Міністерство освіти і науки України
Національний технічний університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»

Факультет прикладної математики
Кафедра «Системного програмування і спеціалізованих комп'ютерних систем»

Лабораторна робота №3
З дисципліни «Комп'ютерна графіка» :
«Алгоритми заливки»

Виконав:
студент III курсу,
група КВ-41
Яковенко Максим

Перевірив:
_____

Київ-2016

Постановка задачі:
Програмно реалізувати алгоритми заливки: простий ітеративний, рекурсивний, з використанням списку реберних точок.

Код програми:

```
from tkinter import *
import time


PIXEL_SIZE = 5
HEIGHT = 700
WIDTH = 1000

REC_MAX_X = 32
REC_MAX_Y = 32

class RasterizationAlgorithms:
    recursive = {
        "border": [[(0, 0), (REC_MAX_X, 0)], [(REC_MAX_X, 0), (REC_MAX_X, REC_MAX_Y)],
[(REC_MAX_X, REC_MAX_Y),
                (0, REC_MAX_Y)], [(0, REC_MAX_Y), (0, 0)]]
    }

    iteration = {
        "border": [[(50, 0), (100, 0)], [(100, 0), (100, 30)], [(100, 30), (150, 30)], [(150,
30), (150, 80)],
                   [(150, 80), (100, 80)], [(100, 80), (100, 110)], [(100, 110), (50, 110)],
[(50, 110), (50, 80)],
                   [(50, 80), (0, 80)], [(0, 80), (0, 30)], [(0, 30), (50, 30)], [(50, 30),
(50, 0)]]
    }

    line = {
        "border":[[(10, 5), (50, 5)], [(50, 5), (50, 50)], [(50, 50), (45, 50)], [(45, 50),
(45, 10)],
                   [(45, 10), (15, 10)], [(15, 10), (15, 50)], [(15, 50), (10, 50)], [(10,
50), (10, 5)]]
    }

    def recursiveFill(self, x, y):
        self.points.append((x, y))
        if (x < REC_MAX_X and x > 0 and y - 1 < REC_MAX_Y and y - 1 > 0):
            if (((x, y - 1) in self.points) == 0) and (((x, y - 1) in self.stack) == 0):
                self.stack.append((x, y - 1))
        if (x + 1 < REC_MAX_X and x + 1 > 0 and y - 1 < REC_MAX_Y and y - 1 > 0):
            if (((x + 1, y - 1) in self.points) == 0) and (((x + 1, y - 1) in self.stack) ==
0):
                self.stack.append((x + 1, y - 1))
        if (x + 1 < REC_MAX_X and x + 1 > 0 and y < REC_MAX_Y and y > 0):
            if (((x + 1, y) in self.points) == 0) and (((x + 1, y) in self.stack) == 0):
                self.stack.append((x + 1, y))
        if (x + 1 < REC_MAX_X and x + 1 > 0 and y + 1 < REC_MAX_Y and y + 1 > 0):
            if (((x + 1, y + 1) in self.points) == 0) and (((x + 1, y + 1) in self.stack) ==
0):
                self.stack.append((x + 1, y + 1))
        if (x < REC_MAX_X and x > 0 and y + 1 < REC_MAX_Y and y + 1 > 0):
            if (((x, y + 1) in self.points) == 0) and (((x, y + 1) in self.stack) == 0):
                self.stack.append((x, y + 1))
        if (x - 1 < REC_MAX_X and x - 1 > 0 and y + 1 < REC_MAX_Y and y + 1 > 0):
            if (((x - 1, y + 1) in self.points) == 0) and (((x - 1, y + 1) in self.stack) ==
0):
                self.stack.append((x - 1, y + 1))
        if (x - 1 < REC_MAX_X and x - 1 > 0 and y < REC_MAX_Y and y > 0):
            if (((x - 1, y) in self.points) == 0) and (((x - 1, y) in self.stack) == 0):
                self.stack.append((x - 1, y))
        if (x - 1 < REC_MAX_X and x - 1 > 0 and y - 1 < REC_MAX_Y and y - 1 > 0):
            if (((x - 1, y - 1) in self.points) == 0) and (((x - 1, y - 1) in self.stack) ==
```

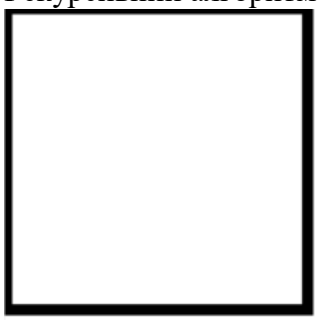
```
0):
                self.stack.append((x - 1, y - 1))
        if (self.stack):
            x, y = self.stack.pop()
            self.recursiveFill(x, y)
        else:
            self.draw(self.points)

    def iterationFill(self, x, y):
        stack = [(x, y)]
        while (stack):
            x, y = stack.pop()
            self.points.append((x, y))

            if (x + 1 < 100 and x + 1 > 50 and y < 110 and y > 0):
                if (((x + 1, y) in self.points) == 0) and (((x + 1, y) in stack) == 0):
                    stack.append((x + 1, y))
            if (x < 100 and x > 50 and y - 1 < 110 and y - 1 > 0):
                if (((x, y - 1) in self.points) == 0) and (((x, y - 1) in stack) == 0):
                    stack.append((x, y - 1))
            if (x - 1 < 100 and x - 1 > 50 and y < 110 and y > 0):
                if (((x - 1, y) in self.points) == 0) and (((x - 1, y) in stack) == 0):
                    stack.append((x - 1, y))
            if (x < 100 and x > 50 and y + 1 < 110 and y + 1 > 0):
                if (((x, y + 1) in self.points) == 0) and (((x, y + 1) in stack) == 0):
                    stack.append((x, y + 1))
            if (x - 1 > 0 and x - 1 < 150 and y < 80 and y > 30):
                if (((x - 1, y) in self.points) == 0) and (((x - 1, y) in stack) == 0):
                    stack.append((x - 1, y))
            if (x + 1 < 150 and x + 1 > 0 and y < 80 and y > 30):
                if (((x + 1, y) in self.points) == 0) and (((x + 1, y) in stack) == 0):
                    stack.append((x + 1, y))
        self.draw(self.points)

    def lineFill(self, x, y):
        stack = [(x, y)]
        while (stack):

            x, y = stack.pop()
            if (y >= 5 and y <= 10):
                newX = x
                while (newX >= 10):
                    self.points.append((newX, y))
                    newX -= 1
                newX = x
                while (newX <= 50):
                    self.points.append((newX, y))
                    newX += 1

            if ((((50, y + 1) in self.points) == 0) and (y >= 5 and y <= 10)):
                stack.append((50, y + 1))
                stack.append((15, y + 1))
            if ((((50, y - 1) in self.points) == 0) and (y >= 5 and y <= 10)):
                stack.append((50, y - 1))

            elif (y > 10 and y <= 50):
                if (x >= 10 and x <= 15):
                    newX = x
                    while (newX >= 10):
                        self.points.append((newX, y))
                        newX -= 1
                    newX = x
                    while (newX <= 15):
                        self.points.append((newX, y))
                        newX += 1

                    if ((((15, y + 1) in self.points) == 0) and (y > 10 and y <= 50)):
                        stack.append((15, y + 1))
                    if ((((15, y - 1) in self.points) == 0) and (y > 10 and y <= 50)):
                        stack.append((15, y - 1))
```

```
                if (x >= 45 and x <= 50):
                    newX = x
                    while (newX >= 45):
                        self.points.append((newX, y))
                        newX -= 1
                    newX = x
                    while (newX <= 50):
                        self.points.append((newX, y))
                        newX += 1

                    if ((((50, y + 1) in self.points) == 0) and (y > 10 and y <= 50)):
                        stack.append((50, y + 1))
                    if ((((50, y - 1) in self.points) == 0) and (y > 10 and y <= 50)):
                        stack.append((50, y - 1))

        self.draw(self.points)

    def Bresenham(self, x1, y1, x2, y2):
        dx = x2 - x1
        dy = y2 - y1
        is_steep = abs(dy) > abs(dx)
        if is_steep:
            x1, y1 = y1, x1
            x2, y2 = y2, x2
        if x1 > x2:
            x1, x2 = x2, x1
            y1, y2 = y2, y1
        dx = x2 - x1
        dy = y2 - y1
        error = int(dx / 2.0)
        ystep = 1 if y1 < y2 else -1
        y = y1
        points = []
        for x in range(x1, x2 + 1):
            coord = (y, x) if is_steep else (x, y)
            points.append(coord)
            error -= abs(dy)
            if error < 0:
                y += ystep
                error += dx
        self.draw(points)


    def draw(self, coords):
        for point in coords:
            self.canvas.create_rectangle(PIXEL_SIZE * point[0], PIXEL_SIZE * point[1],
                                         PIXEL_SIZE * point[0] + PIXEL_SIZE, PIXEL_SIZE *
point[1] + PIXEL_SIZE,
                                         fill="black", tag="lab3")

    def clean(self):
        self.points = []
        self.stack = []
        self.canvas.delete("lab3")

    def callback(self, func_name):
        if func_name == "Recursive border":
            def func():
                for letter, lines in self.recursive.items():
                    for line in lines:
                        getattr(self, "Bresenham")(line[0][0], line[0][1], line[1][0],
line[1][1])
            return func
        if func_name == "Iteration border":
            def func():
                for letter, lines in self.iteration.items():
                    for line in lines:
                        getattr(self, "Bresenham")(line[0][0], line[0][1], line[1][0],
line[1][1])
            return func
        if func_name == "Line border":
```

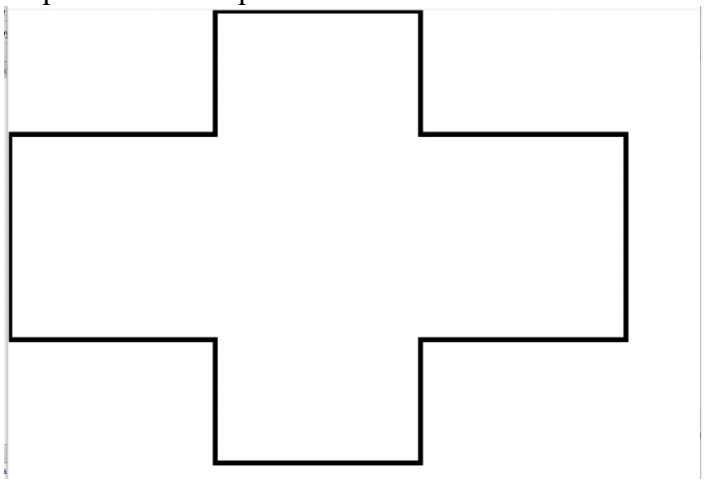
```
            def func():
                for letter, lines in self.line.items():
                    for line in lines:
                        getattr(self, "Bresenham")(line[0][0], line[0][1], line[1][0],
line[1][1])
            return func
        if func_name == "recursiveFill":
            return lambda func_name=func_name: getattr(self, func_name)(2, 2)
        if func_name == "iterationFill":
            return lambda func_name=func_name: getattr(self, func_name)(51, 50)
        if func_name == "lineFill":
            return lambda func_name=func_name: getattr(self, func_name)(12, 5)

    def __init__(self):
        self.points = []
        self.stack = []
        window = Tk()
        window.title("Labwork 3")
        self.canvas = Canvas(window, width=WIDTH, height=HEIGHT, bg="white")
        self.canvas.pack()
        frame = Frame(window)
        frame.pack()
        rec_btn = Button(frame, text="Rec. algoritm", command=self.callback("recursiveFill"))
        rec_btn.grid(row=1, column=4)
        iter_btn = Button(frame, text="Iter. algoritm",
command=self.callback("iterationFill"))
        iter_btn.grid(row=1, column=5)
        line_btn = Button(frame, text="Line", command=self.callback("lineFill"))
        line_btn.grid(row=1, column=6)
        rec_border_btn = Button(frame, text="Rec. border", command=self.callback("Recursive
border"))
        rec_border_btn.grid(row=1, column=1)
        itr_border_btn = Button(frame, text="Iter. border", command=self.callback("Iteration
border"))
        itr_border_btn.grid(row=1, column=2)
        line_border_btn = Button(frame, text="Line border", command=self.callback("Line
border"))
        line_border_btn.grid(row=1, column=3)
        scal_minus_btn = Button(frame, text="Clear", width=10, command=self.clean)
        scal_minus_btn.grid(row=1, column=7)
        window.mainloop()

if __name__ == "__main__":
    RasterizationAlgorithms()
```

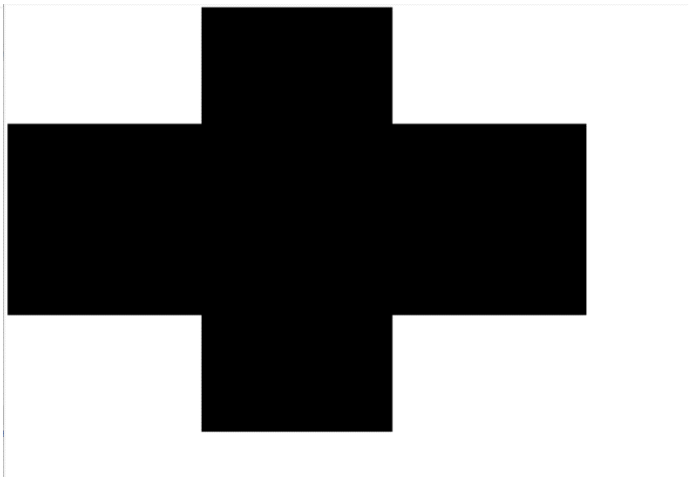
Результат:

Рекурсивний алгоритм:



Ітеративний алгоритм





З використанням списку реберних точок: