Содержание

ВВЕДЕНИЕ

Компьютерная графика используется в современном мире повсеместно: реклама на ТВ, создание спецэффектов в кино, обработка фотографий, рендеринг 3d моделей.

Однако вся современная красота строго запрограммирована. Любая сложная задача требует решить сначала простую. Тривиальная задача должна быть решена с использованием минимальных ресурсов и наиболее быстро.

Одной из таких тривиальных задач в компьютерной графике считается задача отсечения отрезка прямоугольной областью. В данном отчёте описывает решение этой проблемы.

Сначала описывается сама задача, затем численный метод для её решения. Далее описываются структуры данных и алгоритм для написания программы.

Отчёт также содержит полный текст программы на языках C и Python, описание всех функций, инструкцию пользователю и тестовый пример. В написанной программе реализован алгоритм Коэна-Сазерленда.

1. ЗАДАЧА ОТСЕЧЕНИЯ ОТРЕЗКА ПРЯМОУГОЛЬНЫМ ОКНОМ

1.1. Содержательное описание задачи

Дана некая прямая линия, и её нужно отсечь прямоугольным окном. Данная задача может возникнуть, например, при работе с векторной графикой.

Так как вся работа с графикой на компьютере представляет собой работу с координатами, то суть задачи будет следующая: некий отрезок, заданный при помощи координат двух точек (начало отрезка и его конец) обрезать прямоугольной областью, так же заданной двумя точками (левый верхний угол и правый нижний угол). Решением данной задачи должно служить координаты двух точек обрезанной прямой.

1.2. Формальная постановка задачи

Задаётся отрезок, описанный координатами его концов. Так же задаётся прямоугольная область, описанная координатами двух углов: левого верхнего и правого нижнего. Требуется отсечь отрезок заданной прямоугольной областью, т.е. найти координаты начала и конца полученного отрезка.

Использовать декартовую систему координат, с координатами X и Y (т.е. плоскость).

Для решения данной задачи использовать алгоритм Коэна-Сазерленда, идея которого состоит в следующем.

Окно отсечения и прилегающие к нему части плоскости вместе образуют 9 областей. Каждой из областей присвоен 4-х битный код.

Две конечные точки отрезка получают 4-х битные коды, соответствующие областям, в которые они попали. Смысл битов кода:

0-ой бит = 1 – точка левее прямоугольного окна;

1-ый бит = 2 – точка правее прямоугольного окна;

2-ой бит = 4 – точка выше прямоугольного окна;

3-ий бит = 8 – точка ниже прямоугольного окна.

Определение того лежит ли отрезок целиком внутри окна или целиком вне окна выполняется следующим образом:

- если коды обоих концов отрезка равны 0, то отрезок целиком внутри окна, отсечение не нужно, отрезок принимается как тривиально видимый;

- если логическое "И"кодов обоих концов отрезка не равно нулю, то отрезок целиком

- вне окна, отсечение не нужно, отрезок отбрасывается как тривиально невидимый; если логическое "И"кодов обоих концов отрезка равно нулю, то отрезок подозрительный, он может быть частично видимым или целиком невидимым; для него нужно определить координаты пересечений со сторонами окна и для каждой полученной части определить тривиальную видимость или невидимость.

При расчете пересечения используется горизонтальность либо вертикальность сторон окна, что позволяет определить координату X или Y точки пересечения без вычислений.

При непосредственном использовании описанного выше способа отбора целиком видимого или целиком невидимого отрезка после расчета пересечения потребовалось бы вычисление кода расположения точки пересечения.

В целом схема алгоритма Сазерленда-Кохена следующая:

1) рассчитать коды конечных точек отсекаемого отрезка. В цикле повторять пункты 2-6;

2) если логическое "И"кодов конечных точек не равно 0, то отрезок целиком вне окна. Он отбрасывается и отсечение закончено;

3) если оба кода равны 0, то отрезок целиком видим. Он принимается и отсечение закончено;

4) если начальная точка внутри окна, то она меняется местами с конечной точкой;

5) анализируется код начальной точки для определения стороны окна с которой

есть пересечение и выполняется расчет пересечения. При этом вычисленная точка пересечения заменяет начальную точку;

6) определение нового кода начальной точки.

2. РАЗРАБОТКА АЛГОРИТМА

2.1. Разработка графического интерфейса пользователя

Для ввода координат отрезка необходимо 4 текстовых поля ввода: для задания координат начала и конца отрезка. Также для задания координат прямоугольника необходимо ещё 4 текстовых поля ввода: для задания координат двух углов: левого верхнего и правого нижнего. Вывод полученной информации осуществляется в отдельных полях-надписях.

Для наглядности нужно поместить панель, представляющей собой I квадрант декартовой системы координат для рисования исходного отрезка и прямоугольной области. Пользователь с помошью левой кнопки мыши сможет рисовать отрезок для отсечения зеленым цветом, а правой кнопкой мыши красным цветом рисуется граница области отсечения. Отсеченный отрезок рисуется красным цветом поверх исходного отрезка для большей наглядности.

В главном меню должны быть кнопки, выполняющие следующие действия: выход, выполнение отсечения, перерисовка области рисования, справка о программе.

Итак, внешний вид разработанного интерфейса представлен на рисунке 1.

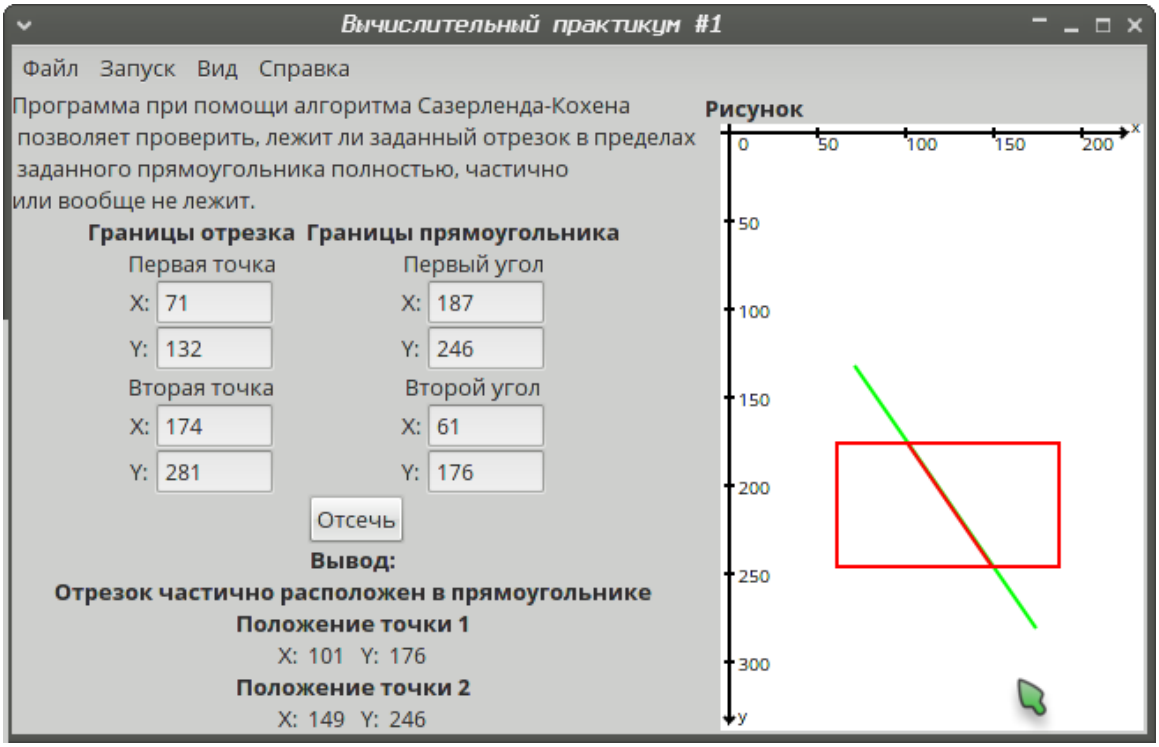

Рисунок 1 – Разработанный интерфейс программы

2.2. Разработка структур данных

Исходя из того, что работа осуществляется с отрезком и прямоугольным окном, логично ввести переменные:

POINT ll_edge,tr_edge – левый нижний и правый верхний углы прямоугольника; POINT b, a – точки, исходные координаты отрезка, после работы программы - координаты концов отсеченного отрезка;

Тип POINT имеет поля x и y - double - координаты точки.

Для битовых операций необходимо задать константы LEFT,TOP,BOT,RIGHT - равные различным степеням двойки.

2.3. Разработка структуры алгоритма

Для осуществления работы алгоритма можно выделить следующие подпрограммы:

1) подпрограмма получения кода относительного положения точки относительно прямоугольного окна, которой передаются углы прямоугольника и точка, код которой надо выяснить;

2) подпрограмма отсечения отрезка прямоугольным окном с помощью алгоритма Сазерленда-Кохена, которая использует подпрограмму 3 для получения кодов положения точек отрезка. Ей передаются начало отрезка, конец отрезка, углы прямоугольника. Программа заменяет переданные ей конец и начало исходного отрезка на координаты концов отсеченного отрезка, и возвращает 0, если он полностью лежит в прямоугольнике, 1 - если чатично, и 2 - если не лежит.

3) Подпрограмма ввода данных input_data

4) Подпрограмма вывода данных output_data

2.4. Схема алгоритма

На рисунке 2 представлена схема алгоритма получения кода относительного метоположения точки относительно прямоугольной области.

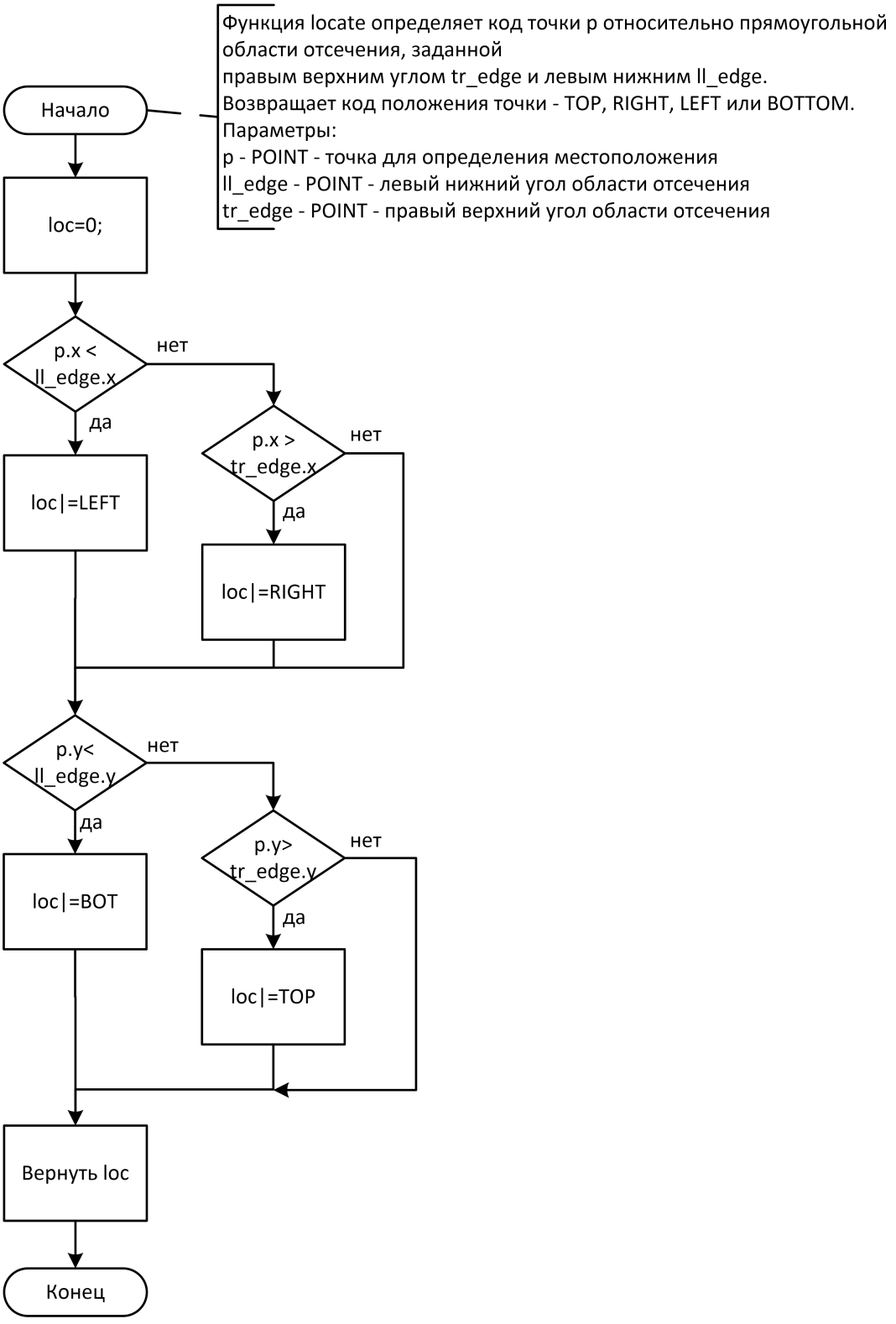

Рисунок 2 – Схема алгоритма получения кода местополодения

На рисунке 3 представлена схема алгоритма Коэна-Сазерленда - отсечения отрезка прямоугольной областью.

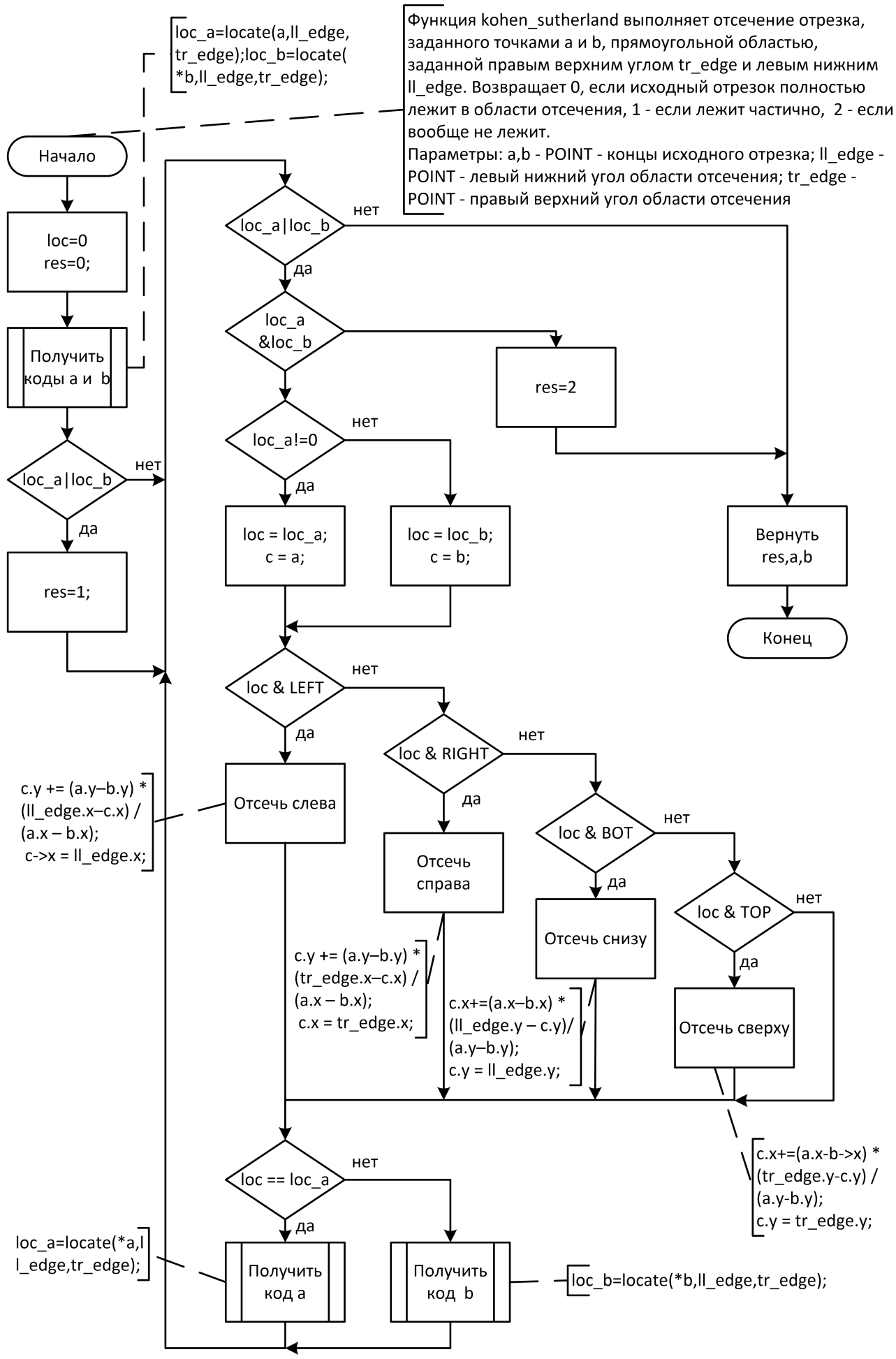

Рисунок 3 – Схема алгоритма Коэна-Сазерленда

3. РАЗРАБОТКА ПРОГРАММЫ

3.1. Описание переменных и структур данных

Исходя из того, что работа осуществляется с отрезком и прямоугольным окном, логично ввести переменные: POINT ll_edge,tr_edge – левый нижний и правый верхний углы прямоугольника; POINT b, a – точки, исходные координаты отрезка, после работы программы - координаты концов отсеченного отрезка;

Тип POINT имеет поля x и y - double - координаты точки.

Для битовых операций необходимо задать константы LEFT,TOP,BOT,RIGHT - равные различным степеням двойки.

3.2. Описание функций

1. int kohen_sutherland(POINT *a,POINT *b, POINT ll_edge, POINT tr_edge)

Функция kohen_sutherland выполняет отсечение отрезка, заданного точками a и b, прямоугольной областью, заданной правым верхним углом tr_edge и левым нижним ll_edge.

Возвращает 0, если исходный отрезок полностью лежит в области отсечения, 1 - если лежит частично, 2 - если вообще не лежит.

Функция модифицирует параметры a и b - в них возвращаются точки отсеченного отрезка.

Параметры функции представлены в таблице 1 :

Таблица 1 – Параметры функции отсечения отрезка

имя	тип	предназначение
a,b	POINT	концы исходного отрезка
ll_edge	POINT	левый нижний угол области отсечения
tr_edge	POINT	правый верхний угол области отсечения

2. int locate(POINT p,POINT ll_edge,POINT tr_edge)

Функция locate определяет код точки p относительно прямоугольной области отсечения, заданной правым верхним углом tr_edge и левым нижним ll_edge.

Возвращает код положения точки - TOP, RIGHT, LEFT или BOTTOM.

Параметры функции представлены в таблице 2 :

Таблица 2 – Параметры функции получения битового кода

имя	**тип**	**предназначение**
p	POINT	точка для определения местоположения
ll_edge	POINT	левый нижний угол области отсечения
tr_edge	POINT	правый верхний угол области отсечения

3. input_data(self):

Вводит исходные данные.

4. output_data(self,p1,p2,ll_edge,tr_edge)

Выводит полученные данные.

5. on_run_clicked(self,button,data=None):

Производит считывание данных, отсечение и вывод результатов.

6. update_draw(self,widget,data=None):

Обновляет рисунок отрезка и прямоугольника.

7. draw_axes(self,cr,widget):

Рисует координатные оси.

8. draw_clipped(self,cr,widget):

Рисует исходный и отсеченный отрезки и прямоугольник.

4. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Данная программа отсекает отрезок прямоугольной областью методом Коэна-Сазерленда. Исходный отрезок задаётся двумя точками: координатами начала и конца отрезка. Исходный прямоугольник задаётся координатами двух противоположных углов. Результат выводится в виде координат отсечённого отрезка.

Данная программа не требует установки. Для её запуска необходимо открыть файл prac1.py. Внимание: для работы приложения на компьютере должен быть установлен Python версии 3, GTK+3 , GObject-introspection. Для ввода начальных данных требуется заполнить все предназначенные для этого поля цифрами, после чего нажать enter или выбрать пункт меню Запуск->Отсечь. Кроме того, можно графически нарисовать отрезок с прямоугольником. Для рисования прямоугольника на белой панели зажмите правую кнопку мыши в месте, где будет находиться левый верхний угол прямоугольника, а затем переместите курсор на позицию правого нижнего угла. Чтобы прекратить рисование, отожмите левую кнопку. Для рисования отрезка проделайте те же операции, только используете левую кнопку мыши.

После обработки результата в нижнем текстовом окне появятся координаты начальной точки и конечной точки отсечённого отрезка. Если отрезок не входит в прямоугольную область, то об этом будет сообщено.

5. ТЕСТОВАЯ ЗАДАЧА

5.1. Аналитическое решение и умозрительные результаты

В тестовом примере возьмём прямую, заданную точками (10;10) и (70;70). Координаты точек прямоугольника следующие: (5;50) и (50;5). Очевидно, что данный отрезок обрезается правым верхним углом прямоугольника.

Следовательно, координаты полученного отрезка следующие: (10;10) и (50;50)

5.2. Решение, полученное с использованием разработанного ПО

Ниже на рисунке 4 представлен пример работы программы отсечения отрезка прямоугольным окном.

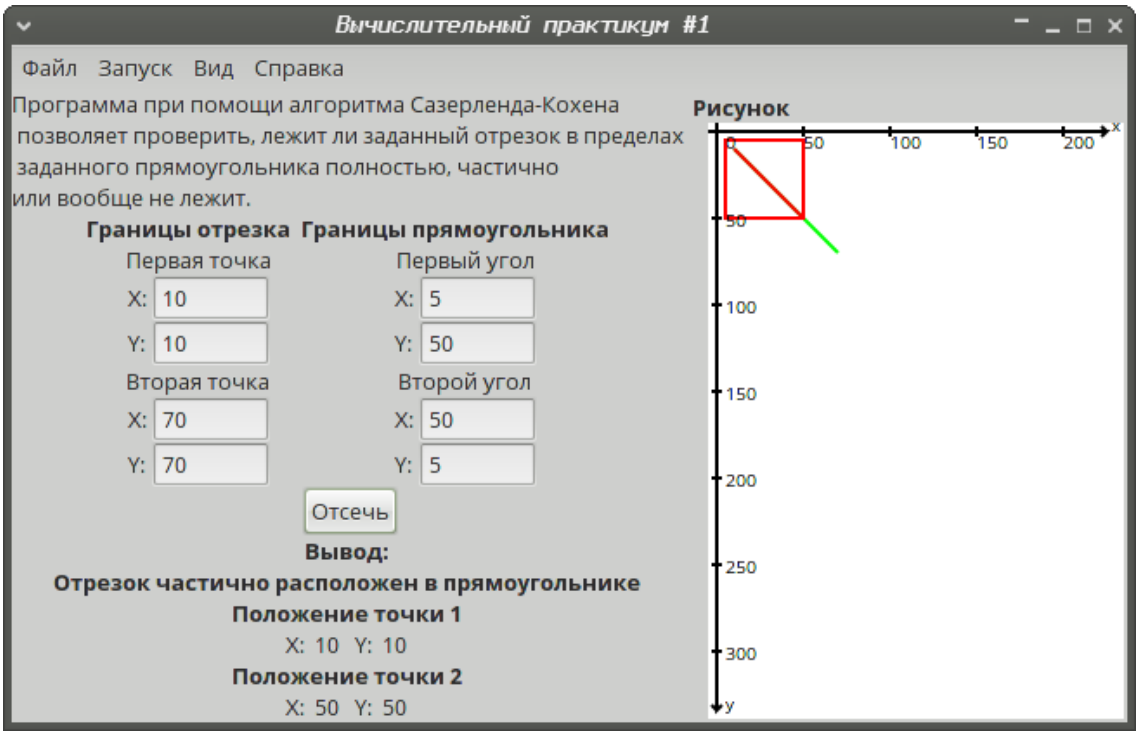

Рисунок 4 – Пример работы программы отсечения отрезка

5.3. Выводы

Данная программа реализует алгоритм Коэна-Сазерленда. По введённым координатам отрезка и прямоугольной области производит вычисления, и в результате выдаёт координаты отсечённого отрезка.

ЗАКЛЮЧЕНИЕ

Отсечения линии прямоугольником повместно используется в компьютерной графике. Продемонстрированная программа выполняет данную задачу одним из ме-

тодов, выводя результат в текстовой форме.

Хотя данная задача и тривиальна, однако для вычисления огромного количества подобных операций требуется использовать специализированные методы, которые работают быстрее, и задействую меньше ресурсов для вычислений.

Алгоритм Коэна-Сазерленда весьма эффективен и прост, что позволяет его использовать в мощных проектах по обработке векторной графики.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://python.org
2. http://www.gtk.org
3. http://ru.wikipedia.org
4. http://en,wikipedia.org

ПРИЛОЖЕНИЕ

Ниже приведен текст модуля расширения Python, реализующего метод Коэна-Сазерленда, и написанного на Си.

```
#include <Python.h>
#include "structmember.h"
typedef struct{
    float x,y;
} POINT;
#define LEFT 1
#define RIGHT 2
#define BOT 8
#define TOP 4
/*
Функция locate определяет код точки p относительно прямоугольной области отсечения,
заданной правым верхним углом tr_edge и левым нижним ll_edge.
Возвращает код положения точки — TOP, RIGHT, LEFT или BOTTOM.
Параметры:
p — POINT — точка для определения местоположения
ll_edge — POINT — левый нижний угол области отсечения
tr_edge — POINT — правый верхний угол области отсечения
```

```
*/
int locate(POINT p,POINT ll_edge,POINT tr_edge){
    int loc=0;
    if (p.x < ll_edge.x)
        loc|=LEFT;
    else if (p.x > tr_edge.x)
        loc|=RIGHT;
    if (p.y< ll_edge.y)
        loc|=BOT;
    else if (p.y>tr_edge.y)
        loc|=TOP;
    return loc;
}
/*
Функция kohen_sutherland выполняет отсечение отрезка, заданного точками a и b,
прямоугольной областью, заданной правым верхним углом tr_edge и
левым нижним ll_edge.
Возвращает 0, если исходный отрезок полностью лежит в области отсечения,
           1 — если лежит частично,
           2 — если вообще не лежит.
Функция модифицирует параметры a и b — в них возвращаются точки отсеченного отрезка.
Параметры:
a,b — POINT — концы исходного отрезка
ll_edge — POINT — левый нижний угол области отсечения
tr_edge — POINT — правый верхний угол области отсечения
*/
int kohen_sutherland(POINT *a,POINT *b,
                             POINT ll_edge,
                             POINT tr_edge){

    int loc=0,loc_a=locate(*a,ll_edge,tr_edge),loc_b=locate(*b,ll_edge,tr_edge);
    POINT *c;
    int res=0;
    if (loc_a|loc_b){
        res=1;
        while (loc_a | loc_b){
            if (loc_a & loc_b){
                res=2;
                break;
            }
            if (loc_a) {
                loc = loc_a;
                c = a;
            } else {
                loc = loc_b;
                c = b;
            }
            if (loc & LEFT) {
                c->y += (a->y - b->y) * (ll_edge.x - c->x) / (a->x - b->x);
                c->x = ll_edge.x;
            } else if (loc & RIGHT){
                c->y += (a->y - b->y) * (tr_edge.x - c->x) / (a->x - b->x);
                c->x = tr_edge.x;
            } else if (loc & BOT){
                c->x += (a->x - b->x) * (ll_edge.y - c->y) / (a->y - b->y);
                c->y = ll_edge.y;
            } else if (loc & TOP){
                c->x += (a->x - b->x) * (tr_edge.y - c->y) / (a->y - b->y);
                c->y = tr_edge.y;
```

```
            }
            if (loc == loc_a)
                loc_a = locate(*a,ll_edge,tr_edge);
            else
                loc_b = locate(*b,ll_edge,tr_edge);
        }
    }
    return res;
}

typedef struct {
    PyObject_HEAD
    double x;
    double y;
} Py_Point;

static void
Py_Point_dealloc(Py_Point* self)
{
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject *
Py_Point_new(PyTypeObject *type, PyObject *args, PyObject *kwds)
{
    Py_Point *self;

    self = (Py_Point *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->x = 0.0;
        self->y = 0.0;
    }

    return (PyObject *)self;
}

static int
Py_Point_init(Py_Point *self, PyObject *args, PyObject *kwds)
{
    if (self!=NULL){
        self->x=0;
        self->y=0;
    }
    if (! PyArg_ParseTuple(args, "|d|d", &self->x, &self->y))
        return -1;

    return 0;
}


static PyMemberDef Py_Point_Members[] = {
    {"x",T_DOUBLE,offsetof(Py_Point,x),0,""},
    {"y",T_DOUBLE,offsetof(Py_Point,y),0,""},
    {NULL}  /* Sentinel */
};

static PyTypeObject Py_Point_Type = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "alg1.Point",             /* tp_name */
```

```
    sizeof(Py_Point),              /* tp_basicsize */
    0,                            /* tp_itemsize */
    (destructor)Py_Point_dealloc, /* tp_dealloc */
    0,                            /* tp_print */
    0,                            /* tp_getattr */
    0,                            /* tp_setattr */
    0,                            /* tp_reserved */
    0,                            /* tp_repr */
    0,                            /* tp_as_number */
    0,                            /* tp_as_sequence */
    0,                            /* tp_as_mapping */
    0,                            /* tp_hash  */
    0,                            /* tp_call */
    0,                            /* tp_str */
    0,                            /* tp_getattro */
    0,                            /* tp_setattro */
    0,                            /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT |
        Py_TPFLAGS_BASETYPE,      /* tp_flags */
    "",           /* tp_doc */
    0,                            /* tp_traverse */
    0,                            /* tp_clear */
    0,                            /* tp_richcompare */
    0,                            /* tp_weaklistoffset */
    0,                            /* tp_iter */
    0,                            /* tp_iternext */
    0,             /* tp_methods */
    Py_Point_Members,             /* tp_members */
    0,                            /* tp_getset */
    0,                            /* tp_base */
    0,                            /* tp_dict */
    0,                            /* tp_descr_get */
    0,                            /* tp_descr_set */
    0,                            /* tp_dictoffset */
    (initproc)Py_Point_init,      /* tp_init */
    0,                            /* tp_alloc */
    Py_Point_new,                 /* tp_new */
};

static PyObject *
kohen_sutherland_wrapper(PyObject *self, PyObject *args){
    #define conv(p,py_p) (p).x=(py_p)->x;(p).y=(py_p)->y
    POINT a,b,ll_edge,tr_edge;
    Py_Point *py_a,*py_b,*py_tr_edge,*py_ll_edge;
    if (!PyArg_ParseTuple(args, "OOOO", &py_a,&py_b,
                                        &py_ll_edge,
                                        &py_tr_edge))
        return NULL;
    conv(a,py_a); conv(b,py_b);conv(ll_edge,py_ll_edge);conv(tr_edge,py_tr_edge);
    int res=kohen_sutherland(&a,&b,ll_edge,tr_edge);
    py_a->x=a.x;py_b->x=b.x;
    py_a->y=a.y;py_b->y=b.y;
    return PyLong_FromLong(res);
}



static PyMethodDef Alg1Methods[] = {
    {"kohen_sutherland",  kohen_sutherland_wrapper, METH_VARARGS,""},
```

```
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
static struct PyModuleDef alg1module = {
   PyModuleDef_HEAD_INIT,
   "alg1",   /* name of module */
   NULL, /* module documentation, may be NULL */
   -1,       /* size of per-interpreter state of the module,
                or -1 if the module keeps state in global variables. */
   Alg1Methods
};

PyMODINIT_FUNC
PyInit_alg1(void)
{
    PyObject* m;

    if (PyType_Ready(&Py_Point_Type) < 0)
        return NULL;

    m = PyModule_Create(&alg1module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&Py_Point_Type);
    PyModule_AddObject(m, "Point", (PyObject *)&Py_Point_Type);
    return m;
}
```

Далее приводится текст основной программы, написанной на Python 3.

```
#!/usr/bin/env python
from copy import copy
import cairo
from gi.repository import Gtk,Gdk
from alg1 import Point, kohen_sutherland

def view_binary(decimal):
    bin_str=""
    while decimal:
        if decimal&1:
            bin_str='1'+bin_str
        else:
            bin_str='0'+bin_str
        decimal>>=1
    while len(bin_str)!=4:
        bin_str='0'+bin_str
    return bin_str

class Application(Gtk.Builder):

    def __init__(self,ui_filename):
        self.clipped=False
        Gtk.Builder.__init__(self)
        self.add_from_file(ui_filename)
        self.connect_signals(self)
        self.margin=5
    def show_msg(self,msg):
        md=Gtk.MessageDialog(None, Gtk.DialogFlags.MODAL,
                             Gtk.MessageType.WARNING, Gtk.ButtonsType.OK, msg);
```

```
        md.run ();
        md.destroy();
    def show(self,form_name):
        window = self.get_object(form_name)
        window.show()
        Gtk.main()
    def on_window_destroy( self,widget, data=None):
        self.get_object("window1").hide()
        Gtk.main_quit()
    def get_value(self,name):
        return int(float(self.get_object(name).get_text()))
    def input_data(self):
        try:
            p1=Point(self.get_value("entry1"),
                    self.get_value("entry2"))
            p2=Point(self.get_value("entry3"),
                    self.get_value("entry4"))
            ll_edge=Point(self.get_value("entry5"),
                    self.get_value("entry6"))
            tr_edge=Point(self.get_value("entry7"),
                    self.get_value("entry8"))
        except Exception as e:
            self.show_msg("Неправильное значение в поле ввода!\n"+str(e))
            return
        ll_edge.x,ll_edge.y,tr_edge.x,tr_edge.y=(
            min(ll_edge.x,tr_edge.x),
            min(ll_edge.y,tr_edge.y),
            max(ll_edge.x,tr_edge.x),
            max(ll_edge.y,tr_edge.y))
        if (ll_edge.x==tr_edge.x  or ll_edge.y==tr_edge.y):
            self.show_msg("Координаты X или Y углов прямоугольника равны!")
            raise
        return p1,p2,ll_edge,tr_edge
    def output_data(self,p1,p2,ll_edge,tr_edge):
        p1x_info=self.get_object("label20");p1y_info=self.get_object("label24");
        p2x_info=self.get_object("label26");p2y_info=self.get_object("label28");
        res_info=self.get_object("label21");
        if not res:
            res_info.set_text("Отрезок расположен в прямоугольнике")
            p1x_info.set_text(str(int(p1.x)))
            p1y_info.set_text(str(int(p1.y)))
            p2x_info.set_text(str(int(p2.x)))
            p2y_info.set_text(str(int(p2.y)))
        elif res==2:
            res_info.set_text("Отрезок не расположен в прямоугольнике")
            p1x_info.set_text("Не существует")
            p1y_info.set_text("Не существует")
            p2x_info.set_text("Не существует")
            p2y_info.set_text("Не существует")
        elif res==1:
            res_info.set_text("Отрезок частично расположен в прямоугольнике")
            p1x_info.set_text(str(int(p1.x)))
            p1y_info.set_text(str(int(p1.y)))
            p2x_info.set_text(str(int(p2.x)))
            p2y_info.set_text(str(int(p2.y)))
        else:
            res_info.set_text("Извините, маленькие технические неполадки")
        self.clipped=True
        self.get_object('drawingarea1').queue_draw()
```

```
    def on_run_clicked(self,button,data=None):
        p1,p2,ll_edge,tr_edge=self.input_data()
        res=kohen_sutherland(p1,p2,ll_edge,tr_edge)
        self.output_data(p1,p2,ll_edge,tr_edge)
    def on_mouse_press(self,widget,event,data=None):
        '''if self.clipped:
            cr=widget.get_window().cairo_create()
            cr.set_source_rgb(1.0,1.0,1.0)
            cr.paint()
            self.clipped=False'''
        if event.button==1:
            self.get_object("entry1").set_text(str(int(event.x)-self.margin))
            self.get_object("entry2").set_text(str(int(event.y)-self.margin))

        elif event.button==3:
            self.get_object("entry5").set_text(str(int(event.x)-self.margin))
            self.get_object("entry6").set_text(str(int(event.y)-self.margin))

    def on_mouse_release(*args):
        pass
    def update_draw(self,widget,data=None):
        drawarea=self.get_object("drawingarea1")
        cr=drawarea.get_window().cairo_create()
        if self.clipped:
            self.on_run_clicked(None)
        else:
            self.draw_clipped(cr,drawarea)

        self.draw_clipped(cr,widget)
    def coord_changed(self,widget,data=None):
        drawarea=self.get_object("drawingarea1")
        cr=drawarea.get_window().cairo_create()
        if self.clipped:
            self.on_run_clicked(None)
        else:
            self.draw_clipped(cr,drawarea)
    def on_move(self,widget,event,pointer=None):
        cr=widget.get_window().cairo_create()
        draw_serie=False;draw_rect=False;
        if event.state&(event.state.BUTTON1_MASK|event.state.BUTTON3_MASK):
            if event.state&event.state.BUTTON1_MASK:
                p1=Point(int(float(self.get_object("entry1").get_text())),
                        int(float(self.get_object("entry2").get_text())))
                p2=Point(int(event.x)-self.margin,int(event.y)-self.margin)
                self.get_object("entry3").set_text(str(int(event.x)-self.margin))
                self.get_object("entry4").set_text(str(int(event.y)-self.margin))
                draw_serie=True
                try:
                    ll_edge=Point(int(float(self.get_object("entry5").get_text())),
                            int(float(self.get_object("entry6").get_text())))
                    tr_edge=Point(int(float(self.get_object("entry7").get_text())),
                            int(float(self.get_object("entry8").get_text())))
                except:
                    pass
                else:
                    draw_rect=True
            elif event.state&event.state.BUTTON3_MASK:
                ll_edge=Point(int(float(self.get_object("entry5").get_text())),
                        int(float(self.get_object("entry6").get_text())))
```

```
                tr_edge=Point(int(event.x)-self.margin,int(event.y)-self.margin)
                self.get_object("entry7").set_text(str(int(event.x)-self.margin))
                self.get_object("entry8").set_text(str(int(event.y)-self.margin))
                draw_rect=True
                try:
                    p1=Point(int(float(self.get_object("entry1").get_text())),
                            int(float(self.get_object("entry2").get_text())))
                    p2=Point(int(float(self.get_object("entry3").get_text())),
                            int(float(self.get_object("entry4").get_text())))
                except:
                    pass
                else:
                    draw_serie=True

            cr.push_group()
            cr.set_source_rgb(1.0,1.0,1.0)
            cr.paint()
            if draw_serie:
                cr.set_source_rgb(0.0,1.0,0.0)
                cr.set_line_width(2)
                cr.move_to(p1.x+self.margin,p1.y+self.margin)
                cr.line_to(p2.x+self.margin,p2.y+self.margin)
                cr.close_path()
                cr.stroke()
            if draw_rect:
                cr.set_source_rgb(1.0,0.0,0.0)
                cr.rectangle(ll_edge.x+self.margin,ll_edge.y+self.margin,
                             tr_edge.x-ll_edge.x,tr_edge.y-ll_edge.y)
                cr.stroke()
            self.draw_axes(cr,widget)
            cr.pop_group_to_source()
            cr.paint()
    def draw_clipped(self,cr,widget):
        if 1:#try:
            draw_serie=False;draw_rect=False;
            try:
                p1=Point(float(self.get_object("entry1").get_text()),
                        float(self.get_object("entry2").get_text()))
                p2=Point(float(self.get_object("entry3").get_text()),
                        float(self.get_object("entry4").get_text()))
            except:
                pass
            else:
                draw_serie=True
            try:
                ll_edge=Point(float(self.get_object("entry5").get_text()),
                        float(self.get_object("entry6").get_text()))
                tr_edge=Point(float(self.get_object("entry7").get_text()),
                        float(self.get_object("entry8").get_text()))
            except:
                pass
            else:
                draw_rect=True
            cr.set_source_rgb(1.0,1.0,1.0)
            cr.paint()
            if draw_serie:
                cr.set_source_rgb(0.0,1.0,0.0)
                cr.set_line_width(2)
                cr.move_to(p1.x+self.margin,p1.y+self.margin)
```

```
            cr.line_to(p2.x+self.margin,p2.y+self.margin)
            cr.close_path()
            cr.stroke()
        if draw_rect:
            ll_edge.x,ll_edge.y,tr_edge.x,tr_edge.y=(
                min(ll_edge.x,tr_edge.x),
                min(ll_edge.y,tr_edge.y),
                max(ll_edge.x,tr_edge.x),
                max(ll_edge.y,tr_edge.y))
            cr.set_source_rgb(1.0,0.0,0.0)
            cr.rectangle(ll_edge.x+self.margin,ll_edge.y+self.margin,
                         tr_edge.x-ll_edge.x,tr_edge.y-ll_edge.y)
            cr.stroke()
        if self.clipped:
            p1=Point(float(self.get_object("label20").get_text()),
                     float(self.get_object("label24").get_text()))
            p2=Point(float(self.get_object("label26").get_text()),
                     float(self.get_object("label28").get_text()))
            cr.set_source_rgb(1.0,0.0,0.0)
            cr.set_line_width(2)
            cr.move_to(p1.x+self.margin,p1.y+self.margin)
            cr.line_to(p2.x+self.margin,p2.y+self.margin)
            cr.close_path()
            cr.stroke()
        self.draw_axes(cr,widget)
        #cr.paint()
    #except:
    #    pass
def draw_axes(self,cr,widget):
    zoom=1;min_x,min_y=0,0
    w=widget.get_allocated_width()
    h=widget.get_allocated_height()
    cr.set_source_rgb(0.0,0.0,0.0)

    cr.move_to(5,0)
    cr.line_to(5,h-5);
    cr.line_to(8,h-8);
    cr.move_to(2,h-8)
    cr.line_to(5,h-5)
    cr.move_to(10,h-5)
    cr.show_text("y")
    cr.stroke()

    cr.move_to(0,5)
    cr.line_to(w-10,5)
    cr.line_to(w-13,8)
    cr.move_to(w-13,2)
    cr.line_to(w-10,5)
    cr.move_to(w-7,5)
    cr.show_text("x")
    cr.stroke()
    cr.move_to(10,15)
    cr.show_text('0')
    x=self.margin+50
    while x<w-self.margin:
        cr.move_to(x,2);
        cr.line_to(x,8);
        cr.move_to(x,15);
        cr.show_text(str(int((x-self.margin)/zoom+min_x)));
```

```
            x+=50
        cr.stroke();
        y=self.margin+50;
        while y<h-self.margin:
            cr.move_to(2,y);
            cr.line_to(8,y);
            cr.move_to(10,y+5);
            cr.show_text(str(int((y-self.margin)/zoom+min_y)));
            y+=50
        cr.stroke()
    def draw(self,widget,cr,data=None):
        self.draw_clipped(cr,widget)
app=Application("prac1.ui")
app.show("window1")
```