**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения фотореалистичных изображений.

Студент: Пищик Евгений Сергеевич

Группа: 80-306

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

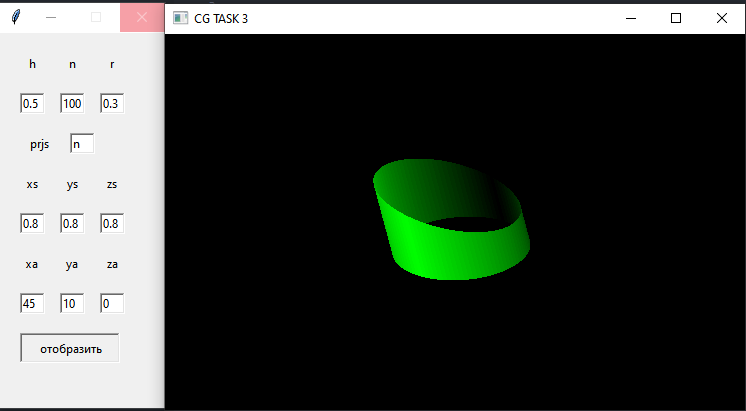
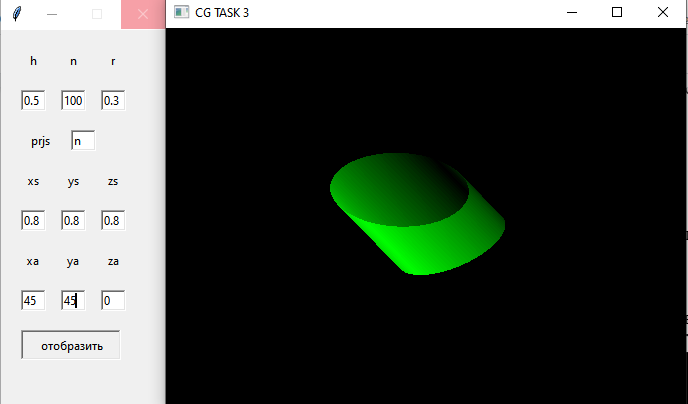
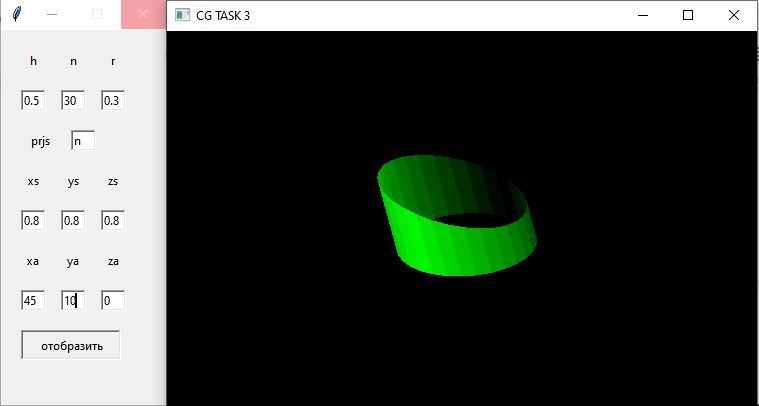
Москва, 2021

1. Постановка задачи

Используя результаты лабораторной работы №2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Вариант многогранника: усеченный прямой круговой цилиндр.

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 аппроксимации его прямоугольника, которые являются набором из 4-х троек координат (x, y, z).

1. Набор тестов
2. Изначальный вид.
3. Поворот по оси y на 45 градусов.
4. Уменьшение точности аппроксимации.
5. Результаты выполнения тестов
6. 
7. 
8. 
9. Листинг программы

main.py

import gui

def main() -> None:

gui.main()

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

main()

gui.py

import tkinter

import subprocess

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\_03\\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\_03\\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 3")

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='r')

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, "100")

ent6.insert(0, "0.3")

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, "10")

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()

draw.py

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

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 figure() -> List[List[Any]]:

points3d = [[], []]

r = R

height = L

approx = N

ang = 30

angles = np.linspace(0.0, 2 \* np.pi, approx)

for angle in angles:

points3d[0].append(Point(r \* np.cos(angle), r \* np.sin(angle), np.tan(ang \* np.pi / 180) \* np.sin(r \* np.cos(angle))))

points3d[1].append(Point(r \* np.cos(angle), r \* np.sin(angle), height))

if PJ == 'x':

points3d[0] = [Point(0.0, p.get\_y(), p.get\_z()) for p in points3d[0]]

points3d[1] = [Point(0.0, p.get\_y(), p.get\_z()) for p in points3d[1]]

elif PJ == 'y':

points3d[0] = [Point(p.get\_x(), 0.0, p.get\_z()) for p in points3d[0]]

points3d[1] = [Point(p.get\_x(), 0.0, p.get\_z()) for p in points3d[1]]

elif PJ == 'z':

points3d[0] = [Point(p.get\_x(), p.get\_y(), 0.0) for p in points3d[0]]

points3d[1] = [Point(p.get\_x(), p.get\_y(), 0.0) for p in points3d[1]]

points3d[0] = [Point(XS \* p.get\_x(), YS \* p.get\_y(), ZS \* p.get\_z()) for p in points3d[0]]

points3d[1] = [Point(XS \* p.get\_x(), YS \* p.get\_y(), ZS \* p.get\_z()) for p in points3d[1]]

points3d[0] = [zflip(ZA, yflip(YA, xflip(XA, pi))) for pi in points3d[0]]

points3d[1] = [zflip(ZA, yflip(YA, xflip(XA, pi))) for pi in points3d[1]]

px = 0.0

py = 0.0

pz = 0.0

for p1, p2 in zip(points3d[0], points3d[1]):

px += p1.get\_x() + p2.get\_x()

py += p1.get\_y() + p2.get\_y()

pz += p1.get\_z() + p2.get\_z()

px /= (len(points3d[0]) + len(points3d[1]))

py /= (len(points3d[0]) + len(points3d[1]))

pz /= (len(points3d[0]) + len(points3d[1]))

points3d[0] = [Point(p.get\_x() - px, p.get\_y() - py, p.get\_z() - pz) for p in points3d[0]]

points3d[1] = [Point(p.get\_x() - px, p.get\_y() - py, p.get\_z() - pz) for p in points3d[1]]

return points3d

def draw() -> None:

points3d = figure()

glColor4f(0, 1, 0, 1)

glLineWidth(1)

pbot, ptop = points3d[0], points3d[1]

resdist = []

for i in range(len(pbot) - 1):

meanx = (pbot[i].get\_x() + pbot[i + 1].get\_x() + ptop[i].get\_x() + ptop[i + 1].get\_x()) / 4.0

meany = (pbot[i].get\_y() + pbot[i + 1].get\_y() + ptop[i].get\_y() + ptop[i + 1].get\_y()) / 4.0

meanz = (pbot[i].get\_z() + pbot[i + 1].get\_z() + ptop[i].get\_z() + ptop[i + 1].get\_z()) / 4.0

resdist.append(((meanx - 1.0)\*\*2 + (meany - 1.0)\*\*2 + (meanz - 1.0)\*\*2)\*\*0.5)

meanx = (pbot[-1].get\_x() + pbot[0].get\_x() + ptop[-1].get\_x() + ptop[0].get\_x()) / 4.0

meany = (pbot[-1].get\_y() + pbot[0].get\_y() + ptop[-1].get\_y() + ptop[0].get\_y()) / 4.0

meanz = (pbot[-1].get\_z() + pbot[0].get\_z() + ptop[-1].get\_z() + ptop[0].get\_z()) / 4.0

resdist.append(((meanx - 10.0)\*\*2 + (meany - 10.0)\*\*2 + (meanz - 10.0)\*\*2)\*\*0.5)

colors = np.linspace(0.0, 1.0, N)

inds = np.argsort(resdist)

color\_ind = [(color, ind) for color, ind in zip(colors, inds)]

for cind in color\_ind:

i = cind[1]

glClearColor(0.0, 0.0, 0.0, 1.0)

glColor4f(0, cind[0], 0, 1)

glBegin(GL\_POLYGON)

glVertex3fv((pbot[i].get\_x(), pbot[i].get\_y(), pbot[i].get\_z()))

glVertex3fv((ptop[i].get\_x(), ptop[i].get\_y(), ptop[i].get\_z()))

glVertex3fv((ptop[i].get\_x(), ptop[i].get\_y(), ptop[i].get\_z()))

if i != N - 1:

glVertex3fv((ptop[i + 1].get\_x(), ptop[i + 1].get\_y(), ptop[i + 1].get\_z()))

glVertex3fv((ptop[i + 1].get\_x(), ptop[i + 1].get\_y(), ptop[i + 1].get\_z()))

glVertex3fv((pbot[i + 1].get\_x(), pbot[i + 1].get\_y(), pbot[i + 1].get\_z()))

glVertex3fv((pbot[i + 1].get\_x(), pbot[i + 1].get\_y(), pbot[i + 1].get\_z()))

else:

glVertex3fv((ptop[0].get\_x(), ptop[0].get\_y(), ptop[0].get\_z()))

glVertex3fv((ptop[0].get\_x(), ptop[0].get\_y(), ptop[0].get\_z()))

glVertex3fv((pbot[0].get\_x(), pbot[0].get\_y(), pbot[0].get\_z()))

glVertex3fv((pbot[0].get\_x(), pbot[0].get\_y(), pbot[0].get\_z()))

glVertex3fv((pbot[i].get\_x(), pbot[i].get\_y(), pbot[i].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 3')

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]) < 1:

raise ValueError("n can't be < 1")

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))

ЛИТЕРАТУРА

1. Документация OpenGL[Электронный ресурс]. URL: <https://www.khronos.org/registry/OpenGL-Refpages/es3/> (дата обращения: 02.11.2021).

2. Документация tkinter[Электронный ресурс]. URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 02.11.2021).