынуждены найти эту точку, так как в ней приращения переменных х и у «меняются местами»: та переменная, которая имела большее приращение, начинает расти меньше, а другая — получает большее приращение. Соответственно в данной точке требуется другой переменной задать шаг = 1.

3. Каноническое уравнение эллипса содержит знаменатели под х и у, поэтому при преобразовании потребуется дополнительной домножить на знаменатель переменной, которую мы хотим выразить.

Код:



Пример работы:



**2. Алгоритм построения эллипса на основе параметрического уравнения.**

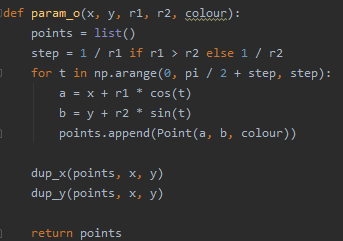
Идея данного алгоритма такая же, как у алгоритма с окружностью, однако нам требуется (как и в алгоритме выше) строить ¼ часть фигуры вместо 1/8, а также появляется проблема с выбором шага, у которой есть 2 не самых лучших решения:

1. Выбрать больший радиус. В таком случае при большой разнице между полуосями мы построим много лишний точек.

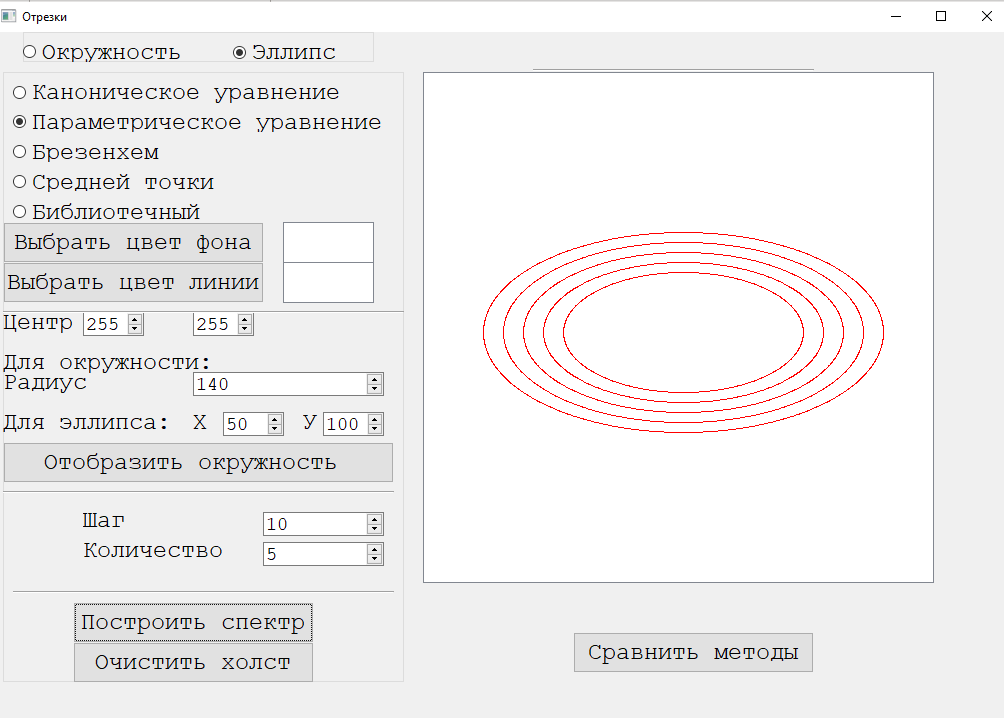
2. Каждый раз высчитывать шаг исходя из текущего угла. Данное действие тоже весьма трудоемкое и при больших значениях полуосей эллипса (то есть при большом количестве точек) будет сильно потреблять вычислительные ресурсы.

Возможно есть и другие, более эффективные решения, однако данный алгоритм является также самым не точным (помимо того, что самый медленный), поэтому ускорять его нет большого смысла (он все равно будет работать медленнее алгоритмов с отсутствием разложений функций в ряд, возведений в квадрат и работой в поле вещественных чисел).

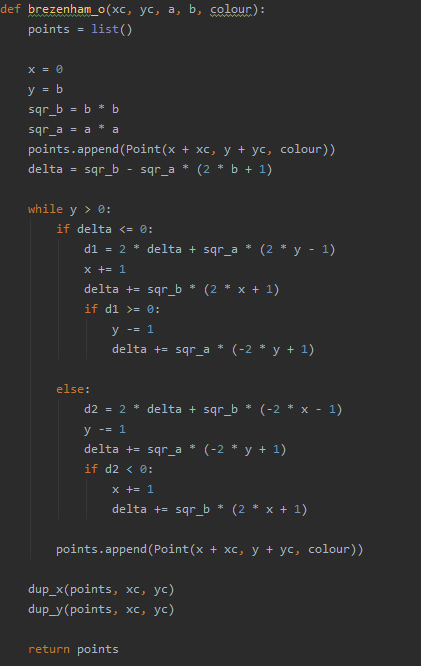
Код:



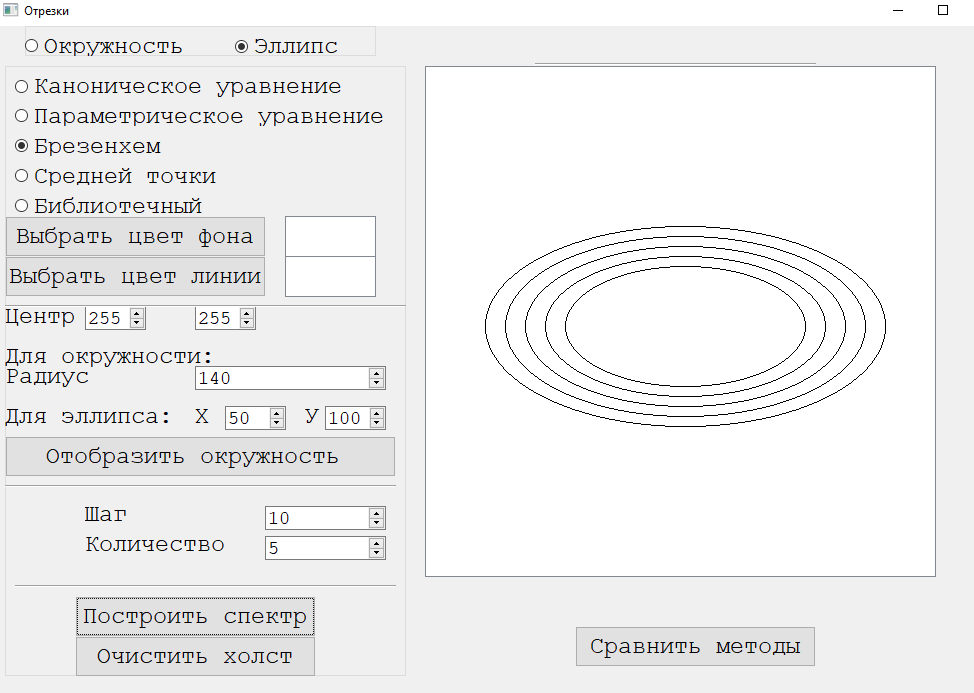
Пример работы:



**3. Алгоритм Брезенхема построения эллипса.**

Идея алгоритма точно такая же, как идея алгоритма построения окружности, отличие лишь в том, что при вычислении всех расстояний использовалось каноническое уравнение эллипса (домноженное на знаменатели под х и у).п.с.: так как алгоритм брезенхема анализирует 3 пиксела, мы легко можем построить ¼ эллипса в первой четверти за 1 цикл.Мой код: 

Пример работы:

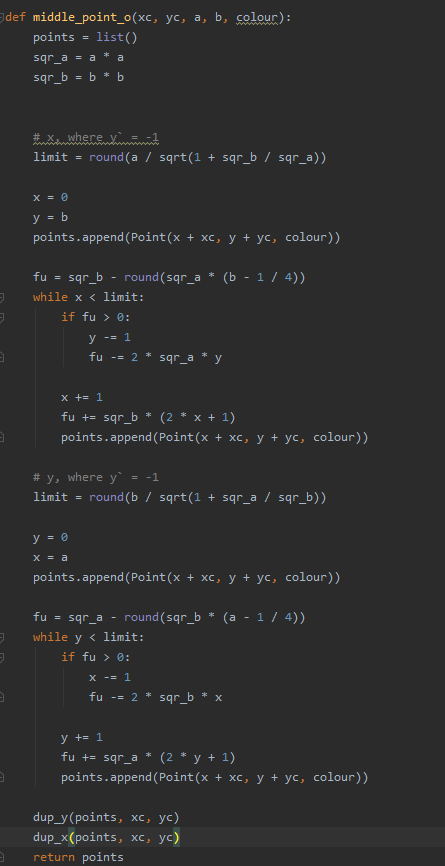


**4. Алгоритм средней точки построения эллипса.**

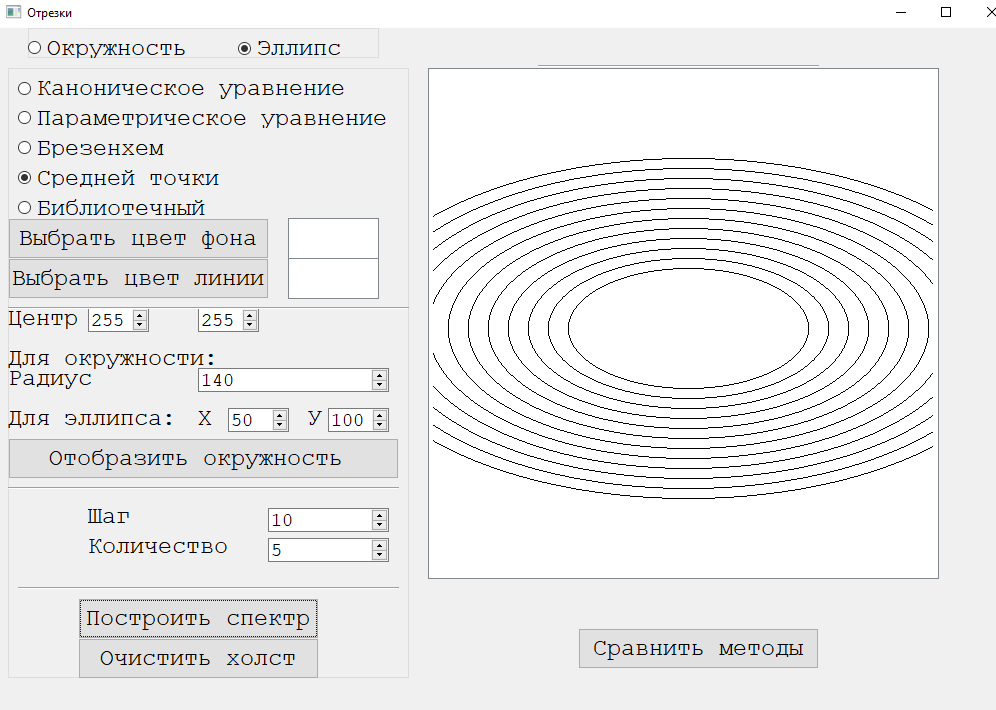
Идея алгоритма та же, расчеты — те же, только с другим уравнением (каноническим уравнением эллипса, домноженным на знаменатели).

Важное отличие заключается в том, что мы, как и в алгоритме на основе канонического уравнения эллипса, вынуждены находить точку, в которой касательная к эллипсу проходит под углом в 45о. Это необходимо из-за того, что алгоритм средней точки выбирает из 2 пикселей и там есть то же условие, что и в каноническом уравнении (одно приращение всегда больше другого). Поэтому алгоритм прогоняется 2 раза: для интервала, где касательные проходят под меньшим углом, и где под большим (сравнение с 45о).

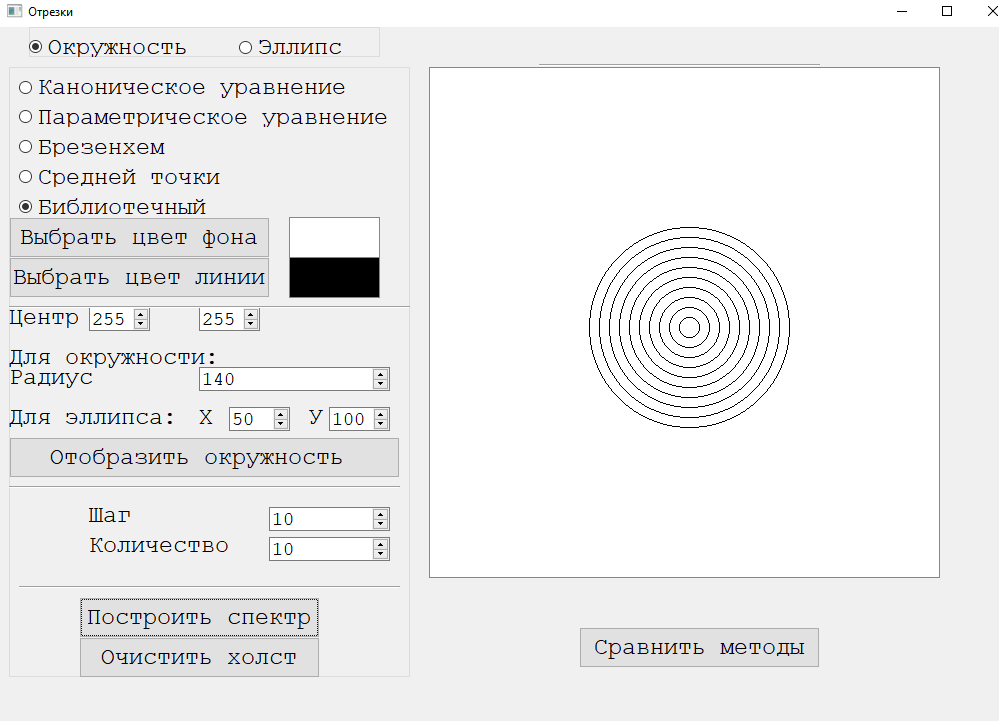
Код:

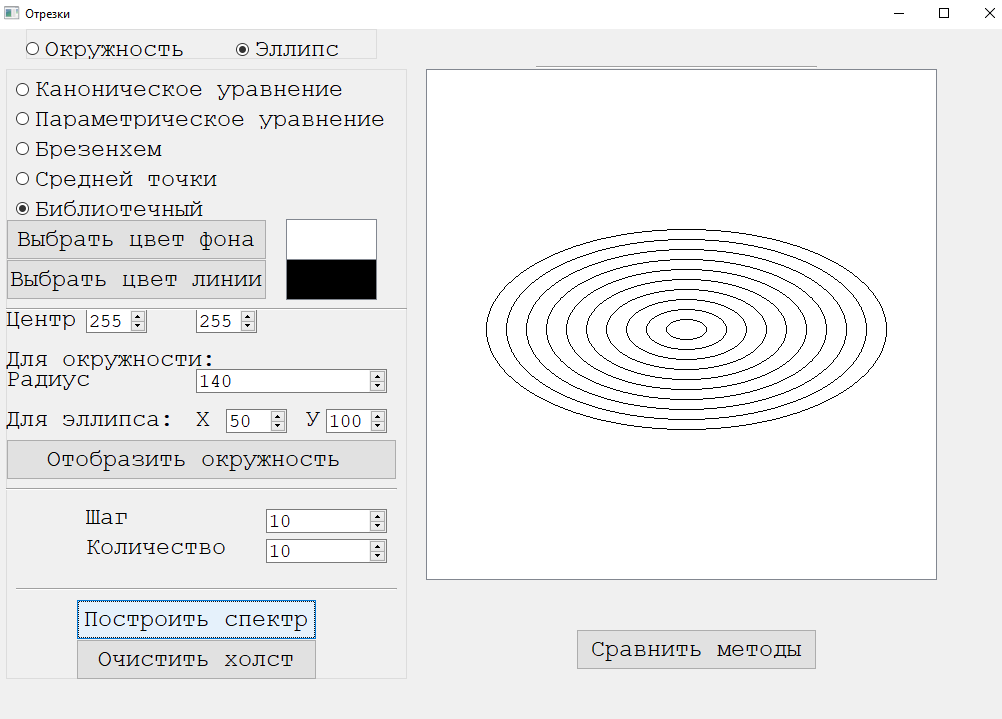


Пример работы программы:



**Библиотечные методы.**





**Сравнение визуальных характеристик.**

Примечание: скриншоты полных наложений прикреплять не буду (просто одноцветное полотно) в связи с отсутствием информативности.

**1**. **Окружность**

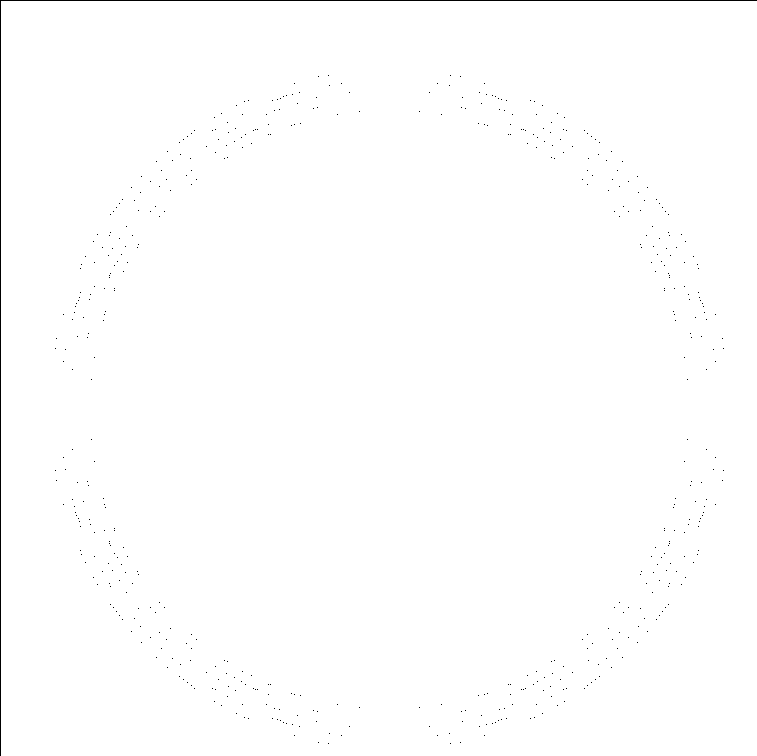
Из реализованных мною 4 алгоритмов 3 выдают абсолютно одинаковые окружности (библиотечная функция реализована одним из этих методов, так как дает такой же результат):

1) Каноническое уравнение.

2) Брезенхем.

3) Средняя точка.

Параметрическое уравнение выдает другой результат (результат наложения параметрического уравнение на один из 3 алгоритмов выше):

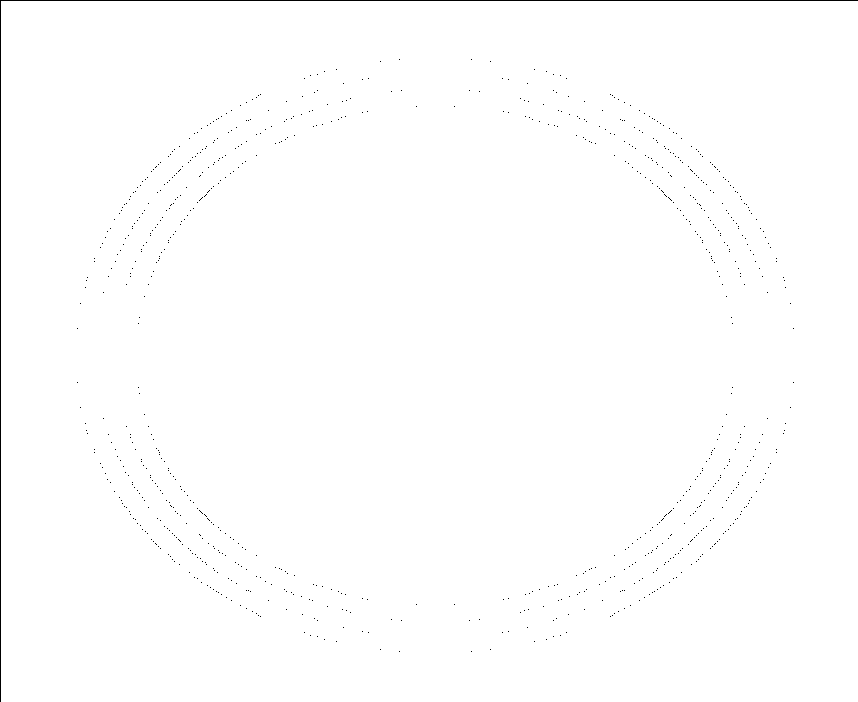


(на рисунке 5 концентрических окружностей)

Данный результат связан с большой погрешностью метода, основанного на параметрическом уравнении: во-первых, шаг в нем измеряется в вещественных величинах, что уже дает некоторую погрешность (если шаг равен 1/3, то мы не можем получить точное значение (дробь несократима)), во-вторых, вычисление синуса и косинуса тоже не идеально (начиная с какого-то по счету символа не точно), что так же вносит вклад в погрешность.

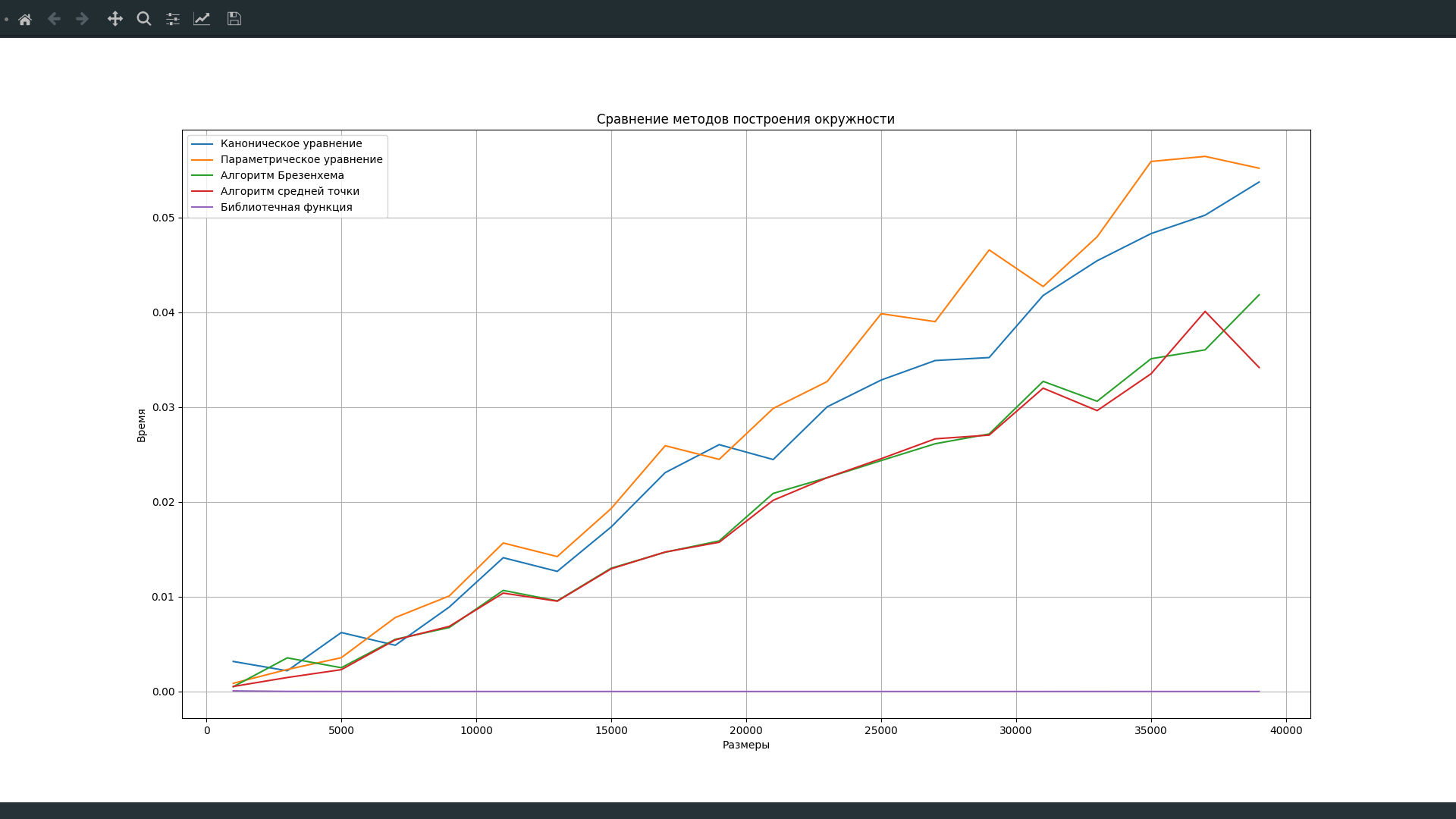
**2. Эллипс**

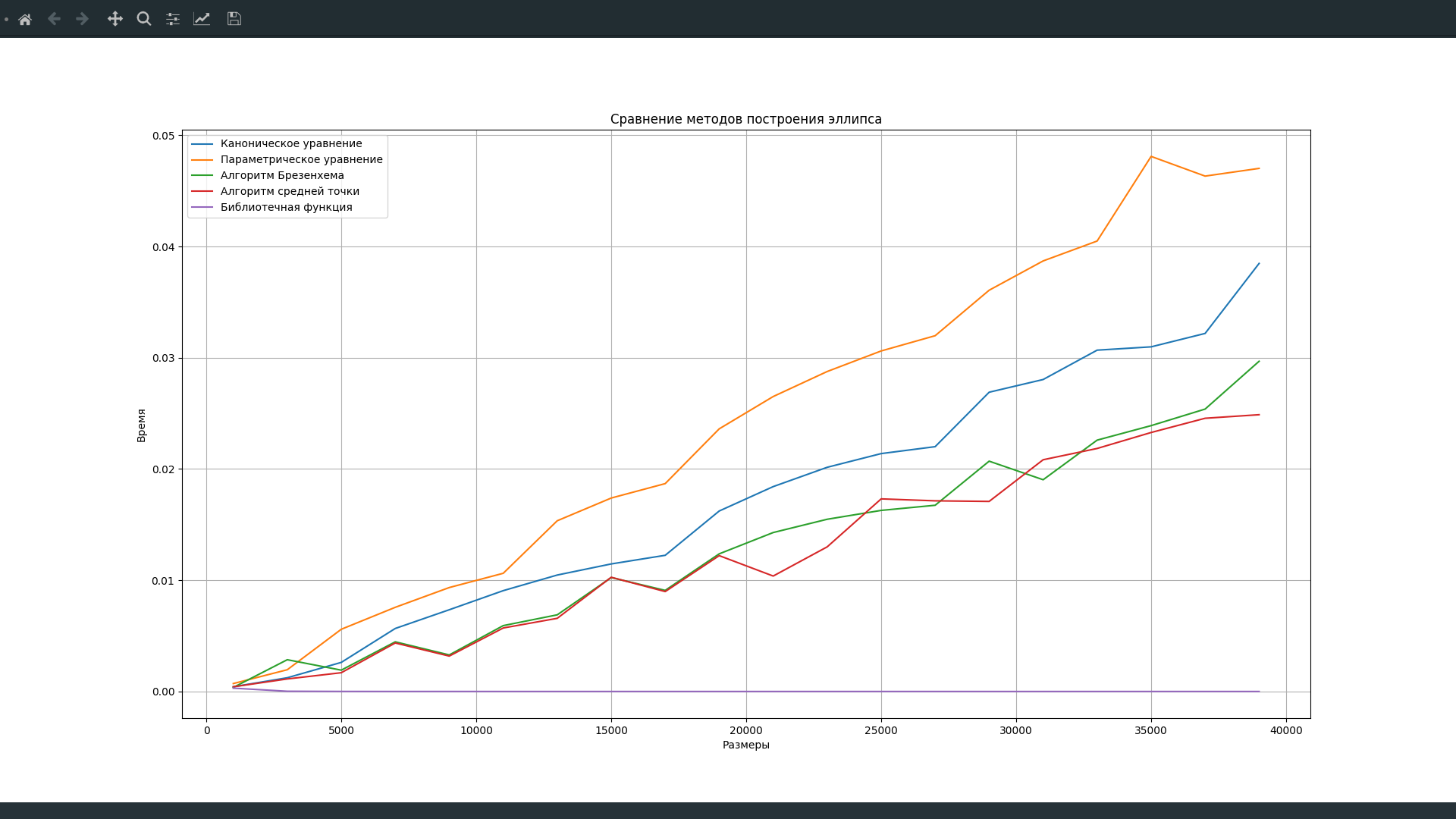
В эллипсах получена аналогичная картина: только параметрических метод отличается ото всех:



На рисунке 4 концентрических эллипса.

**Сравнение временных характеристик.**

Окружность:

Эллипс:

Таким образом мы видим, что временные характеристики алгоритмов при построении окружностей и эллипсов похожи и хорошо видно следующее:

1. Алгоритмы, работающие с целыми числами, без квадратов, корней и синусов работают быстрее всего (метод средней точки работает чуть быстрее Брезенхема из-за того, что анализирует 2 пиксела вместо 3 и вследствие этого имеет меньше вычислений).

2. Алгоритм канонического уравнения и параметрических уравнений работают медленнее вследствие наличия вычислительно сложных операций (причем вычисление синуса и косинуса сложнее чем просто вычисление корня, поэтому алгоритм на основе системы параметрических уравнений работает медленнее алгоритма на основе канонического уравнения)

3. Так как в параметрическом методе эллипса я проводил какое-то кол-во лишних вычислений (связаных с рассмотрением большего радиуса), то этот метод для эллипсов еще больше отстает от остальных по времени, чем его аналог для окружностей.

**Общий вывод:**

При выборе алгоритма построения окружностей или эллипсов следует брать:

1. Алгоритм средней точки (самый быстрый и при этом точный)

2. Алгоритм Брезенхема (быстрый и точный)

3. Алгоритм на основе канонического уравнения (имеет вычислительные сложности, средний по скорости, но точный)

4. Алгоритм на основе параметрического уравнения (самый долгий в связи с вычислительными сложностями, при этом еще и имеет большую погрешность)