Компьютерная геометрия и геометрическое моделирование

Лабораторная работа №3

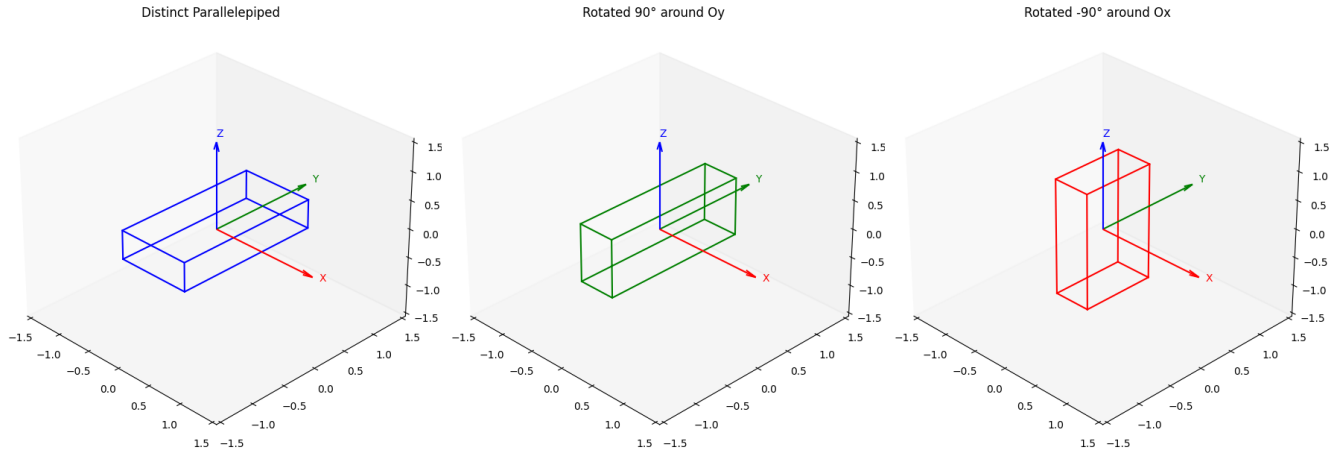
- Ф.И.О: **Мухамедияр Адиль**
- Ноиер студ. билета: **1032205725**
- Группа: **НКНбд-01-20**


```
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon, RegularPolygon
import numpy as np
```

№1

```
def plot_axes(ax, length=1.5):
    """Plot 3D axes on the given ax."""
    # X axis (red)
    ax.quiver(0, 0, 0, length, 0, 0, color='red', arrow_length_ratio=0.1)
    ax.text(length + 0.1, 0, 0, 'X', color='red')

    # Y axis (green)
    ax.quiver(0, 0, 0, 0, length, 0, color='green', arrow_length_ratio=0.1)
    ax.text(0, length + 0.1, 0, 'Y', color='green')

    # Z axis (blue)
    ax.quiver(0, 0, 0, 0, 0, length, color='blue', arrow_length_ratio=0.1)
    ax.text(0, 0, length + 0.1, 'Z', color='blue')
theta_y = np.radians(90)
theta_x = np.radians(-90)

# Матрица поворота вокруг оси Oy в однородных координатах
def homogeneous_rotation_matrix_around_y(theta):
    c, s = np.cos(theta), np.sin(theta)
    return np.array([
        [c, 0, s, 0],
        [0, 1, 0, 0],
        [-s, 0, c, 0],
        [0, 0, 0, 1]
    ])

# Матрица поворота вокруг оси Ox в однородных координатах
def homogeneous_rotation_matrix_around_x(theta):
    c, s = np.cos(theta), np.sin(theta)
    return np.array([
        [1, 0, 0, 0],
        [0, c, -s, 0],
        [0, s, c, 0],
        [0, 0, 0, 1]
    ])

# Функция для построения параллелепипеда по его вершинам
def plot_parallelepiped(vertices, ax, color='blue'):
    edges = [
        [vertices[j] for j in [0, 1, 3, 2, 0, 4, 5, 7, 6, 4]], # нижняя и верхняя грани
        [vertices[j] for j in [5, 1]], # вертикальные ребра
        [vertices[j] for j in [7, 3]],
        [vertices[j] for j in [6, 2]]
    ]
    for edge in edges:
        edge = np.array(edge)
        ax.plot(edge[:, 0], edge[:, 1], edge[:, 2], color=color)

# Изменяем исходные вершины для формирования отдельной формы параллелепипеда
# Размеры: 1x2x0.5, расположенный в центре координат
vertices_distinct = np.array([
    [-0.5, -1, -0.25, 1],  # A
    [-0.5, -1, 0.25, 1],   # B
    [-0.5, 1, -0.25, 1],   # C
    [-0.5, 1, 0.25, 1],    # D
    [0.5, -1, -0.25, 1],   # E
    [0.5, -1, 0.25, 1],    # F
    [0.5, 1, -0.25, 1],    # G
    [0.5, 1, 0.25, 1]      # H
])
```

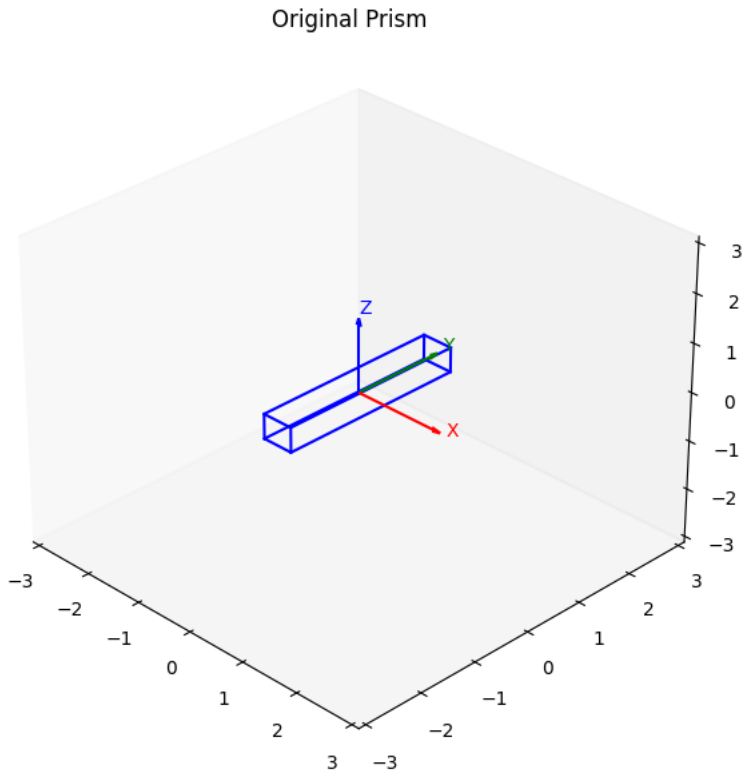
```
# Применяем повороты к отдельному параллелепипеду
vertices_rotated_y_distinct = vertices_distinct.dot(homogeneous_rotation_matrix_around_y(theta_y))
vertices_rotated_x_distinct = vertices_rotated_y_distinct.dot(homogeneous_rotation_matrix_around_x(theta_x))

# Удаляем однородные координаты для построения
vertices_distinct = vertices_distinct[:, :3]
vertices_rotated_y_distinct = vertices_rotated_y_distinct[:, :3]
vertices_rotated_x_distinct = vertices_rotated_x_distinct[:, :3]

# Визуализация с новой формой
fig = plt.figure(figsize=(18, 6))
ax1 = fig.add_subplot(131, projection='3d')
ax2 = fig.add_subplot(132, projection='3d')
ax3 = fig.add_subplot(133, projection='3d')

plot_parallelepiped(vertices_distinct, ax1)
ax1.set_title('Distinct Parallelepiped')
plot_axes(ax1)

plot_parallelepiped(vertices_rotated_y_distinct, ax2, color='green')
ax2.set_title('Rotated 90° around Oy')
plot_axes(ax2)

plot_parallelepiped(vertices_rotated_x_distinct, ax3, color='red')
ax3.set_title('Rotated -90° around Ox')
plot_axes(ax3)

for ax in [ax1, ax2, ax3]:
    ax.view_init(30, -45)
    ax.set_xlim([-1.5, 1.5])
    ax.set_ylim([-1.5, 1.5])
    ax.set_zlim([-1.5, 1.5])
    ax.grid(False)

plt.tight_layout()
plt.show()
```



№ 2

```
# Координаты вершин для призмы из таблицы
vertices_distinct_prism = np.array([
    [-0.25, -1.5, -0.25, 1],  # A
    [-0.25, -1.5, 0.25, 1],   # B
    [-0.25, 1.5, -0.25, 1],   # C
    [-0.25, 1.5, 0.25, 1],    # D
    [0.25, -1.5, -0.25, 1],   # E
    [0.25, -1.5, 0.25, 1],    # F
    [0.25, 1.5, -0.25, 1],    # G
    [0.25, 1.5, 0.25, 1]      # H
])
```

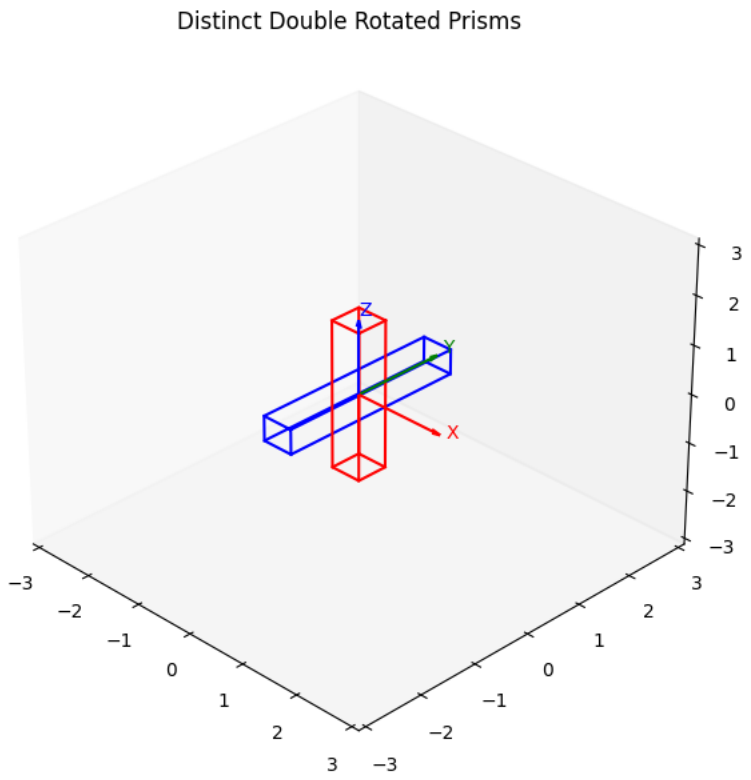
```
# Визуализация исходной призмы
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')

plot_parallelepiped(vertices_distinct_prism, ax)
ax.set_title('Original Prism')
plot_axes(ax)

ax.view_init(30, -45)
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_zlim([-3, 3])
ax.grid(False)

plt.tight_layout()
plt.show()
```




```
# Размеры: 0.5x3x0.5, с центром в начале координат
vertices_distinct_prism = np.array([
    [-0.25, -1.5, -0.25, 1],  # A
    [-0.25, -1.5, 0.25, 1],   # B
    [-0.25, 1.5, -0.25, 1],   # C
    [-0.25, 1.5, 0.25, 1],    # D
    [0.25, -1.5, -0.25, 1],   # E
    [0.25, -1.5, 0.25, 1],    # F
    [0.25, 1.5, -0.25, 1],    # G
    [0.25, 1.5, 0.25, 1]      # H
])
```

```
# Размеры: 0.5x3x0.5, с центром в начале координат
vertices_distinct_prism = np.array([
    [-0.25, -1.5, -0.25, 1],  # A
    [-0.25, -1.5, 0.25, 1],   # B
    [-0.25, 1.5, -0.25, 1],   # C
    [-0.25, 1.5, 0.25, 1],    # D
    [0.25, -1.5, -0.25, 1],   # E
    [0.25, -1.5, 0.25, 1],    # F
    [0.25, 1.5, -0.25, 1],    # G
    [0.25, 1.5, 0.25, 1]      # H
])

# Поворот особой призмы на +90° вокруг оси Oy
vertices_rotated_y_distinct_prism = vertices_distinct_prism.dot(homogeneous_rotation_matrix_around_y(np.radians(90)))

# Поворот призмы на +90° вокруг оси Ox после предыдущего поворота
vertices_double_rotated_distinct_prism = vertices_rotated_y_distinct_prism.dot(homogeneous_rotation_matrix_around_x(np.radians(90)))
```

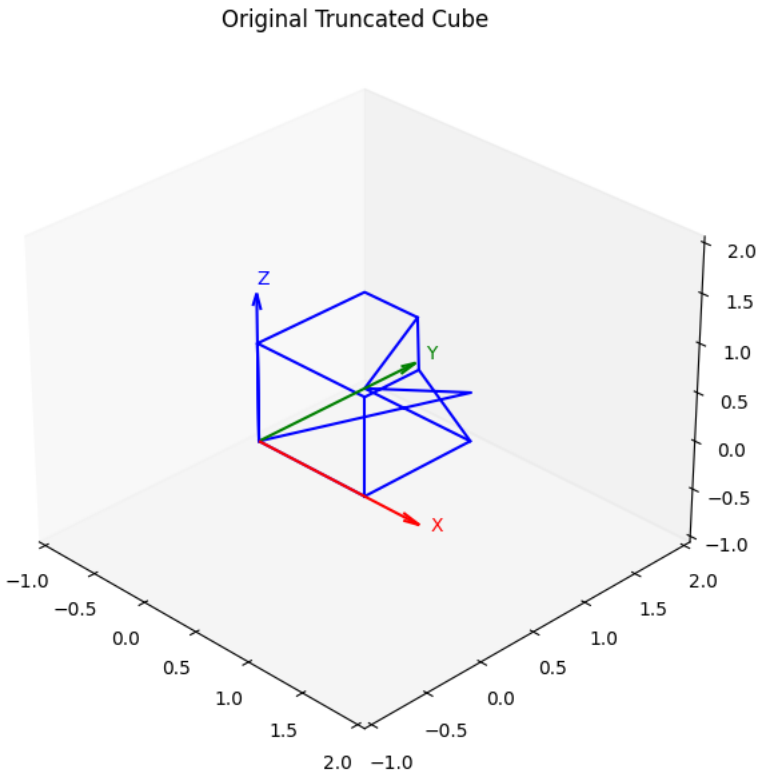
```
# Удаление однородных координат для отображения
vertices_rotated_y_distinct_prism = vertices_rotated_y_distinct_prism[:, :3]
vertices_double_rotated_distinct_prism = vertices_double_rotated_distinct_prism[:, :3]

# Визуализация повернутых особых призм
fig, ax = plt.subplots(1, 1, figsize=(6, 6), subplot_kw={'projection': '3d'})

plot_parallelepiped(vertices_rotated_y_distinct_prism, ax, color='blue')
plot_parallelepiped(vertices_double_rotated_distinct_prism, ax, color='red')
ax.set_title('Distinct Double Rotated Prisms')
plot_axes(ax)

ax.view_init(30, -45)
ax.set_xlim([-3, 3])
ax.set_ylim([-3, 3])
ax.set_zlim([-3, 3])
ax.grid(False)

plt.tight_layout()
plt.show()
```



№3

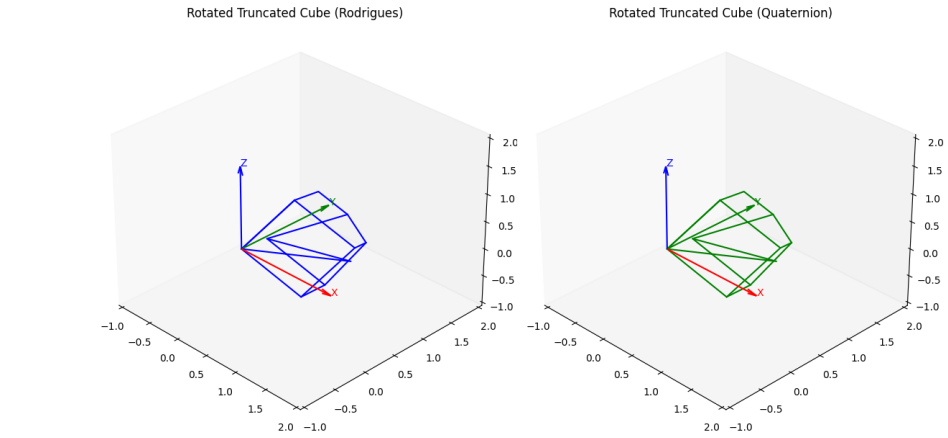
```
# Координаты вершин для усеченного куба из таблицы
vertices_truncated_cube = np.array([
    [0, 0, 1],    # 1
    [1, 0, 1],    # 2
    [1, 0.5, 1],  # 3
    [0.5, 1, 1],  # 4
    [0, 1, 1],    # 5
    [0, 0, 0],    # 6
    [1, 0, 0],    # 7
    [1, 1, 0],  # 8
    [0, 1, 0],  # 9
    [1, 1, 0.5]     # 10
])

# Визуализация исходного усеченного куба
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')

# Использование модифицированной функции построения графика для визуализации усеченного куба
def plot_truncated_cube(vertices, ax, color='blue'):
    edges = [
        [vertices[j] for j in [0, 1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 5]],
        [vertices[j] for j in [2, 7]],
```

```
            [vertices[j] for j in [3, 8]],
            [vertices[j] for j in [1, 6]],
            [vertices[j] for j in [0, 5]]
        ]
        for edge in edges:
            edge = np.array(edge)
            ax.plot(edge[:, 0], edge[:, 1], edge[:, 2], color=color)

plot_truncated_cube(vertices_truncated_cube, ax)
ax.set_title('Original Truncated Cube')
plot_axes(ax)

ax.view_init(30, -45)
ax.set_xlim([-1, 2])
ax.set_ylim([-1, 2])
ax.set_zlim([-1, 2])
ax.grid(False)

plt.tight_layout()
plt.show()
```



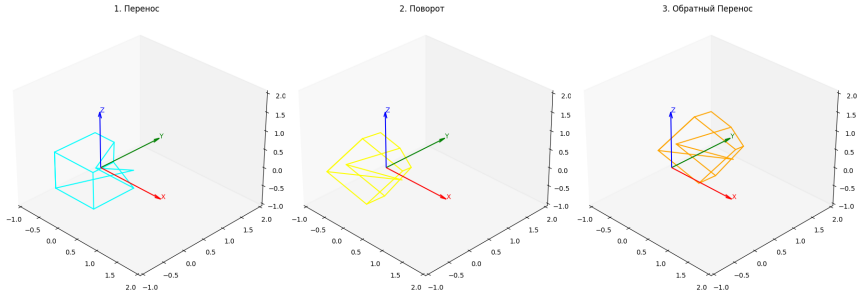

```
# Извлечение координат требуемых вершин
v5 = vertices_truncated_cube[4]
v2 = vertices_truncated_cube[1]
v3 = vertices_truncated_cube[2]
v9 = vertices_truncated_cube[8]

# Нахождение центра грани с вершинами 2, 3 и 9
center_face = (v2 + v3 + v9) / 3

# Определение направления оси вращения
rotation_axis = center_face - v5
rotation_axis = rotation_axis / np.linalg.norm(rotation_axis)  # Нормализация оси вращения

# Ротация Родригеса
def rodrigues_rotation(v, k, theta):
    """Rotate vector v by angle theta around axis k using Rodrigues' formula."""
    theta_rad = np.deg2rad(theta)
    v_rot = v * np.cos(theta_rad) + np.cross(k, v) * np.sin(theta_rad) + k * np.dot(k, v) * (1 - np.cos(theta_rad))
    return v_rot

# Вращение кватерниона
def quaternion_multiply(q1, q2):
    """Multiply two quaternions."""
    w1, x1, y1, z1 = q1
    w2, x2, y2, z2 = q2
    w = w1 * w2 - x1 * x2 - y1 * y2 - z1 * z2
    x = w1 * x2 + x1 * w2 + y1 * z2 - z1 * y2
```

```
    y = w1 * y2 + y1 * w2 + z1 * x2 - x1 * z2
    z = w1 * z2 + z1 * w2 + x1 * y2 - y1 * x2
    return w, x, y, z

def quaternion_conjugate(q):
    """Return the conjugate of a quaternion."""
    w, x, y, z = q
    return w, -x, -y, -z

def rotate_vector_using_quaternion(v, k, theta):
    """Rotate vector v by angle theta around axis k using quaternions."""
    theta_rad = np.deg2rad(theta) / 2
    q = (np.cos(theta_rad), *(k * np.sin(theta_rad)))
    v_quaternion = (0, *v)
    v_rot_quaternion = quaternion_multiply(quaternion_multiply(q, v_quaternion), quaternion_conjugate(q))
    return np.array(v_rot_quaternion[1:])

# Поворот вершин усеченного куба на -45 градусов вокруг оси вращения
vertices_rotated = np.array([rodrigues_rotation(v, rotation_axis, -45) for v in vertices_truncated_cube])
vertices_rotated_quaternion = np.array([rotate_vector_using_quaternion(v, rotation_axis, -45) for v in vertices_truncated_cube])

# Визуализация
def plot_axes(ax):
    """Plot the 3D axes for reference."""
    ax.quiver(0, 0, 0, 1.5, 0, 0, color='r', arrow_length_ratio=0.1)
    ax.quiver(0, 0, 0, 0, 1.5, 0, color='g', arrow_length_ratio=0.1)
    ax.quiver(0, 0, 0, 0, 0, 1.5, color='b', arrow_length_ratio=0.1)
    ax.text(1.5, 0, 0, 'X', color='r')
    ax.text(0, 1.5, 0, 'Y', color='g')
    ax.text(0, 0, 1.5, 'Z', color='b')

def plot_truncated_cube(vertices, ax, color='blue'):
    edges = [
        [vertices[j] for j in [0, 1, 2, 3, 4, 0, 5, 6, 7, 8, 9, 5]],
        [vertices[j] for j in [2, 7]],
        [vertices[j] for j in [3, 8]],
        [vertices[j] for j in [1, 6]],
        [vertices[j] for j in [0, 5]]
    ]
    for edge in edges:
        edge = np.array(edge)
        ax.plot(edge[:, 0], edge[:, 1], edge[:, 2], color=color)

# Визуализация повернутого усеченного куба
fig = plt.figure(figsize=(12, 6))

# Используя вращение Родригеса
ax1 = fig.add_subplot(121, projection='3d')
plot_truncated_cube(vertices_rotated, ax1, color='blue')
plot_axes(ax1)
ax1.set_title('Rotated Truncated Cube (Rodrigues)')
ax1.view_init(30, -45)
ax1.set_xlim([-1, 2])
ax1.set_ylim([-1, 2])
ax1.set_zlim([-1, 2])
ax1.grid(False)

# Использование вращения кватернионов
ax2 = fig.add_subplot(122, projection='3d')
plot_truncated_cube(vertices_rotated_quaternion, ax2, color='green')
plot_axes(ax2)
ax2.set_title('Rotated Truncated Cube (Quaternion)')
ax2.view_init(30, -45)
ax2.set_xlim([-1, 2])
ax2.set_ylim([-1, 2])
ax2.set_zlim([-1, 2])
ax2.grid(False)

plt.tight_layout()
plt.show()
```



№4

```
# Определим ось вращения и выполним все необходимые шаги

# Определение оси вращения
rotation_axis_2 = center_face - vertices_truncated_cube[4]
rotation_axis_2 = rotation_axis_2 / np.linalg.norm(rotation_axis_2)  # Нормализация

# 1. Перенос: переместим куб так, чтобы центр грани был в начале координат
translated_vertices_2 = vertices_truncated_cube - center_face

# 2. Поворот: применяем формулу Родрига
rotated_translated_vertices_2 = np.array([rodrigues_rotation(v, rotation_axis_2, -45) for v in translated_vertices_2])

# 3. Обратный перенос
final_rotated_vertices_2 = rotated_translated_vertices_2 + center_face

# Визуализация каждого шага
fig = plt.figure(figsize=(18, 6))

# 1. Перенос
ax1 = fig.add_subplot(131, projection='3d')
plot_truncated_cube(translated_vertices_2, ax1, color='cyan')
plot_axes(ax1)
ax1.set_title('1. Перенос')
ax1.view_init(30, -45)
ax1.set_xlim([-1, 2])
ax1.set_ylim([-1, 2])
ax1.set_zlim([-1, 2])
ax1.grid(False)

# 2. Поворот
ax2 = fig.add_subplot(132, projection='3d')
plot_truncated_cube(rotated_translated_vertices_2, ax2, color='yellow')
plot_axes(ax2)
ax2.set_title('2. Поворот')
ax2.view_init(30, -45)
ax2.set_xlim([-1, 2])
ax2.set_ylim([-1, 2])
ax2.set_zlim([-1, 2])
ax2.grid(False)

# 3. Обратный перенос
ax3 = fig.add_subplot(133, projection='3d')
plot_truncated_cube(final_rotated_vertices_2, ax3, color='orange')
plot_axes(ax3)
ax3.set_title('3. Обратный Перенос')
ax3.view_init(30, -45)
ax3.set_xlim([-1, 2])
ax3.set_ylim([-1, 2])
ax3.set_zlim([-1, 2])
ax3.grid(False)

plt.tight_layout()
plt.show()
```


1. Перенос
2. Поворот
3. Обратный Перенос