**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 2**

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий

Студент: Пищик Евгений Сергеевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна. Вариант многогранника: гранная прямая правильная призма.

1. Описание программы

Программа состоит из кода, создающего GUI при помощи библиотеки tkinter для ввода параметров, кода который при помощи библиотеки PyOpenGL отрисовывает заданный многогранник при помощи линий соединяющих пары точек из массива координат.

1. Набор тестов

3.1 Изначальный вид многогранника.

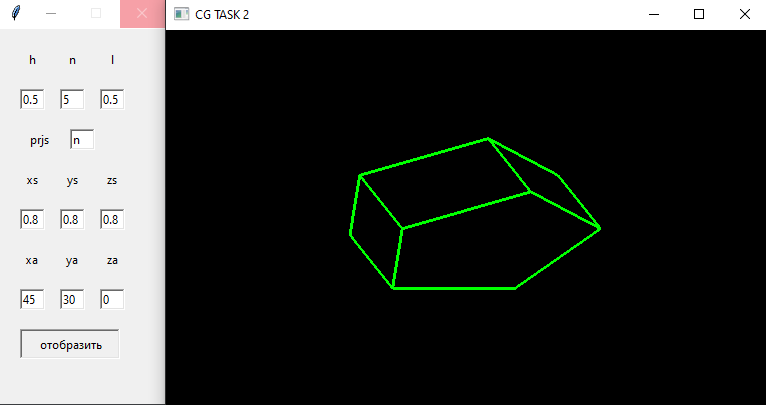
3.2 Вращение многогранника.

3.3 Изменение числа сторон.

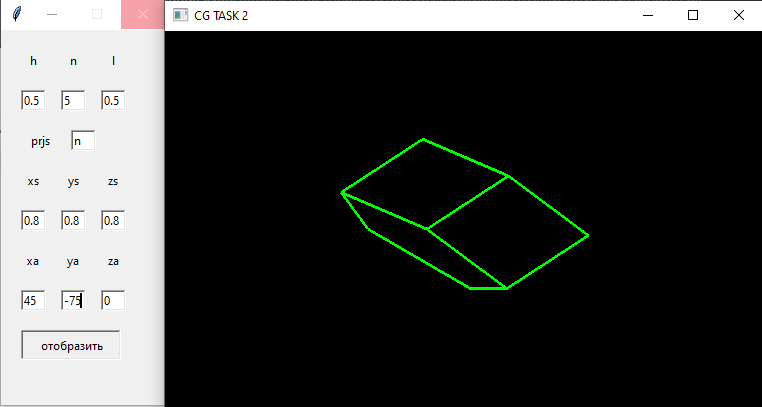
3.4 Уменьшение масштаба.

1. Результаты выполнения тестов

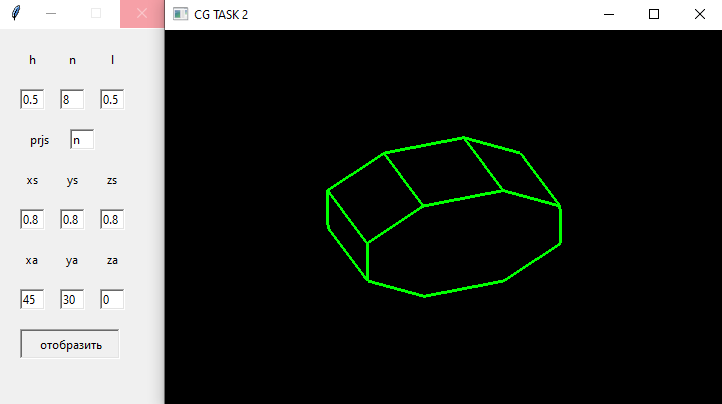
3.1



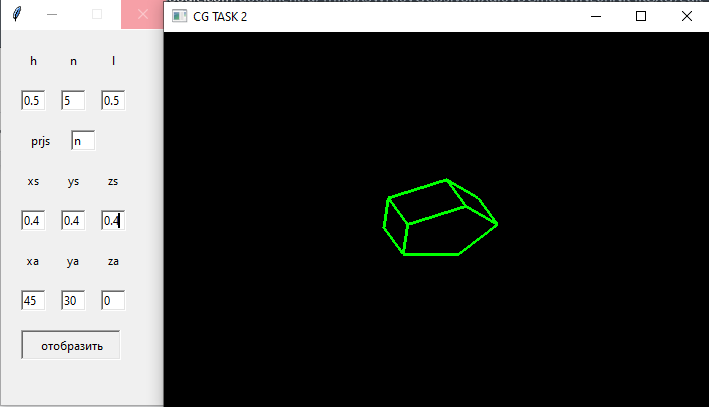
3.2



3.3



3.4



1. Листинг программы

5.1 main.py

import gui

def main() -> None:

gui.main()

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

main()

5.2 gui.py

import tkinter

import subprocess

from typing import NewType, Dict, Any, Tuple, List, Union, Set

def update(ent4, ent5, ent6, ent7, ent8, ent9, ent10, ent11, ent12, ent13) -> None:

try:

subprocess.run(f'C:/Users/jenja/anaconda3/python.exe C:\\Users\\jenja\\Downloads\\VSC\\computer\_graphics\\cg\_exercise\_02\\draw.py {ent4.get()} {ent5.get()} {ent6.get()} {ent7.get()} {ent8.get()} {ent9.get()} {ent10.get()} {ent11.get()} {ent12.get()} {ent13.get()}')

except Exception as \_:

subprocess.run(f'C:\\Users\\SuperPC\\anaconda3\\envs\\p39\\python.exe C:\\Users\\SuperPC\\Downloads\\VSC\\CG\\cg\_exercise\_02\\draw.py {ent4.get()} {ent5.get()} {ent6.get()} {ent7.get()} {ent8.get()} {ent9.get()} {ent10.get()} {ent11.get()} {ent12.get()} {ent13.get()}')

def main() -> None:

WINDOWW = 375

WINDOWH = 165

BUTX = 20

BUTY = 300

BUTWIDTH = 100

BUTHEIGHT = 30

LABX = [(20 + i \* 40) for i in range(3)]

Y = [(20 + i \* 40) for i in range(9)]

LABWIDTH = 25

ENTWIDTH = 25

window = tkinter.Tk()

window.title("task 2")

window.geometry(f'{WINDOWH}x{WINDOWW}')

panel = tkinter.Frame(window, width=WINDOWH, height=WINDOWW)

panel.place(x=0, y=0, width=WINDOWH, height=WINDOWW)

lab4 = tkinter.Label(panel, text='h')

lab4.place(x=LABX[0], y=Y[0], width=LABWIDTH)

lab5 = tkinter.Label(panel, text='n')

lab5.place(x=LABX[1], y=Y[0], width=LABWIDTH)

lab6 = tkinter.Label(panel, text='l')

lab6.place(x=LABX[2], y=Y[0], width=LABWIDTH)

lab7 = tkinter.Label(panel, text='prjs')

lab7.place(x=LABX[0], y=Y[2], width=LABWIDTH + 15)

lab8 = tkinter.Label(panel, text='xs')

lab8.place(x=LABX[0], y=Y[3], width=LABWIDTH)

lab9 = tkinter.Label(panel, text='ys')

lab9.place(x=LABX[1], y=Y[3], width=LABWIDTH)

lab10 = tkinter.Label(panel, text='zs')

lab10.place(x=LABX[2], y=Y[3], width=LABWIDTH)

lab11 = tkinter.Label(panel, text='xa')

lab11.place(x=LABX[0], y=Y[5], width=LABWIDTH)

lab12 = tkinter.Label(panel, text='ya')

lab12.place(x=LABX[1], y=Y[5], width=LABWIDTH)

lab13 = tkinter.Label(panel, text='za')

lab13.place(x=LABX[2], y=Y[5], width=LABWIDTH)

ent4 = tkinter.Entry(panel, bd=2)

ent4.place(x=LABX[0], y=Y[1], width=ENTWIDTH)

ent5 = tkinter.Entry(panel, bd=2)

ent5.place(x=LABX[1], y=Y[1], width=ENTWIDTH)

ent6 = tkinter.Entry(panel, bd=2)

ent6.place(x=LABX[2], y=Y[1], width=ENTWIDTH)

ent7 = tkinter.Entry(panel, bd=2)

ent7.place(x=LABX[1] + 10, y=Y[2], width=ENTWIDTH)

ent8 = tkinter.Entry(panel, bd=2)

ent8.place(x=LABX[0], y=Y[4], width=ENTWIDTH)

ent9 = tkinter.Entry(panel, bd=2)

ent9.place(x=LABX[1], y=Y[4], width=ENTWIDTH)

ent10 = tkinter.Entry(panel, bd=2)

ent10.place(x=LABX[2], y=Y[4], width=ENTWIDTH)

ent11 = tkinter.Entry(panel, bd=2)

ent11.place(x=LABX[0], y=Y[6], width=ENTWIDTH)

ent12 = tkinter.Entry(panel, bd=2)

ent12.place(x=LABX[1], y=Y[6], width=ENTWIDTH)

ent13 = tkinter.Entry(panel, bd=2)

ent13.place(x=LABX[2], y=Y[6], width=ENTWIDTH)

ent4.insert(0, "0.5")

ent5.insert(0, "5")

ent6.insert(0, "0.5")

ent7.insert(0, "n")

ent8.insert(0, "0.8")

ent9.insert(0, "0.8")

ent10.insert(0, "0.8")

ent11.insert(0, "45")

ent12.insert(0, "30")

ent13.insert(0, "0")

but = tkinter.Button(panel, text="отобразить", command=lambda: update(ent4, ent5, ent6, ent7, ent8, ent9, ent10, ent11, ent12, ent13))

but.place(x=BUTX, y=BUTY, width=BUTWIDTH, height=BUTHEIGHT)

window.mainloop()

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

main()

5.3 draw.py

from os import pipe

from typing import Tuple

from OpenGL.GL import \*

from OpenGL.GLU import \*

from OpenGL.GLUT import \*

from typing import NewType, Dict, Any, Tuple, List, Union, Set

import sys

import numpy as np

class Point():

def \_\_init\_\_(self, x: float, y: float, z: float) -> None:

self.x = x

self.y = y

self.z = z

def get\_x(self) -> float:

return self.x

def get\_y(self) -> float:

return self.y

def get\_z(self) -> float:

return self.z

class Line():

def \_\_init\_\_(self, p1: Point, p2: Point) -> None:

self.p1 = p1

self.p2 = p2

def length(self) -> float:

return ((self.p2.get\_x() - self.p1.get\_x())\*\*2 + (self.p2.get\_y() - self.p1.get\_y())\*\*2 + (self.p2.get\_z() - self.p1.get\_z())\*\*2)\*\*0.5

def get\_p1(self) -> Point:

return self.p1

def get\_p2(self) -> Point:

return self.p2

class Plane():

def \_\_init\_\_(self, n, p) -> None:

self.pl = []

for i in range(n):

self.pl.append(p[i])

def \_\_len\_\_(self) -> int:

return len(self.pl)

def get\_pi(self, i) -> Point:

return self.pl[i]

def xflip(angle: float, point: Point) -> Point:

mx = np.matrix([[1.0, 0.0, 0.0], [0.0, np.cos(np.pi \* angle / 180.0), -np.sin(np.pi \* angle / 180.0)], [0.0, np.sin(np.pi \* angle / 180.0), np.cos(np.pi \* angle / 180.0)]])

inp = np.matrix([point.get\_x(), point.get\_y(), point.get\_z()]).T

resp = mx @ inp

return Point(float(resp[0]), float(resp[1]), float(resp[2]))

def yflip(angle: float, point: Point) -> Point:

my = np.matrix([[np.cos(np.pi \* angle / 180.0), 0.0, np.sin(np.pi \* angle / 180.0)], [0.0, 1.0, 0.0], [-np.sin(np.pi \* angle / 180.0), 0.0, np.cos(np.pi \* angle / 180.0)]])

inp = np.matrix([point.get\_x(), point.get\_y(), point.get\_z()]).T

resp = my @ inp

return Point(float(resp[0]), float(resp[1]), float(resp[2]))

def zflip(angle: float, point: Point) -> Point:

mz = np.matrix([[np.cos(np.pi \* angle / 180.0), -np.sin(np.pi \* angle / 180.0), 0.0], [np.sin(np.pi \* angle / 180.0), np.cos(np.pi \* angle / 180.0), 0.0], [0.0, 0.0, 1.0]])

inp = np.matrix([point.get\_x(), point.get\_y(), point.get\_z()]).T

resp = mz @ inp

return Point(float(resp[0]), float(resp[1]), float(resp[2]))

def planeeq(pl: Plane) -> List[Any]:

x1, y1, z1 = pl.get\_pi(0).get\_x(), pl.get\_pi(0).get\_y(), pl.get\_pi(0).get\_z()

x2, y2, z2 = pl.get\_pi(1).get\_x(), pl.get\_pi(1).get\_y(), pl.get\_pi(1).get\_z()

x3, y3, z3 = pl.get\_pi(2).get\_x(), pl.get\_pi(2).get\_y(), pl.get\_pi(2).get\_z()

a1 = x2 - x1

b1 = y2 - y1

c1 = z2 - z1

a2 = x3 - x1

b2 = y3 - y1

c2 = z3 - z1

a = b1 \* c2 - b2 \* c1

b = a2 \* c1 - a1 \* c2

c = a1 \* b2 - b1 \* a2

d = -a \* x1 - b \* y1 - c \* z1

return [a, b, c, d]

def figure() -> Tuple[List[Any], List[Any]]:

points = []

angle = 2 \* np.pi / N

for i in range(N):

points.append(Point(R \* np.sin(angle \* i), R \* np.cos(angle \* i), 0.0))

for i in range(N):

points.append(Point(points[i].get\_x(), points[i].get\_y(), points[i].get\_z() + L))

if PJ == 'x':

points = [Point(0.0, p.get\_y(), p.get\_z()) for p in points]

elif PJ == 'y':

points = [Point(p.get\_x(), 0.0, p.get\_z()) for p in points]

elif PJ == 'z':

points = [Point(p.get\_x(), p.get\_y(), 0.0) for p in points]

points = [Point(XS \* p.get\_x(), YS \* p.get\_y(), ZS \* p.get\_z()) for p in points]

points = [zflip(ZA, yflip(YA, xflip(XA, pi))) for pi in points]

px = 0.0

py = 0.0

pz = 0.0

for p in points:

px += p.get\_x()

py += p.get\_y()

pz += p.get\_z()

px /= len(points)

py /= len(points)

pz /= len(points)

points = [Point(p.get\_x() - px, p.get\_y() - py, p.get\_z() - pz) for p in points]

planes = []

for i in range(N - 1):

planes.append(Plane(4, [points[i], points[i + 1], points[i + N + 1], points[i + N]]))

planes.append(Plane(4, [points[N - 1], points[0], points[N], points[2 \* N - 1]]))

planes.append(Plane(N, [points[i] for i in range(N)]))

planes.append(Plane(N, [points[i] for i in range(N, 2 \* N)]))

view = [planeeq(planes[i])[2] < 0 for i in range(len(planes))]

view[-2] = False

lines = []

for i in range(len(view)):

if view[i]:

for j in range(planes[i].\_\_len\_\_() - 1):

lines.append(Line(planes[i].get\_pi(j), planes[i].get\_pi(j + 1)))

lines.append(Line(planes[i].get\_pi(planes[i].\_\_len\_\_() - 1), planes[i].get\_pi(0)))

return lines, points

def draw() -> None:

lines, \_ = figure()

glColor4f(0, 1, 0, 1)

glLineWidth(3)

for line in lines:

glBegin(GL\_LINE\_STRIP)

points = [line.get\_p1(), line.get\_p2()]

for p in points:

glVertex3fv((p.get\_x(), p.get\_y(), p.get\_z()))

glEnd()

def display() -> None:

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

glClearColor(0.0, 0.0, 0.0, 1.0)

glLoadIdentity()

draw()

glutSwapBuffers()

def main(h, n, r, pj, xs, ys, zs, xa, ya, za) -> None:

global L, N, R, PJ, XS, YS, ZS, XA, YA, ZA

L, N, R, PJ, XS, YS, ZS, XA, YA, ZA = h, n, r, pj, xs, ys, zs, xa, ya, za

W, H = 720, 480

OUR\_W, OUR\_H = 1366, 768

glutInit()

glutInitDisplayMode(GLUT\_RGBA)

glutInitWindowSize(W, H)

glutInitWindowPosition(int((OUR\_W - W) / 2), int((OUR\_H - H) / 2))

glutCreateWindow('CG TASK 2')

glutDisplayFunc(display)

glutIdleFunc(display)

glutMainLoop()

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

try:

if float(sys.argv[1]) < 0.0:

raise ValueError("h can't be < 0")

if int(sys.argv[2]) < 3:

raise ValueError("n can't be < 3")

if float(sys.argv[3]) < 0.0:

raise ValueError("r can't be < 0")

if float(sys.argv[5]) < 0.0 or float(sys.argv[5]) > 1.0:

raise ValueError("xs must be >= 0 and <= 1")

if float(sys.argv[6]) < 0.0 or float(sys.argv[6]) > 1.0:

raise ValueError("ys must be >= 0 and <= 1")

if float(sys.argv[7]) < 0.0 or float(sys.argv[7]) > 1.0:

raise ValueError("zs must be >= 0 and <= 1")

if float(sys.argv[8]) < -90.0 or float(sys.argv[8]) > 90.0:

raise ValueError("xa must be >= -90 and <= 90")

if float(sys.argv[9]) < -90.0 or float(sys.argv[9]) > 90.0:

raise ValueError("ya must be >= -90 and <= 90")

if float(sys.argv[10]) < -90.0 or float(sys.argv[10]) > 90.0:

raise ValueError("zs must be >= -90 and <= 90")

main(h=float(sys.argv[1]), n=int(sys.argv[2]), r=float(sys.argv[3]), pj=str(sys.argv[4]), xs=float(sys.argv[5]), ys=float(sys.argv[6]), \

zs=float(sys.argv[7]), xa=float(sys.argv[8]), ya=float(sys.argv[9]), za=float(sys.argv[10]))

except Exception as e:

print(str(e))

1. Выводы

Выполнив данную лабораторную работу, я изучил алгоритм удаления невидимых линий и плоскостей, поворот фигуры при помощи матриц поворота.

ЛИТЕРАТУРА

1. Документация PyOpenGL: [PyOpenGL documentation](http://pyopengl.sourceforge.net/documentation/index.html)
2. Документация tkinter: [tkinter documentation](https://docs.python.org/3/library/tkinter.html)
3. Матрица поворота: [wiki матрица поворота](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%BE%D0%B2%D0%BE%D1%80%D0%BE%D1%82%D0%B0)
4. Удаление невидимых линий: [алгоритм Робертса](http://algolist.ru/graphics/roberts.php)