Университет ИТМО Кафедра ВТ

Компьютерная графика

Доклад на тему

«Имитация камеры в WebGL»

Выполнил: студент группы P3417

Плюхин Дмитрий Алексеевич

Преподаватель: Королёва Юлия Александровна

Санкт-Петербург
2018 год

Оглавление

Поскольку WebGL представляет собой всего лишь библиотеку, реализующую набор достаточно простых функций для выполнения рендеринга, особый интерес представляет собой имитация камеры. В данном докладе мы как раз коснемся некоторых наиболее распространенных преобразований в WebGL, связанных с имитацией камеры.

Введение

Прежде всего необходимо определить, что понимается под камерой в WebGL и для чего камера нужна. Чтобы ничего не упустить из виду, проследим, каким образом создается изображение при помощи WebGL.

Типы координат WebGL и их взаимосвязь

Хочется отметить, что в WebGL используется правая система координат – это значит, что если ось Y направлена вверх, ось X направлена вправо, то ось Z будет указывать вперед. Приведенное определение касается лишь направления осей – о единицах измерения в нем нет ни слова. Действительно, в чем измеряются координаты той или иной точки? Ответ такой – в трехмерных координатах нет смысла говорить о какой-либо универсальной системе мер – в двумерном случае можно использовать пиксели, но в трехмерном, поскольку у нас появляется третья ось, направленная перпендикулярно дисплею, нельзя измерять расстояния в пикселях. Поэтому в WebGL не говорится о том, в чем необходимо измерять координаты точек – по большому счету, это просто числа, а фактическое отображение объектов зависит только от нашей интерпретации этих чисел.

Интуитивно должно быть понятно, что, тем не менее, абсолютно все объекты во всех координатах отображать не имеет смысла, чаще всего мы имеем какое-то подмножество координат, которые должны попасть в поле зрения наблюдателя. Таким образом, мы приходим к тому, что необходимо каким-то образом задавать, какие координаты будут отображаться на экране, а какие – нет. Теперь, кажется, самое время сказать о том, что на самом деле в WebGL существует несколько типов координат, которые отличаются между собой, главным образом, логически.

Объектные координаты

Это – исходные координаты точек, которые мы задаем вручную. Они заданы относительно центра модели, то есть точки, относительно которой будет осуществляться масштабирование, поворот, а также смещение – все три операции представляют собой преобразование объекта. Кстати, необходимо вспомнить, что все три преобразования объектов в WebGL определяются матрицами – для каждого типа преобразования есть матрица определенного вида, на которую умножаются координаты той или иной точки:

$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \text{масштабирование}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} - \text{смещение}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c & s & 0 \\ 0 & -s & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} c & 0 & -s & 0 \\ 0 & 1 & 0 & 0 \\ s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, -\text{повороты вокруг осей } X, Y, Z \text{ соответственно}$$

Почему именно матрицы? Рассмотрим, как производится смещение точки на некоторые величины вдоль координатных осей. А делается это очень просто – все, что нам нужно, это умножить матрицу, состоящую из координат точки и единицы (единица нужна для обеспечения возможности нелинейных преобразований, то есть преобразований, при которых не просто меняется коэффициент при переменной, а есть какая-то аддитивная константа, смещение – как раз пример нелинейного преобразования). Умножаем матрицу координат точки на матрицу смещения:

$$[x \quad y \quad z \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} = \begin{bmatrix} x*1+y*0+z*0+1*tx \\ x*0+y*1+z*0+1*ty \\ x*0+y*0+z*1+1*tz \\ x*0+y*0+z*0+1*1 \end{bmatrix}^T = [x+tx \quad y+ty \quad z+tz \quad 1]$$

Результат именно тот, который интуитивно должен был получиться при смещении точки. Можно заметить, что в качестве промежуточного результата приведена транспонированная матрица – это сделано только с целью более удобной записи.

Аналогичным образом можем произвести масштабирование координат точки, если умножим матрицу координат на матрицу масштабирования:

$$[x \quad y \quad z \quad 1] * \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x*sx+y*0+z*0+1*0 \\ x*0+y*sy+z*0+1*0 \\ x*0+y*0+z*sz+1*0 \\ x*0+y*0+z*0+1*1 \end{bmatrix}^T = [x*sx \quad y*sy \quad z*sz \quad 1]$$

Координаты при этом масштабируются относительно центра координат, потому что фигура будет как-бы «растягиваться» в обе стороны от 0, который, в свою очередь, останется на месте.

Что самое удобное в использовании матриц для преобразований координат точек – мы можем их умножать друг на друга и получать матрицы для одновременного применения нескольких эффектов к какой-либо точке, например, мы можем с легкостью совместить матрицу смещения с матрицей масштабирования следующим образом:

$$\begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \boldsymbol{sx} * \mathbf{1} + 0 * 0 + 0 * 0 + 0 * tx & sx * 0 + 0 * 1 + 0 * 0 + 0 * ty & sx * 0 + 0 * 0 + 0 * 1 + 0 * tz & sx * 0 + 0 * 0 + 0 * 0 + 0 * 1 \\ 0 * 1 + sy * 0 + 0 * 0 + 0 * tx & 0 * 0 + \boldsymbol{sy} * \mathbf{1} + 0 * 0 + 0 * ty & 0 * 0 + sy * 0 + 0 * 1 + 0 * tz & 0 * 0 + sy * 0 + 0 * 0 + 0 * 1 \\ 0 * 1 + 0 * 0 + sz * 0 + 0 * tx & 0 * 0 + 0 * 1 + sz * 0 + 0 * ty & 0 * 0 + 0 * 0 + \boldsymbol{sz} * \mathbf{1} + 0 * tz & 0 * 0 + 0 * 0 + sz * 0 + 0 * 1 \\ 0 * 1 + 0 * 0 + 0 * 0 + \mathbf{1} * \boldsymbol{tx} & 0 * 0 + 0 * 1 + 0 * 0 + \mathbf{1} * \boldsymbol{ty} & 0 * 0 + 0 * 0 + 0 * 1 + \mathbf{1} * \boldsymbol{tz} & 0 * 0 + 0 * 0 + 0 * 0 + \mathbf{1} * \mathbf{1} \end{bmatrix} =$$

$$= \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

Теперь если мы умножим координаты точки на полученную матрицу, то сделаем сразу и смещение, и масштабирование:

$$[x \quad y \quad z \quad 1] * \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ tx & ty & tz & 1 \end{bmatrix} = \begin{bmatrix} x * sx + y * 0 + z * 0 + 1 * tx \\ x * 0 + y * sy + z * 0 + 1 * ty \\ x * 0 + y * 0 + z * sz + 1 * tz \\ x * 0 + y * 0 + z * 0 + 1 * 1 \end{bmatrix}^T = [x * sx + tx \quad y * sy + ty \quad z * sz + tz \quad 1]$$

Надо заметить, что другой способ применения нескольких матриц – последовательное умножение координат точки на них. В этом случае имеет большое значение порядок, в котором происходят преобразования, вот пример преобразования:

$$10 * 2 + 2 = 22 - \text{сначал сделали масштабирование, потом смещение}$$
$$(10 + 2) * 2 = 24 - \text{сначала сделали смещение, потом масштабирование}$$

Второй случай является более правильным, так как при нем преобразуются также взаимные расстояния между объектами.

Итак, вернемся к типам координат. Изначально у нас были координаты относительно центра модели. После того, как мы трансформировали модель (сместили, масштабировали, повернули), координаты стали определять её положение относительно некой общей (с другими моделями) точки отсчета, поскольку другие модели точно также могли быть трансформировать независимо друг от друга.

Мировые координаты

После модификации каждой модели по-отдельности мы перешли к так называемым «мировым координатам» (world coordinates). Как было сказано, теперь координаты всех точек определены относительно общего центра, а не центра модели, который ранее у каждого объекта был свой. На этом, тем не менее, преобразования координат не заканчиваются. Вероятно, мы теперь захотим переместить все объекты как единое целое. Это аналогично тому, как в фильмах перемещается камера от одного объекта к другому – однако в WebGL нет такого объекта, как камера. В действительности, если мы хотим теперь переместить все объекты как единое целое, мы создадим некую общую матрицу, на которую будем умножать координаты каждой точки. После подобной трансформации начало координат приобретает уже другой смысл, отличный от предыдущего значения как «центра мира».

Координаты относительно камеры

Получается, что мы «вращаем мир» относительно наблюдателя – это аналогично тому, как если бы наблюдатель перемещался в этом мире. Поэтому теперь начало координат приобретает значение камеры, перемещающейся в «мире», а координаты всех точек теперь заданы относительно расположения камеры. Отдельной ситуацией, представляющей интерес, является реализация такого поведения камеры, при котором какое бы ни было задано положение камеры, она всегда была бы направлена на один и тот же объект. Как решается такая задача мы рассмотрим далее.

После этого, когда все модели уже трансформированы как нам нужно, наступает момент для определения того, какие объекты будут отображены на экране, иными словами – преобразование координат относительно камеры в нормализованные координаты.

Нормализованные координаты

Нормализованные координаты – координаты, подчиняющиеся некоему стандарту, который определяет, какие объекты и в какой позиции будут видны пользователю, а какие – нет. Поскольку WebGL не предусматривает

каких-либо единиц измерения координат, задача была решена следующим образом – определяется куб со стороной 2, точка пересечения диагоналей которого совпадает с центром координат. Все точки, попавшие в этот куб, будут показаны на дисплее, не попавшие – будут отброшены. Иными словами, координаты точек, который будут отображены на экране, должны находиться в диапазоне от -1 до 1. Задача программиста на данном этапе, таким образом, заключается в том, чтобы задать законы отображения координаты из пространства координат относительно камеры в пространство нормализованных координат. От того, какой вид будут иметь эти законы, очень многое зависит, в частности – будет ли реализован **эффект перспективы**, о котором мы поговорим чуть позже.

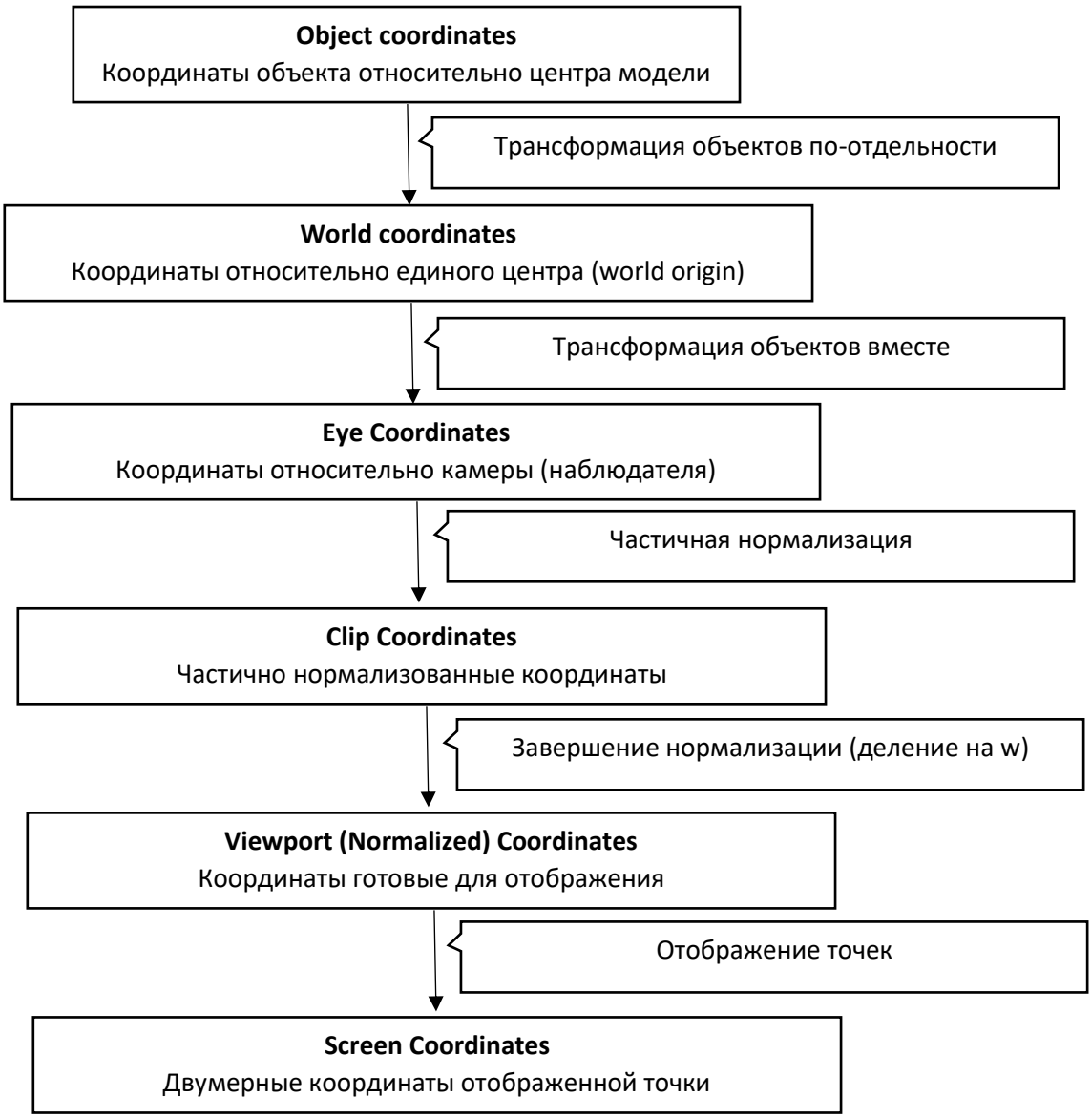
На самом деле, на пути преобразования координат точки относительно камеры в нормализованные координаты, есть один промежуточный этап, который называется – clip coordinates. Этот промежуточный шаг связан с четвертой компонентой координат, которую мы видели ранее. Переход от координат, заданных относительно камеры к клиповым происходит благодаря все тому же умножению на некоторую матрицу, при этом четвертая компонента, которая обозначается буквой w, становится равной некоторому числу. Шаг перехода от клиповых координат к нормализованным заключается лишь в автоматическом делении каждой из координат x, y и z на координату w – на практике эта возможность оказывается очень удобной, ведь все равно координата w нам больше не будет нужна, а возможность вынесения за скобки общего для всех координат делителя, который может зависеть от любой из трех координат, существенно повышает гибкость.

Координаты на экране

Последний этап в жизненном цикле координат точки – это её координаты в двумерной системе дисплея, определенные в пикселях.

Обобщение

Обобщая все вышесказанное, хотелось бы привести схему, которая в компактной форме иллюстрирует все типы координат в WebGL и переходы между ними.



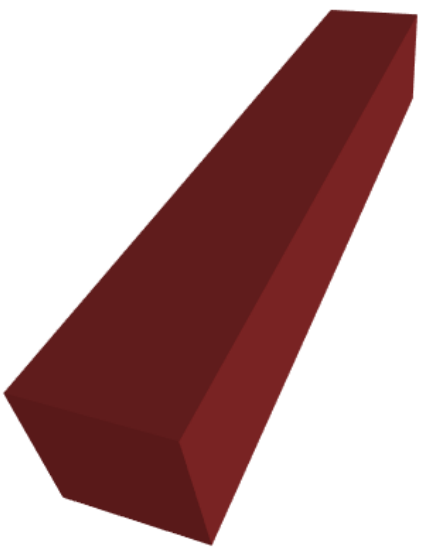
Некоторые матрицы для имитации камеры

В предыдущем разделе было отмечено, что существует несколько типов координат, которые отличаются друг от друга логической интерпретацией. Также было проиллюстрировано, что все преобразования координат в WebGL происходят при помощи матричного умножения и почему именно такой способ был выбран. В этой части доклада будет представлен вывод нескольких часто используемых матриц, связанных с камерой, которые могут быть наиболее полезны на практике.

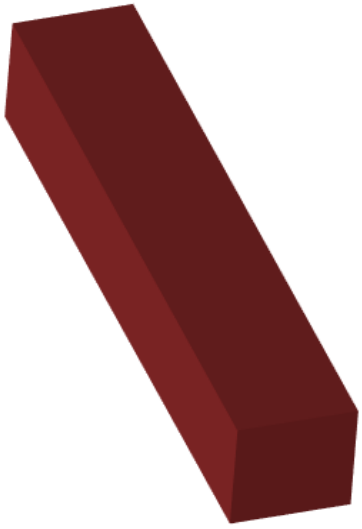
Матрица перспективы

Необходимость создания эффекта перспективы

В качестве первого пункта рассмотрим получение матрицы перспективы. И прежде всего рассмотрим, для чего она нужна. Во-первых, как видно из названия, для имитации перспективы, то есть эффекта, при котором объекты, расположенные дальше от камеры, кажутся меньше тех, которые расположены близко к камере. Особенно этот эффект проявляется, если мы смотрим на протяженные объекты, как видно на рисунке 1.



Изображение параллелепипеда, сделанное с использованием перспективы



Изображение параллелепипеда, сделанное с использованием ортографии

Рис. 1 Перспектива и ортография

Ортография

Согласно Википедии, ортографическая проекция — это изображение какого-нибудь предмета на плоскости, посредством проектирования отдельных его точек при помощи перпендикуляров к этой плоскости. Ортографические проекции используются обычно в инженерных чертежах.

Как было сказано в разделе «Введение» перспектива используется при нормализации координат. Ортография применяется на этом же этапе и может быть осуществлена достаточно простым способом – все, что нам необходимо знать – это диапазоны координат точек, которые мы хотим увидеть на экране. Если обозначим эти диапазоны как

$$[left; right], [bottom; top], [far; near] - \text{для координат } x, y, z \text{ соответственно}$$

То матрица нормализации координат в случае ортографии примет вид

$$\begin{bmatrix} \frac{2}{right - left} & 0 & 0 & 0 \\ 0 & \frac{2}{top - bottom} & 0 & 0 \\ 0 & 0 & \frac{2}{near - far} & 0 \\ \frac{left + right}{left - right} & \frac{bottom + top}{bottom - top} & \frac{near + far}{near - far} & 1 \end{bmatrix}$$

Доказать, что она именно такова – не составляет труда. Для этого необходимо только составить систему из двух уравнений, к примеру, для x, в которой требуется отразить, что крайняя левая координата по оси x (left) должна быть отображена на значение -1, а крайняя правая (right) - на 1 при условии, что все промежуточные значения должны быть распределены линейно:

$$\begin{cases} k * left + c = -1 \ (1) \\ k * right + c = 1 \ (2) \end{cases}$$

Решаем систему уравнений, сначала выразим c через k из 1 уравнения:

$$c = -1 - k * left \ (3)$$

Подставим (3) в (2):

$$k * right + c = k * right - 1 - k * left = 1$$

Переносим -1 в правую часть, выносим k за скобки и переносим разность, опять же, вправо. Получаем:

$$k = \frac{\mathbf{2}}{\boldsymbol{right - left}}$$

Теперь найдем с из (3):

$$c = -1 - k * left = -1 - \frac{2 * left}{right - left} = \frac{-right - left}{right - left} = \frac{\boldsymbol{right + left}}{\boldsymbol{left - right}}$$

Точно так же можно найти y. Для z решение будет немного другое, так как, во-первых, значения near и far должны быть домножены на -1 для получения реальных координат по оси z (наблюдатель находится в начале координат и смотрит в сторону отрицательной полуоси z, значит все z должны быть отрицательными, но удобнее передавать положительные параметры, а затем просто взять их с минусом), кроме того, ближнее значение z (near) должно отобразиться на значение -1, а не дальнее (far), как следовало бы ожидать, поэтому:

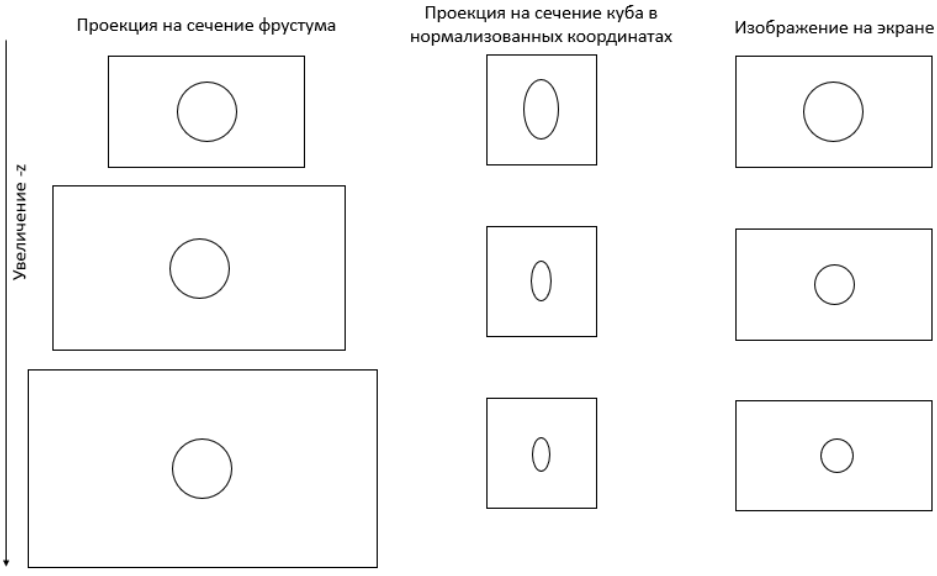
$$k = \frac{\mathbf{2}}{(-\boldsymbol{far}) - (-\boldsymbol{near})} = \frac{\mathbf{2}}{\boldsymbol{near - far}}$$

$$c = \frac{(-\boldsymbol{far}) + (-\boldsymbol{near})}{(-\boldsymbol{near}) - (-\boldsymbol{far})} = \frac{\boldsymbol{near + far}}{\boldsymbol{near - far}}$$

Перспектива

В отличие от ортографии, перспектива – это техника изображения пространственных объектов на какой-либо поверхности в соответствии с теми кажущимися сокращениями их размеров, изменениями очертаний формы и светотеневых отношений, которые наблюдаются в натуре.

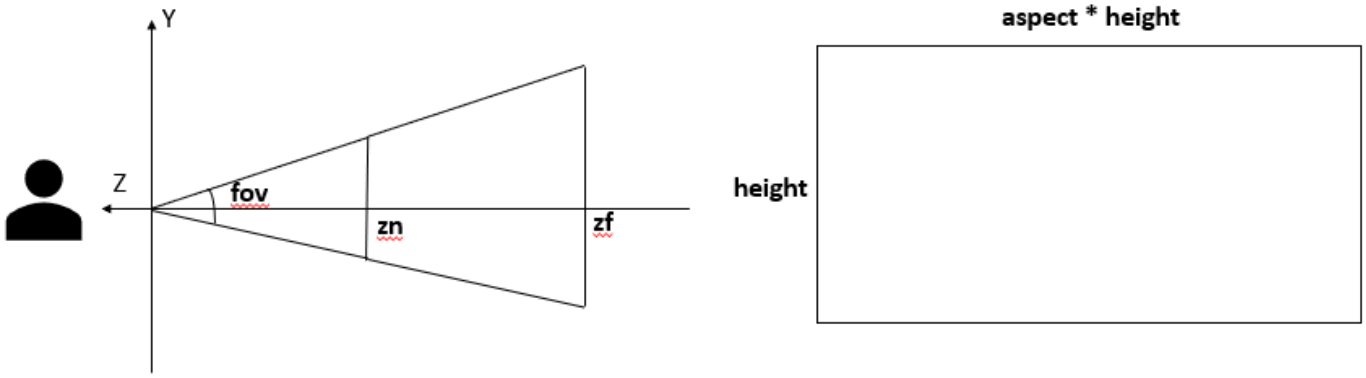
Для того, чтобы в модели, где в результате необходимо отобразить все видимые точки внутрь единичного куба, и при этом создать некоторый эффект перспективы, необходимо каким-то искусственным образом уменьшить объекты, имеющие меньшую координату z, по сравнению с объектами, имеющими большую координату. В исходной модели при перемещении объекта вдоль оси Z его размер относительно площади сечения куба постоянен, но что если мы сделаем так, что при удалении объекта его относительный размер будет уменьшаться, то есть, что если мы используем обычное масштабирование для того, чтобы добиться эффекта перспективы?



Например, в приведенной выше матрице для создания ортографической проекции если бы границы left и right раздвигались по мере увеличения z, мы бы увидели, что с возрастанием z объекты как бы сжимались – эффект, которого нам и нужно достичь. Так, мы приходим к фигуре, которая называется фрустум.

В получившейся фигуре необходимо задать угол наклона граней друг к другу. Также требуется определить, в каких пределах будет изменяться координата z – как было сделано в ортографической проекции. Задавать фиксированные границы по осям x и y не имеет смысла, поскольку они будут меняться в зависимости от z, однако необходимо задать отношение ширины к высоте для правильного масштабирования объектов внутри фрустума.

Обобщая все вышесказанное, приходим к следующей модели:

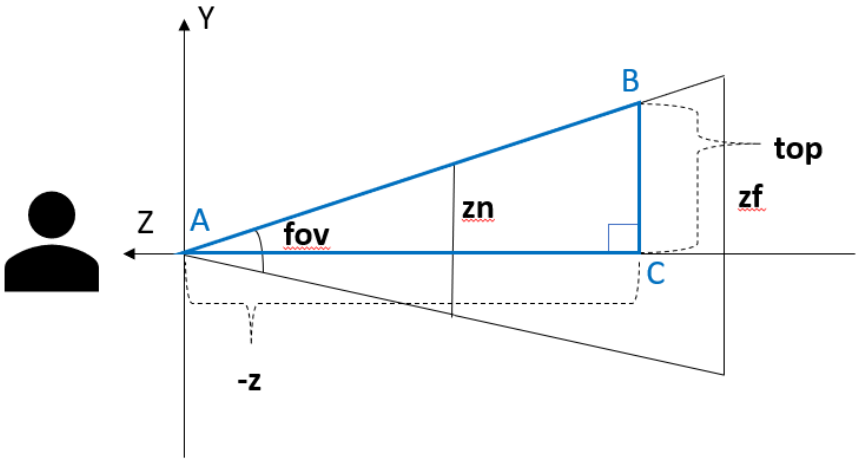


На приведенном рисунке:

fov (field of view) – телесный угол в вершине фрустума, совпадающей с точкой наблюдения
near – минимальная координата по оси z, которую должна иметь точка, чтобы попасть в поле зрения
far – максимальная координата по оси z, которую может иметь точка, чтобы попасть в поле зрения
aspect – отношение высоты сечения фрустума к ширине

Теперь переходим непосредственно к выведению соотношений, определяющих нормализацию координат точки в случае создания эффекта перспективы. Необходимо еще раз напомнить, что на данном этапе координаты всех точек заданы относительно точки наблюдения. Координату z будем брать с минусом, поскольку известно, что для всех видимых точек она будет отрицательной.

Прежде всего, мы не знаем, чему равна высота сечения фрустума – неплохо было бы её узнать. Очевидно, она должна увеличиваться с увеличением (по модулю) значения z. Ширину искать не очень удобно, поэтому будем рассматривать наибольшую координату по оси y, которую может иметь точка, чтобы попасть внутрь фрустума. Теперь внимательно посмотрим на рисунок и определим, что связывает значение -z и максимальное значение y.



Рассмотрим прямоугольный треугольник ABC. В нем угол A равен половине угла fov, а тангенс этого угла (отношение противолежащего катета, который равен максимальному значению y для данного сечения, которое обозначено как top, к прилежащему, который равен -z) может быть выражен двояко:

$$\frac{top}{-z} = tg\left(\frac{fov}{2}\right)$$

Из этого выражения можем выразить top:

$$top = -z * tg\left(\frac{fov}{2}\right)$$

Так, мы выразили y-координату верхней точки сечения фрустума через точку пересечения с осью z и угол при точке наблюдения. Возвращаясь к нормализованным координатам, вспомним, что если точка имеет координату top, она после нормализации должна быть равна 1. Напротив, если точка имеет координату –top, после нормализации она должна иметь координату -1. Таким образом, поскольку при движении вдоль оси OY значения должны быть распределены равномерно, для нормализации ординаты точки можем воспользоваться простым делением:

$$y_{normalized} = \frac{y}{top} = \frac{y}{-z * tg\left(\frac{fov}{2}\right)}$$

Для удобства обозначим величину, обратную найденному тангенсу как f для удобства записи, поскольку далее она будет встречаться довольно часто:

$$\boldsymbol{y_{normalized}} = \frac{\boldsymbol{y * f}}{\boldsymbol{-z}} \text{ где } f = \frac{1}{tg\left(\frac{fov}{2}\right)} = tg\left(\frac{\pi}{2} + \frac{fov}{2}\right)$$

Легко увидеть, что при увеличении значения -z для одинаковых y и f будут получаться меньшие значения – именно такого эффекта мы и хотели добиться.

Поскольку наибольшую координату сечения фрустума по оси X (обозначим ее right) можно выразить через наибольшую координату по оси Y, вывести формулу нормализации для координаты X не составит особого труда:

$$\boldsymbol{x_{normalized}} = \frac{x}{right} = \frac{x}{top * aspect} = \frac{\boldsymbol{x * f}}{\boldsymbol{-z * aspect}} \text{ где } f = \frac{1}{tg\left(\frac{fov}{2}\right)} = tg\left(\frac{\pi}{2} + \frac{fov}{2}\right)$$

Что касается нормализации координаты z – можно было бы, как и в предыдущих случаях, использовать линейное отображение исходных координат на нормализованные. Однако на практике взаимное расположение далеких объекты не имеет большого значения, гораздо лучше было бы детализировать расположение ближних объектов относительно друг друга, то есть, мы хотим, чтобы на ближние объекты отводилась большая часть пространства так называемого буфера глубины (по факту это пространство представляет тот же интервал от -1 до 1).

Иными словами, если сделать отображение исходных координат на нормализованные линейным, то несмотря на расположение объектов по оси z расстояние между ними будет одинаковым. Мы же хотим сделать так, чтобы приращение ненормализованной координаты z ближнего объекта дало большее приращение нормализованной координаты z по сравнению с дальним объектом – это обеспечит более высокую точность задания положения точки вблизи наблюдателя.

Добиться заданных требований можно, используя функцию обратной пропорциональности:

$$f(x) = \frac{m}{x} + c$$

Для определения множителя m и константы c достаточно использовать значения двух точек для составления уравнения, подобно тому, как мы делали при выводе соотношений для ортографической проекции:

$$\begin{cases} \frac{m}{-near} + c = -1 \ (1) \\ \frac{m}{-far} + c = 1 \ (2) \end{cases}$$

Из 1 уравнения следует, что

$$c = -1 - \frac{m}{-near} \ (3)$$

Подставим (3) в (2) и выразим m:

$$\frac{m}{-far} + c = \frac{m}{-far} - 1 - \frac{m}{-near} = 1$$

$$\frac{-near + far}{near * far} = \frac{2}{m}$$

$$\boldsymbol{m = -\frac{2 * near * far}{near - far}}$$

Наконец, найдем константу c:

$$\boldsymbol{c = -1 - \frac{m}{-near} = -1 - \frac{2 * far}{near - far} = \frac{-near - far}{near - far}}$$

Так, нормализация координаты z происходит следующим образом:

$$\boldsymbol{z_{normalized} = \frac{m}{-z} + c = \frac{2 * near * far}{-z * (near - far)} - \frac{near + far}{near - far}}$$

Заметим, что нормализация координат x и y предполагает деление на -z – можем этим воспользоваться, если, во-первых, искусственно вынесем в формуле нормализации координаты z деление на -z за скобки, во-вторых, поскольку преобразование из клиповых координат в нормализованные предполагает деление на w, учтем при построении матрицы перспективы, что w должно быть равно -z. В таком случае клиповые координаты будут иметь вид:

$$\boldsymbol{x_{clip} = \frac{x * f}{aspect}}$$

$$\boldsymbol{y_{clip} = y * f}$$

$$\boldsymbol{z_{clip} = z * \frac{near + far}{near - far} + \frac{2 * near * far}{near - far}}$$

$$\boldsymbol{w_{clip} = -z}$$

Наконец, составим матрицу, обеспечивающую выполнение приведенных преобразований:

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near + far}{near - far} & -1 \\ 0 & 0 & \frac{2 * near * far}{near - far} & 0 \end{bmatrix} \text{где } f = \frac{1}{tg\left(\frac{fov}{2}\right)} = tg\left(\frac{\pi}{2} + \frac{fov}{2}\right)$$

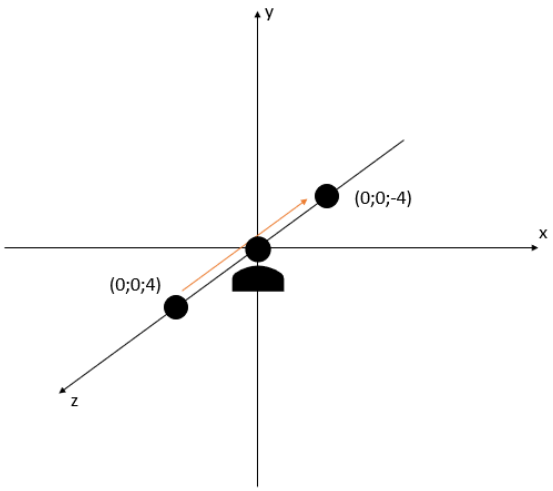
Таким образом, была выведена матрица перспективы, которая, пожалуй, является одной из самых важных и используется в практически любом проекте WebGL.

Матрица слежения за объектом

Если выведенная ранее матрица используется для преобразования координат точки относительно камеры в нормализованные координаты, то сейчас будет рассмотрен вывод матрицы для преобразования мировых координат в координаты относительно камеры таким образом, чтобы камера постоянно была направлена на какой-либо объект.

Принципы вычисления и действия матрицы слежения за объектом

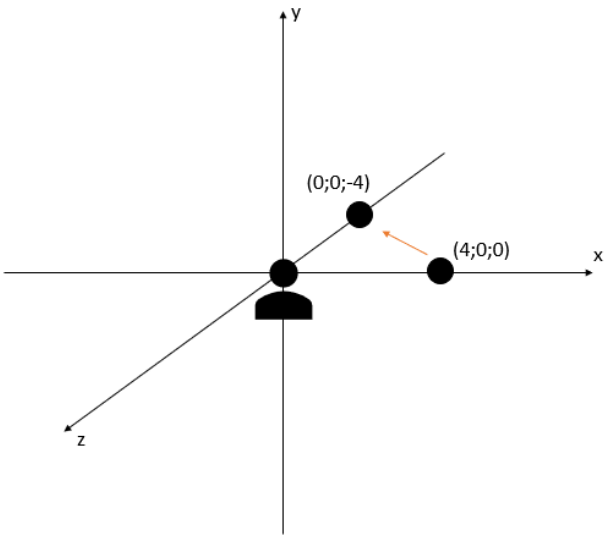
Чтобы понять, как работает матрица слежения за объектом, начнем с простого примера. Предположим, что у нас есть точка, расположенная в координатах (0;0;4). Нужно ответить на вопрос, что необходимо сделать для того, чтобы «направить камеру» на неё.



В предыдущих разделах было показано, что в результате преобразований мировых координат в координаты относительно камеры наблюдатель должен находиться в центре координат и смотреть против оси OZ – отсюда следует, что наблюдатель может видеть только точки, имеющие отрицательную z-координату, поэтому естественное решение, которое приходит в голову – это переместить точку так, чтобы её координата z стала отрицательной, то есть равной -4. Сделать это можно, используя матричное умножение, таким образом:

$$[0 \quad 0 \quad 4 \quad 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad -4 \quad 1]$$

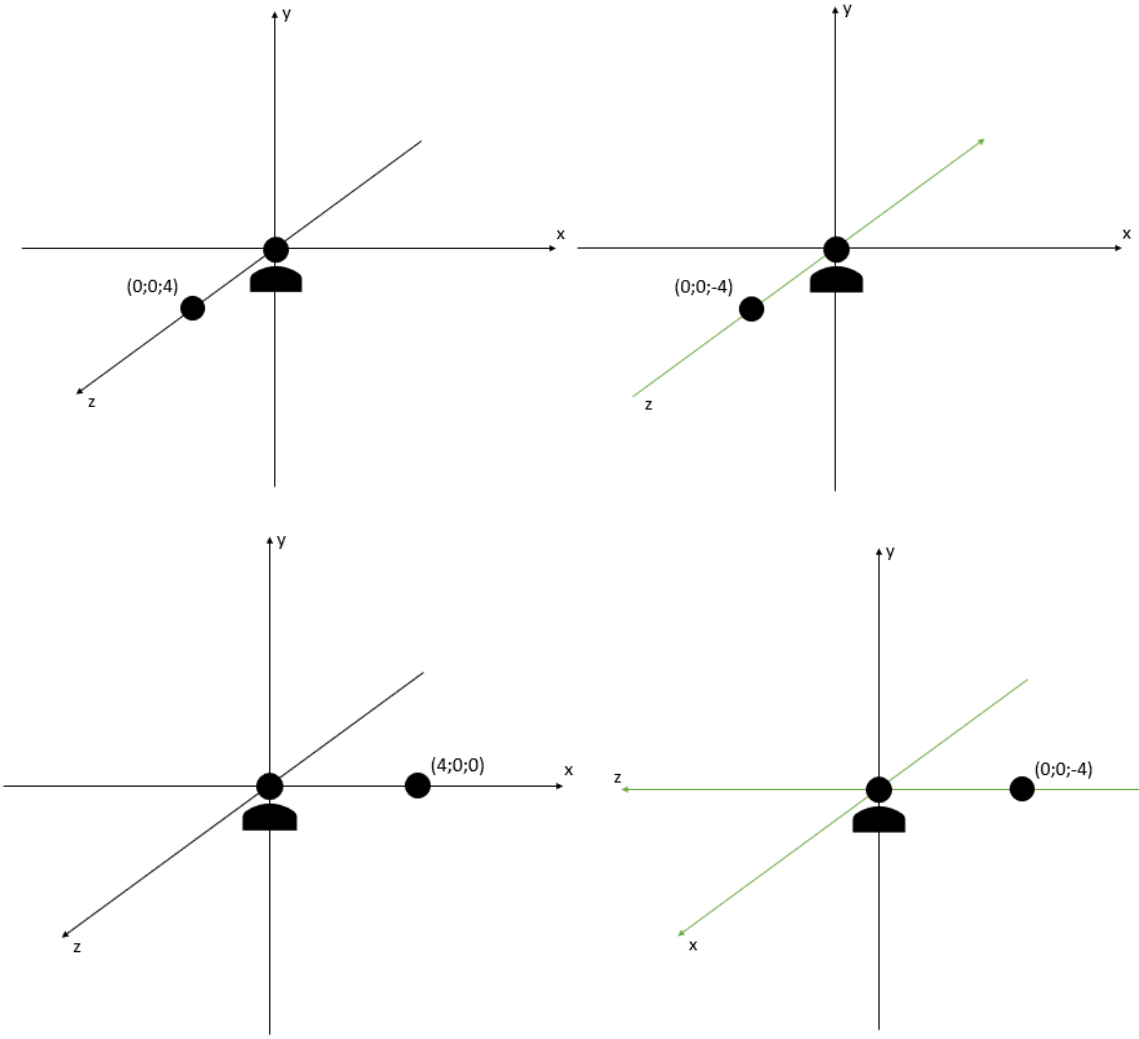
Теперь возьмем немного более сложный случай – что, если точка, на который нужно направить камеру, находится на оси абсцисс, к примеру, имеет координаты (4;0;0). Его, опять же, надо каким-то образом перенести на отрицательную полуось координатной оси Z для того, чтобы он стал видимым для наблюдателя.



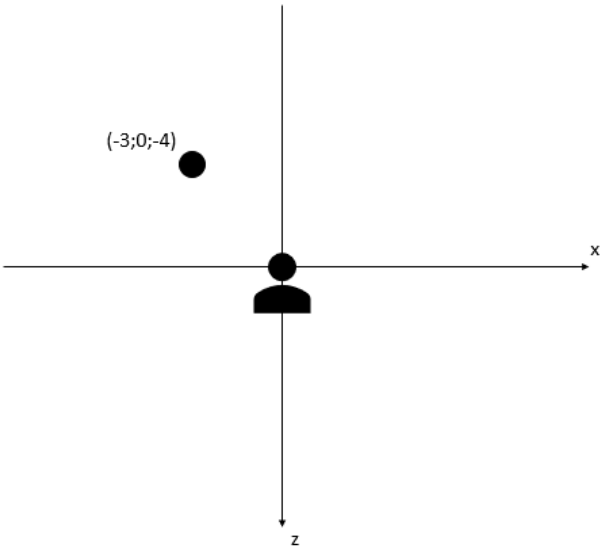
Осуществить такое преобразование, опять же, можно при помощи матриц. В данном случае видно, что после преобразования z должен быть равен -x, y должен остаться неизменным, а x в таком случае должен быть равен -z (иными словами, при таком преобразовании оси x и z как бы поменяются местами).

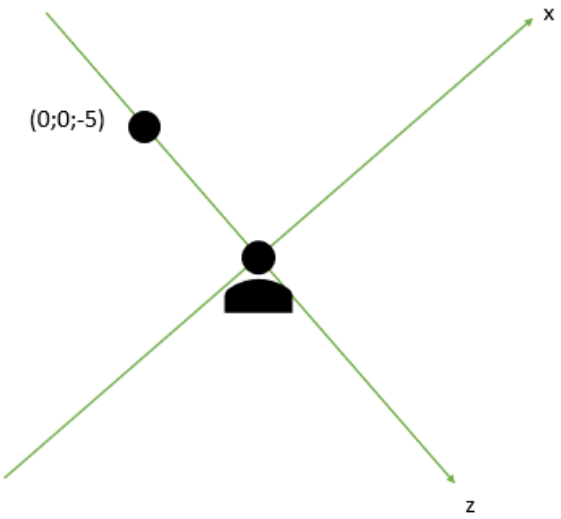
$$[4 \quad 0 \quad 0 \quad 1] * \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad -4 \quad 1]$$

Для того, чтобы было проще понимать, как происходят такие преобразования мы можем вместо перемещения объектов мысленно перемещать координатные оси, как показано на рисунках ниже.



Третий случай, который хотелось бы рассмотреть – промежуточный. Интерес представляет, что будет происходить, если точка не находится на координатной оси, а имеет, к примеру, координаты (-3; 0; -4). Далее для понятности графиков будет показана только двумерное пространство XOZ, поскольку координата Y в дальнейших расчетах всегда равна нулю.





Так, согласно описанной модели, в результате наших преобразований точка должна получить координаты (0;0;-5). Видно, что расстояние от наблюдателя до точки остается постоянным. Попробуем получить необходимую матрицу. Прежде всего, начальные координаты x и z должны вместе определять конечную координату z, в противном случае становится возможным перемещение точки вдоль одной из двух осей и получение каждый раз одной и той же итоговой точки, чего быть не должно. То есть, касательно итоговой координаты z должно выполняться условие

$$-3a - 4b = -5$$

Можем выразить b через a:

$$b = \frac{-3a + 5}{4}$$

Во-вторых, видно, что итоговая координата по оси x равна нулю. Такое возможно либо в том случае, если в матрице все коэффициенты первого столбца будут равны 0, что не верно, поскольку тогда мы получим, что абсолютно все точки будут находиться на оси Z. Второй возможный случай – коэффициенты такие, что именно при значения -3 для x-координаты и -4 для z – координаты получается 0, то есть имеет место уравнение

$$-3c - 4d = 0$$

Можем избавиться от одной переменной, выразив d через c:

$$d = -\frac{3}{4}c$$

Искомая матрица, таким образом, должна иметь вид

$$\begin{bmatrix} c & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{4}c & 0 & \frac{5-3a}{4} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В принципе, a и c теперь можно было бы выбрать любыми, но заметим следующее: если представить, что в первых трех строках первые три значения – это координаты вектора, то во всех предыдущих матрицах длина векторов, имеющих такие координаты, равна единице. Соблюдем это условие, необходимость которого станет ясна чуть позже, и в нашей новой матрице:

$$\begin{cases} \sqrt{a^2 + c^2} = 1\ (1) \\ \sqrt{\left(\frac{3}{4}c\right)^2 + \left(\frac{5 - 3a}{4}\right)^2} = 1\ (2) \end{cases}$$

Из первого уравнения:

$$c^2 = 1 - a^2\ (3)$$

Подставим (3) в (2):

$$9 - 9a^2 + 25 - 30a + 9a^2 = 16$$

$$34 - 30a = 16$$

$$\boldsymbol{a = \frac{18}{30} = 0,6}$$

$$\boldsymbol{c = \sqrt{1 - 0,36} = 0,8}$$

Таким образом, искомая матрица примет вид:

$$\begin{bmatrix} 0,8 & 0 & 0,6 & 0 \\ 0 & 1 & 0 & 0 \\ -0,6 & 0 & 0,8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Проверим полученный результат:

$$[-3 \quad 0 \quad -4 \quad 1] * \begin{bmatrix} 0,8 & 0 & 0,6 & 0 \\ 0 & 1 & 0 & 0 \\ -0,6 & 0 & 0,8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [-2,4 + 2,4 \quad 0 \quad -1,8 - 3,2 \quad 1] = [0 \quad 0 \quad -5 \quad 1]$$

Так, получен верный результат, но, вероятно, коэффициенты матрицы можно было бы найти каким-то другим способом. Примечательно, что полученные коэффициенты хорошо делятся на координаты точки. Заметим также следующее: в наших преобразованиях мы поворачивали только оси OX и OZ, ось OY же осталась неизменной. Осталась неизменной вместе с тем и вторая строка матрицы преобразования, первая и третья же поменялись. В связи с этим сделаем предположение, что в двух поменявшихся строках записаны координаты векторов, образующих новую систему координат. Чтобы проверить это, составим матрицу из координат векторов требуемой системы координат.

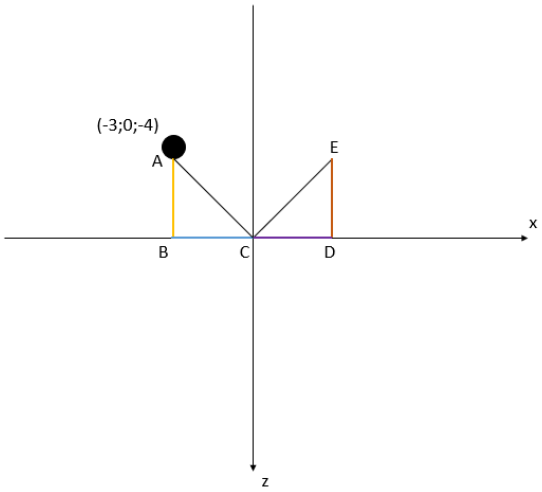
Направление нового вектора Z по определению новой системы координат совпадает с направлением вектора от цели до точки наблюдения, которая в данном случае находится в начале координат. Для нахождения координат вектора вычтем координаты начальной точки из координат конечной:

$$(0 - (-3) \quad 0 - 0 \quad 0 - (-4)) = (3 \quad 0 \quad 4)$$

Выполним нормализацию:

$$\left(\frac{3}{\sqrt{3^2 + 0^2 + 4^2}} \quad 0 \quad \frac{4}{\sqrt{3^2 + 0^2 + 4^2}}\right) = (0,6 \quad 0 \quad 0,8)$$

Для того, чтобы найти координаты вектора, соответствующего новому направлению оси X, заметим, что синус угла ACB



(см. рисунок) равен 0,8. Угол ECD равен 90 градусов – угол ACB, откуда следует, что косинус угла ECD равен синусу угла ACB, то есть, 0,8. Если длина CE равна 1, то исходя из вышесказанного CD = 0,8, а ED в таком случае равно 0,6. То есть, координаты вектора, определяющего направление оси X следующие:

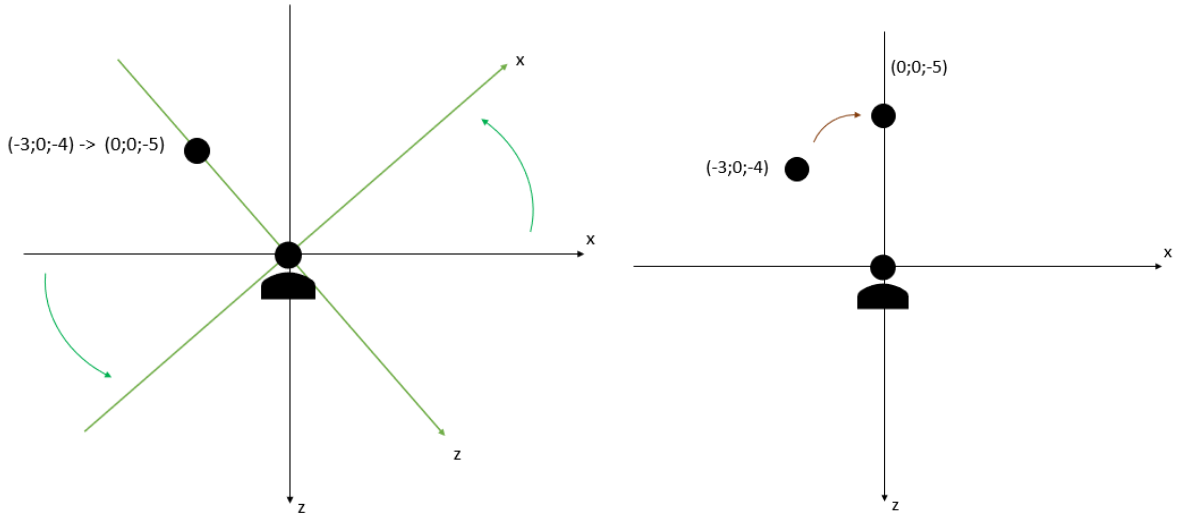
$$(0{,}8 \quad 0 \quad -0{,}6)$$

Координата z взята с минусом, поскольку направление вектора противоположно направлению оси Z.

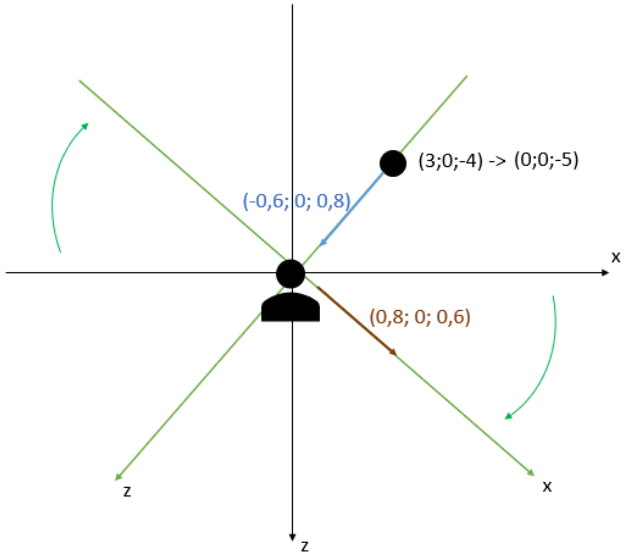
Теперь составим матрицу единичных векторов, определяющих новые направления координатных осей (расширим также её до 4 измерений для единообразия):

$$\begin{bmatrix} 0{,}8 & 0 & -0{,}6 & 0 \\ 0 & 1 & 0 & 0 \\ 0{,}6 & 0 & 0{,}8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Смысл полученной матрицы, составленной из координат векторов, определяющих новые направления координатных осей заключается в том, что, используя её, можно повернуть какую-либо точку на углы относительно трех координатных осей, на которые нам пришлось повернуть систему координат, чтобы ось Z проходила через определенную точку. Это не совсем то, что нам необходимо было получить, однако по модулю все числа соответствуют требованиям. На самом деле такой результат получился как раз по той причине, что мы от задачи перемещения объекта перешли к задаче перемещения координатных осей. То есть, для перехода к исходной задаче нам нужно инвертировать рассчитанные преобразования – это значит, что если, к примеру, в данной задаче мы при неподвижном объекте вращали координатные оси по часовой стрелке, то при инверсии этого преобразования нам нужно будет при неподвижных координатных осях повернуть объект по часовой стрелке на ту же величину.



Правильный результат можно было бы получить, если бы мы решали «зеркальную» задачу, то есть, если бы наша точка находилась не слева, а справа от оси Z, соответственно, и оси надо было бы вращать в другую сторону:



Тогда получается верный ответ:

$$\begin{bmatrix} 0{,}8 & 0 & 0{,}6 & 0 \\ 0 & 1 & 0 & 0 \\ -0{,}6 & 0 & 0{,}8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Но постоянно менять условие задачи неудобно, есть способ модификации полученного ответа для «поворачивающихся координатных осей» так, чтобы он соответствовал трансформации объекта. Этот способ вытекает из вышеизложенных рассуждений о взаимообратимости двух преобразований – **нам необходимо найти такую матрицу, которая бы при умножении на матрицу, составленную из координат новых векторов, давала бы матрицу, задающую текущую конфигурацию системы координат**. Иными словами, метод заключается в вычислении обратной матрицы, то есть такой матрицы, произведение которой на исходную дает единичную матрицу (то есть по диагонали расположены единички). Принцип вычисления обратной матрицы широко используется для перехода от задачи вычисления положения объектов относительно камеры к вычислению положения камеры относительно начала координат. Вычисление обратной матрицы – в общем случае непростая задача, поэтому рассматривать, как она решается, мы не будем. Однако проверим, что вычисление обратной матрицы действительно дает в нашем случае желаемый эффект. Для этого умножим две матрицы, полученные разными способами в ходе наших рассуждений. В результате должны получить единичную матрицу:

$$\begin{bmatrix} 0{,}8 & 0 & 0{,}6 & 0 \\ 0 & 1 & 0 & 0 \\ -0{,}6 & 0 & 0{,}8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0{,}8 & 0 & -0{,}6 & 0 \\ 0 & 1 & 0 & 0 \\ 0{,}6 & 0 & 0{,}8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0{,}64 + 0{,}36 & 0 & -0{,}6 * 0{,}8 + 0{,}8 * 0{,}6 & 0 \\ 0 & 1 * 1 & 0 & 0 \\ -0{,}6 * 0{,}8 + 0{,}6 * 0{,}8 & 0 & -0{,}6 * (-0{,}6) + 0{,}8 * 0{,}8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обобщение

Из постановки задачи слежения камеры за объектом следует тот факт, что нам должны быть известны координаты двух точек – желаемого положения камеры (обозначим как from) и точки, на которую камера должна быть направлена (обозначим как to). Надо заметить, что несмотря на то, что раньше рассматривалось расположение камеры только в центре координат, добавление смещения фактически ни на что не влияет – вычисление координат векторов для новых координатных осей будет выглядеть точно так же как и в рассмотренном примере, а перемещение камеры в заданную точку будет проведено как обыкновенное смещение камеры – то есть, мы заполним координатами новой позиции последнюю строку матрицы преобразования и затем возьмем обратную матрицу, поскольку переходили от задачи перемещения объектов относительно камеры к перемещению камеры относительно центра координат. Необходимо обратить внимание, что, если используется матрица направления камеры на некоторую точку, все другие повороты камеры не будут играть роли.

Для расчета матрицы нам понадобятся три вектора, которые будут задавать новую систему координат, центр которой сместится в точку, где должна находиться камера, а ось Z будет проходить через точку, которую камера должна отслеживать.

Первый из векторов, который мы обозначим как forward, является единичным и должен быть направлен от точки расположения цели в сторону камеры. Найти его можно, посчитав разность векторов и нормализовав результат:

$$\boldsymbol{forward(from, to)} = \left(\frac{\boldsymbol{from_x - to_x}}{\boldsymbol{length}}, \quad \frac{\boldsymbol{from_y - to_y}}{\boldsymbol{length}}, \quad \frac{\boldsymbol{from_z - to_z}}{\boldsymbol{length}}\right)$$

$$\text{где } length = \sqrt{(from_x - to_x)^2 + \left(from_y - to_y\right)^2 + (from_z - to_z)^2}$$

Требование ко второму вектору, который обозначим как right, одно - он должен быть перпендикулярен найденному вектору forward. Проще всего было бы найти такой вектор, если бы у нас был еще какой-нибудь вектор, отличный от forward, назовем его tmp, тогда мы могли бы их перемножить и по правилу переменожения векторов получили бы вектор, перпендикулярный обоим векторам. В качестве такого вспомогательного вектора возьмем единичный вектор, направленный вверх, поскольку нам, скорее всего, не понадобится направлять камеру напрямую вверх:

$$tmp = (0, 1, 0)$$

Найдем произведение векторов:

$$right(forward, tmp) =$$

$$= (forward_y * tmp_z - forward_z * tmp_y, forward_z * tmp_x - forward_x * tmp_z, forward_x * tmp_y - forward_y * tmp_x)$$

Теперь можем легко найти третий вектор (обозначим его up), который должен быть перпендикулярен двум предыдущим. Снова воспользуемся правилами произведения векторов:

$$up(forward, right) =$$

$$= (forward_y * right_z - forward_z * right_y, forward_z * right_x - forward_x * right_z, forward_x * right_y - forward_y * right_x)$$

Так, мы нашли три вектора, на базе которых можем построить новую систему координат. Искомая матрица при этом выглядит следующим образом:

$$\begin{bmatrix} right_x & right_y & right_z & 0 \\ up_x & up_y & up_z & 0 \\ forward_x & forward_y & forward_z & 0 \\ from_x & from_y & from_z & 1 \end{bmatrix}$$

Используя такую матрицу, мы, грубо говоря, подменяем стандартные единичные вектора, соответствующие координатным осям, своими собственными. Если посмотреть на то, как выглядит исходная матрица, используемая для преобразования положения камеры, то можно заметить, что там, куда мы подставили новые значения векторов, содержатся изначальные координаты:

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

Таким образом, вектор forward соответствует оси OZ, вектор right – оси OX, а вектор up – оси OY.

Заключение

Итак, в докладе были описаны два преобразования, часто используемые в WebGL для имитации камеры. Были показаны принципы работы этих преобразований и проиллюстрирован вывод необходимых матриц.

Первое преобразование обеспечивает создание эффекта перспективы, за счет чего все происходящее на сцене становится реалистичным в том плане, что соответствует визуальному опыту пользователя, по этой причине данное преобразование является одним из самых важных, но в то же время достаточно неочевидным на первый взгляд.

Второе преобразование связано с созданием такого полезного эффекта, как слежение камеры за каким-то определенным объектом на сцене – оно сильно облегчает трансформацию камеры, которое в ином случае приходилось бы искусственно направлять на объект при каждом изменении положения – в связи с этим данное преобразование также имеет большое значение. Единственным слабым местом описанного метода является необходимость аккуратного выбора промежуточного вектора – к решению этой проблемы есть несколько подходов, один из которых – наложение запрета на ориентацию камеры в направлении выбранного промежуточного вектора. Кроме того, был проведен краткий обзор типов координат, используемых в WebGL и проиллюстрирована необходимость взятия обратной матрицы при трансформировании камеры.