Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Джабаров Р.А. ИУ9-32Б

Компьютерная графика.

Лабораторная работа №4

Москва

2023

CODE:

import glfw

import numpy as np

from OpenGL.GL import \*

class Point:

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

def \_\_str\_\_(self):

return "(" + str(self.x) + ", " + str(self.y) + ")"

def \_\_eq\_\_(self, other):

if isinstance(other, Point):

return self.x == other.x and self.y == other.y

return False

width, height = 720, 640

points = []

points\_rast = []

raster\_buffer = np.zeros((height, width, 3), dtype=np.uint8)

def bresenham\_line(p0, p1):

x0, y0 = p0.x, p0.y

x1, y1 = p1.x, p1.y

dx = abs(x1 - x0)

dy = abs(y1 - y0)

sx = -1 if x0 > x1 else 1

sy = -1 if y0 > y1 else 1

err = dx - dy

x = x0

y = y0

while True:

yield x, y

if x == x1 and y == y1:

break

e2 = 2 \* err

if e2 > -dy:

err -= dy

x += sx

if e2 < dx:

err += dx

y += sy

def scanline\_fill(start\_x, start\_y, fill\_color):

stack = [(start\_x, start\_y)]

old\_color = [0, 0, 0]

while stack:

x, y = stack.pop()

while x >= 0 and np.array\_equal(raster\_buffer[x, y], old\_color):

x -= 1

x += 1

left = x

while x < raster\_buffer.shape[0] and np.array\_equal(raster\_buffer[x, y], old\_color):

raster\_buffer[x, y] = fill\_color

x += 1

right = x - 1

if y > 0:

i = left

while i <= right:

if np.array\_equal(raster\_buffer[i, y - 1], old\_color):

while i < right and np.array\_equal(raster\_buffer[i, y - 1], old\_color):

i += 1

if np.array\_equal(raster\_buffer[i, y - 1], old\_color):

stack.append((i, y - 1))

else:

stack.append((i - 1, y - 1))

i += 1

if y < raster\_buffer.shape[1] - 1:

i = left

while i <= right:

if np.array\_equal(raster\_buffer[i, y + 1], old\_color):

while i < right and np.array\_equal(raster\_buffer[i, y + 1], old\_color):

i += 1

if np.array\_equal(raster\_buffer[i, y + 1], old\_color):

stack.append((i, y + 1))

else:

stack.append((i - 1, y + 1))

i += 1

def display(window):

global width, height, raster\_buffer, points\_rast

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

glClearColor(1.0, 1.0, 1.0, 1.0)

new\_width, new\_height = glfw.get\_framebuffer\_size(window)

if new\_width != width or new\_height != height:

width, height = new\_width, new\_height

raster\_buffer = np.zeros((height, width, 3), dtype=np.uint8)

points\_rast = []

glViewport(0, 0, width, height)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

glDrawPixels(width, height, GL\_RGB, GL\_UNSIGNED\_BYTE, raster\_buffer)

glfw.swap\_buffers(window)

glfw.poll\_events()

def main():

if not glfw.init():

return

window = glfw.create\_window(width, height, "Lab4", None, None)

if not window:

glfw.terminate()

return

glfw.make\_context\_current(window)

glfw.set\_key\_callback(window, key\_callback)

glfw.set\_mouse\_button\_callback(window, mouse\_callback)

glEnable(GL\_DEPTH\_TEST)

while not glfw.window\_should\_close(window):

display(window)

glfw.terminate()

def mouse\_callback(window, button, action, mods):

if action == glfw.PRESS:

x\_pos, y\_pos = glfw.get\_cursor\_pos(window)

x\_pos = (x\_pos - width / 2) \* 2 / width

y\_pos = (height / 2 - y\_pos) \* 2 / height

if button == glfw.MOUSE\_BUTTON\_LEFT:

points.append(Point(x\_pos, y\_pos))

x\_pos, y\_pos = glfw.get\_cursor\_pos(window)

points\_rast.append(Point(height - y\_pos, x\_pos))

if len(points\_rast) > 1:

for point in bresenham\_line(points\_rast[-1], points\_rast[-2]):

raster\_buffer[int(point[0])][int(point[1])][0] = 255

if button == glfw.MOUSE\_BUTTON\_RIGHT:

x\_pos, y\_pos = glfw.get\_cursor\_pos(window)

points.append(Point(height - y\_pos, x\_pos))

if len(points\_rast) > 2:

for point in bresenham\_line(points\_rast[0], points\_rast[-1]):

raster\_buffer[int(point[0])][int(point[1])][0] = 255

def key\_callback(window, key, scancode, action, mods):

global points, points\_rast

if action == glfw.PRESS:

if key == glfw.KEY\_SPACE:

points = []

points\_rast = []

if key == glfw.KEY\_R:

scanline\_fill(int(points[-1].x), int(points[-1].y), [0, 255, 0])

if \_\_name\_\_ == '\_\_main\_\_':

main()

I. Теория:

Алгоритм Брезенхема:

Алгоритм Брезенхема используется для рисования линий на экране с использованием целочисленных операций.

Он основан на идее использования инкрементальных ошибок для определения пикселей, через которые должна проходить линия.

В данном коде функция bresenham\_line реализует алгоритм Брезенхема для рисования линии между двумя точками.

Заполнение по сканирующей строке:

Алгоритм заполнения по сканирующей строке используется для закрашивания замкнутых контуров на экране.

Он проходит через все строки изображения и определяет границы замкнутых областей.

В данном коде функция scanline\_fill реализует алгоритм заполнения по сканирующей строке для закрашивания контуров.

II. Практическая реализация:

Класс Point:

Класс Point представляет точку на плоскости с координатами x и y.

Он имеет методы \_\_str\_\_, который возвращает строковое представление точки, и \_\_eq\_\_, который определяет равенство двух точек.

Глобальные переменные:

Переменные width и height задают ширину и высоту окна.

Переменные points и points\_rast хранят точки, добавленные пользователем для рисования и заполнения.

Массив raster\_buffer представляет собой буфер изображения для визуализации.

Функция bresenham\_line:

Функция bresenham\_line принимает две точки p0 и p1 и возвращает последовательность точек, через которые проходит линия между этими точками с использованием алгоритма Брезенхема.

Функция scanline\_fill:

Функция scanline\_fill принимает начальные координаты start\_x и start\_y и цвет fill\_color для заполнения.

Она использует алгоритм заполнения по сканирующей строке для закрашивания области, ограниченной контуром, заданным точками в raster\_buffer.

Функция display:

Функция display отображает содержимое raster\_buffer на экране с помощью библиотеки OpenGL.

Она также обновляет размеры raster\_buffer и points\_rast в соответствии с текущими размерами окна.

Функции обратного вызова mouse\_callback и key\_callback:

Функции mouse\_callback и key\_callback вызываются при событиях нажатия кнопок мыши и клавиш соответственно.

Они обрабатывают действия пользователя, добавляя точки в points и points\_rast и запуская заполнение области или очищая все точки при нажатии соответствующих клавиш.

Функция main:

Функция main инициализирует окно и контекст OpenGL с помощью библиотеки GLFW.

Она также устанавливает функции обратного вызова для обработки событий мыши и клавиатуры.

В основном цикле программы она вызывает функцию display для отображения содержимого на экране и обрабатывает события GLFW до закрытия окна.

III. Заключение:

Данный код реализует простую программу для рисования и заполнения замкнутых контуров. Он использует алгоритм Брезенхема для рисования линий и алгоритм заполнения по сканирующей строке для закрашивания областей. Благодаря использованию библиотеки GLFW и OpenGL, пользователь может взаимодействовать с программой с помощью мыши и клавиатуры.