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).

```
verticesCube = [
    -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
    ];

```

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.

```
facesCube = [
          1, 2, 6, 5;
          2, 3, 7, 6;
          3, 4, 8, 7;
          4, 1, 5, 8;
          1, 2, 3, 4;
          5, 6, 7, 8
         ];
```

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, '
       FaceColor', color);
23 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?

Для того, чтобы задать другие фигуры, можно изменить количество вершин путем добавления матрице $verticesCube$ новых столбцов, также можно менять и расширять матрицу, задающую грани фигуры $facesCube$.

2 Задание. Изменить масштаб кубика.

Для изменения масштаба кубика использовалась матрица вида:

$$\begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & a_2 & 0 & 0 \\ 0 & 0 & a_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

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

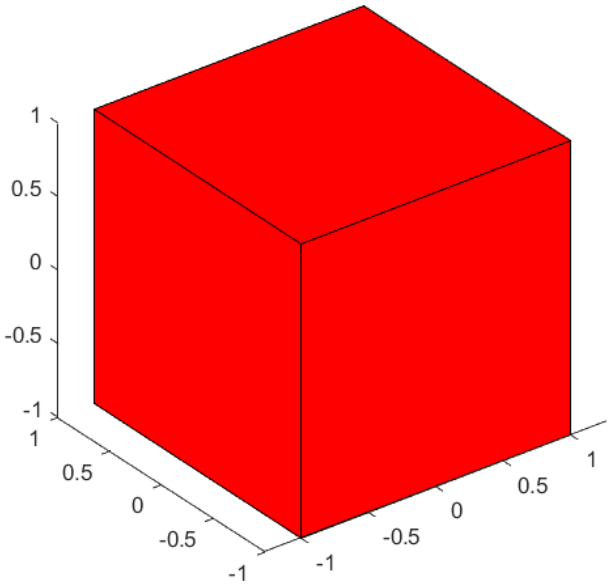


Рис. 4. Оригинальный масштаб,при $a_i = 1$.

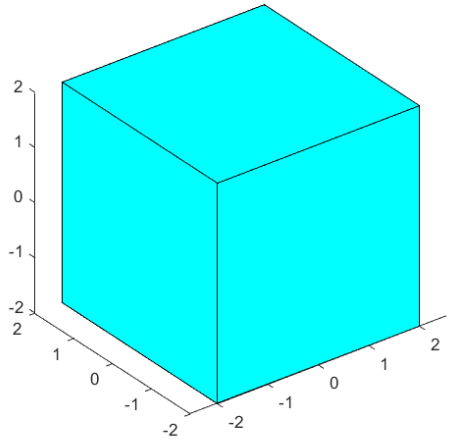


Рис. 5. Результат при $a_i = 2$.

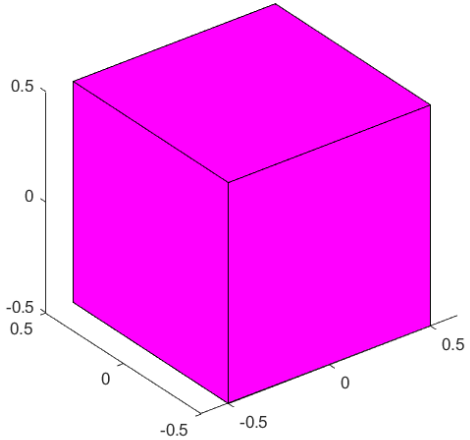


Рис. 6. Результат при $a_i = 0.5$.

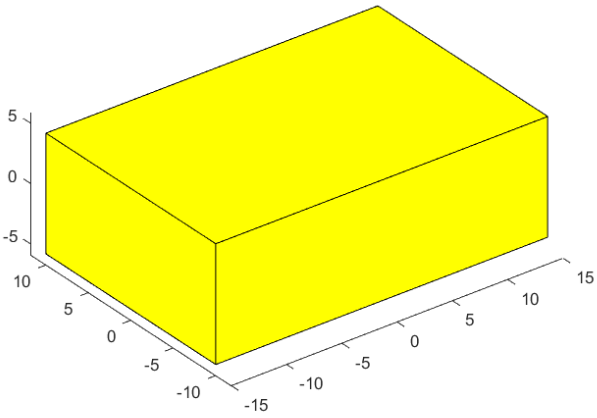


Рис. 7. Результат при $a_1 = 15$, $a_2 = 10$, $a_3 = 5$.

```
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 масшабирование кубика.

3 Задание. Переместить кубик.

Для перемещения кубика используется матрица вида:

$$\begin{bmatrix} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

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:

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

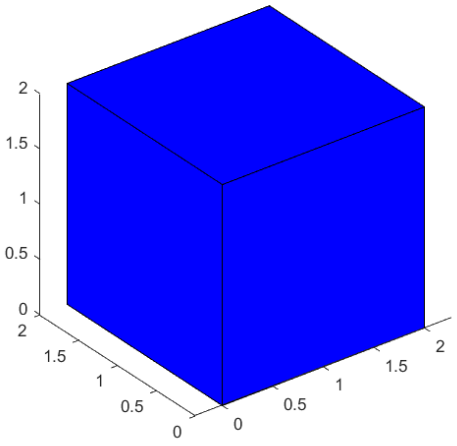


Рис. 8. Сдвиг графика на вектор $(1, 1, 1)$.

```
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:

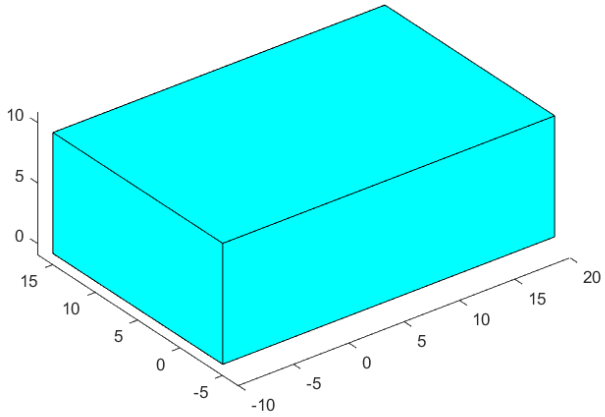


Рис. 9. Масштабирование при $a_1 = 15$, $a_2 = 10$, $a_3 = 5$ *и сдвиг графика на вектор* $(5, 5, 5)$.

```
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 масшабируется, а затем сдвигается, то есть $newVertices = moveMatrix * (sizeMatrix * verticesCube) = moveMatrix * sizeMatrix * verticesCube$.
Если сначала выполнить сдвиг, а после – масшабирование, 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:

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

Матрица вращения вокруг оси Y:

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

Матрица вращения вокруг оси Z:

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

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')
```

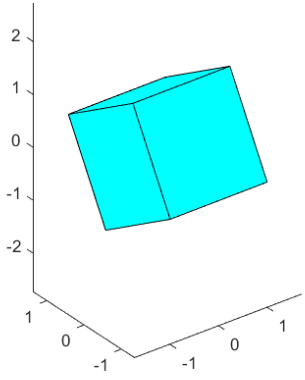


Рис. 10. Вращение кубика вокруг оси X на 60°.

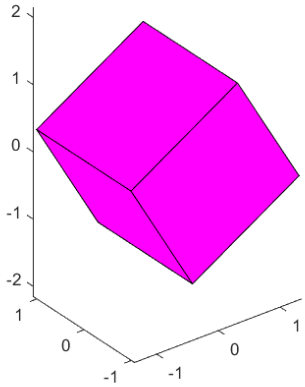


Рис. 11. Вращение кубика вокруг оси Y на 60°.

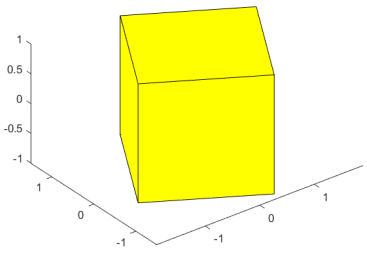


Рис. 12. Вращение кубика вокруг оси Z на 60°.

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

Построить комбинации поворотов. Относительно оси X, затем Y:

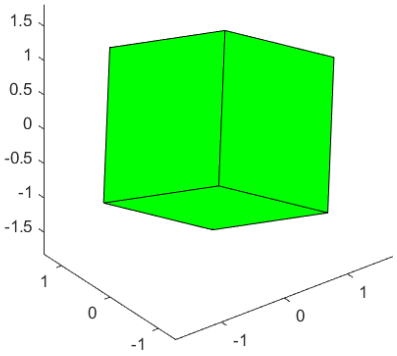


Рис. 13. Вращение кубика вокруг оси X на 60°, затем Y на 30°.

Теперь поменяем порядок:

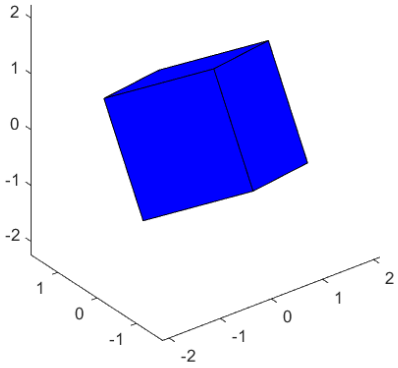


Рис. 14. Вращение кубика вокруг оси Y на 30°, затем X 60°.

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* $60°$

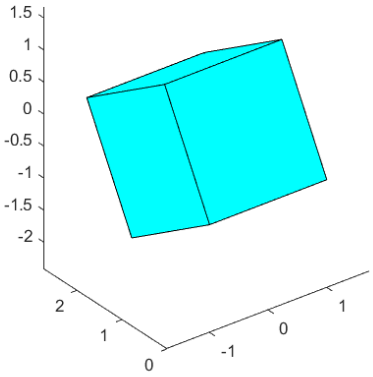
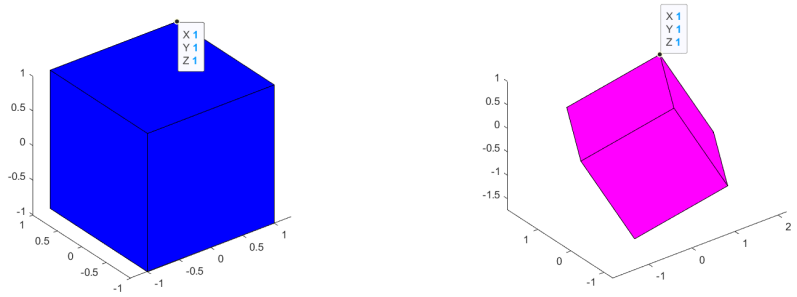


Рис. 15. Вращение кубика около вершины $(1, 1, 1)$ *вокруг оси параллельной* X *на* $60°$.



a) b)

Рис. 16. a) Исходное положение кубика, b) Вращение кубика около вершины $(1, 1, 1)$ *вокруг оси параллельной* X *на 60°, затем – Z на 60°.*

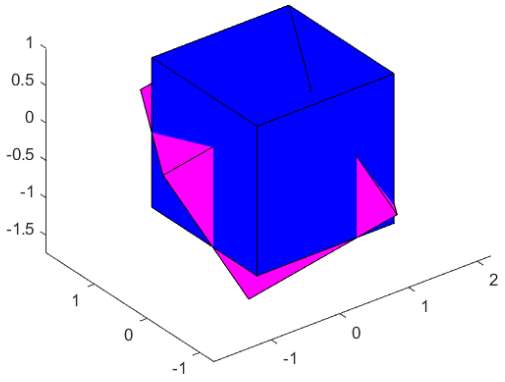


Рис. 17. Оригинал (синий) и кубик после поворота (фиолетовый), случай изображенный на рисунке 16.

6 Задание. Реализация камеры.

Матрица камеры осуществляет преобразование точек из мировых координат в координаты камеры и наоборот. Матрица камеры задается как обратная к действиям камеры, так как мы производим преобразования пространства. То есть, для того, чтобы продемонстрировать тот вид, который мы получим при смещении камеры влево, мы должны преобразовать пространство как будто оно сдвинулось от нас вправо.
Матрицу камеры можно задать так:

$$M_{camera} = (M_{rotation} \cdot M_{move})^{-1}, \tag{7}$$

где $M_{rotation}$ – матрица, задающая поворот камеры, M_{move} – матрица, задающая смещение камеры.

Реализуем камеру для сцены из 3 кубиков (рисунок 18).

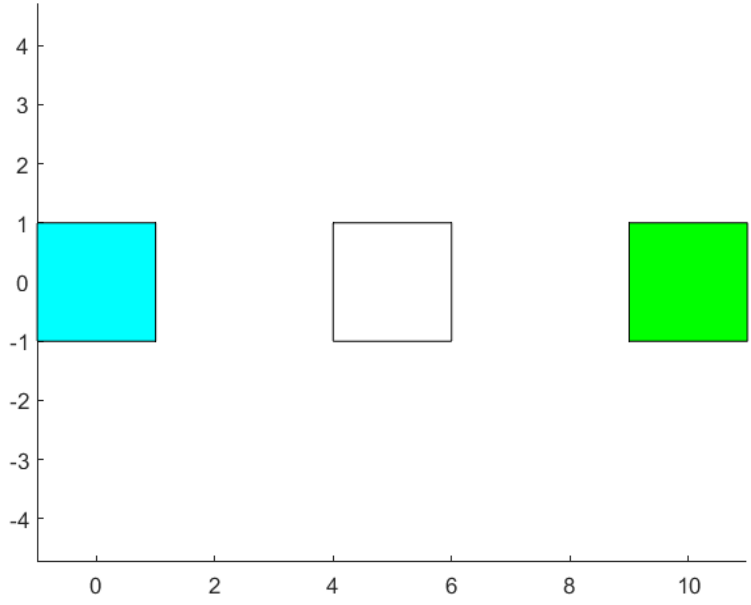


Рис. 18. Исходная сцена при $view(0, 90)$.

Переместим камеру в точку $(1, 2, 3)$.

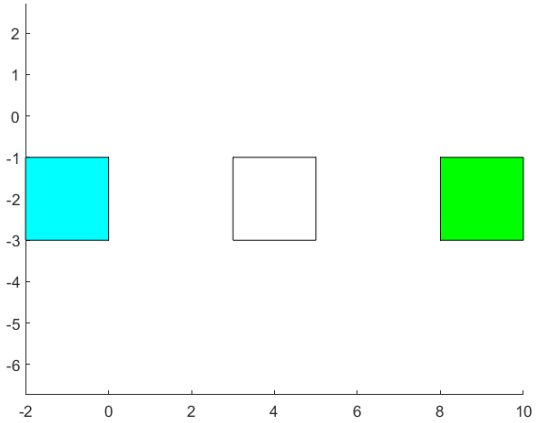


Рис. 19. Сцена после перемещения камеры в точку $(1, 2, 3)$.

Повернем камеру вокруг оси X на $30°$.

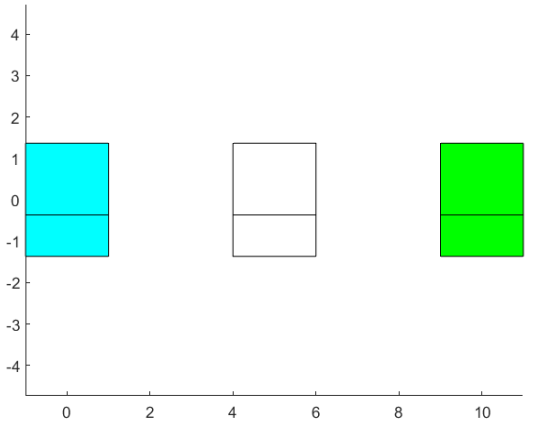


Рис. 20. Сцена после поворота камеры относительно X *на* $30°$.

```
move1 = [
    1, 0, 0, 0;
    0, 1, 0, 0;
    0, 0, 1, 0;
    0, 0, 0, 1;
    ];

teta = pi/4;
rotateYMatrix = [
    cos(teta), 0, sin(teta), 0;
    0, 1, 0, 0;
    -sin(teta), 0, cos(teta), 0;
    0, 0, 0, 1
    ];

phi1 = pi/6;
rotateXMatrix = [
    1, 0, 0, 0;
    0, cos(phi1), -sin(phi1), 0;
    0, sin(phi1), cos(phi1), 0;
    0, 0, 0, 1
    ];

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

nv1 = (rotateXMatrix * move1)^(-1) * verticesCube1;
nv2 = (rotateXMatrix * move1)^(-1) * verticesCube2;
nv3 = (rotateXMatrix * move1)^(-1) * verticesCube3;

DrawShape (nv1, facesCube, 'c')
DrawShape (nv2, facesCube, 'w')
DrawShape (nv3, facesCube, 'g')
```

Листинг 6. Часть кода для реализации камеры.

7 Задание. Реализация перспективы.

Для того, чтобы реализовать перспективу необходимо спроецировать пространство, в нашем случае, на плоскость XY (матрица M_1), и воспользоваться матрицей перспективы (матрица M_2, a – коэффициент, отвечающий за степень искажения, для наглядности результатов возьмем $a = 0.1$).

$$M_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad M_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & a & 1 \end{bmatrix} \tag{8}$$

Матрица перспективы $M = M_1 \cdot M_2$.

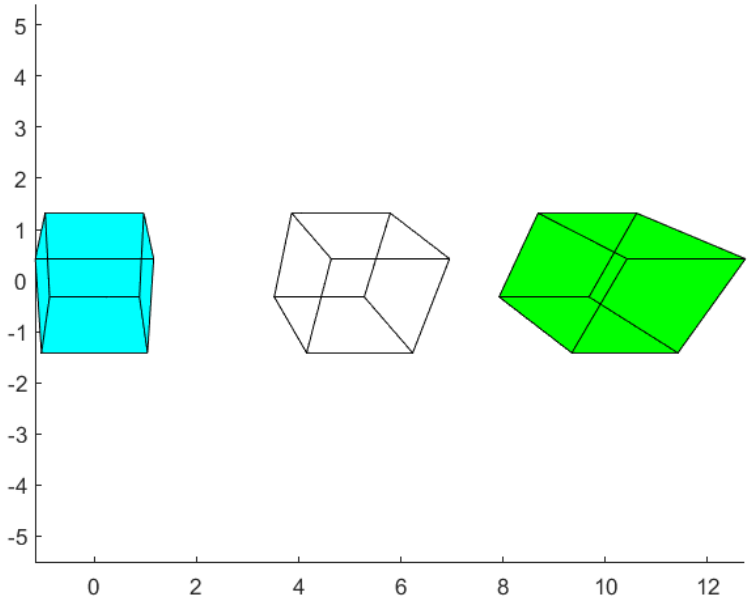


Рис. 21. Сцена после поворота камеры относительно X на 30°и добавлением перспективы.

Сместим камеру вправо на 7.

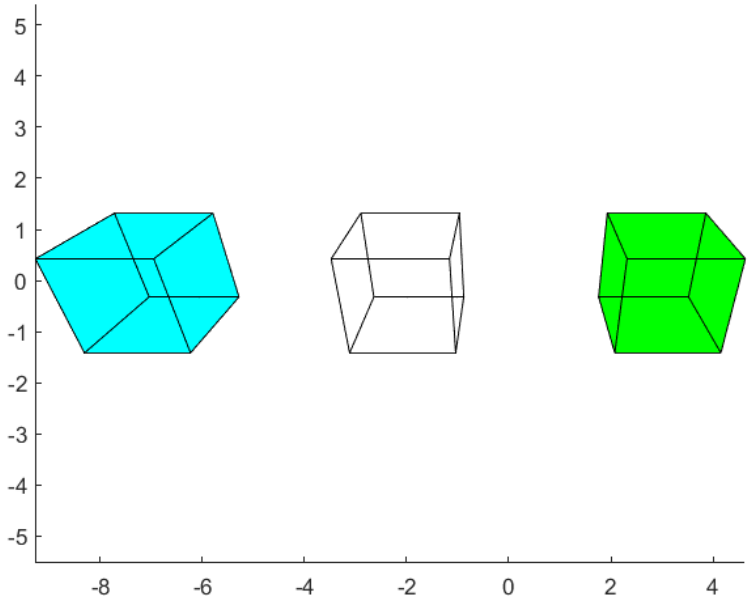


Рис. 22. Сцена после поворота камеры относительно X на $30\,^{\circ}$, смещением вправо на 7 и добавлением перспективы.

Реализация перспективы добавило реалистичности сцене с кубиками.

8 Задание. * Почти Minecraft.

Построим домик с помощью преобразований матриц.

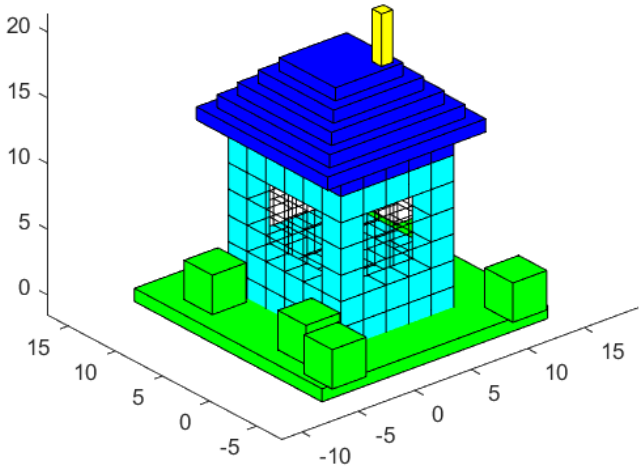


Рис. 23. Построенный домик до добавления камеры и перспективы.

Для визуализации был написан код на языке *Matlab*.
Код расположен на **GitHub**.