|  |  |
| --- | --- |
| 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 № 7**

|  |  |
| --- | --- |
| **Тема: Изучение и программная реализация простого алгоритма отсечения отрезка.**  **Студент** ВоякинА. Я.  **Группа ИУ7-44Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:**

Цель работы: изучение и программная реализация алгоритма отсечения отрезка. Алгоритм отсечения - простой. Необходимо обеспечить ввод регулярного отсекателя - прямоугольника. Высветить его первым цветом. Также необходимо обеспечить ввод нескольких (до десяти) различных отрезков (высветить их вторым цветом). Отрезки могут иметь произвольное расположение: горизонтальные, вертикальные, имеющие произвольный наклон. Ввод осуществлять с помощью мыши и нажатия других клавиш. Выполнить отсечение отрезков, показав результат третьим цветом. Исходные отрезки не удалять.

**Теоретические сведения:**

Регулярным (стандартным) отсекателем на плоскости является прямоугольник со сторонами, параллельными координатным осям объектного пространства или экрана. Такое окно задается левым, правым, верхним и нижним двумерными ребрами. Для выполнения отсечения необходимо задать абсциссы X л, X п левого и правого ребер и ординаты Y н ,Y в нижнего и верхнего ребер. Цель отсечения будет состоять в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Поскольку отсечение выполняется достаточно часто в машинной графике, то актуальным является вопрос обеспечения высокой эффективности работы этих алгоритмов. Часто большое количество исследуемых точек и отрезков реальной сцены лежит полностью в пределах окна или полностью за пределами окна. Поэтому желательно быстро определять такие точки и отрезки. Точки, лежащие целиком внутри окна, удовлетворяют условию (Xл ≤ ≤ X ≤ Xп) & (Yн ≤ Y ≤ Yп), где (X, Y) - координаты точки. Считается, что точки, лежащие на границе окна, принадлежат внутренней области окна.

Отрезок целиком лежит внутри окна, если обе его концевые точки лежат внутри окна. Однако обратное утверждение, к сожалению, верно не всегда. Отрезок, концевые точки которого лежат вне окна, может быть как полностью невидимым, так и частично видимым. Полностью невидимым называется отрезок, целиком лежащий вне отсекателя. Частично видимым называется отрезок, одна часть которого лежит в пределах отсекателя, а другая - вне его. Если обе концевые точки отрезка невидимы, то он будет заведомо невидимым, если они (вершины отрезка) одновременно лежат левее или правее или ниже или выше окна.

С помощью следующего простого теста можно определить полностью видимые, часть полностью невидимых отрезков, остающиеся отрезки требуют дальнейшего исследования для установления факта их частичной видимости или полной невидимости. Собственно для идентификации именно таких отрезков и требуются специальные алгоритмы отсечения.

Обозначим концевые точки отрезка как (a,b). Тест определяет полностью видимый отрезок как такой, у которого ни одна из координат концов отрезка не находится вне отсекателя. Если же оба конца отрезка лежат целиком слева, справа, снизу или сверху от окна, то он является невидимым.

1. Определение полностью видимого отрезка:

если (Xa < Xл) || (Xb > Xл) то переход к п.2;

если (Xb < Xл) || (Xb > Xл) то переход к п.2;

если (Ya < Yн) || (Ya > Yв) то переход к п.2;

если (Yb < Yн) || (Yb > Yв) то переход к п.2; иначе отрезок видим, переход к п.3.

2. Определение полностью невидимого отрезка:

если (Xa < Xл) & (Xb < Xл) то переход к п.3 (отрезок невидим);

если (Xa > Xп) & (Xb > Xп) то переход к п.3 (отрезок невидим);

если (Ya < Yн) & (Yb < Yв) то переход к п.3 (отрезок невидим);

если (Ya > Yв) & (Yb > Yв) то переход к п.3 (отрезок невидим);

иначе отрезок частично видим или является невидимым (требуется дополнительный анализ).

3. Конец.

Д.Коэн и А.Сазерленд предложили формализовать описанную процедуру. Для определения принадлежности точки одной из девяти областей, на которые разбивается плоскость продолжениями ребер отсекателя, вводится четырехразрядный (битовый) код (рис.3.6.2). Биты этого кода заполняются по следующему правилу:

T1 = 1, если точка лежит левее окна, и 0 в противном случае;

T2 = 1, если точка лежит правее окна, и 0 в противном случае;

T3 = 1, если точка лежит ниже окна, и 0 в противном случае;

T4 = 1, если точка лежит выше окна, и 0 в противном случае.

Первым считается крайний правый бит.

Точка лежит внутри отсекателя, если все биты ее кода - нулевые. Отрезок будет полностью видимым, если коды обоих его концов нулевые. Введенные коды концов отрезка можно использовать и для определения полностью невидимых отрезков. Если побитное логическое произведение кодов концов отрезка не равно нулю, то отрезок является полностью невидимым и его можно отбросить. Однако, если логическое произведение равно нулю, то ничего определенного об отрезке сказать нельзя. В этом случае отрезок может быть целиком или частично видимым или даже целиком невидимым. Таким образом, на основе анализа кодов концов отрезка легко определяются полностью видимые и тривиально невидимые отрезки. Отрезки, для которых логическое произведение кодов концов равно нулю, а побитная сумма этих кодов для концевых точек (одной или обеих) отлична от нуля, требуется выполнение собственно алгоритма отсечения с целью нахождения точек пересечения отрезка с границами отсекателя или установления факта полной невидимости отрезка.

Пересечение отрезка с ребром отсекателя ищется как решение системы двух уравнений с двумя неизвестными. Уравнениями системы являются уравнения, задающие положение отрезка и ребра отсекателя на плоскости. Решение такой системы - координаты точки пересечения этих двух отрезков. Уравнения отрезков можно задавать как в параметрической, так и в непараметрической формах. В алгоритмах отсечения в силу большего удобства решения конкретной задачи используется параметрический способ задания отрезка в виде:

Y=m(X - X1) + Y1 или Y=m(X - X2) + Y2, где (X1, Y1) и (X2, Y2) - координаты двух точек, через которые проходит отрезок, а m = (Y2 - Y1) / (X2 - X1) - тангенс угла наклона отрезка.

Поскольку для отсекателя заданы абсциссы левой и правой границ, ординаты нижней и верхней границ, то для нахождения координат точек пересечения отрезка с границами отсекателя достаточно подставить в уравнение отрезка абсциссы боковых ребер отсекателя для нахождения ординат точек пересечения с этими ребрами. Поскольку точка пересечения принадлежит одновременно и отрезку, и соответствующей границе отсекателя, то для нахождения точки пересечения с боковыми ребрами необходимо найти только ординату точки пересечения, а абсцисса совпадает с абсциссой ребра отсекателя. При нахождении точки пересечения с нижним или верхним ребрами отсекателя известна ордината ребра отсекателя, ее значение совпадает с ординатой точки пересечения и остается найти только абсциссу точки пересечения. Для этого известная ордината подставляется в уравнение отрезка, откуда и находится искомая абсцисса.

В итоге координаты искомых точек пересечения отрезка с границами отсекателя находятся из следующих выражений:

Y= m(X л -X 1 )+Y 1 , m ≠ inf (с левой границей)

Y= m(X п -X 1 )+Y 1 , m ≠ inf (с правой границей)

X=(1/m)(Y н -Y 1 )+X 1 , m ≠ 0 (с нижней границей)

X=(1/m)(Y в -Y 1 )+X 1 , m ≠ 0 (с верхней границей).

Указанные ограничения на значение тангенса угла наклона отрезка выполняются, т.к. в алгоритмах предварительно анализируется характер отрезка (вертикальный, горизонтальный или общего положения) и в дальнейшем для вертикальных отрезков не ищутся точки пересечения с боковыми ребрами, а для горизонтальных отрезков не ищутся точки пересечения с нижним и верхним ребрами отсекателя.

Изложенные соображения положены в основу простого алгоритма отсечения отрезка регулярным отсекателем. На основе сформированных кодов концов отрезка проверяется полная видимость, невидимость отрезка или видимость одной из вершин отрезка. Если отрезок не является полностью видимым, то далее ищутся точки пересечения отрезка с границами отсекателя. При этом сначала анализируется возможность пересечения отрезка с очередной границей отсекателя. Если пересечение возможно, то находятся координаты точки пересечения, после чего проводится анализ ее корректности. Под корректностью понимается принадлежность найденной точки ребру отсекателя, а не его продолжению. В случае корректности найденного пересечения точка заносится в результат и ищется вторая точка пересечения отрезка с границами отсекателя. После нахождения двух точек пересечения визуализируется полученный результат. Если же пересечение не является корректным, то выполненные вычисления оказываются фактически напрасными и в этом случае ищется пересечение со следующей стороной отсекателя. Если не было найдено ни одного корректного пересечения, то отрезок является невидимым.

Простой алгоритм отсечения является довольно эффективным, за счет возможности сразу определить тривиально видимые отрезки, а также сразу откинуть часть невидимых отрезков. Из минусов алгоритма можно выделить необходимость следить за точкой пересечения.

**Инструкция к программе:**

Рисование отсекателя происходит с помощью правой кнопки мыши.

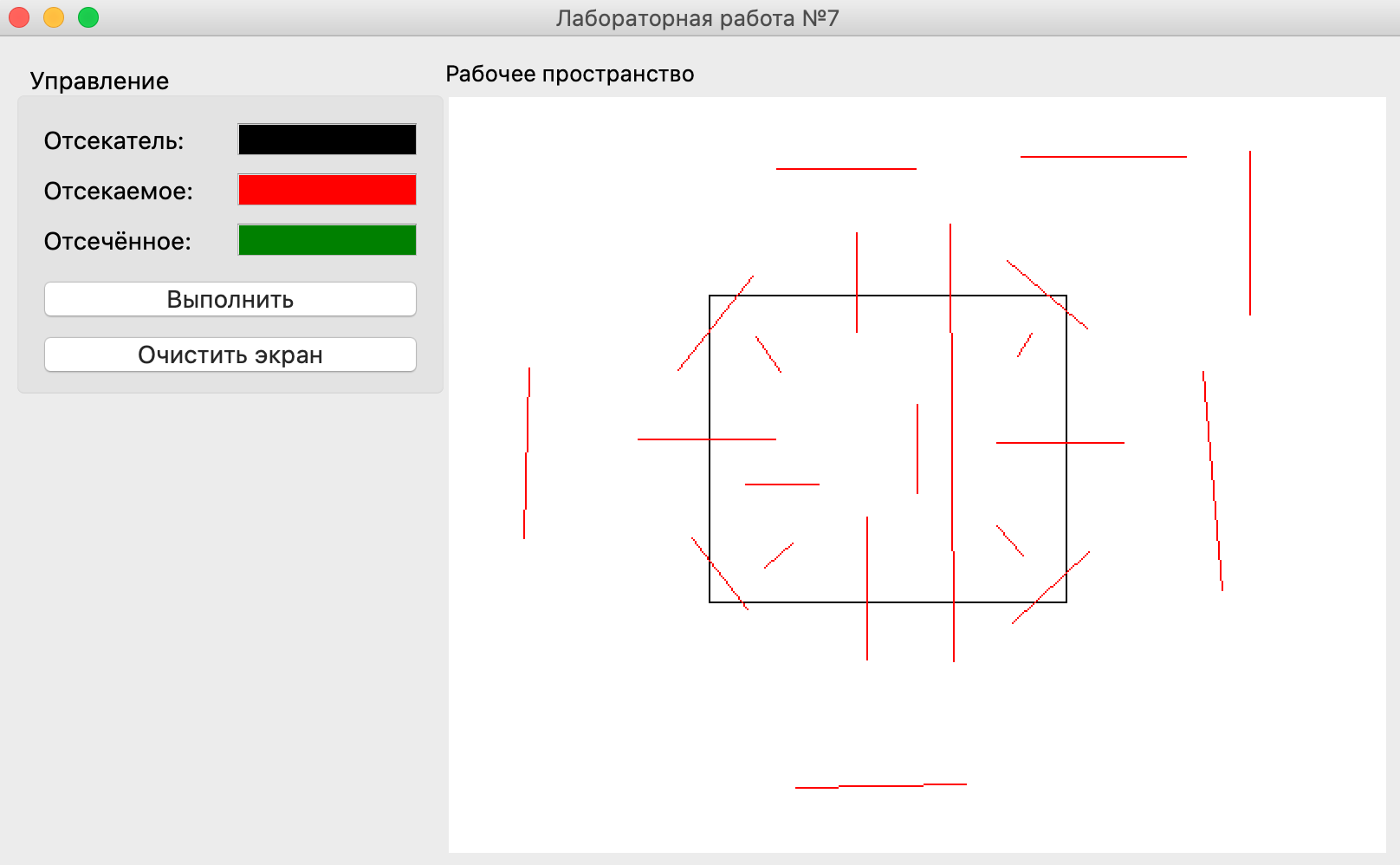
Рисование линий происходит с помощью левой кнопки мыши.

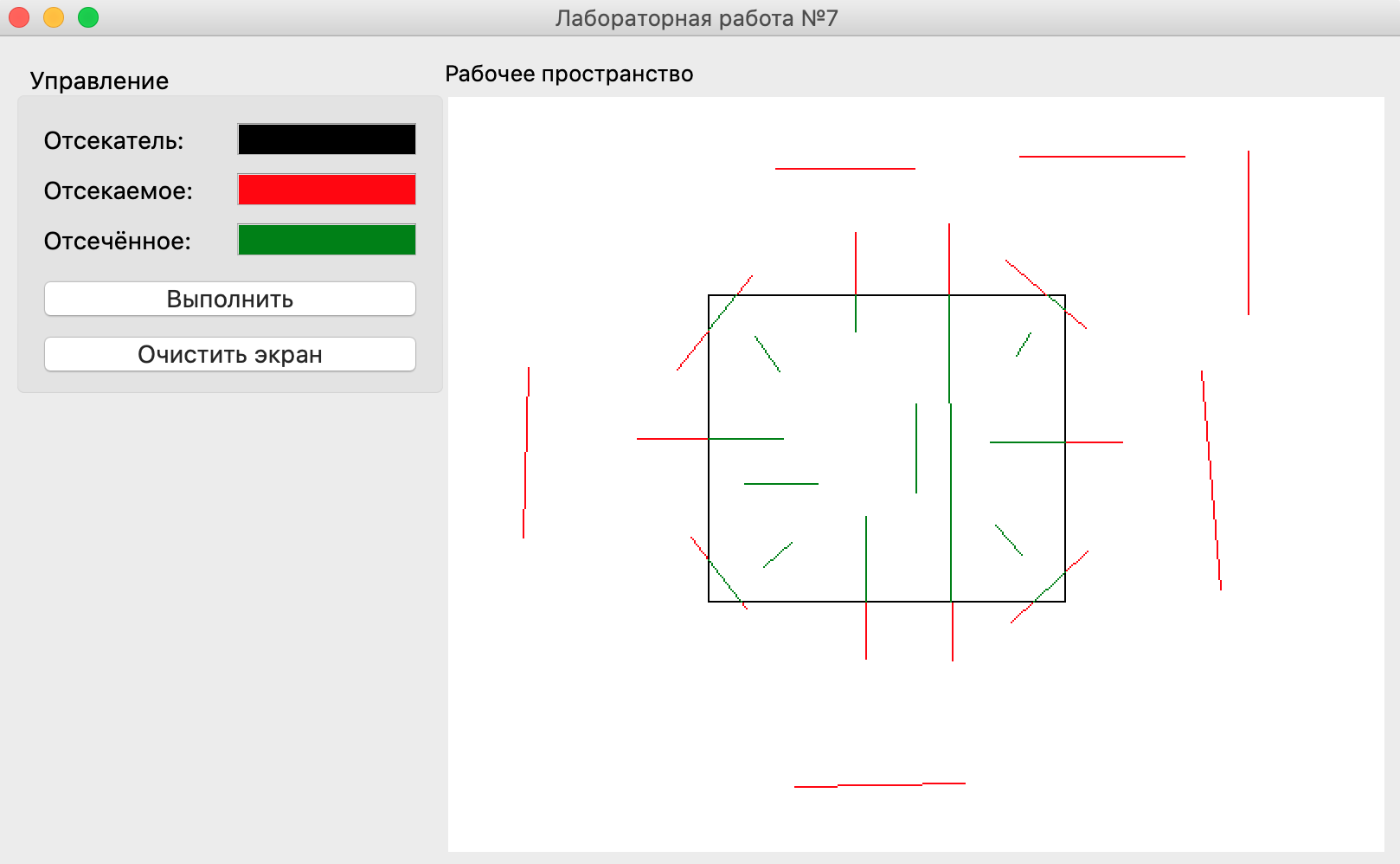
Рисование вертикальных и горизонтальных линий происходит с помощью левой кнопки мыши с зажатыми клавишами «V» и «H» соответственно.

Выбор цвета происходит по нажатии на прямоугольник с цветом, который хотите поменять.

**Код алгоритма:**

def proc(*self*):  
 *self*.img = *self*.pm.toImage()  
 for i in range(0, len(*self*.p\_x), 2):  
 *self*.makeCut(*self*.p\_x[i], *self*.p\_y[i], *self*.p\_x[i + 1], *self*.p\_y[i + 1])  
 *self*.pm = *self*.pm.fromImage(*self*.img)  
 *self*.label.setPixmap(*self*.pm)  
 *self*.label.repaint()  
  
def makeCut(*self*, p1\_x, p1\_y, p2\_x, p2\_y):  
 t1 = [0, 0, 0, 0]  
 t2 = [0, 0, 0, 0]  
 p1res\_x = p1\_x  
 p1res\_y = p1\_y  
 p2res\_x = p2\_x  
 p2res\_y = p2\_y  
 sum1 = 0  
 sum2 = 0  
 mult = 0  
 flag = True  
 number = 0  
 m = inf  
 if p1\_x < *self*.cutter\_ld\_x: t1[3] = 1  
 if p1\_x > *self*.cutter\_ru\_x: t1[2] = 1  
 if p1\_y < *self*.cutter\_ld\_y: t1[1] = 1  
 if p1\_y > *self*.cutter\_ru\_y: t1[0] = 1  
 if p2\_x < *self*.cutter\_ld\_x: t2[3] = 1  
 if p2\_x > *self*.cutter\_ru\_x: t2[2] = 1  
 if p2\_y < *self*.cutter\_ld\_y: t2[1] = 1  
 if p2\_y > *self*.cutter\_ru\_y: t2[0] = 1  
 for i in range(0, 4):  
 sum1 += t1[i]  
 sum2 += t2[i]  
 if sum1 == 0 and sum2 == 0:  
 goto .q7  
 for i in range(0, 4):  
 mult += (t1[i] + t2[i]) // 2  
 if mult != 0:  
 flag = False  
 goto .q7  
 if sum1 == 0:  
 number = 1  
 p1res\_x = p1\_x; p1res\_y = p1\_y  
 p\_x = p2\_x; p\_y = p2\_y  
 goto .q2  
 if sum2 == 0:  
 number = 2  
 p1res\_x = p2\_x; p1res\_y = p2\_y  
 p\_x = p1\_x; p\_y = p1\_y  
 goto .q2  
 number = 0  
  
 label .q1  
 if number != 0:  
 if number == 1: p1res\_x = p\_x; p1res\_y = p\_y  
 else: p2res\_x = p\_x; p2res\_y = p\_y  
 number += 1  
 if number > 2:  
 goto .q7  
 if number == 1: p\_x = p1\_x; p\_y = p1\_y  
 else: p\_x = p2\_x; p\_y = p2\_y  
  
 label .q2  
 if p2\_x - p1\_x == 0:  
 goto .q4  
 m = (p2\_y - p1\_y) / (p2\_x - p1\_x)  
 if *self*.cutter\_ld\_x < p\_x:  
 goto .q3  
 y = m \* (*self*.cutter\_ld\_x - p\_x) + p\_y  
 if *self*.cutter\_ru\_y < y < *self*.cutter\_ld\_y:  
 goto .q3  
 p\_y = y; p\_x = *self*.cutter\_ld\_x  
 goto .q1  
  
 label .q3  
 if *self*.cutter\_ru\_x > p\_x:  
 goto .q4  
 y = m \* (*self*.cutter\_ru\_x - p\_x) + p\_y  
 if *self*.cutter\_ru\_y < y < *self*.cutter\_ld\_y:  
 goto .q4  
 p\_y = y; p\_x = *self*.cutter\_ru\_x  
 goto .q1  
  
 label .q4  
 if m == 0:  
 goto .q1  
 if *self*.cutter\_ru\_y > p\_y:  
 goto .q5  
 x = (1 / m) \* (*self*.cutter\_ru\_y - p\_y) + p\_x  
 if *self*.cutter\_ru\_x < x < *self*.cutter\_ld\_x:  
 goto .q5  
 p\_x = x; p\_y = *self*.cutter\_ru\_y  
 goto .q1  
  
 label .q5  
 if *self*.cutter\_ld\_y < p\_y:  
 return  
 x = (1 / m) \* (*self*.cutter\_ld\_y - p\_y) + p\_x  
 if *self*.cutter\_ru\_x < x < *self*.cutter\_ld\_x:  
 goto .q6  
 p\_x = x; p\_y = *self*.cutter\_ld\_y  
 goto .q1  
  
 label .q6  
 flag = False  
  
 label .q7  
 if not flag:  
 goto .q8  
 *self*.bres\_int(p1res\_x, p1res\_y, p2res\_x, p2res\_y, *self*.colour\_line\_in)  
  
 label .q8  
 return

**Демонстрация работы программы:**

****

**Полный код программы:**

import sys  
from PyQt5 import QtWidgets, QtGui, QtCore  
import design  
from math import fabs, inf  
from trace\_goto import goto, label  
  
class Visual(QtWidgets.QMainWindow, design.Ui\_MainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.label.setPalette(QtGui.QPalette(QtCore.Qt.white))  
 self.white = QtGui.QColor(QtCore.Qt.white)  
  
 self.colour\_line\_out = QtGui.QColor(QtCore.Qt.red)  
 self.colour\_line\_in = QtGui.QColor(QtCore.Qt.darkGreen)  
 self.colour\_cutter = QtGui.QColor(QtCore.Qt.black)  
  
 self.labelColourCutter.setPalette(QtGui.QPalette(QtCore.Qt.black))  
 self.labelColourLineIn.setPalette(QtGui.QPalette(QtCore.Qt.darkGreen))  
 self.labelColourLineOut.setPalette(QtGui.QPalette(QtCore.Qt.red))  
  
 self.pm = QtGui.QPixmap(541, 436)  
 self.pm.fill(self.white)  
 self.img = self.pm.toImage()  
  
 self.horizontal = False  
 self.vertical = False  
  
 self.p\_x = []  
 self.p\_y = []  
  
 self.left\_click\_num = 0  
 self.cutter\_ld\_x = 0  
 self.cutter\_ld\_y = 0  
 self.cutter\_ru\_x = 0  
 self.cutter\_ru\_y = 0  
  
 # Связи кнопок и функций.  
 self.pushButtonScreenClean.clicked.connect(self.clean\_screen)  
 self.pushButtonProc.clicked.connect(self.proc)  
  
 def bres\_int(self, x\_start, y\_start, x\_end, y\_end, colour):  
 dx = x\_end - x\_start  
 dy = y\_end - y\_start  
 sx = sign(dx)  
 sy = sign(dy)  
 dx = fabs(dx)  
 dy = fabs(dy)  
 if dy >= dx:  
 dx, dy = dy, dx  
 fl = 1  
 else:  
 fl = 0  
 f = 2 \* dy - dx  
 x = round(x\_start)  
 y = round(y\_start)  
 i = 1  
 while i <= dx + 1:  
 self.img.setPixel(x, y, colour.rgb())  
 if f >= 0:  
 if fl == 1:  
 x += sx  
 else:  
 y += sy  
 f -= 2 \* dx  
 if f <= 0:  
 if fl == 1:  
 y += sy  
 else:  
 x += sx  
 f += 2 \* dy  
 i += 1  
  
 def mousePressEvent(self, press):  
 if press.pos().x() < self.label.pos().x() or press.pos().y() < self.label.pos().y():  
 return  
 self.img = self.pm.toImage()  
 if press.button() == QtCore.Qt.LeftButton:  
 if self.vertical:  
 self.p\_x.append(self.p\_x[len(self.p\_x) - 1])  
 else:  
 self.p\_x.append((press.pos() - self.label.pos()).x())  
 if self.horizontal:  
 self.p\_y.append(self.p\_y[len(self.p\_y) - 1])  
 else:  
 self.p\_y.append((press.pos() - self.label.pos()).y())  
 if len(self.p\_x) % 2 == 0:  
 self.bres\_int(self.p\_x[len(self.p\_x) - 2], self.p\_y[len(self.p\_y) - 2],  
 self.p\_x[len(self.p\_x) - 1], self.p\_y[len(self.p\_y) - 1], self.colour\_line\_out)  
 elif press.button() == QtCore.Qt.RightButton:  
 if self.left\_click\_num == 0:  
 self.cutter\_ld\_x = (press.pos() - self.label.pos()).x()  
 self.cutter\_ld\_y = (press.pos() - self.label.pos()).y()  
 elif self.left\_click\_num == 1:  
 self.cutter\_ru\_x = (press.pos() - self.label.pos()).x()  
 self.cutter\_ru\_y = (press.pos() - self.label.pos()).y()  
 self.bres\_int(self.cutter\_ld\_x, self.cutter\_ld\_y, self.cutter\_ld\_x, self.cutter\_ru\_y, self.colour\_cutter)  
 self.bres\_int(self.cutter\_ld\_x, self.cutter\_ld\_y, self.cutter\_ru\_x, self.cutter\_ld\_y, self.colour\_cutter)  
 self.bres\_int(self.cutter\_ld\_x, self.cutter\_ru\_y, self.cutter\_ru\_x, self.cutter\_ru\_y, self.colour\_cutter)  
 self.bres\_int(self.cutter\_ru\_x, self.cutter\_ld\_y, self.cutter\_ru\_x, self.cutter\_ru\_y, self.colour\_cutter)  
 self.left\_click\_num += 1  
 self.pm = self.pm.fromImage(self.img)  
 self.label.setPixmap(self.pm)  
  
 def keyPressEvent(self, press):  
 if int(press.key()) == QtCore.Qt.Key\_H:  
 self.horizontal = not self.horizontal  
 self.vertical = False  
 elif int(press.key()) == QtCore.Qt.Key\_V:  
 self.vertical = not self.vertical  
 self.horizontal = False  
  
 def proc(self):  
 self.img = self.pm.toImage()  
 for i in range(0, len(self.p\_x), 2):  
 self.makeCut(self.p\_x[i], self.p\_y[i], self.p\_x[i + 1], self.p\_y[i + 1])  
 self.pm = self.pm.fromImage(self.img)  
 self.label.setPixmap(self.pm)  
 self.label.repaint()  
  
 def makeCut(self, p1\_x, p1\_y, p2\_x, p2\_y):  
 t1 = [0, 0, 0, 0]  
 t2 = [0, 0, 0, 0]  
 p1res\_x = p1\_x  
 p1res\_y = p1\_y  
 p2res\_x = p2\_x  
 p2res\_y = p2\_y  
 sum1 = 0  
 sum2 = 0  
 mult = 0  
 flag = True  
 number = 0  
 m = inf  
 if p1\_x < self.cutter\_ld\_x: t1[3] = 1  
 if p1\_x > self.cutter\_ru\_x: t1[2] = 1  
 if p1\_y < self.cutter\_ld\_y: t1[1] = 1  
 if p1\_y > self.cutter\_ru\_y: t1[0] = 1  
 if p2\_x < self.cutter\_ld\_x: t2[3] = 1  
 if p2\_x > self.cutter\_ru\_x: t2[2] = 1  
 if p2\_y < self.cutter\_ld\_y: t2[1] = 1  
 if p2\_y > self.cutter\_ru\_y: t2[0] = 1  
 for i in range(0, 4):  
 sum1 += t1[i]  
 sum2 += t2[i]  
 if sum1 == 0 and sum2 == 0:  
 goto .q7  
 for i in range(0, 4):  
 mult += (t1[i] + t2[i]) // 2  
 if mult != 0:  
 flag = False  
 goto .q7  
 if sum1 == 0:  
 number = 1  
 p1res\_x = p1\_x; p1res\_y = p1\_y  
 p\_x = p2\_x; p\_y = p2\_y  
 goto .q2  
 if sum2 == 0:  
 number = 2  
 p1res\_x = p2\_x; p1res\_y = p2\_y  
 p\_x = p1\_x; p\_y = p1\_y  
 goto .q2  
 number = 0  
  
 label .q1  
 if number != 0:  
 if number == 1: p1res\_x = p\_x; p1res\_y = p\_y  
 else: p2res\_x = p\_x; p2res\_y = p\_y  
 number += 1  
 if number > 2:  
 goto .q7  
 if number == 1: p\_x = p1\_x; p\_y = p1\_y  
 else: p\_x = p2\_x; p\_y = p2\_y  
  
 label .q2  
 if p2\_x - p1\_x == 0:  
 goto .q4  
 m = (p2\_y - p1\_y) / (p2\_x - p1\_x)  
 if self.cutter\_ld\_x < p\_x:  
 goto .q3  
 y = m \* (self.cutter\_ld\_x - p\_x) + p\_y  
 if self.cutter\_ru\_y < y < self.cutter\_ld\_y:  
 goto .q3  
 p\_y = y; p\_x = self.cutter\_ld\_x  
 goto .q1  
  
 label .q3  
 if self.cutter\_ru\_x > p\_x:  
 goto .q4  
 y = m \* (self.cutter\_ru\_x - p\_x) + p\_y  
 if self.cutter\_ru\_y < y < self.cutter\_ld\_y:  
 goto .q4  
 p\_y = y; p\_x = self.cutter\_ru\_x  
 goto .q1  
  
 label .q4  
 if m == 0:  
 goto .q1  
 if self.cutter\_ru\_y > p\_y:  
 goto .q5  
 x = (1 / m) \* (self.cutter\_ru\_y - p\_y) + p\_x  
 if self.cutter\_ru\_x < x < self.cutter\_ld\_x:  
 goto .q5  
 p\_x = x; p\_y = self.cutter\_ru\_y  
 goto .q1  
  
 label .q5  
 if self.cutter\_ld\_y < p\_y:  
 return  
 x = (1 / m) \* (self.cutter\_ld\_y - p\_y) + p\_x  
 if self.cutter\_ru\_x < x < self.cutter\_ld\_x:  
 goto .q6  
 p\_x = x; p\_y = self.cutter\_ld\_y  
 goto .q1  
  
 label .q6  
 flag = False  
  
 label .q7  
 if not flag:  
 goto .q8  
 self.bres\_int(p1res\_x, p1res\_y, p2res\_x, p2res\_y, self.colour\_line\_in)  
  
 label .q8  
 return  
  
 def clean\_screen(self):  
 self.pm.fill(self.white)  
 self.img = self.pm.toImage()  
 self.label.setPixmap(self.pm)  
 self.label.repaint()  
 self.p\_x = []  
 self.p\_y = []  
 self.left\_click\_num = 0  
  
def sign(x):  
 if x < 0:  
 return -1  
 elif x > 0:  
 return 1  
 return 0  
  
def main():  
 app = QtWidgets.QApplication(sys.argv)  
 window = Visual()  
 window.show()  
 app.exec\_()  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()