Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Чувашский государственный педагогический университет

им. И.Я. Яковлева»

Физико-математический факультет

Информатики и информационно-коммуникационных технологий

КУРСОВАЯ РАБОТА

Создание ПО для просмотра 3D объектов на языке программирования Python

с использованием framework Numpy и Pygame.

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Григорьев Ю.В.

подпись, дата

Студент ПИ-2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ Пигальцев Т.Е.

зачетной книжки подпись дата инициалы

Чебоксары2021

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc74692924)

[1.ОСНОВНЫЕ ВОЗМОЖНОСТИ ГРАФИЧЕСКИХ СИСТЕМ 3D-ДВИЖКОВ 5](#_Toc74692925)

[1.1. Точки, векторы и базовый принцип трехмерного пространства 5](#_Toc74692926)

[1.2. Основы линейных преобразований 7](#_Toc74692927)

[2. ОСНОВЫ БИБЛИОТЕК NUMPY И PYGAME 11](#_Toc74692928)

[2.1. Обоснование выбора numpy и pygame 11](#_Toc74692929)

[2.2 Массивы в numpy 11](#_Toc74692930)

[2.3. Основные возможности модуля pygame 12](#_Toc74692931)

[3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ ПРОСМОТРА 3D ОБЪЕКТОВ 16](#_Toc74692932)

[3.1 Модуль настройки основных параметров 16](#_Toc74692933)

[3.2 Модуль формирования матрицы функций, зависящий от входных параметров 18](#_Toc74692934)

[3.3 Метод формирования вершин массива и отрисовки полигонов 20](#_Toc74692935)

[3.4 Метод построение и управление камеры 23](#_Toc74692936)

[3.5 Метод формирование матрицы проекций 26](#_Toc74692937)

[ЗАКЛЮЧЕНИЕ 28](#_Toc74692938)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 29](#_Toc74692939)

[ПРИЛОЖЕНИЕ А 31](#_Toc74692940)

[ПРИЛОЖЕНИЕ Б 32](#_Toc74692941)

[ПРИЛОЖЕНИЕ В 33](#_Toc74692942)

[ПРИЛОЖЕНИЕ Г 35](#_Toc74692943)

[ПРИЛОЖЕНИЕ Д 37](#_Toc74692944)

# ВВЕДЕНИЕ

Представление данных на мониторе компьютера в графическом виде впервые было реализовано в середине 50-х годов для больших ЭВМ, применявшихся в научных и военных исследованиях. С тех пор графический способ отображения данных стал неотъемлемой частью подавляющего числа компьютерных систем, в особенности персональных.

Компьютерная графика — это специальная область информатики, изучающая методы и средства создания и обработки изображений с помощью программно-аппаратных вычислительных комплексов. Она охватывает все виды и формы представления изображений, доступных для восприятия человеком либо на экране монитора, либо в виде копии на внешнем носителе (бумага, киноплёнка, ткань и прочее).

Развитие технологий привело к быстрому росту в области компьютерной техники и программного обеспечения. Обычным явление благодаря массовому распространению программного обеспечения по созданию компьютерной графики и трёхмерного моделирования. Программы трёхмерной графики - самый интересный по своим возможностям, но сложны в освоениях приложения.

Трёхмерная графика уже прочно вошла в нашу жизнь и не замечаем её. Разглядывая рекламный щит, играя в игру, в которой взрывается машина это результат мастера трёхмерной графики. Область применения трёхмерной графики необычайно широк начиная от: рекламы, киноиндустрий и дизайна интерьера и производства компьютерных игр.

Использование компьютерных технологий при проектировании и разработке дизайна интерьера помогает увидеть конечный вариант задолго до того, как обстановка будет воссоздана. Трехмерная графика позволяет создавать трехмерные макеты различных объектов (кресел, диванов, стульев и т.д.), повторяя их геометрическую форму и имитируя материал, из которого они созданы. Чтобы получить полное представление об определенном объекте, необходимо осмотреть его со всех сторон, с разных точек, при различном освещении. Трехмерная графика позволяет создать демонстрационный ролик, в котором будет запечатлена виртуальная прогулка по этажам будущего коттеджа, который только начинает строиться.

Объект исследования: трехмерная графика

Предмет исcледования: создание ПО для просмотра 3D объектов

Цель курсовой работы: создание ПО для просмотра 3D объектов

Для достижения этой цели, необходимо решить следующие задачи:

- рассмотреть основные элементы и базовый принцип трехмерного пространства;

- изучить основы линейных преобразований трехмерного пространства;

- обосновать выбор библиотек numpy и pygame для реализации просмотра 3D;

- рассмотреть возможности применения массивов в numpy;

- изучить основные возможности модуля pygame;

- разработать модуль настройки основных параметров;

- разработать модуль формирования матрицы функций, зависящий от входных параметров;

- разработать модуль формирования вершин массива и отрисовки полигонов;

- разработать модуль построение и управление камеры;

- разработать модуль формирование матрицы проекций;

# 1.ОСНОВНЫЕ ВОЗМОЖНОСТИ ГРАФИЧЕСКИХ СИСТЕМ 3D-ДВИЖКОВ

## 1.1. Точки, векторы и базовый принцип трехмерного пространства

Для трёхмерной графики требуется концепция трёхмерного пространства. Наиболее часто из всех видов пространств используется декартово пространство, которое позволяет нам применять декартовы координаты (стандартная запись (x, y) и двухмерные графики).

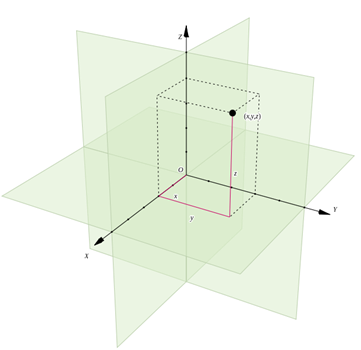
Трёхмерное декартово пространство даёт нам оси x, y и z (описывающие положение по горизонтали, вертикали и в глубину). Координаты любой точки в этом пространстве обозначаются как несколько чисел (в нашем случае три числа, потому что у нас три оси). На двухмерной плоскости запись обозначается как (x, y), а в трёхмерном пространстве — как (x, y, z). Эта запись (кортеж) показывает положение точки относительно исходной точки пространства (которая обычно обозначается как (0, 0, 0). Внешний вид трёхмерного декоратора смотрите на рис. 1.1.

Рис. 1.1

Каждый из элементов кортежа является скалярным числом, определяющим положение вдоль базисного вектора. Каждый базисный вектор должен иметь единичную длину (его длина равна 1), то есть такие кортежи, как (1, 1, 1) и (2, 2, 2) не могут быть базисными векторами, потому что они слишком длинные.

Мы определим три базисных вектора в нашем пространстве:

(1.1)

Точку начала координат системы координат можно обозначить точкой O, которая описывается кортежем из трёх элементов (0,0,0). Это значит, что математическое представление системы координат можно изобразить так:

(1.2)

Этой записью мы можем сказать, что (x, y, z) представляют собой положение точки относительно начала координат. Такое определение означает, что любую точку P, (a, b, c) можно представить как:

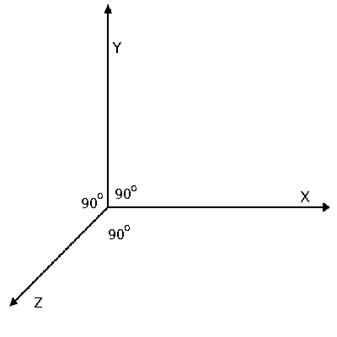
(1.3)

где a, b и c – это скаляры;

X, Y, Z – это базисные векторы;

(1.4)

Система координат, которую используем, обладает очень ценным свойством: взаимной перпендикулярностью. Это значит, что в пересечении каждой из осей на своей соответствующей плоскости угол между ними равен 90 градусам. Внешний вид системы координат взаимной перпендикулярностью смотрите на рис. 1.2.

Рис. 1.2

Векторное произведение можно определить следующим уравнением (при условии, что у нас есть два кортежа из трёх элементов):

(1.5)

Аксиома — это логическое утверждение, считаемое достаточно очевидным для принятия без доказательств.

При работе с точками и векторами мы будем использовать две основные аксиомы.

Аксиома 1: разность между двумя точками — это вектор, то есть

(1.6)

Аксиома 2: сумма точки и вектора — это точка, то есть

(1.7)

## 1.2. Основы линейных преобразований

Линейные преобразования имеют следующую форму:

(1.8)

Есть функция преобразования F (), в качестве входных данных используется вектор A, а на выходе мы получим вектор B

Каждую из этих частей — два вектора и функцию — можно представить в виде матрицы: вектор B — как матрицу 1x3, вектор A — как ещё одну матрицу 1x3, а линейное преобразование F () — как матрицу 3x3 (матрицу преобразований).

(1.9)

(1.10)

Поворот, по самому определению — это круговое движение объекта вокруг точки поворота. Точка поворота в нашем пространстве может принадлежать плоскости XY, плоскости XZ или плоскости YZ где каждая плоскость составлена из двух базисных векторов.

Три точки поворота означают, что у нас есть три отдельных матрицы вращения:

Матрица поворота по XY:

(1.11)

Матрица поворота по XZ:

(1.12)

Матрица поворота по YZ:

(1.13)

Для поворота точки AA вокруг плоскости XY на 90 градусов (π/2 радиан — в большинстве математических библиотек есть функция преобразования градусов в радианы.

(1.14)

То есть если начальная точка A имела координаты (3,4,5), то выходная точка B будет иметь координаты (−4,3,5).

Масштабирование — это преобразование, увеличивающее или уменьшающее объект в соответствии с заданным масштабом.

Преобразование масштабирования требует двух типов входных данных: входного вектора и кортежа масштабирования из трёх элементов, который определяет масштаб входного вектора по каждой из осей пространства.

Матрица масштабного преобразования имеет следующий вид (где , и — это элементы кортежа масштабирования):

(1.15)

Входной вектор A (a0, a1, a2) вдвое больше по оси X (используя кортеж S = (2, 1, 1), вычисления должны иметь следующий вид:

(1.16)

При входном векторе A = (3, 4, 0) выходной вектор BB будет равен (6, 4, 0).

Отсечение — это выборка объектов из большей группы объектов. В трехмерном движке меньшей группой будут точки, которые нужно отрисовать на экране. Большей группой объектов будет множество всех существующих точек.

Благодаря отсечению движок значительно снизит потребление памяти системы. Он будет отрисовывать только то, что камера может увидеть, а не весь мир точек.

Пространство обзора будет задаваться по всем трём традиционным осям: x, y и z. Границы по x состоят из всего между левой и правой границами окна, границы по y — из всего между верхней и нижней границами окна, а границы по z находятся в пределах от 0 куда установлена камера до расстояния видимости камеры мы будем использовать произвольно выбранное значение 100.

Перед отрисовкой точки класс камеры будет проверять, находится ли точка в пространстве обзора. Если находится, то точка отрисовывается, в противном случае — не отрисовывается.

# 2. ОСНОВЫ БИБЛИОТЕК NUMPY И PYGAME

## 2.1. Обоснование выбора numpy и pygame

Numpy — это open-source модуль для python, который предоставляет общие математические и числовые операции в виде пре-скомпилированных, быстрых функций. Они объединяются в высокоуровневые пакеты. Они обеспечивают функционал, который можно сравнить с функционалом MatLab. Numpy (Numeric Python) предоставляет базовые методы для манипуляции с большими массивами и матрицами. SciPy (Scientific Python) расширяет функционал numpy огромной коллекцией полезных алгоритмов, таких как минимизация, преобразование Фурье, регрессия, и другие прикладные математические техники.

Данная библиотека nympy позволяет намного быстрее вычислять большие математический алгоритмы что позволило улучшить обработку отрисовки векторов

Pygame — набор модулей (библиотек) языка программирования Python, предназначенный для написания компьютерных игр и мультимедиаприложений. Pygame базируется на мультимедийной библиотеке SDL. В то же время, с точки зрения классификации программного обеспечения, Pygame является API для Питона к API библиотеки SDL.

Данная библиотека pygame позволяет создать приложение на python с интерфейсом и визуализировать отрисовку векторов, что позволило сделать интерфейс программы и создать приложение под windows.

## 2.2 Массивы в numpy

import numpy

Тем не менее, для большого количества вызовов функций numpy, становиться утомительно писать numpy. X снова и снова. Это выражение позволяет нам получать доступ к numpy объектам используя np. X вместо numpy. X.

import numpy as np

Главной особенностью numpy является объект array. Массивы схожи со списками в python, исключая тот факт, что элементы массива должны иметь одинаковый тип данных, как float и int. С массивами можно проводить числовые операции с большим объемом информации в разы быстрее и, главное, намного эффективнее чем со списками.

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]

])

## 2.3. Основные возможности модуля pygame

import pygame

Для большого количества вызовов функций pygame, становиться утомительно писать pygame. X снова и снова. Это выражение позволяет нам получать доступ к pygame объектам используя pg. X вместо pygame. X.

import pygame as pg

Таблица 1 –Основные модули

|  |  |
| --- | --- |
| **Модуль** | **Назначение** |
| pygame.cdrom | Доступ к CD-приводам и управление ими |
| pygame.cursors | Загружает изображения курсора |
| pygame.display | Доступ к дисплею |
| pygame.draw | Рисует фигуры, линии и точки |
| pygame.event | Управление внешними событиями |
| pygame.font | Использует системные шрифты |
| pygame.image | Загружает и сохраняет изображение |
| pygame.joystick | Использует джойстики и аналогичные устройства |
| pygame.key | Считывает нажатия клавиш с клавиатуры |
| pygame.mixer | Загружает и воспроизводит мелодии |
| pygame.mouse | Управляет мышью |
| pygame.movie | Воспроизведение видеофайлов |
| pygame.music | Работает с музыкой и потоковым аудио |
| pygame.overlay | Доступ к расширенным видеоизображениям |
| pygame | Содержит функции Pygame высокого уровня |
| pygame.rect | Управляет прямоугольными областями |
| pygame.sndarray | Манипулирует звуковыми данными |
| pygame.sprite | Управление движущимися изображениями |
| pygame.surface | Управляет изображениями и экраном |
| pygame.surfarray | Манипулирует данными пикселей изображения |
| pygame.time | Управления временем и частотой кадров |
| pygame.transform | Изменение размера и перемещение изображений |

Модуль pygame.key –этот модуль содержит функции для работы с клавиатурой. Очередь событий получает события pygame.KEYDOWN и pygame.KEYUP при нажатии и отпускании клавиш клавиатуры.

Оба события имеют ключевой атрибут, который представляет собой целочисленный идентификатор, представляющий каждую клавишу на клавиатуре.Событие pygame.KEYDOWN имеет дополнительные атрибуты: unicode и scancode. unicode представляет собой одну символьную строку, которая соответствует введённому символу.

key = pg.key.get\_pressed()

if key[pg.K\_a]:

self.position -= self.right \* self.moving\_speed

Модуль pygame.time содержит объект Clock, который можно использовать для отслеживания

времени. Чтобы создать объект типа: время, вызывается конструктор pygame.time.Clock:

clock = pygame.time.Clock()

Когда создан объект clock, можно вызвать его функцию tick один раз за кадр, которая возвращает время, прошедшее со времени предыдущего вызова в миллисекундах:

time\_passed = clock.tick ()

Функция tick может использовать необязательный параметр для установления максимальной частоты кадров. Этот параметр нужен, если игра запущена на рабочем компьютере и необходимо контролировать, чтобы она не использовала всю его вычислительная мощность на 100%:

time\_passed = clock.tick (30)

Игра будет работать со скоростью не более 30 кадров в секунду.

# 3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА ДЛЯ ПРОСМОТРА 3D ОБЪЕКТОВ

## 3.1 Модуль настройки основных параметров

Данный фал создается как шаблон приложения в нем создан class SoftwareRender в конструкторе, которого были описаны стандартные объекты для pygame такие как разрешение, поверхность для отрисовки и настройки желаемого количества кадров.

class SoftwareRender:

def \_\_init\_\_(self):

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()

Метод def create\_objects(self) данный метод для создания экземпляра объекта, который вызывается из конструктора класса основного приложения здесь задается положение камеры и положение объекта в трехмерном пространстве, а также добавляется экземпляр класса проекций.

def create\_objects(self):

self.camera = Camera(self, [-5, 6, -55])

self.projection = Projection(self)

self.object = self.get\_object\_from\_file('resources/mm.obj')

self.object.rotate\_y(-math.pi / 4)

Метод def draw(self) в данном методе происходит отрисовка трехмерном пространстве.

def draw(self):

self.screen.fill(pg.Color('darkslategray'))

self.object.draw()

Метод def run(self) в данном методе описан главный цикл программы, в котором происходит отрисовка объектов в трехмерном пространстве так же проверка на выход из приложения и вывод количество кадров.

def run(self):

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)

Создание экземпляра запуска класса и вызов class SoftwareRender и вызов метода def run(self).

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

app = SoftwareRender()

app.run()

## 3.2 Модуль формирования матрицы функций, зависящий от входных параметров

В данном файле формируются матрицы функций, которые будут формировать зависимости от входных параметров, представленные в пункте 1, где каждая матрица будет массивом библиотеки nympy.

Метод def translate(pos) в данном методе описан массив для вычисления входных данных и построения объекта по координатам. Математический алгоритм данного массива описан в пункте 1.5.

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]

])

Метод def rotate\_x(a), def rotate\_y(a), def rotate\_z(a) в данный метод описан массив расчёта координат объекта при повороте массива внутри трехмерной сцены. Математический алгоритм данного массива описан в пункте 1.6.

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]

])

Метод def scale(n) в данном методе описан массив, который масштабирует координаты при смени ракурса или отдалений от объекта в трехмерной сцене. Математический алгоритм данного массива описан в пункте 1.7.

def scale(n):

return np.array([

[n, 0, 0, 0],

[0, n, 0, 0],

[0, 0, n, 0],

[0, 0, 0, 1]

])

## 3.3 Метод формирования вершин массива и отрисовки полигонов

Здесь задается объект суда же импортируются матричные функций из пункта 3.2, так же создается класс class Object3D который будет взаимодействовать с главной программой пункт 3.1. Здесь так же описаны вершины массива в nympy и цветовая отрисовка полигонов.

class Object3D:

def \_\_init\_\_(self, render, vertexes='', faces=''):

self.render = render

self.vertexes = np.array([np.array(v) for v in vertexes])

self.faces = np.array([np.array(face) for face in faces])

self.translate([0.0001, 0.0001, 0.0001])

self.font = pg.font.SysFont('Arial', 30, bold=True)

self.color\_faces = [(pg.Color('orange'), face) for face in self.faces]

self.movement\_flag, self.draw\_vertexes = True, False

self.label = ' '

Ряд методов def translate(self, pos), def scale(self, scale\_to), def rotate\_x(self, angle), def rotate\_y(self, angle), def rotate\_z(self, angle) для изменения нашего объекта в пространстве при помощи импортированных функций из пункта 3.2, матриц перемещения, вращения и масштабирования. Здесь же используется преимущество библиотеки nympy, то есть весь массив вершин можно умножить на нужную матрицу.

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)

Метод def screen\_projection(self) в данном методе переносятся вершины объектов в пространстве камеры и приносится матрица пространства отсечения, нормализуются координаты вершин путем отсечения координат больше единицы или меньше минуса одного, отображение объекта путем прохождения всех граней а принимать отображение нужные грани будем из отсечённых вершин которые приняли нулевое значение тем самым отрисовывать объект в пространстве.

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 index, color\_face in enumerate(self.color\_faces):

color, face = color\_face

polygon = vertexes[face]

if not any\_func(polygon, self.render.H\_WIDTH, self.render.H\_HEIGHT):

pg.draw.polygon(self.render.screen, color, polygon, 1)

if self.label:

text = self.font.render(self.label[index], True, pg.Color('white'))

self.render.screen.blit(text, polygon[-1])

if self.draw\_vertexes:

for vertex in vertexes:

if not any\_func(vertex, self.render.H\_WIDTH, self.render.H\_HEIGHT):

pg.draw.circle(self.render.screen, pg.Color('white'), vertex, 2)

## 3.4 Метод построение и управление камеры

Создан класс class Camera подается начально расположение камеры основой класса будут векторы ориентаций, которые задаются в виде нормализованных единичных векторов так же определяется горизонтальное и вертикальная область видимости для камеры и задается ближняя и дельное отсечение видимости для отсечённой пирамиды обзора так же для управления камеры задается скорость перемещения и вращения.

class Camera:

def \_\_init\_\_(self, render, position):

self.render = render

self.position = np.array([\*position, 1.0])

self.forward = np.array([0, 0, 1, 1])

self.up = np.array([0, 1, 0, 1])

self.right = np.array([1, 0, 0, 1])

self.h\_fov = math.pi / 3

self.v\_fov = self.h\_fov \* (render.HEIGHT / render.WIDTH)

self.near\_plane = 0.1

self.far\_plane = 100

self.moving\_speed = 0.3

self.rotation\_speed = 0.015

Метод def translate\_matrix(self), def rotate\_matrix(self), def camera\_matrix для формирования вышеизложенных матриц, которые нужны для перехода в систему координат камеры.

def translate\_matrix(self):

x, y, z, w = self.position

return np.array([

[1, 0, 0, 0],

[0, 1, 0, 1],

[0, 0, 1, 0],

[-x, -y, -z, 1]

])

def rotate\_matrix(self):

rx, ry, rz, w = self.right

fx, fy, fz, w = self.forward

ux, uy, uz, w = self.up

return np.array([

[rx, ux, fx, 0],

[ry, uy, fy, 0],

[rz, uz, fz, 0],

[0, 0, 0, 1]

])

def camera\_matrix(self):

return self.translate\_matrix() @self.rotate\_matrix()

Метод def control(self) в данном методе задано управление камеры при помощи стандартных методов библиотеки pygame для перемещения в дол всех трёх осей мировой координаты и вращение вокруг своей оси.

key = pg.key.get\_pressed()

if key[pg.K\_a]:

self.position -= self.right \* self.moving\_speed

if key[pg.K\_d]:

self.position += self.right \* self.moving\_speed

if key[pg.K\_w]:

self.position += self.forward \* self.moving\_speed

if key[pg.K\_s]:

self.position -= self.forward \* self.moving\_speed

if key[pg.K\_q]:

self.position += self.up \* self.moving\_speed

if key[pg.K\_e]:

self.position -= self.up \* self.moving\_speed

if key[pg.K\_LEFT]:

self.camera\_yaw(-self.rotation\_speed)

if key[pg.K\_RIGHT]:

self.camera\_yaw(self.rotation\_speed)

if key[pg.K\_UP]:

self.camera\_pitch(-self.rotation\_speed)

if key[pg.K\_DOWN]:

self.camera\_pitch(self.rotation\_speed)

## 3.5 Метод формирование матрицы проекций

Создан класс class Projection в котором будут простые атрибуты, которые формируют необходимые значения для формирования матрицы проекций который мы возьмем из класса class Camera и определим элементы матрицы и сформируем матрицу проекций и формирования преобразования координат вершин в разрешения экрана.

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]

])

# ЗАКЛЮЧЕНИЕ

Главным результатом курсовой работы является создание системы, способной формировать и визуализировать трехмерное пространство.

В данном проекте была затронута актуальная и бурно развивающаяся область индустрии разработки компьютерных игровых программ.

В рамках курсовой работы были получены следующие теоретические и практические результаты:

- проведен анализ модулей, составляющих игровые приложения;

- рассмотрены основные компоненты и возможности существующих популярных игровых движков;

- в ходе анализа предметной области определены основные цели и задачи, а также область применения разрабатываемой системы;

- рассмотрена логика построения и отображения трехмерного виртуального пространства;

- разработан алгоритм работы базового приложения и его модулей;

- произведено тестирование разработанного на базе системы приложения, показывающее возможности графического движка, а также зн

Перечисленные результаты способствовали созданию гибкого приложения с возможностью его расширения и модификации, который обладает хорошей производительностью и стабильностью.

Разработанная система модулей может использоваться:

- как графический движок для создания игровых приложений для образовательных или развлекательных целей;

- как приложение для создания трехмерных демонстрационных видеороликов;

- как приложение для быстрой визуализации трехмерных сцен;

- как дополнительный модуль для других создаваемых приложений.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Мэтиз, Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения: учебник / Э. Мэтиз. –Санкт-Петербург : Питер, 2020. –232 с.

Пол, Б. Изучаем программирование на Python, веб-приложения: учебник / Б. Пол. –Санкт-Петербург : Питер, 2020. –128 с.

Лутц, М. Изучаем Python. Том 1, веб-приложения: учебник / М. Лутц. –Москва : Вильямс, 2019. –205 с.

Билл, Л. Простой Python. Современный стиль программирования, веб-приложения: учебник / Л. Билл. – Санкт-Петербург : Питер, 2019. –164 с.

Дэн, Б. Чистый Python. Тонкости программирования для профи, веб-приложения: учебник / Б. Дэн. – Санкт-Петербург : Питер, 2018. –220 с.

Лучано, Р. Python. К вершинам мастерства, веб-приложения: учебник / Р. Лучано. – Москва : ДМК Пресс, 2018. –159 с.

Даг, Х. Стандартная библиотека Python 3, веб-приложения: учебник / Х. Даг. – Санкт-Петербург : Питер, 2018. –196 с.

Дэвид, М. Python. Книга Рецептов, веб-приложения: учебник / М. Девид. – Москва : ДМК Пресс, 2019. –243 с.

Гарри, П. Python. Разработка на основе тестирования, веб-приложения: учебник / П. Гарри. – Москва : ДМК Пресс, 2018. –267 с.

Мигель, Г. Разработка веб-приложений с использованием Flask на языке Python, веб-приложения: учебник / Г. Мигель. – Москва : ДМК Пресс, 2016. –267 с.

Андреас, М. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными, веб-приложения: учебник / М. Андреас. – Москва : Вильямс, 2017. –167 с.

Вандер, П. Python для сложных задач. Наука о данных и машинное обучение, веб-приложения: учебник / П. Вандер. – Санкт-Петербург : Питер, 2018. –115 с.

Франсуа, Ш. Глубокое обучение на Python, веб-приложения: учебник / Ш. Франсуа. – Санкт-Петербург : Питер, 2018. –232 с.

Эрик, С. Программирование компьютерного зрения на Python, веб-приложения: учебник / С. Эрик. – Москва : ДМК Пресс, 2016. –177 с.

Штефан, В. Элегантный SciPy, веб-приложения: учебник / В. Штефан. – Москва : ДМК Пресс, 2018. –279 с.

# ПРИЛОЖЕНИЕ А

main.py

from object\_3d import \*  
from camera import \*  
from projection import \*  
import pygame as pg  
  
  
class SoftwareRender:  
 def \_\_init\_\_(self):  
 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, [-5, 6, -55])  
 self.projection = Projection(self)  
 self.object = self.get\_object\_from\_file('resources/mm.obj')  
 self.object.rotate\_y(-math.pi / 4)  
  
 def get\_object\_from\_file(self, filename):  
 vertex, faces = [], []  
 with open(filename) as f:  
 for line in f:  
 if line.startswith('v '):  
 vertex.append([float(i) for i in line.split()[1:]] + [1])  
 elif line.startswith('f'):  
 faces\_ = line.split()[1:]  
 faces.append([int(face\_.split('/')[0]) - 1 for face\_ in faces\_])  
 return Object3D(self, vertex, faces)  
  
 def draw(self):  
 self.screen.fill(pg.Color('darkslategray'))  
 self.object.draw()  
  
 def run(self):  
 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()

# ПРИЛОЖЕНИЕ Б

matrix\_functions.py

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]  
 ])

# ПРИЛОЖЕНИЕ В

object\_3d.py

import pygame as pg  
from matrix\_functions import \*  
from numba import njit  
  
  
@njit(fastmath=True)  
def any\_func(arr, a, b):  
 return np.any((arr == a) | (arr == b))  
  
  
class Object3D:  
 def \_\_init\_\_(self, render, vertexes='', faces=''):  
 self.render = render  
 self.vertexes = np.array([np.array(v) for v in vertexes])  
 self.faces = np.array([np.array(face) for face in faces])  
 self.translate([0.0001, 0.0001, 0.0001])  
  
 self.font = pg.font.SysFont('Arial', 30, bold=True)  
 self.color\_faces = [(pg.Color('orange'), face) for face in self.faces]  
 self.movement\_flag, self.draw\_vertexes = True, False  
 self.label = ''  
  
 def draw(self):  
 self.screen\_projection()  
 self.movement()  
  
 def movement(self):  
 if self.movement\_flag:  
 self.rotate\_y(-(pg.time.get\_ticks() % 0.005))  
  
 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 index, color\_face in enumerate(self.color\_faces):  
 color, face = color\_face  
 polygon = vertexes[face]  
 if not any\_func(polygon, self.render.H\_WIDTH, self.render.H\_HEIGHT):  
 pg.draw.polygon(self.render.screen, color, polygon, 1)  
 if self.label:  
 text = self.font.render(self.label[index], True, pg.Color('white'))  
 self.render.screen.blit(text, polygon[-1])  
  
 if self.draw\_vertexes:  
 for vertex in vertexes:  
 if not any\_func(vertex, self.render.H\_WIDTH, self.render.H\_HEIGHT):  
 pg.draw.circle(self.render.screen, pg.Color('white'), vertex, 2)  
  
 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)  
  
  
class Axes(Object3D):  
 def \_\_init\_\_(self, render):  
 super().\_\_init\_\_(render)  
 self.vertexes = np.array([(0, 0, 0, 1), (1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1)])  
 self.faces = np.array([(0, 1), (0, 2), (0, 3)])  
 self.colors = [pg.Color('red'), pg.Color('green'), pg.Color('blue')]  
 self.color\_faces = [(color, face) for color, face in zip(self.colors, self.faces)]  
 self.draw\_vertexes = False  
 self.label = 'XYZ'

# ПРИЛОЖЕНИЕ Г

camera.py

import pygame as pg  
from matrix\_functions import \*  
  
  
class Camera:  
 def \_\_init\_\_(self, render, position):  
 self.render = render  
 self.position = np.array([\*position, 1.0])  
 self.forward = np.array([0, 0, 1, 1])  
 self.up = np.array([0, 1, 0, 1])  
 self.right = np.array([1, 0, 0, 1])  
 self.h\_fov = math.pi / 3  
 self.v\_fov = self.h\_fov \* (render.HEIGHT / render.WIDTH)  
 self.near\_plane = 0.1  
 self.far\_plane = 100  
 self.moving\_speed = 0.3  
 self.rotation\_speed = 0.015  
  
 def control(self):  
 key = pg.key.get\_pressed()  
 if key[pg.K\_a]:  
 self.position -= self.right \* self.moving\_speed  
 if key[pg.K\_d]:  
 self.position += self.right \* self.moving\_speed  
 if key[pg.K\_w]:  
 self.position += self.forward \* self.moving\_speed  
 if key[pg.K\_s]:  
 self.position -= self.forward \* self.moving\_speed  
 if key[pg.K\_q]:  
 self.position += self.up \* self.moving\_speed  
 if key[pg.K\_e]:  
 self.position -= self.up \* self.moving\_speed  
  
 if key[pg.K\_LEFT]:  
 self.camera\_yaw(-self.rotation\_speed)  
 if key[pg.K\_RIGHT]:  
 self.camera\_yaw(self.rotation\_speed)  
 if key[pg.K\_UP]:  
 self.camera\_pitch(-self.rotation\_speed)  
 if key[pg.K\_DOWN]:  
 self.camera\_pitch(self.rotation\_speed)  
  
 def camera\_yaw(self, angle):  
 rotate = rotate\_y(angle)  
 self.forward = self.forward @ rotate  
 self.right = self.right @ rotate  
 self.up = self.up @ rotate  
  
 def camera\_pitch(self, angle):  
 rotate = rotate\_x(angle)  
 self.forward = self.forward @ rotate  
 self.right = self.right @ rotate  
 self.up = self.up @ rotate  
  
 def translate\_matrix(self):  
 x, y, z, w = self.position  
 return np.array([  
 [1, 0, 0, 0],  
 [0, 1, 0, 1],  
 [0, 0, 1, 0],  
 [-x, -y, -z, 1]  
 ])  
  
 def rotate\_matrix(self):  
 rx, ry, rz, w = self.right  
 fx, fy, fz, w = self.forward  
 ux, uy, uz, w = self.up  
 return np.array([  
 [rx, ux, fx, 0],  
 [ry, uy, fy, 0],  
 [rz, uz, fz, 0],  
 [0, 0, 0, 1]  
 ])  
  
 def camera\_matrix(self):  
 return self.translate\_matrix() @ self.rotate\_matrix()

# ПРИЛОЖЕНИЕ Д

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]  
 ])