

1. Двумерные геометрические (аффинные) преобразования. Композиция и коммутативность геометрических преобразований

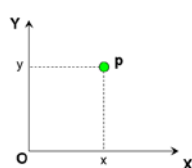
точка определяется координатным вектором:

вектор-столбец: $\begin{bmatrix} x \\ y \end{bmatrix}$
вектор строка: $\begin{bmatrix} x & y \end{bmatrix}$

Матрица общего преобразования: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

$$P' = PM = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (ax+cy) & (bx+dy) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}$$

$$\begin{aligned} x' &= ax+cy \\ y' &= bx+dy \end{aligned}$$



Единичная матрица

$$\begin{aligned} c &= b = 0 \\ a &= d = 1 \end{aligned}$$

$$\begin{aligned} x' &= 1x+0y = x \\ y' &= 0x+1y = y \end{aligned}$$

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

Масштабирование

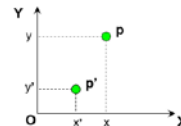
$$c = b = 0 \quad a \neq 0 \quad d \neq 0$$

$$P' = PS = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} = \begin{bmatrix} (a \cdot x + 0 \cdot y) & (0 \cdot x + d \cdot y) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}$$

$$\begin{aligned} x' &= a \cdot x \\ y' &= d \cdot y \end{aligned}$$

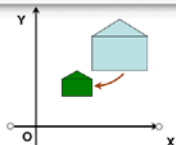
Матрица масштабирования

$$S = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

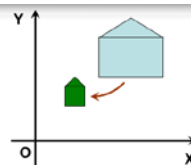


Масштабирование

$$\begin{aligned} Sx &= 0.5 \\ Sy &= 0.5 \end{aligned} \quad S = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$



$$\begin{aligned} Sx &= 0.25 \\ Sy &= 0.5 \end{aligned} \quad S = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.5 \end{bmatrix}$$



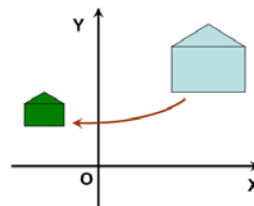
$$P' = PS = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} = \begin{bmatrix} (a \cdot 0 + 0 \cdot 0) & (0 \cdot 0 + d \cdot 0) \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Обратная матрица масштабирования

$$S^{-1} = \begin{bmatrix} 1/Sx & 0 \\ 0 & 1/Sy \end{bmatrix}$$

Масштабирование и отображение

$$\begin{aligned} Sx &= -0.5 \\ Sy &= 0.5 \end{aligned}$$

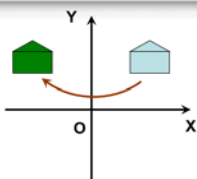


$$S = \begin{bmatrix} -0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Отображение

$$|a| = |d| = 1$$

$$M = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



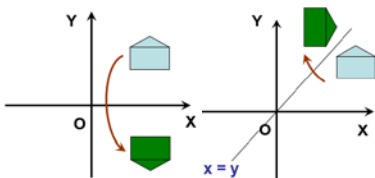
Отображение

$$|c| = |b| = 1 \quad a = d = 0$$

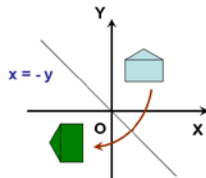
$$P' = PM = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} (0 \cdot x + 1 \cdot y) & (1 \cdot x + 0 \cdot y) \end{bmatrix} = \begin{bmatrix} x' & y' \end{bmatrix}$$

$$x' = y \quad y' = x$$

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



$$M = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$



$$M = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Масштабирование – умножение исходных координат на масштабирующий коэф.

Sx, Sy – масштабирующие коэф. $S > 1 \Rightarrow$ **Расширение**, $S < 1 \Rightarrow$ **Сжатие**.

$Sx \neq Sy \Rightarrow$ искажение формы объекта (искажение пропорций).

Если масштабирующий коэф. < 0 , то наряду с масштабированием будет происходить **отображение** объектов. Если дать единичный коэф. по модулю, то будет только отображение, без масштабирования.

Поворот.

Происходит вокруг некоторой точки. Наиболее просто описывается относительно начала координат.



Перенос

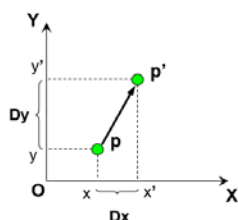
Преобразование переноса определяется неким вектором с учетом значения переноса

Перенос

$$\begin{aligned}x' &= x + Dx \\ y' &= y + Dy\end{aligned}$$

$$T = \begin{bmatrix} Dx & Dy \end{bmatrix}$$

$$P' = P + T$$



Однородные координаты

Для того, чтобы описать в матричной форме преобразование переноса. Точка на плоскости определяется тройкой X, Y, W . Эти координаты могут быть получены из декартовых умножением на масштабирующий коэф. Декартовы можно получить делением этих на масштабирующий. Чтобы перевести в однородные координаты матрицы поворота и масштабирования, они приводятся к размеру 3×3 , добавленные элементы заполняются 0, кроме главной диагонали, где ставятся 1.

Скос

$$P = \begin{bmatrix} W \cdot x & W \cdot y & W \end{bmatrix} \quad W \neq 0$$

$$P = \begin{bmatrix} X & Y & W \end{bmatrix} \quad W \neq 0$$

$$x = X/W \quad y = Y/W$$

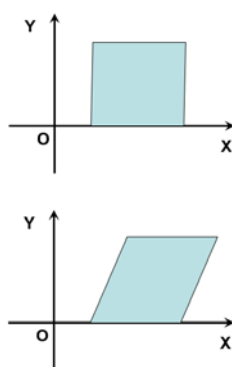
$$W = 1 \quad x = X/1 \quad y = Y/1$$

$$\begin{aligned}x' &= x + y \cdot \operatorname{ctg} \theta \\ y' &= y\end{aligned}$$

$$S = \begin{bmatrix} 1 & 0 & 0 \\ \operatorname{ctg} \theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}x' &= x + Dx \\ y' &= y + Dy\end{aligned}$$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Dx & Dy & 1 \end{bmatrix}$$



$$P * M = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix} = \begin{bmatrix} (a \cdot x + c \cdot y + m) & (b \cdot x + d \cdot y + n) & (1) \end{bmatrix} = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$$

Произвольный поворот

Выполняется 1) преобразование переноса на вектор до начала координат 2) выполняется поворот на требуемый угол 3) перенос в начальное положение (перед 1 действием)

Композиция и Коммутативность

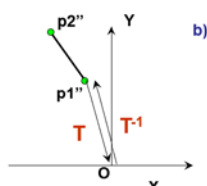
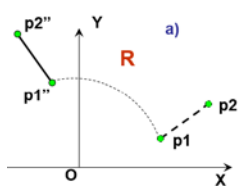
В общем случае геометрические аффинные преобразования на плоскости и в пространстве не коммутативны (нельзя менять местами порядок). Композиция состоит в возможности сложения разных действий для получения результата.

Композиция

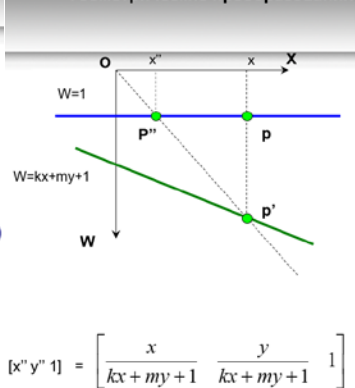
$$P' = P * T * R * T^{-1} = P * M_c$$

$$P'' = P * R * T * T^{-1}$$

$$P' \neq P''$$



Геометрические преобразования



Коммутативность

M1	M2
Перенос	Перенос
Масштабирование	Масштабирование
Поворот (вокруг одной и той же оси)	Поворот
Масштабирование (при $S_x = S_y = S_z$)	Поворот

2. Геометрические (аффинные) преобразования в пространстве. Композиция и коммутативность геометрических преобразований

Также используются прямоугольные декартовы координаты, которые бывают 2 видов: левосторонние и правосторонние. Формируются соответственно по правилу левой и правой руки. Если указательный палец левой руки = Y, большой = X, то средний = Z. Правосторонние координаты более привычны, поэтому в них происходит описание моделей чаще всего. Для визуализации более естественны левосторонние координаты.

Отображение относительно плоскостей. **Поворот** производится относительно векторов.

Масштабирование

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Перенос

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix}$$

Отображение

Поворот $R=R_x R_y R_z$

$$R_z = \begin{bmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{xz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Произвольный поворот

1) Для выполнения поворота вокруг произвольного вектора необходимо совместить точку привязку с началом координат за счет преобразования переноса, определяющегося вектором, соединяющим точку привязки с началом координат. Получим ситуацию на рисунке.

2) Далее, необходимо совместить вектор с 1 из координатных положительных полуосей. Мы возьмем полуось Z. Выполнить поворот вокруг оси X на угол α , переводящий вектор в плоскость XOZ.

3) Выполнить поворот вокруг положительной полуоси Y на угол β , совмещающий вектор с полуосью Z.

4) Выполнить поворот вокруг положительной полуоси Z на требуемый угол.

5) Выполнить обратные преобразования, приводящие сцену в исходное состояние не меняя порядок. (т.е. последовательность действий: 1-2-3-4-3-2-1) Композиция из двумерных, но суть та же)

Коммутативность

M1	M2
Перенос	Перенос
Масштабирование	Масштабирование
Поворот (вокруг одной и той же оси)	Поворот
Масштабирование (при $S_x=S_y=S_z$)	Поворот

Композиция

$$P' = P * T * R * T^{-1} = P * M_c$$

$$P'' = P * R * T * T^{-1}$$

$$P' \neq P''$$

Геометрические преобразования как изменения координатных систем

Геометрические преобразования как изменения координатных систем

Геометрические преобразования как изменения координатных систем

Преобразование всех точек модели = изменение системы координат.

Проекции(общее)

При выводе пространственных объектов они отсекаются по границе видимого объема и после этого преобразуются в поле вывода. Несоответствие между (чем?) устраняется путем введения проекции, которая отображает 3мерный объект на 2мерной проекционной плоскости. В общем случае, проекции преобразуют систему координат размером N в размерность меньше N .

Проекция 3мерного объекта представляется совокупностью точек. Строится при помощи прямых проекционных лучей проектора, проходящих через каждую точку объекта и пересекающих проекционную плоскость, образуя проекцию. Описанный таким образом класс называется **плоскими геометрическими проекциями**

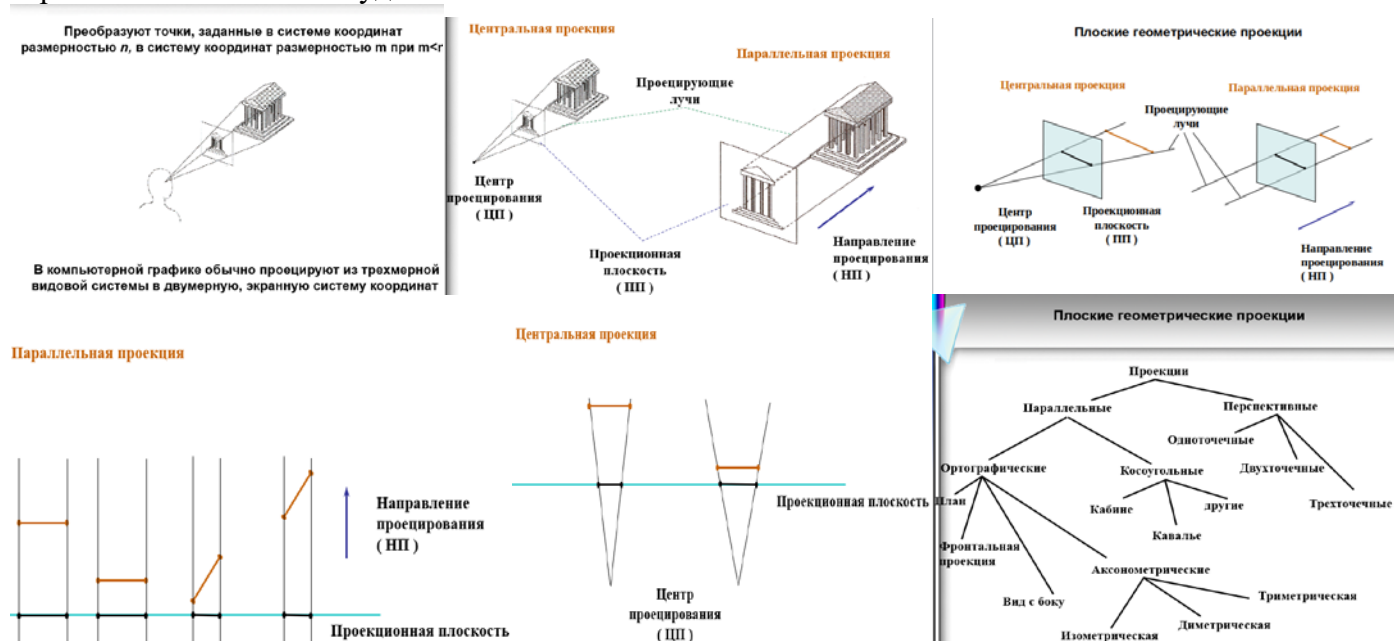
Выделяют 2 класса:

- Центральные (перспективные)
- параллельные

Разница в положении центра проецирования.

Если центр унесен в бесконечность, то это параллельные в силу того, что лучи проектора становятся параллельными. В таком случае говорят о направлении проецирования.

Если центр находится близко, то проекции центральные (перспективные). Перспектива отражает зависимость от удаленности.



В ходе проецирования преобразованные размеры проекции могут отличаться от размеров объекта.

Укорачивание – отличие длины отрезка от длины проекции

Величина укорачивания описывается коэф. укорачивания = длина отрезка/длина проекции.

Если длина объекта отличается от длины проекции, то получаем коэффициент укорачивания (КУ), который зависит от угла наклона относительно проекционной плоскости.

В параллельных проекциях зависит от угла наклона отрезка к проекционной плоскости. Если отрезок параллелен проекционной плоскости, то укорачивание отсутствует. Не может быть больше 1. Не зависит от расстояния между объектом и проекционной областью. На параллельных проекциях можно произвести вычисления с точностью до скалярного множества. Объекты на одном расстоянии и под одним углом будут иметь одинаковый КУ.

В центральных(перспективных) проекциях зависит как от угла наклона, так и от расстояния. В общем случае, имеет место перспективное укорачивание. Прямые могут быть не параллельны, углы искажаются (?). Проекции параллельных линий пересекаются, если не лежат в плоскости, параллельной плоскости проецирования. Объекты, которые лежат в плоскости, параллельной плоскости проецирования, имеют одинаковый КУ

3. Плоские геометрические проекции. Ортогографические проекции.

(Орто – направленный под 90)

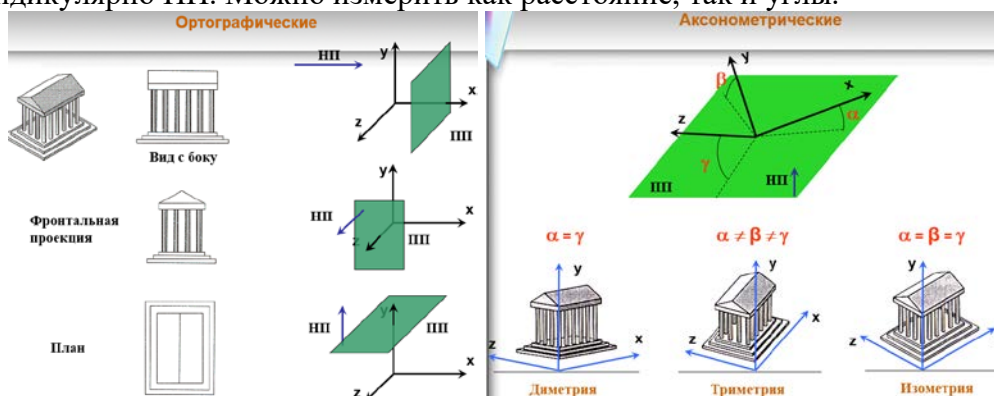
В зависимости от соотношения нормали проекционной плоскости (ПП) и направления проецирования выделяют:

- Если совпадают — ортогографические
- Если нет — косоугольные

Ортогографические делят на:

- План/вид сверху (по полуоси Y)
- Вид спереди (по полуоси Z)
- Вид сбоку (по полуоси X)

Соответствие оси главной координатной системы объекта (в которой он описан) перпендикулярно ПП. Можно измерить как расстояние, так и углы.



В аксонометрических проекциях ни одна из главных координатных осей не перпендикулярна ПП, следовательно, отображается несколько сторон объекта. В этом схожесть с центральными (но здесь другие КУ), но в силу зависимости КУ только от угла, можно изменить только размер и расстояние, делая поправки на постоянный коэффициент вдоль каждой оси.

Сохраняется параллельность прямых, а углы искажаются.

Аксонометрические делятся на:

- Изометрия (углы между нормалью ПП и главными координатными осями объекта (углы наклона осей к ПП) равны (КУ равны, измеряем расстояние с 1 коэф.). изо – равный (по всем осям). углы наклона равны между нормалью и коорд. системой)
- Диметрия (2 угла одинаковы, по 2 одинаково измеряем, 2 КУ)
- Триметрия (3 угла разные)

Аксонометрические проекции

Аксонометрические проекции

Аксонометрические проекции

$$[T] = [R_x][R_y][P_z] =$$

$$\begin{bmatrix} \cos \varphi & 0 & -\sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[U][T] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} [T] = \begin{bmatrix} \cos \varphi & \sin \varphi \sin \theta & 0 & 1 \\ 0 & \cos \theta & 0 & 1 \\ \sin \varphi & -\cos \varphi \sin \theta & 0 & 1 \end{bmatrix} \begin{bmatrix} x_x' & y_x' & 0 & 1 \\ x_y' & y_y' & 0 & 1 \\ x_z' & y_z' & 0 & 1 \end{bmatrix}$$

$$f_x^2 = x_x'^2 + y_x'^2 = \cos^2 \varphi + \sin^2 \varphi \sin^2 \theta \quad (5-1)$$

$$f_y^2 = x_y'^2 + y_y'^2 = \cos^2 \theta \quad (5-2)$$

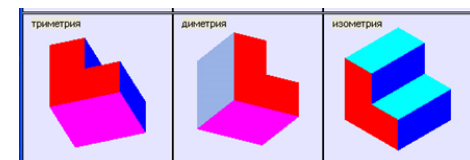
$$f_z^2 = x_z'^2 + y_z'^2 = \sin^2 \varphi + \cos^2 \varphi \sin^2 \theta \quad (5-3)$$

$$[T] = \begin{bmatrix} \cos \varphi & \sin \varphi \sin \theta & 0 & 0 \\ 0 & \cos \theta & 0 & 0 \\ \sin \varphi & -\cos \varphi \sin \theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$f_x = \sqrt{x_x'^2 + y_x'^2}$$

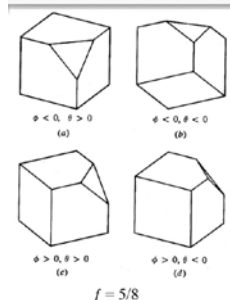
$$f_y = \sqrt{x_y'^2 + y_y'^2}$$

$$f_z = \sqrt{x_z'^2 + y_z'^2}$$



Диметрические проекции

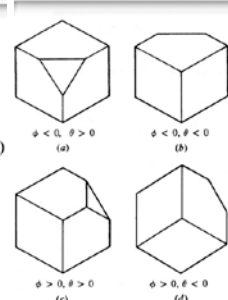
Изометрические проекции



$$\varphi = \arcsin(\pm f_z / \sqrt{2 - f_z^2})$$

$$\theta = \arcsin(\pm f_z / \sqrt{2})$$

$$f = 5/8$$



$$\varphi = \pm 45^\circ$$

вокруг Y

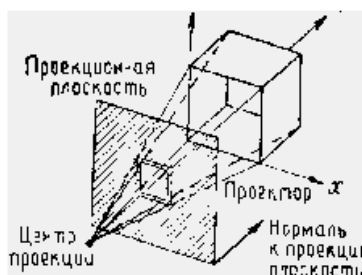
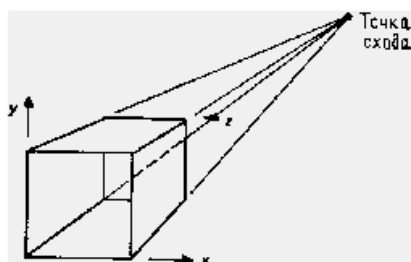
$$\theta = \pm 35,264^\circ$$

вокруг X

$$f = \sqrt{\cos^2 \theta} = \sqrt{2/3} = 0.8165$$

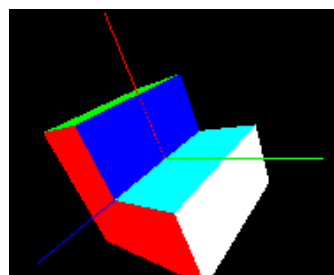
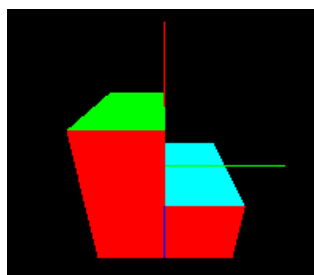
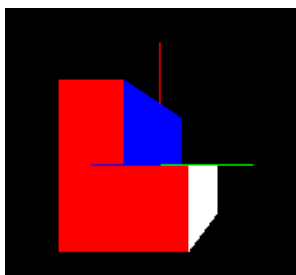
4. Плоские геометрические проекции. Центральные проекции.

Центральная проекция — совокупность параллельных прямых, которые не параллельны ПП и сходятся в точке схода (не центре проекции!)



Если параллельны одной из осей, то точка схода называется главной точкой схода. Выделяют:

- Одноточечную Двухточечную Трехточечную



1 и 2 точечная находят наибольшее применение, 3 точечная дает мало новой информации, а сложность возрастает.

Одноточечное преобразование задается матрицей:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & rz+1 \end{bmatrix}$$

При применении масштабирующего коэфф. не равного 1, чтобы вернуть точку в декартовы координаты нужно поделить координаты на величину этого масшт. коэфф. - перспективное деление.

Если расположить масш. коэфф. на той строке где связь с X, Y, Z, то точка схода будет на этой оси.

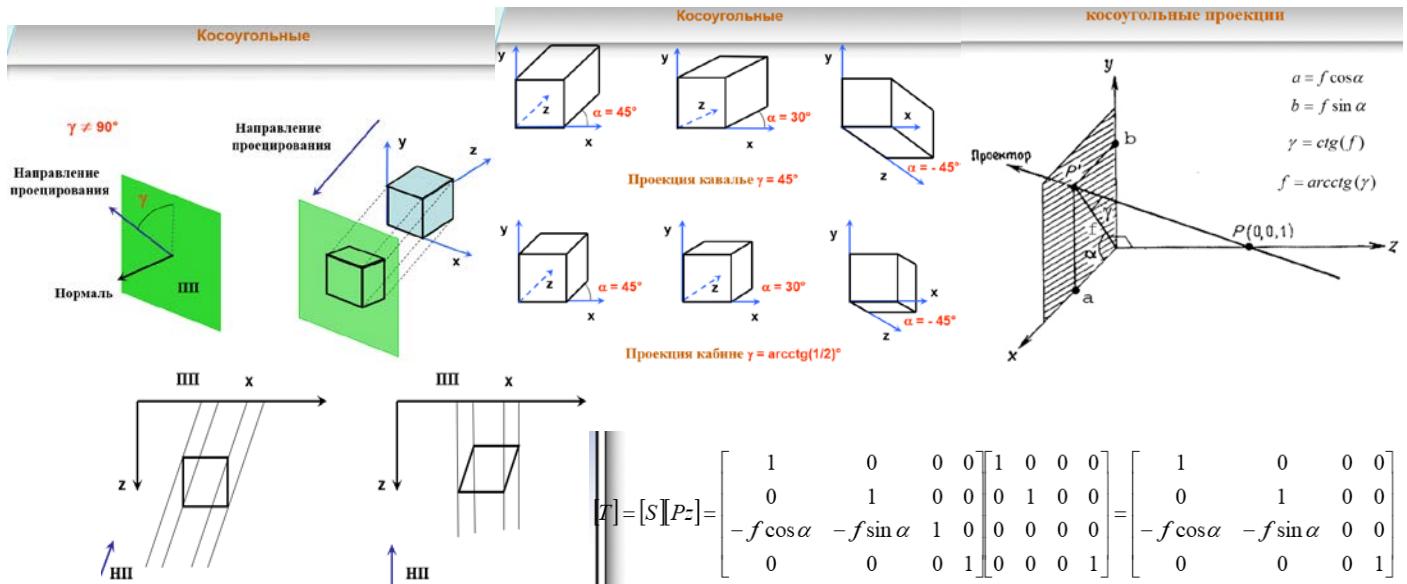
5. Плоские геометрические проекции. Косоугольные проекции.

Косоугольные проекции является смесью ортографических (отсутствие укорачивания) и аксонометрических (видно несколько сторон).

Две главные координатные оси параллельны ПП, следовательно КУ = 0.

КУ по 3ей оси зависит от угла наклона (между направлением проецирования и ПП):

- Кавалье (КУ по Z = 0, угол (между направлением проецирования и ПП) = 45)
- Кабине (КУ по Z = 1/2 (наиболее соответствует нашему зрительному блику(?) и для нас более естественны, угол (между проекцией направления проецирования на ПП и оси X главной координатной системы объекта ПП/оси, параллельной оси проецирования) = $\arcsctg(1/2)$)



Свет. Общее

Физическая природа света очень сложна и расчет модели освещения, близкий к реальности, потребует больших затрат. Поэтому, в компьютерной графике используется упрощенная модель освещения.

Свет, падающий на поверхность объекта, может быть отражен, поглощен или пропущен.

Объект можно увидеть только если он поглощает или отражает свет.

Если объект пропускает весь свет, то он невидим и называется абсолютно черным.

Если объект отражает весь падающий свет, то называется зеркалом, или идеальным отражателем(?)

Если при освещении объекта светом поглощается весь свет, то объект будет черным, если поглощаются только определенные длины волн, то он будет цветным. **Цвет – свойство отражения.**

Отраженный от объекта свет мб диффузным или зеркальным.

Чтобы сделать имитацию освещения объектов, необходимо создать модель, дающую реалистичные результаты и обладающую высокой эффективностью.

Выделяют:

1) Глобальные модели

При расчете освещенности учитываются как прямо падающий от источника свет, так и свет, отраженный или преломленный другими источниками. Отражаясь от некоторых объектов, свет падает на другой объект, меняя условия его освещения. Отраженный от него падает на исходный и также меняет его свойства, т.е. мы приходим к рекурсивному процессу. Рост вычислительных затрат на пиксели(?) и отсюда (?) примитивность таких моделей.

2) Локальные модели

Опирается на свет, идущий от источника. Следовательно, нет теней. В результате, влияние других объектов сцены исключается.

Из соображений эффективности расчет интенсивности выполняется только для некоторых точек

6. Простая модель освещения

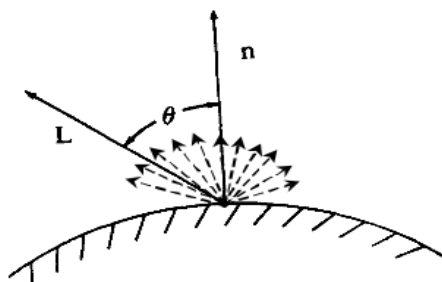
Диффузный свет – главная составляющая освещения.

Если мы будем строить модель только на основе диффузного отражения, надо учесть, что отражение света происходит за счет шероховатости поверхности объекта. Лучи света, отраженные от идеального диффузного отражателя, расходятся равномерно во всех направлениях.

Есть угол падения, и от него будет зависеть только интенсивность(световой поток на единицу площади(?)).

Говорят, что свет как бы поглощается поверхностью, а затем вновь испускается.

$$I_d = I_l K_d \cos \theta \quad 0 \leq \theta \leq \pi/2$$



I_d – интенсивность диффузного света

I_l – интенсивность источника света

K_d – коэффициент диффузного отражения $0 < K_d < 1$. 0 – нет отражения; 1 – идеальный диффузный отражатель. Его величина зависит от отражающего материала и от свойств падающего света.

Θ – угол падения луча

Если Θ не принадлежит $[0; \pi/2]$, то источник света находится за объектом и его надо исключить из расчета.

N – нормаль поверхности в данной точке

L – вектор направленный на источник(?)

Если учитывать только диффузный свет, модели будут слишком контрастными (места куда не попадает свет будут черными). Поэтому нужно добавить рассеянный свет.

Рассеянный свет – константа окружающего освещения (свет, отраженный от окружающей обстановки).

Модель диффузного освещения с рассеянным светом:

$$I_d = I_a K_a + I_l K_d \cos \theta \quad 0 \leq \theta \leq \pi/2$$

I_a – интенсивность рассеянного света

K_a – коэффициент отражения рассеянного света

Чтобы объекты не сливались нужно учесть затухание. В реальном мире интенсивность света изменяется обратно пропорционально квадрату расстояния до источника света. В простой модели квадратичное затухание не используется т.к. свет затухает слишком сильно. Вместо этого используется линейное затухание. Рассеянный свет оно не затрагивает.

Диффузное освещение с затуханием:

$$I_d = I_a K_a + I_l K_d * \cos(\Theta) / (d + K)$$

d – расстояние до центра проекции

K – произвольная постоянная

Зеркальный свет – производит блики.

$$I_s = I_l * w(\Theta, \lambda) * \cos^n \alpha$$

$w(\Theta, \lambda)$ – кривая отражения (Θ – угол падения луча, λ – длина волны)

$\cos^n \alpha$ – функция аппроксимации распределения зеркально отраженного света. Чем больше n – тем ярче блики (маленький конус распределения)

Обычно $w(\Theta, \lambda)$ заменяют на константу. Тогда модель зеркального света:

$$I_s = I_l K_s * \cos^n \alpha$$

Простая модель освещения для одного источника света:

$$I = I_a K_a + I_l (K_d * \cos(\Theta) + K_s * \cos^n \alpha) / (d + K)$$

Для нескольких источников рассеянный свет является общим, а отражения диффузного и зеркального от разных источников суммируются

Для нескольких источников нужно суммировать интенсивности для диффузного и отраженного света:

$$I = I_a K_a + \sum_{j=1}^m I_{1j} (K_d \cos \theta_j + K_s \cos^n \alpha_j) / (d+K)$$

m – число источников

Эту модель можно задать в векторной форме:

$$\cos \theta = \mathbf{n} \cdot \mathbf{L} / |\mathbf{n}| |\mathbf{L}| = \underline{n} \cdot \underline{L}$$

$$\cos \alpha = \mathbf{R} \cdot \mathbf{S} / |\mathbf{R}| |\mathbf{S}| = \underline{R} \cdot \underline{S}$$

$$I = I_a K_a + I_1 (K_d (\underline{n} \cdot \underline{L}) + K_s (\underline{R} \cdot \underline{S})^n) / (d+K)$$

(Представьте, что все латинские буквы ниже написаны с чертой вверху т.е. это векторы. Мне сильно лень рисовать черту для каждого). Все векторы единичные.

\mathbf{n} – нормаль (можно получить декартовым произведением векторов многоугольника)

\mathbf{L} – направление к источнику

\mathbf{R} – вектор отраженного света


\mathbf{S} – направление взгляда

7. OpenGL графические примитивы, координатные системы, трансформации. Основные матрицы и работа с ними

Синтаксис OpenGL

glCommandName

после названия мб суффикс(количество и тип параметров)

Синтаксис команд																												
<p>Описания моделей →  → Образ сцены</p> <p>glEnable() – разрешить glDisable() – запретить</p> <p>Примеры: glEnable(GL_LINE_SMOOTH) glColor3f(0.5, 0.5, 0.5)</p> <p>glGetBoolean(), glGetFloatv(), glGetError()</p> <p>Занести - glPushAttrib() и glPushClientAttrib() Извлечь - glPopAttrib() и glPopClientAttrib()</p>	<p>glCommandName[1 2 3 4][b s i f d ub us ui][V](список параметров)</p> <p>CommandName – имя команды; [1 2 3 4] – допустимые значения числа аргументов команды; [b s i f d ub us ui] – допустимые типы аргументов [V] означает, что в качестве аргумента используется указатель на массив.</p> <table border="1"> <thead> <tr> <th>Символ</th> <th>Тип OpenGL</th> <th>Описание</th> </tr> </thead> <tbody> <tr> <td>b</td> <td>GLbyte</td> <td>8 разрядов, целое</td> </tr> <tr> <td>s</td> <td>GLshort</td> <td>16 разрядов, целое</td> </tr> <tr> <td>i</td> <td>GLint, GLsizei</td> <td>32 разряда, целое</td> </tr> <tr> <td>f</td> <td>GLfloat</td> <td>32 разряда, плавающая точка</td> </tr> <tr> <td>d</td> <td>GLdouble, GLclampd</td> <td>64 разряда, плавающая точка</td> </tr> <tr> <td>ub</td> <td>GLubyte, GLboolean</td> <td>8 разрядов, беззнаковое целое</td> </tr> <tr> <td>us</td> <td>GLushort</td> <td>16 разрядов, беззнаковое целое</td> </tr> <tr> <td>ui</td> <td>GLuint, GLenum</td> <td>32 разряда, беззнаковое целое</td> </tr> </tbody> </table> <p>GL_LINE_STRIP, GL_COLOR_BUFFER_BIT.</p>	Символ	Тип OpenGL	Описание	b	GLbyte	8 разрядов, целое	s	GLshort	16 разрядов, целое	i	GLint, GLsizei	32 разряда, целое	f	GLfloat	32 разряда, плавающая точка	d	GLdouble, GLclampd	64 разряда, плавающая точка	ub	GLubyte, GLboolean	8 разрядов, беззнаковое целое	us	GLushort	16 разрядов, беззнаковое целое	ui	GLuint, GLenum	32 разряда, беззнаковое целое
Символ	Тип OpenGL	Описание																										
b	GLbyte	8 разрядов, целое																										
s	GLshort	16 разрядов, целое																										
i	GLint, GLsizei	32 разряда, целое																										
f	GLfloat	32 разряда, плавающая точка																										
d	GLdouble, GLclampd	64 разряда, плавающая точка																										
ub	GLubyte, GLboolean	8 разрядов, беззнаковое целое																										
us	GLushort	16 разрядов, беззнаковое целое																										
ui	GLuint, GLenum	32 разряда, беззнаковое целое																										

Константы

С префикса GL заглавными буквами, разделяются слова `_GL_LINE_STRIP`

Состояние конвейера OpenGL описывается набором переменных(состояния). Для каждой из них есть значение по умолчанию. Некоторых из них просты и их состояние меняется через **glEnable/glDisable**. Как правило, это переменные управления режимами. В любой момент времени можно запросить у системы текущие значения переменных состояния: **glGet...(type)**. Часто нужно временно задать значение переменной, для этого используется стек атрибутов: **glPush/Pop(Client)Attrib()**.

Изменение состояний некоторых элементов выполняется неявно – при выполнении различных команд. При вызове команды, задающей проекцию, изменится состояние ряда переменных: тип проекции, матрица проецирования, переменная, отвечающая за видимый объем.

Сначала изменение параметров, затем визуализация.

Матричные переменные:

`GL_MODELVIEW` `GL_PROJECTION` `GL_TEXTURE` `GL_COLOR_MATRIX`

По умолчанию, они единичные. С каждым типом матриц связан свой стек. Текущая матрица – та, что на вершине стека. Механизм управления един для всех матриц.

Матрица видового преобразования - **GL_MODELVIEW**;
 Матрица проецирования - **GL_PROJECTION**;
 Матрица текстуры - **GL_TEXTURE**;
 Матрица цвета – **GL_COLOR_MATRIX**;

glMatrixMode()

glMatrixMode(GL_MODELVIEW)

glLoadIdentity() - заменяет текущую матрицу, выбранного типа, единичной;

glPushMatrix() - заносит в стек копию текущей матрицы;

glPopMatrix() - извлекает матрицу из стека.

glMultMatrix() – перемножает текущую матрицу с матрицей аргументом.

Проекция задается всегда, иначе изображение будет вывернуто.

Неявно изменяют состояние конвейера:

glScale[f d] (sx, sy, sz).

glRotate[f d](angle, x, y, z)

glTranslate[f d] (x, y, z)

Наша схема: $p' = ((p * M1) * M2) * M3$

Схема OpenGL: $p' = M3 * (M2 * (M1 * p))$.

Scale – создает матрицу масштабирования по заданным аргументам и перемножает ее с текущей (текущей должна быть ModelView).

Rotate – аналогично матрица поворота, относительно вектора, вектор считается привязанным к OXYZ.

Векторные примитивы – точки, линии, многоугольники. Описываются вершинами. Смысл этих понятий отличается от математического. Прямая – по сути отрезок. Многоугольник всегда выпуклый

GL_LINE_STRIP – полилиния, каждая вершина задает отрезок вместе с предыдущей

Векторные примитивы

```
glVertex[2 3 4][s i f d](координаты)
glVertex[2 3 4][s i f d][v](координаты)
```

```
glBegin(mode)
  Команды задания вершин и их атрибутов,
  например цвета
glEnd;
```

GL_POINTS – каждая вершина задает точку

GL_LINE_STRIP - полилиния.

GL_LINE_LOOP - замкнутая полилиния.

GL_LINE_STRIP - последовательность отдельных отрезков

Каждая вершина может иметь свой цвет. OpenGL может в интерполяцию цвета.

Треугольники – самый частый примитив.

GL_TRIANGLES - независимые треугольники.

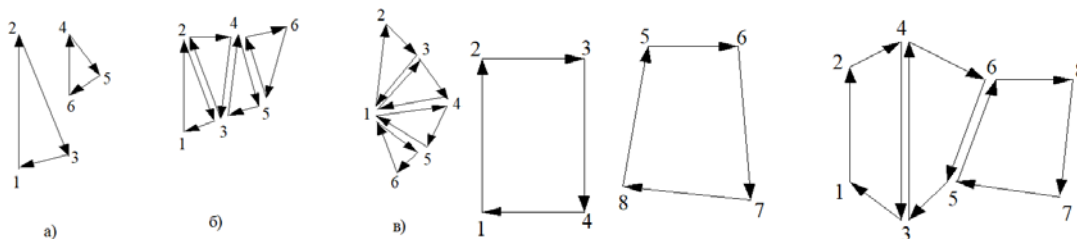
GL_TRIANGLE_STRIP - связанные треугольники.

GL_TRIANGLE_FAN - веер треугольников.

GL_QUADS - независимые четырехугольники.

GL_QUAD_STRIP - связанные четырехугольники.

GL_POLYGON - многоугольник.



При задании примитивов важен порядок обхода вершин

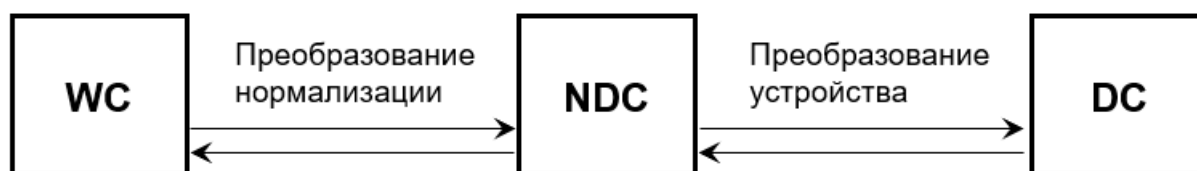
Степень аппроксимации задается явно.

Для примитива можно задать цвет (если нет света), либо свойства поверхности (при освещении), размер точек / толщину линии, тип линии, способ представления/заполнения многоугольника, шаблоны заполнения. Для вершин можно задавать нормали и привязывать к ним текстурные координаты.

`GLClear(GL_COLOR_BUFFER_BIT/GL_DEPTH_BUFFER_BIT)`

При использовании двойной буферизации в конце программы вызывается `SwapBuffers()`

Координатные системы двумерного видового конвейера



WC – мировые координаты. Система пользователя, исп-ся прикладным программистом, зависит от реальных размеров.

NDC – нормализованная координатная система. Независимая от устройства декартова система координат, приведенная к фиксированному диапазону (обычно приводят к диапазону $[0;1]$). Начало координат – центр симметрии.

DC – координаты устройства. Координатная система, определяющая конкретные устройства вывода информации.

В двумерном конвейере есть 2 преобразования:

- 1) Преобразование нормализации
- 2) Преобразование устройства

Их схемы структурно одинаковы и преобразование происходит из одного прямоугольника в другой. Прямоугольник, из которого происходит преобразование, называется **окном**, а прямоугольник в целевой координатной системе называется **полем вывода**. Окно и поле вывода должны быть подобными, чтобы не было искажения(квадрат)

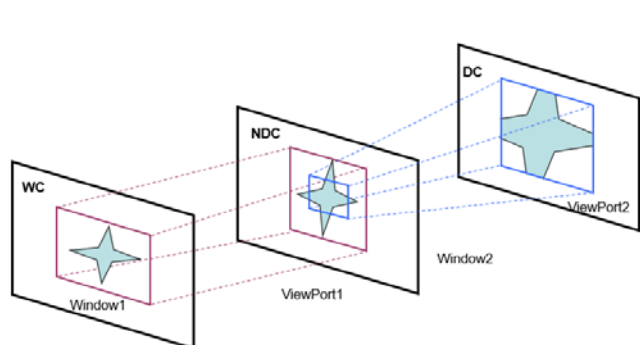


Рисунок 3.2 Преобразования геометрического конвейера 2D.

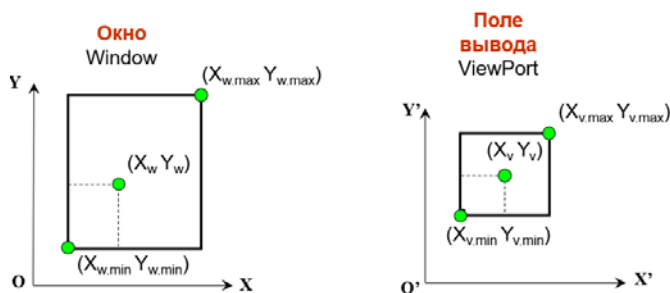


Рисунок 3.3 Схема преобразования.

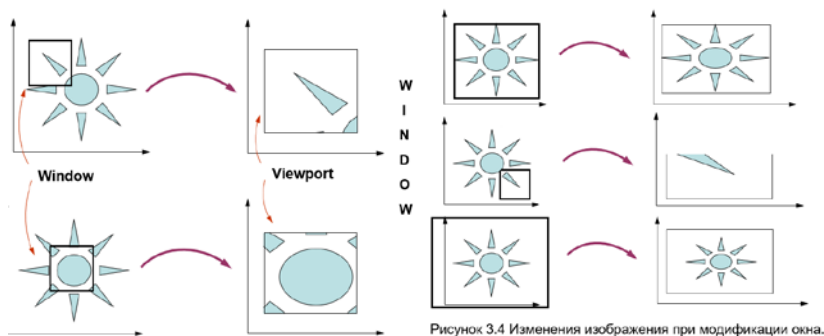
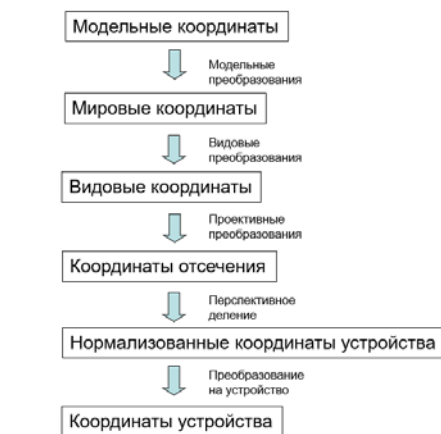


Рисунок 3.4 Изменения изображения при модификации окна.



Рисунок 3.5 Последовательность операций конвейера



Координатные системы трехмерного видового конвейера

Та часть пространства, попадая в которую фрагменты модели подвергаются преобразованию, называется видимым объемом.

Во многих графических системах используется двойная буферизация. Буфер кадра – прямоугольная область памяти, сопоставленная с растром экрана, хранит цвета пикселей на экране. Передний буфер показывает пользователю, задний готовится и заменяет передний.

На вход конвейера подается инфа 2 типов: инфа о вершинах и пиксельные данные. Вычислители – методы, которые генерируют координаты вершин, нормали поверхности, координаты текстур, опираясь на контрольные точки. Операции с пикселями выполняются в отдельном маршруте. Данные распаковываются с учетом возможных форматов, потом масштабируются, сдвигаются, передаются на операции растреризации.

Сборка текстуры – связывание текстурной карты с объектом.

Растреризация – процесс преобразования геометрических данных в фрагменты

Фрагмент – прообраз пикселя в буфере кадров.

После сборки фрагментов следует серия тестов, от результатов которых зависит будет ли заменена инфа в буфере кадров или нет.

8. Свойства материала в OpenGL Грани.

Грань – сторона многоугольника. 2 грани — лицевая и обратная. В данный момент может быть видна только 1 грань. Грани различают после того, как многоугольник был спроецирован на экран.

glFrontFace — если GL_CW, то лицевой стороной будут многоугольники с обходом по часовой стрелке; если GL_CCW — против часовой. (по умолчанию)

GL_CULL_FACE — включение/отключение рассмотрения обратных полигонов

Свойства материалов задаются командой glMaterial. Она принимает три аргумента:

1. Тип грани, для которой задается свойство
 - a. GL_FRONT
 - b. GL_BACK

c. GL_FRONT_AND_BACK

2. Свойство, которое будет задаваться:

a. GL_AMBIENT – коэффициент отражения рассеянного света

b. GL_DIFFUSE – коэффициент отражения диффузного света

c. GL_SPECULAR – коэффициент отражения зеркального света

d. GL_EMISSION – излучение света материалом

e. GL_SHININESS – определяет блики от зеркального света. Может принимать значение от 0 до 128, которое подставляется на место n в формулу $I_s = I_l * w(\theta, \lambda) * \cos^n \alpha$ (формула интенсивности отраженного света. $\cos^n \alpha$ определяет блики). Используется для задания пространственного распределения зеркально отраженного света. Если зеркало, то не рассеивается.

f. GL_AMBIENT_AND_DIFFUSE – задает одинаковые коэффициенты отражения для рассеянного и диффузного света

g. GL_COLOR_INDEXES – задает индексы цветов для рассеянного, диффузного и зеркального отражения. Хз, что это значит

9. Реализация проекций в OpenGL

Ортографические

План, Вид спереди и сбоку – поворот относительно соответствующей полуоси

изометрия

диметрия

триметрия

$$\varphi = \pm 45$$

вокруг Y

$$\theta = \pm 35,264$$

вокруг X

```
glRotate(-45, 0, 1, 0)
glRotate(-45, 1, 0, 0)
```

```
glRotate(35, 0, 1, 0)
glRotate(60, 1, 0, 0)
glRotate(20, 0, 0, 1)
```

Реализация в OpenGL:

- glOrtho(left, right, bottom, top, near, far) — создает матрицу параллельного проецирования и перемножает с текущей матрицей и определяет видимый объем(параллелепипед)

Ортографические проекции

$$P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Near и Far задаются в видовой системе координат => near < far.

Формируемая матрица приводит видимый объем к каноническому.

Косоугольные

Для создания используется матрица скоса, а затем проекция на Z=0.

Кабине:

```
glLoadIdentity()
```

```
matr = numpy.array(glGetFloatv(GL_MODELVIEW_MATRIX))
```

```
matr[2][0] = -m.cos(m.radians(45)) # f = 1
```

```
matr[2][1] = -m.sin(m.radians(45)) # f = 1
```

```
glLoadMatrixf(matr)
```

Кавалье:

```
glLoadIdentity()
```

```
matr = numpy.array(glGetFloatv(GL_MODELVIEW_MATRIX))
```

```
matr[2][0] = -m.cos(m.radians(45)) / 2 # f = 1/2
```

```
matr[2][1] = -m.sin(m.radians(45)) / 2 # f = 1/2
```

```
glLoadMatrixf(matr)
```

Кабине

```
current_model_view = glGetFloatv(GL_MODELVIEW_MATRIX)
current_model_view[2][0] = -math.cos(math.radians(45)) * (math.pi / 2 - math.atan(1 / 2))
current_model_view[2][1] = -math.sin(math.radians(45)) * (math.pi / 2 - math.atan(1 / 2))
glLoadMatrixf(current_model_view)
```

Кавалье

```
current_model_view = glGetFloatv(GL_MODELVIEW_MATRIX)
current_model_view[2][0] = -math.cos(math.radians(45)) * math.atan2(1, 2)
current_model_view[2][1] = -math.sin(math.radians(45)) * math.atan2(1, 2)
glLoadMatrixf(current_model_view)
```

Центральные(перспективные)

Реализация в OpenGL:

- `glFrustum(left, right, bottom, top, near, far)` — создает матрицу односточечного центрального проецирования и перемножает с текущей и определяет видимый объем.

Центр проекции (точка наблюдения) находится в начале видовой системы координат, поэтому `near` и `far` обязательно > 0 , $near < far$

`gluPerspective(fovy, aspect, near, far)` аналогично.

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity()
glFrustum(-1.0, 1.0, -1.0, 1.0, 1, 5)
glMatrixMode(GL_MODELVIEW)
glLoadIdentity()

# односточечная
glPushMatrix()
glViewport(0, 0, int(view_width / 4), int(view_height / 3))
glTranslatef(0, 0, -2.5)
glTranslatef(-1, -1, 0)
draw_figure()
glPopMatrix()

# двусточечная
glPushMatrix()
glViewport(int(view_width / 4), 0, int(view_width / 4), int(view_height / 3))
glTranslatef(0, -1, -2.5)
glRotatef(30, 1, 0, 0)
draw_figure()
glPopMatrix()

# трехточечная
glPushMatrix()
glViewport(int(view_width / 4) * 2, 0, int(view_width / 4), int(view_height / 3))
glTranslatef(-1, -1, -2.5)
glRotatef(-15, 1, 1, 1)
draw_figure()
glPopMatrix()
```

10. Позиционные источники света в OpenGL

Позиционный источник света является точечным. Данный источник располагается непосредственно на сцене, на 2 разные точки на сцене свет будет падать под разными углами и с разным рассеиванием.

Осн свойства – `x`-ки светового потока и направление откуда светит.

`GL_POSITION` — для позиционного источника определяет положение источника на сцене

11. Направленные источники света в OpenGL

Направленный источник света отсутствует затухание, бесконечно удален от сцены. Расчет освещения прост, тк угол падения одинаков для всех точек плоскости(как солнце).

Осн свойства – позиция, направление, характеристики светового потока, затухание, распределение света в конусе.

GL_POSITION — для направленного источника задает направление освещения, определяя вектор, соединяющий точку x, y, z с началом координат.

12. Модель освещения в OpenGL

Поток света в OpenGL можно расщепить на:

- свет, который может отразиться от поверхности диффузно;
- свет, который может отразиться от поверхности зеркально;
- рассеянный свет.

В свою очередь, каждый из этих потоков расщепляется на красный, синий и зеленый. В основе расчетов лежит простая модель освещения. (векторная форма)

$$I = I_a K_a + I_l (K_d (\underline{n} * \underline{L}) + K_s (\underline{R} * \underline{S})^n) / (d + K)$$

I_a — интенсивность рассеянного света, K_a — коэффициент рассеивания.

I_l — интенсивность диффузного света, K_d — коэффициент диффузного отражения.

$K_s = w(\text{тета}, \text{лямбда})$ — кривая отражающая отношение

n — нормаль, R — направление отражённого света, S — вектор наблюдения

L — вектор направления на источник, $\cos \text{тета} = \underline{n} * \underline{L}$, $\cos \text{альфа} = \underline{R} * \underline{S}$

Процессом освещения можно управлять через свойства источников, через свойства материалов, через модель и отдельными командами, включающими и выключающими расчет освещения, источники и режим интерполяции

Свойства источника: позиция, направление света, затухание, световой конус, положение в пространстве

Чтобы модель освещения работала:

1 — Необходимо определить вектора нормали для всех вершин (или включить автонормализацию нормалей)

2 — Включить расчет освещения и хотя бы 1 источник света. Задать свойства источника, если необходимые свойства по умолчанию выключены.

3 — Определить свойства модели освещения, при необходимости. В том числе общее фоновое освещение и положение наблюдателя.

4 — Задать свойства материалов

Нормали и вертикали задаются семейством команд `glNormal()`. Для правильного расчета освещения нормали должны быть единичной длины. Можно включить режим нормализации `glEnable(GL_NORMALIZE)`+

Управление методом интерполяции ос-ся с помощью команды `glShadeModel(GL_FLAT)` – плоскость закрашивания/`GL_SMOOTH` – интерполяция)

Источник света — `GL_LIGHTi = GL_LIGHT0 + i`

Задание свойств — `glLight[i f]v(источник, свойство, параметры)`

`GL_SPOT_EXPONENT` — распределение интенсивности света внутри светового конуса — от 0 до 128 (по умолчанию 180). скалярная версия команды

`GL_SPOT_CUTOFF` — световой конус — от 0 до 90 и 180 (по умолчанию 180) угол задается между направлением освещения и границей светового конуса. скалярная версия команды

`GL_CONSTANT_ATTENUATION`, `GL_LINEAR`, `GL_QUADRATIC` — затухание — (по умолчанию 1, 0, 0) (для позиционного). скалярная версия команды

Ниже векторная версия команды

`GL_AMBIENT` — рассеиваемый свет — (по умолчанию [0, 0, 0, 1])

GL_DIFFUSE — диффузный свет — [0, 0, 0, 1] для всех и [1, 1, 1, 1] для нулевого

GL_SPECULAR — зеркальный свет — [0, 0, 0, 1] для всех и [1, 1, 1, 1] для нулевого

GL_POSITION — тип источника, положение или направление — (x, y, z, w). Если w = 0, направленный источник, задает направление освещения, определяя вектор, соединяющий точку x, y, z с началом координат. Если w = 1, позиционный источник, определяет положение источника на сцене. По умолчанию [0, 0, 1, 0]

GL_SPOT_DIRECTION — направление освещения для позиционного источника — по умолчанию [0, 0, -1] — подвергается взаимодействию с матрицей GL_MODEL_VIEW

Включение света осуществляется через константу GL_LIGHTING.

Свойства на уровне модели освещения — glLightModel[i f]v(свойство, параметр). Свойства:

GL_LIGHT_MODEL_LOCAL_VIEWER — GL_FALSE — наблюдатель удален бесконечно, иначе в начале системы координат. По умолчанию GL_FALSE

GL_LIGHT_MODEL_TWO_SIDE — GL_FALSE — расчет только для лицевой грани, иначе для обеих. По умолчанию GL_FALSE

GL_LIGHT_MODEL_COLOR_CONTROL — GL_SEPARATE_SPECULAR_COLOR или GL_SINGLE_COLOR. Считываем всю цветовую модель или отдельно зеркальное освещение. По умолчанию GL_SINGLE_COLOR

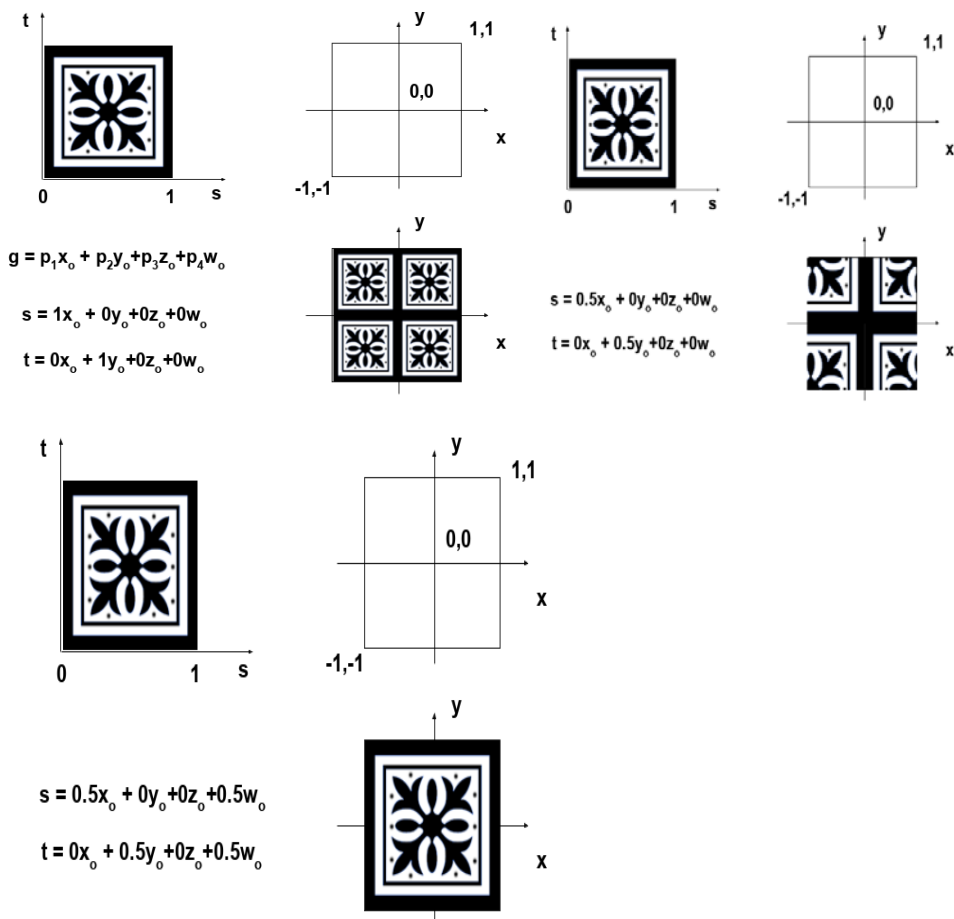
GL_LIGHT_MODEL_AMBIENT — фоновая освещенность. По умолчанию [0.2, 0.2, 0.2, 1.0]

13. Текстуры

Текстуры – прямоугольные массивы данных о цвете. Эти массивы называются картами текстур, а их элементы – текстелями.

Для отображения текстуры нужно отразить текстели в пиксели.

Привязать текстурные координаты к координатам объекта можно с помощью параметрической функции. Координаты текстуры имеют значение от 0 до 1.



Если текстурные координаты выходят за пределы от 0 до 1, то можно сделать 2 вещи:

1. Зафиксировать значения крайних текселей текстуры (Clamp)



2. Отбросить целую часть координаты и взять дробную. Тогда текстура будет повторяться (Repeat)



Для текстурирования нужно:

1. Подготовить данные для текстуры
2. Создать текстурный объект
3. Загрузить данные текстуры в этот объект (это трудоемкая задача по вычислениям)
4. Установить фильтры для увеличения/уменьшения текстуры
5. Определить поведение текстуры при выходе за пределы координат(clamp, repeat)
6. Задать функцию наложения текстуры (определить, как смешивается информация о свете в пикселе и текстеле)
7. Привязать координаты текстуры к координатам объекта. Если используется автоматическое генерирование текстурных координат, то задать метод генерирования
8. Включить использование текстур

Есть 2 подхода к текстурированию:

1. Использовать 1 текстурный объект и загружать в него разные данные по необходимости (просто, но неэффективно)
2. Заранее создать нужное кол-во текстурных объектов и при формировании сцены делать нужный объект текущим.(эффективнее)

Включить\выключить текстурирование можно командой glEnable\glDisable с параметром GL_TEXTURE_2D.

glTexCoor позволяет связывать координаты текстуры с координатами объекта. Изменяет текущие значения текстурных координат. Все вершины после этой команды получают значения по переданным координатам.

Свойства текстуры задаются командой glTexImage2D. Принимает следующие параметры:

1. Target – должен быть GL_TEXTURE_2D/GL_PROXY_2D – создается текстура/делается попытка создания, определяется возможность создания текстуры с данными настройками
2. Level – устанавливает уровень детализации текстуры. Используется для мультитекстурирования. Чтобы заработало нужно загрузить все уровни текстуры, все размеры по степеням двойки от текущего вниз(?). Чем ниже уровень, тем выше детализация (больше размер текстуры).
3. InternalFormat – определяет какую информацию может хранить текстель. Принимает значения GL_RGBA – полная информация о цвете, GL_ALPHA – только альфа-канал, GL_R3_G3_B2 – по 3 бита на цвет кроме синего.
4. Width – ширина, только степень двойки
5. Height – высота, только степень двойки
6. Border – ширина обводки. Может быть 0 или 1,

7. Format – определяет формат данных пикселя (обычно GL_RGBA)

8. Type – тип данных информации о пикселе (GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT)

9. Pixels – указатель на массив данных (сама текстура)

Текстура должна иметь размер только степени двойки.

Фильтры используются при масштабировании текстуры. Их можно задать командой glTexParameter. Аргументы:

1. Target – обычно GL_TEXTURE_2D, тип используемой текстуры

2. Pname – какой фильтр настраиваем – для уменьшения или для увеличения (GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER)

3. Value – как фильтровать

Value может быть:

1. GL_NEAREST – берется значение текселя, центр которого ближе всего к текстурной координате.

2. GL_LINEAR - определяется тексель, центр которого ближе всего к текстурной координате и вычисляется среднее значение четырех текселей вокруг него

3. Комбинации предыдущих двух. В таком случае используются две текстурные карты, метод на первом месте используется для отбора значений текселей на каждой текстуре отдельно; метод на втором месте используется для выбора текстуры, которую надо использовать: если nearest – то берется одна ближайшая по размеру текстура, если linear, то берется 2 ближайшие по размеру, после чего по первому методу вычисляются 2 значения и находится их среднее арифметическое.

Всего получается 6 фильтров. Пример комбинаций:

GL_NEAREST_MIPMAP_NEAREST: Выбирает ближайший мипмап, соотносящийся с размером пикселя и также используется интерполяция ближайшего соседа для сэмпинга текстур.

Фильтрация происходит аналогично nearest, но происходит не на 0 уровне, а на уровне, выбранном системой исходя из размеров объекта(ближайший по размеру)

GL_LINEAR_MIPMAP_NEAREST: Выбирает ближайший мипмап и сэмплирует его методом линейной интерполяции.

GL_NEAREST_MIPMAP_LINEAR: Линейная интерполяция между двумя ближайшими мипмапами и сэмплирование текстур с помощью линейной интерполяции.

Комбинации надо использовать только для уменьшающих фильтров.

Мультитекстурирование – создание множества текстурных карт по размерам степеней двойки. Для объекта выбирается ближайшая