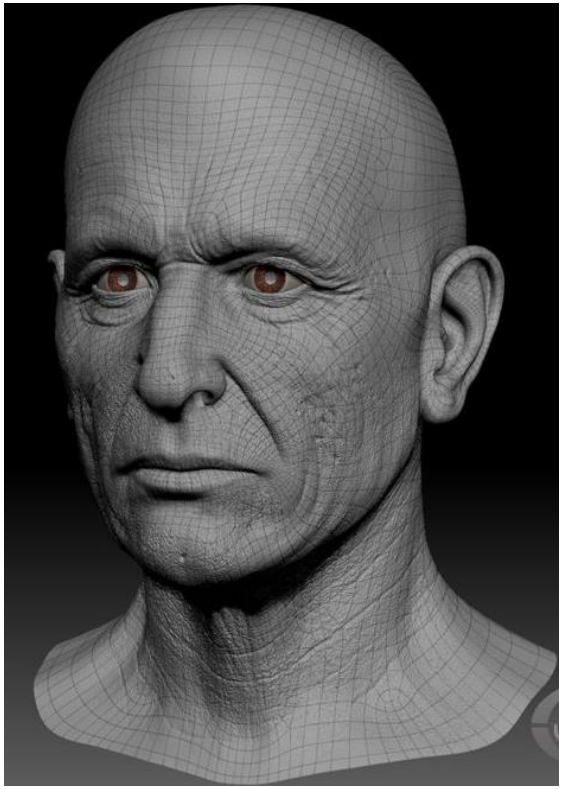
Рендеринг

Рендеринг - это процесс преобразования объекта или сцены, созданных в приложении трехмерной графики, для вывода на дисплей, который представляет собой ***двухмерную плоскость***.



На стадии рендеринга по описанию треугольников генерируются пиксели изображения.

В отличие от механизма геометрических преобразований в процессе рендеринга объем операций с плавающей точкой не столь велик и в основном состоит из простых операций над пикселями.

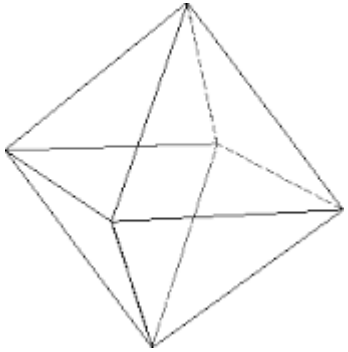
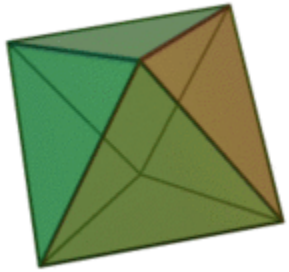
Удаление невидимых линий

Трехмерная сцена составлена из множества объектов со взаимным расположением по глубине в зависимости от точки визирования.

Объекты и сцены можно вращать и наблюдать с разных позиций, в том числе спереди, сбоку, сзади, сверху и снизу.

Позиция объекта меняется в зависимости от положения и направленности наблюдателя.

Например, при виде спереди объект может быть видим полностью, а с положения сзади - заслоняться другим объектом.


К сожалению, картинка из одних лишь вершин (и даже вершин, соединенных ребрами) удовлетворит далеко не всех пользователей. Поэтому на стадии рендеринга производится удаление невидимых линий.

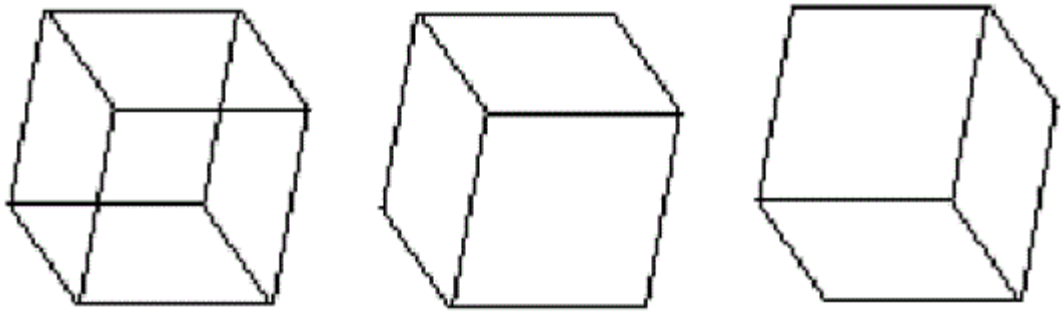
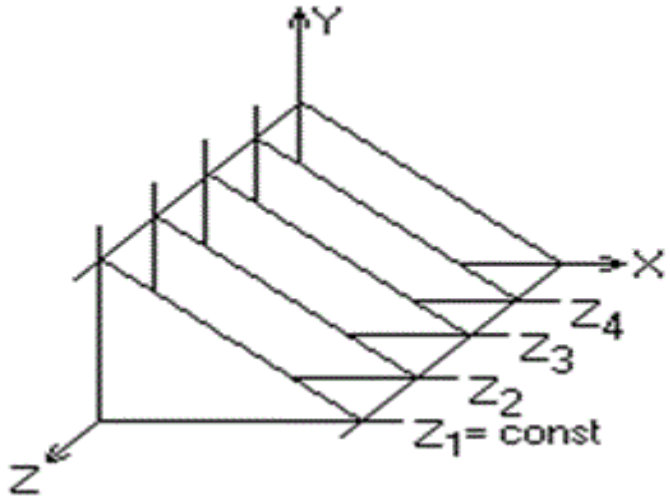


Рисунок куба можно интерпретировать двояко: как вид куба сверху, слева или снизу, справа. Удаление тех линий или поверхностей, которые невидимы с соответствующей точки зрения, позволяют избавиться от неоднозначности.

Алгоритм плавающего горизонта

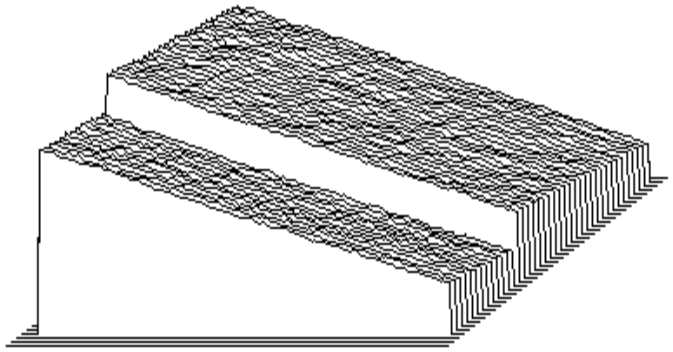


Рассмотрим задачу построения графика функций трех переменных

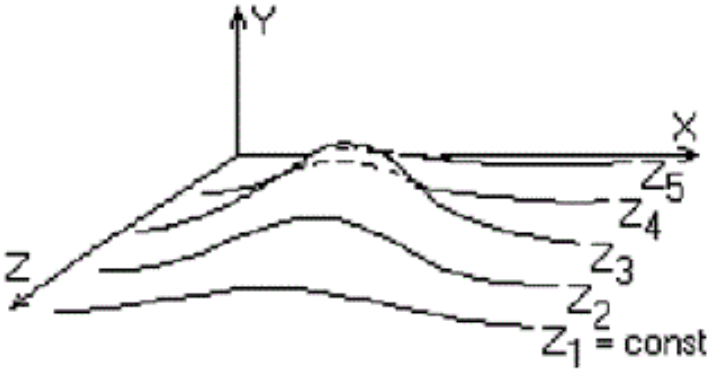
$$f(x,y,z)=0$$

в виде сетки координатных линий для которых

$$z = const$$



Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z.



Функция $f(x,y,z)=0$ сводится к последовательности линий, лежащих в каждой их заданных параллельных плоскостей $y = f(x, z_i)$, где z_i постоянна для каждой из плоскостей.

Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис.

Заметим, что каждая линия семейства $y = f(x, z_i)$ лежит в своей плоскости $z = z_i$, причем эти плоскости параллельны и, следовательно, не пересекаются.

Поэтому при $z_j > z_i$ линия $y = f(x,z_j)$ не может закрывать линию $y = f(x,z_i)$.

Тогда возможен следующий алгоритм удаления невидимых линий:

Линии рисуются в порядке удаления (возрастания по z) и при рисовании очередной линии рисуется только та ее часть, которая не закрывается ранее нарисованными линиями. Такой алгоритм называется методом плавающего горизонта.

Для определения частей линии $y=f(x,z_k)$, которые не закрывают ранее нарисованные линии, вводятся линии горизонта или контурные линии.

Реализация данного алгоритма достаточно проста.

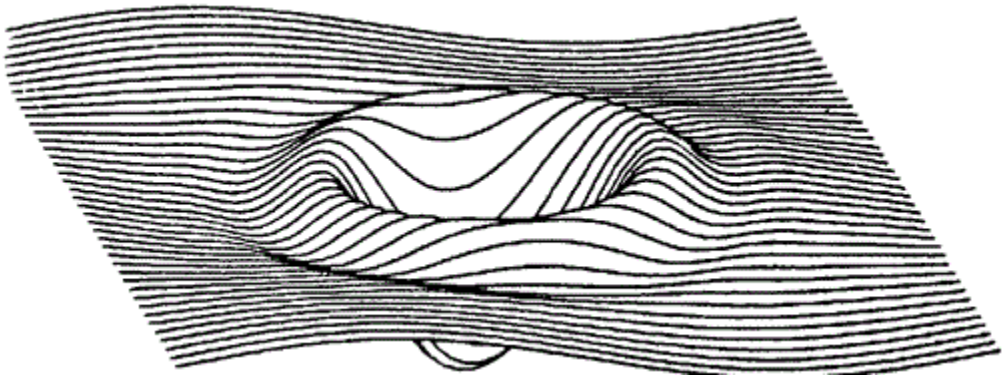
Для хранения максимальных значений y при каждом значении x используется массив, длина которого равна числу различимых точек (разрешению) по оси x в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения «горизонта». Поэтому по мере рисования каждой очередной кривой этот горизонт «всплывает». Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией

Пусть проекцией линии $y=f(x,z_k)$ на картинную плоскость является линия $Y=f_k(X)$, где (X,Y) – координаты на картинной плоскости.

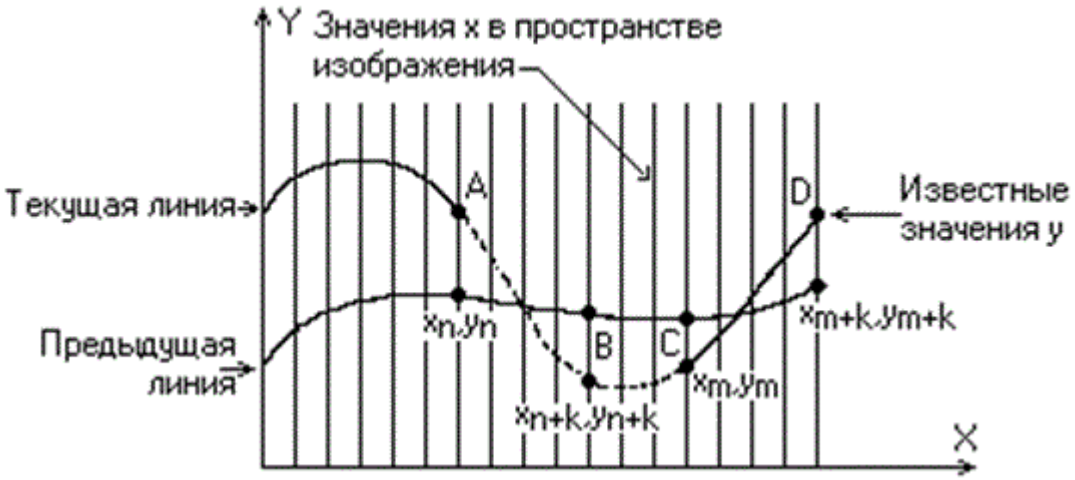
Контурные линии $Y^k_{max}(X)$ и $Y^k_{min}(X)$ определяются следующими соотношениями:

$$Y^k_{max}(X)=\max Y_i(X),$$
$$Y^k_{min}(X)=\min Y_i(X), \text{ где } 1<=k<=k-1$$

На экране рисуются только те части линии $Y=Y_k(X)$, которые находятся выше линии $Y^k_{max}(X)$ или ниже линии $Y^k_{min}(X)$.



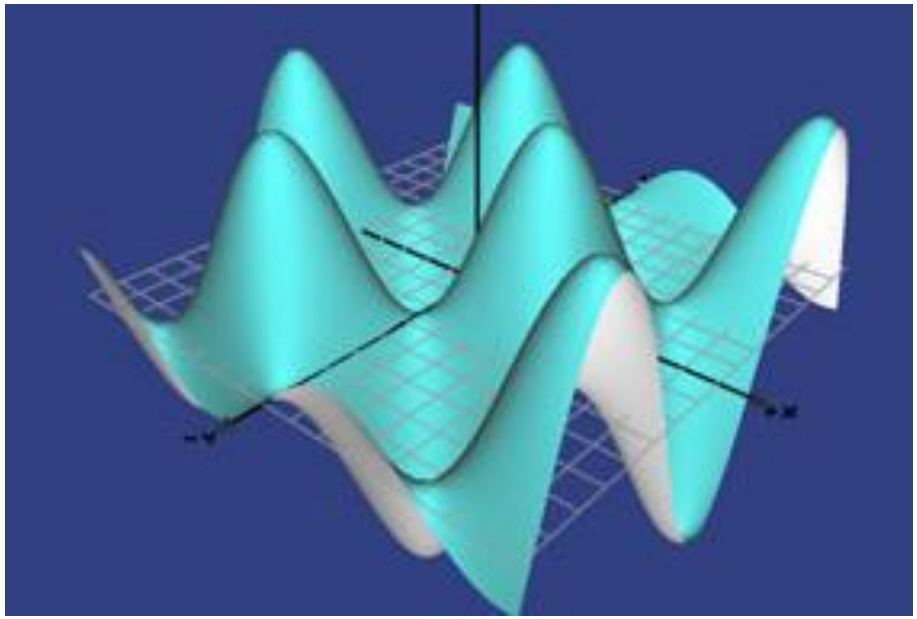
Одной из наиболее простых и эффективных реализации данного метода является растровая реализация, при которой в области задания вводится сетка $\{ (x_j,y_j), i=1, , n_i \}$ и каждая из линий $Y=Y_k(X)$ представляется в виде ломаной.



Размер сетки может быть меньше, чем видимых точек в сечении X.
Размер массивов плавающего горизонта должен совпадать с разрешением экрана монитора.

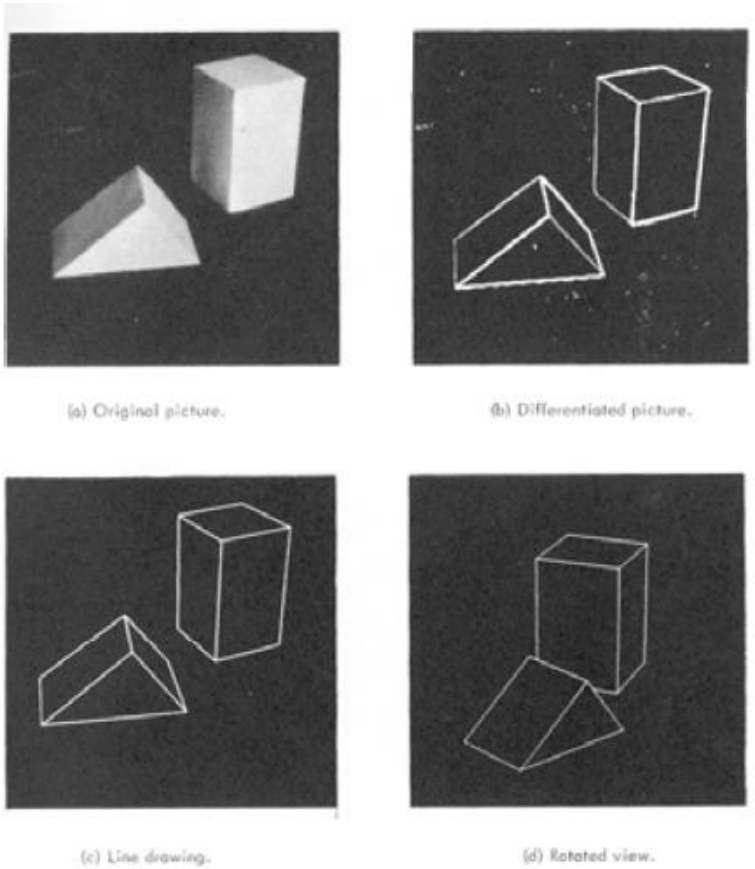
Поэтому для рисования сегментов этой ломаной используется модифицированный алгоритм Брезенхейма, который перед выводом очередного пиксела сравнивает его ординату с верхней и нижними контурными линиями, представляющими в этом случае массивы значений координат.

Очень хорошо выглядят графики, построенные по методу плавающего горизонта, если для каждого квадрата сетки определить соответствующий ему многоугольник, определить нормаль к нему и в зависимости от нормали раскрасить его пропорционально направлению освещения.



Алгоритм Робертса (1969 г.)

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий.



(a) Original picture.

(b) Differentiated picture.

(c) Line drawing.

(d) Rotated view.

1960 г:
L. G. Roberts, *Machine Perception of Three Dimensional Solids,* Ph.D. thesis, MIT Department of Electrical Engineering

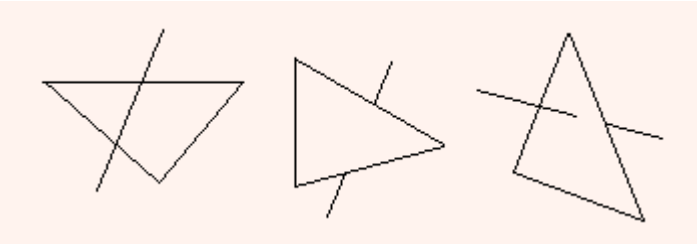
Алгоритм требует, чтобы все изображаемые тела или объекты были ***выпуклыми***. Невыпуклые тела должны быть разбиты на выпуклые.

Алгоритм работает в объектной плоскости.

Сначала удаляются из каждого тела те ребра или грани, которые экранируются самим телом.

Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения какая его часть экранируется этими телами.

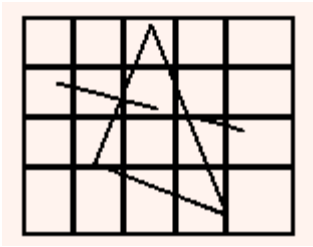
Возможны следующие варианты:



- грань не закрывает ребро;
- грань полностью закрывает ребро (тогда она удаляется из списка ребер);
- грань частично закрывает ребро (В этом случае ребро разбивается на несколько частей, из которых видимыми являются не более 2. Само ребро удаляется из списка, но в список добавляются его части не закрываемые гранью).

Если общее количество граней равно n, то время работы пропорционально n^2.

Можно заметно сократить число проверок, если разбить картинную плоскость на равные клетки. Для каждой клетки составляется список тех граней, проекции которых имеют пересечение с этой клеткой.



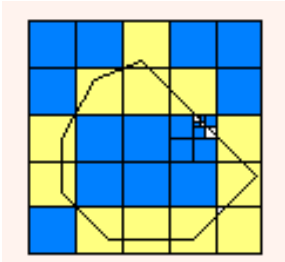
Для проверки произвольного ребра сначала находятся все клетки, в которые попадает проекция этого ребра, и рассматриваются только те грани, которые содержатся в стеках этих клеток.

Алгоритм Варнока (1968)

Этот вариант при удачном выборе разбиения пропорционален n.

Видимую часть картинной плоскости разобъём на 4 равные части. В случаях, когда часть полностью накрывается проекцией ближайшей грани и часть не накрывается проекцией ни одной грани – вопрос ясен.

В случае, когда ни одно из условий не выполнено, данная часть разбивается опять на 4 и т.д.



Разбиение имеет смысл проводить до тех пор, пока размер части больше, чем размер пиксела. В этом случае, явно находится ближайшая к ней грань и осуществляется закрашивание.

Реализация алгоритма Робертса для выпуклых многоугольников

Требуется, чтобы все изображаемые тела или объекты были выпуклые. Многогранное тело с плоскими гранями должно представляться набором пересекающихся плоскостей.

Уравнение плоскости

$$ax_1+b_1+c_1z+d_1=0$$

Любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей

$$[V]=\begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{bmatrix}$$

Любая точка пространства представима в однородных координатах вектором [s]=[x,y,z,1].

Если точка s лежит на плоскости, то $[s][P]^T$ ($[P]^T$ - столбец [V]). Если же точка не лежит на плоскости, то знак этого скалярного произведения показывает по какую сторону от плоскости расположена эта точка.

Пример. 6 плоскостей описывают единичный куб:

$$x_1=1/2\ x_2=-1/2\ y_3=1/2\ y_4=-1/2\ z_5=1/2\ x_6=-1/2$$

Уравнение первой плоскости

$$x_1=1/2 \Rightarrow x_1-1/2=0 \Rightarrow 2x_1-1=0.$$

Матрица тела для куба

$$[V]=\begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \end{array}$$

Знаки плоскостей задаются произвольно. (Верно и $2x_1-1=0$ и $-2x_1+1=0$). Необходимо матрицу тела проверить с помощью точки, лежащей, например, внутри, чтобы убедится, что знаки у координат всех плоскостей заданы верно.

Если знак скалярного произведения для какой-либо плоскости <0, то соответствующее уравнение плоскости следует умножить на -1.

Выберем точку внутри куба [1/4 1/4 1/4 1]. Скалярное произведение

$$[s][V] = [1\ 1\ 1\ 4] \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} = [-2\ 6\ -2\ 6\ -2\ 6]$$

Результаты для 1, 3, 5 уравнений <0, следовательно коэффициенты этих уравнений необходимо умножить на –1 для получения корректной матрицы.

Корректная матрица

$$[V] = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

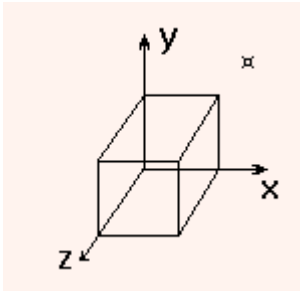
Теперь определим грани объекта, которые не видны.

Способ эквивалентен вычислению нормали к поверхности.

Отрицательность нормали у поверхности показывает, что нормаль направлена в сторону от наблюдателя и, следовательно, данный многоугольник (грань) не виден.

Если зритель находится в бесконечности (ось z направлена от наблюдателя), вектор такого наблюдения равен.

$$[E] = [0, 0, -1, 0].$$



Фактически [E] представляет любую точку, лежащую на плоскости $z = -\infty$. Т.е. любую точку типа $[x, y, -\infty]$. Поэтому, если скалярное произведение этого вектора на столбец в [V], соответствующий какой-либо плоскости <0, то [E] лежит по отрицательную сторону этой плоскости. Следовательно эти плоскости ***невидимы*** из любой точки наблюдения, лежащей в плоскости $-\infty$.

$$[E][V] = [0,\ 0,\ -1,\ 0] \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = [0,\ 0,\ 0,\ 0,\ 2,\ -2]$$

Отрицательное число в 6 столбце показывает, что грань с этим номером нелицевая. Нулевые результаты соответствуют плоскостям параллельным направлению взгляда.

Этот метод является простейшим алгоритмом удаления невидимых поверхностей для тел, представляющих собой одиночные выпуклые многогранники. Метод используется также для удаления задних граней из сцен перед применением других алгоритмов удаления невидимых линий.

Для выпуклых многогранников число граней при этом сокращается наполовину.

Пример:

Рассмотрим единичный куб с центром в начале координат, повернутый на 45 градусов вокруг оси y. Матрица преобразований будет иметь вид.

$$[R_y] = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразование матрицы тела получается умножением исходной матрицы V слева на матрицу обратную $[R_y]$. Для чистого поворота обратная матрица сводится к транспонированной.

$$[R_y]^{-1} = [R_y]^T = \begin{bmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

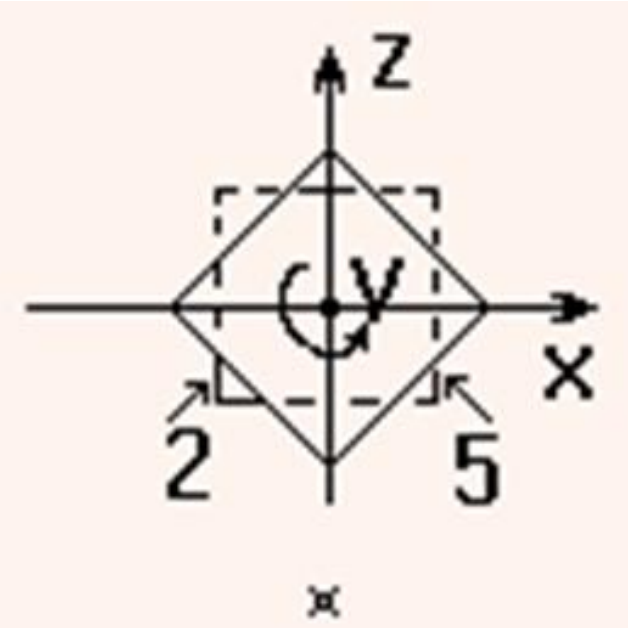
Само преобразование

$$[VT] = [R_y]^{-1}[V] = \begin{bmatrix} -2\sqrt{2} & 2\sqrt{2} & 0 & 0 & -2\sqrt{2} & 2\sqrt{2} \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 2\sqrt{2} & -2\sqrt{2} & 0 & 0 & -2\sqrt{2} & 2\sqrt{2} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Скалярное произведение

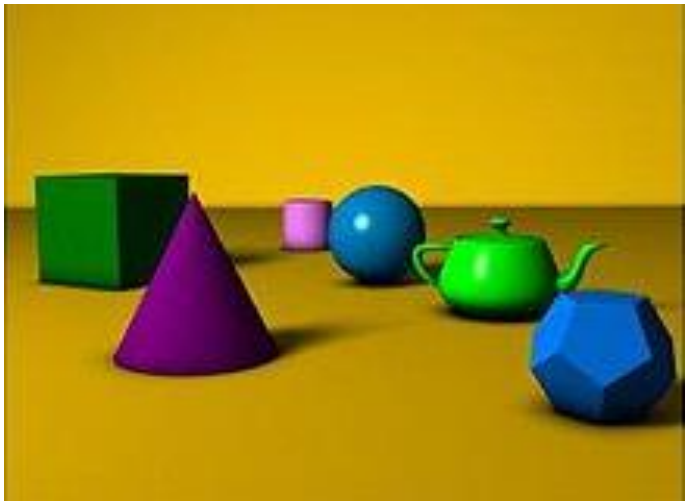
$$[E][VT] = \begin{vmatrix} -2\sqrt{2} & 2\sqrt{2} & 0 & 0 & 2\sqrt{2} & -2\sqrt{2} \end{vmatrix}$$

ИТОГО. Видны 2 и 5 грани.



Метод z-буфера

Один из самых простых методов. В настоящее время один из самых популярных. Очень эффективен и практически не имеет недостатков, легко реализуется аппаратно.



Z-буфер представляет собой двумерный массив, каждый элемент которого соответствует пикселю на экране.

Основной недостаток Z-буферизации состоит в потреблении большого объёма памяти: в работе используется так называемый буфер глубины или Z-буфер.

Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости (x, y); обычно бывает достаточно 20 бит. Буфер кадра размером 512x512x24 бит в комбинации с z-буфером размером 512x512x20 бит требует почти *1.5 мегабайт* памяти.

Сопоставим каждому пикселу (x,y) картинной плоскости, кроме цвета, его расстояние до картинной плоскости вдоль направления проектирования z(x,y). (глубина).

Вначале в z-буфер заносятся максимально возможные значения z, а значения цвета соответствует фону.

Представление в Z-буфере

Затем для каждой грани делаем растровое представление на картинной плоскости и для каждого пиксела этой грани находится его глубина.

В случае, если эта глубина меньше значения глубины, хранящейся в z-буфере, пиксел рисуется (заполняется цвет) и его глубина заносится в буфер.

Построчное сканирование с z-буфером

Альтернативой созданию специальной памяти для z-буфера является использование для этой цели оперативной памяти. Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос.

В предельном варианте можно использовать z-буфер размером в одну строку развертки.

Окно визуализации имеет высоту в одну сканируемую строку и ширину во весь экран.

Например: 1 x 1024 x 24 бита буфер кадра (цвета)

1 x 1024 x double z-буфер

Для каждой сканирующей строки буфер кадра инициализируется фоновым значением интенсивности, а z-буфер минимальным значением z координаты.

Определяется пересечение сканирующей строки с двумерной проекцией каждого многоугольника сцены.

Глубина каждого элемента плоскости вдоль отрезка, образующегося при пересечении, сравнивается с глубиной в соответствующем элементе z-буфера.

Пусть некоторый многоугольник построчно преобразуется в растровую форму. Вычисление координаты z для каждой точки сканирующей строки можно упростить, воспользовавшись тем, что многоугольник плоский.

Уравнение плоскости $Ax + By + Cz + D = 0 \quad z = (-D-Ax-By)/C$.

Теперь, если в точке (x,y) найдено значение z_1,

следовательно в точке (x+dx, y)

$$z = z_1 - \frac{A}{C} dx$$

Отношение A/C – постоянно, а dx=1 поэтому, если задана глубина в точке x,y, для вычисления глубины в точке (x+1,y) требуется выполнить лишь одно вычитание.

Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование z-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены.

Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост. Существует много модификаций этого алгоритма. Для сокращения операций заводятся дополнительные массивы (например: массив по y, в котором указываются многогранники имеющие пересечение со строкой при этом значении y, и т.п.).