Лабораторная работа №3

Задание №1. Инспектор Гаджет исследует код

Давайте посмотрим на код и разберём подробно, как он работает:

```
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
];
```

В массиве перед нами восемь столбцов и четыре строки. Это означает, что вершины куба располагаются в массиве как вертикальные векторы, т.е. сначала в строку пишутся x-координаты, затем — y, потом — z и, наконец, в последней строке пишется для каждого вектора некий параметр w. В рамках курса Learn OpenGL, ссылка на который представлена в методическом пособии к лабораторной работе, рассказывается, что компонента w — параметр однородных (гомогенных) координат. Благодаря нему разница между точками и векторами практически стирается, и мы можем прибавлять к вектору константы при помощи матрицы преобразования, чего не получилось бы сделать без w-компоненты.
Заметим, что w у всех векторов равна 1 — именно поэтому мы часто опускаем эту компоненту, т.к. преобразование четырёхкомпонентного вектора $[x, y, z, w]$ в трёхкомпонентный выглядит как $[x/w, y/w, z/w]$.

Рассмотрим массив граней:

```
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
];
```

Здесь грани отображены как горизонтальный набор точек, пронумерованных от 1 до 8 по количеству точек у куба. Всего граней у куба шесть, поэтому и строк в массиве ровно столько же.

Теперь глянем на магию, которая отрисовывает кубик с осями и делает всё красиво:

```
DrawShape(varticesCube, facesCube, 'blue')
axis equal;
view(3);

function DrawShape(vertices, faces, color)
    patch('Vertices', (vertices(1:3,:)./vertices(4,:))', 'Faces', faces, 'FaceColor', color);
end
```

Итак, здесь вызывается функция `DrawShape`, которая отрисовывается все грани куба синим цветом в координатной плоскости. Далее пришлось поэкспериментировать, убрав команды `axis equal` и `view(3)`, чтобы выяснить, за что они отвечают: `axis equal` масштабирует оси соразмерно друг другу, чтобы одна ось не была растянута сильнее, чем друга, а `view(3)` подсказывает Matlab, что мы работаем в 3D, а не 2D (который установлен в программе по умолчанию).

Рубрика «наблюдения»: посмотрите-ка, здесь мы в `patch` передаём вершины, грани и цвет граней — передаются только первые три координаты каждой из вершин, разделённые на четвёртую. А ведь это то, что мы обсуждали выше о том, как четырёхкомпонентный вектор преобразовывается в трёхкомпонентный.

Мы можем построить любую другую фигуру, изменив как координаты точек, на которых будет строится фигура, так и порядок и количество точек в гранях, по котором сами грани будут строиться.

Задание №2. Wide Putin Walking

NB: все дальнейшие преобразования, необходимые для выполнения этого и других заданий, будут сделаны с использованием библиотеки `manim` в **Python** — хоть и матрицы преобразования будут представлены как четырёхкомпонентные, на самом же деле в коде будут использоваться трёхкомпонентные с учётом w.

Первое преобразование, которое нам предстоит сделать в пространстве — масштабирование или, по-другому говоря, умножение каждой из координат вектора на константы. К примеру, увеличим x- и y-координаты в 4 раза, а z-координату уменьшим в 2 раза. Для этого нам подойдёт следующая матрица A, которую мы сразу же и проверим, применив её к вектору $[x, y, z, w]$:

$$Av = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 4x + 0y + 0z + 0w \\ 0x + 4y + 0z + 0w \\ 0x + 0y + 0.5z + 0w \\ 0x + 0y + 0z + 1w \end{bmatrix} = \begin{bmatrix} 4x \\ 4y \\ 0.5z \\ w \end{bmatrix}$$

Таким нетрудным телодвижением мы проверили, что координаты действительно ведут себя именно так, как мы хотим, а w-компонента остаётся нетронутой и в принципе всегда фиксирована в ходе любых трансформаций вектора. Посмотреть анимашку для такого преобразования можно здесь ←

Задание №3. Outstanding move

Теперь пришла очередь уникального в своём роде преобразования пространства — речь о перемещении. Элементы матрицы умножаясь и складываясь поэлементно, образуют сумму $\alpha x+\beta y+\gamma z$ с некоторыми коэффициентами перед базисными векторами. Эта сумма, как вы видите, складывается из координат исходного вектора с некоторыми коэффициентом — у нас нет независимого элемента, который мог бы прибавиться ко всем координатам, не домножив их на коэффициенты. Учитывая, что получить в произведении сумму можно только поэлементным сложением, то нам не остаётся ничего, кроме как добавить w-компоненту в векторы — вот откуда она появилась и вот зачем она нужна.

Гораздо интереснее вопрос: как в таком случае поведут себя базисные вектора $\vec{i}$, $\vec{j}$, $\vec{k}$, которые расположены на осях координат, и сдвинутся ли они вместе с кубиком? Но перед ответом на этот вопрос, мы посмотрим, как выглядит матрица B такого преобразования и узнаем при помощи умножения, почему она выглядит именно так:

$$Bv = \begin{bmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1x + 0y + 0z + 3w \\ 0x + 1y + 0z - 2w \\ 0x + 0y + 1z - 1w \\ 0x + 0y + 0z + 1w \end{bmatrix} = \begin{bmatrix} x + 3w \\ y - 2w \\ z - w \\ w \end{bmatrix}$$

И вновь компонента w осталась неизменная в преобразованном векторе, а вот к координатам эта компонента прибавилась с неким коэффициентом — это объясняет, почему матрица выглядит так. Диагональ сохраняет элементы вектора неизменными, а последний столбец добавляет к каждой из координат известную компоненту.

А что происходит с базисными векторами? Давайте рассмотрим каждый из них, просто заменив соответствующие неизвестные в преобразованном векторе:

$$\vec{i}^* = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \vec{j}^* = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 1 \end{bmatrix} \qquad \vec{k}^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

С одной стороны может показаться, что вектор $\vec{k}$ после преобразования матрицей схлопнулся в точку, но это не так. Убедиться в этом можно, посмотрев мою анимашку ←

Мы наблюдаем, как вектор $\vec{k}$ оказывается в плоскости XOY. Появилось ли у вас в таком случае понимание, почему значение вектора после преобразования оказалось нулевое? Я сделал важное замечание в начале своего отчёта — при добавлении w-компоненты разница между векторами и точками практически стирается. Т.е. фактически векторы перенесли свои концы (точки, прочно связанные с кубом) сообразно с движением куба.

Но точки не могут быть базисами, поэтому, чтобы они ими стали, векторы забывают о том, что они в трёхмерном пространстве и действуют так, будто перед ними только их ось. Если взглянуть внимательно на концы векторов после преобразование, то у вектора $\vec{i}$ по оси x значение 4, у вектора $\vec{j}$ по оси y значение -1, а у вектора $\vec{k}$ по оси z как раз значение 0. Надеюсь, теперь картина прояснилась ;)

***

Теперь попробуем скомбинировать преобразования. Сначала посмотрим здесь, как действуют последовательно перемещение и масштабирование, а затем поменяем преобразования местами, выполнив сначала масштабирование, а затем перемещение, и посмотрим на эту красоту здесь.

Причина, по которой результаты различных комбинаций преобразований не сходятся и в первом случае фигура улетает далеко от координатной сетки, а во втором случае практически остаётся на месте, связана с порядком операций. Рассмотрим масштабирование как умножение координаты на α, а перемещение, как сумму координаты и β. И вот как влияет порядок операций:

$$\begin{gathered} x \;\to\; \alpha x \;\to\; \alpha x + \beta \\ x \;\to\; x + \beta \;\to\; \alpha(x + \beta) = \alpha x + \alpha\beta \end{gathered}$$

Во втором случае перемещение также расширилось под действием масштабирования. Поэтому порядок применения матриц преобразования важен.

Задание №4. Вот это поворот!

Матрицы поворота в 3D неким образом схожи с матрицами поворота в 2D — здесь тоже фигурируют косинусы и синусы. Опять придётся запоминать, где что ставится, да?.. Если в плоскости возможно вращение вокруг точки, то индуктивно из этого следует, что в пространстве вращение происходит вокруг прямых (или осей в нашем случае). Итак, независимых возможных вращений в 3D пространстве ровно три и им соответствуют некоторые матрицы, в которых угол θ — угол поворота.

Повороту вокруг оси OX соответствует матрица:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \\ y\cos\theta - z\sin\theta \\ y\sin\theta + z\cos\theta \\ w \end{bmatrix}$$

Повороту вокруг оси OZ соответствует матрица:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ z \\ w \end{bmatrix}$$

Иной студент впал бы в ступор при поиске матрицы для поворота вокруг оси OY, но здесь тоже ничего сложного:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x\cos\theta + z\sin\theta \\ y \\ z\cos\theta - x\sin\theta \\ w \end{bmatrix}$$

Проверим, обладают ли повороты в 3D свойством коммутативности, и заодно посмотрим, как меняется кубик в зависимости от выбранной комбинации матриц — анимашку можно глянуть здесь и здесь. Приходим к выводу, что матрицы поворотов не коммутативны, в отличие от матриц поворота в 2D. Это связано с тем, что независимых поворотов в 3D три, а в 2D всего лишь один.

Задание №5. You spin my head round...

Чтобы вращение кубика происхоило около одной вершины, необходимо её как-то зафиксировать. Для этого спросим себя: какая из точек при повороте пространства остаётся фиксированной? Конечно же, нулевая ;)

Остаётся просто закрепить один из углов куба преобразованием перемещения в точке $(0, 0, 0)$ и выполнить поворот. Итак, вот какие матрицы мы будем применять:

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Эта матрица сдвинет нижний угол куба со стороной равной 2 в точку начала координат.

$$D = \begin{bmatrix} 1/2 & -\sqrt{3}/4 & -3/4 & 0 \\ \sqrt{3}/2 & 1/4 & \sqrt{3}/4 & 0 \\ 0 & -\sqrt{3}/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

А эта матрица совершит поворот на 60° вокруг осей OX и OY одновременно. Давайте посмотрим, что получилось — анимашка уже готова :)

Задание №6. Свет, камера, мотор

Студент увидел слово MATLAB и упал в обморок... Мы его долго откачивали, но он сказал, что у него он не установлен, а там 20 Гб — они же будут целую вечность качаться :/ *А камеры в* `manim` *устроены непросто...*