

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

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Лабораторная работа № 3 "Матрицы в 3D-графике "

по дисциплине Практическая линейная алгебра

Выполнила: студентка гр. **R3238**

Нечаева А. А.

Преподаватель: *Перегудин Алексей Алексеевич*

Санкт-Петербург, 2023-2024

1 Задание. Создайте кубик.

1.1 Как работает код?

В первой части кода (рисунок 1) задаются координаты вершин куба: каждый столбец – вершина и сверху вниз в нем заданы координаты x , y , z в пространстве и w (последняя отвечает за перспективу).

```
1 verticesCube = [  
2     -1, 1, 1, -1, -1, 1, 1, -1;  
3     -1, -1, 1, 1, -1, -1, 1, 1;  
4     -1, -1, -1, -1, 1, 1, 1, 1;  
5     1, 1, 1, 1, 1, 1, 1, 1  
6 ];  
7
```

Рис. 1. Исходный код кубика, часть 1.

Вторая часть (рисунок 2) отвечает за задание плоскостей граней куба, в каждой строчке записаны 4 вершины куба, по которым строится грань.

```
8 facesCube = [  
9     1, 2, 6, 5;  
10    2, 3, 7, 6;  
11    3, 4, 8, 7;  
12    4, 1, 5, 8;  
13    1, 2, 3, 4;  
14    5, 6, 7, 8  
15 ];  
16
```

Рис. 2. Исходный код кубика, часть 2.

Функция *DrawShape* отвечает за отрисовку кубика, сначала строятся точки вершин по 3 координатам и с учетом перспективы, затем изображаются грани.

```
16
17 DrawShape(verticesCube, facesCube, 'blue')
18 axis equal;
19 view(3);
20
21 function DrawShape(vertices, faces, color)
22     patch('Vertices', (vertices(1:3,:)./vertices(4,:))', 'Faces', faces, '
23         FaceColor', color);
24 end
```

Рис. 3. Исходный код кубика, часть 3.

1.2 Почему используется четырехкомпонентный вектор, а не трех?

Четвертый компонент в векторе позволяет реализовывать перспективную проекцию, а не только отображать ортогональную проекцию. Кроме того, с помощью матрицы 4×4 реализуются такие преобразования как сдвиги, повороты и т.д. в трехмерном пространстве.

1.3 Как задать другие фигуры?

2 Задание. Изменить масштаб кубика.

Для изменения масштаба кубика использовалась матрица вида:

$$\begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Где, a_1 , a_2 , a_3 отвечают за изменение масштаба по x , y и z соответственно.

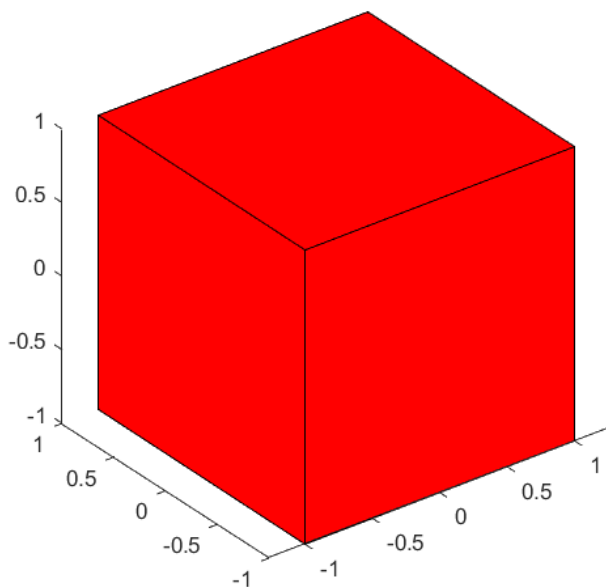


Рис. 4. Оригинальный масштаб, при $a_i = 1$.

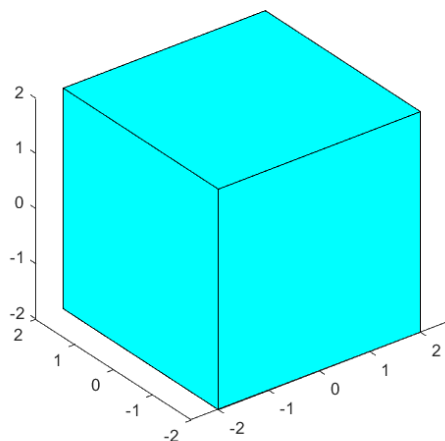


Рис. 5. Результат при $a_i = 2$.

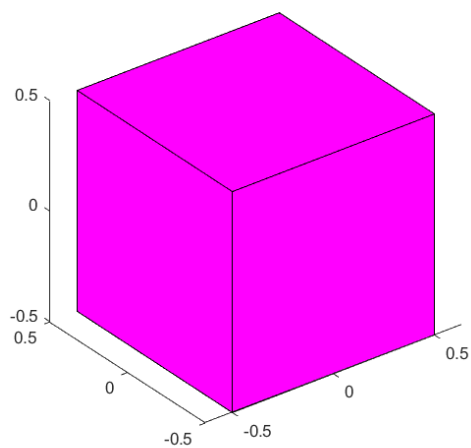


Рис. 6. Результат при $a_i = 0.5$.

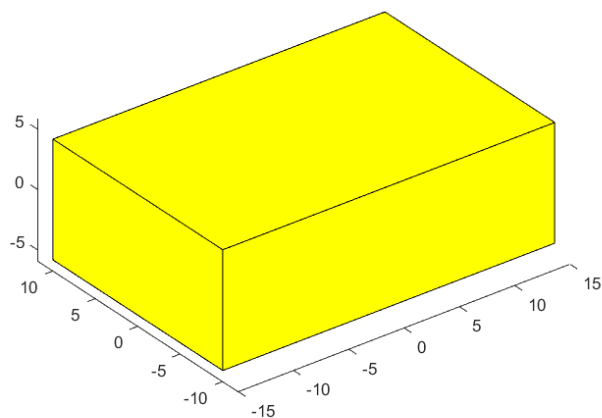


Рис. 7. Результат при $a_1 = 15$, $a_2 = 10$, $a_3 = 5$.

```
sizeMatrix = [  
    15, 0, 0, 0;  
    0, 10, 0, 0;  
    0, 0, 5, 0;  
    0, 0, 0, 1  
];  
  
newVertices = sizeMatrix * verticesCube;  
  
DrawShape (newVertices, facesCube, 'y')
```

Листинг 1. Часть кода, отвечающая за масштабирование кубика.

3 Задание. Переместить кубик.

Для перемещения кубика используется матрица вида:

$$\begin{bmatrix} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Сдвиг осуществляется на вектор (b_1, b_2, b_3) , подобная операция возможна из-за наличия четвертой координаты точки, так как она умножается на b_i и складывается с x , y , или z :

$$\begin{bmatrix} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + b_1 \\ y + b_2 \\ z + b_3 \\ 1 \end{bmatrix} \quad (3)$$

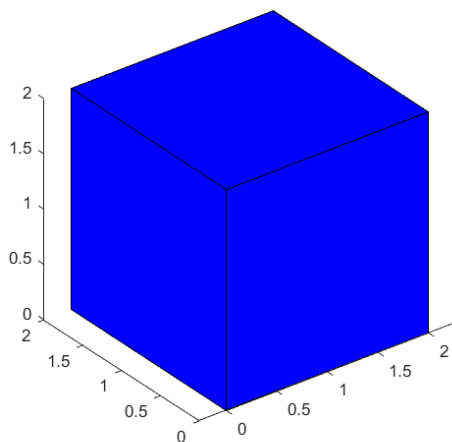


Рис. 8. Сдвиг графика на вектор $(1, 1, 1)$.

```
moveMatrix = [  
    1, 0, 0, 2;  
    0, 1, 0, 3;  
    0, 0, 1, 4;  
    0, 0, 0, 1  
];  
  
newVertices = moveMatrix * verticesCube;  
  
DrawShape (newVertices , facesCube , 'y')
```

Листинг 2. Часть кода, отвечающая за перемещение кубика.

Применим последовательно операции масштабирования и перемещения кубика:

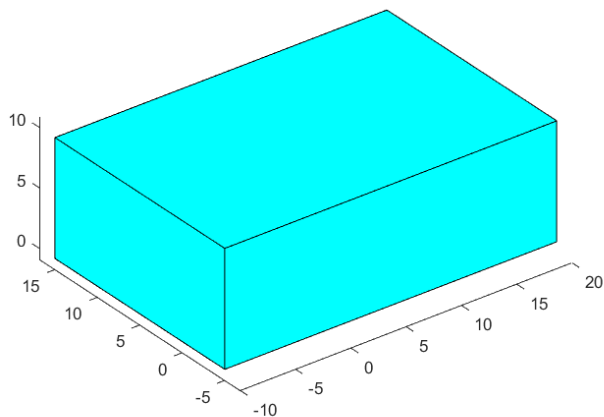


Рис. 9. Масштабирование при $a_1 = 15$, $a_2 = 10$, $a_3 = 5$ и сдвиг графика на вектор $(5, 5, 5)$.


```
moveMatrix = [  
    1, 0, 0, 5;  
    0, 1, 0, 5;  
    0, 0, 1, 5;  
    0, 0, 0, 1  
];  
  
sizeMatrix = [  
    15, 0, 0, 0;  
    0, 10, 0, 0;  
    0, 0, 5, 0;  
    0, 0, 0, 1  
];  
  
newVertices = moveMatrix * sizeMatrix * verticesCube;  
  
DrawShape (newVertices, facesCube, 'c')
```

Листинг 3. Часть кода, отвечающая за масштабирование и перемещение кубика.

Заметим, что для получения корректного результата важен порядок умножения матриц: сначала кубик масштабируется, а затем сдвигается, то есть $newVertices = moveMatrix * (sizeMatrix * verticesCube) = moveMatrix * sizeMatrix * verticesCube$.

Если сначала выполнить сдвиг, а после – масштабирование, то матрица сдвига тоже будет влиять на масштаб результата.

4 Задание. Выполнить вращение кубика.

Матрица вращения определена для каждой оси в 3D пространстве. Пусть угол поворота ϕ .

Матрица вращения вокруг оси X :

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \cos \phi - z \sin \phi \\ y \sin \phi + z \cos \phi \\ 1 \end{bmatrix} \quad (4)$$

Матрица вращения вокруг оси Y :

$$\begin{bmatrix} \cos \phi & 0 & \sin \phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \phi & 0 & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \cos \phi + z \sin \phi \\ y \\ -x \sin \phi + z \cos \phi \\ 1 \end{bmatrix} \quad (5)$$

Матрица вращения вокруг оси Z :

$$\begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \cos \phi - y \sin \phi \\ x \sin \phi + y \cos \phi \\ z \\ 1 \end{bmatrix} \quad (6)$$

Вращение по нескольким осям может привести к *проблеме шарнирного замка*, поэтому обычно используется вращение вокруг конкретной оси, например заданную вектором $\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$, ось должна быть задана единичным вектором.

В данной работе будем проводить вращение относительно осей X , Y , Z :

`phi = pi/3;`

```
rotateXMatrix = [
    1, 0, 0, 0;
    0, cos(phi), -sin(phi), 0;
    0, sin(phi), cos(phi), 0;
    0, 0, 0, 1];
```

```
newVertices = rotateXMatrix * verticesCube;
DrawShape (newVertices, facesCube, 'c')
```

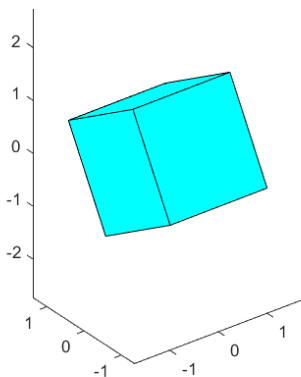


Рис. 10. Вращение кубика вокруг оси X на 60° .

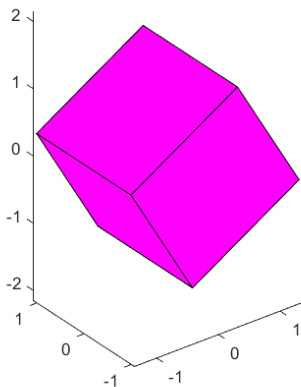


Рис. 11. Вращение кубика вокруг оси Y на 60° .

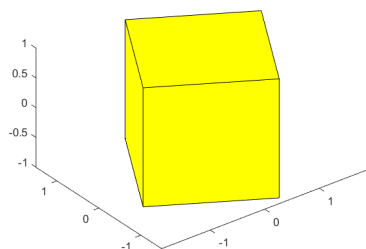


Рис. 12. Вращение кубика вокруг оси Z на 60° .

Листинг 4. Часть кода, вращение кубика вокруг оси X на 60°

Построить комбинации поворотов. Относительно оси X , затем Y :

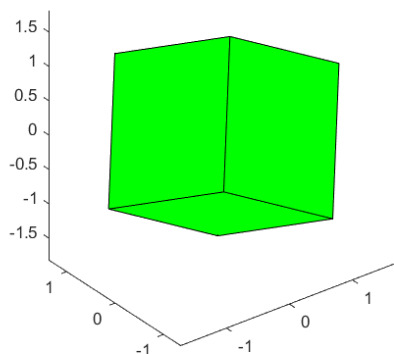


Рис. 13. Вращение кубика вокруг оси X на 60° , затем Y на 30° .

Теперь поменяем порядок:

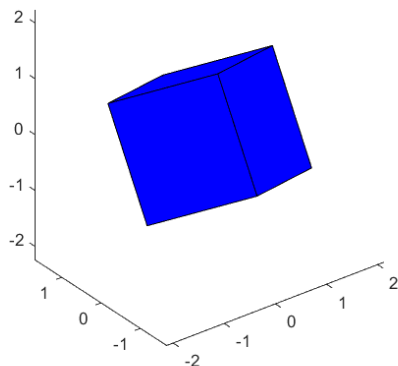


Рис. 14. Вращение кубика вокруг оси Y на 30° , затем X 60° .

Рисунки 13 – 14 иллюстрируют отсутствие коммутативности у преобразований поворота в трехмерном пространстве. То есть порядок вращения вокруг осей имеет значение. Когда мы вращаем куб относительно, например, оси X , его оси Y и Z должны были бы сместиться, но с помощью матриц мы задаем вращение только относительно начальных осей, они не сдвигаются вместе с изменением положения кубика.

5 Задание. Выполнить вращение кубика около одной вершины.

Для того, чтобы выполнить вращение кубика около одной вершины, выполним последовательно операции перемещения кубика так, чтобы вершина, около которой будет происходить вращение оказалась в начале координат, дальше выполним поворот вокруг оси x , y или z , после переместим кубик в исходное положение.

```
phi = pi/3;  
teta = pi/6;
```

```
rotateYMatrix = [  
    cos(teta), 0, sin(teta), 0;  
    0, 1, 0, 0;  
    -sin(teta), 0, cos(teta), 0;  
    0, 0, 0, 1  
];
```

```
rotateZMatrix = [  
    cos(phi), -sin(phi), 0, 0;  
    sin(phi), cos(phi), 0, 0;  
    0, 0, 1, 0;  
    0, 0, 0, 1  
];
```

```
rotateXMatrix = [  
    1, 0, 0, 0;  
    0, cos(phi), -sin(phi), 0;  
    0, sin(phi), cos(phi), 0;  
    0, 0, 0, 1  
];
```

```
moveMatrix1 = [  
    1, 0, 0, 1;  
    0, 1, 0, 1;  
    0, 0, 1, 1;  
    0, 0, 0, 1  
];
```

```
moveMatrix2 = [  
    1, 0, 0, -1;  
    0, 1, 0, -1;  
    0, 0, 1, -1;  
    0, 0, 0, 1  
];  
  
newVertices = moveMatrix1 * rotateXMatrix *  
    moveMatrix2 * verticesCube;  
  
DrawShape (newVertices , facesCube , 'c')
```

Листинг 5. Часть кода, вращение кубика около вершины (1,1,1) вокруг оси параллельной X на 60°

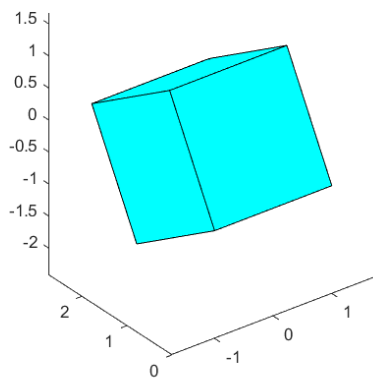
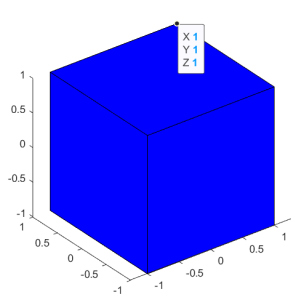
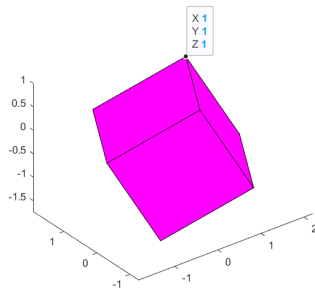


Рис. 15. Вращение кубика около вершины (1,1,1) вокруг оси параллельной X на 60° .



a)



b)

Рис. 16. а) Исходное положение кубика, б) Вращение кубика около вершины (1, 1, 1) вокруг оси параллельной X на 60° , затем $-Z$ на 60° .

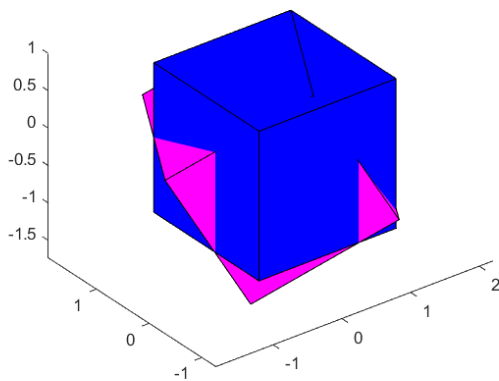


Рис. 17. Оригинал (синий) и кубик после поворота (фиолетовый), случай изображенный на рисунке 16.

6 Задание. Реализация камеры.

Матрица камеры осуществляет преобразование точек из мировых координат в координаты камеры и наоборот. Матрица камеры задается с помощью матрицы смещения и матриц поворота:

```
moveCamera = [  
    1, 0, 0, 2;  
    0, 1, 0, 2;  
    0, 0, 1, 2;  
    0, 0, 0, 1;  
];  
  
phi = 0;  
teta = 0;  
  
rotateYMatrix = [  
    cos(teta), 0, sin(teta), 0;  
    0, 1, 0, 0;  
    -sin(teta), 0, cos(teta), 0;  
    0, 0, 0, 1  
];  
  
rotateZMatrix = [  
    cos(phi), -sin(phi), 0, 0;  
    sin(phi), cos(phi), 0, 0;  
    0, 0, 1, 0;  
    0, 0, 0, 1  
];  
rotateXMatrix = [  
    1, 0, 0, 0;  
    0, cos(phi), -sin(phi), 0;  
    0, sin(phi), cos(phi), 0;  
    0, 0, 0, 1  
];  
  
newVertices = (moveCamera * rotateZMatrix *  
                rotateXMatrix)^(-1) * verticesCube;  
DrawShape (newVertices, facesCube, 'm')
```

Листинг 6. Часть кода, отвечающая за реализацию камеры

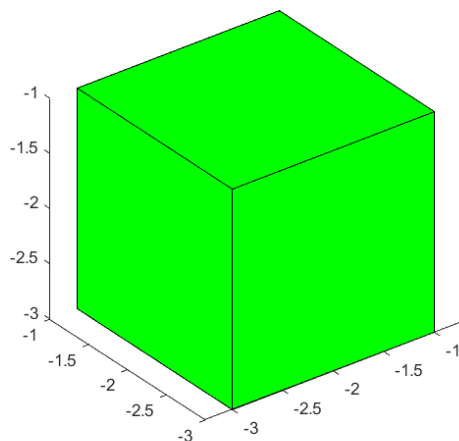


Рис. 18. Реализация камеры (на основе листинга 6).

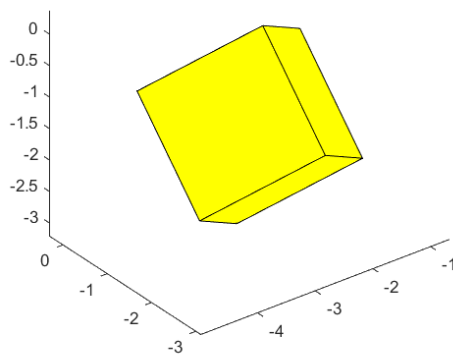


Рис. 19. Реализация камеры на основе листинга 6 при $\phi = \frac{\pi}{4}$, $\theta = \frac{\pi}{4}$.

7 Задание. Реализация перспективы.

8 Задание. * Почти Minecraft.

Для визуализации был написан код на языке *Python* с использованием библиотек *Matplotlib* и *Numpy*.

Код расположен на **GitHub**.

Отражение (симметрию) плоскости относительно прямой $y = ax$, в нашем случае после подстановки $a = 2$, получаем $y = 2x$. Задача – найти матрицу вида: