

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Дисциплина « Компьютерная графика»**

**Лабораторная работа №4**

**по теме:**

**«РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ АЛГОРИТМОВ ГЕНЕРАЦИИ ОКРУЖНОСТИ И ЭЛЛИПСА.»**

**Работу выполнил:**

студент группы ИУ7-43Б

Сукочева А.

**Работу проверил:**

Куров А. В.

2020 г.

**Цель работы:**

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

**Задание:**

1.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

2. 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

- Алгоритма средней точки

- 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.

**Теоретический материал:**

Уравнения окружности:

(X-Xo)^2 + (Y-Yo)^2 = R^2 - Каноническое.

X=Xo+Rcost, Y=Yo+Rsint - Параметрическое.

Где Xo,Yo - координаты центра, R - радиус окружности, t - параметр (0<=t<=2)

Уравнения эллипса:

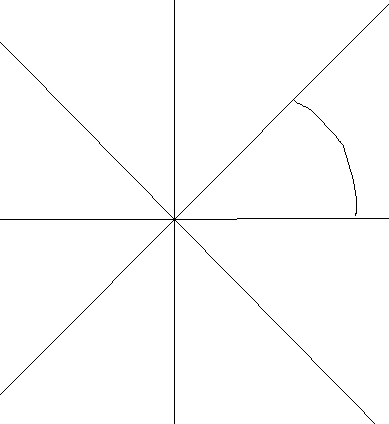
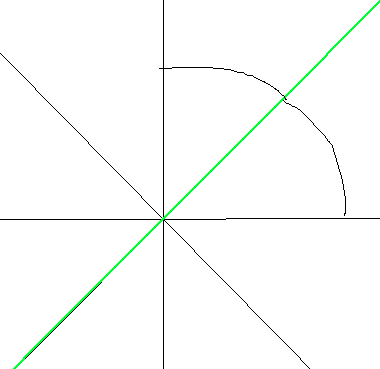
(X-Xo)^2/a^2 + (Y-Yo)^2/b^2 = 1 - Каноническое.

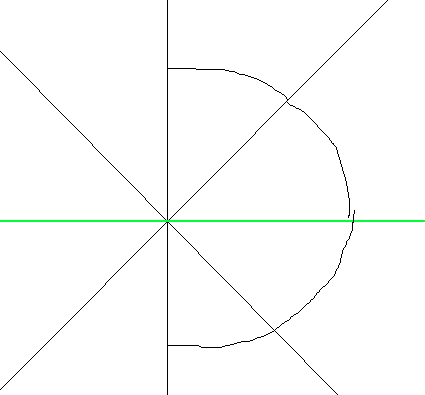
X=Xo+a cost, Y=Yo+b sint - Параметрическое.

a, b - полуоси.

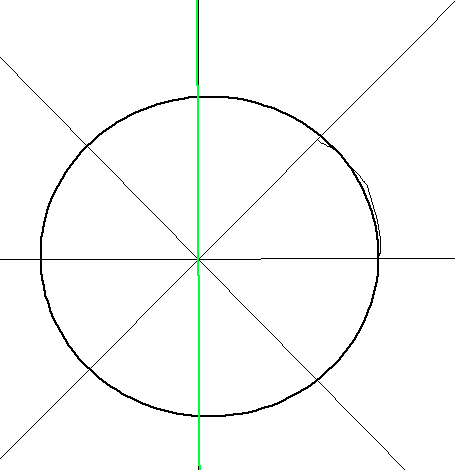
Мы можем построить ⅛ часть окружности а затем отразить ее:

Сначала отражаем относительно прямой x=y





Получили ¼ часть, затем отражаем относительно прямой y=0:

Получили ½ часть. Теперь отражаем относительно прямой x=0 получаем:  
Тем самым, для того, что построить полную окружность нам достаточно построить построить ⅛ часть, а остальные части получаются путем симметричного отражения относительно определенной прямой.

**Замечание:** Для всех методов рисования окружности я нахожутолько ⅛ часть, а непосредственно перед рисованием отражаю. А для всех методов построения эллипса я нахожу ¼ и перед рисованием отражаю.

Построение окружности (и эллипса) путем использования их уравнений очень затратно (Т.к. нам нужно будет каждый раз вычислять sin, cos или корень) Поэтому были предложены алгоритмы построения окружности.

**Алгоритм построения окружности на основе канонического уравнения:**

Из канонического уравнения мы выражаем y. Приращение аргумента устанавливаем = 1 и находим в каждой точке значение функции. Чтобы шаг по x был равен 1 и это не приводило к прерываниям функции нужно для y иметь меньшее приращение (т.е. нужно чертить только ту часть окружности, которая имеет наклон <= 45 градусов)

**def canonical\_circle(center, radius):**

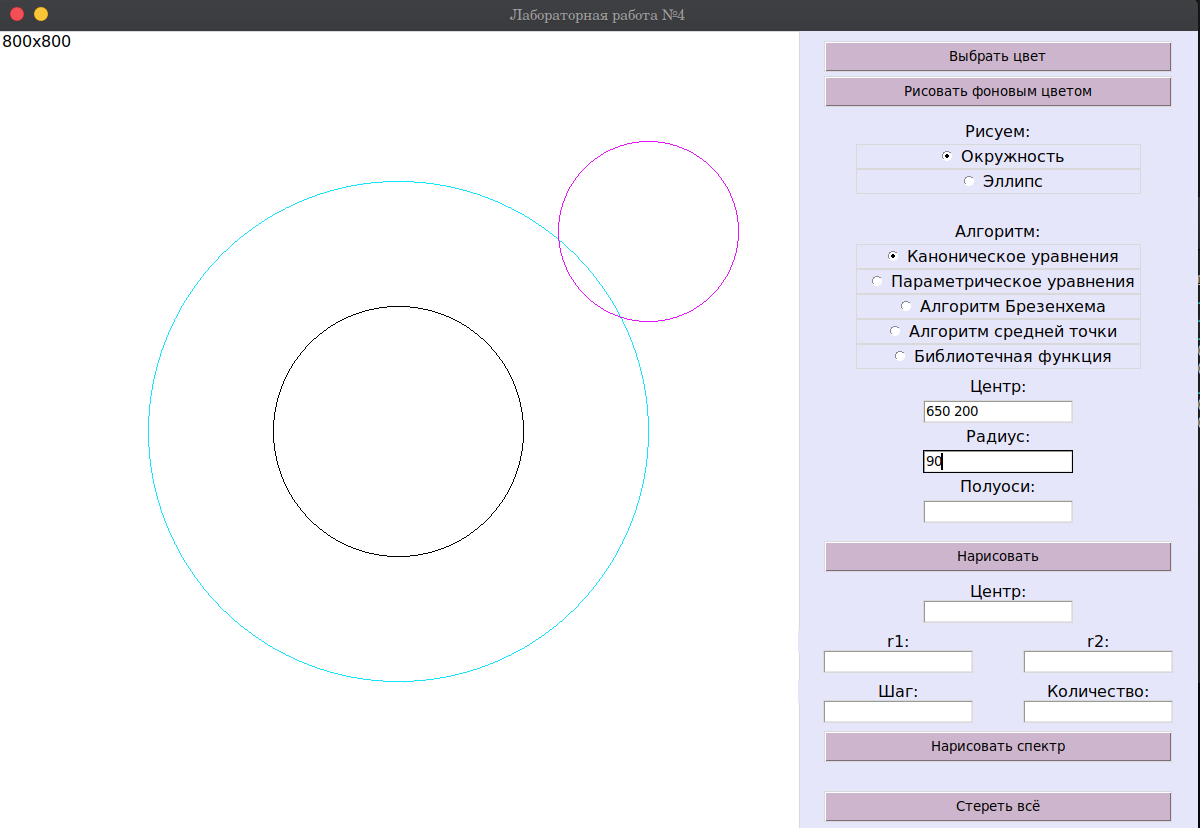
**list\_points = list()**

**R = radius \* radius**

**for x in range(center[0], round(center[0]+radius/sqrt(2)+1)):**

**y = center[1] + sqrt(R - (x - center[0]) \*\* 2)**

**list\_points.append([x, y])**

**return list\_points**

**Алгоритм построения окружности на основе системы параметрических уравнений:**

Используя параметрическое уравнение мы снова рисуем ⅛ часть окружности (от 0 до pi/4)

**def parametric\_circle(center, radius):**

**list\_points = list()**

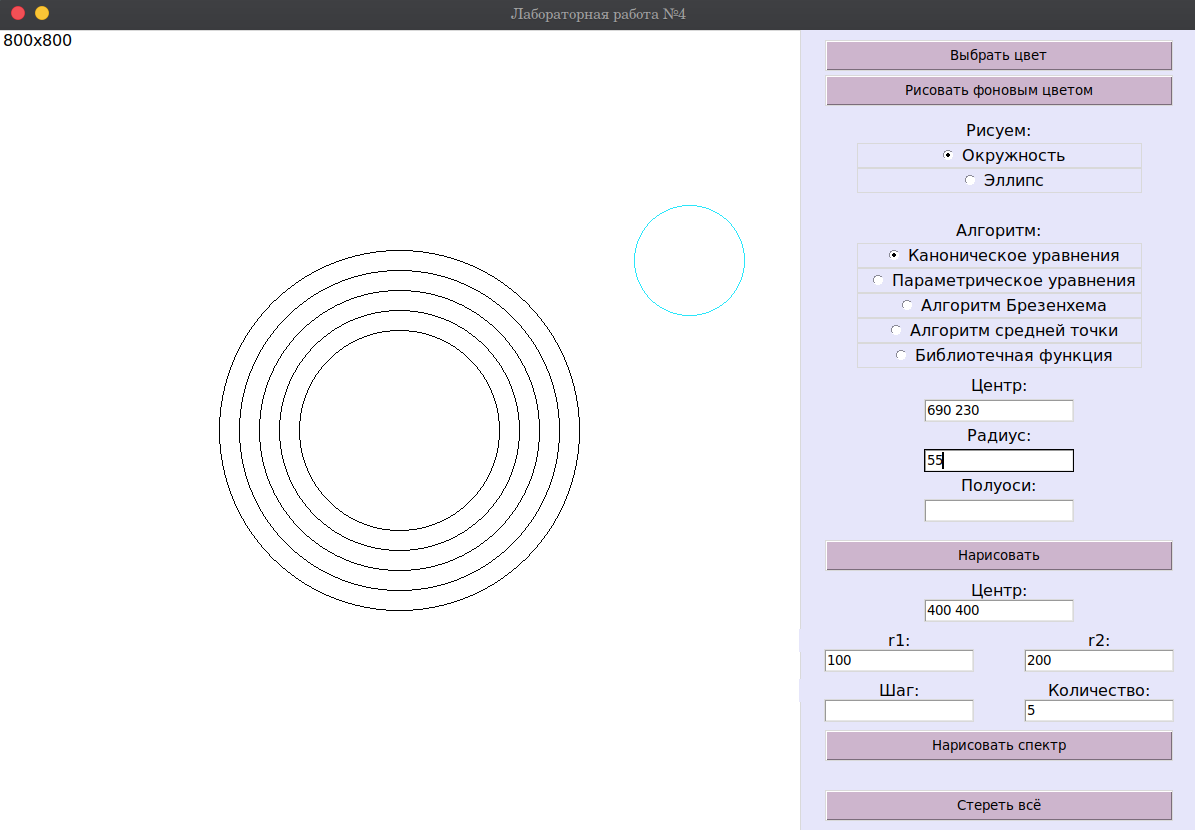
**step = 1 / radius**

**for t in arange(0, pi/4 + step, step):**

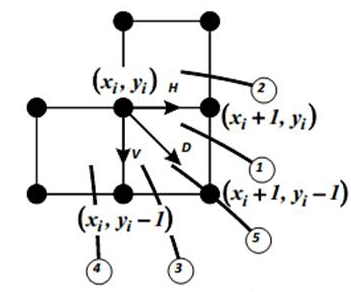
**x = radius \* cos(t) + center[0]**

**y = radius \* sin(t) + center[1]**

**list\_points.append([x, y])**

**return list\_points**

**Алгоритм Брезенхема построения окружности:**

В данном алгоритме мы рассмотрим возможные ситуации прохода дуги и рассмотрим расстояние до ближайших пикселей.

Имея пиксель (xi, yi) перед нами встает задача выбора следующего пикселя ((xi+1, yi) или (xi+1, yi-1) или (xi, yi-1)). Будем называть (xi+1, yi) - горизонтальным пикселем, (xi, yi-1) - вертикальным пикселем, а (xi+1, yi-1) - диагональным пикселем.

Введем величину: i = (xi + 1)^2 + (yi - 1)^2 - R^2.

Рассмотрим данную величину:

i < 0, то расстояние от центра до окружности больше, чем до диагонального пикселя. (Случаи 1 и 2)

i = 0 окружность проходит через диагональный пиксель. (Случай 5)

i > 0, то расстояние от центра до диагонального пикселя больше, чем до окружности. (Случаи 3 и 4)

i < 0:

d1 = |(xi + 1)^2 + yi^2 - R^2| - |(xi + 1)^2 + (yi - 1)^2 - R^2|

Первый модуль - расстояние между горизонтальным пикселем и нашей окружность, а второй модуль расстояние между диагональным пикселем и окружностью. Таким образом если величина отрицательная, то расстояние до горизонтального пиксела меньше, иначе — до диагонального меньше.

Раскроем модули в зависимости от положения дуги:

Случай 1: (xi + 1)^2 + yi^2 - R^2 > 0 и (xi + 1)^2 + (yi - 1)^2 - R^2 < 0 =>

d1 = (xi + 1)^2 + yi^2 - R^2 + (xi + 1)^2 + (yi - 1)^2 - R^2 =

2(xi + 1)^2 + yi^2 - 2yi + 1 + 2yi - 1 - 2R^2 + (yi - 1)^2 =

2(xi + 1)^2 + 2(yi - 1)^2 -2R + 2yi - 1 = 2i + 2yi - 1 = 2(i + yi) - 1

Случай 2: (xi + 1)^2 + yi^2 - R^2 < 0 и (xi + 1)^2 + (yi - 1)^2 - R^2 < 0 =>

d1 = -(xi + 1)^2 - yi^2 + R^2 + (xi + 1)^2 + (yi - 1)^2 - R^2=

- yi^2 + (yi - 1)^2 = - yi^2 + yi^2 -2yi + 1 = 1-2yi

i > 0:

d2 = |(xi + 1)^2 + (yi - 1)^2 - R^2| - |xi^2 + (yi - 1)^2 - R^2|

Первый модуль — расстояние до диагонального пиксела.

Первый модуль - расстояние между диагональным пикселем и нашей окружность, а второй модуль расстояние между вертикальным пикселем и окружностью. Таким образом если величина отрицательная, то расстояние до диагонального пиксела меньше, иначе — до вертикального меньше.

Случай 3: d2 = 2 (Δi + xi) - 1

Случай 4: d1 = 2 xi + 1 > 0

Случай 5. Выбор очевиден - диагональный пиксел лежит на окружности.

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

Горизонтальный шаг:

Δ(i+1) = (x(i+1) + 1)^2 + (y(i+1) - 1)^2 - R^2.

x(i+1) = xi + 1

y(i+1) = yi =>

Δ(i+1) = (xi + 2)^2 + (y - 1)^2 - R^2 = (xi + 1)^2 + (yi - 1)^2 - R^2 + 2xi + 3 = Δi + 2xi + 3 = Δi + 2x(i+1) + 1

Вертикальный шаг:

x(i+1) = xi

y(i+1) = yi - 1

Δ(i+1) = Δi - 2y(i+1) + 1

Диагональный шаг:

x(i+1) = xi + 1

y(i+1) = yi - 1;

Δ(i+1) = Δi + 2 (x(i+1) - y(i+1) + 1)

Начальное значение Δ = 2(1-R)

**def brezenham\_circle(center, r):**

**list\_points = list()**

**x = 0**

**y = r**

**list\_points.append([x + center[0], y + center[1]])**

**delta = 2 - r - r**

**while x < y:**

**if delta <= 0:**

**d1 = delta + delta + y + y - 1**

**x += 1**

**if d1 >= 0:**

**y -= 1**

**delta += 2 \* (x - y + 1)**

**else:**

**delta += x + x + 1**

**else:**

**d2 = 2 \* (delta - x) - 1**

**y -= 1**

**if d2 < 0:**

**x += 1**

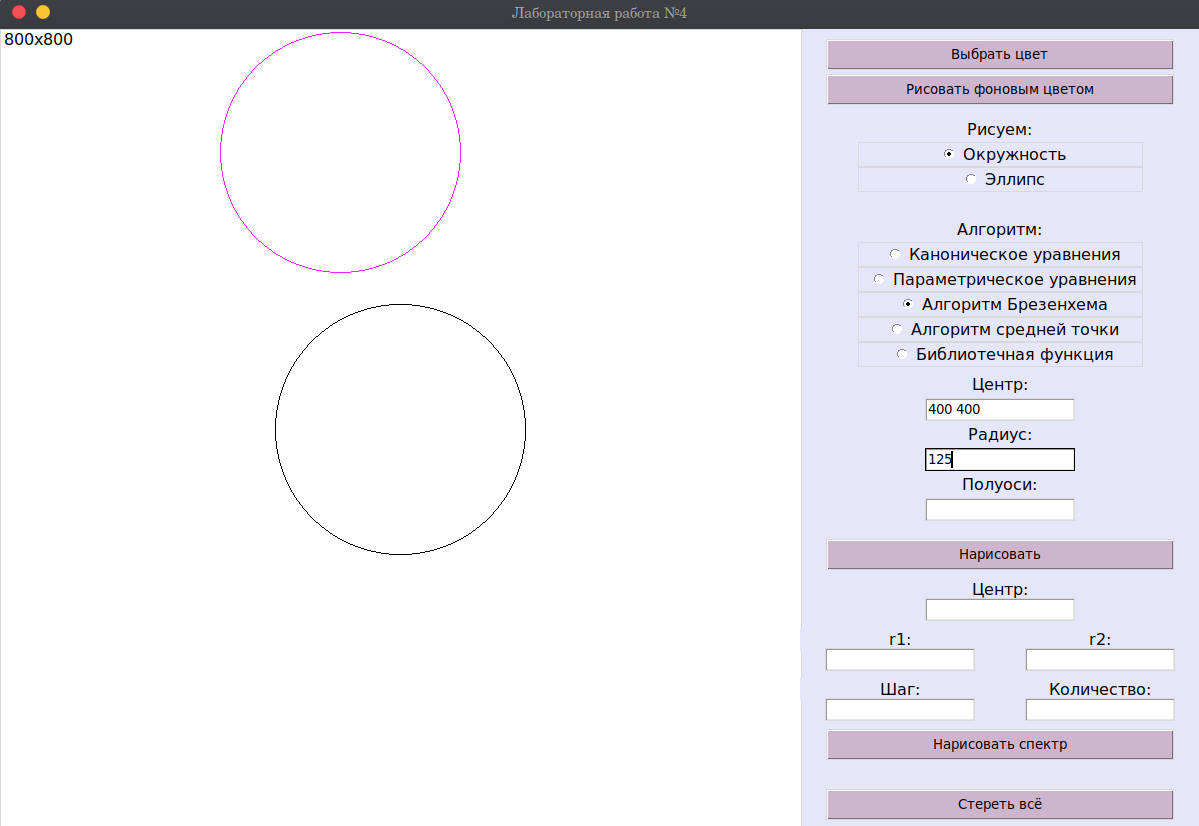
**delta += 2 \* (x - y + 1)**

**else:**

**delta -= y + y - 1**

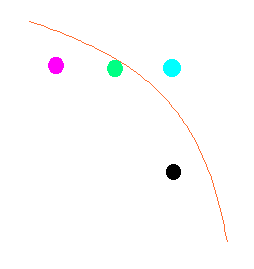
**list\_points.append([x + center[0], y + center[1]])**

**return list\_points**



**Алгоритм средней точки построения окружности:**

В данном алгоритме шаг по y = 1, будет ли сделан шаг по x зависит от положения средней точки. Если средняя точка лежит внутри окружности, то истинной окружности окружности ближе вертикальный пиксель (И шага по х нет) иначе ближе диагональный (Шаг по х = -1)

Черная точка - (xi, yi)

Голубая точка - (xi, yi+1)

Фиолетовая точка - (xi-1, yi+1)

Зеленая точка (Средняя точка) - (xi-½, yi+1)

В данном случае средняя точка лежит внутри окружности, по-этому шага по x нет.

Имея уравнение окружности: X^2 + Y^2 = R^2

Введем функцию (И подставим туда координаты средней точки):

F = (x-0.5)^2 + (y+1)^2 - R^2

х, y - координаты пиксела, выбранного на последнем шаге.

Если F<0, то средняя точка расположена внутри окружности. (Нужно выбирать голубой пиксель)

Если F=0, то средняя точка расположена на точно на дуге.(Выбираем любой пиксель)

Если F>0, то средняя точка расположена вне окружности (Выбираем фиолетовый пиксель)

Вычислим то, как меняется данная функция в зависимости от шага:

F(i+1) = Fi + Δ

Горизонтальный шаг:

x(i+1) = xi

y(i+1) = yi + 1

Δ = F(i+1) - Fi = (x(i+1)-0.5)^2 + (y(i+1)+1)^2-R^2 - (xi-0.5)^2-(yi+1)^2+R^2 = (xi-0.5)^2 + (yi+2)^2-R^2 - (xi-0.5)^2 - (yi+1)^2+R^2 = (yi+2)^2 - (yi+1)^2 = 2yi + 3 = 2(yi+1) + 1

Диагональный шаг:

Δ = 2(yi + 1) + 1 - 2(xi-1)

**def middle\_point\_circle(center, r):**

**list\_points = list()**

**x = r**

**y = 0**

**list\_points.append([center[0] + x, center[1] + y])**

**f = 1 - r**

**while x > y:**

**y += 1**

**if f >= 0:**

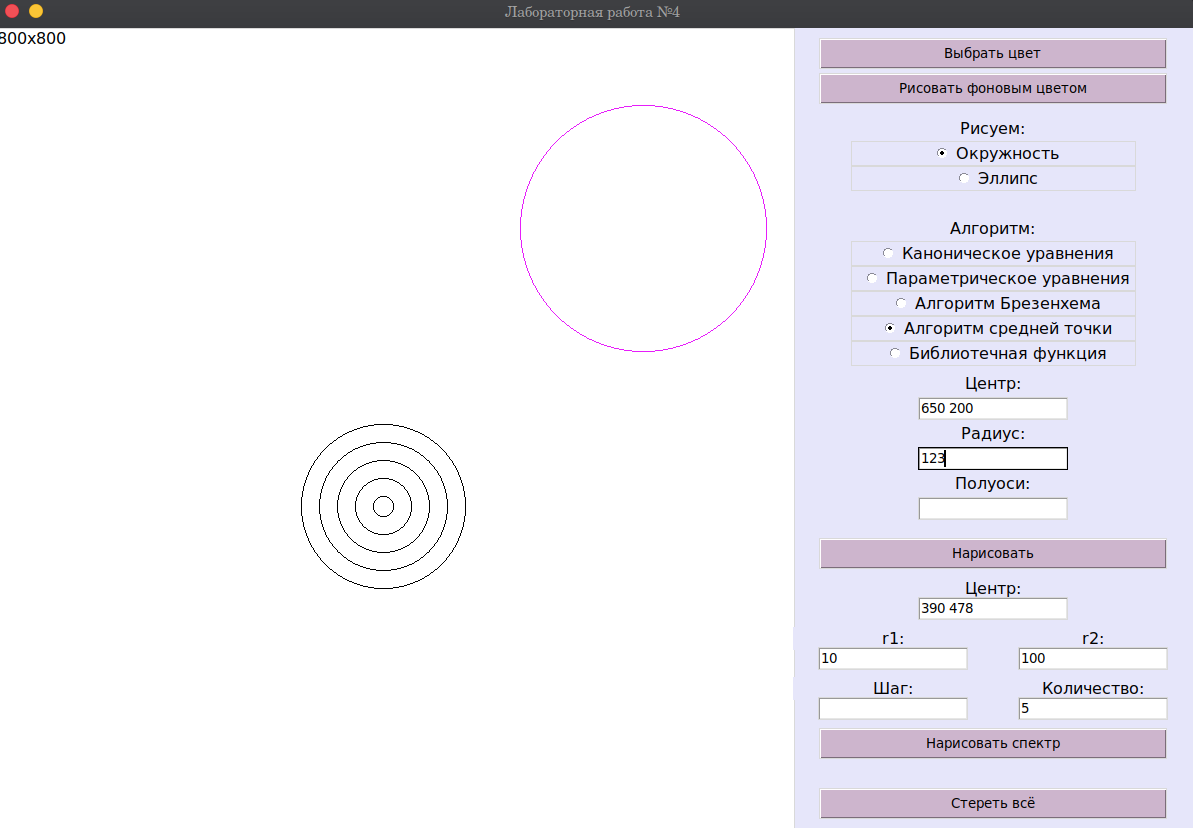
**x -= 1**

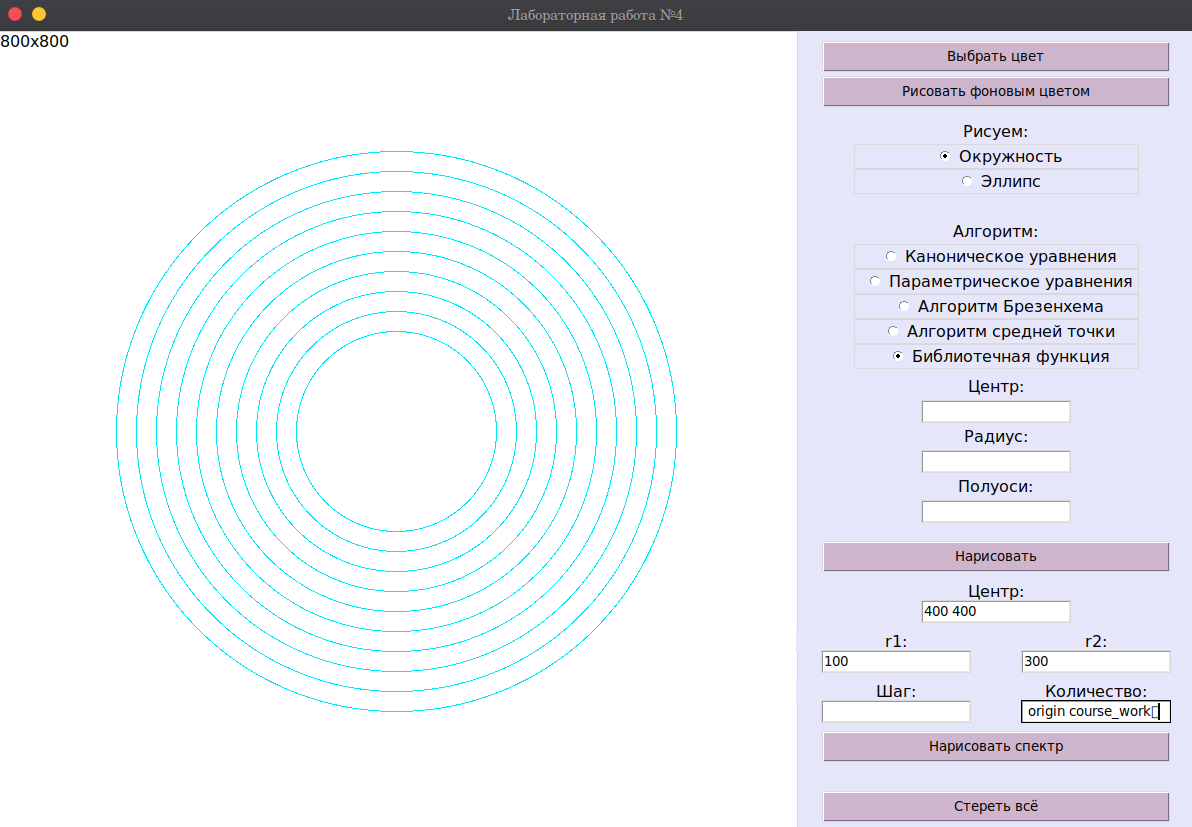
**f -= x + x**

**f += y + y + 1**

**list\_points.append([center[0] + x, center[1] + y])**

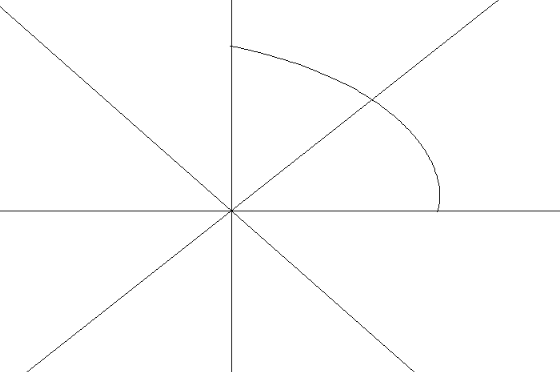
**return list\_points**

****



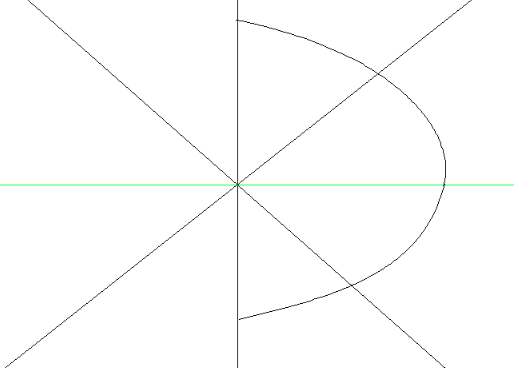
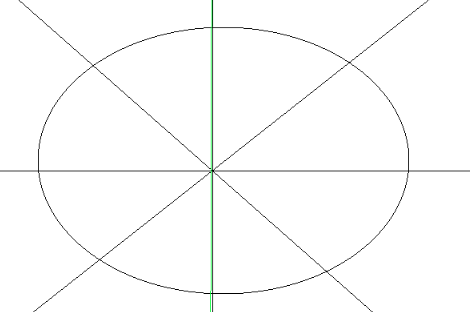
**Библиотечный метод:**

**Алгоритм построения эллипса на основе канонического уравнения:**

Так как эллипс не имеет симметрии относительно прямой y=x требуется строить ¼, а не ⅛. 

Сначала отразим относительно прямой y=0

Затем относительно прямой x=0



(X-Xo)^2/a^2 + (Y-Yo)^2/b^2 = 1 - Каноническое уравнение.

Выразим y и x:

(Y-Yo)^2/b^2 = [1 - (X-Xo)^2/a^2 ]

(Y-Yo)^2 = b^2[1 - (X-Xo)^2/a^2 ]

Y-Yo = b\*sqrt(1 - (X-Xo)^2/a^2)

Y = Yo + b\*sqrt(1 - (X-Xo)^2/a^2)

X = Xo + a\*sqrt(1 - (Y-Yo)^2/b^2)

**def ellipse\_canonical(center, a, b):**

**list\_points = list()**

**for x in range(0, a + 1, 1):**

**y = round(sqrt(1 - (x/a)\*\*2) \* b)**

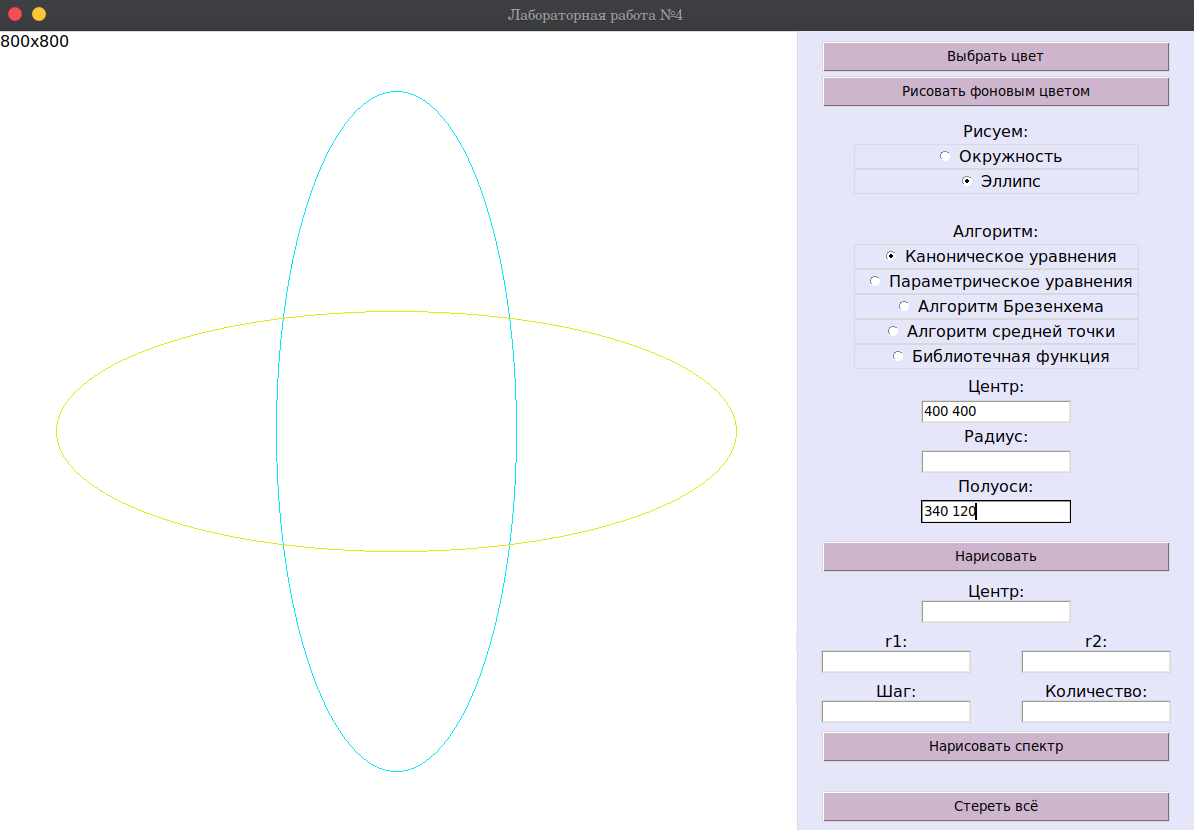
**list\_points.append([center[0] + x, center[1] + y])**

**for y in range(0, b + 1, 1):**

**x = round(sqrt(1 - (y / b) \*\* 2) \* a)**

**list\_points.append([center[0] + x, center[1] + y])**

**return list\_points**

****

**Алгоритм построения эллипса на основе параметрического уравнения:**

Данный алгоритм является очень медленным, потому что нам нужно каждый раз вычислять sin и cos (разложение в ряд занимает много времени). Также нужно рисовать ¼ часть ,а потом отражать относительно двух прямых (x=0 и y=0).

**def parametric\_ellipse(center, axis):**

**list\_points = list()**

**step = 1 / max(axis[0], axis[1])**

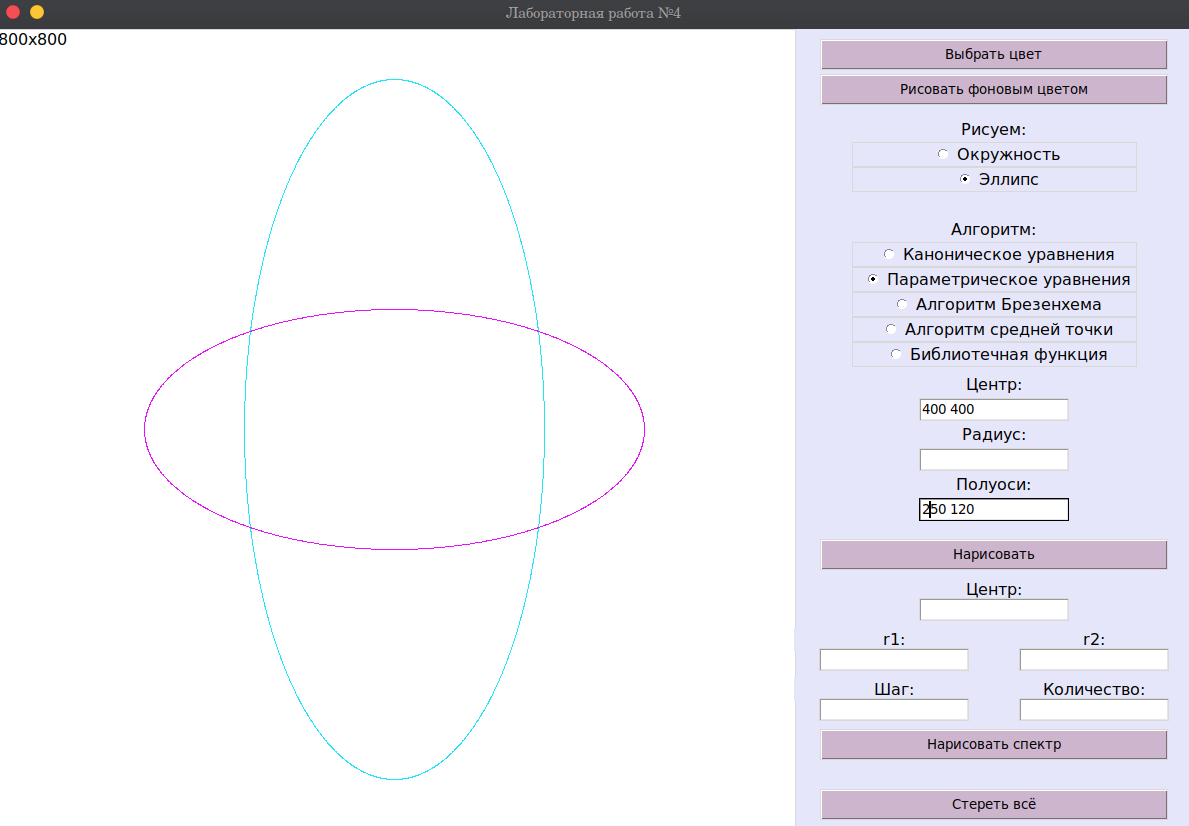
**for t in arange(0, pi/2 + step, step):**

**x = axis[0] \* cos(t) + center[0]**

**y = axis[1] \* sin(t) + center[1]**

**list\_points.append([x, y])**

**return list\_points**

****

**Алгоритм Брезенхема построения эллипса:**

Суть данного алгоритма такая же, как и в алгоритме Брезенхема построения окружности, отличается он тем, что при вычислениях всех расстояний использовалось каноническое уравнение эллипса.

**def brezenham\_ellipse(center, a, b):**

**list\_points = list()**

**x = 0**

**y = b**

**sqr\_b = b \* b**

**sqr\_a = a \* a**

**list\_points.append([x + center[0], y + center[1]])**

**delta = sqr\_b - sqr\_a \* (2 \* b + 1)**

**while y > 0:**

**if delta <= 0:**

**d1 = 2 \* delta + sqr\_a \* (2 \* y - 1)**

**x += 1**

**delta += sqr\_b \* (2 \* x + 1)**

**if d1 >= 0:**

**y -= 1**

**delta += sqr\_a \* (-2 \* y + 1)**

**else:**

**d2 = 2 \* delta + sqr\_b \* (-2 \* x - 1)**

**y -= 1**

**delta += sqr\_a \* (-2 \* y + 1)**

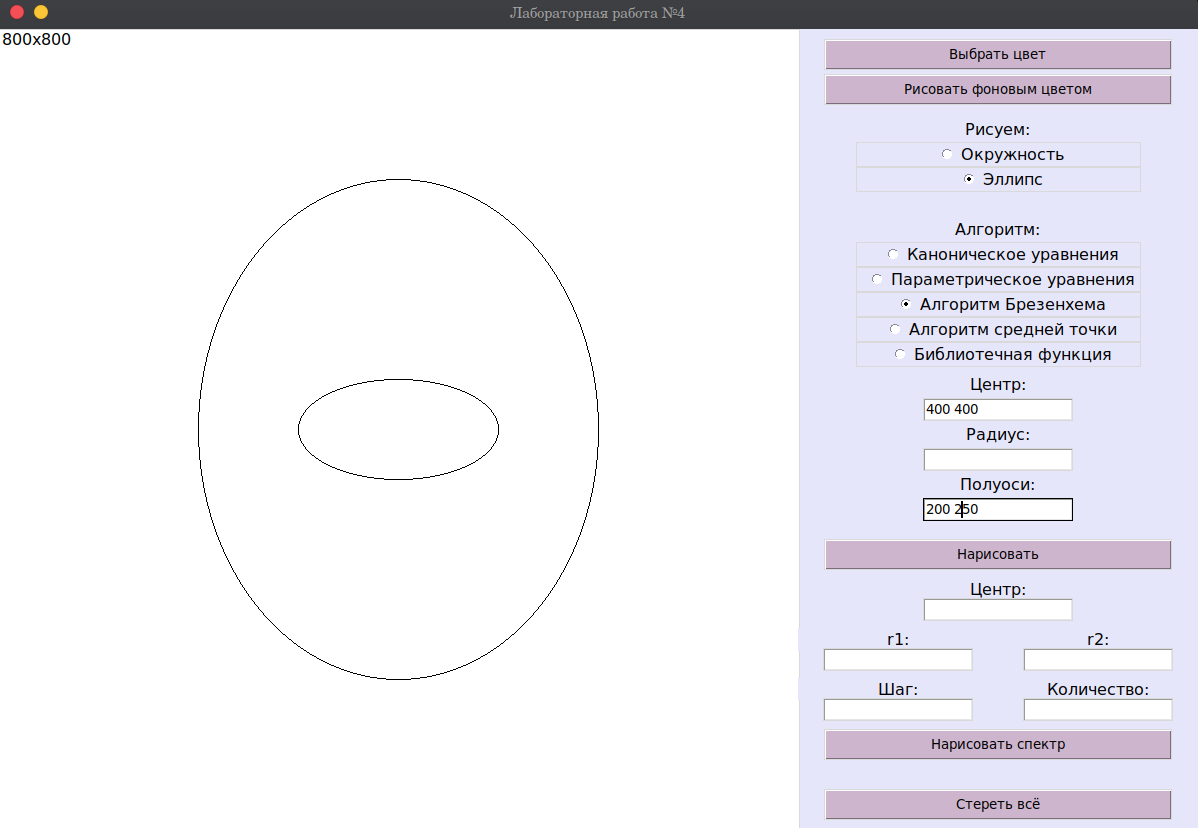
**if d2 < 0:**

**x += 1**

**delta += sqr\_b \* (2 \* x + 1)**

**list\_points.append([x + center[0], y + center[1]])**

**return list\_points**



**Алгоритм средней точки построения эллипса:**

Т.к. нам приходится строить ¼ часть эллипса нам нужно найти точку, в которой наклон касательной к эллипсу становится меньше 45. Т.к. в этой точке та переменная, которая имела большее приращение, начинает расти меньше, а другая получает большее приращение. Это необходимо из-за того, что в алгоритме средней точки одно приращение всегда больше другого. Расчеты такие же, как и в алгоритме средней точки построения окружности, только используется другое уравнение.

**def middle\_point\_ellipse(center, a, b):**

**list\_points = list()**

**sqr\_a, sqr\_b = a \* a, b \* b**

**limit = round(a / sqrt(1 + sqr\_b / sqr\_a))**

**x, y = 0, b**

**list\_points.append([x + center[0], y + center[1]])**

**f = sqr\_b - round(sqr\_a \* (b - 1/4))**

**while x < limit:**

**if f > 0:**

**y -= 1**

**f -= 2 \* sqr\_a \* y**

**x += 1**

**f += sqr\_b \* (2 \* x + 1)**

**list\_points.append([x + center[0], y + center[1]])**

**limit = round(b / sqrt(1 + sqr\_a / sqr\_b))**

**y, x = 0, a**

**list\_points.append([x + center[0], y + center[1]])**

**f = sqr\_a - round(sqr\_b \* (a - 1/4))**

**while y < limit:**

**if f > 0:**

**x -= 1**

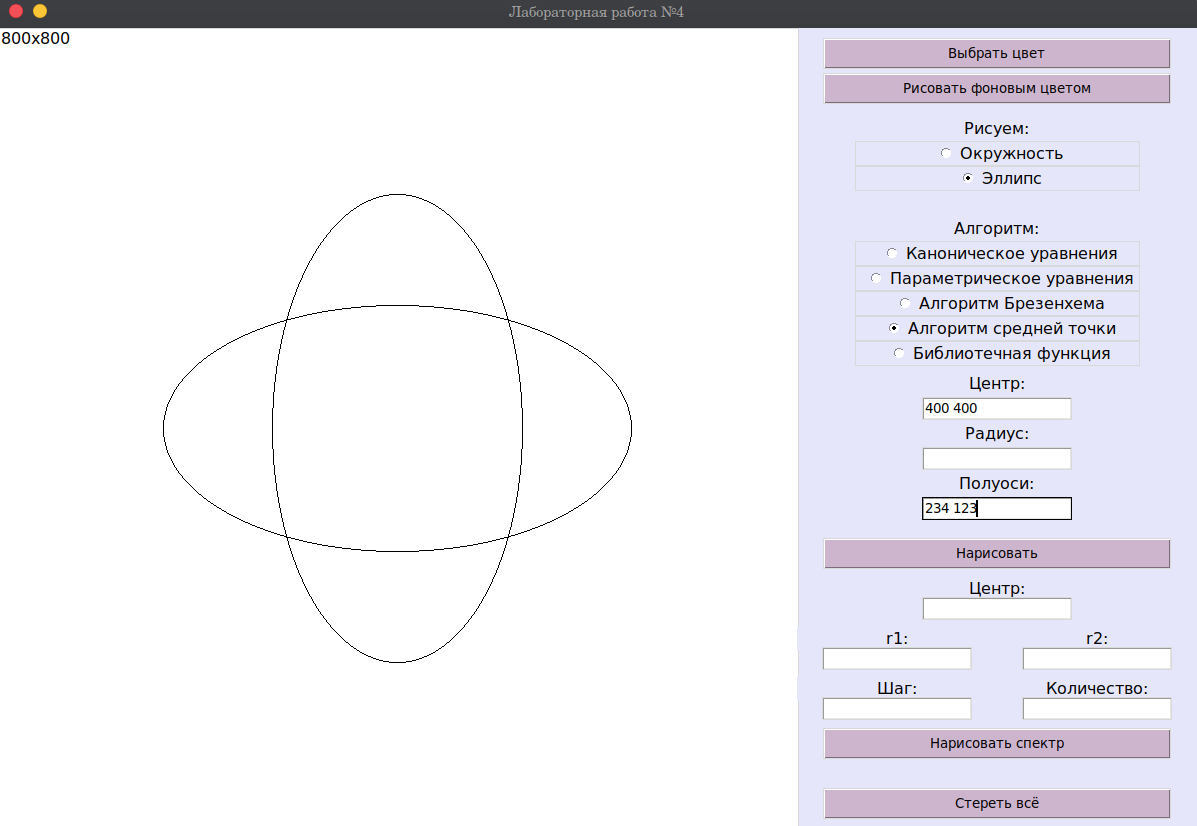
**f -= 2 \* sqr\_b \* x**

**y += 1**

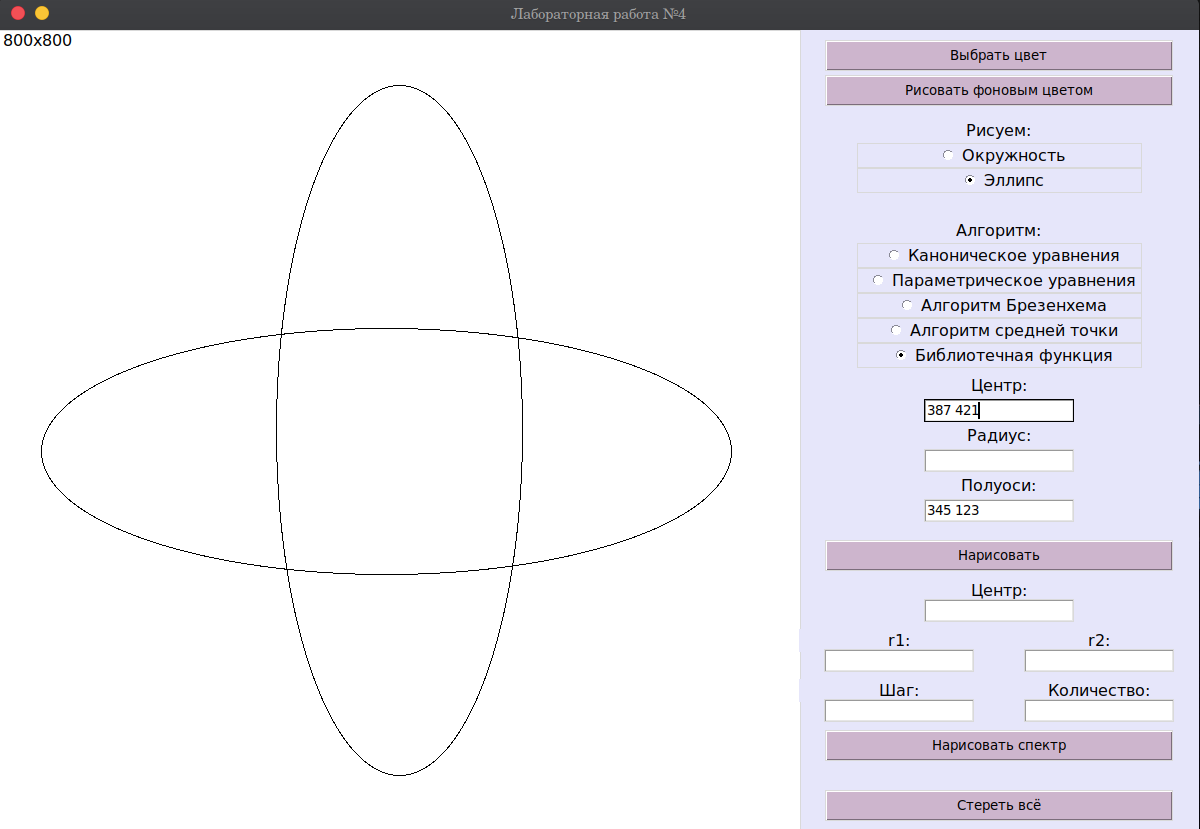
**f += sqr\_a \* (2 \* y + 1)**

**list\_points.append([x + center[0], y + center[1]])**

**return list\_points**

****

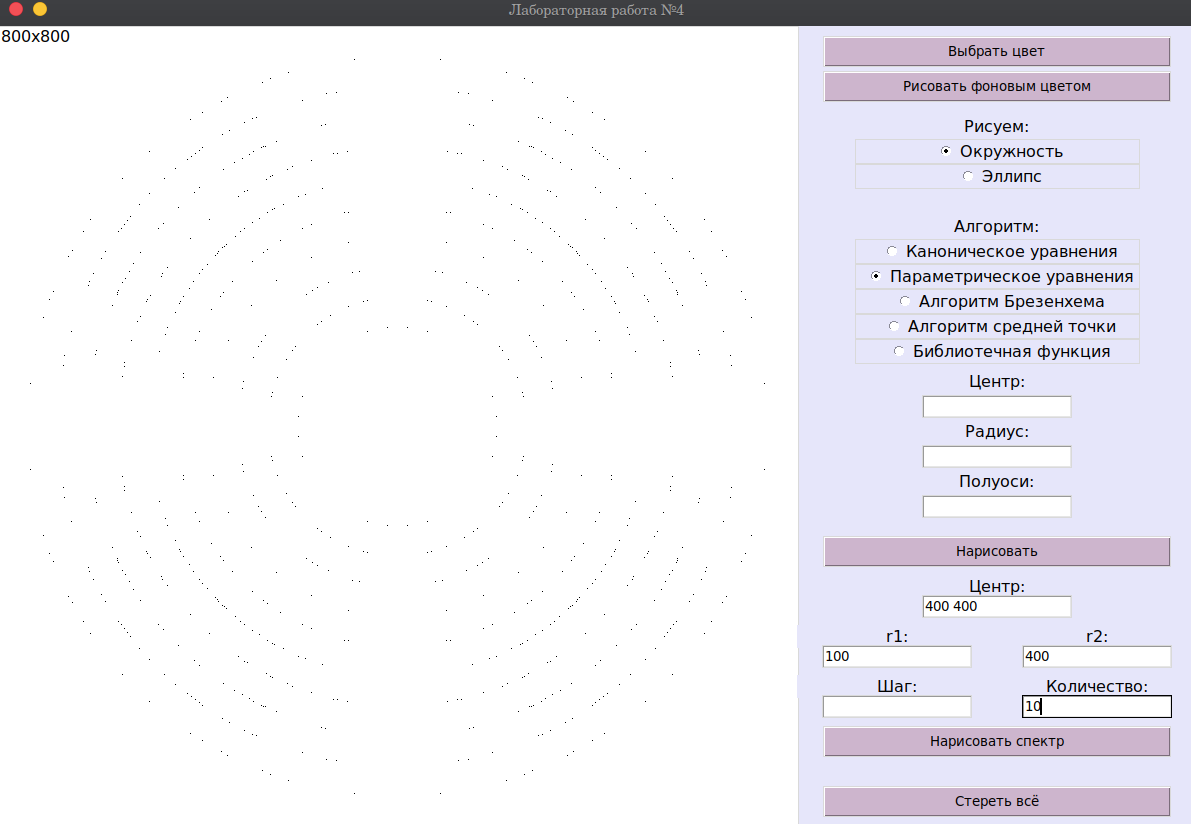
**Библиотечный метод:**



**Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических окружностей:**

Я рисую одним алгоритмом, а затем рисую другим алгоритмом цветом фона. После чего 3 алгоритма выдают одинаковые визуальные характеристики (Каноническое уравнение, Брезенхем, Средняя точка). А именно - они оставляют полотно белым. Библиотечный метод тоже стирает другие методы (Но если сначала нарисовать библиотечным, а поверх другим методом, то останутся точки, т.е. он закрашивает другие алгоритмы, но его другие алгоритмы не закрашивают) .

Параметрическое уравнение выдает данный результат: (Поверх канонических 10 окружностей я нарисовала, цветом фона, 10 окружностей с использованием параметрического уравнения) :



Т.к. шаг в параметрическом уравнение является вещественным числом - это приводит к погрешности.

**Сравнение визуальных характеристик разных алгоритмов при рисовании спектра концентрических эллипсов:**

Такая же ситуация, как и с окружностями.Отличие параметрического метода (Сначала нарисовала алгоритмом средней точки, а потом параметрическим):

