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
О ЛАБОРАТОРНОЙ РАБОТЕ № 3

По дисциплине «Практическая линейная алгебра»
на тему: «матрица в 3D-графике»

Студенты:
Гизбрехт В.Д. группа 1
Ли Х.С. группа 1
Лаврик В.В. группа 4

Проверил:

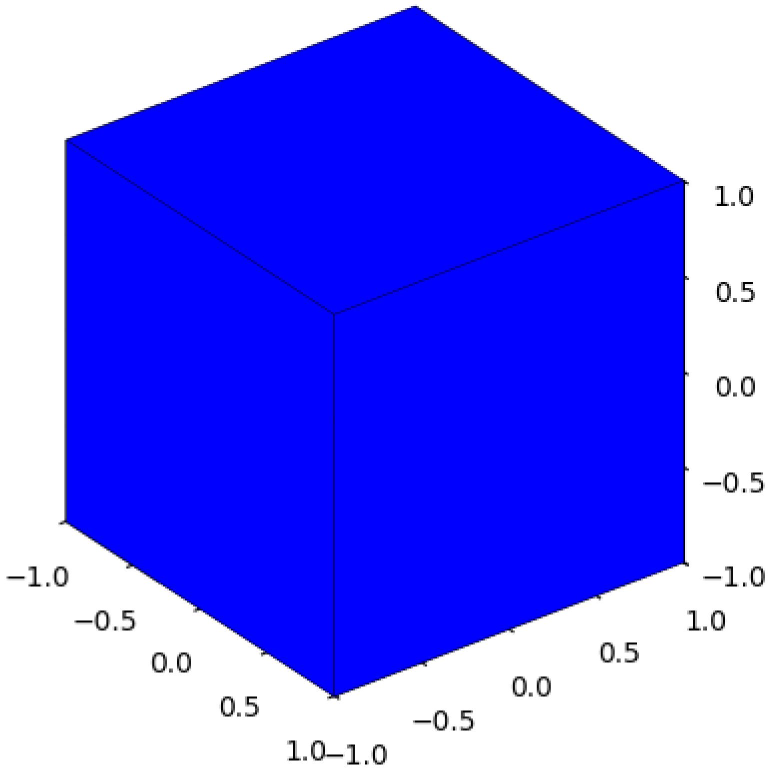
г. Санкт-Петербург
2024

Подготовка

Для выполнения данной лабораторной работы нужно начать с создания кубика. Кубик будет создаваться с помощью Python, используя код, предоставленный в задачнике по 3 лабораторной работе:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
vertices_cube = np.array ([
[-1, 1, 1, -1, -1, 1, 1, -1],
[-1, -1, 1, 1, -1, -1, 1, 1],
[-1, -1, -1, -1, 1, 1, 1, 1],
[ 1, 1, 1, 1, 1, 1, 1, 1]
])
faces_cube = np.array ([
[0, 1, 5, 4],
[1, 2, 6, 5],
[2, 3, 7, 6],
[3, 0, 4, 7],
[0, 1, 2, 3],
[4, 5, 6, 7]
])
fig = plt.figure ()
ax = fig.add_subplot(projection ='3d', proj_type = 'ortho')
def draw_shape( vertices , faces , color ):
    vertices = ( vertices [:3 , :] / vertices [3 , :]).T
    ax.add_collection3d ( Poly3DCollection ( vertices [ faces ] , facecolors = color,
    edgecolors ='k', linewidths =0.2) )

draw_shape(vertices_cube , faces_cube , 'blue')

ax.set_box_aspect ([1 ,1 ,1])
ax.set_xlim ( -1 , 1) ; ax.set_ylim ( -1 , 1) ; ax.set_zlim ( -1 , 1)
ax.view_init ( azim = -37.5 , elev =30)
ax.set_xticks ( np.linspace ( -1 , 1 , 5) ) ; ax.set_yticks ( np . linspace ( -1 , 1 , 5) )
ax.set_zticks ( np.linspace ( -1 , 1 , 5) )

plt.show ()
```

После выполнения данного кода, мы получаем данный куб:



Задание №1 Разбор кода Python

Для начала, стоит разобраться, что делает данный код.

Код создает и визуализирует трехмерный куб с помощью библиотеки Matplotlib и ее модуля для 3D-графики.

vertices_Cube – матрица 4x8, которая определяет координаты вершин куба. Каждый столбец представляет собой координаты x, y и z вершины, а в последней строке содержатся единицы. **faces_cube** – это матрица размером 6x4, которая определяет связи вершин куба для создания его граней. Каждая строка представляет грань куба, и значения в каждой строке соответствуют индексам вершин, определенных в *vertices_Cube*.

Функция *draw_shape(vertices, faces, color)* — рисует форму с заданными вершинами, гранями и цветом.

vertices[:3, :] / vertices[3, :] нормализует однородные координаты, переводя их в обычные 3D-координаты.

Poly3DCollection создает поверхность фигуры (граней куба) с заданным цветом, черными краями *(edgecolors='k')* и толщиной линий *(linewidths=0.2)*

ax.add_collection3d добавляет созданные полигоны к 3D-оси *ax*

ax.set_box_aspect([1,1,1]) — задает равномерное соотношение сторон по осям для создания правильного куба.

ax.set_xlim, ax.set_ylim, ax.set_zlim — задают пределы отображения для осей.

ax.view_init(azim=-37.5, elev=30) — устанавливает угол обзора камеры для лучшего вида на куб.

ax.set_xticks, ax.set_yticks, ax.set_zticks — устанавливают разметку осях, чтобы сделать график более читабельным.

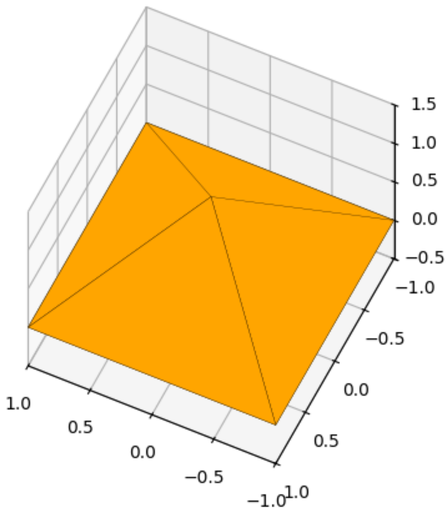
В данном задании используется четырехкомпонентный вектор (включая компоненту w) для представления вершин куба в однородных (гомогенных) координатах. Это необходимо для выполнения линейных преобразований, таких как сдвиг и перспективные проекции в трехмерном пространстве.

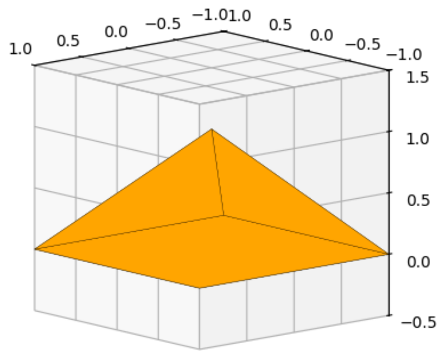
Основные моменты использования гомогенных координат в 3D-графике:

1. **Сдвиги и проекции**: Однородные координаты позволяют выполнять не только вращения и масштабирования, но и сдвиги, которые невозможно сделать с обычными трехмерными векторами без изменения их значения напрямую.
2. **Перспективное отображение**: для реалистичного 3D-рендеринга необходимо моделировать перспективу, где дальние объекты кажутся меньше. Это достигается благодаря гомогенной координате w, которая масштабирует координаты x, y и z относительно точки наблюдения, что дает эффект перспективного искажения.
3. **Вектор направления**: когда компонент w=0, вектор считается направлением (или бесконечно удаленной точкой), а не позицией в пространстве. Такой вектор не может быть сдвинут, что полезно, например, для представления направлений и нормалей, которые не должны изменяться при трансляции.

Гомогенные координаты — это способ представления точек в пространстве с помощью дополнительных координат, который упрощает выполнение геометрических преобразований, таких как сдвиги, повороты, масштабирование и проекции.

Для преобразования куба в другую фигуру нам нужно поработать с нашими вершинами в массивах





После простого преобразования, мы получаем пирамиду с помощью такого кода:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

# Задание координат вершин пирамиды
vertices_pyramid = np.array([
    [0, 1, -1, -1, 1],       # x-координаты
    [0, -1, -1, 1, 1],       # y-координаты
    [1, 0, 0, 0, 0],         # z-координаты
    [1, 1, 1, 1, 1]          # w-координаты
])

# Задание граней пирамиды как списка списков
faces_pyramid = [
    [0, 1, 2],  # треугольная грань
    [0, 2, 3],  # треугольная грань
    [0, 3, 4],  # треугольная грань
    [0, 4, 1],  # треугольная грань
    [1, 2, 3, 4] # основание (квадратная грань)
]

# Создание фигуры и осей для 3D-графика
fig = plt.figure()
ax = fig.add_subplot(projection='3d', proj_type='ortho')

# Функция для рисования фигуры
def draw_shape(vertices, faces, color):
    vertices = (vertices[:3, :] / vertices[3, :]).T  # Преобразуем однородные координаты в 3D
    ax.add_collection3d(Poly3DCollection([vertices[face] for face in faces], facecolors=color, edgecolors='k', linewidths=0.2))

# Отрисовка пирамиды
draw_shape(vertices_pyramid, faces_pyramid, 'orange')

# Настройки осей
ax.set_box_aspect([1, 1, 1])
ax.set_xlim(-1, 1); ax.set_ylim(-1, 1); ax.set_zlim(-0.5, 1.5)
ax.view_init(azim=-37.5, elev=30)
ax.set_xticks(np.linspace(-1, 1, 5)); ax.set_yticks(np.linspace(-1, 1, 5))
ax.set_zticks(np.linspace(-0.5, 1.5, 5))

plt.show()
```

vertices_pyramid: задает координаты вершин. Пирамида будет иметь пять вершин: одну на вершине и четыре на основании.
faces_pyramid: задает пять граней: четыре треугольные грани и одно квадратное основание.

Задание №2 Изменение масштаба куба

Пусть S_x, S_y, и S_z — коэффициенты масштабирования вдоль осей x, y, и z соответственно. Тогда матрица масштабирования для 3D-графики будет выглядеть так:

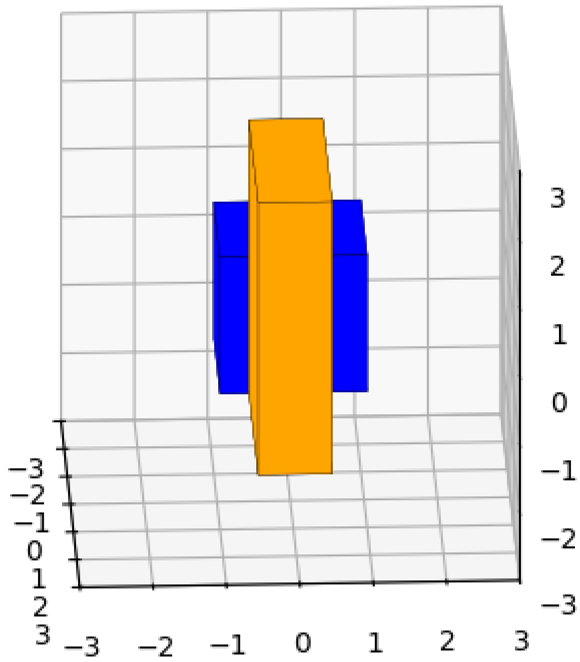
$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Применив эту матрицу к каждому вектору (точке) куба, мы изменим его размер.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

vertices_cube = np.array([
    [-1, 1, 1, -1, -1, 1, 1, -1],
    [-1, -1, 1, 1, -1, -1, 1, 1],
    [-1, -1, -1, -1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1]
])
faces_cube = np.array([
    [0, 1, 5, 4],
    [1, 2, 6, 5],
    [2, 3, 7, 6],
    [3, 0, 4, 7],
    [0, 1, 2, 3],
    [4, 5, 6, 7]
])

S_x, S_y, S_z = 1.5, 0.5, 2
scale_matrix = np.array([
    [S_x, 0, 0, 0],
    [0, S_y, 0, 0],
    [0, 0, S_z, 0],
    [0, 0, 0, 1]
])

vertices_scaled = scale_matrix @ vertices_cube

fig = plt.figure()
ax = fig.add_subplot(projection='3d', proj_type='ortho')

def draw_shape(vertices, faces, color):
    vertices = (vertices[:3, :] / vertices[3, :]).T
    ax.add_collection3d(Poly3DCollection(vertices[faces], facecolors=color, edgecolors='k', linewidths=0.2))

draw_shape(vertices_cube, faces_cube, 'blue')       # Исходный куб
draw_shape(vertices_scaled, faces_cube, 'orange')   # Масштабированный куб

ax.set_box_aspect([1, 1, 1])
ax.set_xlim(-3, 3); ax.set_ylim(-3, 3); ax.set_zlim(-3, 3)
ax.view_init(azim=-37.5, elev=30)
ax.set_xticks(np.linspace(-3, 3, 7)); ax.set_yticks(np.linspace(-3, 3, 7))
ax.set_zticks(np.linspace(-3, 3, 7))

plt.show()
```

scale_matrix задает матрицу масштабирования с коэффициентами S_x=1.5, S_y=0.5, S_z=2.

vertices_scaled — результат умножения матрицы масштабирования на вершины куба. Это новые координаты куба после масштабирования. Мы отрисовываем исходный куб (синий) и масштабированный куб (оранжевый) на одном графике для сравнения



Задание №3 <u>Перемещение куба</u>

Матрица перемещения T (или трансляции) отвечает за сдвиг фигуры в пространстве. Она добавляет фиксированные значения к координатам объекта, перемещая его на заданные расстояния вдоль осей x, y и z

Для трансляции на Tx, Ty и Tz вдоль осей x, y и z соответственно, матрица трансляции T выглядит так:

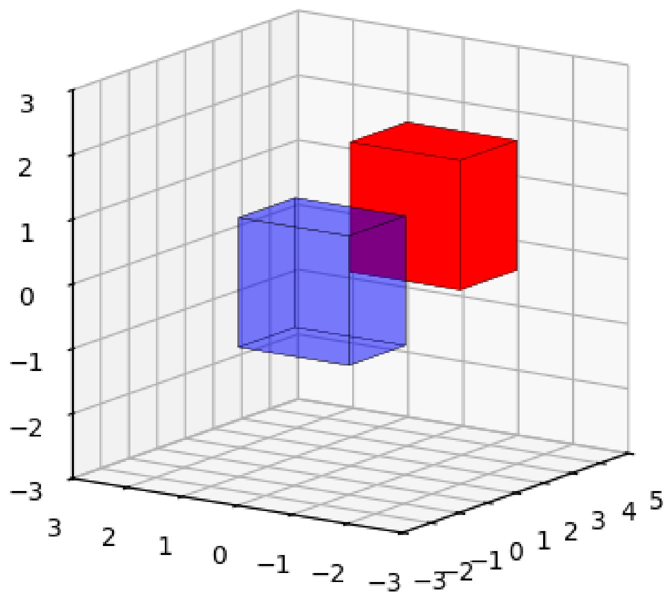
$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Эта матрица добавляет значения Tx , Ty и Tz к координатам x, y и z всех вершин, сдвигая их в пространстве.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

vertices_cube = np.array([
    [-1, 1, 1, -1, -1, 1, 1, -1],
    [-1, -1, 1, 1, -1, -1, 1, 1],
    [-1, -1, -1, -1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1]
])
faces_cube = np.array([
    [0, 1, 5, 4],
    [1, 2, 6, 5],
    [2, 3, 7, 6],
    [3, 0, 4, 7],
    [0, 1, 2, 3],
    [4, 5, 6, 7]
])
T_x, T_y, T_z = 2, -1, 1
translation_matrix = np.array([
    [1, 0, 0, T_x],
    [0, 1, 0, T_y],
    [0, 0, 1, T_z],
    [0, 0, 0, 1]
])
vertices_translated = translation_matrix @ vertices_cube
fig = plt.figure()
ax = fig.add_subplot(projection='3d', proj_type='ortho')
def draw_shape(vertices, faces, color, alpha=1.0):
    vertices = (vertices[:3, :] / vertices[3, :]).T  # Преобразуем однородные координаты в 3D
    ax.add_collection3d(Poly3DCollection(vertices[faces], facecolors=color, edgecolors='k', linewidths=0.2, alpha=alpha))
draw_shape(vertices_cube, faces_cube, 'blue', alpha=0.3)      # Исходный куб (прозрачный)
draw_shape(vertices_translated, faces_cube, 'red')            # Куб после трансляции

ax.set_box_aspect([1, 1, 1])
ax.set_xlim(-3, 5); ax.set_ylim(-3, 3); ax.set_zlim(-3, 3)
ax.view_init(azim=-37.5, elev=30)
ax.set_xticks(np.linspace(-3, 5, 9)); ax.set_yticks(np.linspace(-3, 3, 7))
ax.set_zticks(np.linspace(-3, 3, 7))

plt.show()
```

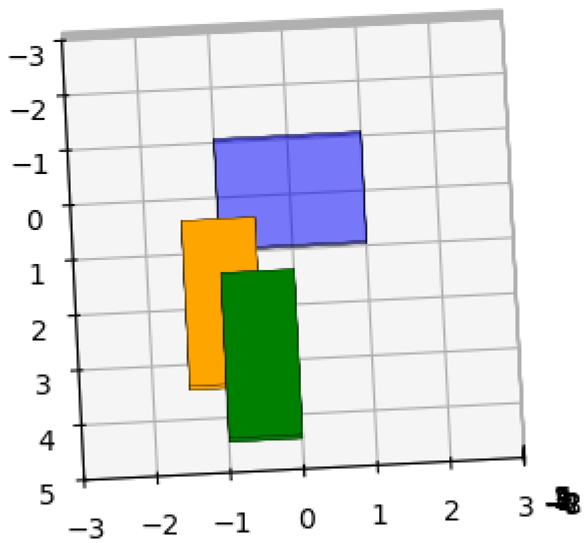
translation_matrix задает матрицу трансляции, которая сдвигает куб на Tx=2 по оси x, Ty=−1 по оси y, и Tz=1 по оси z.
vertices_translated — результат умножения матрицы трансляции на вершины куба, представляющий новый набор координат после трансляции.

Мы отображаем исходный куб (синий) и сдвинутый куб (красный) на одном графике для наглядного сравнения.



Теперь мы попробуем комбинировать матрицы трансляции и масштабирования, применяя их в разном порядке, и исследуем, как это влияет на куб.

1. **TS**: сначала выполняется масштабирование, затем — трансляция.
2. **ST**: сначала выполняется трансляция, затем — масштабирование.



Синий куб — исходный размер и положение.
Оранжевый куб (TS) — сначала масштабируется, затем перемещается. В этом случае масштабирование происходит относительно центра координат, а затем куб перемещается, поэтому он будет пропорционально увеличен и перемещен.

Зеленый куб (ST) — сначала перемещается, затем масштабируется. В этом случае перемещенная фигура масштабируется относительно новой позиции, поэтому её центр смещен относительно исходной точки, а также изменен её размер.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
vertices_cube = np.array([
    [-1, 1, 1, -1, -1, 1, 1, -1],
    [-1, -1, 1, 1, -1, -1, 1, 1],
    [-1, -1, -1, -1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1]
])
faces_cube = np.array([
    [0, 1, 5, 4],
    [1, 2, 6, 5],
    [2, 3, 7, 6],
    [3, 0, 4, 7],
    [0, 1, 2, 3],
    [4, 5, 6, 7]
])
S_x, S_y, S_z = 1.5, 0.5, 2
scale_matrix = np.array([
    [S_x, 0, 0, 0],
    [0, S_y, 0, 0],
    [0, 0, S_z, 0],
    [0, 0, 0, 1]
])
T_x, T_y, T_z = 2, -1, 1
translation_matrix = np.array([
    [1, 0, 0, T_x],
    [0, 1, 0, T_y],
    [0, 0, 1, T_z],
    [0, 0, 0, 1]
])
vertices_TS = translation_matrix @ (scale_matrix @ vertices_cube)
vertices_ST = scale_matrix @ (translation_matrix @ vertices_cube)
fig = plt.figure()
ax = fig.add_subplot(projection='3d', proj_type='ortho')
def draw_shape(vertices, faces, color, alpha=1.0):
    vertices = (vertices[:3, :] / vertices[3, :]).T  # Преобразуем однородные координаты в 3D
    ax.add_collection3d(Poly3DCollection(vertices[faces], facecolors=color, edgecolors='k', linewidths=0.2, alpha=alpha))
draw_shape(vertices_cube, faces_cube, 'blue', alpha=0.3)       # Исходный куб (прозрачный)
draw_shape(vertices_TS, faces_cube, 'orange')                  # Куб после TS
draw_shape(vertices_ST, faces_cube, 'green')                   # Куб после ST
ax.set_box_aspect([1, 1, 1])
ax.set_xlim(-3, 5); ax.set_ylim(-3, 3); ax.set_zlim(-3, 5)
ax.view_init(azim=-37.5, elev=30)
ax.set_xticks(np.linspace(-3, 5, 9)); ax.set_yticks(np.linspace(-3, 3, 7))
ax.set_zticks(np.linspace(-3, 5, 9))
plt.show()
```

Эквивалентны ли такие преобразования?

Нет, последовательности **TS** и **ST** не эквивалентны. Причина в том, что:

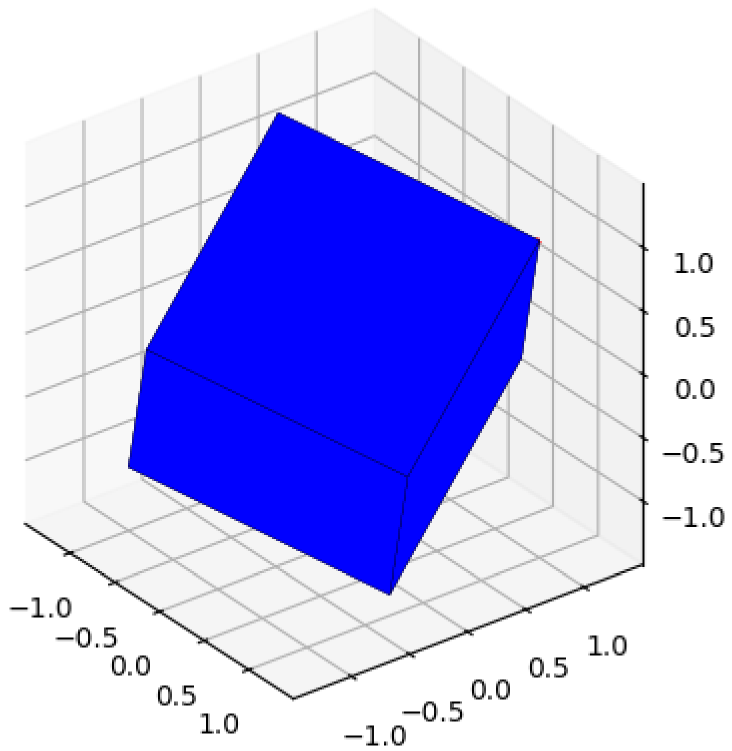
При **TS** масштабирование происходит вначале относительно начала координат, а затем объект перемещается.

При **ST** объект сначала перемещается, а затем масштабируется относительно новой позиции, что приводит к другому результату.

Таким образом, порядок выполнения преобразований влияет на конечный результат, и матрицы **TS** и **ST** не дают эквивалентные преобразования.

Задание 4. Выполните вращение кубика.

1. **Вектор вращения v** задаёт ось, вокруг которой мы поворачиваем куб.
2. **Матрица J** создаётся на основе v и определяет вращение вокруг этой оси.
3. **Матрица поворота $R_v(\theta)=e^{J}$ θ**— описывает поворот куба на угол θ вокруг оси v.
4. Если v направлен вдоль x, y или z, получаем стандартные матрицы вращения $R_x(\theta)$, $R_y(\phi)$, $R_z(\psi)$.
5. **Матрицы** $R_x(\theta)$, $R_y(\phi)$, $R_z(\psi)$достаточны для описания любого вращения в 3D (Теорема Эйлера).
6. Зная матрицу вращения, можно восстановить ось вращения.



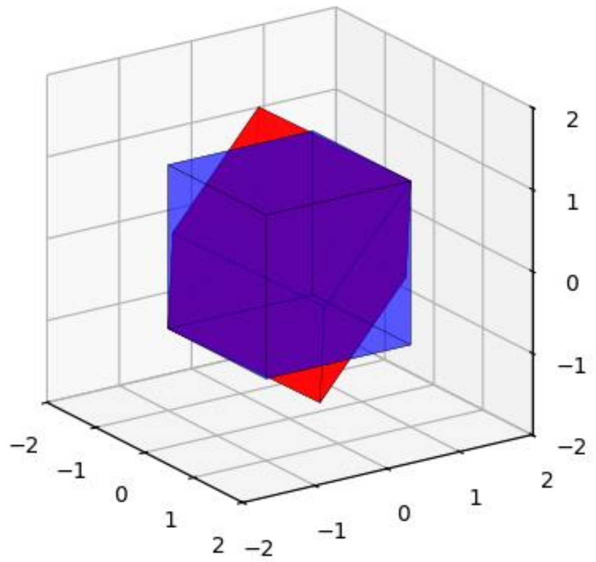
Задание 5. Вращение кубика вокруг одной вершины.

В этом задании требуется:

Найти матрицу вращения, чтобы кубик вращался вокруг одной из своих вершин.

Применить это вращение и построить график, где будут показаны исходный и повернутый кубик.

Чтобы вращать куб вокруг вершины, нужно сначала сдвинуть куб так, чтобы эта вершина стала центром координат (применить трансляцию), затем повернуть его (матрицей поворота), и вернуть обратно. Эти шаги можно выразить как последовательность операций: сдвиг, поворот, обратный сдвиг.



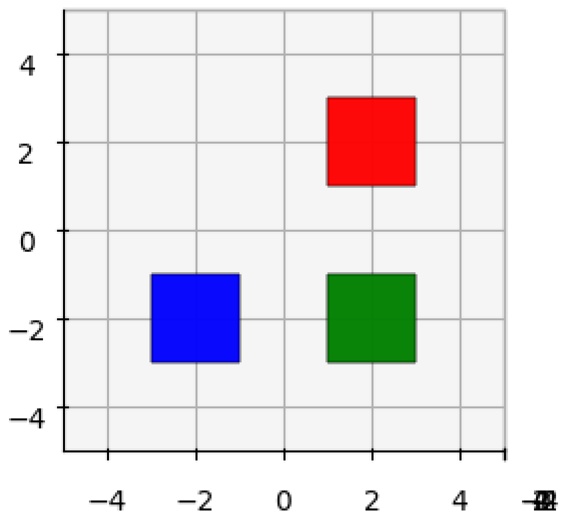
Задание 6. Реализация камеры.

Здесь задача — создать «сцену» с несколькими кубиками, используя команды камеры, например, view в MATLAB или ax.view_init в Python.

Нужно выбрать матрицы T_c и $R_c(\theta)$, которые определят позицию и поворот камеры, чтобы смотреть на сцену с некоторого расстояния и под углом.

Затем требуется найти матрицу C−1, которая перемещает и поворачивает камеру обратно в стандартное положение. Применяя C−1 ко всем объектам сцены, создаётся вид «съёмки с камеры».

Эффект — изменение восприятия сцены, так как камера будет показывать объекты с определённого ракурса, создавая иллюзию трёхмерности.

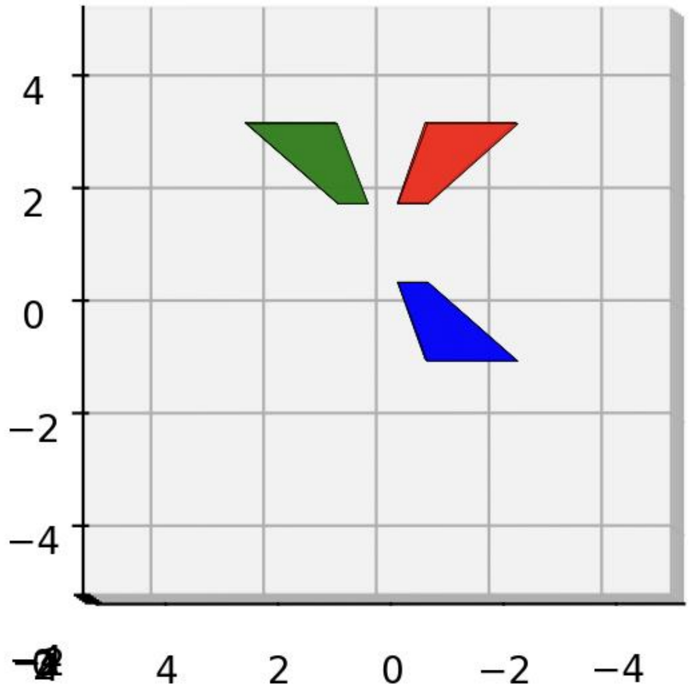


Задание 7. Реализация перспективы.

Матрица перспективной проекции P используется для отображения трёхмерного пространства на двумерную плоскость экрана. Она принимает во внимание глубину (ось z) и создает эффект перспективы — объекты дальше от камеры выглядят меньше.

P = $\begin{bmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_f + z_n}{z_n - z_f} & \frac{2 * z_f * z_n}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$ - матрица перспективной проекции P,

- f – фокусное расстояние
- a – соотношение сторон экрана
- z_n – ближняя граница видимой области
- z_f – дальняя граница видимой области



Задание 8. Построение домика.

Матрица масштабирования служит для изменения размеров объекта по осям x, y, z и задается следующим образом:

S $(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, где s_x, s_y, s_z – коэффициенты масштабирования вдоль осей.

Матрица поворота на угол α:

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

Матрица перемещения служит для сдвига объекта в пространстве на заданные значения t_x, t_y, t_z и задается следующим образом:

T $(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$, где t_x, t_y, t_z – величины сдвига по осям.