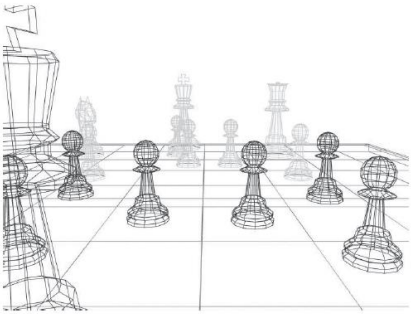
Лекция 2.2. Удаление невидимых поверхностей

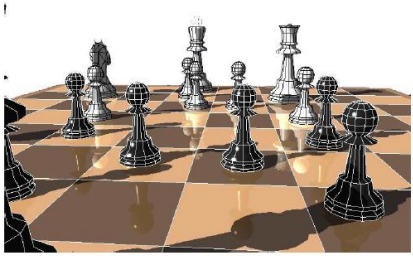
Задача экранирования

Конечное изображение является плоским и не способно передать объема.

Перекрывающие друг друга объекты в трехмерном пространстве экранируют друг друга

Имитация экранирования объектов на плоскости является одной из задач HSR-алгоритмов.





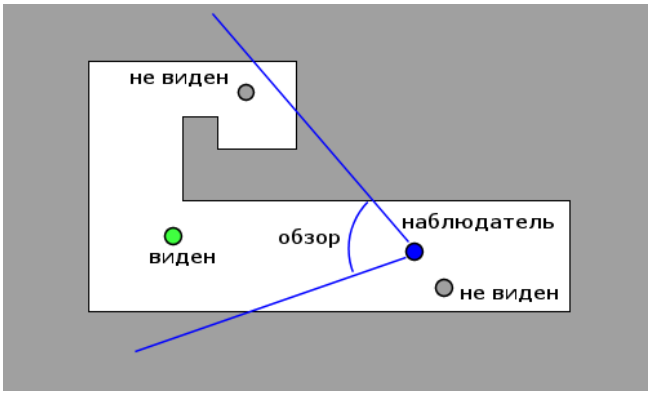
Все алгоритмы удаления в трехмерном пространстве можно классифицировать несколькими способами:

1. По объекту удаления:
   - линий
   - поверхностей
   - объектов
   - объемов
2. По порядку обработки объектов:
   - в произвольном порядке
   - в порядке, определяемом алгоритмом
3. По рабочему пространству:
   - пространство объекта
   - пространство мира
   - пространство изображения

Дополнительная задача

Рост производительности графических систем привел к быстрому росту детализации и размеров сцен.

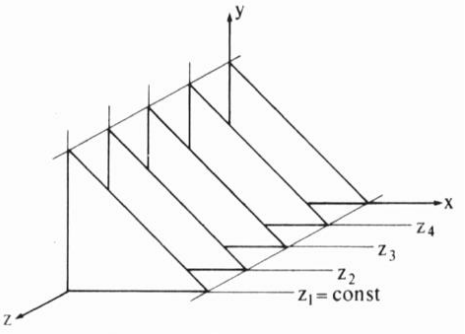
На настоящий момент HSR алгоритмы вынуждены решать задачи связанные не только с отображением корректной экранированной модели сцены, но так же и задачи сокращения обрабатываемого материала – отбрасывания тривиально невидимых объектов и целых семейств объектов.



Плавающий горизонт

Данный алгоритм работает в пространстве изображения. Основное назначение – корректное отображение экранирующей саму себя аналитической поверхности, задаваемой в пространстве функцией вида

$$F(x, y, z) = 0.$$



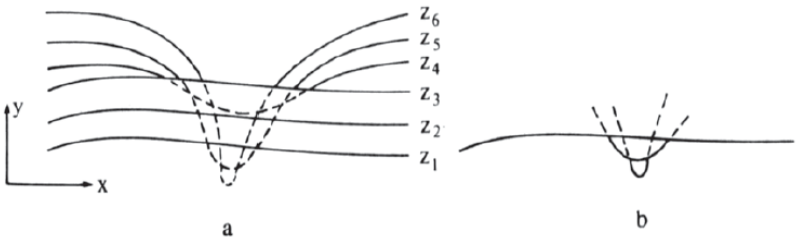
Идея алгоритма – сведение трехмерной задачи к двухмерной. Изначальная поверхность последовательно пересекается параллельными плоскостями, имеющих постоянные значения координат x, *y* или z. Внутри подобных плоскостей находятся очертания кривых, описывающих поверхность, на базе которых строится конечное изображение.

Так как далеко не все поверхности можно описать аналитически, данный алгоритм имеет достаточно ограниченную область применения.

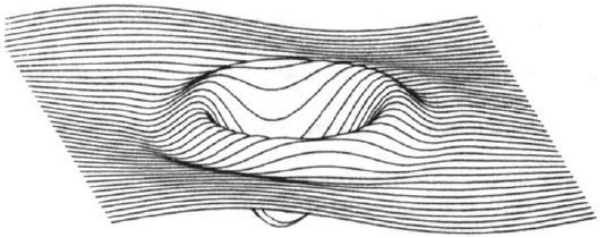
Алгоритм основан на фиксации значения, например z, то есть $y=f(x, z)$ или $x=g(y, z)$. Проще говоря, тело разбивается на параллельные плоскости. Поверхность складывается из последовательности кривых, лежащих в каждой из этих плоскостей. Предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость $z=0$, то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности. Алгоритм сначала упорядочивает плоскости $z=const$ по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты x в пространстве изображения определяется соответствующее значение y.

Алгоритм удаления невидимой линии заключается в следующем:

Если на текущей плоскости при некотором заданном значении x соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x, то текущая кривая видима. В противном случае она невидима. Набор максимальных значений y для каждого значения x называется верхней линией горизонта, набор минимальных значений — нижней линией горизонта.



Обработка нижней стороны поверхности



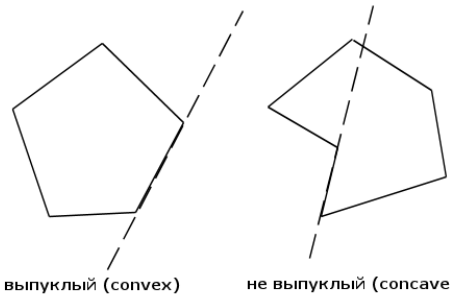
Функция y = (1/5) sin x cos z – (3/2) cos(7a/4)×exp(-a), a = (x – π)2 + (z - π)2, изображенная в интервале (0, 2π) с помощью алгоритма плавающего горизонта

Алгоритм Робертса

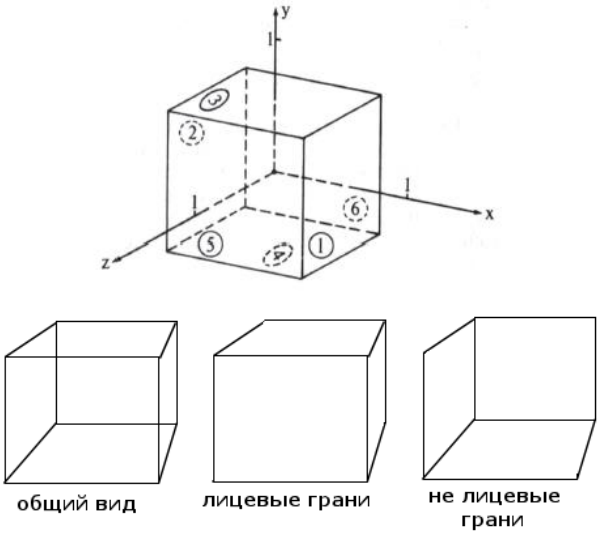
Первое известное решение задачи об удалении невидимых линий (1963 г). *Этот метод работает в объектном пространстве.*

Предполагается, что все объекты являются выпуклыми оболочками (hull), либо могут быть представлены в виде совокупности выпуклых оболочек.

Выпуклой оболочкой считается такой полигональный объект, что для любого образующего его полигона все остальные полигоны будут находится по одну сторону плоскости этим полигоном образованной.



Алгоритм, прежде всего, удаляет 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 того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Более поздние реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси *z* и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов.



В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части. В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей.

Уравнение произвольной плоскости в трехмерном пространстве имеет вид:

$$ax + by + cz + d = 0 \quad \text{или}$$

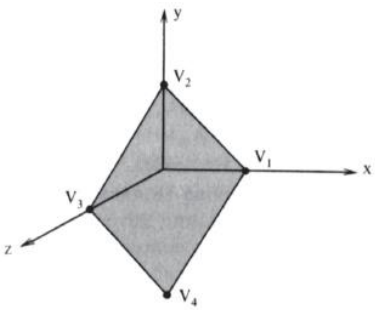
$$S \cdot P^T = 0, \text{ где}$$

$$S = [x \quad y \quad z \quad 1]$$

$$P^T = [a \quad b \quad c \quad d]$$

Поэтому любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

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



Если некоторая точка S лежит на плоскости, то

$$S \cdot P^T = 0.$$

Если же S не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. В алгоритме Робертса предполагается, что точки, лежащие внутри тела, дают положительное скалярное произведение.

Метод, предложенный Мартином Ньюэлом, позволяет найти как точное решение для уравнений плоскостей, содержащих плоские многоугольники, так и «наилучшее» приближение для неплоских многоугольников. Этот метод эквивалентен определению нормали в каждой вершине многоугольника посредством векторного произведения прилежащих ребер и усреднения результатов. Если *a, b, c, d* — коэффициенты уравнения плоскости, то а *d* вычисляется с помощью любой точки на плоскости.

$$a = \sum_{i=1}^{n} \sum_{j=1}^{n} (y_i - y_j)(z_i - z_j)$$

$$b = \sum_{i=1}^{n} \sum_{j=1}^{n} (z_i - z_j)(x_i - x_j)$$

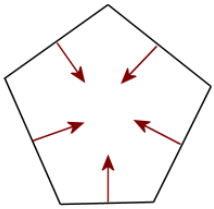
$$c = \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i - x_j)(y_i - y_j)$$

Выпуклые оболочки определяются как совокупность пересекающихся плоскостей. В таком случае они однозначно могут определить, находится та или иная точка внутри, либо снаружи.

Представим, что плоскость можно описать уравнением вида

$$ax + by + cz + d = 0$$

Причем нормаль плоскости направлена внутрь оболочки:

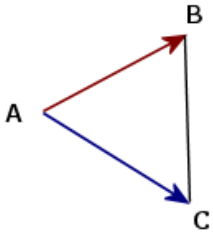


В этом случае точка будет лежать внутри оболочки тогда и только тогда, когда при подстановке ее координат в уравнения **всех** плоскостей, результат будет **положительным**.

Соответственно, если **хотя бы один** из результатов будет отрицателен, то можно говорить, что точка находится вне оболочки.

Коэффициенты плоскости **[a, b, c, d]** могут быть определены через три вершины треугольника следующим образом:

Вектор **(a, b, c)** является вектором нормали треугольника и может быть найден через **нормализованное** (приведенное к единичному вектору) векторное произведение **AB x AC**.



В этом случае коэффициент **d** может быть найден как скалярное произведение нормали и координат любой из вышеупомянутых вершин, взятое с обратным знаком:

$$d = -(a, b, c) * A$$

В дальнейшем будем считать, что все полигоны, образующие оболочку имеют две грани – **лицевую** и **не лицевую**.

Характерной чертой выпуклых оболочек является тот факт, что все не лицевые грани всегда экранируются лицевыми, то есть, с точки зрения алгоритма являются **невидимыми**.

В то же время ни одна из лицевых граней не может экранировать какую-либо другую лицевую грань.

Данные два свойства позволяют выделить суть проблемы: **при формировании изображения на экран достаточно вывести лишь лицевые грани, т.е. полигоны «повернутые» к наблюдателю «лицевой» гранью. В этом случае полученная фигура не будет экранировать сама себя.**

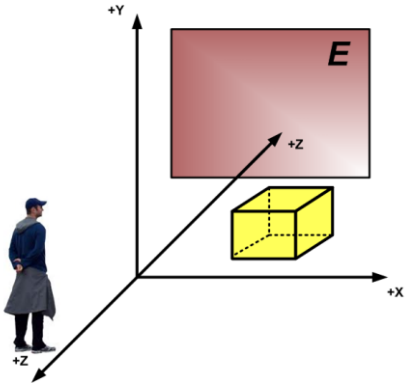
Тот факт, что плоскости имеют бесконечную протяженность и что скалярное произведение точки на матрицу тела отрицательно, если точка лежит вне этого тела, позволяет предложить метод, в котором матрица тела используется для определения граней, которые экранируются самим этим телом. Отрицательное скалярное произведение дает только такая плоскость (столбец) в матрице тела, относительно которой точка лежит снаружи.

Если зритель находится в бесконечности на положительной полуоси z и смотрит на начало координат, то его взгляд направлен в сторону отрицательной полуоси z. В однородных координатах вектор такого направления равен:

$$E = (0\ 0\ -1\ 0)$$

который служит, кроме того, образом точки, лежащей в бесконечности на отрицательной полуоси z. Фактически E представляет любую точку, лежащую на плоскости $z = -\infty$ т. е. любую точку типа $(x, y, -\infty)$. Поэтому, если скалярное произведение E на столбец, соответствующий какой-нибудь плоскости в матрице тела, отрицательно, то E лежит по отрицательную сторону этой плоскости. Следовательно, эти плоскости невидимы из любой точки наблюдения, лежащей в плоскости $z = \infty$ а пробная точка на $z = -\infty$ экранируется самим телом.

Такие плоскости называются нелицевыми, а соответствующие им грани — задними.



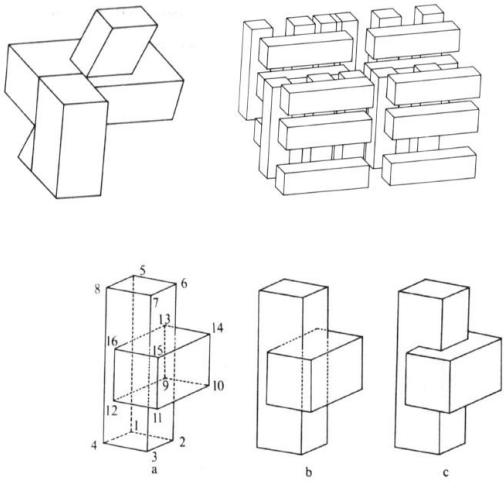
Этот метод является простейшим алгоритмом удаления невидимых поверхностей для тел, представляющих собой одиночные выпуклые многогранники. Он также используется для удаления нелицевых или задних граней из сцены перед применением одного из алгоритмов удаления невидимых линий. Этот способ часто называют отбрасыванием задних плоскостей. Для выпуклых многогранников число граней при этом сокращается примерно наполовину. Метод эквивалентен вычислению нормали к поверхности для каждого отдельного многоугольника. Отрицательность нормали к поверхности показывает, что нормаль направлена в сторону от наблюдателя и, следовательно, данный многоугольник не виден. Этот метод можно использовать также и для простой закраски. Интенсивность или цветовой оттенок многоугольника делается пропорциональным проекции нормали к поверхности на направление взгляда.

После определения нелицевых плоскостей остается найти нелицевые отрезки. Нелицевой отрезок образуется в результате пересечения пары нелицевых плоскостей.

После первого этапа удаления нелицевых отрезков необходимо выяснить, существуют ли такие отрезки, которые экранируются другими телами в картинке или в сцене. Для этого каждый оставшийся отрезок или ребро нужно сравнить с другими телами сцены или картинки. При этом использование приоритетной сортировки (z-сортировки) и простого минимаксного тестов позволяет удалить

целые группы или кластеры отрезков и тел. Например, если все тела в сцене упорядочены в некотором приоритетном списке, использующем значения z ближайших вершин для представления расстояния до наблюдателя, то никакое тело из этого списка, у которого ближайшая вершина находится дальше от наблюдателя, чем самая удаленная из концевых точек ребра, не может закрывать это ребро. Более того, ни одно из оставшихся тел, прямоугольная оболочка которого расположена полностью справа, слева, над или под ребром, не может экранировать это ребро. Использование этих приемов значительно сокращает число тел, с которыми нужно сравнивать каждый отрезок или ребро.

Алгоритм Робертса делится на три этапа. На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей. На втором этапе проверяется экранирование оставшихся в каждом теле ребер всеми другими телами с целью обнаружения их невидимых отрезков. На третьем этапе вычисляются отрезки, которые образуют новые ребра при протыкании телами друг друга. В данном алгоритме предполагается, что тела состоят из плоских полигональных граней, которые в свою очередь состоят из ребер, а ребра — из отдельных вершин. Все вершины, ребра и грани связаны с конкретным телом.



Алгоритм Художника

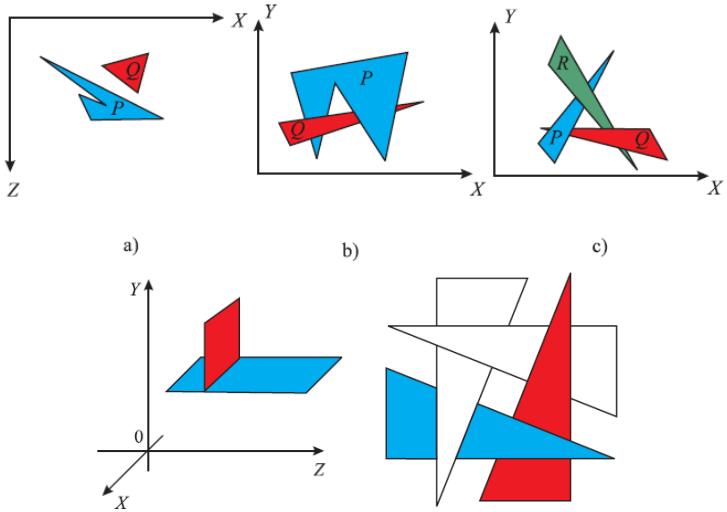
Суть данного алгоритма заключается в **сортировке** объектов по **глубине** относительно наблюдателя с последующей отрисовкой в порядке **убывания** глубины.

При этом не происходит разбиения объектов на составляющие (что увеличивает точность изображения).

Отрисованные в таком порядке объекты (**от дальнего к ближнему**) будут корректно экранировать друг друга.

Недостатки алгоритма

- Данный алгоритм не отбрасывает невидимые наблюдателю объекты, поэтому снижения затрат на отрисовку не происходит.
- При изменении положения наблюдателя необходимо производить сортировку объектов заново, даже, если геометрия статична.
- Сортируемые объекты лишь в исключительных случаях располагаются в параллельных друг другу плоскостях, поэтому сравнение глубин двух объектов является задачей с неочевидным решением.
- Данный алгоритм не способен справляться со случаями пересечения объектов и циклического наложения:



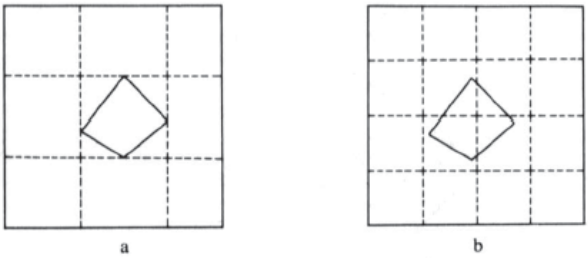
Алгоритм Варнока

Основные идеи, на которые опирается алгоритм Варнока, обладают большой общностью. Они основываются на гипотезе о способе обработки информации, содержащейся в сцене, глазом и мозгом человека. Эта гипотеза заключается в том, что тратится очень мало времени и усилий на обработку тех областей, которые содержат мало информации. Большая часть времени и труда затрачивается на области с высоким информационным содержимым. В качестве примера рассмотрим поверхность стола, на которой нет ничего, кроме вазы с фруктами. Для восприятия цвета, фактуры и других аналогичных характеристик всей поверхности стола много времени не нужно. Все внимание сосредоточивается на вазе с фруктами. В каком месте стола она расположена? Велика ли она? Из какого материала сделана: из дерева, керамики, пластика, стекла, металла? Каков цвет вазы: красный, синий, серебристый; тусклый или яркий и т. п.?

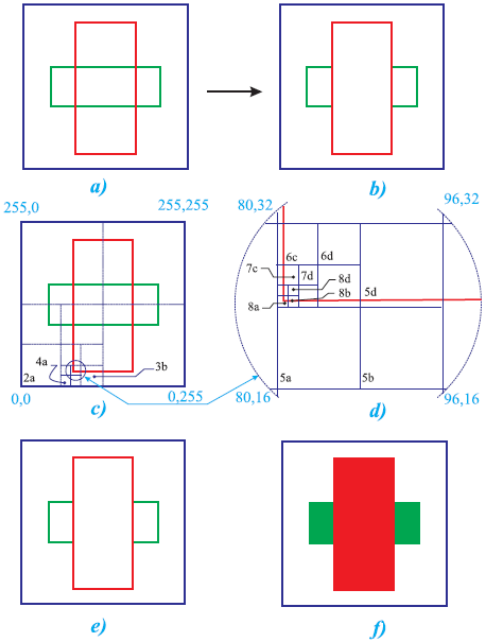
В каждом случае, по мере сужения сферы интереса, возрастает уровень требуемой детализации. Далее, если на определенном уровне детализации на конкретный вопрос нельзя ответить немедленно, то он откладывается на время для последующего рассмотрения. В алгоритме Варнока и его вариантах делается попытка извлечь преимущество из того факта, что большие области изображения однородны, например поверхность стола в приведенном выше примере. Такое свойство известно как когерентность, т.е. смежные области (пиксели) вдоль обеих осей x и y имеют тенденцию к однородности.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом. Устранение лестничного эффекта можно реализовать, доведя процесс разбиения до

размеров, меньших, чем разрешение экрана на один пиксель, и усредняя атрибуты подпикселей, чтобы определить атрибуты самих пикселей.



Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. В оригинальной версии алгоритма Варнока каждое окно разбивалось на четыре одинаковых подокна. Кэтмул применил основную идею метода разбиения к визуализации криволинейных поверхностей.



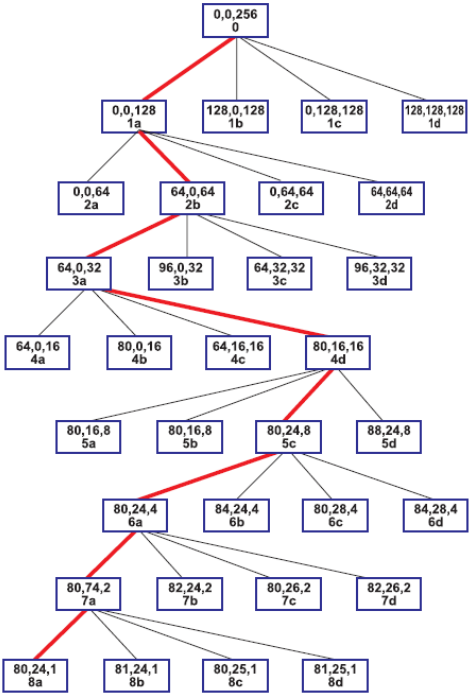
На рис. показан результат простейшей реализации алгоритма Варнока. Здесь окно, содержимое которого слишком сложно изображать, разбито на четыре одинаковых подокна. Окно, в котором что-то есть, подразбивается далее то тех пор, пока не будет достигнут предел разрешения экрана. На рис. 1, a показана сцена, состоящая из двух простых многоугольников. На рис. 1, b показан результат после удаления невидимых линий. Заметим, что горизонтальный прямоугольник частично экранирован вертикальным. На рис. 1, c и d показан процесс разбиения окон на экране с разрешением 256×256. Поскольку 256 = 28, требуются не более восьми шагов разбиения для достижения предела разрешения экрана.

Пусть подокна рассматриваются в следующем порядке: нижнее левое, нижнее правое, верхнее левое, верхнее правое. Будем обозначать подокна цифрой и буквой, цифра — это номер шага разбиения, а буква - номер квадранта. Тогда для окна 1 a подокна 2a, 4a, 4c, 5a, 5b оказываются пустыми и изображаются с фоновой интенсивностью в процессе разбиения. Первым подокном, содержимое которого не пусто на уровне пикселей, оказывается 8a. Теперь необходимо решить вопрос о том, какой алгоритм желательно применить: удаления невидимых линий или удаления невидимых поверхностей. Если желательно применить алгоритм удаления невидимых линий, то пиксель, соответствующий подокну 8a, активируется, поскольку через него проходят видимые ребра. В результате получается изображение видимых ребер многоугольников в виде последовательности точек размером с пиксель каждая (рис. 1, e).

Следующее рассмотрение окна, помеченного как 8d на рис. 1, d, лучше всего проиллюстрирует различие между реализациями алгоритмов удаления невидимых линий и поверхностей. В случае удаления невидимых линий окно 8d размером с пиксель не содержит ребер ни одного многоугольника сцены. Следовательно, оно объявляется пустым и изображается с фоновой интенсивностью или цветом. Для алгоритма удаления невидимых поверхностей проверяется охват этого окна каждым многоугольником сцены. Если такой охват обнаружен, то среди охватывающих пиксель многоугольников

выбирается ближайший к точке наблюдения на направлении наблюдения, проходящем через данный пиксель.

Проверка проводится относительно центра пикселя. Затем этот пиксель изображается с интенсивностью или цветом ближайшего многоугольника. Если охватывающие многоугольники не найдены, то окно размером с пиксель пусто. Поэтому оно изображается с фоновым цветом или интенсивностью. Окно 8d охвачено вертикальным прямоугольником. Поэтому оно изображается с цветом или интенсивностью этого многоугольника. Соответствующий результат показан на рис. 1, f.



Процесс подразбиения порождает для подокон структуру данных, являющуюся деревом, которое показано на рис. 2 (В *алгоритме Варнока впервые была реализована такая структура данных, как кватернарное дерево*). Корнем этого дерева является визуализируемое

окно. Каждый узел изображен прямоугольником, содержащим координаты левого нижнего угла и длину стороны подокна. Предположим, что подокна обрабатываются в следующем порядке: *abcd,* т. е. слева направо на каждом уровне разбиения в дереве. На рис. 2 показан активный путь по структуре данных дерева к окну 8а размером с пиксель.

Активные узлы на каждом уровне изображены толстой линией. Рассмотрение рис. 1 и 2 показывает, что на каждом уровне все окна слева от активного узла пусты. Поэтому они должны были визуализироваться ранее с фоновым значением цвета или интенсивности. Все окна справа от активного узла на каждом уровне будут обработаны позднее, т. е. будут объявлены пустыми или будут подразделены при обходе дерева в обратном порядке.

При помощи изложенного алгоритма можно удалить либо невидимые линии, либо невидимые поверхности. Однако простота критерия разбиения, а также негибкость способа разбиения приводят к тому, что количество подразбиений оказывается велико. Можно повысить эффективность этого алгоритма, усложнив способ и критерий разбиения, при этом окна могут быть неквадратными.

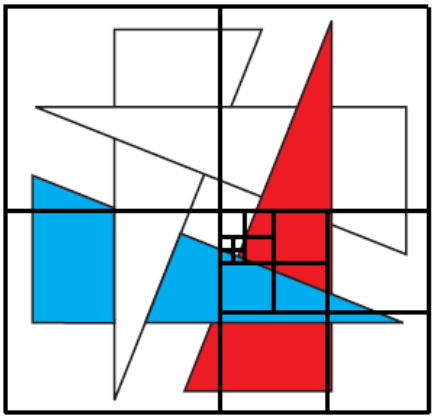
Для рассмотрения более сложных критериев разбиения будет полезно определить способы расположения многоугольников относительно окна. Один объект может располагаться относительно окна следующим образом:

- *внешним,* если он находится целиком вне окна;
- *внутренним,* если он находится целиком внутри окна;
- *пересекающим,* если он пересекает границу окна;
- *охватывающим,* если окно находится целиком внутри него.

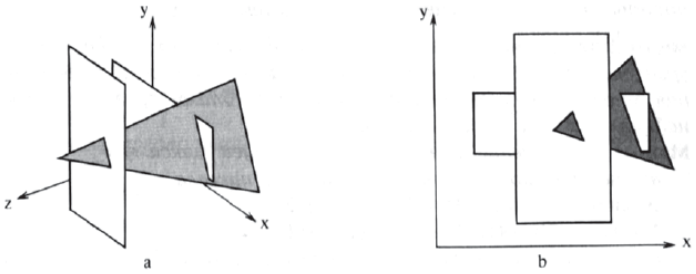
В случае одного объекта все перечисленные случае являются достаточно простыми для формирования изображения.

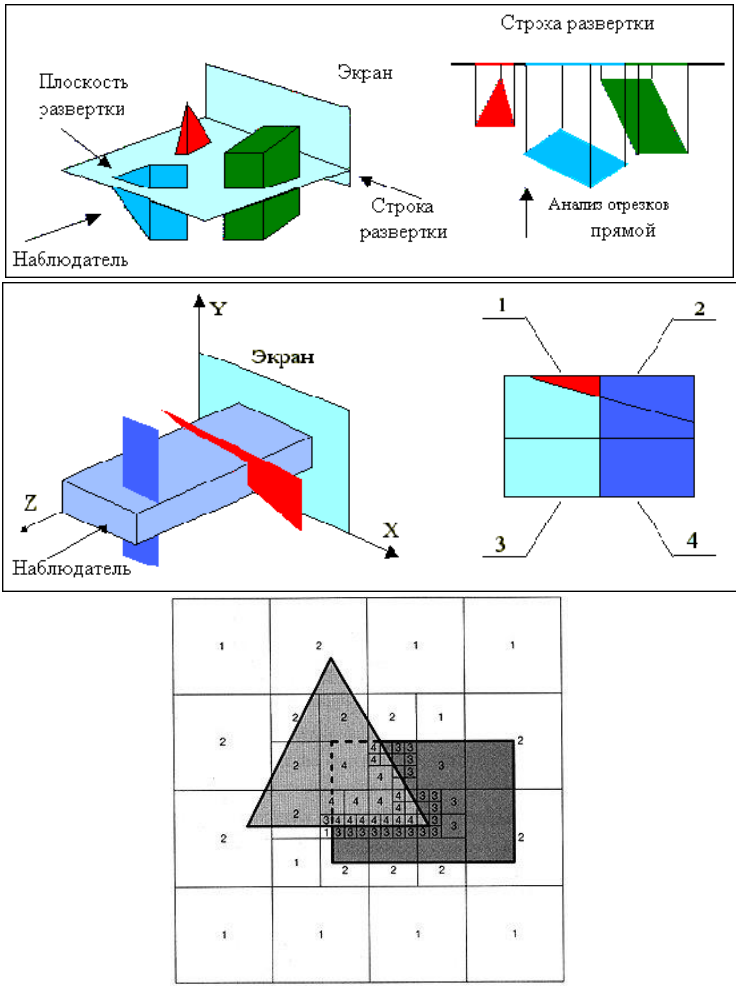
В случае наличия в окне нескольких объектов следует разбивать данное окно на более мелкие, пока:

– Найдется перекрывающий окно объект, глубина которого будет больше, чем у остальных.
– Окно перестанет содержать объекты.
– В окне останется только один объект.



При размерах окна в один пиксель, любой из входящих в него объектов можно считать перекрывающим, а ситуацию – не вызывающей затруднения.





Z-Буфер

Алгоритм предложен Эдом Кэтмулом и представляет собой обобщение буфера кадра. Обычный буфер кадра хранит коды цвета для каждого пиксела в пространстве изображения. Идея алгоритма состоит в том, чтобы для каждого пиксела дополнительно хранить еще и координату *Z* или глубину. При занесении очередного пиксела в буфер кадра значение его Z-координаты сравнивается с Z-координатой пиксела, который уже находится в буфере. Если Z-координата нового

пиксела меньше, чем координата старого, т.е. он ближе к наблюдателю, то атрибуты нового пиксела и его Z-координата заносятся в буфер, если нет, то ничего не делается.

Идея данного алгоритма заключается в том, что величина *Z* по соответствующей оси является показателем того, какая грань сцены находится ближе, а какая дальше.

Этот алгоритм наиболее простой из всех алгоритмов удаления невидимых поверхностей, но требует большого объема памяти. Данные о глубине для реалистичности изображения обычно достаточно иметь с разрядностью порядка 20 бит. В этом случае при изображении нормального телевизионного размера в 768×576 пикселов для хранения Z-координат необходим объем памяти порядка *1* Мбайта. Суммарный объем памяти при 3 байтах для значений RGB составит более 2,3 Мбайта.

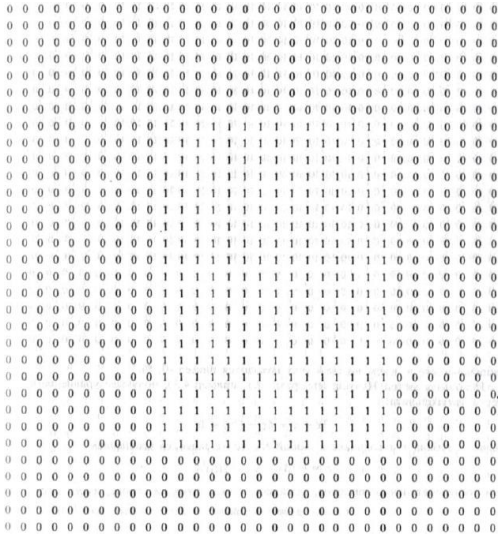
Время работы алгоритма не зависит от сложности сцены. Многоугольники, составляющие сцену, могут обрабатываться в произвольном порядке. Для сокращения затрат времени нелицевые многоугольники могут быть удалены.

Основной недостаток алгоритма с Z-буфером — дополнительные затраты памяти. Для их уменьшения можно разбивать изображение на несколько прямоугольников или полос. Понятно, что это приведет к увеличению времени, так как каждый прямоугольник будет обрабатываться столько раз, на сколько областей разбито пространство изображения.

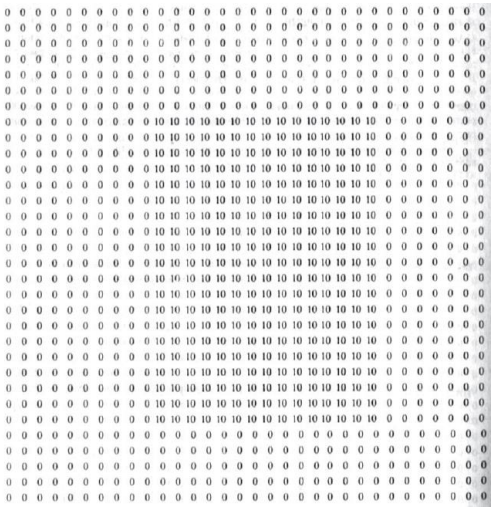
Другие недостатки алгоритма с Z-буфером заключаются в том, что пикселы в буфер заносятся в произвольном порядке и возникают трудности с реализацией эффектов прозрачности или просвечивания, устранением лестничного эффекта с использованием предфильтрации, когда каждый пиксел экрана трактуется как точка конечного размера и его атрибуты устанавливаются в зависимости от того какая часть пиксела изображения попадает в пиксел экрана. Но другой подход к устранению лестничного эффекта, основанный на постфильтрации—усреднении значений пиксела с использованием

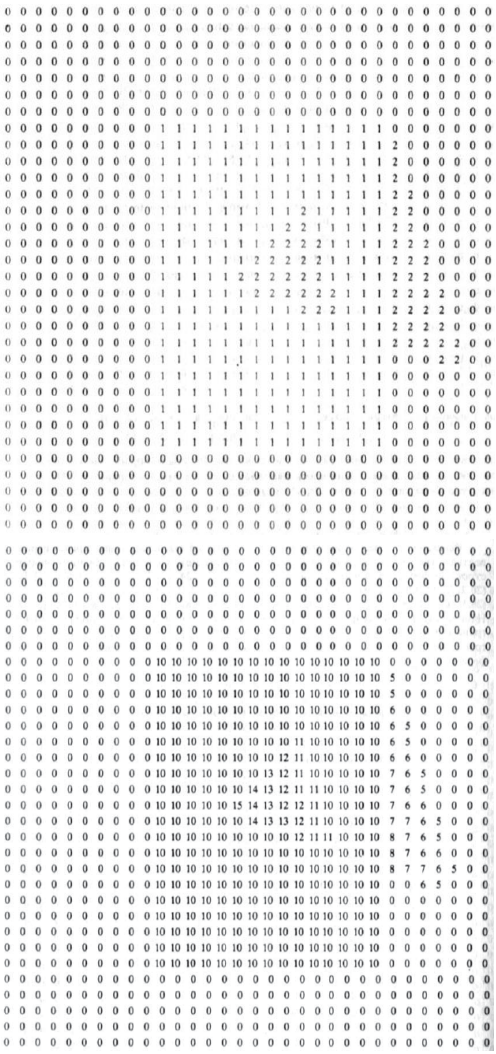
изображения с большим разрешением реализуется сравнительно просто за счет увеличения расхода памяти и времени.

Буфер кадра



Содержимое z-буфера таково:





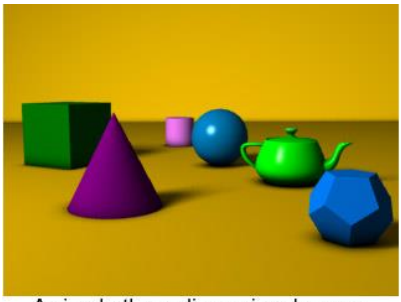
Одним из наиболее распространённых модификаций алгоритма Z-буфера является алгоритм иерархического Z-буфера. Назовём грань скрытой (невидимой) по отношению к Z-буферу, если для всех её точек их глубины не меньше соответствующих значений в Z-буфере (при попытке вывести такую грань в Z-буфер ничего не изменится). Куб (прямоугольный параллелепипед) назовём скрытым по отношению к

z-буферу, если все его лицевые грани являются скрытыми. Теперь объединим несколько граней сцены и обведём вокруг них куб. Для вывода этих граней в Z-буфер проверяется, является ли этот куб скрытым по отношению к Z-буферу. Если это так, то и все грани, объединённые кубом, являются скрытыми по отношению к Z-буферу и не выводятся в него, сокращая количество вычислений и ненужных операций сравнения. Если же это не так, то всё равно не все грани, объединённые кубом, являются видимыми и этот куб разбивается на части, а затем для каждой части выполняются такие же операции сравнения. Таким образом реализация данного алгоритма может выглядеть следующим образом. Вокруг всей сцены описывается куб, он разбивается на 8 частей (тоже кубы), а каждая из частей, в свою очередь, разбивается ещё на 8 частей и так до тех пор, пока количество граней в кубе не станет меньше заданного числа, при котором уже нет смысла в дальнейшей разбивке на части. Далее для вывода очередного куба в Z-буфер для него проверяется, является он скрытым или нет. Если да, то все грани, объединённые этим кубом игнорируются, а если нет, то соответствующая проверка выполняется для всех его частей и т.д.

Для ещё большей оптимизации вывода в Z-буфер и для снижения количества вычислений используется тот факт, что чем раньше видимая грань будет выведена в Z-буфер, тем больше невидимых граней, закрываемых ею, будет отвергнуто. Для этого строится список тех граней, которые были выведены в предыдущем кадре. В последующем кадре эти грани выводятся в Z-буфер самыми первыми, так как почти наверняка они будут оставаться видимыми и в этом кадре.

Следует отметить, что алгоритм Z-буфера очень удобен для аппаратной реализации.

Многие алгоритмы удаления невидимых линий и поверхностей создавались не для аппаратной, а для программной реализации.



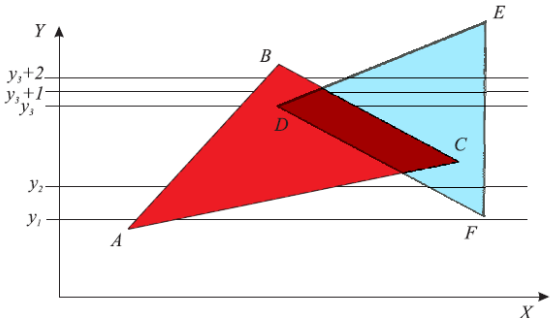
A simple three dimensional scene

Z-buffer representation

Построчное сканирование

Алгоритм построчного сканирования является попыткой решить проблему избыточности z-буфера.

Данный алгоритм строится на понятии сканирующей строки (которая может определять глубину произвольного пикселя внутри треугольника) и метода списка активных ребер (CAP) растеризации треугольника.



Растеризация нескольких треугольников может проводится одновременно полосами вдоль сканирующей строки.

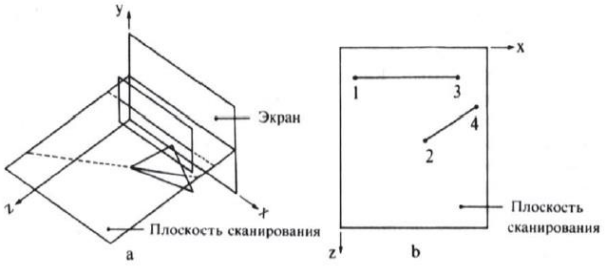
Сканирующая строка при пересечении с треугольниками будет создавать отрезки трех типов:

– пустые отрезки
– отрезки с одной фигурой
– отрезки с несколькими фигурами

В последнем случае для определения того, какая фигура внесет свой вклад в данный отрезок, используют тест глубины.

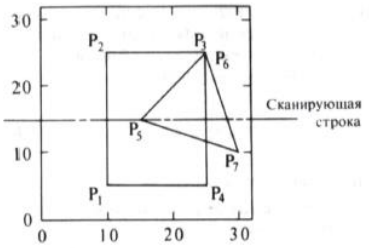
Данный алгоритм более оптимален с точки зрения расходов памяти, однако современные масштабы детализации сцены делают его крайне медленным по производительности.

В настоящий момент сканирующие строки используются в GPU для ускоренной растеризации треугольников. Так же, для достижения определенной доли оптимизации алгоритм построчного сканирования часто объединяют с z-буфером.



Сканирующая плоскость.

Алгоритм работает в пространстве изображения с окном высотой в одну строку и шириной в экран, тем самым трехмерная задача сводится к двумерной.



Буфер кадра для строки

0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0

z-буфер для строки

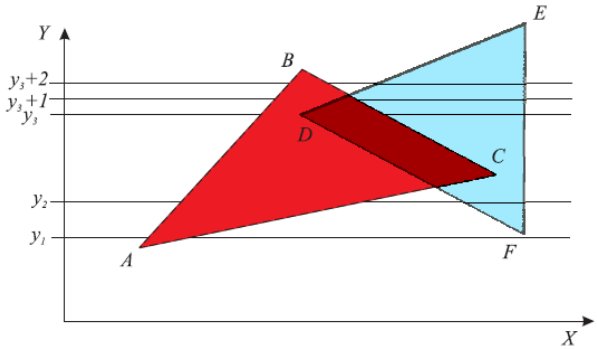
0 0 0 0 0 0 0 0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0 0 0 0 0

Буфер кадра для строки

0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 2 2 0 0 0 0 0

z-буфер для строки

0 0 0 0 0 0 0 0 0 0 10 10 10 10 15 14 13 12 12 11 10 10 10 10 10 6 6 0 0 0 0 0



К тому времени, когда алгоритм дойдет до сканирующей строки y=y1, список активных ребер будет содержать AB и AC в указанном порядке. Ребра рассматриваются в направлении слева направо. Приступая к обработке AB, инвертируем, прежде всего, флаг треугольника ABC. Тогда флаг примет значение истины и, следовательно, мы окажемся «внутри» треугольника, который необходимо рассмотреть. Теперь, поскольку мы находимся внутри

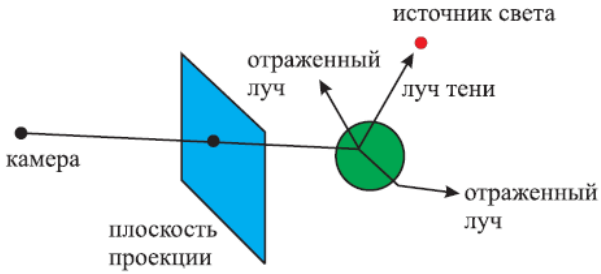
только одного треугольника, последний должен быть видимым и, следовательно, на интервале от ребра AB до следующего ребра AC в списке активных ребер будет вестись закраска, требуемая для текущего треугольника. При прохождении ребра AC флаг треугольника меняется на противоположное значение, то есть теперь мы не находимся внутри ни одного треугольника. Поскольку AC является последним ребром в списке активных ребер, обработка сканирующей строки завершается.

Когда встретится сканирующая строка y=y2, в отсортированном списке активных ребер будут находить¬ся AB, AC, FD и FE. Со сканирующей строкой пересекаются два треугольника, но мы в каждый момент будем находиться «внутри» только одного из них.

Наиболее интересна ситуация для строки y=y3. При входе в ABC флаг треугольника устанавливается равным истине. На интервале от точки входа до следующего ребра DE ведется закраска, соответствующая треугольнику ABC. В точке пересечения с DF флаг DEF также устанавливается в истину, то есть мы будем находится внутри обоих треугольников. Теперь требуется решить, какой из треугольников расположен ближе к наблюдателю. Определение производится путем вычисления координаты z, соответствующей каждому треугольнику. Правило закраски будет действовать для треугольника z-координата которого меньше. Так проводится сканирование каждой строки.

Обратная трассировка луча

Наиболее непроизводительный алгоритм HSR. Применяется по большей части в CG по причине его полезных свойств с точки зрения создания реалистичных изображений.

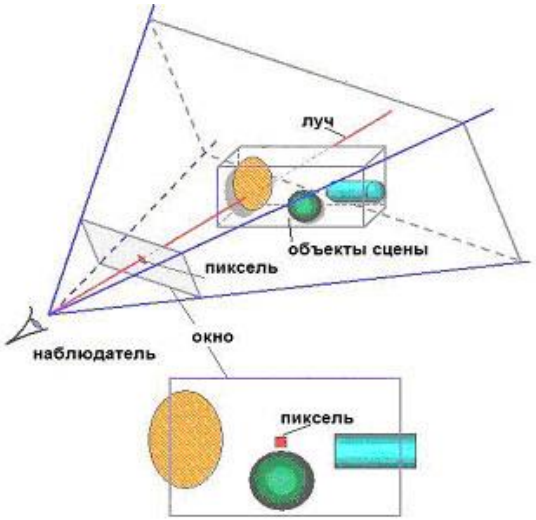


Основная идея заключается в том, что из каждой ячейки буфера экрана в пространство испускается проектор. Данный проектор может пересечь произвольное количество фигур, при этом подсчитывая конечные световые характеристики. Проходя чрез различные материалы поверхностей данный луч может отражаться и преломляться средой, что делает его наиболее подходящим для нужд CG и абсолютно неприемлемым для HSR.

При рассмотрении этого алгоритма предполагается, что наблюдатель находится на положительной полуоси Z, а экран дисплея перпендикулярен оси *Z* и располагается между объектом и наблюдателем.

Удаление невидимых (скрытых) поверхностей в алгоритме трассировки лучей выполняется следующим образом:

1. Для каждой точки на проектируемой плоскости строится луч, начало которого совпадает с положением наблюдателя, а угол луча по отношению к направлению наблюдения определяется характеристиками наблюдателя (максимальный угол зрения относительно горизонтальной и вертикальной плоскостей) и точкой плоскости проецирования, для которой строится луч.
2. После построения луча определяется его пересечение со всеми гранями, из которых состоит сцена и выбирается та грань, расстояние от которой до наблюдателя минимально.
3. В данной точке плоскости проецирования рисуется пиксель цветом, определяемым в точке пересечения луча и выбранной грани.



Каждый раз, когда луч пересекает некоторую поверхность в некоторой точке, из этой точки испускаются дополнительные лучи. Если поверхность отражающая, то генерируется отраженный луч. Если поверхность пропускает свет, то генерируется преломленный луч. Некоторые поверхности одновременно и отражают, и пропускают лучи, и тогда испускаются оба вида лучей. Пути этих лучей отслеживаются по всей модели, и если лучи пересекают другие поверхности, то снова испускаются лучи. В каждой точке, где луч пересекает поверхность, рисуется луч тени из точки пересечения к каждому источнику света. Если этот луч пересекает другую поверхность перед тем, как достигнуть источника света, то на ту поверхность, с которой был послан луч, падает тень с поверхности, блокирующей свет.

Таким образом, при рекурсивной трассировке лучей один луч начинает отражаться, преломляться, снова отражаться и т.д. Для того, чтобы ограничить глубину рекурсии, основываются на некотором здравом смысле, заключающемся в том, что когда луч «уходит» слишком далеко, он ослабляется. Например, можно взять какое-то

число уровней рекурсии и тогда, если дерево добирается до данного уровня, оно перестаёт дальше развиваться.

Иногда применяют также правило «русской рулетки»: в какой-то момент мы с некоторой вероятностью решаем вопрос о том, убить луч или позволить ему продолжаться дальше (обязательно найдётся случай, когда этот луч прекратит своё существование).

Отмечается, что в рассмотренных нами алгоритмах около 75%-95% времени выполнения задачи затрачивается на вычисление пересечений луча с объектами (треугольниками, сферами, цилиндрами, конусами, плоскостями и т.д.). Поэтому возникает необходимость ускорения решения поставленной задачи.

BSP – Binary Space Partition

Алгоритм двоичного разбиения пространства является продолжателем идей алгоритма художника, обладавшего явными недостатками, делавшими его неприменимым для нужд графики.

Изначально назначением алгоритма являлась сортировка полигонов в пространстве. Однако с появлением аппаратного z-буфера, данная область применения была признана неперспективной.

В настоящий момент BSP-деревья используются для реализации пространственной иерархии объектов, а также их локализации.

Впервые BSP-деревья были применены компанией idSoftware в компьютерном экшене Quake, совершившем революцию в мире 3d-графики.

Сейчас BSP-деревья де-факто являются стандартом для реализации графических движков на базе закрытых помещений. Кроме того, с их помощью в сотни раз можно оптимизировать множество других алгоритмов.

BSP-деревья существуют в пространстве, состоящем из полигонов и разбитого некой плоскостью на два полупространства.

Каждое из них так же разбивается пополам некой плоскостью до тех пор, пока не будет найдено пространство полигонов, образующих некую выпуклую оболочку.

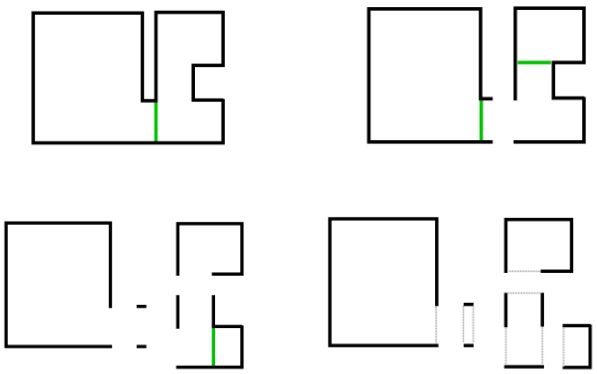
В этом случае дальнейшее разбиение прекращается.

Такое разбиение изначального пространства в конечном итоге описывается бинарным деревом. Оно и получило название BSP.

При создании BSP-дерева очень важно, чтобы оно было сбалансированным, т.е. каждая плоскость разбивала бы множество полигонов на два равных по количеству подмножества.

Зачастую бывает трудно найти такую плоскость, которая не разбивала бы хотя бы один полигон. В таком случае, полигон превращается в два, распределяемые по соответствующим полупространствам.

Задача о выборе бьющей плоскости в общем случае может быть нетривиальной, поэтому в качестве таковой обычно выбирается плоскость одного из присутствующих в пространстве полигонов.



При отрисовке сцены необходимо спуститься с корня BSP-дерева к листьям.

Для каждого узла поочередно отобразить оба поддерева, причем поддерево, в котором находится наблюдатель отобразить вторым.

Определить, в каком поддереве находится наблюдатель можно через положение точки относительно плоскости.

Особое свойство выпуклых оболочек, которыми фактически и являются листья дерева, заключается в том, что полигоны, образующие оболочку, при отрисовке никогда не экранируют друг друга – значит, могут быть отрисованы в любом порядке.

Дополнительное свойство:

Если листья BSP-дерева (выпуклые оболочки) принять за некоторые ячейки пространства, то любой объект, чье положение относительно некой плоскости может быть определено, может быть локализован внутри одной, либо нескольких ячеек.

Таким образом, BSP-дерево можно использовать для представления иерархии пространства.

Недостатки:

BSP-дерево является статичным объектом. Если хотя бы один из полигонов изменит свое положение в пространстве, то это потребует полного пересчета структуры, что приведет к немалым затратам.

Несмотря на то, что BSP-деревья решают задачу корректного экранирования объектов, они, как и алгоритм художника не сокращают множество отрисовываемых объектов до необходимого.