КГ упрощенная версия

silvia.lesnaia

19 ноября 2025 г.

1 Интерактивное приложение. Интерактивная машинная графика

Интерактивное приложение — приложение с которым пользователь имеет возможность динамически взаимодействовать. Интерактивная графика — графика с которой пользователь имеет возможность динамически взаимодействовать, манипулировать содержимым изображения, его формой, размером и цветом на поверхности дисплея с помощью интерактивных устройств управления. Такими устройствами могут быть любые устройства ввода: клавиатура, мышь или события операционной системы.

2 Событийно-ориентированное программирование. Основные варианты событий и их обработчиков при построении интерактивного графического приложения.

Событийно-ориентированное программирование — это парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета). Событию можно назначить обработчик события, то есть функцию, которая сработает, как только событие произошло. Парадигма наличия обработчиков события не обязательно связана с ООП.

3 Декартова система координат (ДСК). Модельная (локальная, объектная) система координат. Система координат экрана.

Декартова система координат

система двух или трёх перпендикулярных друг другу осей, пересекающихся в одной точке — начале отсчёта (начале координат) с общей единицей длины. Также ДСК называется прямоугольной СК.

Модельная (локальная, объектная) система координат

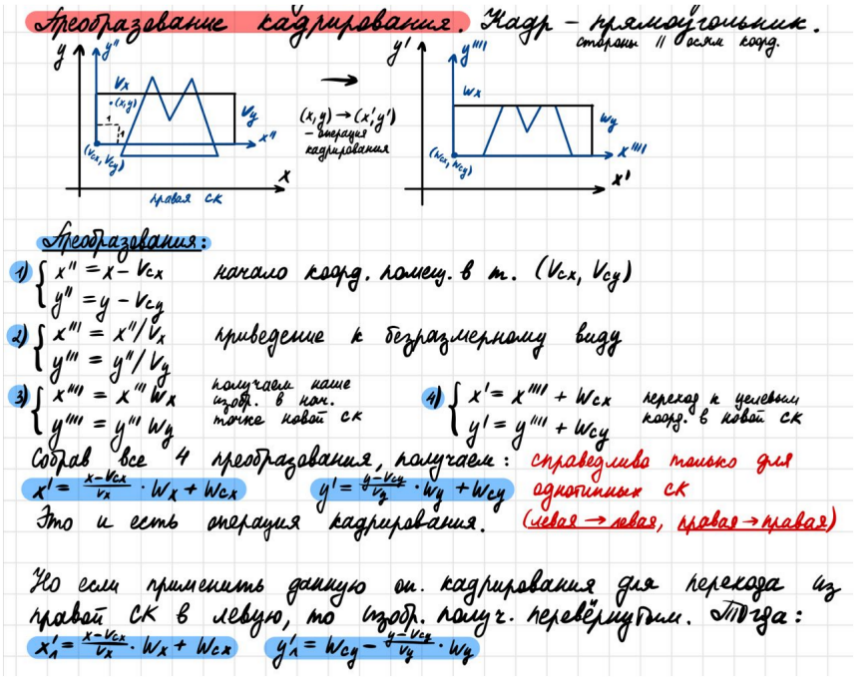
координатная система, которая является локальной для объекта, т.е. начинается в той же точке, что и сам объект. Все вершины модели находятся в локальном пространстве: все их координаты являются локальными по отношению к объекту. Далее модельная система координат может совершать со своим объектом все движения в мировой системе координат.

Система координат экрана

система координат экрана монитора. В отличие от перечисленных выше систем координат, экранная обычно имеет дискретные целочисленные координаты. Может быть левой (начало находится слева сверху) или правой (начало находится слева снизу).

4 Вывод преобразования кадрирования без сохранения пропорций изображения.

Кадр — прямоугольник со сторонами, параллельными осям координат (если это условие не выполняется, необходимо перейти к другой системе координат).



5 Сохранение пропорций изображения при преобразовании кадрирования.

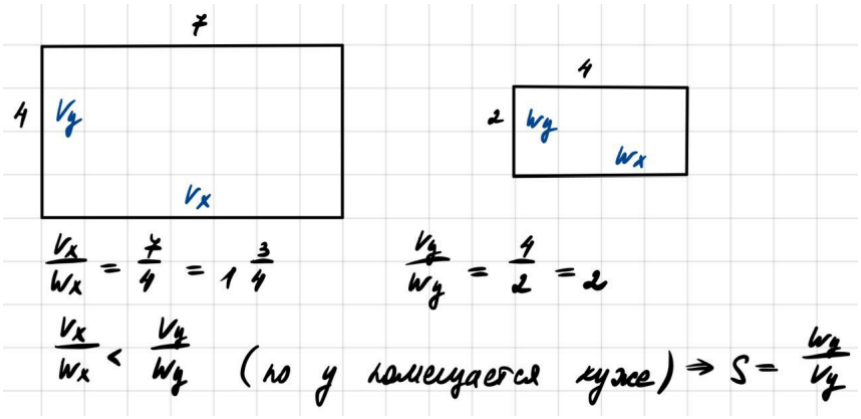
Чтобы сохранить пропорции изображения при операции кадрирования, нужно домножать обе размерности на один и тот же коэффициент.

Чтобы не выходить за кадр нужно:

1. Соотношение сторон рисунка и окна рисования $\frac{V_x}{W_x}$. Больший коэффицент покажет, по какой размерности выйдет наше изображение за рамки кадра/экрана. Нужно вычислить коэффицент по которой изображение не выйдет за рамки кадра/экрана.

2. $\frac{V_x}{W_x} > \frac{V_y}{W_y} \Rightarrow \frac{W_x}{V_x}$

$\frac{V_x}{W_x} < \frac{V_y}{W_y} \Rightarrow \frac{W_y}{V_y}$



6 Однородные координаты. Переход от евклидовых к однородным координатам и обратно.

Однородные координаты — это координаты, обладающие тем свойством, что определяемый ими объект не меняется, когда все координаты умножаются на одно и то же число.

Однородными координатами точки $P \in R^n$ называется вектор столбец

$$\begin{bmatrix} \chi_1 & \chi_2 & \cdots & \chi_n & \chi_{n+1} \end{bmatrix}^T \in R^{n+1},$$

среди элементов которого хотя бы один элемент χ_i должен быть отличен от нуля.

Если $\chi_{n+1} = 0$, то точка P является бесконечно удалённой.

Евклидовыми координатами точки P будут являться координаты

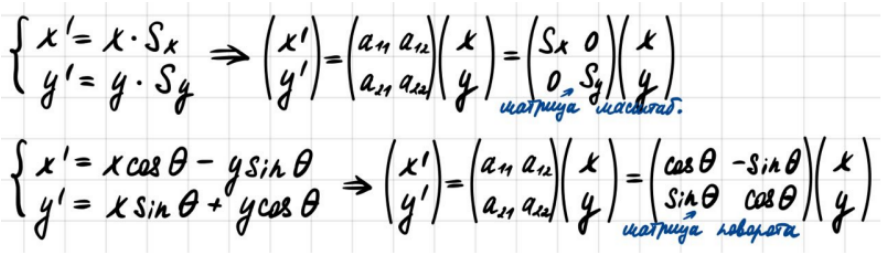
$$\left(\frac{\chi_1}{\chi_{n+1}}, \frac{\chi_2}{\chi_{n+1}}, .., \frac{\chi_n}{\chi_{n+1}} \right).$$

Преобразование из однородных координат в евклидовы однозначно; преобразование из евклидовых координат в однородные — нет.

Однородные координаты получаются из евклидовых следующим образом: последняя координата χ_{n+1} выбирается произвольно ($\chi_{n+1} \neq 0$), а остальные координаты получаются путём умножения соответствующих евклидовых координат на χ_{n+1}.

Пример: Из евклидовых в однородные
$P(2, 3, 2) \Rightarrow (2 * 2, 2 * 3, 2 * 2, 2)$
Из однородных в евклидовые
$(4, 6, 4, 2) \Rightarrow (4/2, 6/2, 4/2) = (2, 3, 2)$

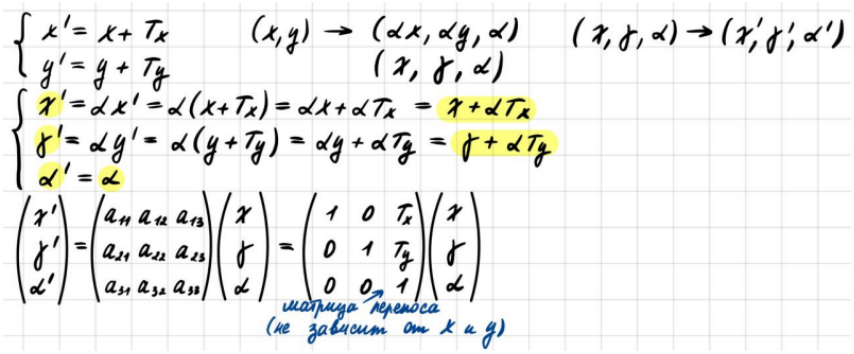
7 2D-преобразования — масштабирование и поворот относительно начала координат. Получение матриц преобразований. Обратные преобразования.



Обратный: $\frac{1}{S_x}\frac{1}{S_y}\frac{1}{S_z}$

$-\theta$

8 2D-преобразование — перенос. Получение матрицы преобразования. Обратное преобразование.



Обратный: -Tx,Ty

9 Совмещение 2D-преобразований. Получение матрицы совмещенного преобразования. Обратное преобразование для совмещенного преобразования.

Когда последовательность элементарных преобразований применяется к изображению, возможны два подхода к выполнению такой задачи: либо рассматривается последовательность изображений, где каждое последующее изображение получено с помощью очередного элементарного преобразования над предыдущим, либо получение результата из исходного изображения происходит в результате применения единого сложного преобразования — комбинации элементарных преобразований. Такое сложное преобразование будем назвать «совмещенное преобразование»

$$\begin{bmatrix} \cos\vartheta & -\sin\vartheta & x_A(1-\cos\vartheta)+y_A\sin\vartheta \\ \sin\vartheta & \cos\vartheta & y_A(1-\cos\vartheta)-x_A\sin\vartheta \\ 0 & 0 & 1 \end{bmatrix}$$

10 Алгоритм Коэна-Сазерленда отсечения невидимых линий.

Этот алгоритм позволяет быстро выявить отрезки, которые могут быть или приняты, или отброшены целиком. Вычисление пересечений требуется, когда отрезок не попадает ни в один из этих классов.

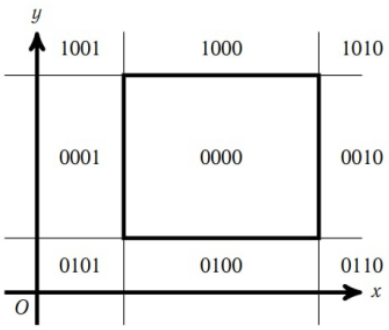
Идея алгоритма состоит в следующем: прямые линии, ограничивающие область видимости, делят всё двумерное пространство на 9 областей. Каждой из областей присваивается 4-х-разрядный двоичный код:

единица в 1-ом разряде — область слева от левой границы области видимости;

единица во 2-ом разряде — область справа от правой границы области видимости;

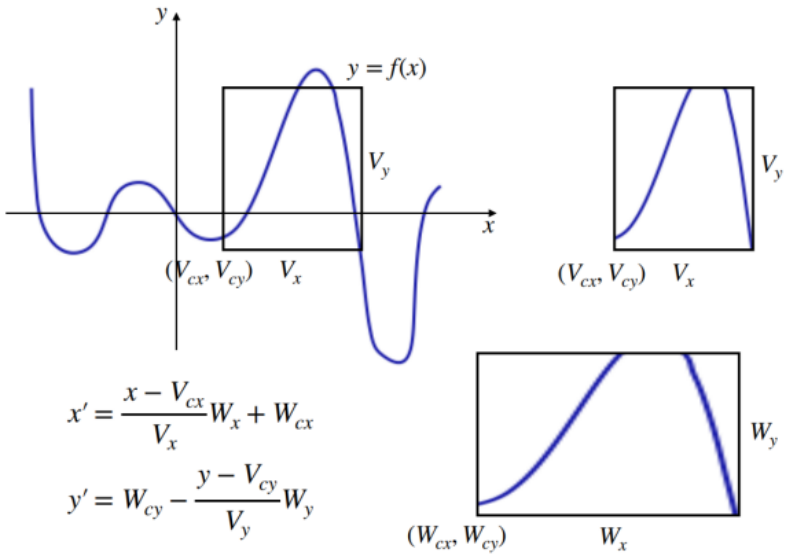
единица в 3-ем разряде — область под нижней границей области видимости;

единица в 4-ом разряде — область над верхней границей области видимости.


y
1001
1000
1010
0001
0000
0010
0101
0100
0110
x
O

11 Алгоритм построения простого графика функции: двумерного, трехмерного

Пусть ставится задача переноса части графика функции, попавшего в окно кадра, в окно кадра на экране. Нам уже известно об операции кадрирование при условии, что переход осуществляется из правой в левую систему координат.



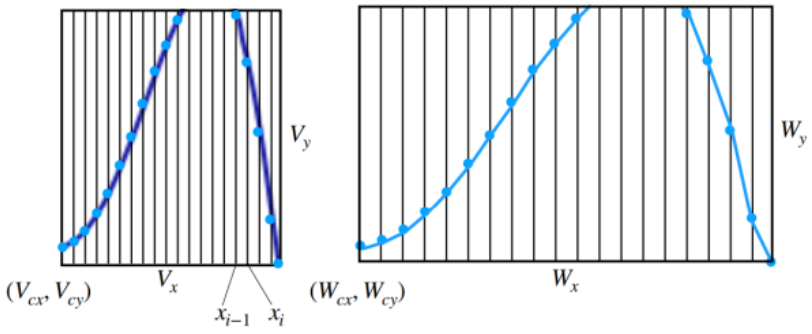
$$x' = \frac{x - V_{cx}}{V_x} W_x + W_{cx}$$

$$y' = W_{cy} - \frac{y - V_{cy}}{V_y} W_y$$

Рис. 13: Пример переноса части графика функции из кадра на графике в кадр на экране

Идея отрисовки графика состоит в том, что мы проводим дискретизацию оси Ox. Но делать это нужно для каждого пикселя окна, чтобы избежать явных прямых линий на графике.

Таким образом, каждое следующее значение x'_i вычисляется как $x'_i = x'_{i-1} + 1$. Но чтобы строить график, нам нужно соответствующее значение x_i из системы координат графика. Сделать это несложно, ведь известно, что $x_1 = V_x$, а шагом в этом пространстве, соответствующему одному пикселю в окне, будет V_x/W_x. То есть $x_i = x_{i-1} + \frac{V_x}{W_x}$.



$$x' = \frac{x - V_{cx}}{V_x} W_x + W_{cx}$$

$$y' = W_{cy} - \frac{y - V_{cy}}{V_y} W_y$$

Рис. 14: Дискретизация оси Ox

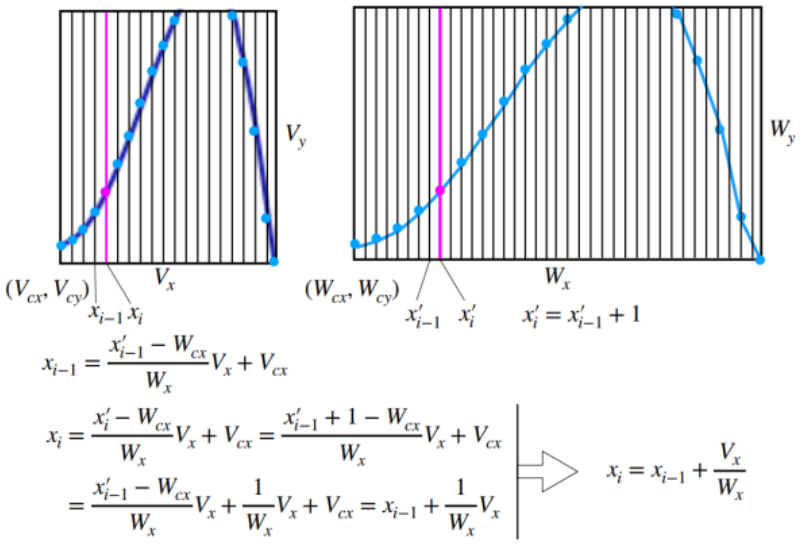


Рис. 15: Пробег по растру в пространстве окна экрана

Уже от x_i подсчитывается значение функции $y_i = f(x_i)$, и это значение переводится назад в оконный кадр $y \to y'$.

Случаи функций с точками разрыва можно описать следующим образом: в таких случаях обязательно нужно проверять, существует ли значение функции в данной точке или нет. Если значение существует, то точка отображается. Однако, есть случаи, когда точка разрыва в пространстве графика приходится на промежуток между колонками растра в окне графика, тогда обе граничные точки до и после точки разрыва существуют, и происходит картина, подобная изображённой на рисунке 16.

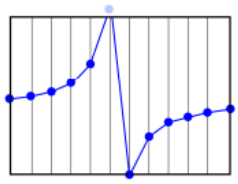


Рис. 16

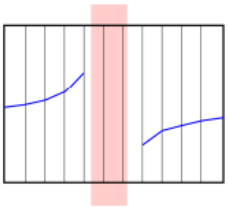


Рис. 17

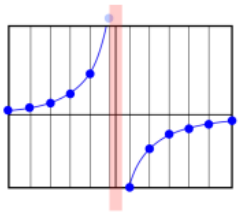
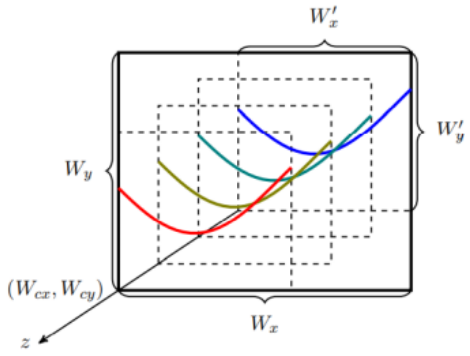


Рис. 18

Чтобы этого избежать, достаточно выбрать некоторое значение ε отступа от точки разрыва, и тогда точки, попавшие в промежуток $(x - \varepsilon, x + \varepsilon)$, где x — точка разрыва, не будут отрисовываться. Стоит заметить, что значение отступа нужно выбирать оптимально-минимальным, чтобы отрисовывалось максимальное количество точек графика, и при этом не захватывалась точка разрыва.

Идея отрисовки трехмерного графика достаточно проста. Как и раньше, в мировой системе координат имеем ограниченную область видимости — параллелепипед с ребрами, параллельными осям координат. Кроме зависимости y от x, добавилась зависимость от z. Если мы зафиксируем конкретное значение $z = z^*$, то получим двумерный график $y = f(x, z^*)$. Если будем фиксировать значения z с некоторым постоянным шагом, то для каждого из фиксированных значений сможем построить двумерный график.

Чтобы придать такому набору двумерных графиков вид единого трехмерного, изобразим их со смещением относительно друг друга по вертикали и горизонтали. То есть, для каждого фиксированного z^* будем рисовать двумерный график функции $y = f(x, z^*)$ в своем рабочем прямоугольнике с шириной W'_x, высотой W'_y и координатами левого нижнего угла $(W_{cx}(z^*), W_{cy}(z^*))$.



Будем считать, что координата z является индикатором удаленности точки от наблюдателя: чем меньше координата z, тем дальше точка. Тогда отрисовывая двумерные графики функции $y = f(x, z^*)$ в цикле по z^* от наименьшего значения (V_{cz}) до наибольшего $(V_{cz} + V_z)$ мы получаем подобие трехмерного образа, в котором близлежащие фрагменты графика могут затенять собой более удаленные.

$$W_{cx}(z_0) = W_{cx} + W_x - W'_x \qquad W_{cy}(z_0) = W_{cy} - W_y + W'_y$$

$$W_{cy}(z_i) = W_{cy}(z_{i-1}) + 1$$

$$W_{cx}(z_i) = W_{cx}(z_{i-1}) - \Delta x$$

$$\Delta x = \frac{W_x - W'_x}{W'_z}$$

12 Мировая система координат. Модельное преобразование. Вывод матрицы варианта модельного преобразования.

Мировая система координат

Система координат всего изображения, в которой привязываются его составные элементы. азис МСК с такой ориентацией осей является право-

сторонним. Как мы уже оговаривали ранее, расстоянию в один пиксел по горизонтали в системе координат экрана соответствует расстояние в мировой системе координат равное V_x/W_x . Координаты в мировом пространстве — это как раз то, о чем говорит их название: координаты всех ваших вершин относительно (игрового) мира. Это координатное пространство, в котором вы бы хотели видеть ваши объекты преобразованными таким образом, что бы они были распределены в пространстве (и желательно реалистично). Координаты вашего объекта преобразуются из локального в мировое пространство — модельное преобразование; это выполняется посредством матрицы модели (матрицы модельного преобразования).

Модельное преобразование

матрица, которая перемещает, масштабирует и/или вращает ваш объект для его расположения в мировом пространстве в позиции/ориентации в которой объект должен находиться. Стоит учитывать, что модельная матрица для рисунка должна представлять собой преобразование в котором сначала производится начальная установка рисунка в начало координат, а потом проводится привязка, описанная командами translate, scale и rotate, то есть она получается в результате произведения матриц M и initM.

13 3D-преобразования — масштабирование и вращение относительно осей координат. Получение матриц преобразований. Обратные преобразования

Формула для трехмерного масштабирования:

$$\begin{cases} x' = S_x x \\ y' = S_y y \\ z' = S_z z \end{cases}$$

Матричная форма:

$$Scale(S_x, S_y, S_z) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Обратный: $\frac{1}{S_x}\frac{1}{S_y}\frac{1}{S_z}$

$$Rotate_z(\vartheta) = \begin{pmatrix} cos\vartheta & -sin\vartheta & 0 & 0 \\ sin\vartheta & cos\vartheta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rotate_x(\vartheta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\vartheta & -sin\vartheta & 0 \\ 0 & sin\vartheta & cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Rotate_y(\vartheta) = \begin{pmatrix} cos\vartheta & 0 & sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\vartheta & 0 & cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

14 3D-преобразование — перенос. Получение матрицы преобразования. Обратное преобразование.

Формула трехмерного переноса:

$$\begin{cases} x' = x + T_x \\ y' = y + T_y \\ z' = z + T_z \end{cases}$$

Матричная форма:

$$Translate(T_x, T_y, T_z) = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Обратный: -Tx,-Ty,-Tz

15 Скалярное произведение. Геометрический смысл скалярного произведения.Векторное произведение. Вывод матрицы векторного произведения.

Скалярным произведением векторов $\overline{p}$ и $\overline{q}$ называется величина. $\overline{pq} = |\overline{p}||\overline{q}| \cos \widehat{\overline{pq}}$
Геометрический смысл скалярного произведения — произведение длины проекции первого вектора на второй и длины второго вектора.

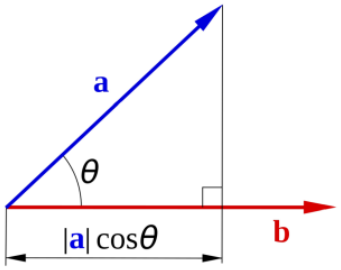


Рис. 19: Геометрический смысл скалярного произведения

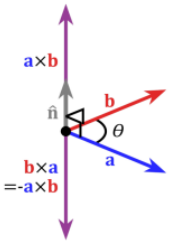


Рис. 20: Векторное произведение

Векторное произведение

$$\bar{p} \times \bar{q} = \begin{bmatrix} p_2 q_3 - p_3 q_2 \\ p_3 q_1 - p_1 q_3 \\ p_1 q_2 - p_2 q_1 \end{bmatrix}$$

может быть представлено в виде умножения матрицы на вектор

$$\bar{p} \times \bar{q} = [\bar{p}]_\times \bar{q},$$

16 Двойственность преобразований (каждого из преобразований). Переход от одного базиса ДСК к другому

Любое из приведенных выше преобразований можно рассматривать как преобразование изображения (например перенос точки из одной части изображения в другую). Но в то же время преобразование, примененное ко всем точкам изображения можно рассматривать как преобразование системы координат. Так, например, перенос точек в положительном направлении оси

Ox можно рассматривать как сдвиг системы координат в отрицательном направлении оси Ox, а поворот точек против часовой стрелки относительно начала координат можно рассматривать как поворот системы координат по часовой стрелке.

Векторное произведение

$$\bar{p} \times \bar{q} = \begin{bmatrix} p_2 q_3 - p_3 q_2 \\ p_3 q_1 - p_1 q_3 \\ p_1 q_2 - p_2 q_1 \end{bmatrix}$$

может быть представлено в виде умножения матрицы на вектор

$$\bar{p} \times \bar{q} = [\bar{p}]_\times \bar{q},$$

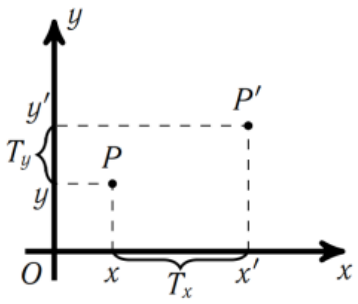


Рис. 21: Перенос точки

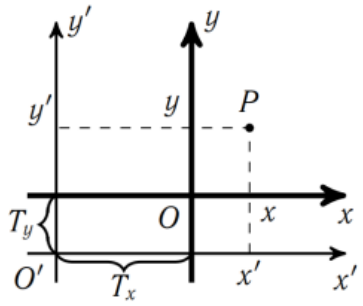


Рис. 22: Перенос начала координат

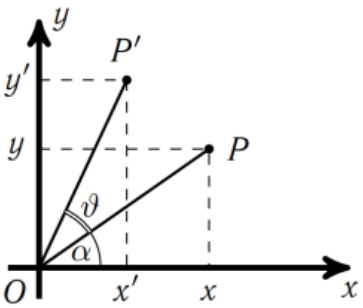


Рис. 23: Поворот точки относительно начала координат

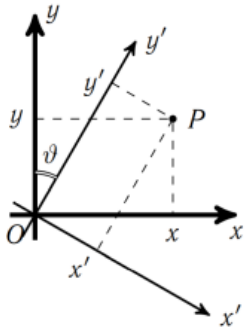


Рис. 24: Поворот системы координат

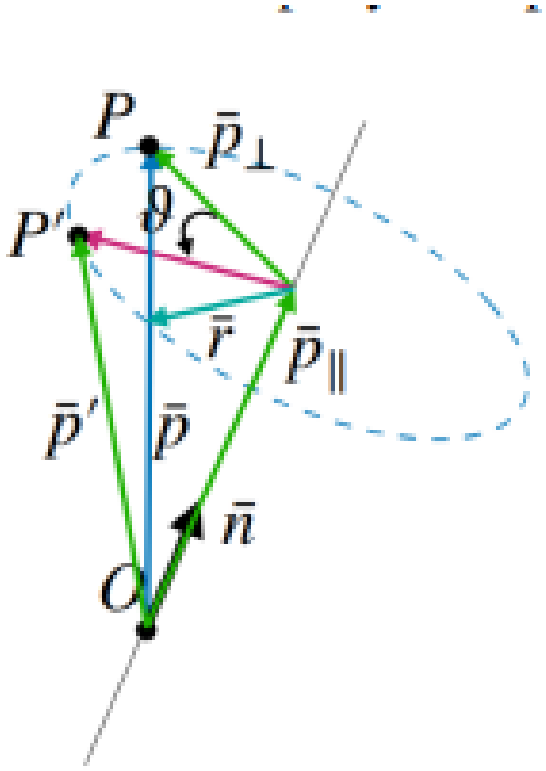
Так как координаты любой точки совпадают с координатами ее радиус-вектора, любое из перечисленных преобразований можно отнести к векторам.

Преобразования трёхмерных вращений можно представить в виде:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1'^T \\ \boldsymbol{e}_2'^T \\ \boldsymbol{e}_3'^T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$

17 Трехмерное вращение относительно оси, проходящей через начало координат. Вывод формулы Родригеса.

Вход: ось — единичный вектор $\overline{n}$; угол θ; точка P и соответствующий ей радиусвектор p. Выход: вектор $\overline{p}\,\overline{p}'$ — радиус-вектор точки P' , являющейся поворотом точки P вокруг оси $\overline{n}$ на θ градусов против часовой стрелки



Соотношние $M = E + [\overline{n}]_\times \sin\vartheta + [\overline{n}]_\times^2(1-\cos\vartheta)$ называется **формулой Родригеса** вращения вокруг произвольной оси $\overline{n}$.

18 Матрица преобразования нормалей.

В компьютерной графике невозможно обойтись без эффектов освещения. А для построения более-менее реалистичной модели освещения нужны нормали объекта. Но мы обычно проводим различные манипуляции с объектами в виде матриц

$$\overline{n}' = \left[\left(M^{-1} \right)^T \right]_{2\times 2} \cdot \overline{n}.$$

$\overline{n}$ - нормаль после применения преобзравонаие
$\overline{n}$ нормаль к одной из граней треугольника
M - матрица viev

19 Система координат наблюдателя (СКН). Переход от мировой системы координат к СКН. Вывод матрицы преобразования LookAt.

Система координат наблюдателя - правая декартова система координат, начало которой лежит в точке наблюдения, а ось O противоположно направлена вектору наблюдения. Таким образом, для перехода от мировой системы координат к системе координат наблюдателя необходимо: 1. перенести начало координат в точку наблюдения;

2. организовать вращение осей координат таким образом, чтобы ось Oz была направлена в сторону, противоположную вектору наблюдения, а ось Oy была направлена вверх (лежала в одной плоскости с осью Oz и вектором $\overline{u}$).

Первое преобразование простое: достаточно к однородным координатам каждой точки применить преобразование переноса с коэффициентами $T_x = -x_s, Ty = -y_s, T_z = -z_s$, т.е. необходимо выполнить преобразование заданное матрицей

Теперь можно получить матрицу вращения системы координат, совместив в ней векторы $\mathbf{e}'_1$, $\mathbf{e}'_2$, $\mathbf{e}'_3$ в качестве строк:

$$R = \begin{bmatrix} & \mathbf{e}'_1 & & 0 \\ & \mathbf{e}'_2 & & 0 \\ & \mathbf{e}'_3 & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Таким образом, для перехода из мировой системы координат в систему координат наблюдателя необходимо для каждой точки $(\chi, \gamma, \zeta, \alpha)$ выполнить преобразование

$$LookAt(S, P, \overline{u}) = R \cdot T$$

20 Организация интерактивного движения в пространстве наблюдателя

Суть движения заключается в переходе от одной СКН к другой, что осуществляется с помощью преобразования $LookAt(S', P', \overline{u}')$, S' — координаты новой точки наблюдения относительно текущей, P' — координаты точки, в которую будет смотреть наблюдатель, и $\overline{u}'$ — новый вектор направления вверх.

В итоге, для осуществления шага вперёд на единицу нам достаточно осуществить преобразование: LookAt((0, 0, -1),(0, 0, -2),(0, 1, 0)).

Повороты наблюдателя (относительно оси Oy): $LookAt((0,0,0), R(-\theta, e2)P, e2)$.

Наклоны наблюдателя: $LookAt((0,0,0), R(-\theta, e1)P, e2)$.

Обходы наблюдателем (вращение вокруг удалённой точки): $LookAt(S', (0,0,-dist), e2)$.

21 Система координат пространства отсечения. Вывод матрицы преобразования Ortho.

Пусть окно наблюдения задано произвольным образом в плоскости $z = -near$ параметрами *left*, *right*, *top* & *bottom*, а видимая глубина сцены ограничена параметром *far*. Тогда нам нужно рассмотреть часть пространства, ограниченную параллелепипедом, состоящим из точек, у которых $left \leq x \leq right$, $bottom \leq y \leq top$, $-far \leq z \leq -near$. Этот параллелепипед будем называть параллелепипедом видимости (см. рис 28).

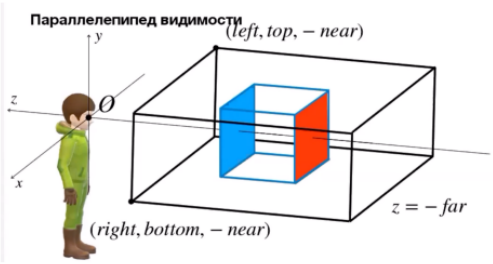


Рис. 28: Параллелепипед видимости

$$\begin{bmatrix} \chi' \\ \gamma' \\ \zeta' \\ \alpha' \end{bmatrix} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$Ortho(left, right, bottom, top, near, far)$$

22 Система координат пространства отсечения. Вывод матрицы преобразования Frustum

Если проведём всевозможные лучи, исходящие из глаза наблюдателя, проходящие через окно наблюдения, то они образуют бесконечную 4-х-угольную пирамиду с вершиной в глазу наблюдателя. Ограничим видимую часть сцены окном наблюдения с одной стороны (z = -near) и плоскостью z = -f ar. Считается, что дальше плоскости z = -f ar наблюдатель ничего не видит.

Получается усечённая 4-х-угольная пирамида (frustum), боковые грани её опираются на окно наблюдения.

$$\begin{bmatrix} \chi'' \\ \gamma'' \\ \zeta'' \\ \alpha'' \end{bmatrix} = \begin{bmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & \frac{-2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

23 Система координат пространства отсечения. Вывод матрицы преобразования Perspective.

Преобразование задаётся следующими параметрами: $near$ и far как в $Frustum$, $aspect = \frac{V_x}{V_y}$ — соотношение сторон окна наблюдения, $fovy$ — угол обзора по вертикали (*field of view y*). Важная деталь: наблюдатель должен смотреть в центр окна наблюдения, то есть ось Oz должна пересекать окно в его центре. Если рассмотрим сцену в проекции на плоскость yOz, то увидим треугольник, одна вершина которого лежит в глазу наблюдателя, а противолежащая ей сторона образована отрезком окна наблюдения, здесь $fovy$ — угол треугольника в точке наблюдения. Получим частный случай преобразования $Frustum$, зависящий от перечисленных выше параметров.

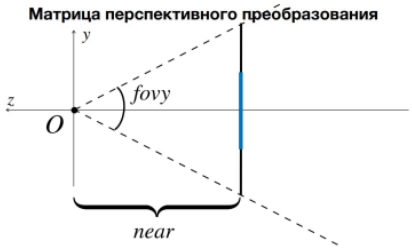


Рис. 37: Параметры преобразования

$$\begin{bmatrix} \chi'' \\ \gamma'' \\ \zeta'' \\ \alpha'' \end{bmatrix} = \begin{bmatrix} \frac{1}{aspect} \operatorname{ctg} \frac{fovy}{2} & 0 & 0 & 0 \\ 0 & \operatorname{ctg} \frac{fovy}{2} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & \frac{-2 \cdot far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рис. 40: Матрица преобразования $Perspective$

24 Растеризация отрезка. Алгоритм Брезенхема.

Алгоритм Брезенхема растеризации отрезка — это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками

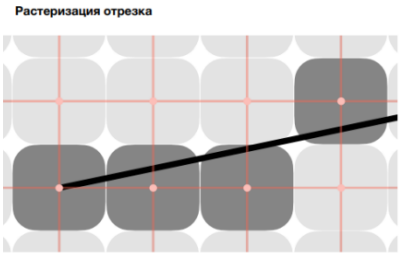


Рис. 41: Пример работы алгоритма Брезенхема

Рис. 42: Шаги по осям

25 Растеризация многоугольника. Алгоритм построчного заполнения

Алгоритм основан на том, что соседние пикселы в строке одинаковы и меняются только там, где строка пересекается с ребром многоугольника, для этого достаточно определить x-координаты пересечений строк сканирования с рёбрами.

Перед началом алгоритма стоит выполнить следующие действия: 1. Подготовить служебные целочисленные массивы y-координат вершин и номеров вершин.

2. Совместно отсортировать y-координаты по возрастанию и массив номеров вершин для того, чтобы можно было определить исходный номер вершины.

3. Определить пределы заполнения по оси Oy — ymin и ymax. Ввести переменные ycur и ycr, последняя служит для поддержания критической точки — следующей ближайшей строки y, на которой встречается вершина.

26 Алгоритм, использующий Z-буфер.

Z-буфер — это специальная память (двумерный массив), которая хранит для каждого пикселя экрана не цвет, а его глубину (координату Z). Глубина показывает, насколько далеко этот пиксель от камеры.

Для решения поставленной задачи создадим двумерный массив, у которого строк и столбцов столько же, сколько и точек растра у экрана. Для начала все элементы такого массива инициализируются максимальным значением координаты z, которое может быть у точки сцены — это 1, а всем точкам растрового пространства придаются атрибуты фона

Теперь поочерёдно выбираем объекты сцены и проводим их растеризацию. Выбираем объект, и, зная координаты z его вершин, проводим растеризацию многоугольника, в результате чего мы получаем соответствие каждому пикселю, на которые проецируется данный объект, некое значение координаты z в диапазоне от -1 до 1. Если значение для пикселя меньше значения в Z-буфере, то мы помещаем это значение в наш двумерный массив, а пиксел перекрашиваем в цвет объекта, который мы проецировали.

27 Элементы OpenGL: Графический конвейер

Для отрисовки объектов на трёхмерной сцене объекты должны пройти множество преобразований. Изначально положение модели задаётся в её собственной локальной СК. Последовательно применяем следующие преобразования:

1. modelView — модельное преобразование, переход в мировую СК.
2. cameraView — переход в СК наблюдателя.
3. clipView — переход в пространство отсечения.
4. C — переход к размерности целевого изображения.

После этого мы проводим растеризацию объектов, проводим тест глубины с использованием алгорима Z-буфера, и нам остаётся только изобразить полученные точки на экране.

Все эти операции необходимо выполнить для пикселя. Это приведёт к замедлению работы приложения, при использовании мощностей центрального процессора. Для решения этой проблемы можно использовать графический процессор (видеокарту). Специальные программы для видеокарты называются шейдерами. Шейдеры пишутся на языке GLSL (OpenGL Shading Language), синтаксис которого схож с синтаксисом языка C. На

изображении ниже можно увидеть примерное представление всех этапов графического конвейера. Синим выделены этапы, для которых можно специфицировать собственные шейдеры.

Графический конвейер содержит большое количество секций, где каждая занимается своей частью обработки вершинных данных в полностью отрисованный пиксел.

На вход конвейера передается массив 3D координат — вершинные данные, набор вершин (вершина — vertex). Каждая вершина задается своими трехмерными координатами и, возможно, дополнительными атрибутами (например, цвет). Во время отрисовки системе следует указать, что составить из переданного набора данных: набор точек, набор треугольников или просто одну ломаную линию (такие фигуры называются примитивами)

Первый этап конвейера — вершинный шейдер (Vertex Shader), который принимает на вход одну вершину. Основная задача вершинного шейдера — преобразование одних 3D-координат в другие. Возможность изменения этого шейдера позволяет выполнять преобразования в зависимости от параметров вершин. Предполагается, что на выходе вершинного шейдера координаты вершины заданы в пространстве отсечения.

Второй этап — сборка примитивов. Это этап, который принимает на вход все вершины от вершинного шейдера, формирующие примитив, и собирает из них сам примитив.

Результат этапа сборки примитивов передается геометрическому шейдеру (Geometry Shader). Он в свою очередь на вход принимает набор вершин, формирующих примитивы, из которого может сформировать новый набор возможно иных примитивов. Результат работы геометрического шейдера передается на этап растеризации, где результирующие примитивы будут соотноситься с пикселами на экране, формируя «фрагмент» для фрагментного шейдера (Fragment Shader). Фрагмент в OpenGL — все данные, которые нужны OpenGL для того, чтобы отрисовать пиксел. Перед выполнением фрагментного шейдера выполняется отсечение примитива (по умолчанию, относительно куба от -1 до 1 по каждой из осей), при котором отбрасываются все его части, которые находятся вне поля зрения, повышая таким образом производительность.

Основная цель фрагментного шейдера — вычисление конечного цвета пиксела. Зачастую фрагментный шейдер содержит всю информацию о 3D сцене, которую можно использовать для модификации финального цвета (типа освещения, теней, цвета источника света и т.д.).

После того, как определение всех соответствующих цветовых значений будет закончено, результат пройдет еще один этап, который называется альфа тестирование и смешивание. Этот этап проверяет соответствующее значение глубины (мы вернемся к этому позже) фрагмента и использует его для проверки местоположения фрагмента относительно других объектов: спереди или сзади. Этот этап также проверяет значения прозрачности и смешивает цвета, если это необходимо. Таким образом, результирующий цвет пиксела может отличаться от цвета, вычисленного фрагментным шейдером. Обычно геометрический шейдер оставляется стандартным. Но на

видеокартах не существует стандартного вершинного и фрагментного шейдера, в следствие чего в современном OpenGL их приходится задавать самостоятельно.

28 Вершинные данные. Вершинный массив. Вершинный буфер

Данные о вершиннах - массив 3D координат

Вершинные данные — это набор вершин

Вершина — это набор данных поверх 3D координаты, которыми могут выступать любые атрибуты, например, цвет.

Вершинный массив - органицаия вершыннх данных.

Вершинный буфер - управление памятью GPU, которая хранит большое количество в памяти вершин.

После определения вершинных данных требуется передать их в первый этап графического конвейера: в вершинный шейдер. Для того, чтобы вершинный шейдер обработал требуемое количество вершин, необходимо выделить память на GPU, в которой будут сохранены вершинные данные, указать OpenGL то, как он должен интерпретировать переданные ему данные, и сообщить GPU количество этих данных.

Эта память управляется через объекты вершинного буфера (vertex buffer objects (VBO)), которые могут хранить большое количество вершин в памяти GPU. Преимущество использования таких объектов буфера заключается в возможности посылать в видеокарту большое количество наборов данных за один раз, а не по одной вершине.

В OpenGL есть большое количество различных типов буферных объектов, так что каждый буферный объект должен быть привязан к контексту, через который и происходит общение с ним. Например, чтобы переслать данные из массива vertices в буферный объект vertexBuffer нужно сначала привязать буферный объект к контексту.

Для VBO характерен контекст GL_ARRAY_BUFFER, то есть буферный объект рассматривается как массив вершинных данных. После этой привязки все операции с GL_ARRAY_BUFFER будут относиться к vertexBuffer.

Вершинные данные организуются с помощью объектов вершинного массива (Vertex Array Object). Создается такой объект с помощью glGenVertexArrays. Для накопления информации в вершинном массиве его следует сделать активным. Теперь последующие команды, относящиеся к наполнению/изменению/чтению памяти GPU, будут ассоциироваться с этим объектом.

29 Шейдерная программа (варианты шейдеров). Порядок работы. Uniformпеременные

Шейдер — программа, исполняемая на графическом процессоре (GPU). Компиляция шейдера производится не во время сборки всего приложения, а в ходе выполнения приложения — для той архитектуры GPU, которая является актуальной при запуске.

Шейдеры загружаются в систему в составе шейдерной программы. Шейдерная программа — комбинация шейдеров, используемая для некоторого набора данных. В эту комбинацию обязательно входят один вершинный шейдер и один фрагментный шейдер. Возможно присутствие по одному экземпляру шейдера каждого другого вида. Для того чтобы составить шейдерную программу, нужно определить текст каждого шейдера, скомпилировать его, провести сборку скомпилированных шейдеров в единое целое.

При соединении шейдеров в программу переменные, объявленные выходными в одном шейдере (если такое объявление имело место), сопоставляются с входными переменными другого шейдера. Если входные и выходные значения не совпадают, сборка программы закончится неудачей, и может быть получен протокол о соответствующих ошибках. Скомпилированные шейдеры должны присоединяться к программе в том порядке, в котором они должны работать: сначала вершинный шейдер, а затем фрагментный.

Uniform-переменные — глобальные переменных шейдеров. Uniform-переменные определяются для шейдерной программы и к ним возможен доступ из любого шейдера. Значение uniform-переменной загружается в GPU с помощью семейства процедур glUniform

30 Типы источников света: точечный, линейный, плоскостной, бесконечно удаленный. Влияние типа источника света на освещенность точки.

Точечный источник представляет из себя точку, которая имеет заданное положение в пространстве, равномерно излучающую свет во всех направлениях. Интенсивность света обратно пропорциональна квадрату расстояния от источника до объекта сцены.

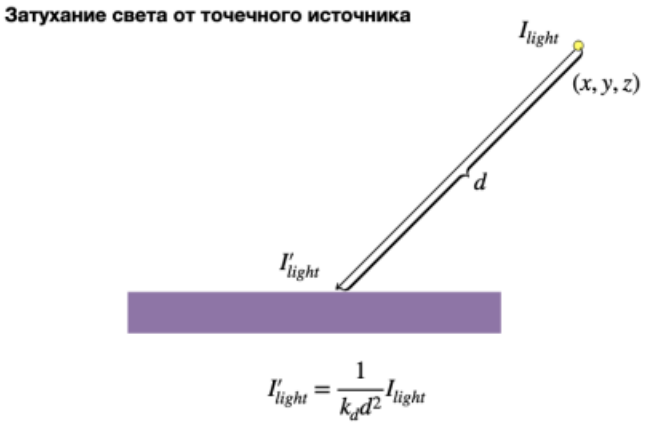


Рис. 55: Точечный источник

Линейный источник представляет из себя бесконечную прямую в заданном положении в пространстве сцены, которая равномерно излучает свет относительно этой линии. Интенсивность освещения вычисляется интегрированием по прямой.

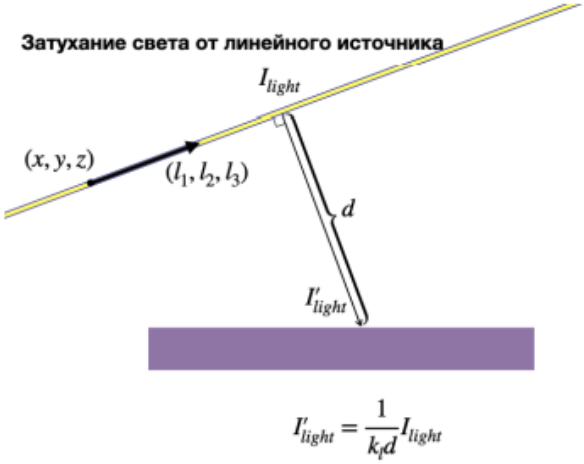


Рис. 56: Линейный источник

Плоскостной источник света имеет заданное положение в пространстве и равномерно излучает свет относительно плоскости, а интенсивность освещения не зависит от расстояния.

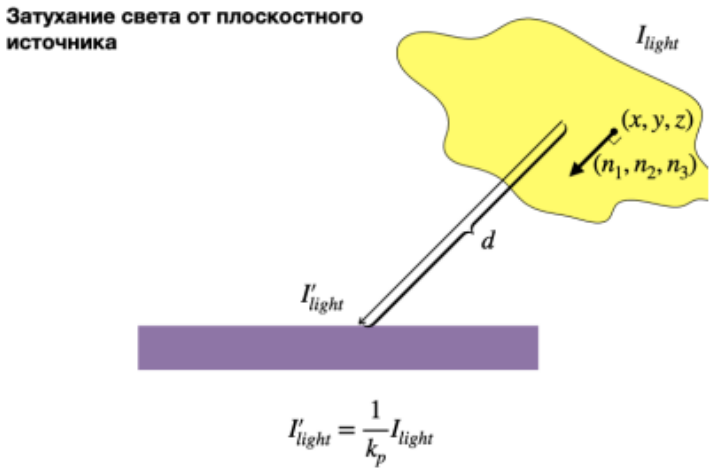


Рис. 57: Плоскостной источник

Бесконечно удаленный источник света — это тот же плоскостной источник за несколькими отличиями:

1. Он всегда освещает только одну и ту же строну объекта, т. к. объект не может его достигнуть и пролететь сквозь него.

2. У бесконечно удалённого источника света нельзя измерить расстояние до объекта.

31 Модель освещения Фонга.

В реальности фоновая освещённость складывается из очень большого количества источников света, подавляющую часть которых составляют отражённые, и воссоздание такой модели либо очень трудно, либо невозможно, т.к. это требует большого количества вычислений. Поэтому используют упрощённый способ: каждому источнику света задают вектор ambient — интенсивность фонового освещения этого источника для всех объектов сцены.

toy - цвет поверхности.

$I_result = Attenuation * I_{toy} * I_{ambient}$

Диффузная освещенность объекта в произвольной точке зависит от угла падения вектора от источника света в точку на поверхности объекта.

Вектор нормали - n

Вектор напрваления на источник света из точки l

$I_result = Attenuation * (I_{toy} * I_{diffuse}) max(0, \overline{n} * \overline{l})$

Бликовая освещенность

Вектор r — вектор отраженного света.

Вектор v — задает вектор направления из точки плоскости в точку наблюдателя.

Параметр глянцевости - shinness.

$I_result = Attenuation * (I_{toy} * I_{specullar})(max(0, \overline{r} * \overline{v}))^{shinness}$

32 Модель освещения Блинна-Фонга.

Данная модель незначительно отличается от предыдущей и её появление обусловлено следующим: из-за выбора угла $]theta$ (аргумента косинуса) подобным образом границы блика могут быть явными, что негативно сказывается на реалистичности изображения. Поэтому в данной модели угол выбирается иным способом. Вводится медианный вектор h, направление которого является средним между направлениями вектора падения и вектора, направленного в глаз наблюдателю, h необходимо нормировать. Далее выбирается угол ψ между этим вектором и вектором нормали к поверхности и уже этот угол будет аргументом косинуса.

$I_result = Attenuation * (I_{toy} * I_{specullar})(max(0, \overline{h} * \overline{n}))^{shinness}$

33 Модели затенения Гуро и Фонга

Иногда для обеспечения реалистичности изображения требуется скруглять углы и линии пересечения граней объекта. Для решения этой проблемы призван алгоритм затенения, заключающийся в следующем: вектор нормали к вершине вычисляется как средний вектор между всеми нормалями граней, которые сходятся в вершине, и нормируется. Зная такой вектор, мы можем вычислить цвет каждой вершины многогранника и приписать его каждой вершине соответствующей грани. Далее в процессе растеризации цвет усреднится по ребрам и граням многогранника. Так получим сглаженный многогранник. Такая модель закраски называется моделью Гуро.

Модель Фонга строится похожим способом, только в процессе растеризации усредняется не цвет, а нормали, то есть, несмотря на то, что грань плоская, в каждой точке грани будет свой вектор нормали. В этом случае плоская грань может восприниматься как выпуклая.

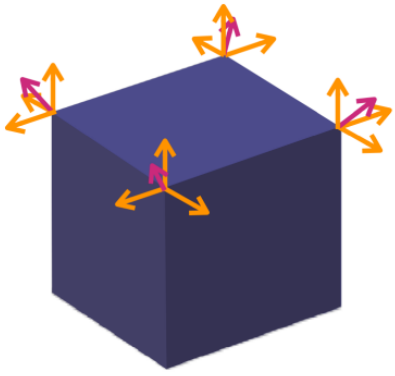


Рис. 58: Модель затенения Гуро

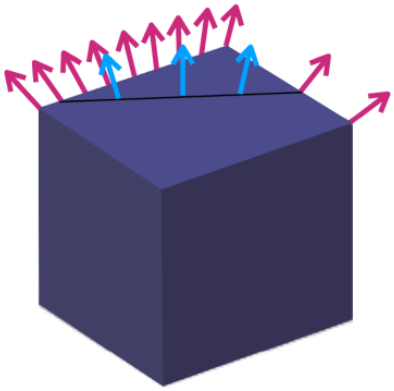


Рис. 59: Модель затенения Фонга

34 Порядок реализации модели Гуро с помощью шейдеров в OpenGL.

На картинке ниже видно, что вычислением цвета занимается вершинный шейдер, фрагментный шейдер же просто «пропускает» сквозь себя вычисленный ранее результат. На этапе растеризации (между работой вершинного и фрагментного шейдеров) интерполируется уже вычисленный ранее цвет, плавно переходя от одной вершины к другой. По этой причине блики отображаться не будут. (координаты в переменной glP ostition — это координаты вершины в СК пространства отсечения)

35 Порядок реализации модели Фонга с помощью шейдеров в OpenGL.

В этой модели вершинный шейдер на выходе даёт нормаль и координаты вершины в СК наблюдателя. В процессе растеризации интерполируются обе эти составляющие, и только во фрагментном шейдере происходит вычисление цвета для каждого фрагмента каждой грани, при этом известно положение наблюдателя — начальная точка СК. Получается, что в данной модели происходит значительно больше вычислений, чем в модели Гуро, но результат является более реалистичным, в том числе происходит корректная обработка бликов.