|  |  |
| --- | --- |
| 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**

|  |  |
| --- | --- |
| **Дисциплина Компьютерная графика**  **Тема Программная реализация основных алгоритмов построения окружностей и эллипсов и исследование их временных и визуальных характеристик**  **Студент Иванов В.А.**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.

**Цель работы**

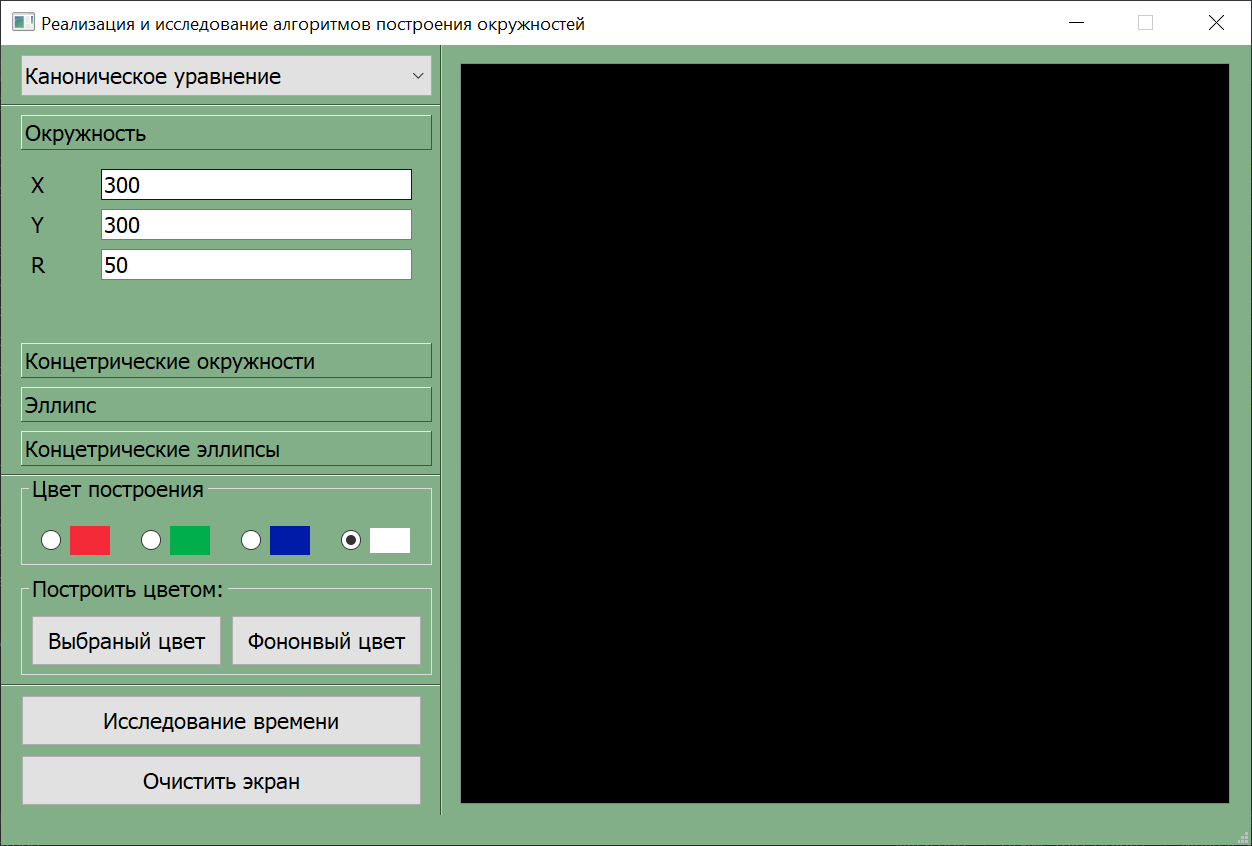
Реализация алгоритмов построения окружности, исследование и сравнение визуальных и временных характеристик алгоритмов.

**Описание задания**

1. Реализовать алгоритмы
2. Сравнить визуально окружности и эллипсы, построенные в соответствии с каждым алгоритмом, а также с отрезком, построенным процедурой языка высокого уровня. Проверить попадание отрезка в заданную конечную точку.
3. Определить время, затрачиваемое на построение окружности и эллипса по каждому из алгоритмов.

**Описание графического интерфейса**

Для использования функционала поставленных задач, был создан графический интерфейс

****

Интерфейс предоставляет возможность:

* Построения окружности, эллипса одиночно или концентрически, в соответствии с выбранным алгоритмом, по заданным параметрам. Отрезок может быть нарисован одним из 4 цветов или цветом фона. Таким образом можно сравнить визуальные характеристики разных алгоритмов, нарисовав их по одинаковым параметрам, но разными цветами
* Исследование временных характеристик, в зависимости от размера фигуры

**Описание алгоритмов**

Задача рассматриваемых окружностей и эллипсов обусловлена тем, что требуется изобразить непрерывные кривые в дискретном (растровом) пространстве. Алгоритмы, как и в случае с отрезками должны создавать изображение, которое выглядят как кривая, и также работать достаточно быстро.

В данной лабораторной рассматриваются различные алгоритмы, основанные на пошаговом принципе. Изучается эффективность этих алгоритмов, а также визуальные различия.

Также схожей чертой в отрисовке данных фигур является расчёт положения пикселей лишь для части изображения. Остальное изображение рисуется путём зеркального отображения отрисованной части. Ввиду симметрии, для окружности это возможно уже после отрисовки 1/8 всей окружности, а для эллипса после 1/4.

Важно отметить, что использованные в данном отчёте формулы для окружности эллипса приведены для случая, когда центр фигур совпадает с началом координатных осей. Сам центр фигур учитывается при отрисовке пикселей, при помощи сдвига на (x0, y0).

**Метод канонического уравнения**

Из названия метода видно, что он основан на использовании канонических уравнений для окружности и эллипса:

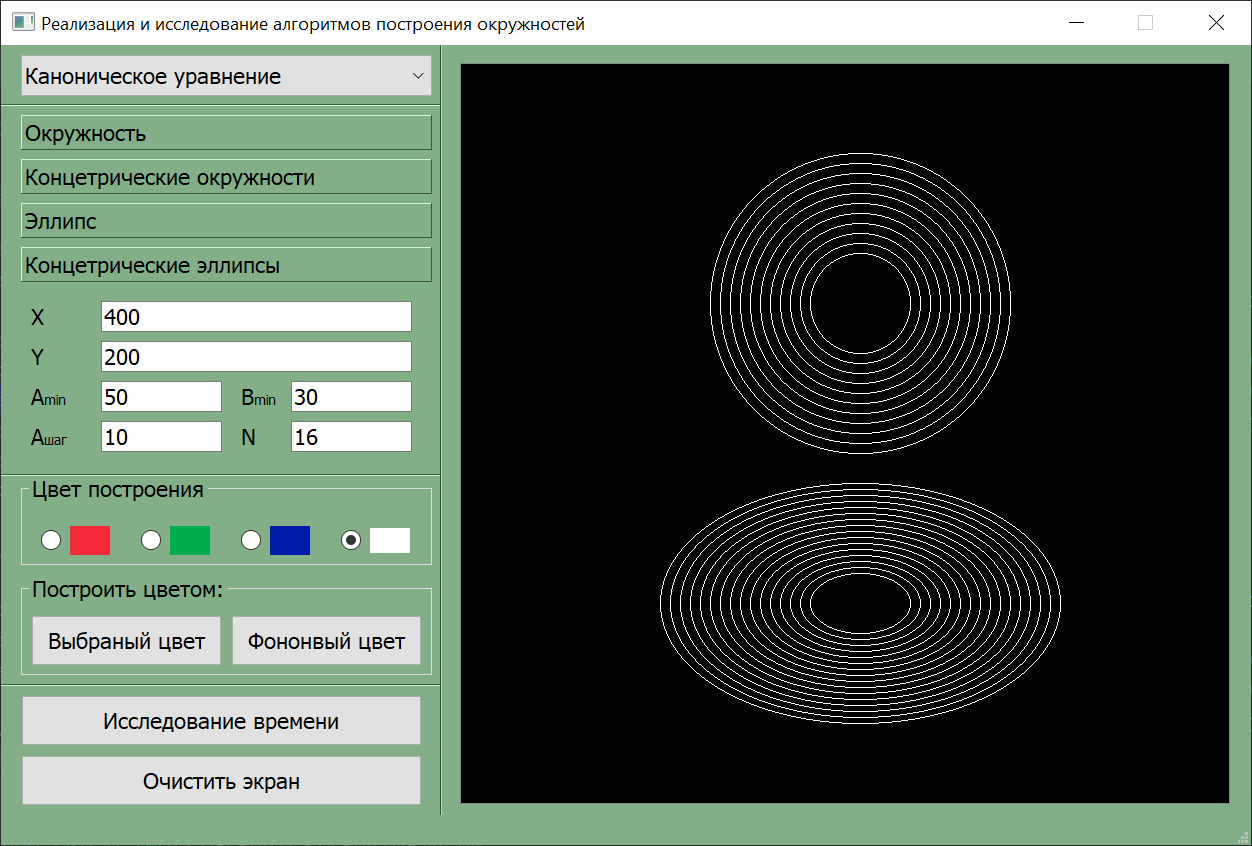
и

Из них можно получить зависимости x(y) или y(x) для определённых четвертей координатной плоскости. Часть фигур, в которой изменений x больше, чем изменения y (в окружности, например, это участок от pi/4 до pi/2), итерация выполняется по x: с каждым шагом он изменяется на 1, а значение y высчитывается из зависимости y(x). Аналогичные действия проводятся и для участка, где изменение y больше, чем x.

Для окружности используется один цикл, итерирующий х от 0 до значения, при котором он станет больше y (т.к. в этот момент следует начинать итерацию по y, как было описано выше). Остальные части окружности прорисовываются симметричным отображением полученной части.

Для эллипса используется два цикла. Первый итерирует x от 0 до max\_x, и использует зависимость y(x). Второй итерирует y от 0 до max\_y, используя x(y). Таким образом происходит отрисовка 1/4 эллипса, которая после отображается в остальные части фигуры. max\_x и max\_y – значения x и y соотв., при которых |dy/dx|=1.

Результат работы алгоритмов:



Код функций:

|  |  |
| --- | --- |
| Окружность | Эллипс |
| def drawc\_canonical(self, x0, y0, r):  r2 = r\*r  x = 0  y = r  while x <= y:  y = round(sqrt(r2 - x\*x))  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   self.draw\_pixel(y + x0, x + y0)  self.draw\_pixel(y + x0, -x + y0)  self.draw\_pixel(-y + x0, x + y0)  self.draw\_pixel(-y + x0, -x + y0)  x += 1 | def drawe\_canonical(self, x0, y0, a, b):  a2 = a\*a  b2 = b\*b  x = 0  max\_x = round(sqrt(a2 / (b2 / a2 + 1)))  while x <= max\_x:  y = round(b \* sqrt(1 - x\*x / a2))  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)  x += 1   y = 0  max\_y = round(sqrt(b2 / (a2 / b2 + 1)))  while y <= max\_y:  x = round(a \* sqrt(1 - y\*y / b2))  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)  y += 1 |

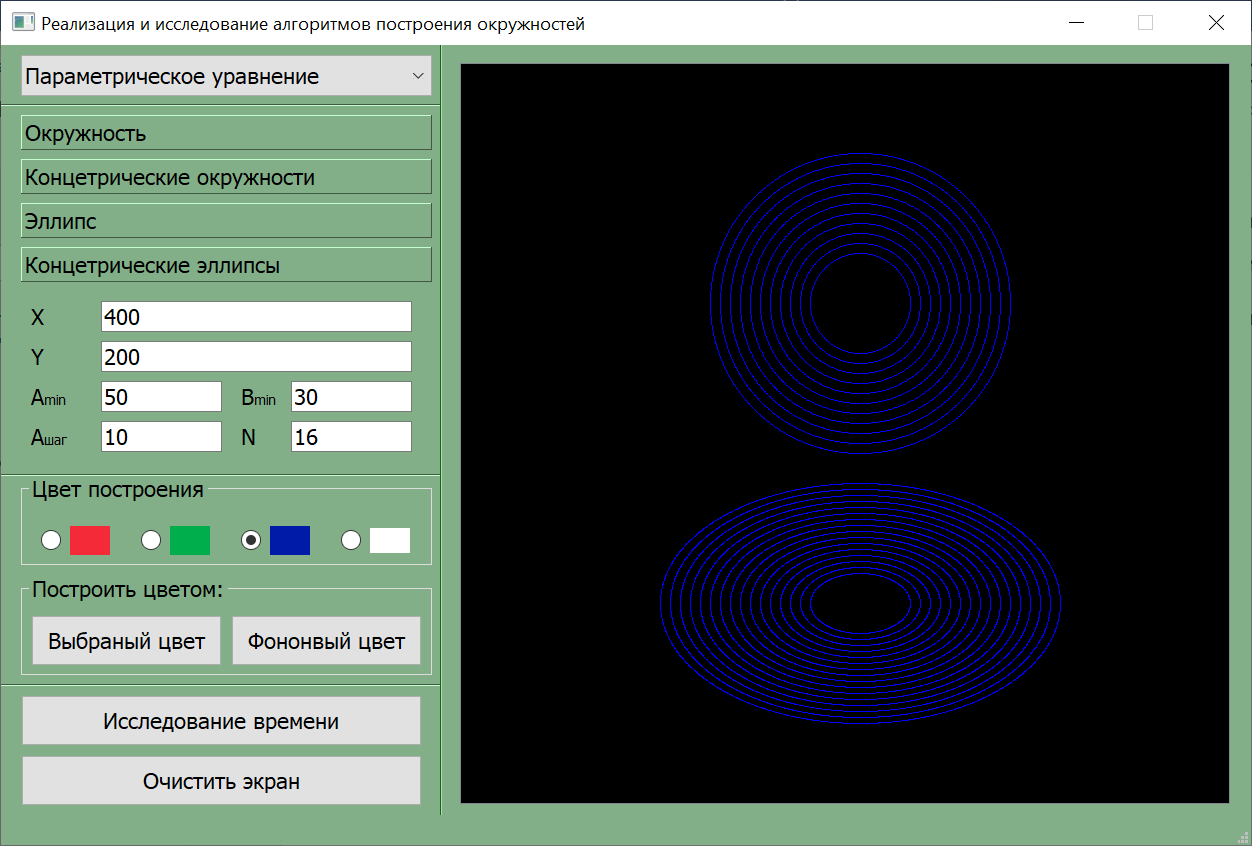
**Метод параметрического уравнения**

В данном случае, метод основан на использовании параметрических уравнений для окружности и эллипса:

и

Угол t проходит от 0 до pi/4 для окружности и pi/2 для эллипса, после чего отрисованные участки отражаются на остальные части фигур. Шаг выбирается согласно формуле . В случае окружности величина кривизны k постоянна и равна 1/r. Для эллипса это значение изменяется между и , поэтому для покрытия пикселями всех участков эллипса следует выбрать минимальный.

Результат работы алгоритмов:



Код функций:

|  |  |
| --- | --- |
| Окружность | Эллипс |
| def drawc\_parametric(self, x0, y0, r):  k = 1/r   deg = 0  while deg <= pi/4 + EPS:  x = round(r \* cos(deg))  y = round(r \* sin(deg))   self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   self.draw\_pixel(y + x0, x + y0)  self.draw\_pixel(y + x0, -x + y0)  self.draw\_pixel(-y + x0, x + y0)  self.draw\_pixel(-y + x0, -x + y0)   deg += k | def drawe\_parametric(self, x0, y0, a, b):  k = min(a / (b\*b), b / (a\*a))   deg = 0  while deg <= pi/2 + EPS:  x = round(a \* cos(deg))  y = round(b \* sin(deg))   self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   deg += k |

**Алгоритм Брезенхема**

Алгоритм работает, перемещаясь в трёх направлениях: на пиксель вправо, вниз, или по диагонали (вниз и вправо) и прорисовывая посещённые пиксели. Начало работы происходит в точке (0, r) для окружности и (0, b) для эллипса. Завершение работы цикла происходит в точке, где x=y для окружности и в точке (а, 0) для эллипса (по вышеописанным соображениям симметрии).

Следующей точкой является та, что наиболее приближена к идеальной окружности. Для оценки этого критерия используется понятие ошибки. В данном случае ошибкой является разность квадрантов расстояния диагонального пикселя и от центра окружности или эллипса. Сравнением этой величины с 0, возможно определить находится диагональная точка внутри окружности или снаружи, или же на самой окружности. При каждом шаге данное значение корректируется в зависимости от направления передвижения.

Возможно 5 случаев прохождения окружности:

1) Над или по горизонтальному пикселю

2) Между диагональным и горизонтальным пикселем

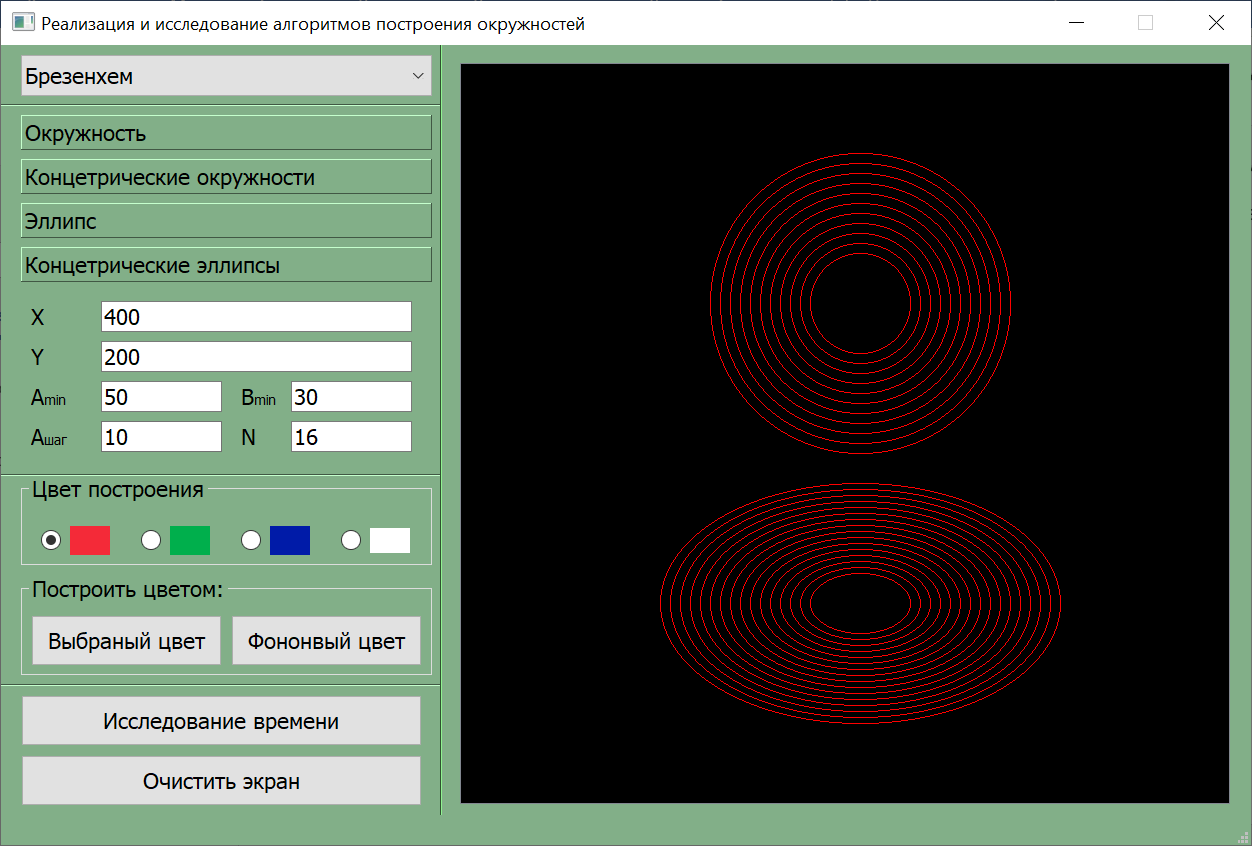
3) По диагональному пикселю

4) Между диагональным и вертикальным пикселем

5) Под или по вертикальному пикселю

В 1 и 2 случае, диагональный пиксель внутри окружности, поэтому d < 0. В 3 случае, диагональный пиксель на окружности, d = 0. В 4 и 5 случае, диагональный пиксель снаружи окружности, поэтому d > 0. Для выбора пикселя в 2 и 4 случае, сравниваются расстояния до центра между двумя пикселями, в более оптимизированной для этого форме.

Результат работы алгоритмов:



Код функций:

|  |  |
| --- | --- |
| Окружность | Эллипс |
| def drawc\_bres(self, x0, y0, r):  x = 0  y = r  d = 2 \* (1 - r)  while y >= x:  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   self.draw\_pixel(y + x0, x + y0)  self.draw\_pixel(y + x0, -x + y0)  self.draw\_pixel(-y + x0, x + y0)  self.draw\_pixel(-y + x0, -x + y0)   if d < 0:  e = 2\*(d + y) - 1  if e <= 0:  x += 1  d += 2\*x + 1  continue  elif d > 0:  e = 2\*(d - x) - 1  if e > 0:  y -= 1  d += -2\*y + 1  continue  x += 1  y -= 1  d += 2 \* (x - y + 1) | def drawe\_bres(self, x0, y0, a, b):  a2 = a\*a  b2 = b\*b   x = 0  y = b  d = 1/a2 + (1 - 2\*b)/b2  while y >= 0:  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   if d < 0:  e = 2\*d + (2\*y - 1)/b2  if e <= 0:  x += 1  d += (2\*x + 1)/a2  continue  elif d > 0:  e = 2\*d - (2\*x + 1)/a2  if e > 0:  y -= 1  d += (-2\*y + 1)/b2  continue  x += 1  y -= 1  d += (2\*x + 1)/a2 + (-2\*y + 1)/b2 |

**Алгоритм средней точки**

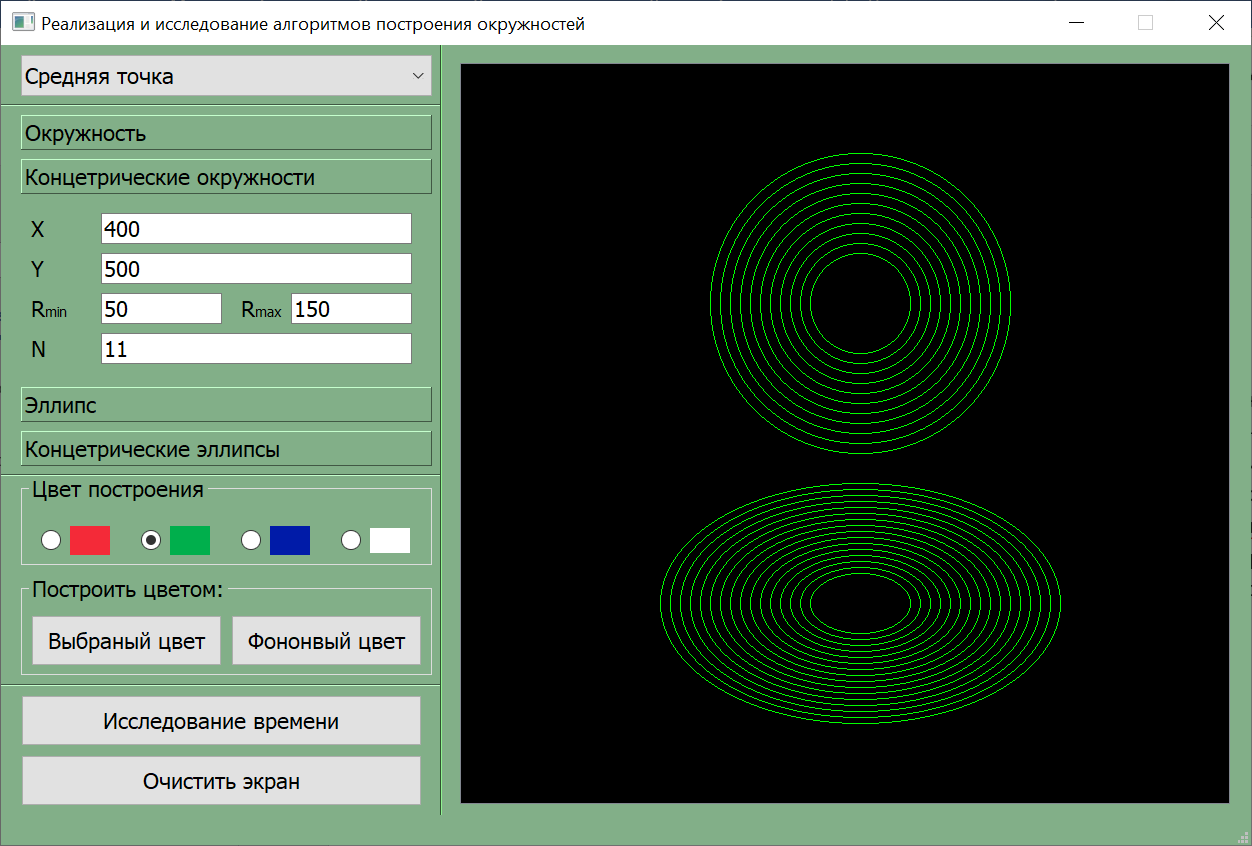
Алгоритм также работает в пошаговом режиме. Выделяется два интервала работы, разделённые точкой |dy/dx| = 1. В обоих случаях рассматриваемой величиной является положение средней точки относительно положения окружности или эллипса. На первом интервале средняя точка берётся между диагональной и вертикальной точкой, на втором – между диагональной и горизонтальной. В отличие от алгоритма Брезенхема, хранится и корректируется лишь одна величина – разность квадрантов расстояния средней точки от центра фигуры и расстояние до идеальной точки. Для отрисовки окружности используется только один интервал, для эллипса используются оба.

Формулы расчёта функций средней точки:

|  |  |  |
| --- | --- | --- |
|  | Функция средней точки | функции при шаге |
| Интервал 1 |  | Горизонтальный шаг:  Диагональный шаг: |
| Интервал 2 |  | Вертикальный шаг:  Диагональный шаг: |

В случае окружности, a=b=r, поэтому значения всех формул можно сократить на . Алгоритм может быть оптимизирован за счёт хранения и корректировки значений функции при шагах, так как они изменяются либо на , либо на

Результат работы алгоритмов:



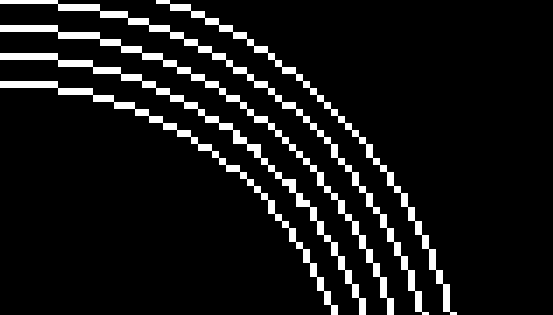
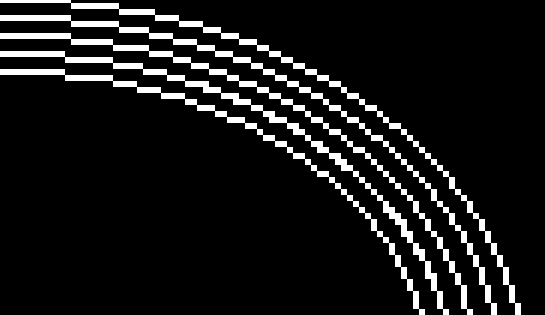
Код функций:

|  |  |
| --- | --- |
| Окружность | Эллипс |
| def drawc\_mid\_point(self, x0, y0, r):  x = 0  y = r   f\_tf = 1.25 - r  dy = 2\*y  dx = 1   while y >= x:  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)   self.draw\_pixel(y + x0, x + y0)  self.draw\_pixel(y + x0, -x + y0)  self.draw\_pixel(-y + x0, x + y0)  self.draw\_pixel(-y + x0, -x + y0)    x += 1  if f\_tf >= 0:  y -= 1  dy -= 2  f\_tf -= dy # 2\*y   dx += 2  f\_tf += dx # 2\*x + 1 | def drawe\_mid\_point(self, x0, y0, a, b):  a2 = a\*a  b2 = b\*b  ad = 2\*a2  bd = 2\*b2   x = 0  y = b  # Интервал 1  f\_tf = b2 - a2\*b + a2/4  dy = y\*ad  dx = b2  while a2\*y >= b2\*x:  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)    x += 1  if f\_tf >= 0:  y -= 1  dy -= ad  f\_tf -= dy # 2\*y\*a2  dx += bd  f\_tf += dx # b2 \* (2\*x + 1)   # Интервал 2  f\_tf -= b2\*(x+0.75) + a2\*(y-0.75)  dx = x\*bd  dy = a2 \* (2\*y - 1)  while y >= 0:  self.draw\_pixel(x + x0, y + y0)  self.draw\_pixel(x + x0, -y + y0)  self.draw\_pixel(-x + x0, y + y0)  self.draw\_pixel(-x + x0, -y + y0)    y -= 1  if f\_tf <= 0:  x += 1  dx += bd  f\_tf += dx # 2\*x\*b2  dy -= ad  f\_tf -= dy # a2 \* (2\*y - 1) |

**Сравнение визуальных характеристик**

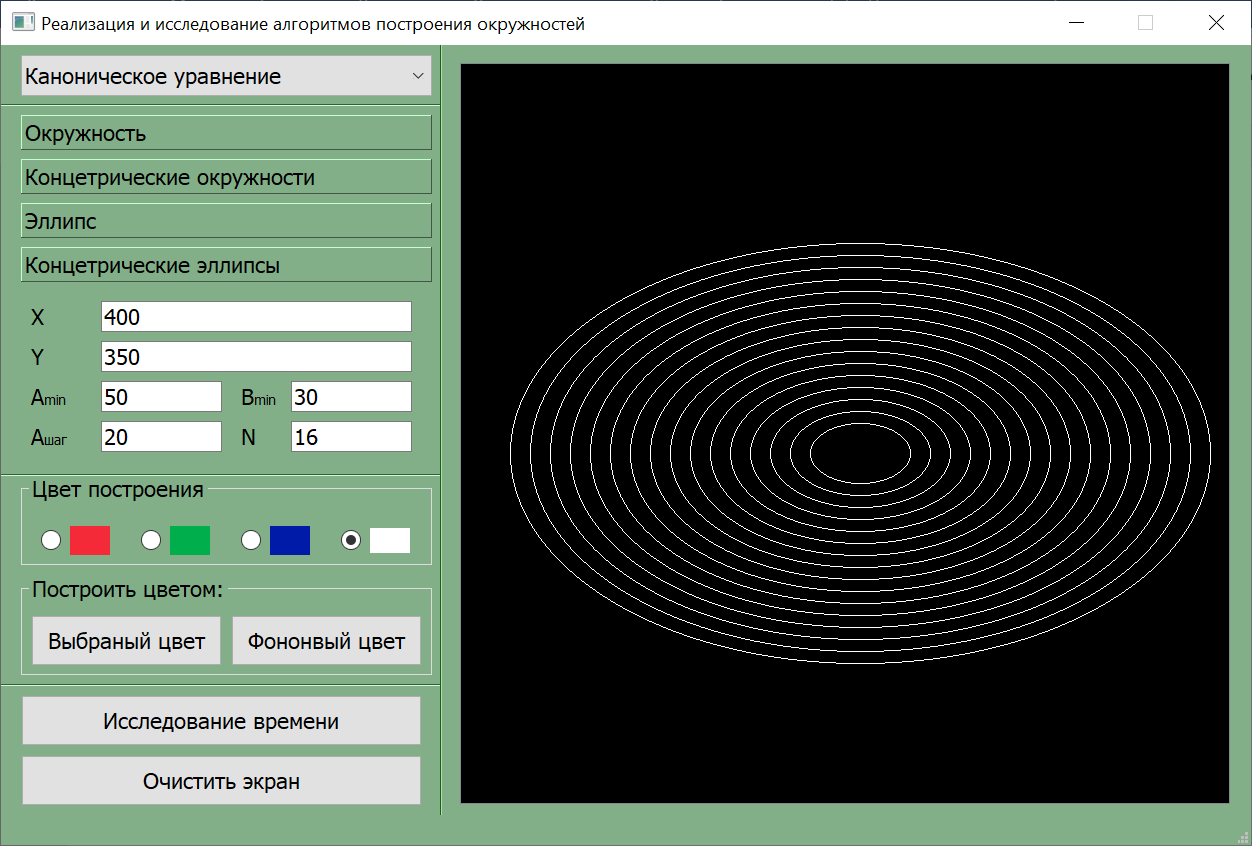
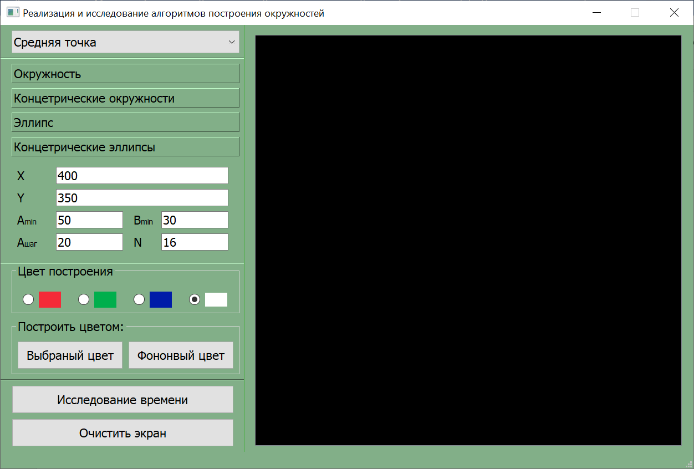
Изнутри-наружу изображены алгоритмы рисования окружности и эллипса:

* Канонического уравнения
* Параметрического уравнения
* Алгоритм Брезенхема
* Алгоритм средней точки
* Встроенный метод (в данном случае использована графическая библиотека PyQt5).

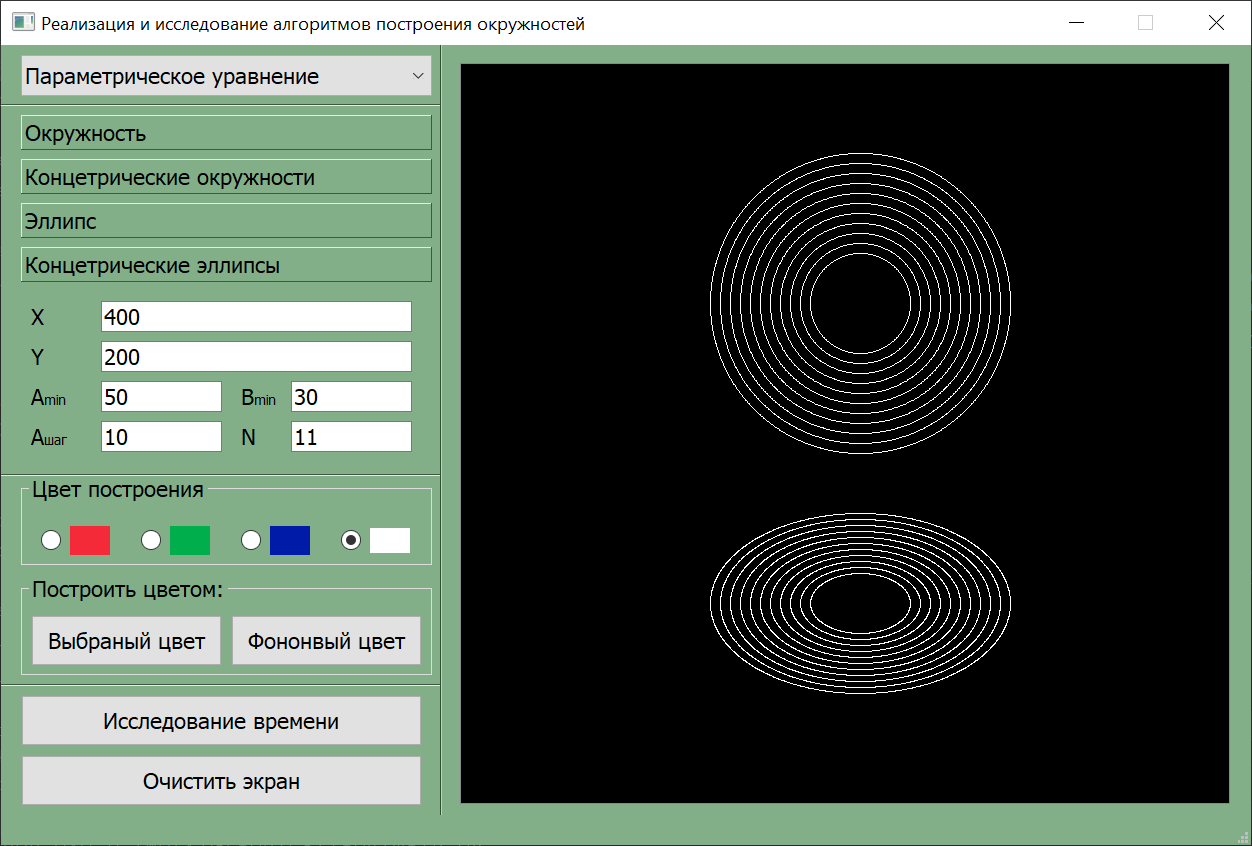


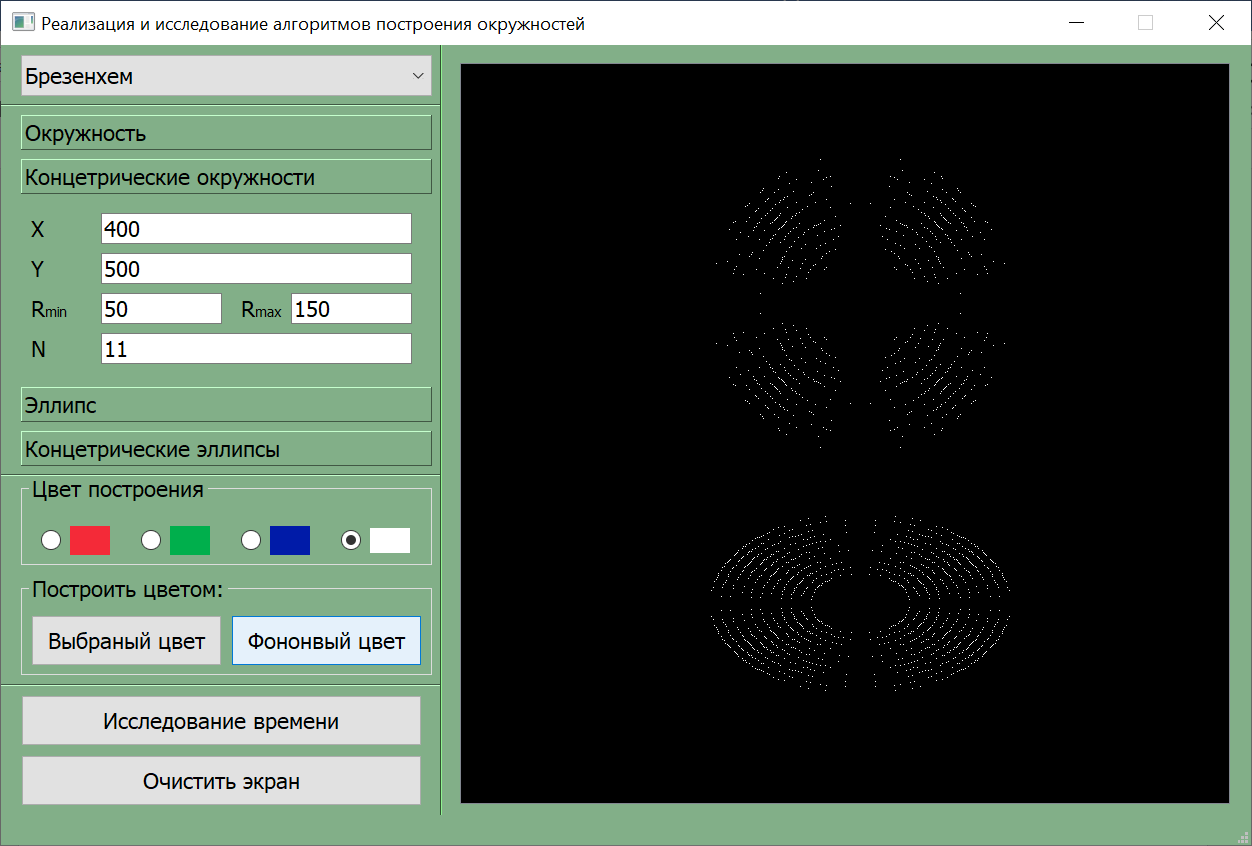
В результате сравнения визуальных характеристик путём закрашивания фоновым цветом, было выявлено, что алгоритмы Брезенхема, средней точки, канонического уравнения, а также встроенный метод дают одинаковые графические результаты как на окружности, так и на эллипсе:

Пример: сравнение алгоритма средней точки и канонического уравнения для эллипса

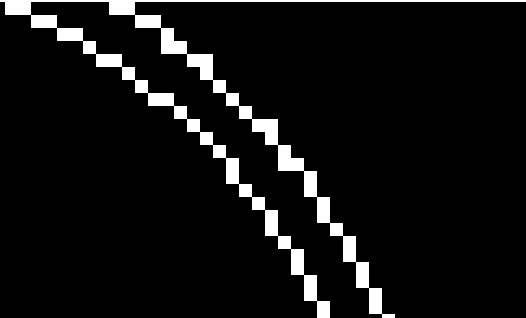


В свою очередь параметрическое уравнение визуально не совпадает в вышеописанными алгоритмами:



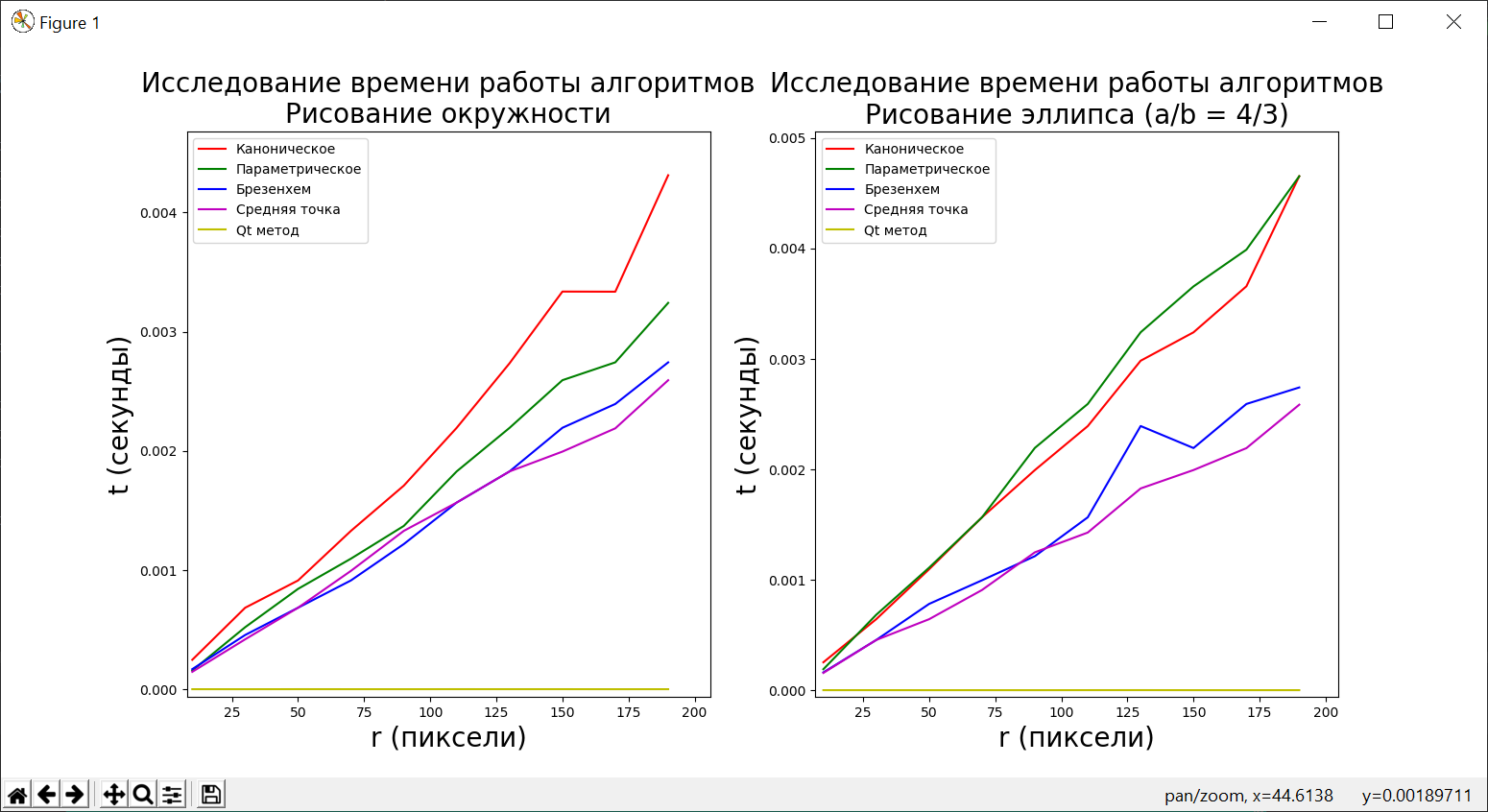


При более близком рассмотрении видно, что параметрическое уравнение (окружность снаружи) рисует некоторые лишние пиксели около “диагональных лестниц”:



**Анализ временных характеристик**

В программе реализовано средство замера времени работы различных алгоритмов для окружностей и эллипсов в зависимости от размера.



Из графика становится видно, что наибольшие временные затраты имеют алгоритмы канонического и параметрического уравнения. Менее затратны алгоритмы Брезенхема и средней точки (последний несколько быстрее). Из чего можно сделать вывод о том, что методы уравнений умеют заметно большие временные затраты. Также подтверждается линейная зависимость времени от радиуса или размера полуоси фигуры.

**Заключение**

В лабораторной работе были изучены различные алгоритмы построения растровых окружностей и эллипсов. Проведено сравнение их визуальных и временных характеристик, выявлены преимущества и недостатки. Выявлено, что методы уравнений, имея те же, или приблизительно те же, визуальные характеристики, ощутимо более затратны по временным характеристикам, чем алгоритмы Брезенхема и средней точки.