LOBACHEVSKY STATE UNIVERSITY
of NIZHNI NOVGOROD
National Research University

Computing Mathematics and Cybernetics faculty

Software department

CG 255. Computer Graphics

Introduction Course

Удаление невидимых поверхностей. Оптимизация. Тени

В.Е. Турлапов, проф. каф. МО ЭВМ
vadim.Turlapov@cs.vmk.unn.ru

Модели трехмерных объектов

Аналитическая модель. Линейно-узловая (полигональная) модель. Стрипы и фэны
Равномерная сетка. Неравномерная сетка (TIN). Изолинии. Воксельная модель.

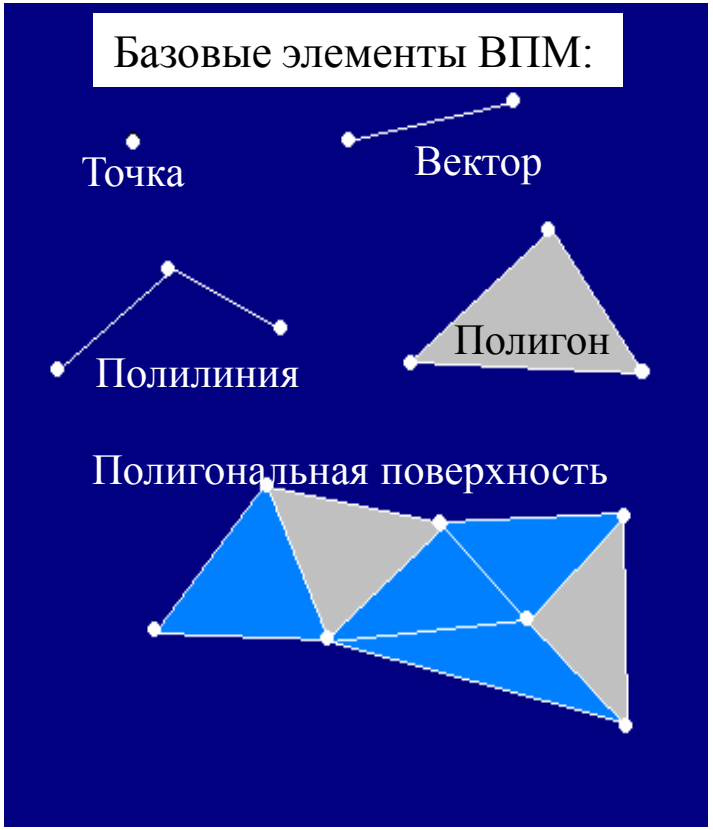
Аналитическая. Параметрическая форма.

Сфера: $x = R \cdot \sin s \cdot \cos t, y = R \cdot \sin s \cdot \sin t, z = R \cdot \cos s$

Поверхность Безье:

$$P(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} C_m^i s^i (1-s)^{m-i} \cdot C_n^j t^j (1-t)^{n-j} \cdot P_{ij}$$

$$C_m^i = \frac{m!}{i!(m-i)!}, \qquad 0 \le s,t \le 1$$



СТРУКТУРЫ ДАННЫХ. Линейно-узловая модель

Способ 1: все грани отдельно и для каждой хранятся координаты вершин. Объем памяти: $M1=N_{Fase} \cdot n_{Vert} \cdot 3 \cdot Byte_{Coor}$

Способ 2: вершины пронумерованы, хранятся в отдельной таблице; каждая грань задана списком индексов вершин. Объем памяти: $M2=n_{Vert} \cdot 3 \cdot Byte_{Coor} + N_{Fase} \cdot N_{FaceVert} \cdot Byte_{Index}$

Способ 3: линейно-узловая модель (иерархия: вершины, ребра, грани). Обьем памяти: $M3=n_{Vert} \cdot 3 \cdot Byte_{Coor} + k_{Edge} \cdot 2 \cdot Byte_{VertIndex} + N_{Fase} \cdot N_{FaceVert} \cdot Byte_{EdgeIndex}$

Св-ва: 1 избыточна; 1и2 дважды рисуют ребра; 2и3 сохраняют топологию.

Модели трехмерных объектов

Пусть в модели куба для координат отведено 8 байтов (тип с плавающей точкой double), а для индексов – по 4 байта. Тогда объем памяти для 3 рассмотренных структур составит:
M1 = 6*4*3*8B = 576 B,
M2 = 8*3*8B + 6*4*4B = 288 B,
M3 = 8*3*8B + 12*2*4B + 6*4*4B = 384 B.

Стрипы и фэны

Strip – полоса, лента из треугольников или квадратов; Fan – веер из треугольников.

Объем памяти для треугольников: $M_{Triang} = n_{Vert} \cdot 3 \cdot Byte_{Coor} + (n_{Vert} - 2) \cdot Byte_{Index}$

Объем памяти для квадратов: $M_{Quad} = n_{Vert} \cdot 3 \cdot Byte_{Coor} + (n_{Vert} - 2)/2 \cdot Byte_{Index}$

Режимы OpenGL: glBegin(GL_TRIANGLE_STRIP), а также GL_QUAD_STRIP; GL_TRIANGLE_FAN.

Равномерная сетка Не надо хранить x,y. Для описания сложных поверхностей необходимо большое количество узлов, которое может быть ограничено объемом памяти компьютера

Неравномерная сетка. (TIN-Triangulated Irregular Network)

Изолинии. Преобразование изолиний в полигональную модель выполняется методами триангуляции

Воксельная модель. Voxel-Volume Element

Удаление невидимых поверхностей. Общие оптимизационные подходы

Анализ видимости одна из наиболее сложных задач компьютерной графики.

Два класса методов анализа видимости объектов:

1 класс - методы, работающие в пространстве 3D объектов;

2 класс – методы, работающие в пространстве картинной плоскости (point-sampling methods).

Методы 1 класса более точные, т.к. работают с числами с плавающей точкой, но трудоемкость $O(n^2)$, где n – число объектов (граней). Методы 2 класса более быстрые: трудоемкость ➔ $O(C \cdot n)$, где C – число пикселов в окне, но дают ошибки дискретизации (aliasing artifacts). ➔ Существует большое количество смешанных методов.

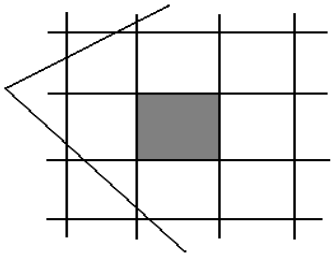
Общие подходы к созданию эффективных алгоритмов удаления невидимых поверхностей

1) Использование методов сортировки (явно или неявно)
2) Использование деревьев (двоичных, четверичных, восьмеричных)
3) Использования методов оптимизации
4) Использование когерентности (от английского coherence - связность)

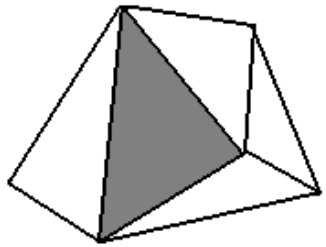
Когерентность

Три типа когерентности

1: Когерентность в картинной плоскости – если данный пиксел принадлежит грани P, то соседние пикселы скорее всего тоже ➔



2: Когерентность в пространстве объектов – если данный объект (грань) видим (невидим), то соседние объекты (грани) скорее всего тоже ➔

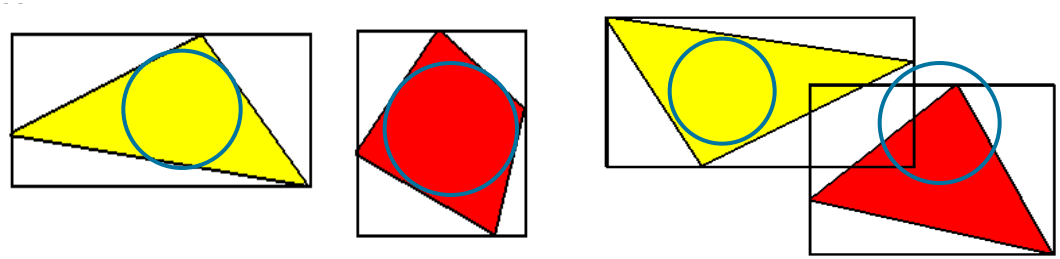


3: Временная когерентность (используется в анимации) – объект (грань), видимый в данном кадре, скорее всего является видимым и в следующем.

Основные методы оптимизации

1.Метод отсечения нелицевых граней (culling)
Позволяет примерно вдвое сократить количество рассматриваемых граней.
Для определения того, является заданная грань лицевой или нет достаточно взять произвольную точку этой грани и проверить выполнение условия (N,L)<=0, где N - нормаль к грани, L – направление проецирования. В OpenGL отбраковка нелицевых граней: glCullFace ().

2.Метод оболочек (Bounding Volumes, Bounding-Volume-Hierarchy (BVH))
Если оболочки не пересекаются, то содержащиеся в них объекты тоже пересекаться не будут. Однако, если оболочки пересекаются, то сами объекты пересекаться не обязаны. В качестве ограничивающих тел чаще всего используются прямоугольные ограничивающие параллелепипеды ➔ Оболочка описывается числами (Xmin,Ymin,Zmin) и (Xmax,Ymax,Zmax) из координат точек исходного объекта (4 для 2D).

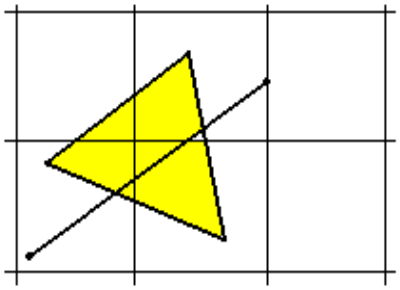


Основные методы оптимизации

3.Разбиение пространства (картинной плоскости)

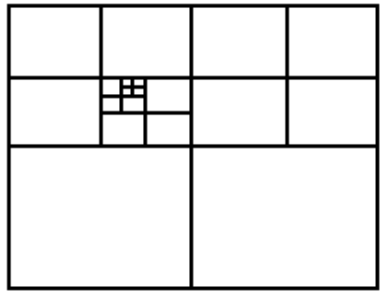
Еще один метод, облегчающий сравнение объектов, позволяющий использовать когерентность как в пространстве, так и в картинной плоскости.

С этой целью разбиения стоятся уже на этапе пре-процессирования, и для каждой клетки разбиения составляется список всех объектов (граней), которые ее пересекают.



Простейшим вариантом разбиения является равномерное разбиение пространства на набор равных прямоугольных клеток ➔

Составляется список объектов, пересекающих клетку разбиения. Для отыскания всех объектов, которые закрывают рассматриваемый объект при проецировании, проверяются только объекты, попадающие в те же клетки разбиения картинной плоскости.



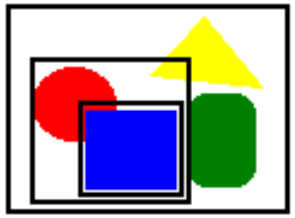
Для сцен с неравномерным распределением объектов имеет смысл использовать неравномерное (адаптивное) разбиение пространства или плоскости. ➔

Основные методы оптимизации

4. Иерархические древовидные структуры

При работе с большими объемами используются различные древовидные (иерархические) структуры. Стандартными формами таких структур являются восьмеричные (Octrees, для 3D), тетрарные (Quad-trees, для 2D), бинарные или BSP-деревья (Binary Space Partitioning Trees) и деревья ограничивающих тел. Иерархии позволяют упорядочивать грани объектов, производить быстрое и эффективное отсечение граней, не удовлетворяющих каким-либо из условий.

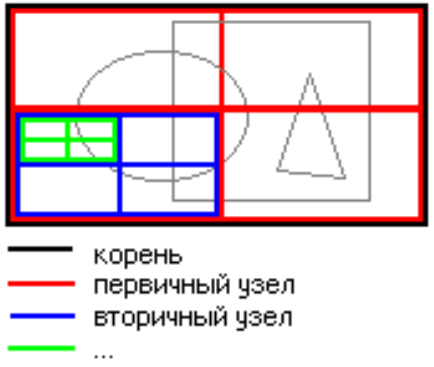
4.1. Иерархия ограничивающих тел ➔



Получается дерево, корнем которого является тело, описанное вокруг всей сцены, а потомками – тела, описанные вокруг первичных, вторичных и др. групп.

Отсечение основного количества объектов происходит уже на ранней стадии достаточно быстро, ценой всего лишь нескольких проверок.

4.2. Иерархии разбиения ➔



Каждая клетка исходного разбиения разбивается на части (которые, в свою очередь, так же могут быть разбиты и т.д. При этом каждая клетка разбиения соответствует узлу дерева).
Иерархии (как и разбиения) позволяют достаточно легко и просто производить частичное упорядочение граней. В результате получается список граней, практически полностью упорядоченный,
что дает возможность применять специальные методы сортировки.

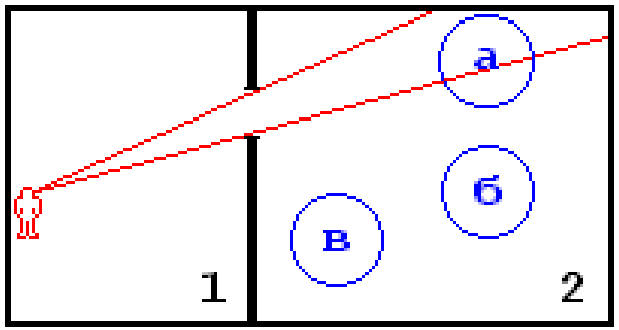
Специальные методы оптимизации

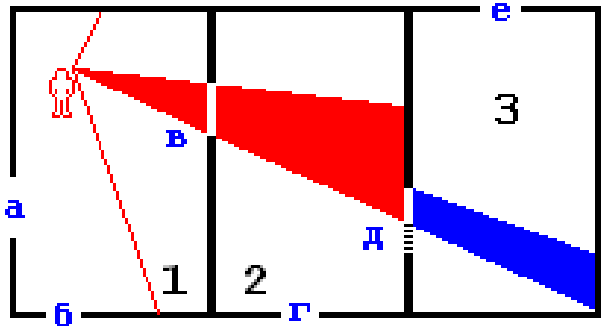
1.Потенциально видимые множества граней (препроцессинг построения PVS)

2.Метод порталов (PVS "на ходу")

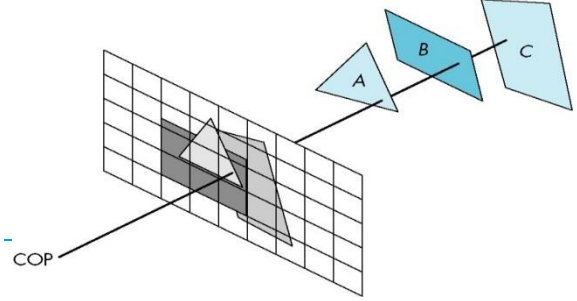
3.Метод иерархических подсцен (модификация метода порталов)

Дополнительный материал по методу порталов (local):
Использование порталов в indoor-сценах
(http://www.gamedev.ru/articles/?id=30118)





Удаление невидимых линий и поверхностей



После того, как вершины прошли все этапы геометрических преобразований и процедуру отсечения, «на конвейере» остались только геометрические объекты, которые потенциально могут попасть в формируемое изображение. Но перед тем, как приступать к их преобразованию в растр, нужно решить еще одну задачу – удалить объекты, перекрываемые с точки зрения наблюдателя другими объектами.

Удаление невидимых линий

Алгоритм Робертса (в пространстве объектов)

Первый из алгоритмов удаления невидимых линий. Требует, чтобы каждая грань была выпуклым многоугольником:

шаг 1: Отбросить ребра, обе инцидентных грани которых являются нелицевыми;

шаг 2: Проверка закрывания каждого оставшегося ребра лицевыми гранями

2.1. грань ребра не закрывает;

2.2. грань полностью закрывает ребро;->раннее отсечение

2.3. частично закрывает – ребро разбивается на части и оставляются только видимые части, ≤ 2

Количественная невидимость. Алгоритм Аппеля

Ввел количественную невидимость (quontative invisibility) точки как число граней ее закрывающих.

Контурная линия состоит из ребер, для которых одна из инцидентных граней – лицевая, другая – нет. Контурные линии разбивают поверхность на фрагменты с постоянной количественной невидимостью. Пример.

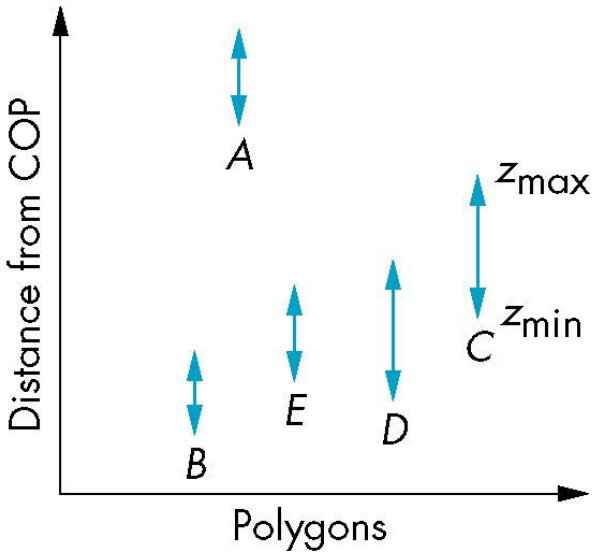
Метод плавающего горизонта (для Wire Frame)

Видно то, что выше верхнего и ниже нижнего горизонта. Порядок вывода: от ближних к дальним.

Удаление невидимых поверхностей

Алгоритмы упорядочения
Метод сортировки по глубине
Алгоритм художника

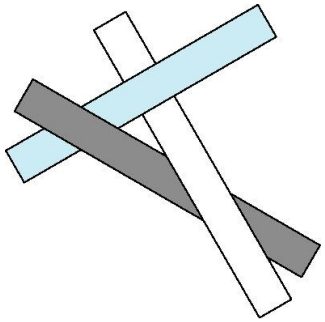
Метод *сортировки по глубине* (depth sort) реализует подход к решению проблемы удаления невидимых поверхностей, основанный на анализе *пространства объектов*. Метод сортировки по глубине является вариантом еще более простого алгоритма, получившего название *алгоритма художника* (painter's algorithm). Используется правило - отображение многоугольников от дальнего к ближнему (back-to-front rendering) подобно методике художников.



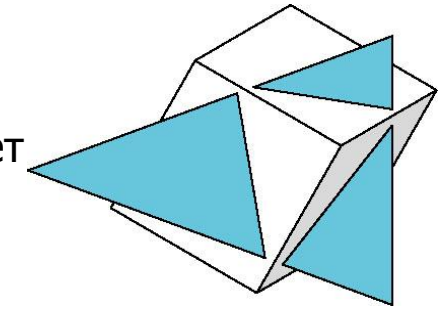
Выполняется сортировка по z для оболочек. Если z-оболочки двух многоугольников перекрываются, то порядок формирования их образов выявляется сравнением каждой пары таких объектов. Используется множество довольно сложных проверок.
Для пары многоугольников, z-оболочки которых перекрываются:
- первой выполняется самая простая проверка – сравнение x- и y-оболочек. Если либо x- либо y-оболочки не перекрываются, то многоугольники не закрываю друг друга. Сравнивать x- и y-оболочки можно только при параллельном проецировании, что говорит в пользу предварительной перспективной нормализации.
- в случае, если x- и y-оболочки перекрываются, можно отыскать порядок формирования их образов, который позволит корректно воспроизвести сцену на экране.

Метод сортировки по глубине. Алгоритм художника

Два случая, с которыми справиться особенно трудно



Во-первых, три многоугольника могут перекрывать друг друга *циклически*.➔
В этом случае вообще не существует такого порядка формирования образа многоугольника, который позволил бы получить корректное изображение. В этом случае делят один из двух перекрывающихся на две части и повторяют анализ образовавшегося набора многоугольников.



Во-вторых, возможна ситуация, когда один многоугольник пронзает другой (или другие). ➔
В такой ситуации придется детально проанализировать пересечение, используя методы отсечения одного многоугольника общего вида другим многоугольником. Если пересекающиеся многоугольники имеют много вершин, можно попытаться использовать какие-либо искусственные методы, учитывающие специфику ситуации и требующие меньшего объема вычислений.

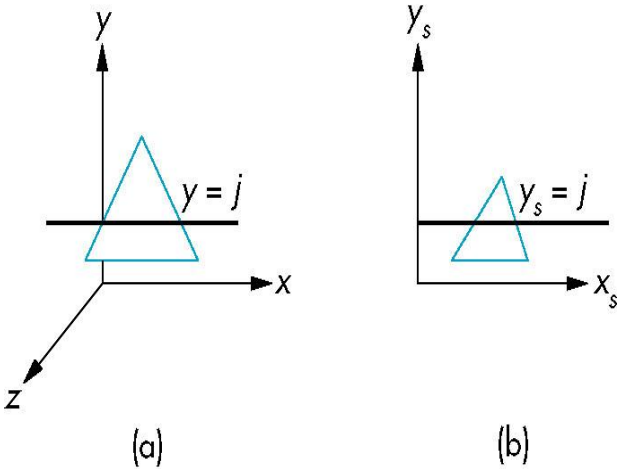
Удаление невидимых поверхностей. Метод Z-буфера

Метод иерархичекого Z-буфера – модификация, использующая все три типа когерентности.

Если многоугольник не отбрасывается в результате отсечения, он передается в модуль закрашивания и удаления невидимых поверхностей. На этой стадии выполнена нормализация проецирования, сохранена информация о глубине. Тщательно продуманный алгоритм **z**-буфера позволяет решить одновременно три задачи : выполнить окончательное ортогональное проецирование, удалить невидимые поверхности и тонировать образ многоугольника.
Метод основан на использовании буферного массива, в котором сохраняется Z-координата для каждого пиксела растра. Сначала буфер заполняется значениями максимальной глубины. Порядок вывода объектов (граней) безразличен. Очередная грань рисуется в данном пикселе, если глубина этой грани меньше записанной в буфере. После чего значение Z для данного пиксела обновляется в буфере значением Z отрисованной точки грани.

Основной процесс алгоритма буфера глубины (псевдокод)

```
for(each face F) //для каждой грани F
  for (each pixel(x,y) covering the face)
  //для каждого пиксела (x,y) покрывающего грань
    {depth = depth of F at (x,y);//глубина F в (x,y)
     if (depth < d[x][y])      //если F ближайшая грань
     {C = color of F at (x,y); //цвет грани F в т.(x,y)
      //set the pixel color at (x,y) to  C
      //устанавливаем цвет пиксела в (x,y) равным C
      d[x][y] = depth;    //обновляем  буфер глубины}}
```



Удаление невидимых поверхностей
Метод Z-буфера. Алгоритм в приращениях

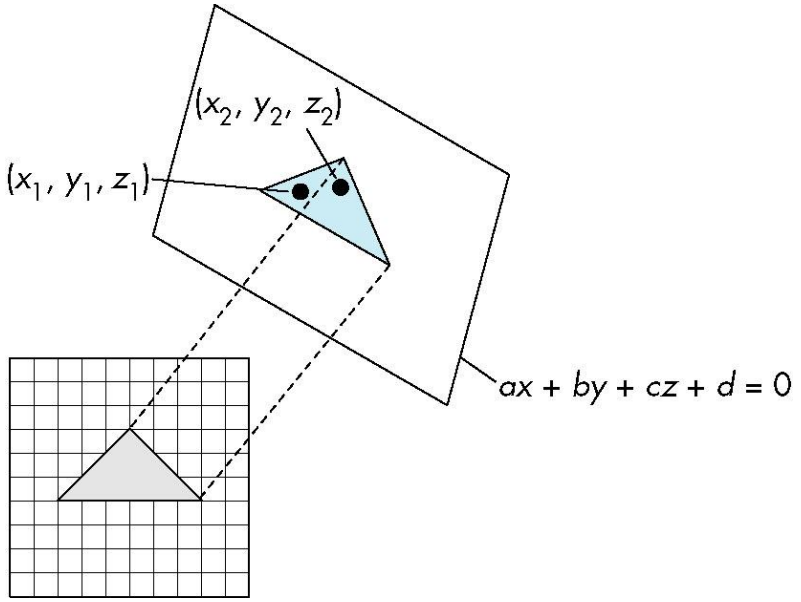
При растеризации многоугольника его, расположенные на плоскости $ax+by+cz+d = 0$, где (a,b,c) – компоненты нормали, обрабатываются последовательно, строка за строкой. Обозначив $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$, $\Delta z = z_2 - z_1$, уравнение плоскости можно записать в приращениях $\mathbf{a\Delta x+b\Delta y+c\Delta z = 0}$.

При движении по строке $y=const$, то $\Delta y=0$, а $\Delta x=1$. Таким образом, при переходе от одного пикселя к другому на той же самой строке растра выполняется соотношение

$$\mathbf{\Delta z = -(c/a)\ \Delta x = -(c/a)}$$

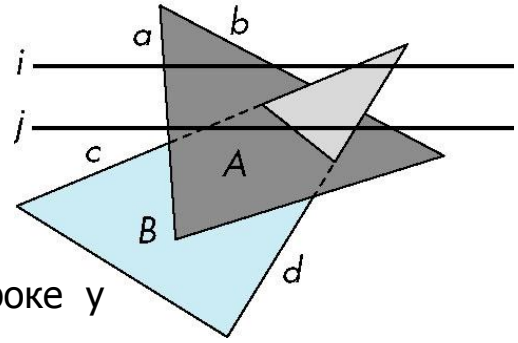
Это значение нужно вычислить только один раз при обработке определенного многоугольника и далее использовать, пока обработка этого многоугольника не завершится.

Минусом алгоритма Z-буфера является то, что значительное количество времени «растрачивается» на визуализацию тех поверхностей, которые позднее перекрывается другими поверхностями.



Удаление невидимых поверхностей. Метод построчного сканирования

С точки зрения экономии памяти метод построчного сканирования считается наиболее эффективным алгоритмом удаления невидимых поверхностей, т.к. сочетает удаление невидимых поверхностей с растеризацией (закраской) поверхности.



Алгоритм строки развертки:

```
for ( each  scan line, y )  // для каждой строки развертки
    for ( each pixel , x , on scan line y ) //для каждого пикселя на строке  y
    {find  the  closest  face  that "covers" the pixel;
     //находим ближайшую грань , "перекрывающую"  данный пиксел
     c = color at (x,y) of the closest face ; //цвет ближайшей грани в точке (x,y)
     set  the  pixel  at (x,y) to color c  //устанавить  цвет пикселя в точке (x,y) в  c }
```

На первый взгляд кажется, что алгоритм напоминает алгоритм z-буфера, но есть одно принципиальное отличие: **алгоритм построчного сканирования при обработке каждого пикселя должен просматривать все многоугольники**, по крайней мере, **претендующие на отображении в данной зоне экрана**, тогда как алгоритм z-буфера в каждый момент времени имеет дело только с одним многоугольником. ➔ Создается массив указателей, по одному на каждую строку, каждый из которых указывает на массив указателей ребер многогранников, пересекаемых данной строкой.

При исполнении алгоритма в результате проверяются только те полигоны, которые претендуют на отображение в текущей строке.

Фотореалистическая визуализация. Построение теней

Основные подходы:

1. Проецирование модели "на землю" и в текстуру
2. Использование буфера теней
3. Теневые объёмы (на основе трассировки)

Тени как текстура

Применяется для плоских принимающих поверхностей и точечных источников. Полигон тени – суперпозиция полигонов теней от каждой грани. Рисуется принимающая поверхность, затем цветом ambient – тень; последним рисуется объект, т.к. он может перекрывать тень.

Формула для вычисления вершин V' полигонов тени по вершинам V объекта:

$$V' = S + (V - S)\frac{\vec{n}(A - S)}{\vec{n}(V - S)}$$

где n, A – нормаль и точка принимающей плоскости, S – точка положения источника света.

Построение теней. Тени как текстура

- Геометрическая оптика Френеля
- Чёткие (*hard shadows*) тени
  - Точечный источник или источник направленного параллельного света

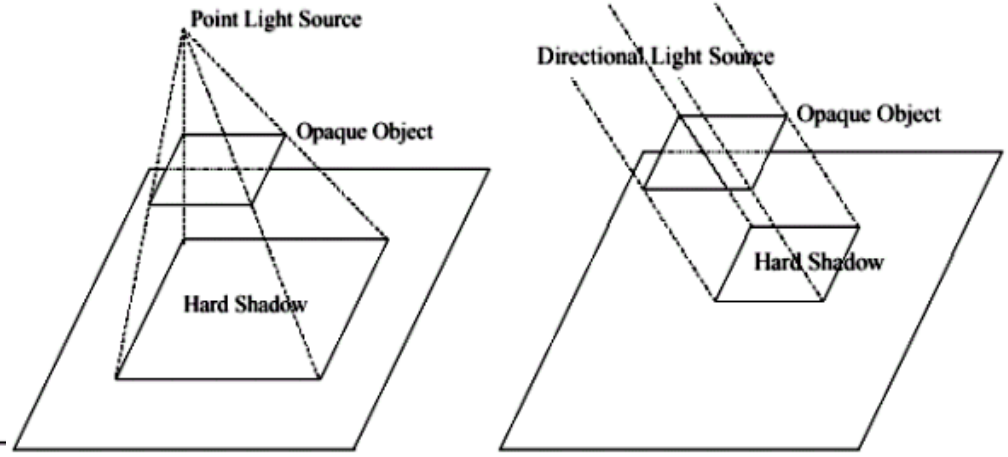


рис. 1.1. чёткая тень.

- Мягкие (*soft shadows*) тени
  - Протяжённый источник света

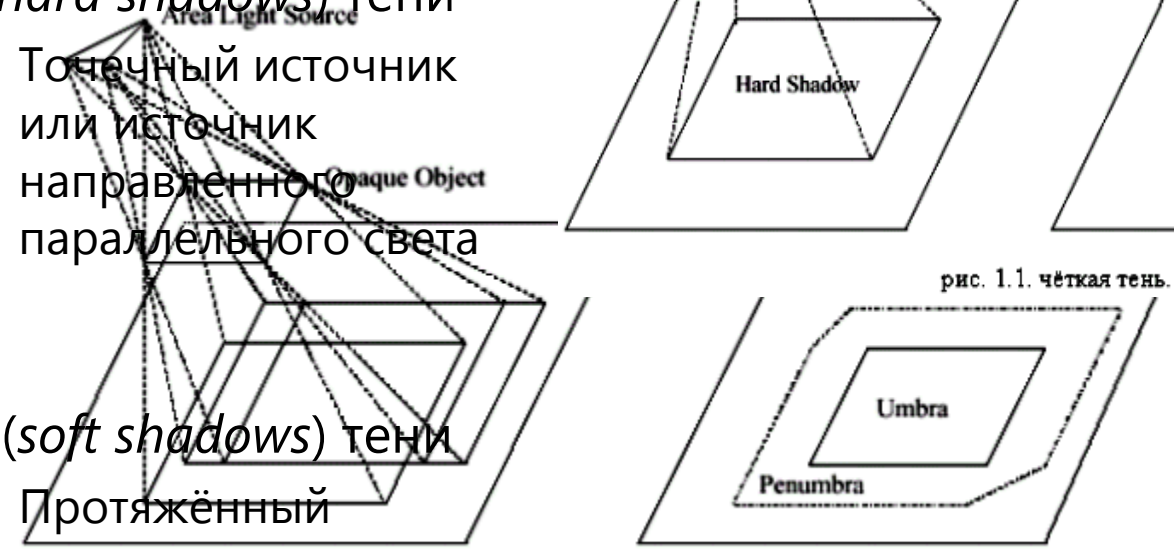


рис. 1.2. мягкая тень.

- **Stencil-буфер**

Построение теней.
Использование буфера теней

- Затенённые точки - это "*hidden surfaces*" с точки зрения источника света.

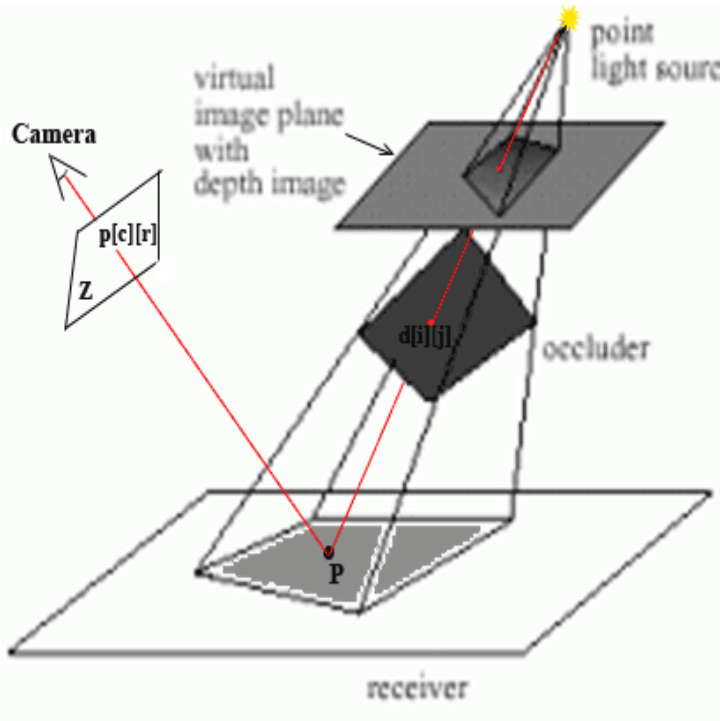
Два этапа:

- Загрузка буфера теней (аналогична Z-буферу, камера в точке источника).
- Визуализация

Визуализация

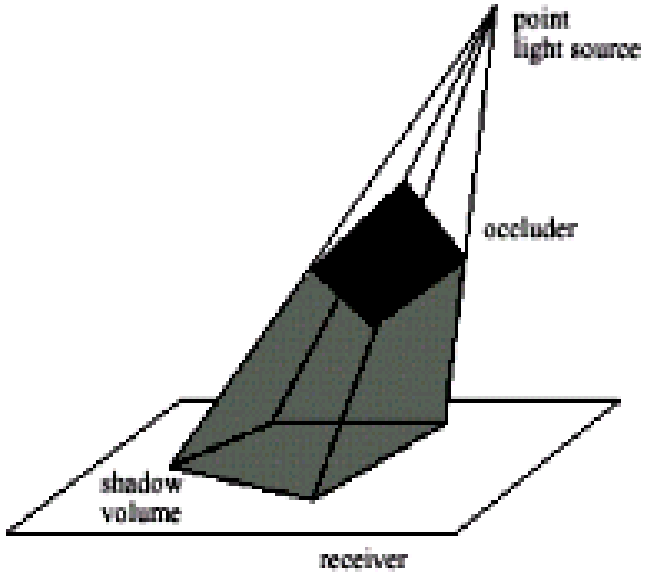
При визуализации точки P через пиксел p[c][r] Z-буфера нужно найти:

- псевдоглубину D для точки P для буфера теней (нормированную в CVV);
- индексы [i][j] элемента буфера теней, подлежащего проверке для точки P;
- значение d[i][j].
- Если значение d<D, то точка P рисуется цветом **ambient.**



Построение теней. Теневые объёмы

- Теневой объём - это представление пространства, из которого не наблюдается источник света, в виде полигонального объекта. При освещении сцены в тени оказываются те объекты, которые попадают внутрь теневого объёма.
- Для того, чтобы точка была затенена, луч света должен войти в теневые объёмы большее число раз, нежели выйти из них, т.е. остаться внутри хотя бы одного теневого объёма. Для упрощения задачи можно принять, что не луч света от источника должен проделать этот путь, а луч от наблюдателя.
- Алгоритм можно разделить на два явных действия: построение "*теневой маски*" (т.е. маски освещённых и затенённых областей 2-х мерной картинки в Stencil-буфере) и отрисовка сцены с использованием *теневой маски*.



Список источников

1) Stanford Computer Graphics Laboratory. Courses in Graphics http://graphics.stanford.edu/courses/
2) 2) The home page of the Graphics Research Group at Harvard University. Computer Graphics courses http://www.eecs.harvard.edu/graphics/#classes , GVI Group http://gvi.seas.harvard.edu/
3) 3) Курсы лаборатории Graphics & Media Lab при ВМК МГУ http://graphics.cs.msu.su/en/study/courses , http://courses.graphicon.ru/main/cg
4) Шикин Е.В., Боресков А.В. Компьютерная графика. Полигональные модели. –М.: ДИАЛОГ-МИФИ, 2001.-464с.
5) Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. – С.Пб : БХВ-Петербург, 2003. –560с.
6) Майкл Ласло. Вычислительная геометрия и компьютерная графика на С++: Пер. с англ. –М.: «БИНОМ», 1997. –304с.
7) Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. - М.: Мир, 2001. - 604с.
8) Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. -М.: Мир, 1989. -512с.
9) Порев В.Н. Компьютерная графика. –СПб.: БХВ-Петербург, 2002. –432с.
10) Хилл Ф. OpenGL. Программирование компьютерной графики. – С.Пб: Питер, 2002. 1088с.