Перспективные матрицы в графическом API или дьявол прячется в деталях

В определённый момент у любого разработчика в области компьютерной графики возникает вопрос: как же работают эти перспективные матрицы? Подчас ответ найти очень непросто и, как это обычно бывает, основная масса разработчиков бросает это занятие на полпути.

Это не решение проблемы! Давайте разбираться вместе!

Будем реалистами с практическим уклоном и возьмём в качестве подопытного OpenGL версии 3.3. Начиная с этой версии каждый разработчик обязан самостоятельно реализовывать модуль матричных операций. Замечательно, это то, что нам нужно. Проведём декомпозицию нашей с вами нелёгкой задачи и выделим основные моменты. Немного фактов из спецификации OpenGL:

- Матрицы хранятся по столбцам (column-major);
- Однородные координаты;
- Канонический объём отсечения (CVV) в левосторонней системе координат.

Существует два способа хранения матриц: column-major и row-major. На лекциях по линейной алгебре как раз используется схема row-major. По большому счёту представление матриц в памяти не имеет значения, потому что матрицу всегда можно перевести в одного вида представления в другое простым транспонированием. А раз разницы нет, то для всех последующих расчётов мы будем использовать классические row-major матрицы. При программировании OpenGL есть небольшая хитрость, которая позволяет отказаться и от транспонирования матриц при сохранении классических row-major расчётов. В шейдерную программу матрицу нужно передавать как есть, а в шейдере производить умножение не вектора на матрицу, а матрицы на вектор.

Однородные координаты – это не очень хитрая система с рядом простых правил по переводу привычных декартовых координат в однородные координаты и обратно. Однородная координата — это матрица-строка размерности [1x4]. Для того чтобы перевести декартову координату в однородную координату необходимо x, y и z умножить на любое действительное число w (кроме 0). Далее необходимо записать результат в первые три компоненты, а последний компонент

будет равен множителю *w*. Другими словами:
$\begin{bmatrix} x & y & z \end{bmatrix}$ — декартовы координаты
w – действительное число, не равное 0
$\begin{bmatrix} wx & wy & wz & w \end{bmatrix}$ — однородные координаты

Небольшой трюк: Если *w* равно единице, то всё что нужно для перевода, это перенести компоненты *x*, *y* и *z* и приписать единицу в последний компонент. То есть получить матрицу-строку:
$\begin{bmatrix} x & y & z & 1 \end{bmatrix}$

Несколько слов о нуле в качестве *w*. С точки зрения однородных координат это вполне допустимо. Однородные координаты позволяют различать точки и вектора. В декартовой же системе координат такое разделение невозможно.
$\begin{bmatrix} x & y & z & 1 \end{bmatrix}$ — точка, где (*x, y, z*) – декартовы координаты
$\begin{bmatrix} x & y & z & 0 \end{bmatrix}$ — вектор, где (*x, y, z*) – радиус-вектор

Обратный перевод вершины из однородных координат в декартовы координаты осуществляется следующим образом. Все компоненты матрицы-строки необходимо разделить на последнюю компоненту. Другими словами:
$\begin{bmatrix} x_h & y_h & z_h & w_h \end{bmatrix}$ — однородные координаты
$\begin{bmatrix} \frac{x_h}{w_h} & \frac{y_h}{w_h} & \frac{z_h}{w_h} \end{bmatrix}$ — декартовы координаты

Главное что необходимо знать, что все алгоритмы OpenGL по отсечению и растеризации работают в декартовых координатах, но перед этим все преобразования производятся в однородных координатах. Переход от однородных координат в декартовы координаты осуществляется аппаратно.

Канонический объём отсечения или Canonic View Volume (CVV) – это одна из мало документированных частей OpenGL. Как видно из рис. 1 CVV – это выровненный по осям куб с центром в начале координат и длиной ребра равной двойке. Всё, что попадает в область CVV подлежит растеризации, всё, что находится вне CVV игнорируется. Всё, что частично выходит за границы CVV, подлежит алгоритмам отсечения. Самое главное, что надо знать — система координат CVV левосторонняя!

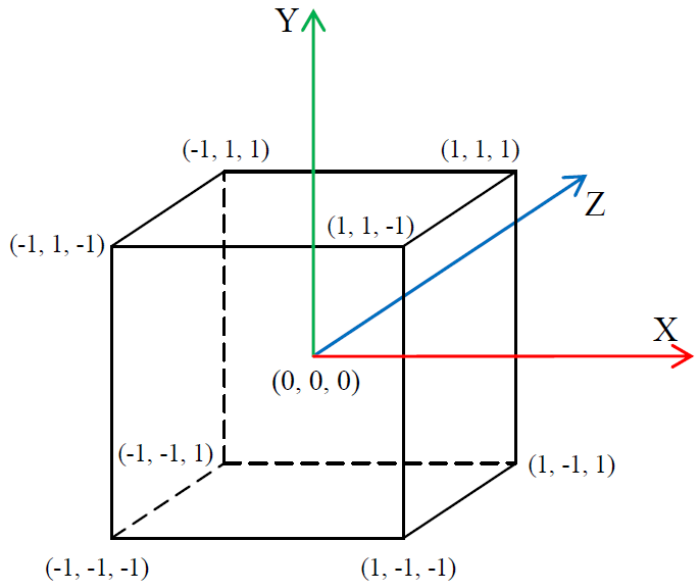

Рис. 1. Канонический объём отсечения OpenGL (CVV)

Левосторонняя система координат? Как же так, ведь в спецификации к OpenGL 1.0 ясно написано, что используемая система координат правосторонняя? Давайте разбираться.

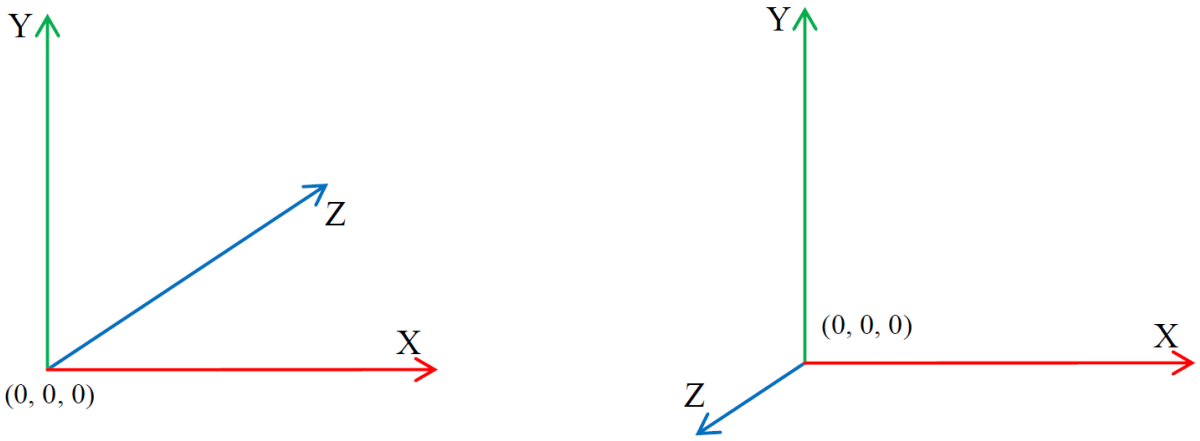

Рис. 2. Системы координат

Как видно из рис. 2 системы координат различаются лишь направлением оси *Z*. В OpenGL 1.0 действительно используется правосторонняя пользовательская система координат. Но система координат CVV и пользовательская система координат это две совершенно разные вещи. Более того, начиная с версии 3.3, больше не существует такого понятия как стандартная система координат OpenGL. Как упоминалось ранее, программист сам реализует модуль матричных операций. Формирование матриц вращения, формирование проекционных матриц, поиск обратной матрицы, умножение матриц – это минимальный набор операций, входящих в модуль матричных операций. Возникает два логичных вопроса. Если объём видимости — это куб с длиной ребра равной двум, то почему сцена размером

в несколько тысяч условных единиц видна на экране? В какой момент происходит перевод пользовательской системы координат в систему координат CVV. Проекционные матрицы – это как раз та сущность, которая занимается решением этих вопросов.

Главная мысль вышеизложенного – разработчик сам волен выбрать тип пользовательской системы координат и должен корректно описать проекционные матрицы. На этом с фактами об OpenGL закончено и подошло время сводить всё воедино.

Одна из наиболее распространённых и сложно постигаемых матриц – это матрица перспективного преобразования. Так как же она связана с CVV и пользовательской системой координат? Почему объекты с увеличением расстояния до наблюдателя становятся меньше? Для того чтобы понять почему объекты уменьшаются с увеличением расстояния, давайте рассмотрим матричные преобразования трёхмерной модели шаг за шагом. Не секрет, что любая трёхмерная модель состоит из конечного списка вершин, которые подвергаются матричным преобразованиям совершенно независимо друг от друга. Для того чтобы определить координату трёхмерной вершины на двухмерном экране монитора необходимо:

1. Перевести декартову координату в однородную координату;
2. Умножить однородную координату на модельную матрицу;
3. Результат умножить на видовую матрицу;
4. Результат умножить на проекционную матрицу;
5. Результат перевести из однородных координат в декартовы координаты.

Перевод декартовой координаты в однородную координату обсуждался ранее. Геометрический смысл модельной матрицы заключается в том, чтобы перевести модель из локальной системы координат в глобальную систему координат. Или как говорят, вынести вершины из модельного пространства в мировое пространство. Скажем проще, загруженный из файла трёхмерный объект находится в модельном пространстве, где координаты отсчитываются относительно самого объекта. Далее с помощью модельной матрицы производится позиционирование, масштабирование и поворот модели. В результате все вершины трёхмерной модели получают фактические однородные координаты в трёхмерной сцене. Модельное пространство относительно мирового пространства является локальным. Из модельного пространства координаты выносятся в мировое пространство (из локального в глобальное). Для этого используется модельная матрица.

Теперь переходим к шагу три. Здесь начинает работу видовое пространство. В этом

пространстве координаты отсчитываются относительно положения и ориентации наблюдателя так, как если бы он являлся центром мира. Видовое пространство является локальным относительно мирового пространства, поэтому координаты в него надо вносить (а не выносить, как в предыдущем случае). Прямое матричное преобразование выносит координаты из некоторого пространства. Чтобы наоборот внести их в него, надо матричное преобразование инвертировать, поэтому видовое преобразование описывается обратной матрицей. Как же получить эту обратную матрицу? Для начала получим прямую матрицу наблюдателя. Чем характеризуется наблюдатель? Наблюдатель описывается координатой, в которой он находится, и векторами направления обзора. Наблюдатель всегда смотрит в направлении своей локальной оси *Z*. Наблюдатель может перемещаться по сцене и осуществлять повороты. Во многом это напоминает смысл модельной матрицы. По большому счёту так оно и есть. Однако, для наблюдателя операция масштабирования бессмысленна, поэтому между модельной матрицей наблюдателя и модельной матрицей трёхмерного объекта нельзя ставить знак равенства. Модельная матрица наблюдателя и есть искомая прямая матрица. Инвертировав эту матрицу, мы получаем видовую матрицу. На практике это означает, что все вершины в глобальных однородных координатах получат новые однородные координаты относительно наблюдателя. Соответственно, если наблюдатель видел определённую вершину, то значение однородной координаты *z* данной вершины в видовом пространстве точно будет положительным числом. Если вершина находилась за наблюдателем, то значение её однородной координаты *z* в видовом пространстве точно будет отрицательным числом.

Шаг четыре — это самый интересный шаг. Предыдущие шаги были рассмотрены так подробно намеренно, чтобы читатель имел полную картину о всех операндах четвёртого шага. На четвёртом шаге однородные координаты выносятся из видового пространства в пространство CVV. Ещё раз подчеркивается тот факт, что все потенциально видимые вершины будут иметь положительное значение однородной координаты *z*.

Рассмотрим матрицу вида:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad (1)$$

И точку в однородном пространстве наблюдателя:

$$\begin{bmatrix} x_v & y_v & z_v & 1 \end{bmatrix}$$

Произведём умножение однородной координаты на рассматриваемую матрицу:

$$\begin{bmatrix} x_v & y_v & z_v & 1 \end{bmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{bmatrix} x_v & y_v & z_v + 1 & z_v \end{bmatrix}$$

Переведём получившиеся однородные координаты в декартовы координаты:

$$\begin{bmatrix} x_v & y_v & z_v & 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} \frac{x_v}{z_v} & \frac{y_v}{z_v} & \frac{z_v + 1}{z_v} \end{bmatrix}$$

Допустим, есть две точки в видовом пространстве с одинаковыми координатами *x* и *y*, но разными координатами *z*. Другими словами одна из точек находится за другой. Из-за перспективного искажения наблюдатель должен увидеть обе точки. Действительно, из формулы видно, что из-за деления на координату *z*, происходит сжатие к точке начала координат. Чем больше значение *z* (чем дальше точка от наблюдателя), тем сильнее сжатие. Вот и объяснение эффекту перспективы.

В спецификации OpenGL сказано, что операции по отсечению и растеризации выполняются в декартовых координатах, а процесс перевода однородных координат в декартовы координаты производится автоматически.

Матрица (1) является шаблоном для матрицы перспективой проекции. Как было сказано ранее, задача матрицы проекции заключается в двух моментах: установка пользовательской системы координат (левосторонняя или правосторонняя), перенос объёма видимости наблюдателя в CVV. Выведем перспективную матрицу для левосторонней пользовательской системы координат.

Матрицу проекции можно описать с помощью четырёх параметров (рис. 3):

- Угол обзора в радианах (*fovy*);
- Соотношение сторон (*aspect*);
- Расстояние до ближней плоскости отсечения (*n*);
- Расстояние до дальней плоскости отсечения (*f*).

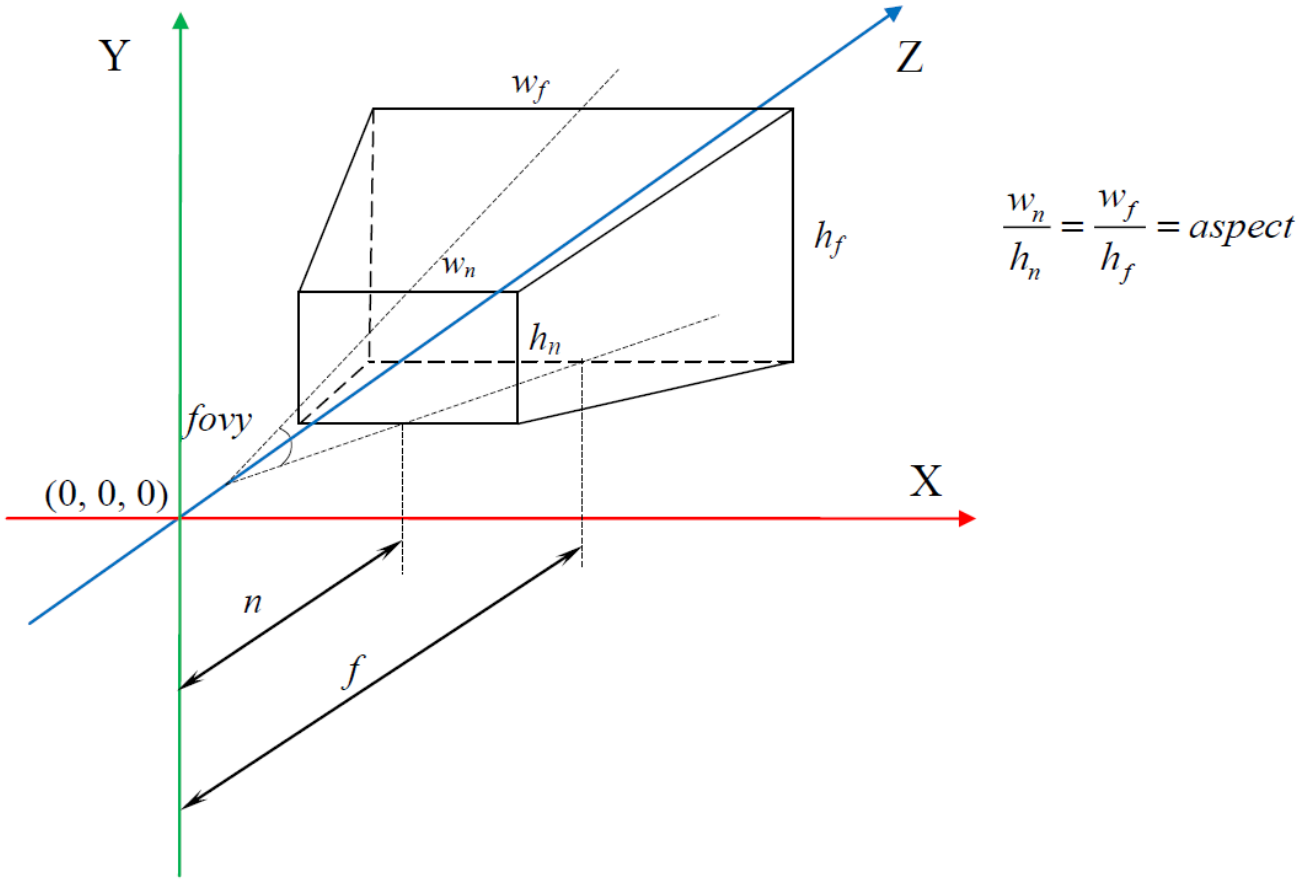

Рис. 3. Перспективный объём видимости

Рассмотрим проекцию точки в пространстве наблюдателя на переднюю грань отсечения перспективного объёма видимости. Для большей наглядности на рис. 4 изображён вид сбоку. Так же следует учесть, что пользовательская система координат совпадает с системой координат CVV, то есть везде пользуется левосторонняя система координат.

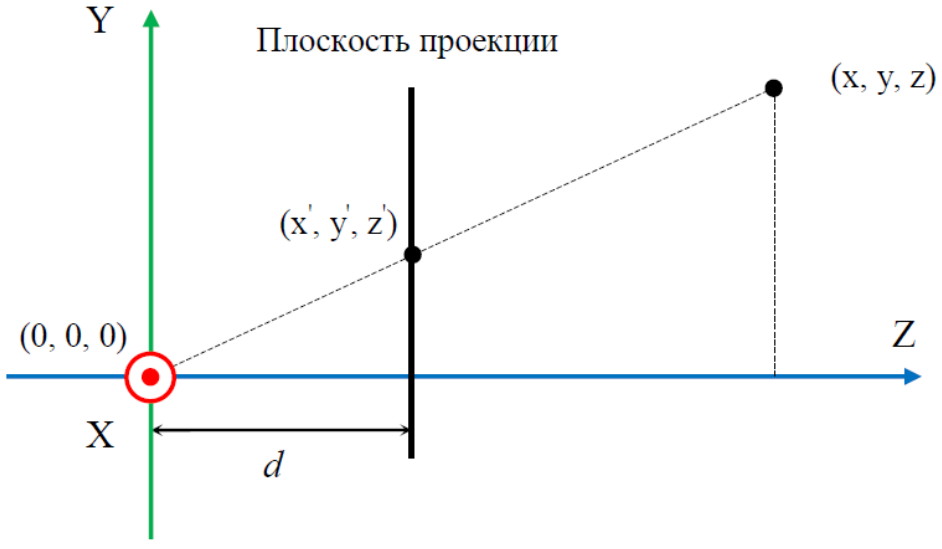

Рис. 4. Проецирование произвольной точки

На основании свойств подобных треугольников справедливы следующие равенства:

$$\frac{y}{y'} = \frac{z}{d} \qquad \frac{x}{x'} = \frac{z}{d}$$

Выразим y' и x':

$$y' = y \cdot \frac{d}{z} \qquad x' = x \cdot \frac{d}{z} \qquad (2)$$

В принципе, выражений (2) достаточно для получения координат точек проекции. Однако для правильного экранирования трёхмерных объектов необходимо знать глубину каждого фрагмента. Другими словами, необходимо хранить значение компоненты *z*. Как раз это значение используется при тестах глубины OpenGL. На рис. 3 видно, что значение *z'* не подходит в качестве глубины фрагмента, потому что все проекции точек умеют одинаковое значение *z'*. Выход из сложившейся ситуации – использование так называемой псевдоглубины.

Свойства псевдоглубины:

1. Псевдоглубина рассчитывается на основании значения *z*;
2. Чем ближе к наблюдателю находится точка, тем меньшее значение имеет псевдоглубина;
3. У всех точек, лежащих на передней плоскости объёма видимости, значение псевдоглубины равно -1;
4. У всех точек, лежащих на дальней плоскости отсечения объёма видимости, значение псевдоглубины равно 1;
5. Все фрагменты, лежащие внутри объёма видимости, имеют значение псевдоглубины в диапазоне [-1 1].

Давайте выведем формулу, по которой будет рассчитываться псевдоглубина. В качестве основы возьмём следующее выражение:

$$z^* = \frac{az + b}{z} \qquad (3)$$

Коэффициенты *a* и *b* необходимо вычислить. Для того чтобы это сделать, воспользуемся свойствами псевдоглубины 3 и 4. Получаем систему из двух уравнений с двумя неизвестными:

$$\begin{cases} -1 = \dfrac{an+b}{n} & (4) \\ 1 = \dfrac{af+b}{f} & (5) \end{cases}$$

Произведём сложение обоих частей системы и умножим результат на произведение *fn*, при этом *f* и *n* не могут равняться нулю. Получаем:

$$0 = f\left(an+b\right)+n\left(af+b\right)$$

Раскроем скобки и перегруппируем слагаемые так, чтобы слева осталась только часть с *a*, а справа только с *b*:

$$0 = afn+bf+afn+bn$$

$$-2afn = bf+bn$$

$$a = \frac{b\left(f+n\right)}{-2fn} \qquad (6)$$

Подставим (6) в (5). Преобразуем выражение к простой дроби:

$$1 = \frac{\dfrac{bf\left(f+n\right)}{-2fn}+b}{f}$$

$$1 = \frac{b\left(f+n\right)-2bn}{-2fn}$$

Умножим обе стороны на *-2fn*, при этом *f* и *n* не могут равняться нулю. Приведём подобные, перегруппируем слагаемые и выразим *b*:

$$-2fn = bf+bn-2nb$$

$$-2fn = bf-bn$$

$$b\left(f-n\right) = -2fn$$

$$b = \frac{-2fn}{f-n} \qquad (7)$$

Подставим (7) в (6) и выразим *a*:

$$a = \frac{-2fn(f+n)}{(f-n)(-2fn)}$$

$$a = \frac{f+n}{f-n}$$

Соответственно компоненты *a* и *b* равны:

$$a = \frac{f+n}{f-n}$$

$$b = \frac{-2fn}{f-n}$$

Теперь подставим полученные коэффициенты в матрицу заготовку (1) и проследим, что будет происходить с координатой *z* для произвольной точки в однородном пространстве наблюдателя. Подстановка выполняется следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & 1 \\ 0 & 0 & b & 0 \end{pmatrix} \quad \text{или} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & 1 \\ 0 & 0 & \frac{-2fn}{f-n} & 0 \end{pmatrix} \qquad (8)$$

Пусть расстояние до передней плоскости отсечения *n* равно 2, а расстояние до дальней плоскости отсечения *f* равно 10. Рассмотрим пять точек в однородном пространстве наблюдателя:

Взаимное расположение точки и объёма видимости

Точка	**Значение** z_v	**Описание**
1	1	Точка находится перед передней плоскостью отсечения объёма видимости. Не проходит растеризацию.
2	2	Точка находится на передней грани отсечения объёма видимости. Проходит растеризацию.
3	5	Точка находится между передней гранью отсечения и дальней гранью отсечения объёма видимости. Проходит растеризацию.
4	10	Точка находится на дальней грани отсечения объёма видимости. Проходит растеризацию.
5	20	Точка находится за дальней гранью отсечения объёма видимости. Не проходит растеризацию.

Умножим все точки на матрицу (8), а затем переведём полученные однородные координаты в декартовые координаты z_d. Для этого нам необходимо вычислить значения новых однородных компонент z_h и w_h.

Точка 1:

$$z_h = z_v \cdot \frac{f+n}{f-n} + 1 \cdot \frac{-2fn}{f-n} = 1 \cdot \frac{10+2}{10-2} + 1 \cdot \frac{-2 \cdot 10 \cdot 2}{10-2} = \frac{12}{8} + \frac{-40}{8} = \frac{-28}{8}$$

$$w_h = z \cdot 1 = 1 \cdot 1 = 1$$

$$z_d = \frac{z_h}{w_h} = \frac{-28}{8 \cdot 1} = -3.5$$

Точка 2:

$$z_h = z_v \cdot \frac{f+n}{f-n} + 1 \cdot \frac{-2fn}{f-n} = 2 \cdot \frac{10+2}{10-2} + 1 \cdot \frac{-2 \cdot 10 \cdot 2}{10-2} = \frac{24}{8} + \frac{-40}{8} = \frac{-16}{8}$$

$$w_h = z \cdot 1 = 2 \cdot 1 = 2$$

$$z_d = \frac{z_h}{w_h} = \frac{-16}{8 \cdot 2} = -1$$

Точка 3:

$$z_h = z_v \cdot \frac{f+n}{f-n} + 1 \cdot \frac{-2fn}{f-n} = 5 \cdot \frac{10+2}{10-2} + 1 \cdot \frac{-2 \cdot 10 \cdot 2}{10-2} = \frac{60}{8} + \frac{-40}{8} = \frac{20}{8}$$

$$w_h = z \cdot 1 = 5 \cdot 1 = 5$$

$$z_d = \frac{z_h}{w_h} = \frac{20}{8 \cdot 5} = 0.5$$

Точка 4:

$$z_h = z_v \cdot \frac{f+n}{f-n} + 1 \cdot \frac{-2fn}{f-n} = 10 \cdot \frac{10+2}{10-2} + 1 \cdot \frac{-2 \cdot 10 \cdot 2}{10-2} = \frac{120}{8} + \frac{-40}{8} = \frac{80}{8}$$

$$w_h = z \cdot 1 = 10 \cdot 1 = 10$$

$$z_d = \frac{z_h}{w_h} = \frac{80}{8 \cdot 10} = 1$$

Точка 5:

$$z_h = z_v \cdot \frac{f+n}{f-n} + 1 \cdot \frac{-2fn}{f-n} = 20 \cdot \frac{10+2}{10-2} + 1 \cdot \frac{-2 \cdot 10 \cdot 2}{10-2} = \frac{240}{8} + \frac{-40}{8} = \frac{200}{8}$$

$$w_h = z \cdot 1 = 20 \cdot 1 = 20$$

$$z_d = \frac{z_h}{w_h} = \frac{200}{8 \cdot 20} = 1.25$$

Обратите внимание, что однородная координата z_v абсолютно верно позиционируется в CVV, а самое главное, что теперь возможна работа теста глубины OpenGL, потому что псевдоглубина полностью удовлетворяет требованиям тестов.

С координатой *z* разобрались, перейдём к координатам *x* и *y*. Как говорилось ранее весь перспективный объём видимости должен умещаться в CVV. Длина ребра CVV равна двум. Соответственно, высоту и ширину перспективного объёма видимости надо сжать до двух условных единиц.

$$\frac{2n}{w} \quad \text{и} \quad \frac{2n}{h}$$

В нашем распоряжении имеется угол *fovy* и величина *aspect*. Давайте выразим высоту и ширину, используя эти величины.

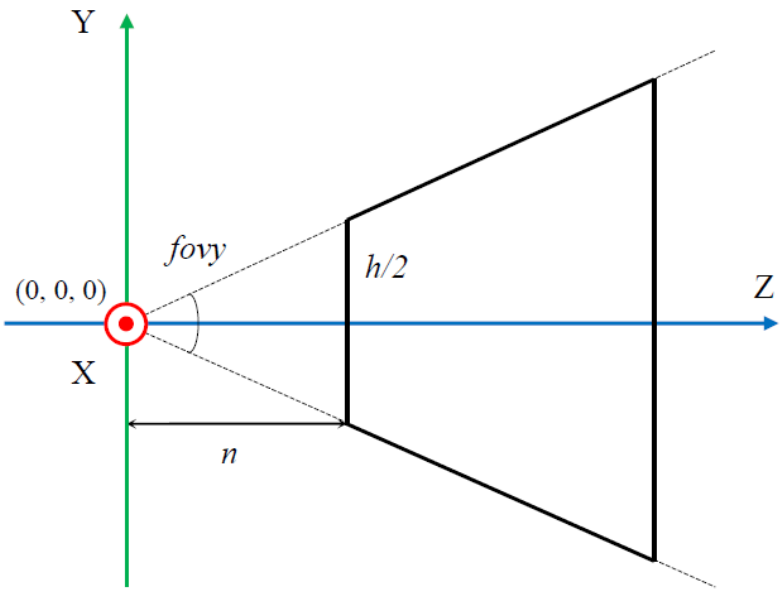

Рис. 5. Объём видимости

Из рис. 5 видно, что:

$$\frac{2n}{h} = ctg\left(\frac{fovy}{2}\right)$$

$$aspect = \frac{w}{h}$$

$$\frac{2n}{w} = \frac{2n}{aspect \cdot h} = \frac{2n}{aspect \cdot \dfrac{2n}{ctg\left(\dfrac{fovy}{2}\right)}} = \frac{ctg\left(\dfrac{fovy}{2}\right)}{aspect}$$

Теперь можно получить окончательный вид перспективной проекционной матрицы для пользовательской левосторонней системы координат, работающей с CVV OpenGL:

$$\begin{pmatrix} \dfrac{ctg\left(\dfrac{fovy}{2}\right)}{aspect} & 0 & 0 & 0 \\ 0 & ctg\left(\dfrac{fovy}{2}\right) & 0 & 0 \\ 0 & 0 & \dfrac{f+n}{f-n} & 1 \\ 0 & 0 & \dfrac{-2fn}{f-n} & 0 \end{pmatrix}$$

На этом вывод матриц закончен.

Пару слов о DirectX — основном конкуренте OpenGL. DirectX отличается от OpenGL только габаритами CVV и его позиционированием. В DirectX CVV — это прямоугольный параллелепипед с длинами по осям *x* и *y* равными двойке, а по оси *z* длина равна единице. Диапазон *x* и *y* равен [-1 1], а диапазон *z* равен [0 1]. Что касается системы координат CVV, то в DirectX, как и в OpenGL, используется левосторонняя система координат.

Для вывода перспективных матриц для пользовательской правосторонней системы координат необходимо перерисовать рис. 2, рис.3 и рис.4 с учётом нового направления оси *Z*. Далее расчёты полностью аналогичны, с точностью до знака. Для матриц DirectX свойства псевдоглубины 3 и 4 модифицируются под диапазон [0 1].

На этом тему перспективных матриц можно считать закрытой.