Московский Авиационный Институт

(Национальный исследовательский университет)

**Факультет прикладной математики и информатики**

**Лабораторные работы**

**По курсу**

**«Компьютерная графика»**

V семестр

Студент: Гусева Софья Романовна

Группа М8О-301Б-20

Руководитель: Филиппов Глеб Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

|  |  |
| --- | --- |
| Лабораторная работа 1 …………........................................................ | 3 |
| Лабораторная работа 2…………………............................................. | 5 |
| Лабораторная работа 3-5……………………………………………. | 10 |
| Лабораторная работа 6………………………………………………. | 16 |
| Лабораторная работа 7………………………………………………. | 21 |

**Лабораторная работа 1**

**Тема:** Построение изображений 2D- кривых.

**Задание**: Написать и отладить программу, строящую изображение заданной замечательной кривой.

**Вариант:** 14. ρ = a\*(1-cosφ)

**Описание**

Для выполнения работы я использовала библиотеку matplotlib. Эта библиотека нужна для работы с двумерными и трехмерными изображениями, так же в ней можно задать достаточно много параметров для настройки графиков.

Так как функция задана в полярных координатах, необходимо указать, чему равны x и y: x=ρ cosφ, y=ρ sinφ.

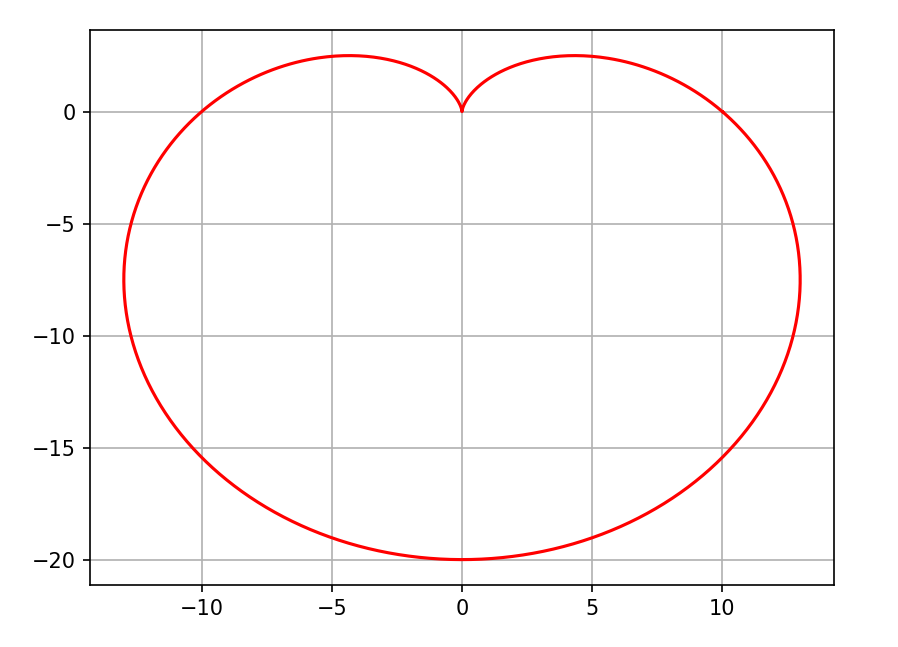
Функция np.arange(0, 2\*np.pi, 0.01) с массивом значений φ от 0 до 2 с шагом 0.01.

**Исходный код**

import numpy as np  
import matplotlib.pyplot as plt  
  
a = float(input("Введите параметр a: "))  
phi = np.arange(0, 2\*np.pi, 0.01)  
r = a \* (1 - np.sin(phi))  
  
x = r \* np.cos(phi)  
y = r \* np.sin(phi)  
  
ax = plt.subplot()  
ax.plot(x, y, color = 'r')  
  
ax.grid(True)  
  
plt.show()

**Результат работы программы:**

При значении параметра a=10



**Вывод:**

Выполнив данную лабораторную работу я попрактиковалась с библиотекой matplotlib и научилась строить графики на python в полярных координатах.

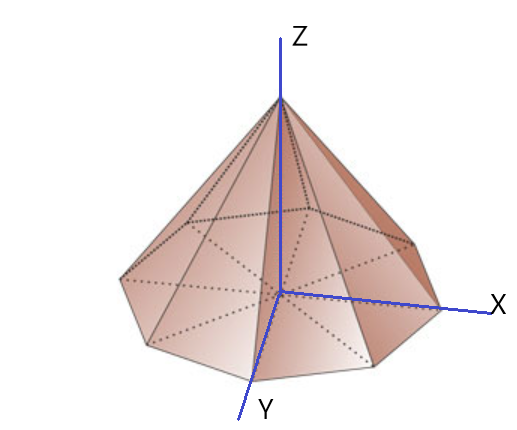
**Лабораторная работа 2**

**Тема:** Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

**Задание:** 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:** 14. 8 – гранная прямая правильная пирамида

**Описание**

 Для выполнения работы я использовала библиотеку matplotlib. Эта библиотека позволяет работать с трехмерными изображениями.

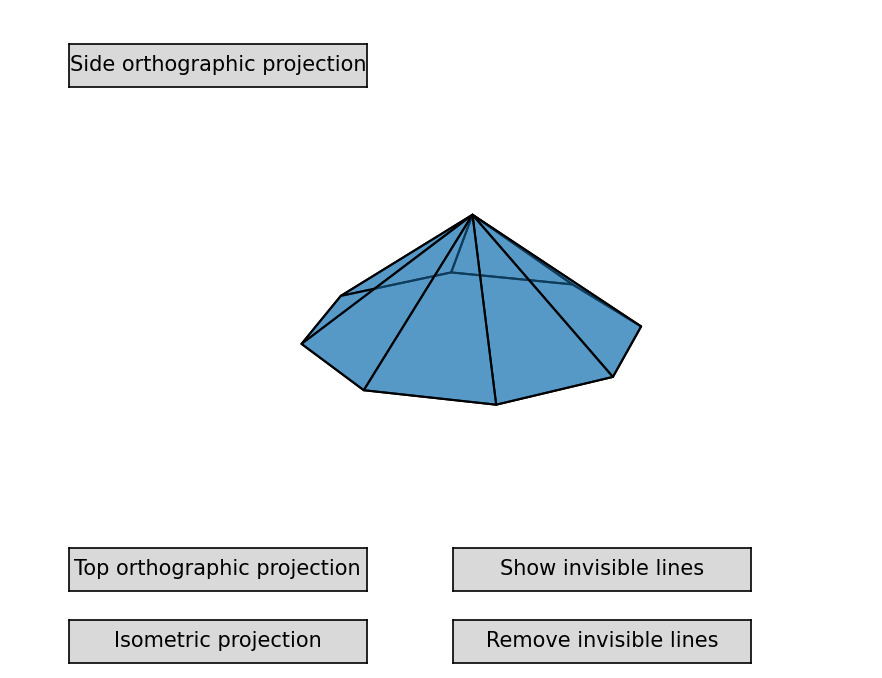
Сначала мне необходимо было расписать каждую грань фигуры. Точка начала координат лежит в центре основания пирамиды. Боковые грани пирамиды – правильные треугольники. Я описывала все грани через радиус описанной основания пирамиды окружности r.

Для того, чтобы пользователю было удобно и понятно пользоваться данным кодом, с помоцью функции def button\_callback\_remove(event) я создала кнопки, с помощью которых можно убирать невидимые линии и поворачивать изображение как угодно.

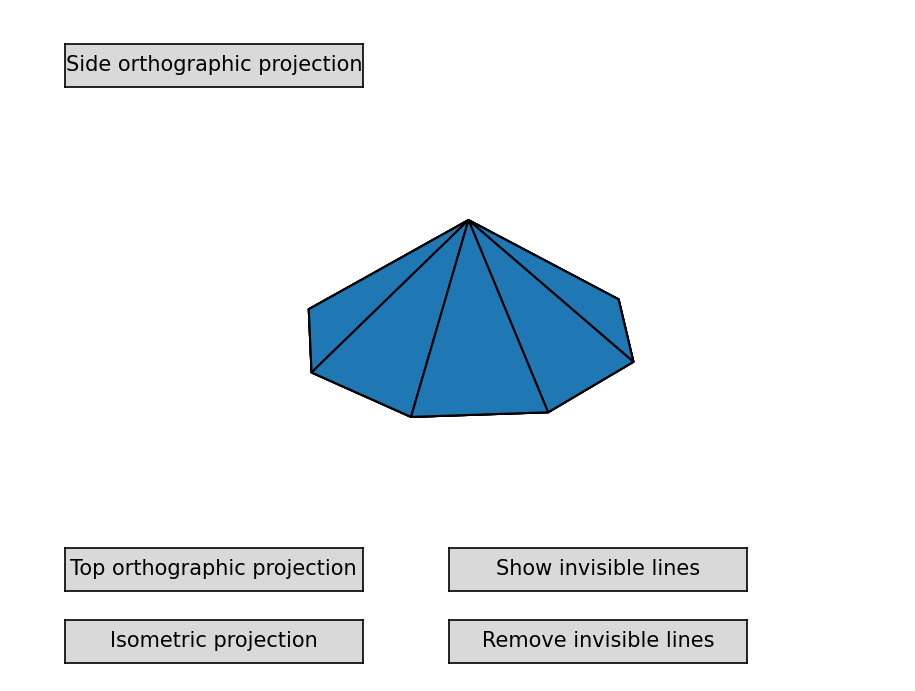
**Исходный код**

import matplotlib.pyplot as plt  
import numpy as np  
from mpl\_toolkits.mplot3d.art3d import Poly3DCollection  
from matplotlib.widgets import Button  
  
fig = plt.figure()  
ax = fig.add\_subplot(111, projection='3d')  
  
r = 6  
  
# Основание пирамиды  
x = [r, np.sqrt(2) \* r / 2, 0, -np.sqrt(2) \* r / 2, -r, -np.sqrt(2) \* r / 2, 0, np.sqrt(2) \* r / 2]  
y = [0, -np.sqrt(2) \* r / 2, -r, -np.sqrt(2) \* r / 2, 0, np.sqrt(2) \* r / 2, r, np.sqrt(2) \* r / 2]  
z = [0, 0, 0, 0, 0, 0, 0, 0]  
sides = [list(zip(x, y, z))]  
  
# Первая боковая грань  
x = [r, np.sqrt(2) \* r / 2, 0]  
y = [0, -np.sqrt(2) \* r / 2, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Вторая боковая грань  
x = [np.sqrt(2) \* r / 2, 0, 0]  
y = [-np.sqrt(2) \* r / 2, -r, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Третья боковая грань  
x = [0, -np.sqrt(2) \* r / 2, 0]  
y = [-r, -np.sqrt(2) \* r / 2, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Четвертая боковая грань  
x = [-np.sqrt(2) \* r / 2, -r, 0]  
y = [-np.sqrt(2) \* r / 2, 0, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Пятая боковая грань  
x = [-r, -np.sqrt(2) \* r / 2, 0]  
y = [0, np.sqrt(2) \* r / 2, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Шестая боковая грань  
x = [-np.sqrt(2) \* r / 2, 0, 0]  
y = [np.sqrt(2) \* r / 2, r, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Седьмая боковая грань  
x = [0, np.sqrt(2) \* r / 2, 0]  
y = [r, np.sqrt(2) \* r / 2, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
# Восьмая боковая грань  
x = [np.sqrt(2) \* r / 2, r, 0]  
y = [np.sqrt(2) \* r / 2, 0, 0]  
z = [0, 0, r]  
sides.append(list(zip(x, y, z)))  
  
poly = Poly3DCollection(sides, alpha=0.5, edgecolors='black')  
  
ax.add\_collection3d(poly)  
  
ax.set\_xlim(-6, 6)  
ax.set\_ylim(-6, 6)  
ax.set\_zlim(-6, 6)  
  
  
def button\_callback\_remove(event):  
 ax.add\_collection3d(Poly3DCollection(sides, alpha=1, edgecolors='black'))  
 plt.draw()  
  
  
button\_ax\_remove = fig.add\_axes([0.5, 0.05, 0.31, 0.06])  
button\_remove = Button(button\_ax\_remove, "Remove invisible lines")  
button\_remove.on\_clicked(button\_callback\_remove)  
  
  
def button\_callback\_show(event):  
 ax.add\_collection3d(Poly3DCollection(sides, alpha=0.5, edgecolors='black'))  
 plt.draw()  
  
  
button\_ax\_show = fig.add\_axes([0.5, 0.15, 0.31, 0.06])  
button\_show = Button(button\_ax\_show, "Show invisible lines")  
button\_show.on\_clicked(button\_callback\_show)  
  
  
def button\_callback\_isometric(event):  
 ax.view\_init(10, 30)  
 plt.draw()  
  
  
button\_ax\_isometric = fig.add\_axes([0.1, 0.05, 0.31, 0.06])  
button\_isometric = Button(button\_ax\_isometric, "Isometric projection")  
button\_isometric.on\_clicked(button\_callback\_isometric)  
  
  
def button\_callback\_orthographic\_top(event):  
 ax.view\_init(90)  
 plt.draw()  
  
  
button\_ax\_orthographic\_top = fig.add\_axes([0.1, 0.15, 0.31, 0.06])  
button\_orthographic\_top = Button(button\_ax\_orthographic\_top, "Top orthographic projection")  
button\_orthographic\_top.on\_clicked(button\_callback\_orthographic\_top)  
  
  
def button\_callback\_orthographic\_hip(event):  
 ax.view\_init(0, 90)  
 plt.draw()  
  
  
button\_ax\_orthographic\_front = fig.add\_axes([0.1, 0.85, 0.31, 0.06])  
button\_orthographic\_front = Button(button\_ax\_orthographic\_front, "Side orthographic projection")  
button\_orthographic\_front.on\_clicked(button\_callback\_orthographic\_hip)  
  
ax.grid(None)  
ax.axis('off')  
plt.show()

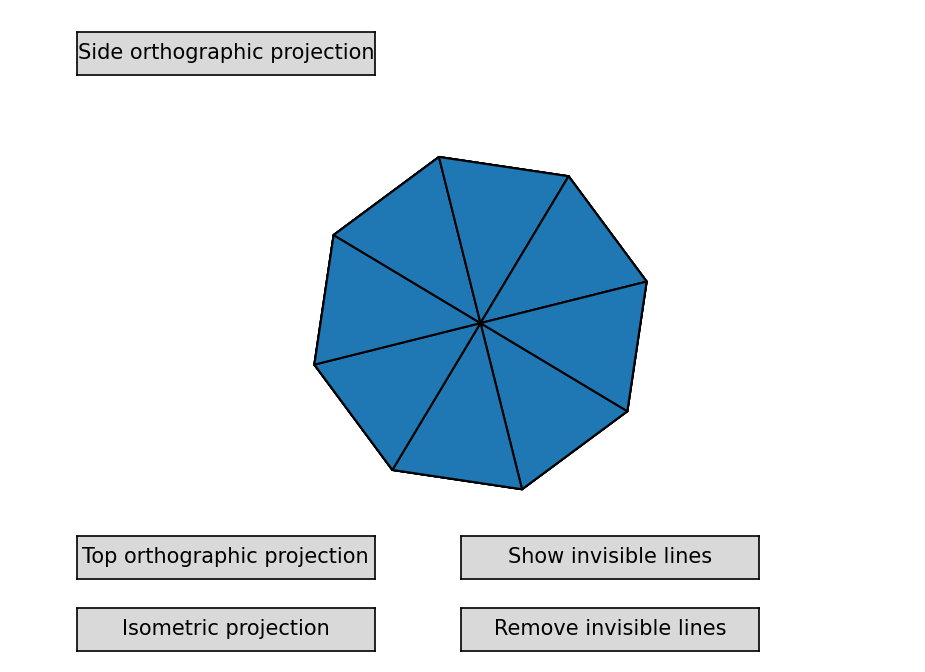
**Результат работы программы:**



Если убрать невидимые линии:



Если посмотреть на фигуру сверху:



**Вывод:**

Выполнив данную лабораторную работу я попрактиковалась с библиотекой matplotlib и научилась строить трехмерные иображения. Так же я изучила еще один виджет для взаимодействия с пользователем: button.

**Лабораторная работа 3-5**

**Тема:** Основы построения фотореалистичных изображений. Ознакомление с технологией OpenGL.

**Задание:** Используя результаты Л.Р.No2, 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.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

**Вариант:** 14. Усеченный прямой эллиптический цилиндр.

**Описание**

Для выполнения работы я использовала библиотеку matplotlib.

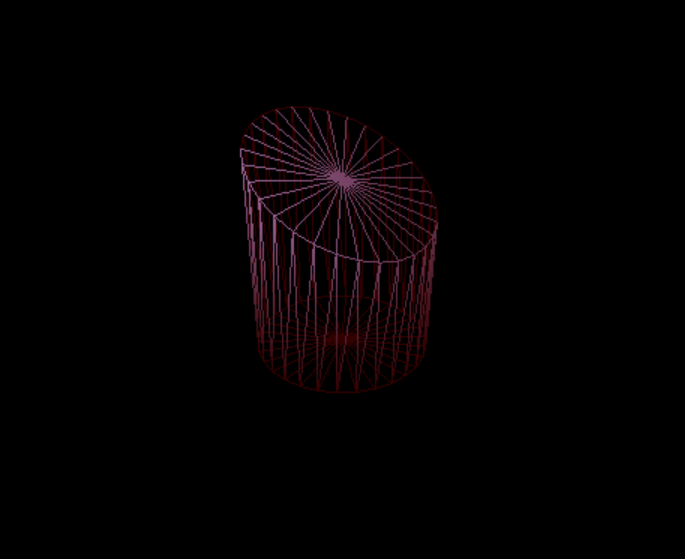
Параметр аппроксимации задается пользователем.

Предусмотрено вращение фигуры с помощью использования клавиш на клавиатуре, изменение числа образующих и маштаба.

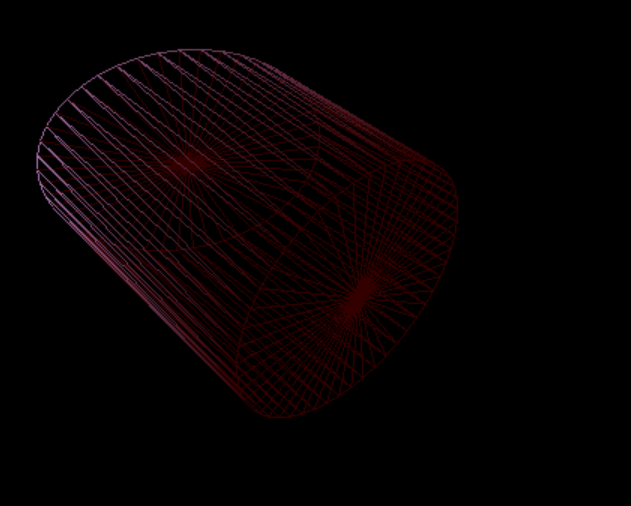
**Исходный код**

import numpy as np  
from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
  
  
# параметры освещения  
light\_pos = (10.0, 0.0, 0.0) # положение источника света  
light\_intensity = 50 # интенсивность света  
reflection = 115  
material = GL\_LINE  
  
ambient = [1.0, 0.0, 0.0, light\_intensity]  
  
x\_rot = 0  
y\_rot = 0  
z\_rot = 0  
  
approximation = int(input("Введите количество образующих (не меньше трех): "))  
R = 2  
size = 1  
  
def cylinder():  
 global approximation, R  
 alpha\_default = np.pi \* (360 / approximation) / 180  
 alpha = alpha\_default  
 vertices = [[0, 0, 0], [0, 2, 0]]  
  
 for i in range(approximation - 1):  
 print((alpha / np.pi) \* 180)  
 x\_i = R \* np.sin(alpha)  
 y\_i = R \* np.sin((np.pi / 2) - alpha)  
 vertices.append([x\_i, y\_i, 0])  
 alpha += alpha\_default  
 print(vertices)  
 for i in range(approximation):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 if i < (approximation - 1):  
 glVertex3fv(vertices[0])  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[i + 2])  
 else:  
 glVertex3fv(vertices[0])  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[1])  
 glEnd()  
  
 vertices.append([0, 0, 4])  
 vertices.append([0, 2, 3])  
 alpha = alpha\_default  
  
 for i in range(approximation - 1):  
 print((alpha / np.pi) \* 180)  
 x\_i = vertices[i+2][0]  
 y\_i = vertices[i+2][1]  
 vertices.append([x\_i, y\_i, (8 - y\_i) / 2])  
 alpha += alpha\_default  
 print(vertices)  
 for i in range(approximation):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 if i < (approximation - 1):  
 glVertex3fv(vertices[approximation + 1])  
 glVertex3fv(vertices[approximation + 2 + i])  
 glVertex3fv(vertices[approximation + 3 + i])  
 else:  
 glVertex3fv(vertices[approximation + 1])  
 glVertex3fv(vertices[approximation + 2 + i])  
 glVertex3fv(vertices[approximation + 2])  
 glEnd()  
  
 for i in range(approximation):  
 if i < (approximation - 1):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[i + 2])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glVertex3fv(vertices[i + 2])  
 glVertex3fv(vertices[i + approximation + 3])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 else:  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glVertex3fv(vertices[approximation])  
 glVertex3fv(vertices[1])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glVertex3fv(vertices[1])  
 glVertex3fv(vertices[approximation + 2])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
  
  
  
  
  
  
def init():  
 glEnable(GL\_DEPTH\_TEST)  
 glDepthFunc(GL\_LEQUAL)  
 glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST) # сглаженные полигоны, больше пикселей для отрисовки  
 glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST) # хорошее качество текстур, цветов  
 glEnable(GL\_NORMALIZE)  
 glEnable(GL\_LIGHTING) # включаем освещение  
 glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient)  
 glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE) # вершины заднего многоугольника зажигаются с помощью параметров  
 # заднего материала и имеют обратную норму перед вычислением уравнения освещения  
  
  
def init\_lighting():  
 glEnable(GL\_LIGHT0) # включаем один источник света  
 glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos) # определяем положение источника света  
  
 l\_dif = (2.0, 2.0, 3.0, light\_intensity)  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, l\_dif)  
 l\_dir = (light\_pos[0], light\_pos[1], light\_pos[2], 1.0)  
 glLightfv(GL\_LIGHT0, GL\_POSITION, l\_dir)  
  
 # делаем затухание света  
 attenuation = float(101 - light\_intensity) / 25.0  
 distance = np.sqrt(pow(light\_pos[0], 2) + pow(light\_pos[1], 2) + pow(light\_pos[2], 2))  
 constant\_attenuation = attenuation / 3.0  
 linear\_attenuation = attenuation / (3.0 \* distance)  
 quadratic\_attenuation = attenuation / (3.0 \* distance \* distance)  
 glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, constant\_attenuation)  
 glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, linear\_attenuation)  
 glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, quadratic\_attenuation)  
  
  
def display():  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(10, 10, 10, 0, 0, 0, 0, 0, 2)  
 glPolygonMode(GL\_FRONT\_AND\_BACK, material)  
 glTranslatef(size, size, size)  
 init\_lighting()  
 glRotatef(x\_rot, 1, 0, 0)  
 glRotatef(y\_rot, 0, 0, 1)  
 glRotatef(z\_rot, 0, 1, 0)  
  
 glPushMatrix() # сохраняем текущее положение "камеры"  
 glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 128 - reflection)  
 cylinder()  
 glPopMatrix() # возвращаем сохраненное положение "камеры"  
 glutSwapBuffers() # выводим все нарисованное в памяти на экран  
  
  
def reshape(width, height):  
 glViewport(0, 0, width, height)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluPerspective(60, float(width) / float(height), 1.0, 60.0)  
 # 1) угол, под которым пользователь видит фигуру, по y;  
 # 2) отношение x/y, которое задаёт положение по x; 3) расстояние до ближней плоскости; 4) до дальней плоскости  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1, 0)  
  
  
def specialkeys(key, x, y):  
 global x\_rot, y\_rot, z\_rot, size, approximation, light\_intensity, material  
 if key == b'w':  
 x\_rot += 5 # вращаем на 5 градусов по оси X  
 if key == b's':  
 x\_rot -= 5 # вращаем на -5 градусов по оси X  
 if key == b'a':  
 y\_rot += 5 # вращаем на 5 градусов по оси Y  
 if key == b'd':  
 y\_rot -= 5 # вращаем на -5 градусов по оси Y  
 if key == b'q':  
 z\_rot += 5 # вращаем на 5 градусов по оси Z  
 if key == b'e':  
 z\_rot -= 5 # вращаем на -5 градусов по оси Z  
 if key == b'=':  
 size += 1 # увеличиваем размер на 1  
 if key == b'-':  
 size -= 1 # уменьшаем размер на 1  
 if key == b'p':  
 approximation += 1 # увеличиваем число образующих на 1  
 if key == b'o':  
 approximation -= 1 # уменьшаем число образующих на 1  
 approximation = max(3, approximation)  
 if key == b'l':  
 light\_intensity += 5 # увеличиваем интенсивность света на 5  
 light\_intensity = min(100, light\_intensity)  
 if key == b'k':  
 light\_intensity -= 5 # уменьшаем интенсивность света на 5  
 light\_intensity = max(-100, light\_intensity)  
 if key == b'z':  
 if material == GL\_FILL:  
 material = GL\_LINE  
 else:  
 material = GL\_FILL  
  
 glutPostRedisplay() # вызываем процедуру перерисовки  
  
  
def main():  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH) # используем двойную буферизацию и формат RGB  
 glutInitWindowSize(500, 500)  
 glutInitWindowPosition(0, 0)  
 glutInit(sys.argv) # инициализируем opengl  
 glutCreateWindow("lab 3-4-5")  
 glutDisplayFunc(display) # определяем функцию для отрисовки  
 glutReshapeFunc(reshape) # определяем функцию для масштабирования  
 glutKeyboardFunc(specialkeys) # определяем функцию для обработки нажатия клавиш  
 init()  
 glutMainLoop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Результат работы программы:**



При изменении числа образующих, увеличении масштаба и поворота цилиндра по осям X, Y, Z:



**Вывод:**

Выполнив данную лабораторную работу я научилась аппроксимировать фигуры в python, разобралась, как интерактивно можно изменять параметры фигуры такие как: положение в пространстве, количество образующих.

**Лабораторная работа 6**

**Тема:** Создание шейдерных анимационных эффектов в OpenGL 2.1

**Задание:** Для поверхности, созданной в л.р. No5, обеспечить выполнение следующего шейдерного эффекта:

**Вариант:** Аннимация. Прозрачность изменяется по синусоидальному закону.

**Описание**

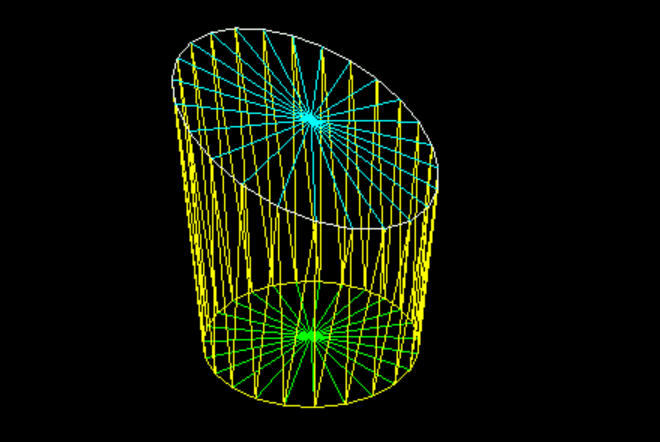
Для выполнения работы я использовала библиотеку matplotlib.

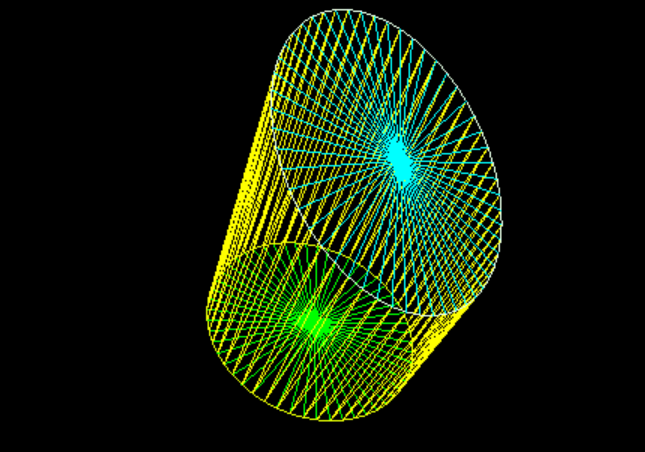
Для создания анимации в программу был добавлен вечный цикл, изменяющий интенсивность источника света по закону.

**Исходный код**

import numpy as np  
from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
  
  
material = GL\_LINE  
  
alpha\_channel = 1  
time = 0  
x\_rot = 0  
y\_rot = 0  
z\_rot = 0  
  
approximation = int(input("Введите количество образующих (не меньше трех): "))  
R = 2  
size = 1  
  
def cylinder():  
 global approximation, R  
 alpha\_default = np.pi \* (360 / approximation) / 180  
 alpha = alpha\_default  
 vertices = [[0, 0, 0], [0, 2, 0]]  
  
 for i in range(approximation - 1):  
 x\_i = R \* np.sin(alpha)  
 y\_i = R \* np.sin((np.pi / 2) - alpha)  
 vertices.append([x\_i, y\_i, 0])  
 alpha += alpha\_default  
 for i in range(approximation):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 if i < (approximation - 1):  
 glColor4f(0, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[0])  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[i + 2])  
 else:  
 glColor4f(0, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[0])  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[1])  
 glEnd()  
  
 vertices.append([0, 0, 4])  
 vertices.append([0, 2, 3])  
 alpha = alpha\_default  
  
 for i in range(approximation - 1):  
 x\_i = vertices[i+2][0]  
 y\_i = vertices[i+2][1]  
 vertices.append([x\_i, y\_i, (8 - y\_i) / 2])  
 alpha += alpha\_default  
 for i in range(approximation):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 if i < (approximation - 1):  
 glColor4f(0, 1, 1, alpha\_channel)  
 glVertex3fv(vertices[approximation + 1])  
 glVertex3fv(vertices[approximation + 2 + i])  
 glVertex3fv(vertices[approximation + 3 + i])  
 else:  
 glColor4f(0, 1, 1, alpha\_channel)  
 glVertex3fv(vertices[approximation + 1])  
 glVertex3fv(vertices[approximation + 2 + i])  
 glVertex3fv(vertices[approximation + 2])  
 glEnd()  
  
 for i in range(approximation):  
 if i < (approximation - 1):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glColor4f(1, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[i + 1])  
 glVertex3fv(vertices[i + 2])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glColor4f(1, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[i + 2])  
 glVertex3fv(vertices[i + approximation + 3])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 else:  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glColor4f(1, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[approximation])  
 glVertex3fv(vertices[1])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
 glBegin(GL\_TRIANGLE\_STRIP)  
 glColor4f(1, 1, 0, alpha\_channel)  
 glVertex3fv(vertices[1])  
 glVertex3fv(vertices[approximation + 2])  
 glVertex3fv(vertices[i + approximation + 2])  
 glEnd()  
  
  
def init():  
 glEnable(GL\_DEPTH\_TEST)  
 glEnable(GL\_BLEND)  
 glDepthFunc(GL\_LEQUAL)  
  
 glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE)  
  
 glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST) # сглаженные полигоны, больше пикселей для отрисовки  
 glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST) # хорошее качество текстур, цветов  
 glEnable(GL\_NORMALIZE)  
  
  
def display():  
 global time, x\_rot, y\_rot, z\_rot, size, material, alpha\_channel  
  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(10, 10, 10, 0, 0, 0, 0, 0, 2)  
 glPolygonMode(GL\_FRONT\_AND\_BACK, material)  
 glTranslatef(size, size, size)  
 glRotatef(x\_rot, 1, 0, 0)  
 glRotatef(y\_rot, 0, 0, 1)  
 glRotatef(z\_rot, 0, 1, 0)  
  
 glPushMatrix() # сохраняем текущее положение "камеры"  
  
 time += 0.01  
 alpha\_channel = np.sin(time)  
  
 cylinder()  
 glPopMatrix() # возвращаем сохраненное положение "камеры"  
 glutSwapBuffers() # выводим все нарисованное в памяти на экран  
 glutPostRedisplay()  
  
  
def reshape(width, height):  
 glViewport(0, 0, width, height)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluPerspective(60, float(width) / float(height), 1.0, 60.0)  
 # 1) угол, под которым пользователь видит фигуру, по y;  
 # 2) отношение x/y, которое задаёт положение по x; 3) расстояние до ближней плоскости; 4) до дальней плоскости  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1, 0)  
  
  
def specialkeys(key, x, y):  
 global x\_rot, y\_rot, z\_rot, size, approximation, light\_intensity, material  
 if key == b'w':  
 x\_rot += 5 # вращаем на 5 градусов по оси X  
 if key == b's':  
 x\_rot -= 5 # вращаем на -5 градусов по оси X  
 if key == b'a':  
 y\_rot += 5 # вращаем на 5 градусов по оси Y  
 if key == b'd':  
 y\_rot -= 5 # вращаем на -5 градусов по оси Y  
 if key == b'q':  
 z\_rot += 5 # вращаем на 5 градусов по оси Z  
 if key == b'e':  
 z\_rot -= 5 # вращаем на -5 градусов по оси Z  
 if key == b'=':  
 size += 1 # увеличиваем размер на 1  
 if key == b'-':  
 size -= 1 # уменьшаем размер на 1  
 if key == b'p':  
 approximation += 1 # увеличиваем число образующих на 1  
 if key == b'o':  
 approximation -= 1 # уменьшаем число образующих на 1  
 approximation = max(3, approximation)  
 if key == b'z':  
 if material == GL\_FILL:  
 material = GL\_LINE  
 else:  
 material = GL\_FILL  
  
 glutPostRedisplay() # вызываем процедуру перерисовки  
  
  
def main():  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH) # используем двойную буферизацию и формат RGBA  
 glutInitWindowSize(500, 500)  
 glutInitWindowPosition(0, 0)  
 glutInit(sys.argv) # инициализируем opengl  
 glutCreateWindow("lab 6")  
 glutDisplayFunc(display) # определяем функцию для отрисовки  
 glutReshapeFunc(reshape) # определяем функцию для масштабирования  
 glutKeyboardFunc(specialkeys) # определяем функцию для обработки нажатия клавиш  
 init()  
 glutMainLoop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Результат работы программы:**





**Вывод:**

Выполнив данную лабораторную работу я создала пройстейший аннимационный эффект для своей фигуры.

**Лабораторная работа 7**

**Тема:** Построение плоских полиномиальных кривых.

**Задание:** Написать программу, строящую полиномиальную кривую по заданным точкам. Обеспечить возможность изменения позиции точек и, при необходимости, значений касательных векторов и натяжения.

**Вариант:** 14. Интерполяционный многочлен Лагранжа по шести точкам.

**Описание**

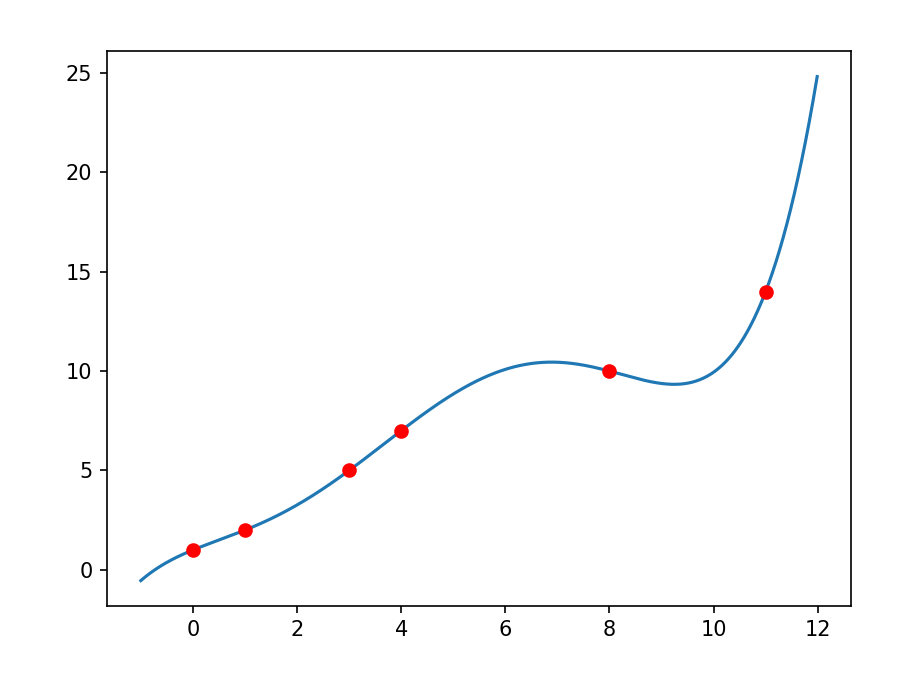
Для выполнения работы я использовала библиотеку matplotlib.

После ввода пользователем координат всех 6 точек кривой, строится интерполяционный многочлен Лагранжа. Для это я использовала модуль lagrange. После чего происходит отображение всех точек по их координатам, взятым из массива, на плоскости координат, и проводится кривая по данным точкам.

**Исходный код**

import numpy as np  
import matplotlib.pyplot as plt  
  
points\_array = []  
  
x1y1 = input("Введите координаты 1 точки в виде \"x y\": ")  
x2y2 = input("Введите координаты 2 точки в виде \"x y\": ")  
x3y3 = input("Введите координаты 3 точки в виде \"x y\": ")  
x4y4 = input("Введите координаты 4 точки в виде \"x y\": ")  
x5y5 = input("Введите координаты 5 точки в виде \"x y\": ")  
x6y6 = input("Введите координаты 6 точки в виде \"x y\": ")  
points\_array.append(int(x1y1.split()[0]))  
points\_array.append(int(x1y1.split()[1]))  
points\_array.append(int(x2y2.split()[0]))  
points\_array.append(int(x2y2.split()[1]))  
points\_array.append(int(x3y3.split()[0]))  
points\_array.append(int(x3y3.split()[1]))  
points\_array.append(int(x4y4.split()[0]))  
points\_array.append(int(x4y4.split()[1]))  
points\_array.append(int(x5y5.split()[0]))  
points\_array.append(int(x5y5.split()[1]))  
points\_array.append(int(x6y6.split()[0]))  
points\_array.append(int(x6y6.split()[1]))  
  
min = min(points\_array[::2]) - 1  
max = max(points\_array[::2]) + 1  
x = np.arange(min, max, 0.01)  
  
def lagrange(points\_array, x):  
 y = points\_array[1] \* (x - points\_array[2]) \* (x - points\_array[4]) \* (x - points\_array[6]) \* (x - points\_array[  
 8]) \* (x - points\_array[10]) /\  
 ((points\_array[0] - points\_array[2]) \* (points\_array[0] - points\_array[4]) \*\  
 (points\_array[0] - points\_array[6]) \* (points\_array[0] - points\_array[8]) \* (points\_array[0] - points\_array[  
 10])) + \  
 points\_array[3] \* (x - points\_array[0]) \* (x - points\_array[4]) \* (x - points\_array[6]) \* (x - points\_array[  
 8]) \* (x - points\_array[10]) /\  
 ((points\_array[2] - points\_array[0]) \* (points\_array[2] - points\_array[4]) \*\  
 (points\_array[2] - points\_array[6]) \* (points\_array[2] - points\_array[8]) \* (points\_array[2] - points\_array[  
 10])) + \  
 points\_array[5] \* (x - points\_array[0]) \* (x - points\_array[2]) \* (x - points\_array[6]) \* (x - points\_array[  
 8]) \* (x - points\_array[10]) /\  
 ((points\_array[4] - points\_array[0]) \* (points\_array[4] - points\_array[2]) \*\  
 (points\_array[4] - points\_array[6]) \* (points\_array[4] - points\_array[8]) \* (points\_array[4] - points\_array[  
 10])) + \  
 points\_array[7] \* (x - points\_array[0]) \* (x - points\_array[2]) \* (x - points\_array[4]) \* (x - points\_array[  
 8]) \* (x - points\_array[10]) /\  
 ((points\_array[6] - points\_array[0]) \* (points\_array[6] - points\_array[2]) \*\  
 (points\_array[6] - points\_array[4]) \* (points\_array[6] - points\_array[8]) \* (points\_array[6] - points\_array[  
 10])) + \  
 points\_array[9] \* (x - points\_array[0]) \* (x - points\_array[2]) \* (x - points\_array[4]) \* (x - points\_array[  
 6]) \* (x - points\_array[10]) /\  
 ((points\_array[8] - points\_array[0]) \* (points\_array[8] - points\_array[2]) \*\  
 (points\_array[8] - points\_array[4]) \* (points\_array[8] - points\_array[6]) \* (points\_array[8] - points\_array[  
 10])) + \  
 points\_array[11] \* (x - points\_array[0]) \* (x - points\_array[2]) \* (x - points\_array[4]) \* (x - points\_array[  
 6]) \* (x - points\_array[8]) /\  
 ((points\_array[10] - points\_array[0]) \* (points\_array[10] - points\_array[2]) \*\  
 (points\_array[10] - points\_array[4]) \* (points\_array[10] - points\_array[6]) \* (points\_array[10] - points\_array[  
 8]))  
 return y  
  
plt.plot(x, lagrange(points\_array, x))  
plt.plot(points\_array[0], points\_array[1], marker="o", color="red")  
plt.plot(points\_array[2], points\_array[3], marker="o", color="red")  
plt.plot(points\_array[4], points\_array[5], marker="o", color="red")  
plt.plot(points\_array[6], points\_array[7], marker="o", color="red")  
plt.plot(points\_array[8], points\_array[9], marker="o", color="red")  
plt.plot(points\_array[10], points\_array[11], marker="o", color="red")  
plt.show()

**Результат работы программы:**



**Вывод:**

Выполнив данную лабораторную работу я научилась строить кривые по точкам, познакомилась с многочленом Лагранжа и научилась применять его в построении графиков.