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: Попов Матвей
Группа: 08-308
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2022

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 22: 6 – гранная прямая правильная усеченная пирамида

2. Описание программы

Программа написана на языке программирования python с применением модулей tkinter и numpy, состоит из 5 файлов: main.py, object.py (отвечает за инициализацию пирамиды), camera.py (отвечает за отрисовку), roberts.py (отвечает за удаление невидимых линий), matrix.py (отвечает за повороты пирамиды и масштабирование).

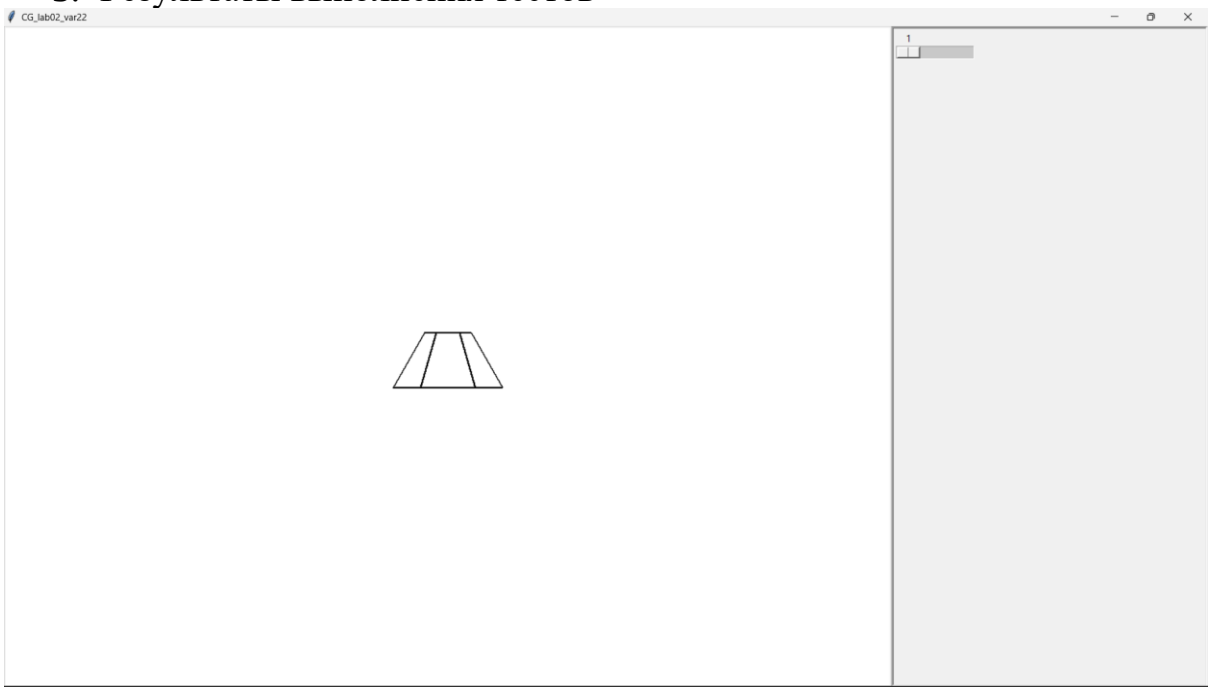
Установка зависимостей (ввести в консоли):

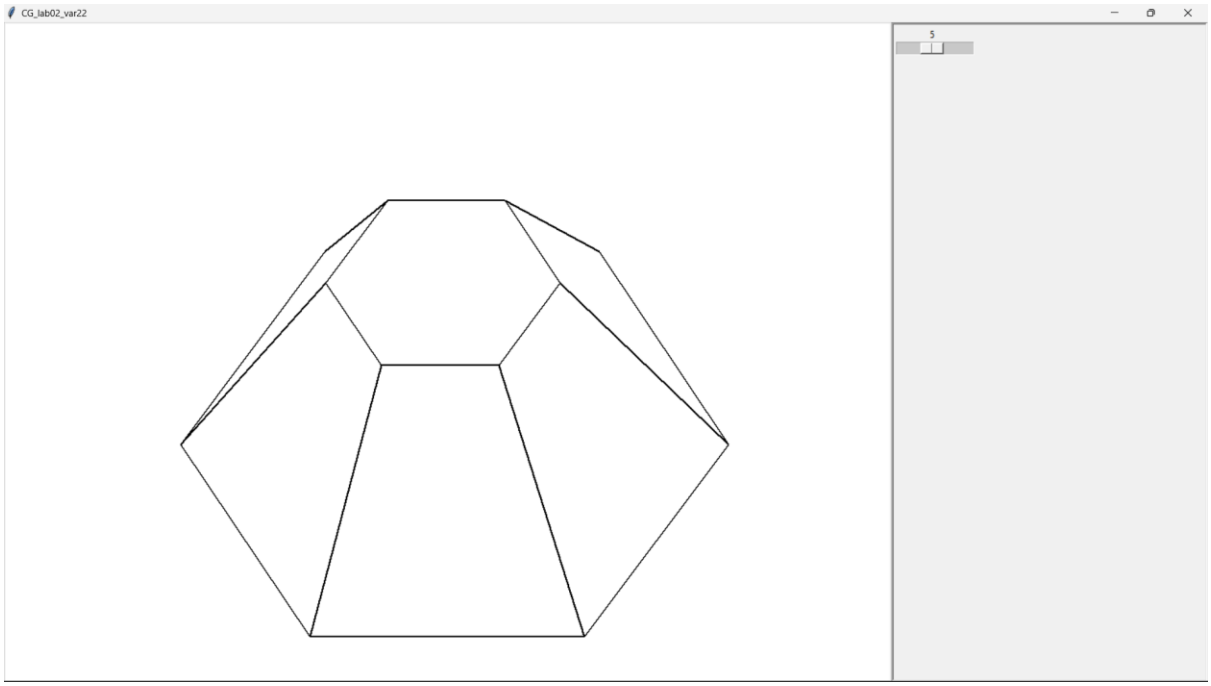
```
pip install numpy
pip install tkinter
```

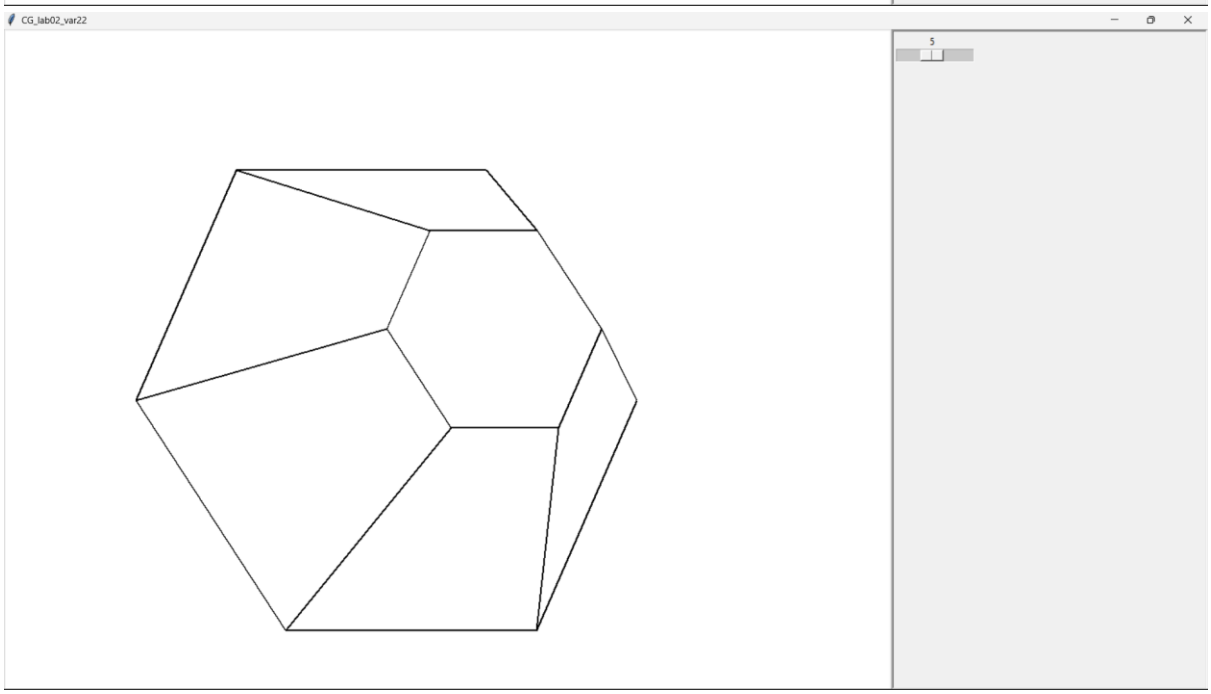
Запуск программы:

```
python main.py
```

3. Результаты выполнения тестов







4. Листинг программы

main.py

```
import tkinter as tk
import numpy as np
import matrix as mtx
import camera as cm
import roberts as rb
import object as obj
from functools import partial

def MVPMatrix(object):
    return mtx.ModelMatrix(object) @ camera.ViewMatrix() @ camera.projectionMatrix()

def OMVPMatrix(object):
    return mtx.ModelMatrix(object) @ camera.ViewMatrix() @ camera.orthogonalMatrix()

def redraw(event, object):
```

```
    camera.height = canvas.winfo_height()
    camera.width = canvas.winfo_width()
    camera.v_fov = camera.h_fov * (camera.height/camera.width)
    drawOrthogonalProjection(object)

def changeStartModel(event, object):
    object.scretchStartModel(sx=scaleX.get(), sy=scaleY.get(), sz=scaleZ.get())
    drawOrthogonalProjection(object)

def changeScaleModel(event, object):
    object.scale = scale.get()
    drawOrthogonalProjection(object)

def keyMove(event, object):
    if event.keycode == 37:
        object.rotY += 3
    elif event.keycode == 38:
        object.rotX -= 3
    elif event.keycode == 39:
        object.rotY -= 3
    elif event.keycode == 40:
        object.rotX += 3
    drawOrthogonalProjection(object)

def drawLine(a, b):
    canvas.create_line(a[0], a[1], b[0], b[1], width=2)

def drawOrthogonalProjection(object):
    canvas.delete("all")
    robAlgo = rb.RobertsAlgo(object.points @ mtx.ModelMatrix(object), object.polygon,
camera.ViewMatrix(), camera.position)
    vertex = object.points @ OMVPMatrix(object)
    for i in range(len(object.polygon)):
        if robAlgo[i] <= 0:
            continue
        for j in range(len(object.polygon[i])):
            drawLine(vertex[object.polygon[i][j]], vertex[object.polygon[i][(j + 1) %
len(object.polygon[i])]])

def drawPerspectiveProjection(object):
    vertex = object.points @ MVPMatrix()
    vertex /= vertex[:, -1].reshape(-1, 1)
    vertex = vertex @ camera.toScreenMatrix()

    for i in range(len(polygon)):
        for j in range(len(polygon[i])):
            a = vertex[object.polygon[i][j]]
            b = vertex[object.polygon[i][(j+1)%len(object.polygon[i])]]
            if not(a and b):
                continue
            drawLine(a, b)

if __name__=="__main__":
    window = tk.Tk()
    window.title("CG_lab02_var22")
    window.columnconfigure(0, weight=4, minsize=550)
    window.columnconfigure([1, 2], weight=1, minsize=10)
    window.rowconfigure([0, 1, 2, 3], weight=1, minsize=100)
    canvas = tk.Canvas(window, bg='white')
    frame = tk.Frame(window, relief="sunken", borderwidth=3)
    labelscale = tk.Label(master=frame, text='Scale')
    scale = tk.Scale(frame, orient='horizontal', resolution=1, from_=1, to=10)
    canvas.grid(row=0, column=0, sticky="nsew", rowspan=4)
    frame.grid(row=0, column=1, sticky="nsew", rowspan=4, columnspan=2)
    scale.grid(row=3, column=2, sticky="nsew", columnspan=3)
    window.geometry("+550+150")
    window.minsize(window.winfo_width(), window.winfo_height())
    camera = cm.Camera(canvas)
    trapeze = obj.Object(sx=10, sy=10, sz=10)
    drawOrthogonalProjection(trapeze)
    scale.bind("<ButtonRelease-1>", partial(changeScaleModel, object=trapeze))
    window.bind("<Key>", partial(keyMove, object=trapeze))
    window.bind("<Configure>", partial(redraw, object=trapeze))
    window.mainloop()
```

object.py

```
import numpy as np
import matrix as mtx

class Object:
    def __init__(self, sx, sy, sz):
        self.rotX = 0
        self.rotY = 0
        self.rotZ = 0
        self.dx = 0
        self.dy = 0
        self.dz = 0
        self.scale = 1
        self.startPoints = []
        self.points = []
        self.polygon = []
        self.readModel()
        self.scretchStartModel(sx, sy, sz)

    def readModel(self):
        self.startPoints.append(np.array(list(map(float, [3.0, -3, 0.0])) + [1]))
        self.startPoints.append(np.array(list(map(float,  [1.5,  -3,  2.59807621135331594])) +
[1]))
        self.startPoints.append(np.array(list(map(float,  [-1.5,  -3,  2.59807621135331594])) +
[1]))
        self.startPoints.append(np.array(list(map(float, [-3, -3, 0])) + [1]))
        self.startPoints.append(np.array(list(map(float, [-1.5, -3, -2.59807621135331594])) +
[1]))
        self.startPoints.append(np.array(list(map(float,  [1.5,  -3,  -2.59807621135331594])) +
[1]))
        self.startPoints.append(np.array(list(map(float, [7.0, 4, 0.0])) + [1]))
        self.startPoints.append(np.array(list(map(float, [3.5, 4, 6.062177826491])) + [1]))
        self.startPoints.append(np.array(list(map(float, [-3.5, 4, 6.062177826491])) + [1]))
        self.startPoints.append(np.array(list(map(float, [-7.0, 4, 0])) + [1]))
        self.startPoints.append(np.array(list(map(float, [-3.5, 4, -6.062177826491])) + [1]))
        self.startPoints.append(np.array(list(map(float, [3.5, 4, -6.062177826491])) + [1]))
        self.polygon.append(list(map(int, [0, 1, 7, 6])))
        self.polygon.append(list(map(int, [1, 2, 8, 7])))
        self.polygon.append(list(map(int, [2, 3, 9, 8])))
        self.polygon.append(list(map(int, [3, 4, 10, 9])))
        self.polygon.append(list(map(int, [4, 5, 11, 10])))
        self.polygon.append(list(map(int, [0, 5, 11, 6])))
        self.polygon.append(list(map(int, [0, 1, 2, 3, 4, 5])))
        self.polygon.append(list(map(int, [6, 7, 8, 9, 10, 11])))

    def scretchStartModel(self, sx, sy, sz):
        self.points = self.startPoints @ mtx.stretchingMatrix(sx, sy, sz)
```

camera.py

```
import math
import numpy as np

class Camera:
    def __init__(self, canvas):
        self.position = [0, 0, 10, 0]
        self.right = [1, 0, 0]
        self.up = [0, 1, 0]
        self.forward = [0, 0, 1]
        self.h_fov = math.pi/3
        self.height = canvas.winfo_height()
        self.width = canvas.winfo_width()
        self.v_fov = self.h_fov * (self.height/self.width)
        self.near_plane = 0.1
        self.far_plan = 100
        self.aspect = self.width/self.height

    def ViewMatrix(self):
        rx, ry, rz = self.right
        ux, uy, uz = self.up
        fx, fy, fz = self.forward
        x, y, z, w = self.position
        matrix = np.array([
            [rx, ux, fx, 0],
            [ry, uy, fy, 0],
            [rz, uz, fz, 0],
            [-x, -y, -z, 1]
        ])
```

```
        return matrix

    def projectionMatrix(self):
        a = 1/(math.tan(self.h_fov/2))
        b = (self.far_plan+self.near_plane)/(self.far_plan-self.near_plane)
        c = -2*self.near_plane*self.far_plan /(self.far_plan-self.near_plane)
        matrix = [
            [a/self.aspect, 0, 0, 0],
            [0, a, 0, 0],
            [0, 0, b, 1],
            [0, 0, c, 0]
        ]
        return matrix

    def orthogonalMatrix(self):
        matrix = np.eye(4, dtype=float)
        matrix[3, 0] = self.width//2
        matrix[3, 1] = self.height//2
        return matrix

    def toScreenMatrix(self):
        matrix = np.eye(4, dtype=float)
        matrix[0, 0] = self.width//2
        matrix[1, 1] = -self.height//2
        matrix[3, 0] = self.width//2
        matrix[3, 1] = self.height//2
        return matrix
```

roberts.py

```
import numpy as np

def RobertsAlgo(modelMatrix, polygon, matrixT, position):
    D = np.array([[-1],[-1],[-1]])
    matrixV = np.zeros((4, len(polygon)))
    for i in range(len(polygon)):
        curMatrix = modelMatrix[polygon[i][:3]][:3, :3]
        matrix = np.linalg.inv(curMatrix) @ D
        matrixV[0][i] = matrix[0]
        matrixV[1][i] = matrix[1]
        matrixV[2][i] = matrix[2]
        matrixV[3][i] = -1

    matrixVT = np.linalg.inv(matrixT) @ matrixV
    final = position @ matrixVT
    return final
```

matrix.py

```
import numpy as np
import math

def rotXMatrix(a):
    rads = math.pi/180 * a
    matrix = np.eye(4, dtype=float)
    matrix[1, 1] = math.cos(rads)
    matrix[1, 2] = math.sin(rads)
    matrix[2, 1] = -math.sin(rads)
    matrix[2, 2] = math.cos(rads)
    return matrix

def rotYMatrix(a):
    rads = math.pi/180 * a
    matrix = np.eye(4, dtype=float)
    matrix[0, 0] = math.cos(rads)
    matrix[0, 2] = -math.sin(rads)
    matrix[2, 0] = math.sin(rads)
    matrix[2, 2] = math.cos(rads)
    return matrix

def rotZMatrix(a):
    rads = math.pi/180 * a
    matrix = np.eye(4, dtype=float)
    matrix[0, 0] = math.cos(rads)
    matrix[0, 1] = math.sin(rads)
    matrix[1, 0] = -math.sin(rads)
    matrix[1, 1] = math.cos(rads)
```

```
    return matrix

def stretchingMatrix(a, b, c):
    matrix = np.eye(4, dtype=float)
    matrix[0, 0] = a
    matrix[1, 1] = b
    matrix[2, 2] = c
    return matrix

def transferMatrix(dx, dy, dz):
    matrix = np.eye(4, dtype=float)
    matrix[3, 0] = dx
    matrix[3, 1] = dy
    matrix[3, 2 ] = dz
    return matrix

def ModelMatrix(object):
    modelMatrix = np.eye(4, dtype=float)
    if object.rotX!=0:
        modelMatrix = modelMatrix @ rotXMatrix(object.rotX)
    if object.rotY!=0:
        modelMatrix = modelMatrix @ rotYMatrix(object.rotY)
    if object.rotZ!=0:
        modelMatrix = modelMatrix @ rotZMatrix(object.rotZ)
    if object.scale>1:
        modelMatrix = modelMatrix @ stretchingMatrix(object.scale, object.scale, object.scale)
    if object.dx!=0 or object.dy!=0 or object.dz!=0:
        modelMatrix = modelMatrix @ transferMatrix(object.dx, object.dy, object.dz)
    return modelMatrix
```

ЛИТЕРАТУРА

1. Numpy documentation [Электронный ресурс]
   URL: https://numpy.org/ (дата обращения: 15.10.2022)
2. Tkinter documentation [Электронный ресурс]
   URL: https://docs.python.org/3/library/tkinter.html (дата обращения 15.10.2022)
3. Алгоритм Робертса [Электронный ресурс]
   URL: http://compgraph.tpu.ru/roberts.htm (дата обращения 15.10.2022)