МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет систем управления и робототехники

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
по дисциплине
«Практическая линейная алгебра»

по теме:
3D ПРЕОБРАЗОВАНИЯ

Студент:
Группа № R3335 *Зыкин Л. В.*

Предподаватель:
должность, уч. степень, уч. звание *Догадин Е. В.*

Санкт-Петербург
2025

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1 Задание 1. Построение куба

Для построения куба используется функция `create_cube()`, которая возвращает массив координат 8 вершин с центром в начале координат:

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -1 & -1 \\ 1 & 1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Грани задаются функцией `get_faces()` (по 4 индекса на грань). Для визуализации применяется функция `plot_cube` из matplotlib, а масштаб осей автоматически подбирается функцией `set_axes_equal`.

Рисунок 1 — Исходный куб, построенный с помощью Python

Почему мы используем четырехкомпонентный вектор, а не трех? Четырехкомпонентный вектор (однородные координаты) позволяет записывать все аффинные преобразования (перенос, масштабирование, поворот) в виде одного матричного умножения. Это удобно для объединения нескольких преобразований в одну матрицу и для реализации графических алгоритмов.

Как задать другие фигуры? Чтобы построить другую фигуру, нужно изменить массив вершин (например, для тетраэдра, пирамиды и т.д.) и соответствующим образом задать грани (список индексов вершин для каждой грани).

Масштабирование реализовано функцией `scale(vertices, sx, sy, sz)`, которая умножает координаты вершин на соответствующие коэффициенты по осям:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразование: $V' = V \cdot S^T$, где V — вектор в однородных координатах $(x, y, z, 1)$.

Как выглядит матрица масштабирования? См. формулу выше. В коде можно менять значения s_x, s_y, s_z и наблюдать, как куб становится вытянутым или сжатым по соответствующим осям.

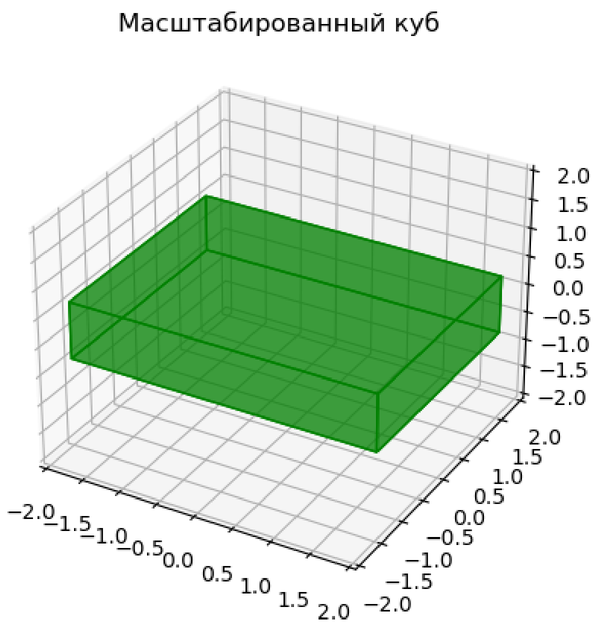

Рисунок 2 — Масштабированный куб (Python)

Для переноса используется функция `translate(vertices, tx, ty, tz)`, которая сдвигает все вершины на заданный вектор:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразование: $V'' = V' \cdot T^T$, где V' — вектор в однородных координатах.

Как выглядит матрица переноса? В Python это просто вектор $T = [t_x, t_y, t_z]$; в однородных координатах это матрица:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Эквивалентны ли преобразования TS и ST? Нет, порядок важен: $TS \neq ST$. Если сначала масштабировать, а потом сдвигать, результат будет отличаться от случая, когда сначала сдвигают, а потом масштабируют (см. рисунки и код).

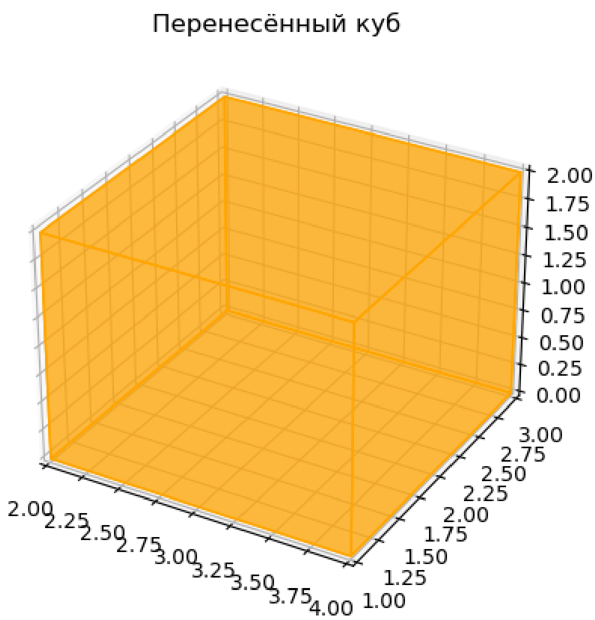

Рисунок 3 — Перенесённый куб (Python)

Поворот реализован функциями `rotate_x`, `rotate_y`, `rotate_z`. Например, матрица поворота вокруг оси Z:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Аналогично для R_x и R_y:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразование: $V''' = V'' \cdot R_z^T$, где V'' — вектор в однородных координатах.

Пример вектора v для поворота Для поворота вокруг оси Z используем $v = (0, 0, 1)$. Для оси X — $v = (1, 0, 0)$, для оси Y — $v = (0, 1, 0)$.

Формула Родрига-Гамильтона для произвольного вектора v Поворот на угол θ вокруг нормированного вектора v задаётся матрицей:

$$R_v^{(4)}(\theta) = \begin{bmatrix} R_v(\theta) & 0 \\ 0 & 1 \end{bmatrix}$$

где $R_v(\theta)$ — обычная 3×3 матрица Родрига-Гамильтона, а 0 — столбец из трёх нулей.

Как меняется кубик при разных осях и углах? Куб вращается вокруг выбранной оси на заданный угол. Если ось совпадает с одной из координатных, поворот происходит в соответствующей плоскости. При изменении угла меняется положение вершин куба.

Как работает формула поворота? Формула Родрига-Гамильтона строит ортогональную матрицу, которая реализует поворот вокруг произвольного вектора v.

Явные формулы R_x, R_y, R_z

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Как добавить вектор оси вращения на график? В Python для этого используется функция `ax.quiver`. Например, чтобы нарисовать вектор v из начала координат:

```
ax.quiver(0, 0, 0, v[0], v[1], v[2], color='black', linewidth=2)
```

Достаточно ли трех матриц R_x, R_y, R_z для описания всех вращений? Да, любая последовательность вращений в 3D может быть представлена как композиция трех вращений вокруг осей (Теорема Эйлера).

Можно ли восстановить ось вращения? Да, по итоговой матрице поворота можно восстановить ось и угол вращения используя Теорему Эйлера.

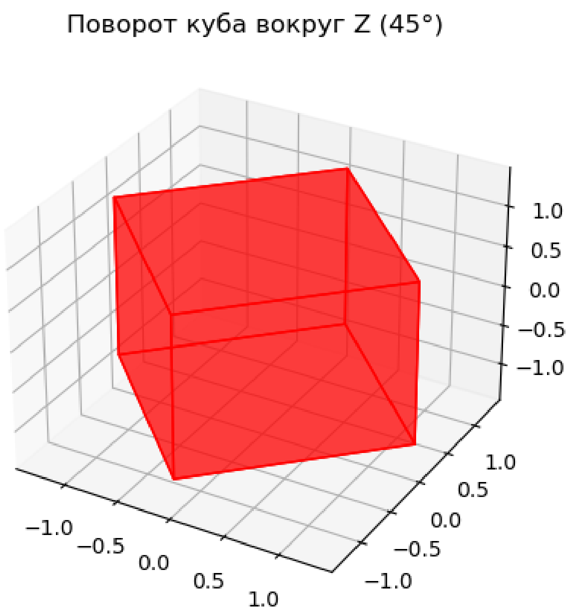

Рисунок 4 — Поворот куба вокруг оси Z (Python)

1.2 Задание 5. Вращение куба вокруг вершины

Для вращения вокруг вершины используется последовательность:

1. Сдвиг куба так, чтобы выбранная вершина стала началом координат: $V_{shift} = V - v_0$
2. Поворот: $V_{rot} = V_{shift} \cdot R^T$
3. Обратный сдвиг: $V_{final} = V_{rot} + v_0$

Как построить матрицу для вращения вокруг вершины? Сначала сдвиньте куб так, чтобы нужная вершина стала началом координат, затем выполните поворот, затем верните куб обратно (см. формулы выше).

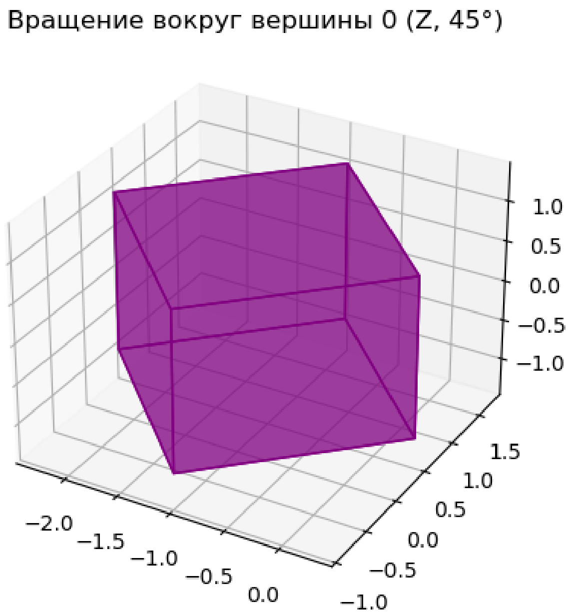

Рисунок 5 — Вращение куба вокруг вершины (Python)

1.3 Задание 6. Имитация камеры

Для имитации движения и поворота камеры используется функция `camera_transform`, которая применяет к кубу повороты и сдвиг всей сцены:

$$V_{cam} = (((V \cdot R_x^T) \cdot R_y^T) \cdot R_z^T) + T$$

где R_x, R_y, R_z — матрицы поворота, T — вектор сдвига.

Как посмотреть на график “снизу”? В Python: `ax.view_init(azim=0, elev=-90)`.

Как выбрать матрицы Tc и Rc(θ) для камеры? В коде: `camera_transform(vertices, tx, ty, tz, rx, ry, rz)` — где Tc задаёт сдвиг, а Rc(θ) — повороты камеры.

Как найти обратную матрицу положения камеры C^{-1}? Обратная матрица для композиции поворотов и сдвига камеры строится как обратные к исходным матрицам, применённые в обратном порядке: сначала обратный сдвиг, затем обратные повороты.

Какой эффект дает обратная матрица камеры? Сцена будет выглядеть так, как если бы камера переместилась в начало координат и смотрела вдоль стандартной оси. Это позволяет удобно анализировать сцену с разных точек зрения.

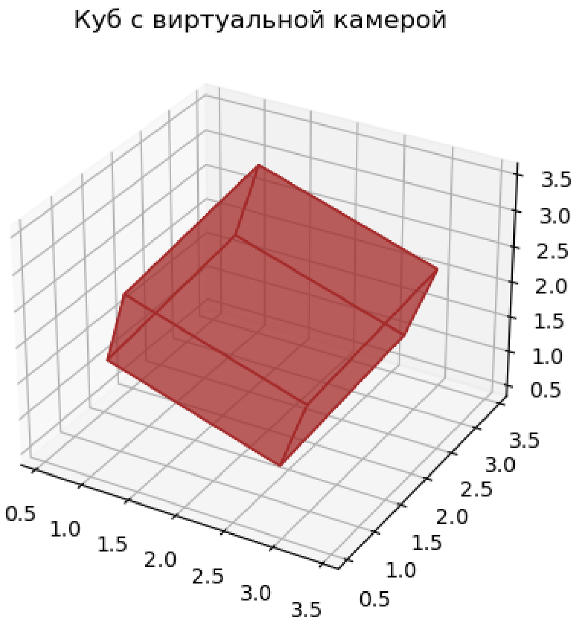

Рисунок 6 — Куб с виртуальной камерой (Python)

Для перспективной проекции используется функция `perspective_projection`, реализующая матричное преобразование:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix}$$

Преобразование:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = P \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Переход к декартовым координатам: $x_{screen} = x'/w'$, $y_{screen} = y'/w'$.

Как исследовать деформацию пространства под влиянием перспективы? Вращайте график с помощью `ax.view_init()` и наблюдайте, как меняется форма и взаимное расположение объектов при разных углах обзора.

Как устроена матрица перспективы и какие параметры выбрать? См. формулу выше: $[x', y', 0] = \frac{d}{z+d}[x, y, 0]$. Параметр d — расстояние до камеры (чем больше d, тем слабее эффект перспективы). Для исследования деформации пространства можно вращать график с помощью `ax.view_init()`.

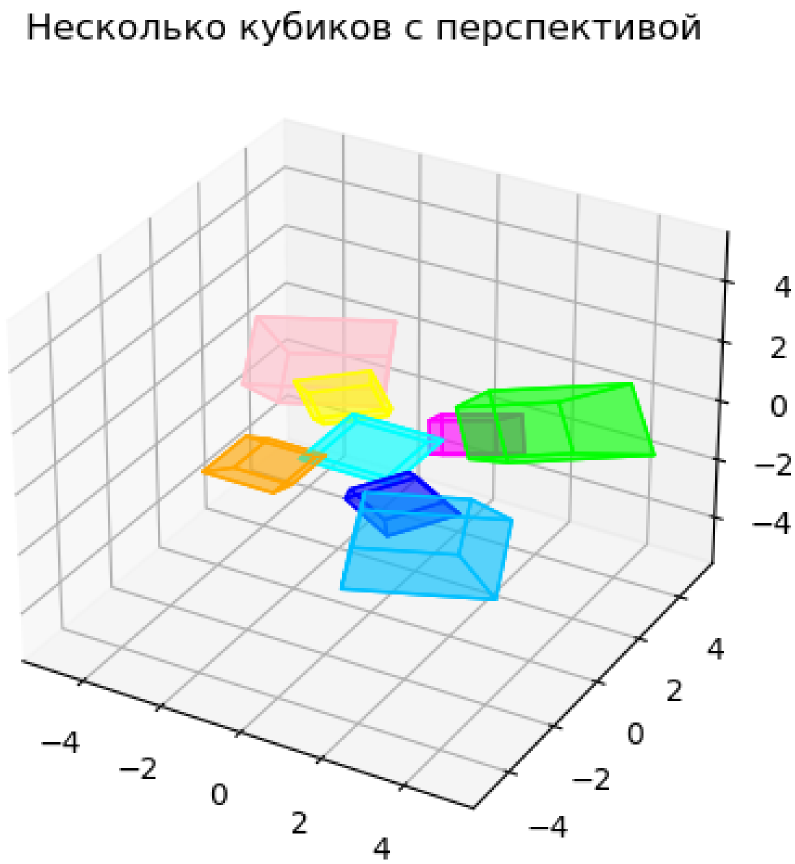

Рисунок 7 — Несколько кубиков с перспективой (Python)

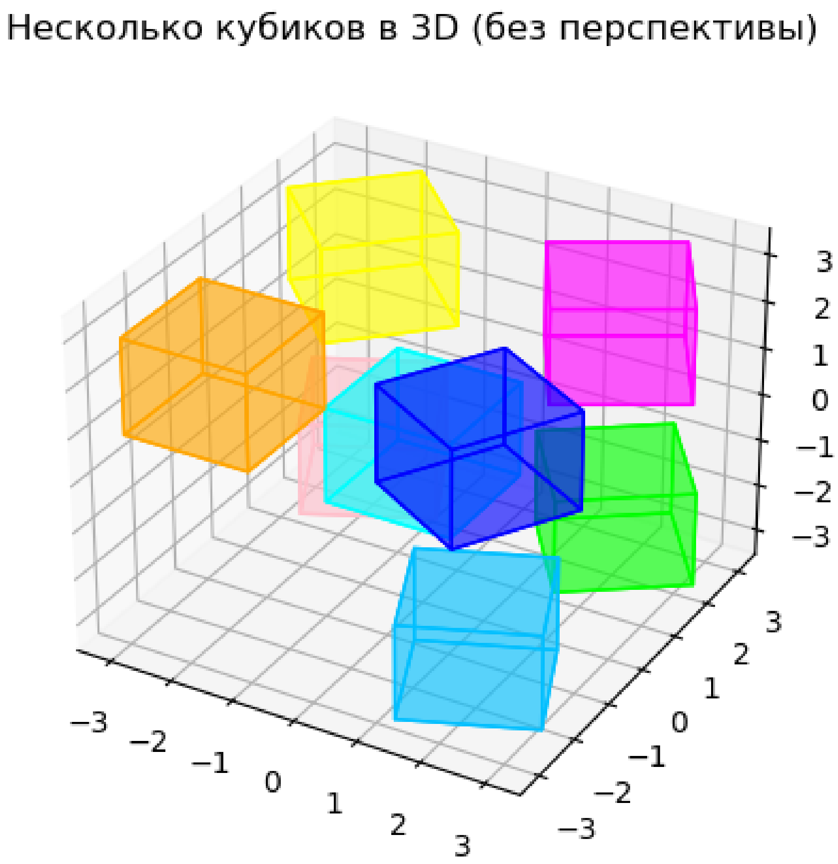

Рисунок 8 — Те же кубики без перспективы (Python)

1.4 Выводы

В ходе работы были реализованы и исследованы основные преобразования в 3D: масштабирование, перенос, поворот, вращение вокруг вершины, а также перспективная проекция и имитация камеры. Все этапы выполнены на Python с использованием numpy и matplotlib, что позволило гибко визуализировать и анализировать результаты.