МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Вятский государственный университет»
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

РЕАЛИЗАЦИЯ АЛГОРИТМОВ РАСТРОВОЙ РАЗВЕРТКИ ЛИНИЙ

Отчет по лабораторной работе №1 дисциплины
«Компьютерная графика»

Выполнил студент группы ИВТ-21 _____/Рзаев А.Э./
Проверил старший преподаватель _____/Вожегов Д.В./

2016 г.

1 Постановка задачи

Написать программу, реализующую алгоритмы построения прямой: простой пошаговый алгоритм и алгоритмы Брезенхема для четырех- и восьмисвязной развертки. Проверить правильность работы программы, нарисовав, например, каждым алгоритмом семейство радиальных прямых, выходящих из одной точки с шагом 15 градусов.

2 Словесное описание алгоритмов

Простой пошаговый алгоритм:

Изначально для реализации алгоритма имеются следующие данные: m – угловой коэффициент ($-1 < m < 1$), (x_1; x_2) и (y_1; y_2) – координаты концов отрезка, цвет для подкрашивания точек отрезка, процедура для подкрашивания некоторым цветом пиксела с координатами (x, y) и функция округления x до ближайшего целого.

Если увеличивать с определенным шагом координату X, а затем находить координату Y используя уравнение прямой $Y = mX + b$ и подкрашивать пиксель с координатами (X, ROUND(Y)), то потребуется много времени (на нецелочисленные операции). Если шаг по X принять равным единице, то $m = dY/dX$ сводится к $m = dY$, т.е. изменение X на единицу приведет к изменению углового коэффициента на m. Таким образом, если $X(I + 1) = X(I) + 1$,то $Y(I + 1)=Y(I) + m$. Алгоритм корректно работает только для отрезков в первом и восьмом октандах. В остальных случаях требует модификации, что предлагается сделать студентам самостоятельно в ходе выполнения данной лабораторной работы. При модификации следует учесть, что при $m > 1$, единичный шаг по X приведёт к такому увеличению Y, при котором две соседние точки на прямой расположатся далеко друг от друга. Поэтому X и Y следует поменять, чтобы увеличивать на единицу Y, а X - на $dX = dY/m = 1/m$.

Общий алгоритм Брезенхема:

Суть алгоритма в следующем: в процессе работы одна из координат либо x, либо y (в зависимости от углового коэффициента) изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния – *e* между действительным положением отрезка и ближайшими координатами растра (e назовем управляющей переменной). Алгоритм построен так, что на каждом шаге проверяется лишь знак e и корректируется ее значение после каждого изменения очередной координаты. Значение исходной управляющей переменной:

$e = 2*(y2 - y1) - (x2 – x1)$,

где x1, y1, x2, y2 - координаты начальной и конечной точек отрезка. В каждом шаге при $e \geq 0$ значение y от предыдущего увеличивается на единицу, а e уменьшается на $2*(x2 - x1)$, в противном случае – y не

меняется, а значение e увеличивается на 2*(y2 - y1). В обоих случаях координата x следующего пиксела увеличивается на единицу от предыдущего значения.

3 Вывод

Для каждого алгоритма (простой, 8-связный алгоритм Брезенхема и 4-связный алгоритм Брезенхема) были написаны процедуры, реализующие эти алгоритмы и программа, позволяющая работать с этими процедурами. Наиболее естественным из приведённых алгоритмов – 8-связный алгоритм Брезенхема, т.к. при рисовании с помощью него получаются наименее угловатые линии, также он более оптимизирован, чем простой алгоритм. Соответственно, наиболее плохим вариантом является простой алгоритм, т.к. при углах, близких к 90° и 270° линия распадается на отдельные точки.

Блок-схемы алгоритмов, листинг процедур, реализующих их, и экранные формы программы приведены в приложениях А, Б и В.

Приложение А
(обязательное)
Блок-схемы алгоритмов

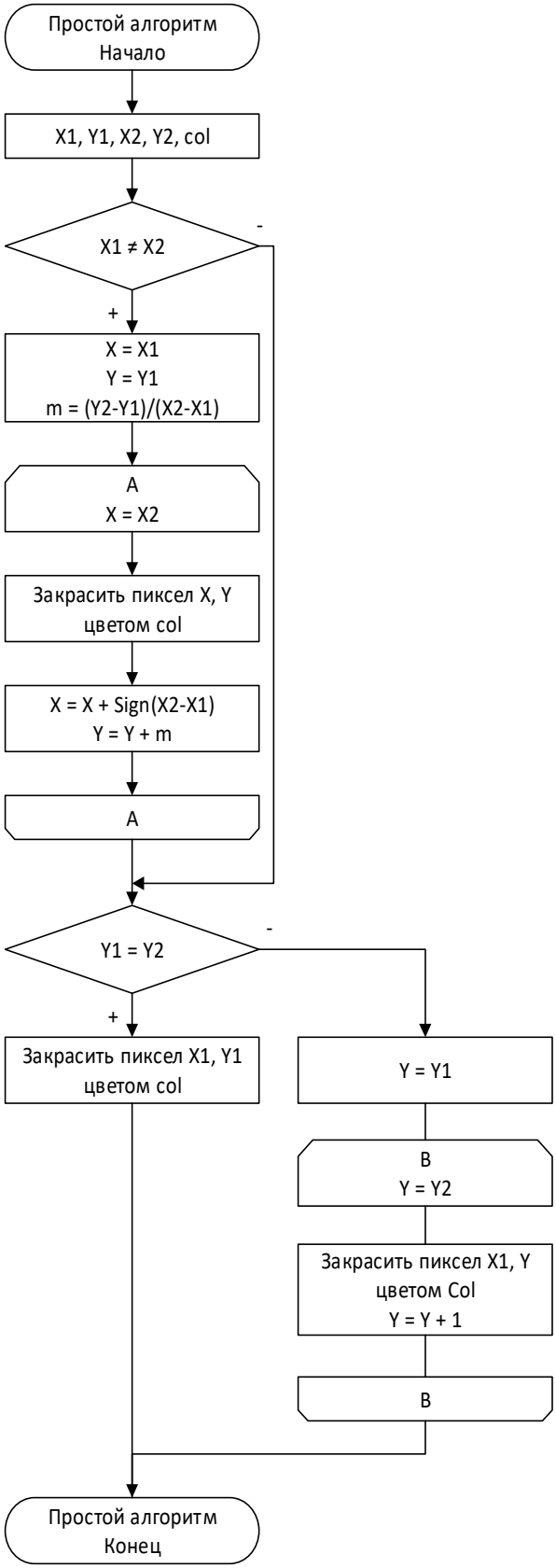

Рисунок А.1 – Схема простого алгоритма развертки отрезка

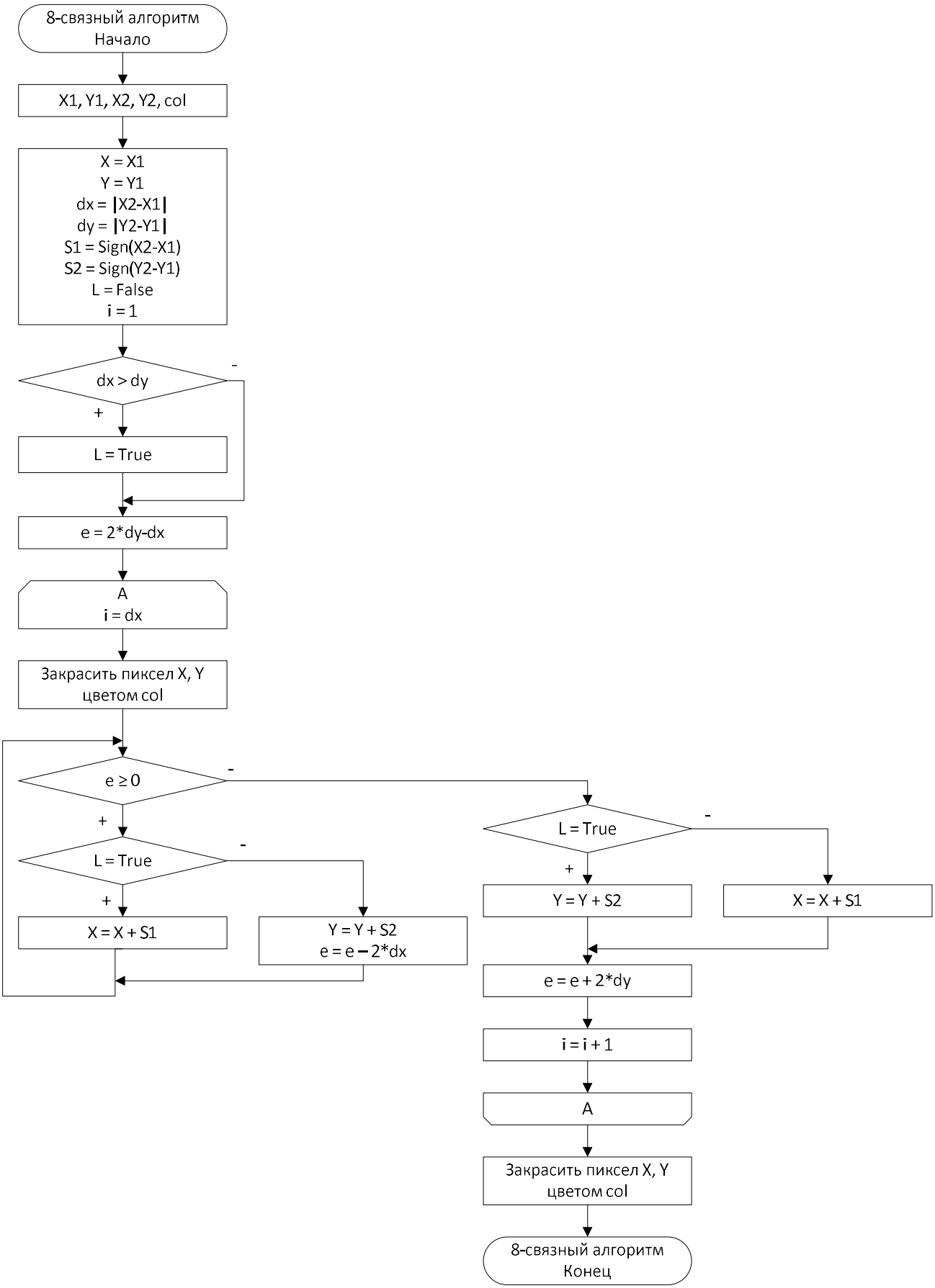

Рисунок А.2 – Схема восьмисвязного алгоритма Брезенхема

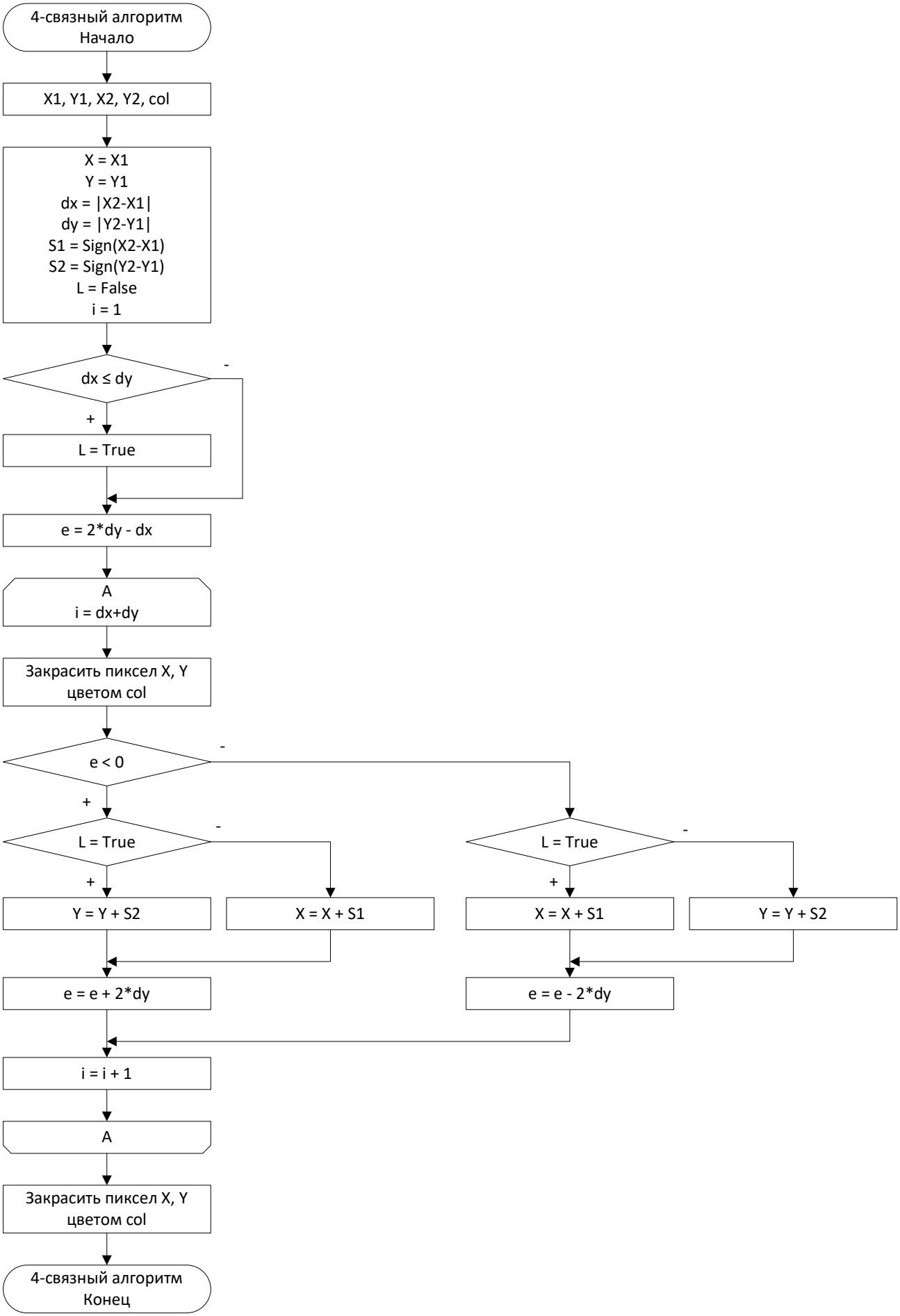

Рисунок А.3 – Схема четырехсвязного алгоритма Брезенхема

Приложение Б
(обязательное)
Листинг процедур

```
def draw_line_simple(self, pos1, pos2):
    # простой алгоритм
    x1, y1 = pos1
    x2, y2 = pos2
    if x1 != x2:
        m = (y2 - y1) / (x2 - x1)
        if x1 < x2:
            y = y1
            for x in range(x1, x2 + 1):
                self.draw_pixel(x, round(y))
                y += m
        elif x1 > x2:
            y = y2
            for x in range(x2, x1 + 1):
                self.draw_pixel(x, round(y))
                y += m
    elif y1 == y2:
        self.draw_pixel(x1, y1)
    else:
        for y in range(min(y1, y2), max(y1, y2) + 1):
            self.draw_pixel(x1, y)

def draw_line_brez8(self, pos1, pos2):
    # 8-связный алгоритм Брезенхема
    sign = lambda a: 1 if a >= 0 else -1
    x1, y1 = pos1; x2, y2 = pos2
    x, y = x1, y1; dx, dy = abs(x2 - x1), abs(y2 - y1)
    s1, s2 = sign(x2 - x1), sign(y2 - y1)

    if dy > dx:
        dx, dy = dy, dx
        f = True
    else:
        f = False

    e = 2 * dy - dx
    for _ in range(1, dx + 1):
        self.draw_pixel(x, y)
        while e >= 0:
            if f:
                x += s1
            else:
                y += s2
            e -= 2 * dx

        if f:
            y += s2
        else:
            x += s1

        e += 2 * dy

    self.draw_pixel(x, y)

def draw_line_brez4(self, pos1, pos2):
    # 4-связный алгоритм Брезенхема
    sign = lambda a: 1 if a >= 0 else -1
    x1, y1 = pos1; x2, y2 = pos2
    x, y = x1, y1; dx, dy = abs(x2 - x1), abs(y2 - y1)
    s1, s2 = sign(x2 - x1), sign(y2 - y1)
```

```
if dy >= dx:
    dx, dy = dy, dx
    f = True
else:
    f = False

e = 2 * dy - dx
for _ in range(1, dx + dy):
    self.draw_pixel(x, y)

    if e < 0:
        if f:
            y += s2
        else:
            x += s1
        e += 2 * dy
    else:
        if f:
            x += s1
        else:
            y += s2
        e -= 2 * dx

self.draw_pixel(x, y)
```

Приложение В
(обязательное)
Экранные формы программы

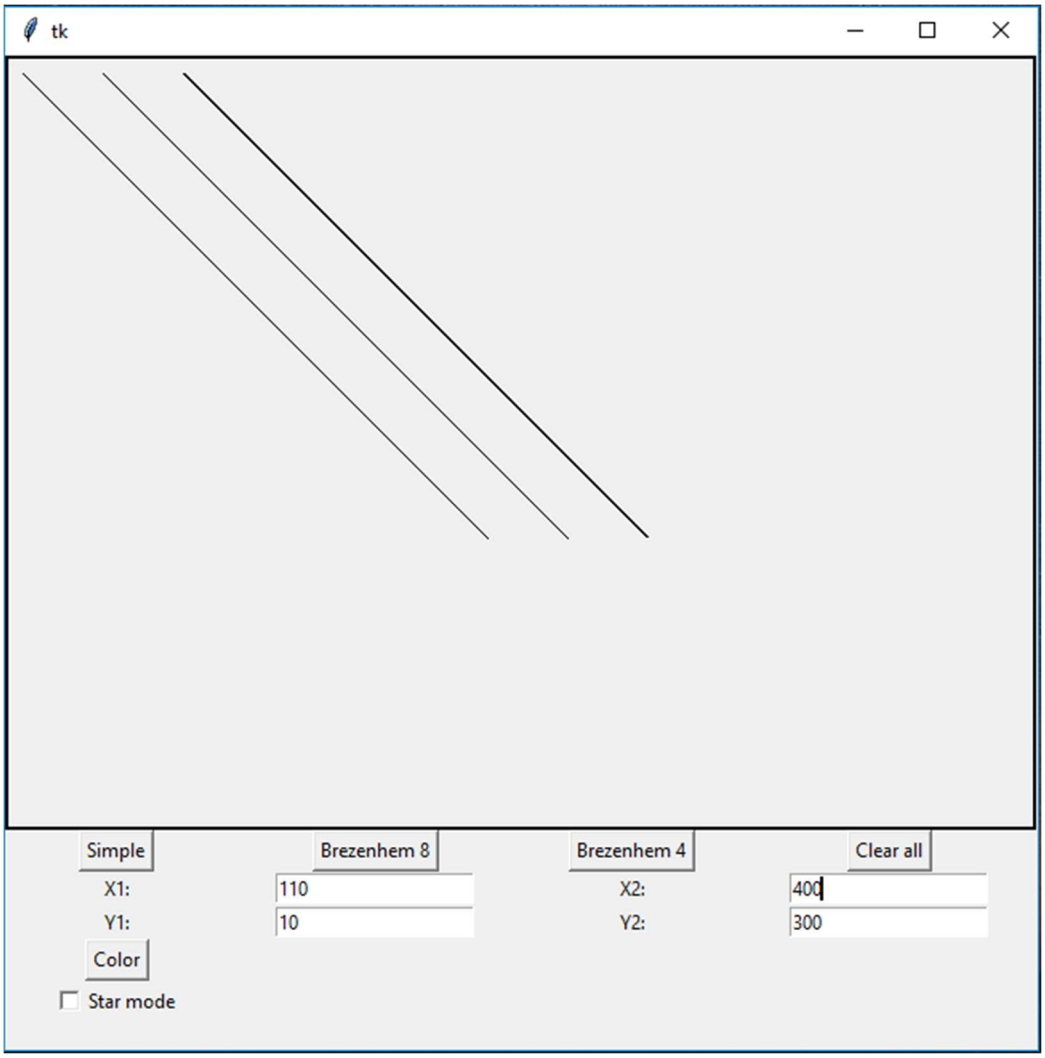

Рисунок В.1 – Развертка отрезка 3 алгоритмами

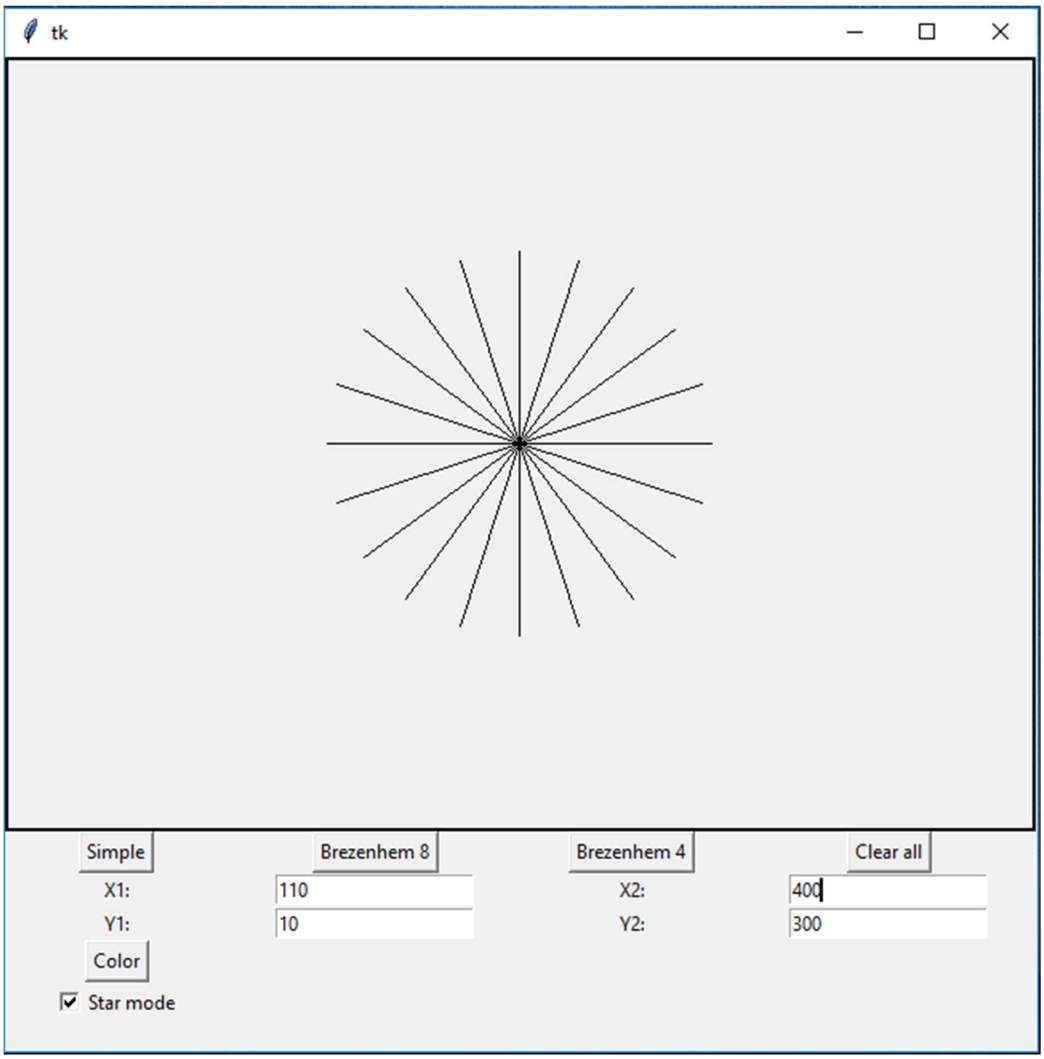

Рисунок В.2 – Развертка радиальных прямых