Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Высшая школа программной инженерии**

**КУРСОВАЯ РАБОТА**

**Моделирование динамической сцены, вариант №2**

по дисциплине «Компьютерная графика»

Выполнил студент гр. в3530904/00330  Баганов В.C.

Руководитель Леонтьева Т. В.

Санкт-Петербург

2022

ЗАДАНИЕ

НА ВЫПОЛНЕНИЕ КУРСОВОЙ РАБОТЫ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| студенту группы | | в3530904/00330 | |  | *Баганов Владимир Сергеевич* | | | | | | | | |
|  | | *(номер группы)* | |  | *(фамилия, имя, отчество)* | | | | | | | | |
| ***1. Тема проекта (работы)*** | | | | *Моделирование динамической сцены,* | | | | | | | | | |
| *Вариант 2* | | | | | | | | | | | | | |
| ***2. Срок сдачи студентом законченного проекта (работы)*** | | | | | | | | | | | | | *08.11.2022* |
| ***3. Исходные данные к проекту (работе)*** | | | | | | | | | *Задание с параметрами* | | | | |
| *согласно варианту, среда разработки PyCharm, графическая* | | | | | | | | | | | | | |
| *библиотека PyopenGL (Python)* | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| ***4.Содержание пояснительной записки***(перечень подлежащих разработке вопросов): | | | | | | | | | | | | | |
| *Задание, описание алгоритма, описание реализации, скрин-шоты,* | | | | | | | | | | | | | |
| *список литературы, приложение (код программы)* | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Примерный объем пояснительной записки | | | | | | *5-10* | | | | страниц машинописного | | | |
| текста | | | | | | | | | | | | | |
| 5. Перечень графического материала (с указанием обязательных чертежей и | | | | | | | | | | | | | |
| плакатов) | ------- | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| 6. Консультанты | | | ------- | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| 7. Дата получения задания: «02.09.2022» | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | |
| Руководитель | | | | |  | | |  | | | *Леонтьева Т. В.* | | |
|  | | | | | *(подпись)* | | |  | | | *(инициалы, фамилия)* | | |
| Задание принял к исполнению | | | | |  | |  | | | | *Баганов В.С.* | | |
|  | | | | | *(подпись)* | |  | | | | *(инициалы, фамилия)* | | |
|  | | | | | | | | | | | | 02.09.2022 | |
|  | | | | | | | | | | | | *(дата)* | |

Оглавление

[ЗАДАНИЕ 4](#_Toc100772096)

[ОПИСАНИЕ АЛГОРИТМА 5](#_Toc100772097)

[ОПИСАНИЕ РЕАЛИЗАЦИИ 6](#_Toc100772098)

[СКРИНШОТЫ ПРИЛОЖЕНИЯ 7](#_Toc100772099)

[СПИСОК ЛИТЕРАТУРЫ 9](#_Toc100772100)

[ПРИЛОЖЕНИЕ 10](#_Toc100772101)

# ЗАДАНИЕ

1. Изобразить текстурированные куб/тетраэдр, расположенный на плоскости.

2. Реализовать освещение (один источник).

3. Рассчитать и изобразить перекатывание куба/тетраэдра по плоскости.

# ОПИСАНИЕ АЛГОРИТМА

В ходе реализации этой части в центре было размещено полотно, в центре полотна был поставлен куб/тетраэдр. На небольшом расстоянии от него был размещён куб/тетраэдр.

Куб/тетраэдр были построены по точкам (c полигонами) с координатам для натяжки текстуры, относительно которых бы вращался куб/тетраэдр, и был подобран вектор вращения и перемещения с поворотом на 90 градусов. Точки должны были быть размещены на ребрах куба/тетраэдра, а результатом вектора вращения должно быть падение на грань или ребро фигуры.

С помощью алгоритма были найдены 4 точки вокруг которых осуществляется поворот, а также был подобран необходимый вектор вращения. Соответственно программа последовательно проходит через эти 4 точки последовательно наращивая угол наклона в каждой из них, пока не дойдёт до определённого предела.

# ОПИСАНИЕ РЕАЛИЗАЦИИ

В начале программы запускаются функции: glutInit(), glutInitDisplayMode(), glutInitWindowSize(), glutInitWindowPosition(), glutCreateWindow(). В них задаются параметры окна.

Дальше идёт функция glutDisplayFunc(), она ответственна за вывод на экран, в неё необходимо передать функцию которая будет отрисовывать объекты. В ней отрисовывается куб и тетраэдр. Задаются точки поворота и векторы наклона.

Куб/тетраэдр были отрисованы по точкам с соединением в полигоны. Для создания куба используется функция glBegin() с параметром GL\_QUADS, для тетраэдра GL\_TRIANGLE\_STRIP.

Каждая тройка вершин трактуется, как отдельный треугольник. n-й треугольник определяют вершины с порядковыми номерами (3n−2), (3n−1) и 3n. Всего нарисуется N/3 треугольников.

GL\_TRIANGLE\_STRIP

Нарисуется группа соединённых треугольников. После первых двух вершин, каждая последующая вершина будет добавлять один треугольник. n-й треугольник будут задавать вершины с порядковыми номерами 1, n+1 и n+2. Нарисуется N−2 треугольников.

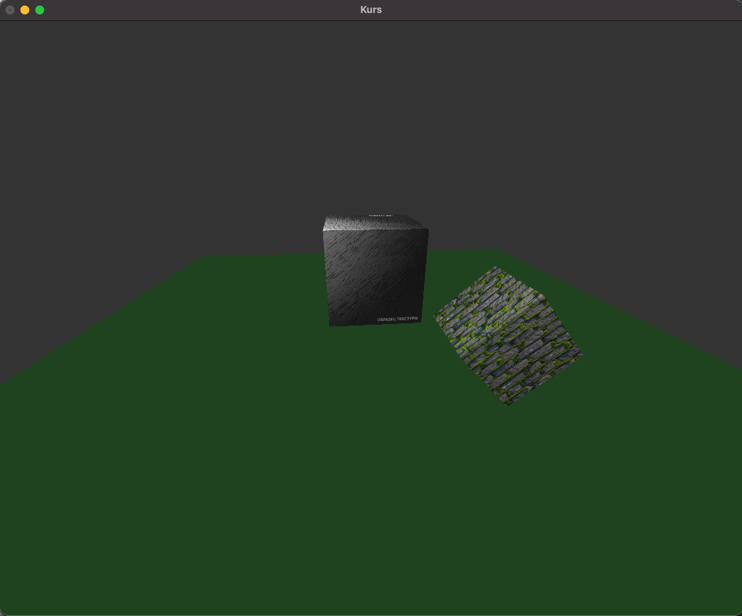
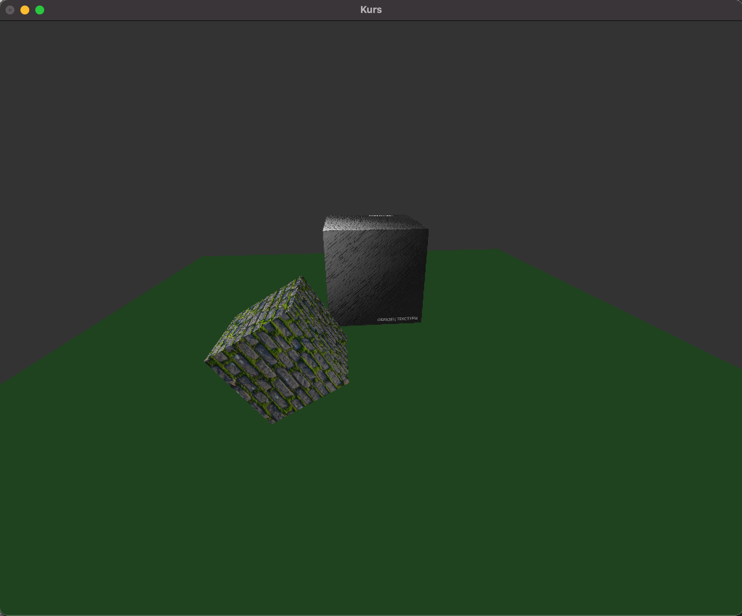
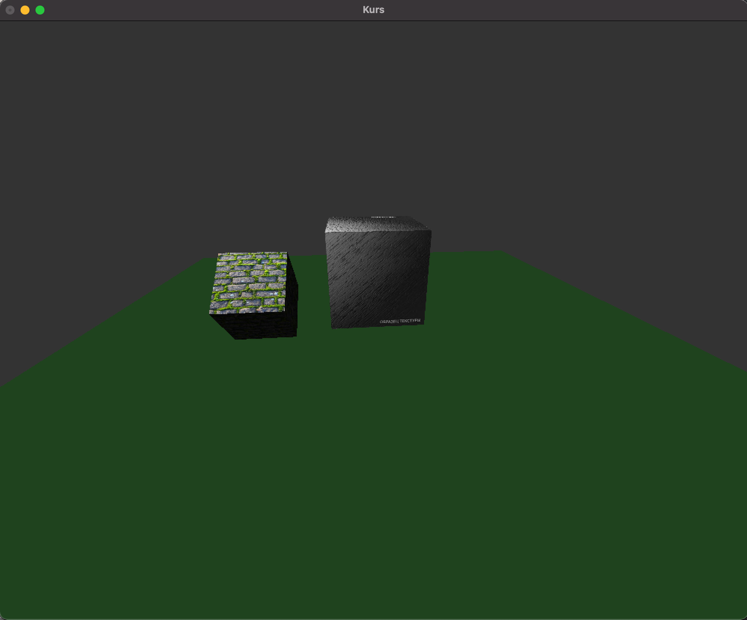
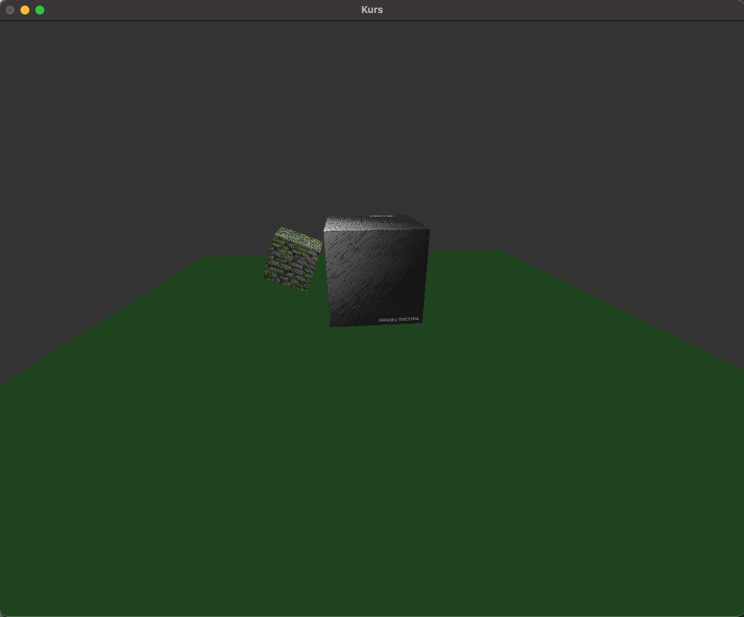
GL\_QUADS

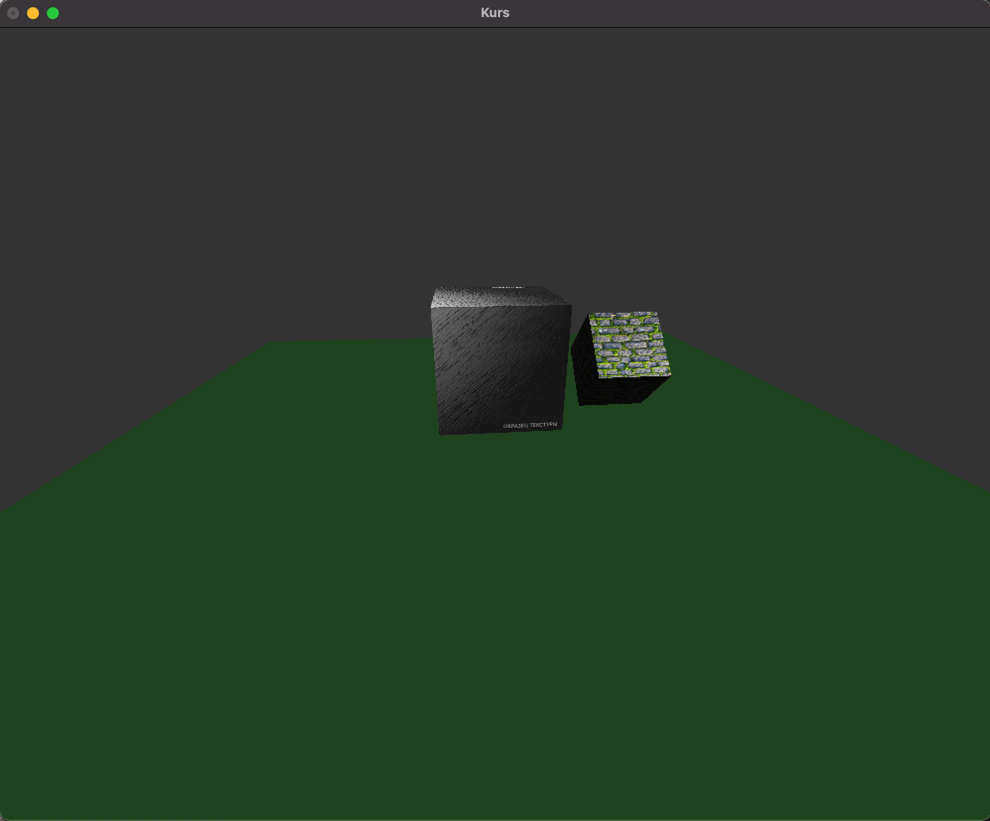
Для изменения угла исользуется функция glRotatef. Функция glTranslated используется для переноса точки поворота.

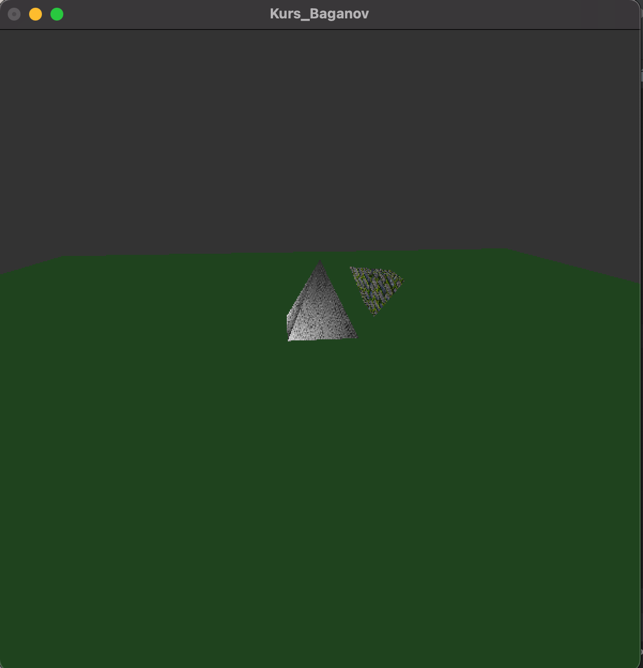
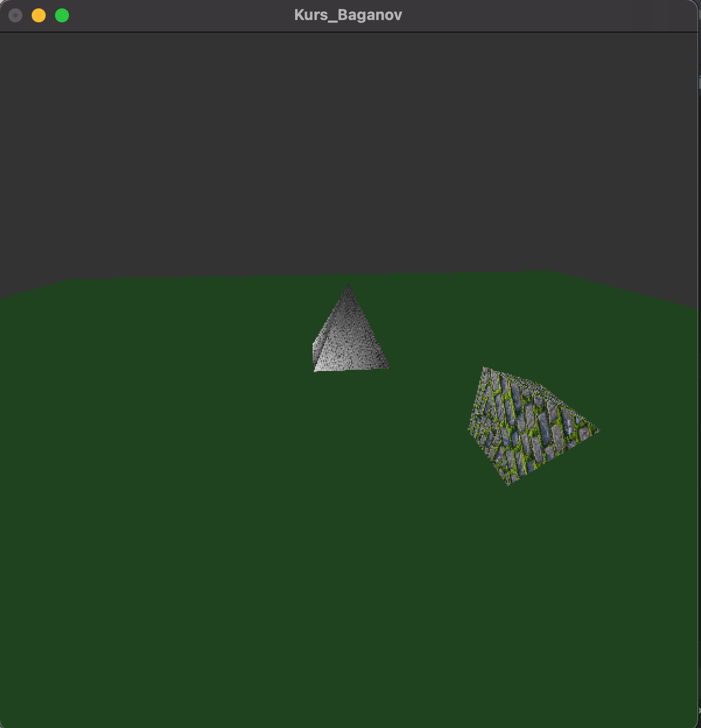
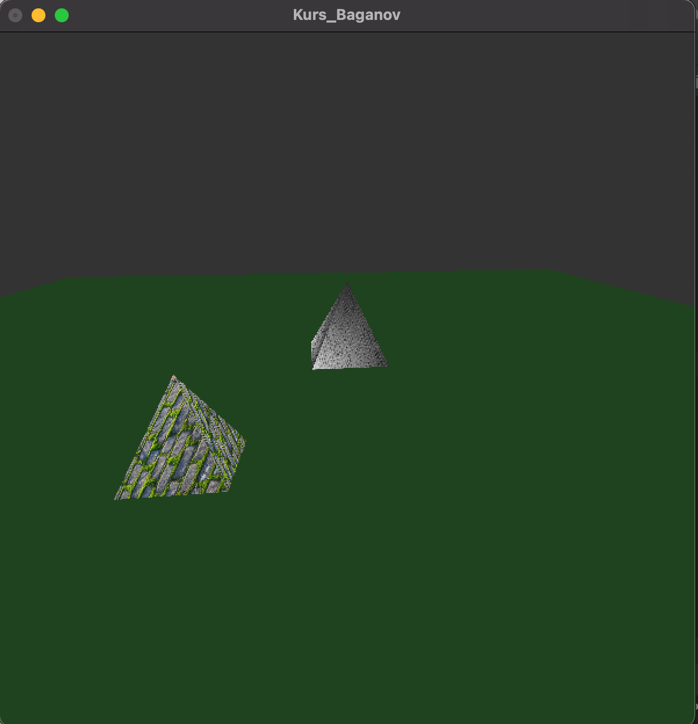
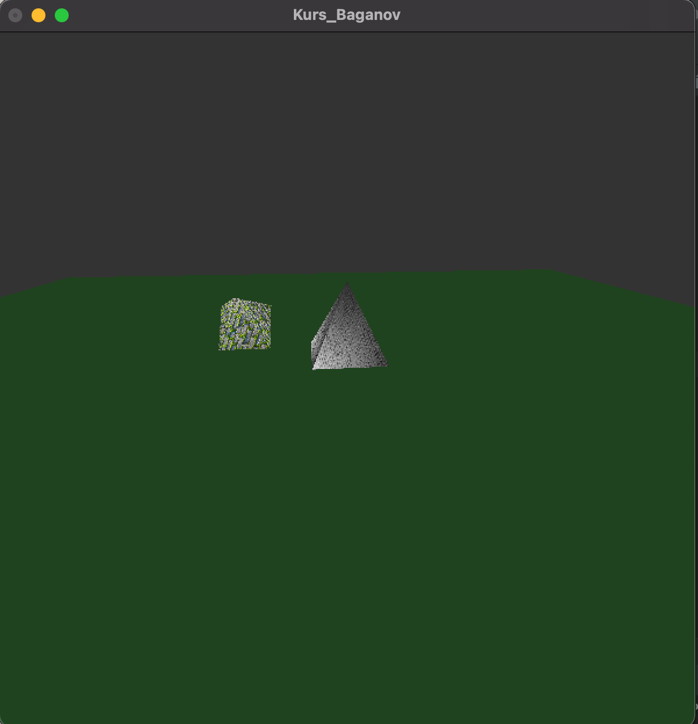
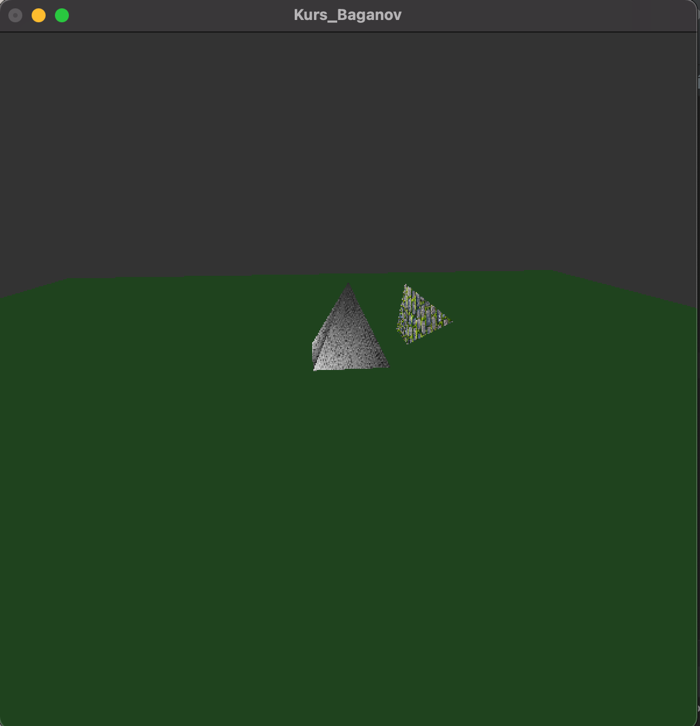
Константами x\_point\_y и x\_angle\_y, где x – это ось, а y – это номер точки, задаются координаты точки поворота, относительно центра двенадцатигранника и вектор поворота соответственно.

Функции glutVisibilityFunc() и glutReshapeFunc() нужны что бы перерисовывать объекты после изменений, а функция glutMainLoop() запускает цикл для того, что бы перерисовка происходила постоянно.

# СКРИНШОТЫ ПРИЛОЖЕНИЯ







# СПИСОК ЛИТЕРАТУРЫ

habr.com [Электронный ресурс]. – <https://habr.com/ru/post/246625/>

proglib.io [Электронный ресурс]. – <https://proglib.io/p/learn-opengl>

OpenGL Red Book (русская версия) [Электронный ресурс]. - <http://www.cosmic-rays.ru/books61/RedBook.pdf>

Ю.М. Баяковский, А.В. Игнатенко - Начальный курс OpenGL (DOC)"

[Электронный ресурс]. – <https://studizba.com/files/show/doc/221937-1-yu-m-bayakovskiy-a-v-ignatenko--nachal.html>

# ПРИЛОЖЕНИЕ

from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
from PIL import Image  
from numpy import array, int8  
  
sqrt\_05 = .5 \*\* .5  
  
xrot = 0  
yrot = 0  
zrot = 0  
  
none = (0, 0, 0, 1)  
light\_ambient = (.2, .2, .2, 1)  
light\_diffuse = array([1, 1, 1, 1])  
light\_specular = (1, 1, 1, 1)  
light\_position = [0, 1, .5, 0]  
  
ground = (.05, .2, .05, 1)  
  
polygon\_base = array([-1.5 - sqrt\_05 / 2, -.5, 2])  
edge\_translation = array([.5, -.5, 0])  
polygon\_angle = 0  
rotation\_angle = 0  
rotation\_angle\_step = 1  
rotation\_axis = (0, 0, 1)  
  
max\_angle = 90  
  
window = None  
cube\_texture = None  
polygon\_texture = None  
plane\_display\_list = None  
cube\_display\_list = None  
polygon\_display\_list = None  
  
  
def init():  
 global window  
 global cube\_texture  
 global polygon\_texture  
 global plane\_display\_list  
 global cube\_display\_list  
 global polygon\_display\_list  
  
 glutInit(sys.argv)  
  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH | GLUT\_RGBA)  
  
 glutInitWindowSize(1000, 800)  
 glutInitWindowPosition(50, 50)  
  
 window = glutCreateWindow(b"Kurs")  
  
 glutDisplayFunc(draw)  
 glutIdleFunc(roll)  
 glutKeyboardFunc(simple\_keys)  
 glutSpecialFunc(control\_keys)  
 glutReshapeFunc(change\_size)  
  
 glClearDepth(1)  
 glClearColor(\*light\_ambient)  
  
 glEnable(GL\_DEPTH\_TEST)  
 glEnable(GL\_NORMALIZE)  
  
 cube\_texture = load\_texture('test1.jpeg')  
 polygon\_texture = load\_texture('test3.jpeg')  
  
 plane\_display\_list = glGenLists(1)  
 glNewList(plane\_display\_list, GL\_COMPILE)  
 draw\_plane(10)  
 glEndList()  
  
 cube\_display\_list = glGenLists(1)  
 glNewList(cube\_display\_list, GL\_COMPILE)  
 draw\_cube(1.5)  
 glEndList()  
  
 polygon\_display\_list = glGenLists(1)  
 glNewList(polygon\_display\_list, GL\_COMPILE)  
 draw\_polygon(1)  
 glEndList()  
  
 glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, light\_ambient)  
 glEnable(GL\_LIGHTING)  
  
  
def load\_texture(filename):  
 img = Image.open(filename)  
 img\_data = array(list(img.getdata()), int8)  
  
 texture\_id = glGenTextures(1)  
 glBindTexture(GL\_TEXTURE\_2D, texture\_id)  
 glPixelStorei(GL\_UNPACK\_ALIGNMENT, 1)  
  
 glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)  
 glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)  
 glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)  
 glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR)  
 glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, img.size[0], img.size[1], 0, GL\_RGB, GL\_UNSIGNED\_BYTE, img\_data)  
  
 glBindTexture(GL\_TEXTURE\_2D, 0)  
  
 return texture\_id  
  
  
def draw\_plane(s):  
 s /= 2  
  
 glBegin(GL\_QUADS)  
 glNormal(0, 1, 0)  
 glVertex(s, 0, s)  
 glVertex(-s, 0, s)  
 glVertex(-s, 0, -s)  
 glVertex(s, 0, -s)  
 glEnd()  
  
  
def draw\_cube(s):  
 s /= 2  
  
 glBegin(GL\_QUADS)  
  
 # face  
 glNormal(0, 0, 1)  
 glTexCoord(0, 1)  
 glVertex(-s, -s, s)  
 glTexCoord(1, 1)  
 glVertex(s, -s, s)  
 glTexCoord(1, 0)  
 glVertex(s, s, s)  
 glTexCoord(0, 0)  
 glVertex(-s, s, s)  
  
 # back  
 glNormal(0, 0, -1)  
 glTexCoord(1, 1)  
 glVertex(s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(s, s, -s)  
 glTexCoord(0, 0)  
 glVertex(-s, s, -s)  
 glTexCoord(0, 1)  
 glVertex(-s, -s, -s)  
  
 # left  
 glNormal(-1, 0, 0)  
 glTexCoord(0, 0)  
 glVertex(-s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(-s, -s, s)  
 glTexCoord(1, 1)  
 glVertex(-s, s, s)  
 glTexCoord(0, 1)  
 glVertex(-s, s, -s)  
  
 # right  
 glNormal(1, 0, 0)  
 glTexCoord(0, 0)  
 glVertex(s, -s, s)  
 glTexCoord(0, 1)  
 glVertex(s, s, s)  
 glTexCoord(1, 1)  
 glVertex(s, s, -s)  
 glTexCoord(1, 0)  
 glVertex(s, -s, -s)  
  
 # top  
 glNormal(0, 1, 0)  
 glTexCoord(0, 0)  
 glVertex(-s, s, s)  
 glTexCoord(1, 0)  
 glVertex(s, s, s)  
 glTexCoord(1, 1)  
 glVertex(s, s, -s)  
 glTexCoord(0, 1)  
 glVertex(-s, s, -s)  
  
 # bottom  
 glNormal(0, -1, 0)  
 glTexCoord(0, 0)  
 glVertex(s, -s, s)  
 glTexCoord(0, 1)  
 glVertex(s, -s, -s)  
 glTexCoord(1, 1)  
 glVertex(-s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(-s, -s, s)  
  
 glEnd()  
  
  
  
  
  
 s /= 2  
  
 glBegin(GL\_QUADS)  
  
 # face  
 glNormal(0, 0, 1)  
 glTexCoord(0, 1)  
 glVertex(-s, -s, s)  
 glTexCoord(1, 1)  
 glVertex(s, -s, s)  
 glTexCoord(1, 0)  
 glVertex(s, s, s)  
 glTexCoord(0, 0)  
 glVertex(-s, s, s)  
  
 # back  
 glNormal(0, 0, -1)  
 glTexCoord(1, 1)  
 glVertex(s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(s, s, -s)  
 glTexCoord(0, 0)  
 glVertex(-s, s, -s)  
 glTexCoord(0, 1)  
 glVertex(-s, -s, -s)  
  
 # left  
 glNormal(-1, 0, 0)  
 glTexCoord(0, 0)  
 glVertex(-s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(-s, -s, s)  
 glTexCoord(1, 1)  
 glVertex(-s, s, s)  
 glTexCoord(0, 1)  
 glVertex(-s, s, -s)  
  
 # right  
 glNormal(1, 0, 0)  
 glTexCoord(0, 0)  
 glVertex(s, -s, s)  
 glTexCoord(0, 1)  
 glVertex(s, s, s)  
 glTexCoord(1, 1)  
 glVertex(s, s, -s)  
 glTexCoord(1, 0)  
 glVertex(s, -s, -s)  
  
 # top  
 glNormal(0, 1, 0)  
 glTexCoord(0, 0)  
 glVertex(-s, s, s)  
 glTexCoord(1, 0)  
 glVertex(s, s, s)  
 glTexCoord(1, 1)  
 glVertex(s, s, -s)  
 glTexCoord(0, 1)  
 glVertex(-s, s, -s)  
  
 # bottom  
 glNormal(0, -1, 0)  
 glTexCoord(0, 0)  
 glVertex(s, -s, s)  
 glTexCoord(0, 1)  
 glVertex(s, -s, -s)  
 glTexCoord(1, 1)  
 glVertex(-s, -s, -s)  
 glTexCoord(1, 0)  
 glVertex(-s, -s, s)  
  
 glEnd()  
  
  
def change\_size(width, height):  
 if height == 0:  
 height = 1  
  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
  
 glViewport(0, 0, width, height)  
 gluPerspective(90, float(width) / height, 0.1, 100)  
  
 glMatrixMode(GL\_MODELVIEW)  
  
  
def draw():  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glLoadIdentity()  
  
 gluLookAt(  
 -.5, 2.5, 5,  
 0, 0, 0,  
 0, 1, 0  
 )  
  
 glPushMatrix()  
 glRotate(xrot, 1, 0, 0)  
 glRotate(yrot, 0, 1, 0)  
 glRotate(zrot, 0, 0, 1)  
  
 glEnable(GL\_LIGHT0)  
  
 glLight(GL\_LIGHT0, GL\_AMBIENT, light\_ambient)  
 glLight(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse)  
 glLight(GL\_LIGHT0, GL\_SPECULAR, light\_specular)  
 glLight(GL\_LIGHT0, GL\_POSITION, light\_position)  
  
 glDisable(GL\_TEXTURE\_2D)  
  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, light\_ambient)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, ground)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, none)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_EMISSION, none)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 10)  
 glCallList(plane\_display\_list)  
  
 glEnable(GL\_TEXTURE\_2D)  
  
 glBindTexture(GL\_TEXTURE\_2D, cube\_texture)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, light\_ambient)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, light\_diffuse)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, light\_specular)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_EMISSION, none)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 50)  
 glTranslate(0, 1.001, 0)  
 glCallList(cube\_display\_list)  
  
 glTranslate(\*(polygon\_base + edge\_translation))  
 glRotate(-rotation\_angle, \*rotation\_axis)  
 glTranslate(\*(array(edge\_translation) \* -1))  
 glRotate(polygon\_angle, \*rotation\_axis)  
 glBindTexture(GL\_TEXTURE\_2D, polygon\_texture)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, light\_ambient)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, light\_diffuse)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, light\_specular)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_EMISSION, none)  
 glMaterial(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 40)  
 glCallList(polygon\_display\_list)  
  
 glBindTexture(GL\_TEXTURE\_2D, 0)  
  
 glPopMatrix()  
 glutSwapBuffers()  
  
  
def roll():  
 global polygon\_base #базовый полигон  
 global edge\_translation # край перевод  
 global rotation\_angle # угол поворота  
 global rotation\_angle\_step #/вращение\_угол\_шаг  
 global rotation\_axis #ось\_вращения  
 global polygon\_angle #polygon\_ угол  
 global max\_angle# макс\_ угол  
 # x\_direction / x\_направление  
  
 x\_direction = rotation\_axis == (0, 0, 1)  
  
 if rotation\_angle == max\_angle:  
 sign = rotation\_angle\_step / abs(rotation\_angle\_step)  
 offset = sign  
 polygon\_angle = (polygon\_angle - rotation\_angle) % (-sign \* 360)  
 edge\_translation = array([int(x\_direction) \* sign / 2, -.5, -int(not x\_direction) \* sign / 2])  
 rotation\_angle = 0  
 max\_angle = sign \* 90  
  
 if x\_direction:  
 if polygon\_angle == -90 or polygon\_angle == 180:  
 edge\_translation = array([0, -.5, 0])  
 max\_angle /= 2  
 elif polygon\_angle == -135 or polygon\_angle == 225:  
 rotation\_angle = -sign \* 45  
 polygon\_angle = (polygon\_angle + rotation\_angle) % (-sign \* 360)  
 edge\_translation = array([0, -.5, 0])  
 max\_angle = 0  
 offset \*= sqrt\_05  
 elif polygon\_angle == -180 or polygon\_angle == 270:  
 offset = 0  
 else:  
 offset \*= -1  
  
 polygon\_base = polygon\_base + array(rotation\_axis[::-1]) \* offset  
  
 if (abs(polygon\_base[0]) > 1.5 and x\_direction) or (abs(polygon\_base[2]) == 2 and not x\_direction):  
 rotation\_axis = rotation\_axis[::-1]  
 sign = -1  
 if polygon\_base[0] > 0 > polygon\_base[2] or polygon\_base[2] > 0 > polygon\_base[0]:  
 rotation\_angle\_step \*= -1  
 max\_angle \*= -1  
 sign = 1  
 edge\_translation[0], edge\_translation[2] = (v \* sign for v in (edge\_translation[2], edge\_translation[0]))  
  
 rotation\_angle\_step /= abs(rotation\_angle\_step)  
  
 if rotation\_angle\_step > 0:  
 rotation\_angle = min(rotation\_angle + rotation\_angle\_step, max\_angle)  
 else:  
 rotation\_angle = max(rotation\_angle + rotation\_angle\_step, max\_angle)  
  
 # fall acceleration  
 # if abs(max\_angle) == 45 and abs(rotation\_angle) > 15:  
 # inc = (rotation\_angle / 15 - rotation\_angle / abs(rotation\_angle)) / 10  
 # if abs(rotation\_angle) > 30:  
 # inc \*= 2  
 # rotation\_angle\_step += inc  
 # elif abs(rotation\_angle) > 45:  
 # inc = (rotation\_angle / 45 - rotation\_angle / abs(rotation\_angle)) / 10  
 # if abs(rotation\_angle) > 60:  
 # inc \*= 2  
 # rotation\_angle\_step += inc  
  
 draw()  
  
  
def simple\_keys(key, x, y):  
 global window  
 global light\_position  
 global light\_diffuse  
  
 pressed = int.from\_bytes(key, 'big')  
  
 if pressed == 27:  
 glutDestroyWindow(window)  
  
 if pressed == ord('a'):  
 light\_position[0] -= .2  
 if pressed == ord('d'):  
 light\_position[0] += .2  
 if pressed == ord('w'):  
 light\_position[2] -= .2  
 if pressed == ord('s'):  
 light\_position[2] += .2  
 if pressed == ord('['):  
 light\_diffuse = light\_diffuse + array([-.1, -.1, -.1, 0])  
 if pressed == ord(']'):  
 light\_diffuse = light\_diffuse + array([.1, .1, .1, 0])  
  
  
def control\_keys(key, x, y):  
 global xrot  
 global yrot  
 global zrot  
  
 if key == GLUT\_KEY\_UP:  
 xrot += 2.0  
 if key == GLUT\_KEY\_DOWN:  
 xrot -= 2.0  
  
 if glutGetModifiers() & GLUT\_ACTIVE\_SHIFT:  
 if key == GLUT\_KEY\_LEFT:  
 zrot -= 2.0  
 if key == GLUT\_KEY\_RIGHT:  
 zrot += 2.0  
 else:  
 if key == GLUT\_KEY\_LEFT:  
 yrot -= 2.0  
 if key == GLUT\_KEY\_RIGHT:  
 yrot += 2.0  
  
 glutPostRedisplay()  
  
  
init()  
glutMainLoop()