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 №5

Москва

2023

CODE:

import glfw

import numpy as np

import math

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

class Line:

def \_\_init\_\_(self, p1, p2):

self.p1 = p1

self.p2 = p2

self.x = self.p2.x - self.p1.x

self.y = self.p2.y - self.p1.y

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

return "(" + str(self.p1) + ", " + str(self.p2) + ")"

def scalar\_product(self, other):

if isinstance(other, Line):

x1 = self.p2.x - self.p1.x

y1 = self.p2.y - self.p1.y

x2 = other.p2.x - other.p1.x

y2 = other.p2.y - other.p1.y

return float(x1 \* x2 + y1 \* y2)

return 0

def cross\_product(self, other):

if isinstance(other, Line):

return float(other.x \* self.y - self.x \* other.y)

return 0

def get\_normal(self):

if self.x == 0:

return Line(Point(0, 0), Point(0, 1))

return Line(Point(0, 0), Point(-1 \* (self.y / self.x), 1))

def get\_midpoint(self):

return Point((self.p2.x + self.p1.x) / 2, (self.p2.y + self.p1.y) / 2)

def reverse(self):

temp = self.p1

self.p1 = self.p2

self.p2 = temp

angle = 0.0

angle\_2 = 0.0

switch = False

sizeX = 1200

sizeY = 900

points = []

clipper\_points = []

segments = []

normals = []

clipper\_segments = []

additional\_points = []

additional\_clippers = []

mouse = Point(0, 0)

DRAWING\_LINES = 0

DRAWING\_POLYGON = 1

CLIPPING = 2

state = DRAWING\_LINES

clipper\_points.append(Point(0, 1.0))

clipper\_points.append(Point(0.5, 0))

clipper\_points.append(Point(0, -0.5))

clipper\_points.append(Point(-0.5, 0))

clipper\_points.append(Point(-0.7, 0.6))

points.append(Point(0.7, 0.3))

points.append(Point(-0.7, -0.3))

def projection1(f, a):

return np.array([

[1.0, 0.0, 0.0, 0.0],

[0.0, np.cos(a), np.sin(a), 0.0],

[0.0, -np.sin(a), np.cos(a), 0.0],

[0.0, 0.0, 0.0, 1.0]

])

def projection2(f, a):

return np.array([

[np.cos(f), 0.0, -np.sin(f), 0.0],

[0.0, 1.0, 0.0, 0.0],

[np.sin(f), 0.0, np.cos(f), 0.0],

[0.0, 0.0, 0.0, 1.0]

])

def rotation\_matrix(i, f):

i = np.asarray(i)

assert i.size == 3, "i must be a 3d vector"

i /= np.linalg.norm(i)

c, s = np.cos(f), np.sin(f)

a = 1 - c

R = np.array([[i[0] \*\* 2 \* a + c, i[0] \* i[1] \* a - i[2] \* s, i[0] \* i[2] \* a + i[1] \* s, 0],

[i[0] \* i[1] \* a + i[2] \* s, i[1] \*\* 2 \* a + c, i[1] \* i[2] \* a - i[0] \* s, 0],

[i[0] \* i[2] \* a - i[1] \* s, i[1] \* i[2] \* a + i[0] \* s, i[2] \*\* 2 \* a + c, 0],

[0, 0, 0, 1]], dtype=np.float32)

return R

def points\_to\_segments(points):

segments = []

if len(points) % 2 == 1:

points.pop()

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

segments.append(Line(points[i], points[i + 1]))

return segments

def points\_to\_polygon(points):

segments = []

for i in range(len(points)):

segments.append(Line(points[i], points[(i + 1) % len(points)]))

return segments

def find\_normals(faces):

normals = []

for i in range(len(faces)):

normal = faces[i].get\_normal()

if normal.scalar\_product(Line(faces[i].p1, faces[(i + 1) % len(faces)].p2)) < 0:

normal.reverse()

normals.append(normal)

return normals

def cyrus\_beck(segment, faces, normals, is\_inner):

t\_start = 0

t\_end = 1

for i in range(len(faces)):

d = segment.scalar\_product(normals[i])

w = normals[i].scalar\_product(Line(faces[i].p1, segment.p1))

if d == 0:

if w < 0:

if is\_inner:

return []

return [segment]

continue

t = -1 \* w / d

if d > 0:

t\_start = max(t\_start, t)

if d < 0:

t\_end = min(t\_end, t)

if t\_start <= t\_end:

if is\_inner:

return [Line(Point(segment.p1.x + segment.x \* t\_start, segment.p1.y + segment.y \* t\_start),

Point(segment.p1.x + segment.x \* t\_end, segment.p1.y + segment.y \* t\_end))]

return [Line(Point(segment.p1.x, segment.p1.y),

Point(segment.p1.x + segment.x \* t\_start, segment.p1.y + segment.y \* t\_start)),

Line(Point(segment.p1.x + segment.x \* t\_end, segment.p1.y + segment.y \* t\_end),

Point(segment.p2.x, segment.p2.y))]

if is\_inner:

return []

return [segment]

def clipping(clipper\_segments, segments, is\_inner):

normals = find\_normals(clipper\_segments)

result = []

for s in segments:

visible = cyrus\_beck(s, clipper\_segments, normals, is\_inner)

if len(visible) > 0:

result.extend(visible)

return result

def complete\_polygon(points):

global additional\_points

additional\_points = []

temp = []

if len(points) < 3:

return

while True:

changes = 0

previous = points[len(points) - 1]

i = 0

while i < len(points):

current = points[i]

next = points[(i + 1) % len(points)]

if (Line(previous, current).cross\_product(Line(current, next)) > 0):

changes += 1

temp.append(previous)

points.pop(i)

i -= 1

elif len(temp) > 0:

temp.append(previous)

temp.append(current)

additional\_points.append(temp)

temp = []

previous = current

i += 1

if len(temp) > 0:

temp.append(previous)

temp.append(points[0])

additional\_points.append(temp)

temp = []

if changes == 0:

break

return points

def draw():

glBegin(GL\_LINE\_LOOP)

for point in clipper\_points:

glVertex2f(point.x, point.y)

if state == DRAWING\_POLYGON:

glVertex2f(mouse.x, mouse.y)

glEnd()

glColor3f(0.0, 1.0, 1.0)

glBegin(GL\_LINES)

for s in segments:

glVertex2f(s.p1.x, s.p1.y)

glVertex2f(s.p2.x, s.p2.y)

glEnd()

def display(window):

global segments, clipper\_segments, clipper\_points, additional\_clippers

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

width, height = glfw.get\_framebuffer\_size(window)

glViewport(0, 0, width, height)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

glPushMatrix()

segments = points\_to\_segments(points)

clipper\_segments = points\_to\_polygon(clipper\_points)

segments = clipping(clipper\_segments, segments, False)

print(segments[1].p1)

draw()

glPopMatrix()

glfw.swap\_buffers(window)

glfw.poll\_events()

def main():

if not glfw.init():

return

# Create the window

window = glfw.create\_window(sizeX, sizeY, "Lab5", None, None)

if not window:

glfw.terminate()

return

glfw.make\_context\_current(window)

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

glEnable(GL\_DEPTH\_TEST)

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

display(window)

# Terminate GLFW

glfw.terminate()

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

global angle, angle\_2, switch

if action == glfw.PRESS:

if key == glfw.KEY\_RIGHT:

angle\_2 += 10

if key == 263:

angle\_2 -= 10

if key == glfw.KEY\_UP:

angle -= 10

if key == glfw.KEY\_DOWN:

angle += 10

if key == glfw.KEY\_SPACE:

angle = 0.0

angle\_2 = 0.0

if key == glfw.KEY\_S:

switch = not switch

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

main()

**Теория:**

Данный код является реализацией алгоритма отсечения отрезков Цируса-Бека, который позволяет отсекать линии или полигоны, нарисованные пользователем в графическом приложении, по определенному многоугольнику-отсекателю. Алгоритм основан на поиске и удалении невидимых отрезков, которые не пересекаются с отсекающим многоугольником. Для реализации алгоритма используются математические методы, такие как скалярное и векторное произведение, получение нормали и середины отрезка.

**Практическая реализация:**

Код содержит определения двух классов: Point и Line, представляющих точку и отрезок на плоскости, соответственно. В начале кода определяются несколько переменных, в том числе размеры окна и начальные точки линий и многоугольника-отсекателя. Далее определена функция points\_to\_segments, которая преобразует список точек в список отрезков. Функция points\_to\_polygon выполняет аналогичную задачу, преобразуя список точек в многоугольник. Затем определены несколько методов класса Line. Метод \_\_init\_\_ инициализирует начальную и конечную точки отрезка, а также вычисляет разницу между координатами x и y. Методы scalar\_product и cross\_product выполняют скалярное и векторное произведения двух отрезков. Метод get\_normal возвращает нормаль отрезка, а метод get\_midpoint возвращает точку, лежащую на середине отрезка. Метод reverse меняет начальную и конечную точки отрезка местами. Далее определены несколько функций, которые выполняют обработку пользовательского ввода. Например, функция key\_callback обрабатывает нажатия клавиш на клавиатуре, а функция cursor\_position\_callback обрабатывает движения мыши.

**Заключение:** Изучен алгоритм отсечения определенного типа в пространстве заданной размерности