Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий  
Кафедра прикладной математики

Отчет защищен с оценкой \_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Отчет

По лабораторной работе №1

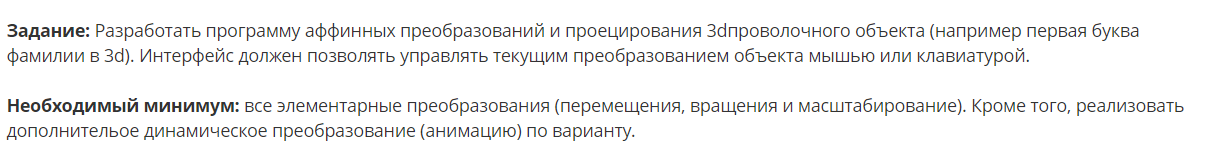
«Афинные преобразования»

по дисциплине «Компьютерная графика»

Студент группы ПИ-02 Чередов Р.А.

Преподаватель Андреева А.Ю.

Барнаул 2023

Задание: Вариант 8  


Анимация: Прыгающий объект (например, по нажатию клавиши пробел и с учетом законов физики, те. с замедлением).

Код:

Main.py

import math  
import pygame as pg  
from object\_3d import \*  
from projection import \*  
from camera import \*  
  
  
class SoftwareRender:  
 def \_\_init\_\_(self): #инициализация окна  
 pg.init()  
 self.RES = self.WIDTH, self.HEIGHT = 1280, 720  
 self.H\_WIDTH, self.H\_HEIGHT = self.WIDTH // 2, self.HEIGHT // 2  
 self.FPS = 60  
 self.screen = pg.display.set\_mode(self.RES)  
 self.clock = pg.time.Clock()  
 self.create\_objects()  
  
 def create\_objects(self):  
 self.camera = Camera(self,[0.5,1,-4])  
 self.projection = Projection(self)  
 self.object = Object3D(self)  
 self.object.translate([0.2, 0.4, 0.2])  
 self.object.rotate\_y(math.pi / 6)  
  
 def draw(self): #заливка экрана  
 self.screen.fill(pg.Color('darkcyan'))  
 self.object.draw()  
 def run(self): #запуск экрана  
 while True:  
 self.draw()  
 self.camera.control()  
 [exit() for i in pg.event.get() if i.type == pg.QUIT]  
 pg.display.set\_caption(str(self.clock.get\_fps()))  
 pg.display.flip()  
 self.clock.tick(self.FPS)  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = SoftwareRender()  
 app.run()

matrix\_functions

import math  
import numpy as np  
def translate(pos):  
 tx, ty, tz = pos  
 return np.array([  
 [1, 0, 0, 0],  
 [0, 1, 0, 0],  
 [0, 0, 1, 0],  
 [tx, ty, tz, 1]  
 ])  
def rotate\_x(a):  
 return np.array([  
 [1, 0, 0, 0],  
 [0, math.cos(a), math.sin(a), 0],  
 [0, -math.sin(a), math.cos(a), 0],  
 [0, 0, 0, 1]  
 ])  
def rotate\_y(a):  
 return np.array([  
 [math.cos(a), 0, -math.sin(a), 0],  
 [0, 1, 0, 0],  
 [math.sin(a), 0, math.cos(a), 0],  
 [0, 0, 0, 1]  
 ])  
def rotate\_z(a):  
 return np.array([  
 [math.cos(a), math.sin(a), 0, 0],  
 [-math.sin(a), math.cos(a), 0, 0],  
 [0, 0, 1, 0],  
 [0, 0, 0, 1]  
 ])  
def scale(n):  
 return np.array([  
 [n, 0, 0, 0],  
 [0, n, 0, 0],  
 [0, 0, n, 0],  
 [0, 0, 0, 1]  
 ])

object\_3d

import numpy as np  
import pygame as pg  
from matrix\_functions import \*  
class Object3D:  
 def \_\_init\_\_(self, render):  
 self.render = render  
 self.vertexes = np.array([(0,0,0,1),(0,1,0,1),(1,1,0,1),(1,0,0,1),(0,0,1,1),(0,1,1,1),(1,1,1,1),(1,0,1,1)])  
  
 self.faces = np.array([(0,1,2,3),(4,5,6,7),(0,4,5,1),(2,3,7,6),(1,2,6,5),(0,3,7,4)])  
  
 def draw(self):  
 self.screen\_projection()  
 key = pg.key.get\_pressed()  
 if key[pg.K\_0]: self.movement()  
 def movement(self):  
 self.rotate\_z(pg.time.get\_ticks() % 0.001)  
  
  
  
  
  
 def translate(self,pos):  
 self.vertexes = self.vertexes @ translate(pos)  
 def scale(self,scale\_to):  
 self.vertexes = self.vertexes @ scale(scale\_to)  
 def rotate\_x(self,angle):  
 self.vertexes = self.vertexes @ rotate\_x(angle)  
 def rotate\_y(self,angle):  
 self.vertexes = self.vertexes @ rotate\_y(angle)  
 def rotate\_z(self,angle):  
 self.vertexes = self.vertexes @ rotate\_z(angle)  
 def screen\_projection(self):  
 vertexes = self.vertexes @ self.render.camera.camera\_matrix()  
 vertexes = vertexes @ self.render.projection.projection\_matrix  
 vertexes /= vertexes[:, -1].reshape(-1, 1)  
 vertexes[(vertexes > 2) | (vertexes < -2)] = 0  
 vertexes = vertexes @ self.render.projection.to\_screen\_matrix  
 vertexes = vertexes[:, :2]  
 for face in self.faces:  
 polygon = vertexes[face]  
 if not np.any((polygon == self.render.H\_WIDTH) | (polygon == self.render.H\_HEIGHT)):  
 pg.draw.polygon(self.render.screen,pg.Color('orange'),polygon,3)  
  
 for vertexes in vertexes:  
 if not np.any((vertexes == self.render.H\_WIDTH) | (vertexes == self.render.H\_HEIGHT)):  
 pg.draw.circle(self.render.screen, pg.Color('white'), vertexes , 6)

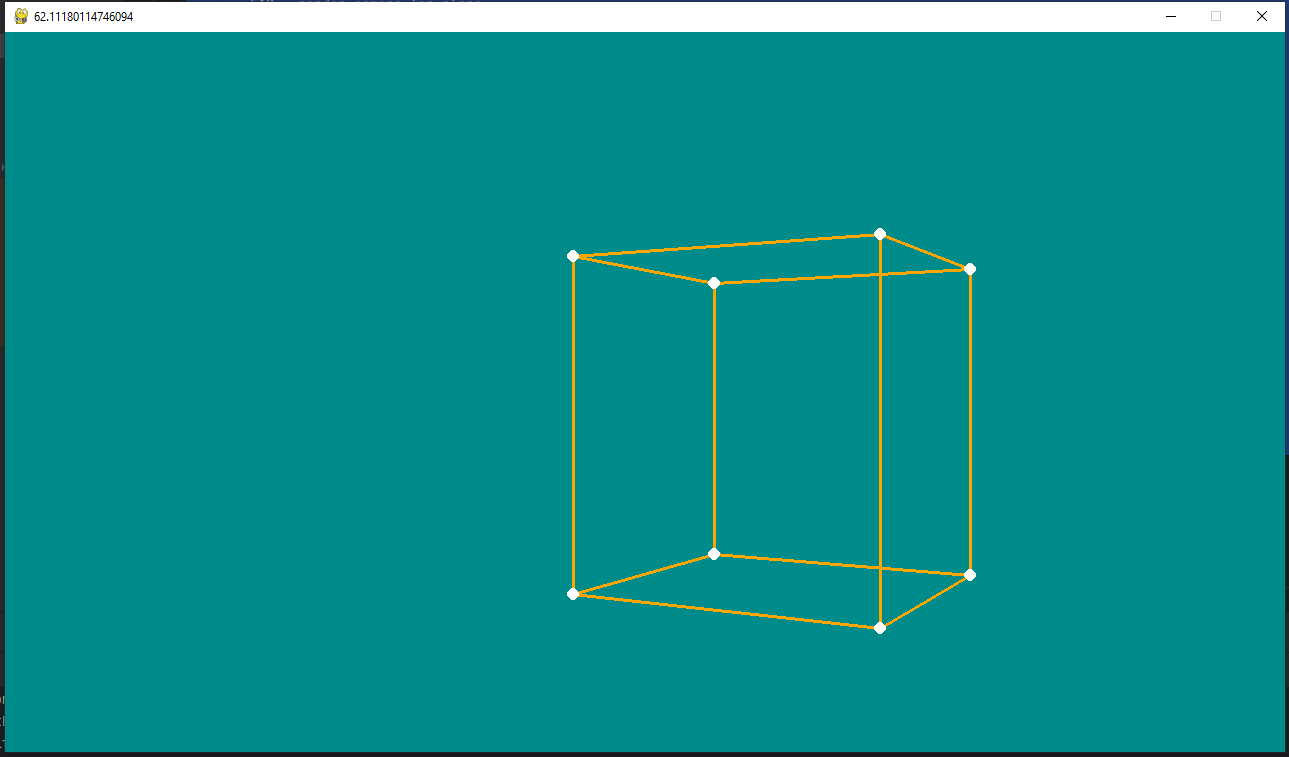
Camera.py

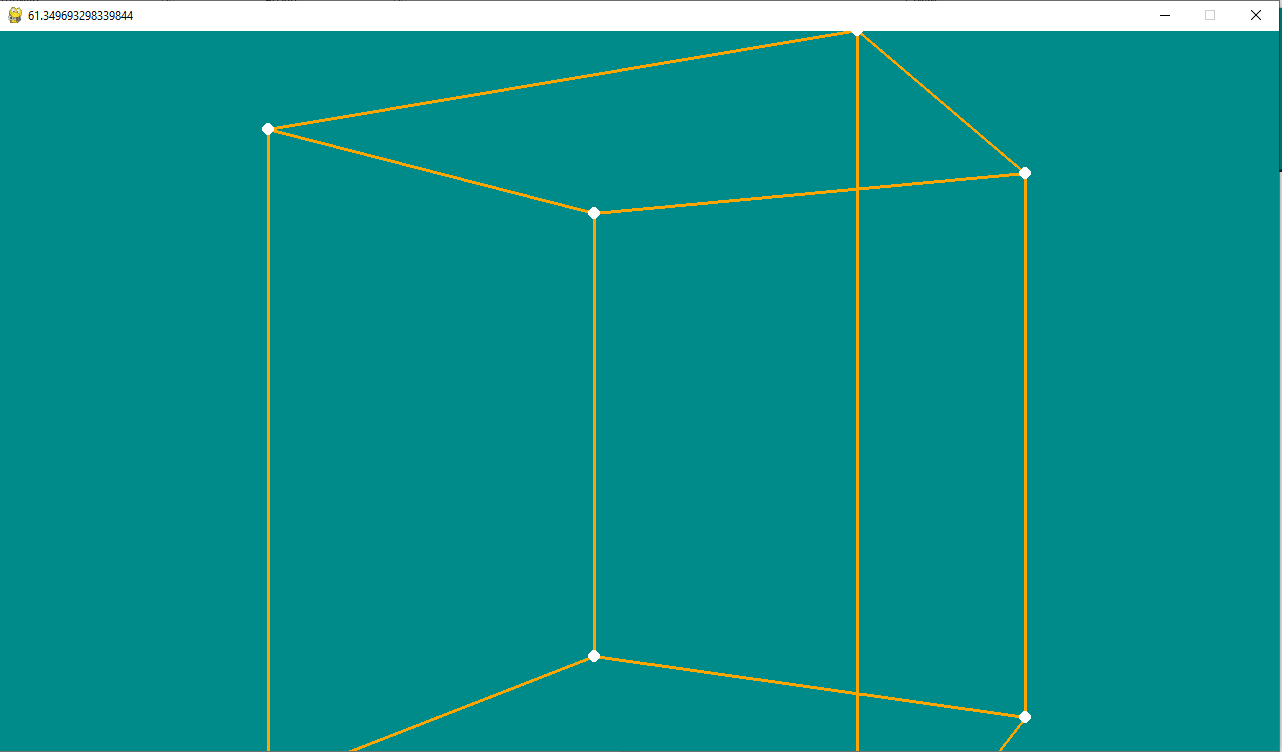
import numpy as np  
import pygame as pg  
from matrix\_functions import \*  
class Object3D:  
 def \_\_init\_\_(self, render):  
 self.render = render  
 self.vertexes = np.array([(0,0,0,1),(0,1,0,1),(1,1,0,1),(1,0,0,1),(0,0,1,1),(0,1,1,1),(1,1,1,1),(1,0,1,1)])  
  
 self.faces = np.array([(0,1,2,3),(4,5,6,7),(0,4,5,1),(2,3,7,6),(1,2,6,5),(0,3,7,4)])  
  
 def draw(self):  
 self.screen\_projection()  
 key = pg.key.get\_pressed()  
 if key[pg.K\_0]: self.movement()  
 def movement(self):  
 self.rotate\_z(pg.time.get\_ticks() % 0.001)  
  
  
  
  
  
 def translate(self,pos):  
 self.vertexes = self.vertexes @ translate(pos)  
 def scale(self,scale\_to):  
 self.vertexes = self.vertexes @ scale(scale\_to)  
 def rotate\_x(self,angle):  
 self.vertexes = self.vertexes @ rotate\_x(angle)  
 def rotate\_y(self,angle):  
 self.vertexes = self.vertexes @ rotate\_y(angle)  
 def rotate\_z(self,angle):  
 self.vertexes = self.vertexes @ rotate\_z(angle)  
 def screen\_projection(self):  
 vertexes = self.vertexes @ self.render.camera.camera\_matrix()  
 vertexes = vertexes @ self.render.projection.projection\_matrix  
 vertexes /= vertexes[:, -1].reshape(-1, 1)  
 vertexes[(vertexes > 2) | (vertexes < -2)] = 0  
 vertexes = vertexes @ self.render.projection.to\_screen\_matrix  
 vertexes = vertexes[:, :2]  
 for face in self.faces:  
 polygon = vertexes[face]  
 if not np.any((polygon == self.render.H\_WIDTH) | (polygon == self.render.H\_HEIGHT)):  
 pg.draw.polygon(self.render.screen,pg.Color('orange'),polygon,3)  
  
 for vertexes in vertexes:  
 if not np.any((vertexes == self.render.H\_WIDTH) | (vertexes == self.render.H\_HEIGHT)):  
 pg.draw.circle(self.render.screen, pg.Color('white'), vertexes , 6)

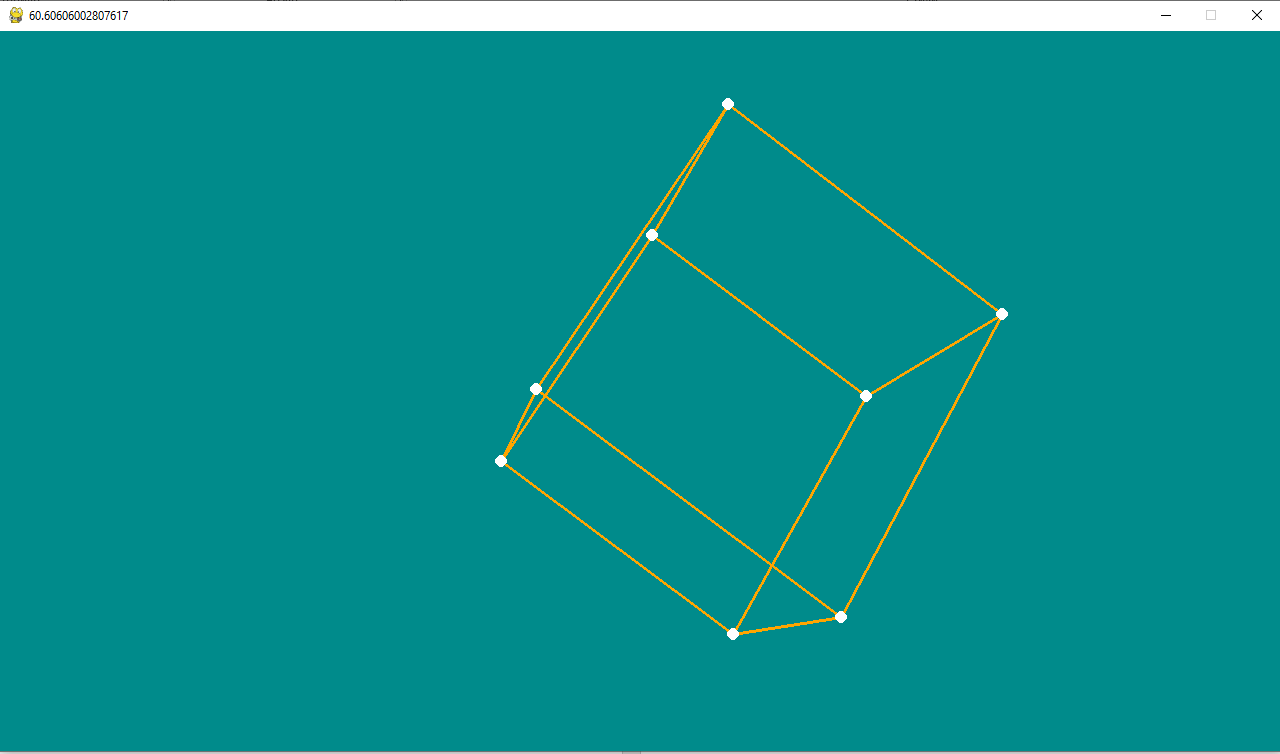
projection.py

import math  
import numpy as np  
class Projection:  
 def \_\_init\_\_(self, render):  
 NEAR = render.camera.near\_plane  
 FAR = render.camera.far\_plane  
 RIGHT = math.tan(render.camera.h\_fov / 2)  
 LEFT = -RIGHT  
 TOP = math.tan(render.camera.v\_fov / 2)  
 BOTTOM = -TOP  
 m00 = 2 / (RIGHT - LEFT)  
 m11 = 2 / (TOP - BOTTOM)  
 m22 = (FAR + NEAR) / (FAR - NEAR)  
 m32 = -2 \* NEAR \* FAR / (FAR - NEAR)  
 self.projection\_matrix = np.array([  
 [m00, 0, 0, 0],  
 [0, m11, 0, 0],  
 [0, 0, m22, 1],  
 [0, 0, m32, 0]  
 ])  
 HW, HH = render.H\_WIDTH, render.H\_HEIGHT  
 self.to\_screen\_matrix = np.array([  
 [HW, 0, 0, 0],  
 [0, -HH, 0, 0],  
 [0, 0, 1, 0],  
 [HW, HH, 0, 1]  
 ])

Тесты:







По пробелу проигрывается анимация