**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра систем автоматизированного проектирования**

**ОТЧЁТ**

**по лабораторной работе №4  
по дисциплине «Компьютерная графика»**

**Тема: Исследование алгоритмов отсечения отрезков и многоугольников**

**окнами различного вида**

|  |  |
| --- | --- |
| Студенты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Матвеева И.В. |

Санкт-Петербург

2024

**Задание на лабораторную работу**

Вариант 3: Обеспечить реализацию алгоритма отсечения массива произвольных отрезков заданным прямоугольным окном с использование четырех битового кода для концов каждого анализируемого отрезка. Вначале следует вывести на экран сгенерированные отрезки полностью, а затем другим цветом или яркостью те, которые полностью или частично попадают в область окна.

**Теоретические положения**

Алгоритм Сазерленда-Коэна использует четырехбитный код для определения положения концов отрезка относительно прямоугольного окна. Каждый бит кода (b0, b1, b2, b3) указывает на расположение соответствующей точки относительно границ окна. При расчете кодов выполняются следующие правила: b0 устанавливается в 0, если x≥xmin; b0 в 1, если x<xmin; b1 в 0, если x≤xmax; 1, x>xmax; b2 в 0, если y≥ymin; b2 в 1, если y<ymin; b3 в 0, если y≤ymax; 1, y>ymax.

После вычисления кодов возможны следующие сценарии: 1) если коды состоят только из нулей, то отрезок полностью находится внутри окна и должен быть отрисован целиком; 2) если коды имеют единичный бит в одной и той же позиции, то отрезок находится за пределами окна и не будет отображен; 3) в остальных случаях окно пересекает отрезок частично, что требует осуществления отсечения.

Для случая 3 происходит итеративное перемещение концов отрезков вдоль самого отрезка с последующим пересчетом кодов. Этот процесс повторяется до тех пор, пока не выполнятся условия 1 или 2 — отрезок либо целиком видим, либо полностью невидим. Этот подход позволяет эффективно определять видимость отрезков относительно окна и осуществлять их корректное отображение.

**Ход работы**

Функция отсечения clipping, которая принимает массив отрезков lines и прямоугольное окно window.

Функции get\_bytes\_point, которая вычисляет и код видимости для заданной точки относительно окна. Код формируется на основе положения точки относительно границ окна. Создание списка points для хранения кодов видимости концов каждого отрезка в соответствии с окном.

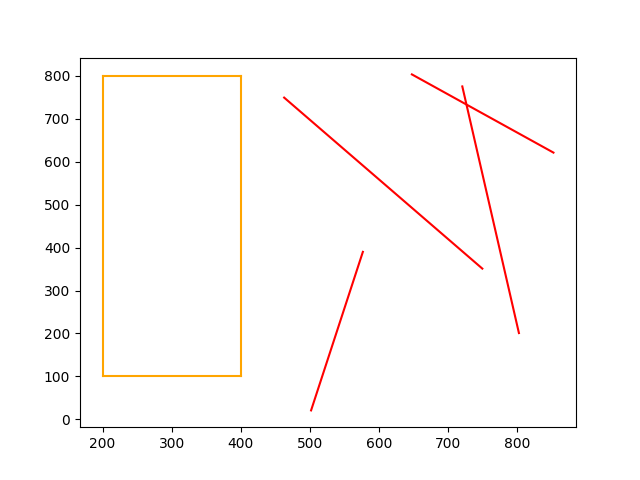
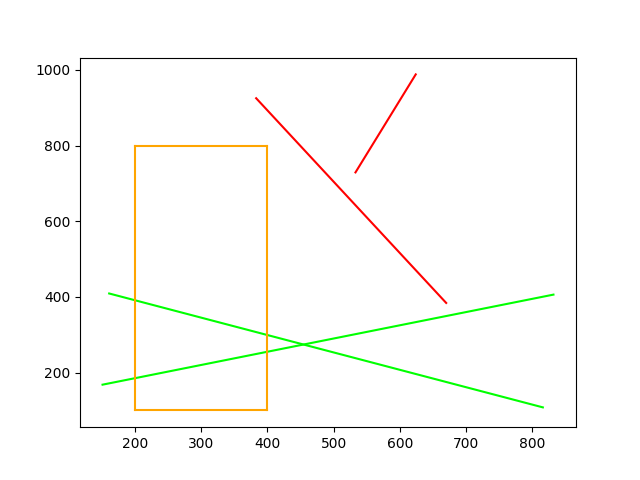
Функция get\_bytes\_point обходит все отрезки и вычисляет коды видимости для их концов.

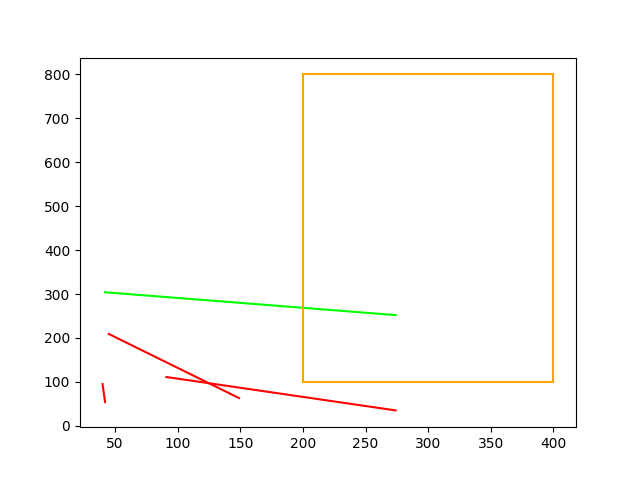
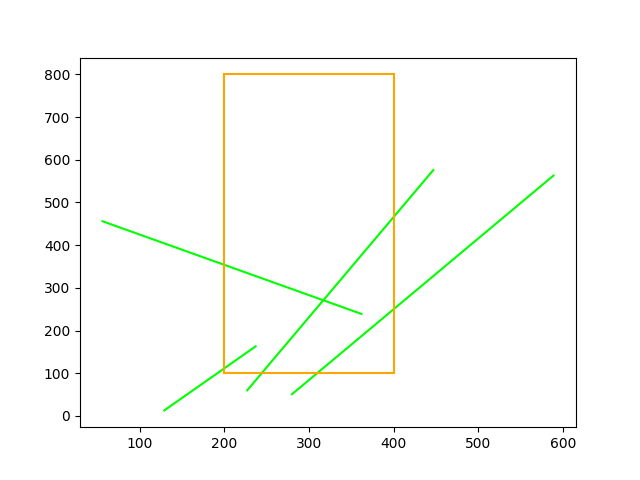
Создание пустого списка mas\_in для хранения результатов видимости отрезков. Проверка каждой пары кодов видимости для отрезков. Если хотя бы один код равен 0, то отрезок полностью видим и добавляется 1 в mas\_in. Если оба кода одинаковы, то отрезок полностью невидим и добавляется 0 в mas\_in. В остальных случаях происходит частичная видимость отрезков, и начинается итерация по каждому пикселю отрезка. Для каждого пикселя рассчитывается код видимости, и если хотя бы один пиксель видим, то отрезок считается частично видимым и добавляется результат 2 в mas\_in. Итоговый список mas\_in содержит результаты видимости для каждого отрезка относительно окна. Функция возвращает список mas\_in с результатами видимости отрезков.

**Результаты лабораторной работы**

Программа была создана на языке Python с использованием графической библиотеки Tkinter.

Для работы программы нужно ввести координаты окна и нажать кнопку “Ок”.

****



**Вывод**

В ходе лабораторной работы обеспечили реализацию алгоритма отсечения массива произвольных отрезков заданным прямоугольным окном с использование алгоритма Коэна-Сазерленда. Вывели на экран сгенерированные отрезки, зеленым цветом пометили те, что полностью или частично попадают в область окна.

**Приложение 1: исходный код программы**

import numpy as np

from mpl\_toolkits.mplot3d import Axes3D

from mpl\_toolkits.mplot3d.art3d import Poly3DCollection, Line3DCollection

import matplotlib.pyplot as plt

import matplotlib.lines

import matplotlib.patches

from tkinter import \*

from tkinter.ttk import \*

import random

def clipping(lines, window):

def get\_bytes\_point(point, window):

x, y = point

x\_min, x\_max = sorted([window[0][0], window[1][0]])

y\_min, y\_max = sorted([window[0][1], window[1][1]])

if x\_min <= x <= x\_max and y\_min <= y <= y\_max:

return 0

elif x < x\_min:

return 1 if y\_min <= y <= y\_max else 2

elif x > x\_max:

return 4 if y\_min <= y <= y\_max else 5

else:

return 6 if y < y\_min else 8 if y > y\_max else 10

points = []

for i in range(4):

for j in range(2):

points.append(get\_bytes\_point(lines[i][j], window))

mas\_in = []

for i in range(0, len(points), 2):

if 0 in points[i:i+2]:

mas\_in.append(1)

elif points[i] == points[i+1]:

mas\_in.append(0)

else:

result = 2

line = lines[(i // 2)]

for k in range(min(line[0][0], line[1][0]), max(line[0][0], line[1][0]), 1):

y = (k - line[0][0]) / (line[1][0] - line[0][0]) \* (line[1][1] - line[0][1]) + line[0][1]

point0 = get\_bytes\_point((k, y), window)

point1 = get\_bytes\_point((k, y), window)

if 0 in [point0, point1]:

result = 1

break

elif point0 != point1:

result = 2

mas\_in.append(result)

return mas\_in

def draw\_lines(lines, window, mas\_true\_points):

colors = ['red', 'lime']

for i in range(4):

color = colors[0]

if mas\_true\_points[i] == 1:

color = colors[1]

plt.plot([lines[i][0][0], lines[i][1][0]], [lines[i][0][1], lines[i][1][1]], color=color)

plt.plot([window[0][0], window[1][0]], [window[0][1], window[0][1]], color='orange')

plt.plot([window[0][0], window[0][0]], [window[0][1], window[1][1]], color='orange')

plt.plot([window[0][0], window[1][0]], [window[1][1], window[1][1]], color='orange')

plt.plot([window[1][0], window[1][0]], [window[0][1], window[1][1]], color='orange')

plt.show()

def generate\_lines():

coordinates = np.array([[[0, 0], [0, 0]], [[0, 0], [0, 0]], [[0, 0], [0, 0]], [[0, 0], [0, 0]]])

N = random.randint(5, 1000)

for i in range(4):

coordinates[i][0][0], coordinates[i][0][1], coordinates[i][1][0], coordinates[i][1][1] = random.randint(0, N),\

random.randint(0, N), random.randint(0, N), random.randint(0, N)

return coordinates

def main\_window():

res\_points = generate\_lines()

window\_points = np.array([[0, 0], [0, 0]])

def counts():

point = [0.0, 0.0]

window\_points[0][0] = float(x1\_field.get())

window\_points[0][1] = float(y1\_field.get())

res\_points[0] = tuple(point.copy())

window\_points[1][0] = float(x2\_field.get())

window\_points[1][1] = float(y2\_field.get())

res\_points[1] = tuple(point.copy())

def click():

counts()

res\_points = generate\_lines()

mas\_true\_points = clipping(res\_points, window\_points)

draw\_lines(res\_points, window\_points, mas\_true\_points)

window = Tk()

window.title("Лаб. работа №4")

window.geometry('300x200')

window.resizable(False, False)

Label(window, text="Введите координаты окна", font=("Consolas", 14)).place(x=35, y=10)

Label(window, text="x1: ", font=("Consolas", 14)).place(x=15, y=50)

x1\_field = Entry(window, width=10)

x1\_field.place(x=50, y=50)

Label(window, text="y1: ", font=("Consolas", 14)).place(x=155, y=50)

y1\_field = Entry(window, width=10)

y1\_field.place(x=190, y=50)

Label(window, text="x2: ", font=("Consolas", 14)).place(x=15, y=90)

x2\_field = Entry(window, width=10)

x2\_field.place(x=50, y=90)

Label(window, text="y2: ", font=("Consolas", 14)).place(x=155, y=90)

y2\_field = Entry(window, width=10)

y2\_field.place(x=190, y=90)

ok\_button = Button(window, text="Ok", command=click)

ok\_button.place(x=108, y=150)

window.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

main\_window()