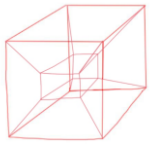
19. Модели трехмерных объектов. Требования, предъявляемые к моделям

Модель - отображение формы и размеров объектов. Основное назначение - правильно их отображать. В основном используются 3 вида моделей:

Каркасная (проволочная) модель.

- простейший вид моделей
- минимум информации: о вершинах и рёбрах объектов
- недостаток: не всегда модель правильно передает представление об объекте. Пример: просто параллелепипед-норм, но с отверстием непонятно, где это отверстие, какие грани оно связывает – перед-зад/левую-правую/верх-низ



Поверхностная модель.

- В графике мы обычно работаем именно с ними (в более простом случае - с каркасными моделями, а после получения результата считать, что на этот каркас натянут материал)
- Поверхность может описываться разными способами, например, аналитически: отдельные участки задаются как участки поверхности того или иного вида (участок сферической поверхности, участок конической поверхности и т. д.). Можем иметь либо аналитическое описание, либо библиотеку поверхностей, хранить те же уравнения и задавать только их параметры и коэффициенты.
- Сложные криволинейные поверхности можно упрощать (например, полигональной аппроксимацией – в виде поверхности многогранника).
- Недостаток: нет информации о том, с какой стороны поверхности находится собственный материал (для моделирования трёхмерных объектов). Решается в объемной модели

Объемная модель.

- Отличие от поверхностной – добавляется указание расположения материала. Проще всего - указать направления внутренней нормали. Нам реально важно!!

Требования к 3-D модели:

1. Компактность - определяется количеством информации, которое необходимо задать и хранить для представления модели. Не всегда основной параметр: если избыточность позволит быстрее выполнять операции (не вычислять каждый раз некоторые параметры, которые будут нужны при моделировании), то можно на нее пойти
2. Способность дополняться новыми свойствами при расширении области применения (1,2-для Курова)
3. Не противоречить исходному объекту.
4. Допускать возможность конструирования тела целиком. (мощность модели)
5. Позволять вычисление геометрических характеристик тела и проведение расчетов

Свойства объемных моделей

Однородность (тело заполнено изнутри), конечность (конечный объем), жесткость (сохранять форму независимо от положения в пространстве)

Требования к программам геометрического моделирования

1. Согласованность операций (любые операции, проводимые над сплошными телами приводят к сплошным телам)
2. Возможность описания (любое тело должно представляться в машинном виде)
3. Непротиворечивость информации (любая точка в пространстве принадлежит только одному телу, и для каждой точки признак – принадлежит кому-то или нет
4. Компактность модели (определяется количеством информации)
5. Открытость модели (применимость при работе с разными алгоритмами)

20. Операции преобразования в трехмерном пространстве. Матрицы преобразований

Аффинное преобразование - При которых плоскость не вырождается в прямую или в точку, сохраняется параллельность прямых (и плоскостей) и существует обратное преобразование. Может быть представлено в виде совокупности переноса, масштабирования, поворота

Матрица преобразований в 3х-мерном пространстве будет иметь размерность 4x4. Координаты точек (x, y, z) заменятся четверкой (wx, wy, wz, w), w != 0.

(x’,y’,z’1)=(x,y,z,1)M

Перенос

- Это изменение местоположения изображения

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$$

Формулы: x' = x + dx; y' = y + dy; z' = z + dz

Масштабирование

- Это изменение размеров и пропорций изображения

$$\begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Формулы: x' = x * kx + (1 - kx) * xm; y' = y * ky + (1 - ky) * ym; z' = z * kz + (1 - kz) * zm

- При использовании матрицы, сначала нужно перенести тело и центр масштабирования так, чтобы центр масштабирования оказался в начале координат, затем вернуть
- При отрицательных значениях коэффициентов масштабирования происходит симметричное отображение масштабируемого объекта относительно соответствующей координатной плоскости: При KX = -1, KY = 1, KZ = 1 - отображение относительно плоскости YOZ (или ей параллельной); KY=-1: XOZ; Kz=-1: XOY.
- При kx = ky = kz = -1 получим симметрию относительно начала координат.

- если увеличить фигуру с центром масштабирования за пределами этой фигуры, то результатом будет не только увеличение объекта, но и смена его местоположения! Не 0

Поворот.

это операция, заключающаяся в изменении ориентации изображения

OZ, OX, OY

$$\begin{pmatrix} \cos & \sin & 0 & 0 \\ -\sin & \cos & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos & \sin & 0 \\ 0 & -\sin & \cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos & 0 & -\sin & 0 \\ 0 & 1 & 0 & 0 \\ \sin & 0 & \cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OZ:

$x' = xc + (x - xc) * \cos\theta - (y - yc) * \sin\theta$

$y' = yc + (x - xc) * \sin\theta + (y - yc) * \cos\theta$

OX:

$y' = yc + (y - yc) * \cos\theta - (z - zc) * \sin\theta$

$z' = zc + (z - zc) * \cos\theta + (y - yc) * \sin\theta$

OY:

$x' = xc + (x - xc) * \cos\theta + (z - zc) * \sin\theta$

$z' = zc + (z - zc) * \cos\theta - (x - xc) * \sin\theta$

Перспективное проецирование на плоскость z = 0.

- ЭТО ПЕРСПЕКТИВНОЕ ПРЕОБРАЗОВАНИЕ, А НЕ АФИННОЕ.
- это преобразование одного трехмерного пространства
- Весь правый столбец отвечает за проецирование на разные плоскости, если бы мы хотели получить проекцию на ось X, то -1/C было бы в первой строке последнего столбца, где C-фокусное расстояние на оси x
- Проекция на одну плоскость - одноточечное проецирование, но можно совмещать и получать двухточечные проекции и тд.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/C \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Коммутативность

Коммутативность - это независимость результата преобразований от порядка, в котором они происходят.

Коммутативные операции: Перенос-перенос, Масштабирование – масштабирование, Поворот-поворот, Однородное масштабирование – поворот. Остальные комбинации - не коммутативны.

Аддитивные

Это значит, что матрица итогового преобразования может быть получена сложением

- Перенос (матрица переноса + матрица переноса)

$$M_{пер-пер}=\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx1+dx2 & dy1+dy2 & 1 \end{pmatrix}$$

- Поворот (матрица поворота + матрица поворота)

$$M_{пов-пов}=\begin{pmatrix} \cos(\theta 1+\theta 2) & -\sin(\theta 1+\theta 2) & 0 \\ \sin(\theta 1+\theta 2) & \cos(\theta 1+\theta 2) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Мультипликативные

Это значит, что матрица итогового преобразования может быть получена умножением

- Масштабирование (матрица масштабирования * матрица масштабирования)

$$M_{масш-масш}=\begin{pmatrix} kx1*kx2 & 0 & 0 \\ 0 & ky1*ky2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

21. Трехмерное отсечение. Виды отсекателей. Вычисление кодов концов отрезка для каждого типа отсекателей. Алгоритм отсечения отрезков средней точкой

Общее

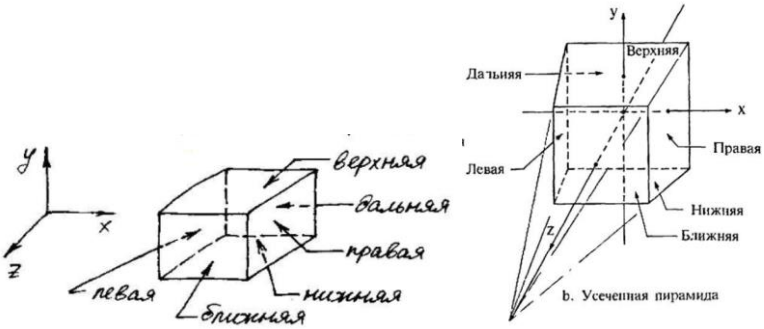
Отсечение - это операция удаления изображения за пределами выделенной области, называемой отсекателем. Стирание – внутри.

Виды отсечений – Двумерное/Трехмерное Отсечение в пространстве либо сводится к отсечению на плоскости, либо сами плоские алгоритмы без труда распространяются на трехмерный случай. Цель отсечения в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Основные **виды трёхмерных отсекателей** - Прямоугольный параллелепипед и Усечённая пирамида (пирамида видимости, исп. При центральном проецировании) (6 граней)

Вычисление кодов концов отрезка

Важно подписать "верхняя грань отсекателя" и т.д



1. Прямоугольный параллелепипед: как и в двумерном отсечении, здесь используется битовый код видимости (6) вершины.

Т1 = 1, если x < xл, 0 – иначе; Т2 x > xп; Т3 y < yн; Т4 y > yв; Т5 z > zб; Т6 z < zд,

2. Усечённая пирамида видимости: отрезок, соединяющий центр проекции и центр пирамиды, совмещают с осью oz, используются 6 пробных функций fп,… для каждой грани (функция, которая позволяет определить, как расположена точка относительно плоскости)

Предварительно вычисляются коэффициенты плоскостей: прав(α1, α2), лев(β1, β2), верх(γ1, γ2), ниж(δ1, δ2)

Уравнение прямой на плоскости xz, несущей проекцию правой грани отсекателя, имеет вид: x=(z-zцп)*xп/(zд-zцп)=z α1+ α2

$$\alpha_1 = x_П / (z_Д - z_{ЦП}) \text{ и } \alpha_2 = -\alpha_1 z_{ЦП}.$$

Подстановка координат x и z точки P в пробную функцию правой грани fп даст следующий результат: fп=x-z α1- α2: >0 – P справа о плоскости, =0 – на, <0-слева

И так далее:

fл=zβ1- β2, где β1=xл/(zд-zцп), β2=- β1zцп: >0 справа

fв=y-zγ1- γ2, где γ1=yв/(zд-zцп), γ2=- γ1zцп: >0 выше

fн=y-zδ1- δ2, где δ 1=yн/(zд-zцп), δ2=- δ1zцп: >0 ниже

fб=z-zб: >0 ближе

fд=z-zд: >0 дальше

Чем ближе zцп к бесконечности, тем больше фигура приближается к прямоугольному параллелепипеду, а пробные функции – к функциям прямоугольного параллелепипеда.

Внимание! если концы отрезка лежат за центром проекции – неверные результаты (так как остальные грани пересекаются в цп). Решение: если z > zцп, обратить первые четыре бита кода.

Сумма кодов концов отрезка

Все как в двумерном

- сумма битов кода 0 - точка внутри отсекателя.
- Отрезок видим S1 = S2 = 0.
- Если S1 = 0 и S2 != 0 или S1 != 0 и S2 = 0, то отрезок частично видимый.
- Если обе суммы ненулевые - частично видимым или полностью невидимым. логическое произведение концов отрезка: P = T1(1) * T2(1) + T1(2) * T2(2)…до 6, где T1 и T2 - массивы с кодами конечных точек отрезка. P!=0-полностью невидим

Алгоритм трехмерного отсечения средней точкой

Как в двумерном. Изменить размерности Ткод, Окно, подпрограммы Конец и Логическое произведение переписать с учетом 3 измерений.

```
подпрограмма вычисления кода концевой точки отрезка относительно трехмерной усеченной пирамиды
subroutine Конец(P, Окно; Ткод, Сумма)
Px, Py, Pz — координаты x, y и z точки P
Окно — массив 1 × 7, содержащий координаты (xЛ, xП, yН, yВ, zБ, zД, zЦП) левой, правой, нижней, верхней, ближней, задней сторон окна и центра проекции
Ткод — масссив 1 × 6, содержащий код концевой точки
Сумма — сумма элементов Ткод
   вычисление α1, α2, β1, β2, γ1, γ2, δ1, δ2
   α1 = xП/(zД − zЦП)
   α2 = −α1 zЦП
   β1 = xЛ/(zД − zЦП)
   β2 = − β1zЦП
   γ1 = yВ/(zД − zЦП)
   γ2 = − γ1 zЦП
   δ1 = yН/(zД − zЦП)
   δ2 = −δ1zЦП
   определение кода концевой точки
   if Px − Pzβ1 − β2 < 0 then Ткод(6) = 1 else Ткод(6) = 0
   if Px − Pzα1 − α2 > 0 then Ткод(5) = 1 else Ткод(5) = 0
   if Py − Pzδ1 − δ2 < 0 then Ткод(4) = 1 else Ткод(4) = 0
   if Py − Pzγ1 − γ2 > 0 then Ткод(3) = 1 else Ткод(3) = 0
   if Pz − zБ > 0 then Ткод(2) = 1 else Ткод(2) = 0
   if Pz − zД < 0 then Ткод(1) = 1 else Ткод(1) = 0
   вычисление суммы
   Сумма = 0
   for i = 1 to 6
     Сумма = Сумма + Ткод(i)
   next i
   return
```

1. Ввод ($X_л$, $X_п$, $Y_н$, $Y_в$), $P_1(X_1,Y_1)$, $P_2(X_2,Y_2)$, точности ε
2. Вычисление кодов концевых точек T1 и T2, вычисление сумм кодов концов

$S_1= \sum_{i=1}^{4} T1_i$, $S_2= =\sum_{i=1}^{4} T2_i$

3. если (S1==0) and (S2==0), то переход к п. 8.визуализация
4. Вычисление побитного логического произведения
5. Если P!=0, то переход к п. 9. Конец
6. Запоминание вершины R=P1, tr=T1
7. Поиск видимой вершины: цикл по _ in range(2)
   7.1. если S2==0, то занесение P2 в результат, переход к 7.4,
   7.2. Поиск точки пересечения. Цикл пока $|P_1- P_2| > \varepsilon$:
      7.2.1. $P_{ср.} = (P_1 + P_2)/2$
      7.2.2. Tcp
      7.2.3. Проверка видимости отрезка [Pcp, P2]. Если отрезок полностью невидим, то P2=Pcp (отрезаем его), иначе P1=Pcp (приближаемся к точке пересечения с другой стороны)
   7.3. Если отрезок не является полностью невидимым t1&t2==0, то занесение в результат вершины P2, иначе весь отрезок объявляется невидимым, к пункту 9
   7.4. Обмен местами вершин P1=P2, t1=t2, P2=R, t2=tr и переход к пункту 7 для поиска второй вершины
8. Визуализация отрезка
9. Конец

22. Отсечение отрезков в трехмерном пространстве. Трехмерный алгоритм Кируса Бека

Общее

Отсечение - это операция удаления изображения за пределами выделенной области, называемой отсекателем. Стирание – внутри.

Виды отсечений – Двумерное/Трехмерное Отсечение в пространстве либо сводится к отсечению на плоскости, либо сами плоские алгоритмы без труда распространяются на трехмерный случай. Цель отсечения в определении точек, отрезков или их частей, которые лежат внутри отсекателя.

Основные виды трёхмерных отсекателей - Прямоугольный параллелепипед и Усечённая пирамида (пирамида видимости, исп. При центральном проецировании) (6 граней)

Условия для Кируса Бека

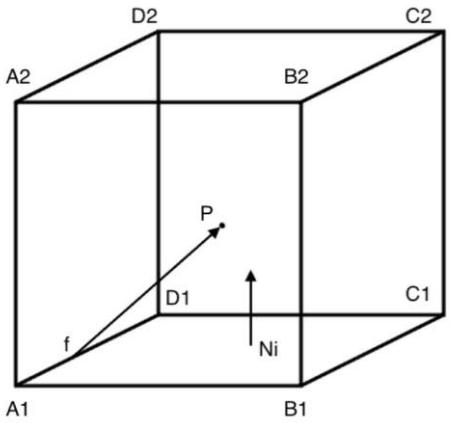
- Все как в двумерном варианте, но теперь ребра заменяются гранями (задаются списками вершин) для опреднлиния (видимость и пересечение), все точки и векторы будут иметь три компоненты, и все операции - в трехмерном пространстве.
- произвольный выпуклый отсекатель

Определение видимости точки

Видимость точек - относительно граней отсекателя, а не ребер, сам способ не меняется:

Ni - вектор внутренней нормали к i-й грани отсекателя, f - произвольная точка грани отсекателя (угловые точки граней)

Вычислим скалярное произведение Ni и (P - f): a*b = ax*bx + ay*by + az*bz: > 0 - точка видима, < 0 - точка невидима, = 0 - точка лежит на грани (из-за угла)



Точки пересечения

Все так же

$n_{вi}\,[P(t) - f_i] = 0$, $n_{вi}\,[P_1 + (P_2-P_1)\,t - f_i] = 0$, или $n_{вi}\,[P_1 - f_i] + n_{вi}\,[P_2-P_1]\,t = 0$.

$D = P_2-P1$- вектор ориентации отрезка (директриса)

$Wi = P_1 - f_i$ – коэффициент, пропорционален расстоянию от начала отрезка до исследуемой граничной точки

Тогда условие: $t\,(n_{вi}\,D) + Wi\,n_{вi} = 0$, откуда $t = -(Wi\;n_{вi})/(D\,n_{вi})$

Знаменатель D $n_{вi}$ принимает нулевое значение в двух случаях:

1. $D=0$ (P2=P1) - вырождение отрезка в точку. Эта точка лежит: числитель Wi $n_{вi} < 0$ - вне, =0 – на границе, >0 - внутри

2. $Dn_{вi}=0$ - отрезок параллелен ребру отсекателя.

Для определения того, по какую сторону параллельный границе отрезок находится, достаточно проверить на видимость произвольную точку отрезка. Знак скалярного произведения, стоящего в числителе, W_i*N_вні позволяет определить положение первой вершины отрезка относительно границы отсекателя. Если W_i*N_вні ≥0, то отрезок является видимым для текущей рассматриваемой стороны отсекателя. Если же W_ск<0, отрезок является полностью невидимым.

Выбор точек пересечения

Если значения t не принадлежат интервалу $0 \le t \le 1$, то их не рассматривают, поскольку они соответствуют точкам, лежащим вне исходного отрезка.

Несмотря на то, что отрезок может пересечь замкнутую выпуклую область не более, чем в двух точках, при решении уравнения можно получить более двух решений в интервале $0 \le t \le 1$. Полученные решения следует разбить на две группы: верхнюю и нижнюю, в зависимости от близости найденной точки пересечения к началу или концу отрезка.

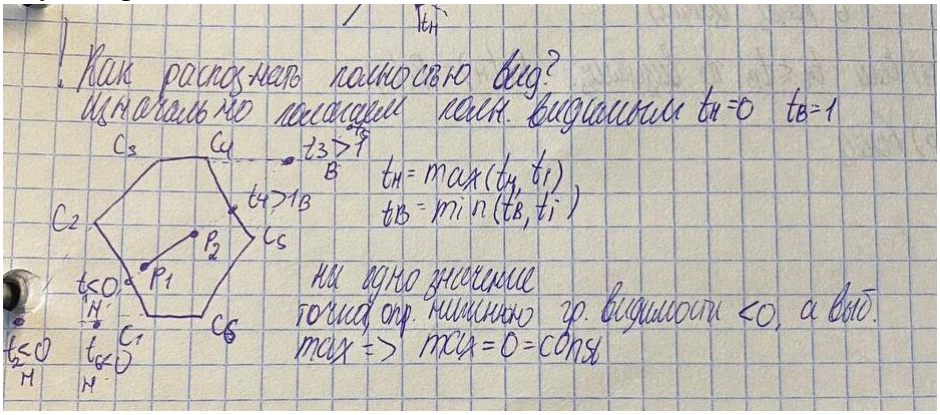
Чтобы отрезок был видимым относительно всего отсекателя, он должен быть видим относительно всех ребер отсекателя одновременно. Концам видимой части отрезка будут соответствовать два значения параметра t, одно из которых является максимальным значением из нижней группы t_{maxmin}, а второе - минимальным из верхней группы t_{minmax},

Найденное значение параметра t для очередной точки пересечения рассматривают в качестве возможного верхнего предела $t_в$, если знаменатель $Dn_{вi} < 0$; в случае же, когда знаменатель положителен, значение параметра t определяет точку, которую относят к нижней границе видимости $t_н$.

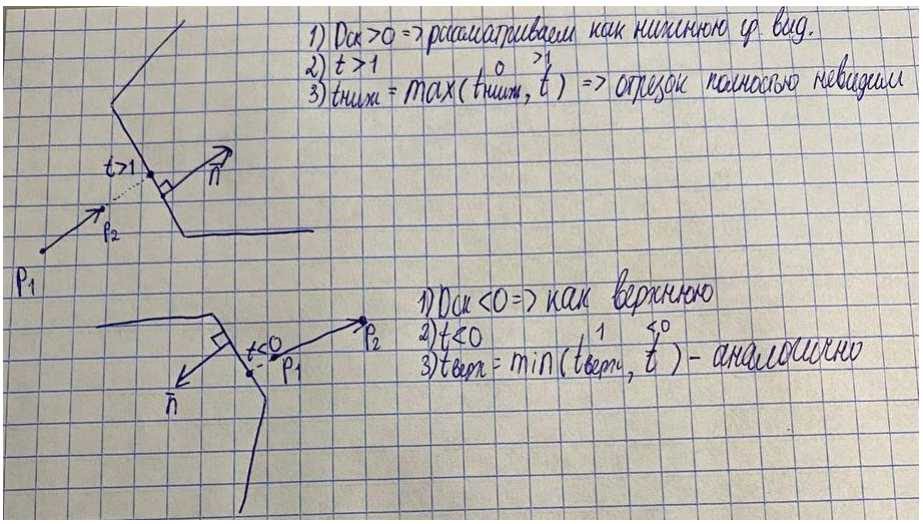
Распознавание

- Как для полностью видимого, так и для полностью невидимого отрезка значения параметра t будут лежать за пределами допустимого интервала. Поэтому не удается получить простой критерий идентификации полностью видимых и полностью невидимых отрезков. Требуется специальный для выпуклого отсекателя метод (придется пройти полный цикл).

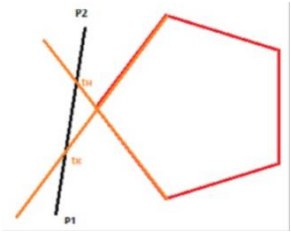
Как же распознается полностью видимый? Мы изначально положим tн=0, tв=1 и к концу алгоритма ничего не изменится.



А для полностью невидимого получатся некорректные значения или частный случай.



Частный случай



получим две точки, соответствующие параметрам t_н и t_к, определяющие начало и конец видимой части отрезка. То есть формально мы получаем два значения параметра, расположенных в интервале от нуля до единицы. Если не выполнить проверку того, что начало видимой части отрезка располагается до его конца, то будет получен неверный результат. То есть будет внесена следующая проверка:

t_н≤=t_к

Алгоритм

Такой же, только 3d

1. Ввод исходных данных: координат концевых точек отрезка $P_1(P_{1.x}, P_{1.y})$, $P_2(P_{2.x}, P_{2.y})$, числа сторон n окна отсечения и координат его вершин (массив C).
2. Проверка отсекателя на выпуклость. Если отсекатель невыпуклый, то вывести ошибку и перейти к пункту 7.
3. Вычисление вектора ориентации отрезка D = P2 - P1.
4. Инициализация пределов значений параметра t при условии, что отрезок полностью видимый, то есть $t_н = 0$, $t_в = 1$.
5. Начало цикла по всем сторонам отсекающего окна (i = 1..n).
   - 5.1. Вычисление вектора внутренней нормали к очередной i-ой стороне окна отсечения - $n_{вi}$
   - 5.2. Вычисление вектора $Wi = P_1 - f_i$ (=Ci)
   - 5.3. Вычисление скалярных произведений $W_{ск\,i} = Wi\ n_{вi}$, $D_{ск\,i} = Dn_{вi}$.
   - 5.4. Если $D_{ск\,i} = 0$ (вырождение отрезка в точку или его параллельность стороне отсекателя) то:

     {если $W_{скi} < 0$, то отрезок (точка) невидим и переход к п. 7;
     Иначе видим и переход к следующему шагу цикла}

     Иначе Вычисление параметра $t = -(W_{скi} / D_{скi})$

5.5. если $D_{cki} > 0$ то, поиск нижней границы параметра t:

- если t>1, то переход к п.7 (отрезок невидим).
- иначе $t_н = \max(t, t_н)$

иначе поиск верхней границы параметра t:

- если t<0, то переход к п.7 (отрезок невидим)
- иначе $t_в = \min(t, t_в)$

5.6. Конец цикла по сторонам отсекателя.

6. Проверка фактической видимости отсеченного отрезка. Если $t_н <= t_в$, то визуализация отрезка в интервале от $P(t_н)$ до $P(t_в)$.
7. Конец алгоритма.

Оценка числа операций растет линейно с ростом числа сторон или граней у отсекателя.

23. Определение факта выпуклости трехмерных тел. Разбиение тела на выпуклые многогранники

Определение факта выпуклости

Двухмерный алгоритм можно обобщить для трехмерных многогранников:

Для каждой полигональной грани:

1. Перенести тело так, чтобы одна из вершин грани оказалась в начале координат
2. Повернуть тело относительно начала координат так, чтобы одна из двух смежных выбранной вершине сторон грани совпала с одной из осей координат (x, например)
3. Повернуть тело вокруг выбранной оси координат так, чтобы выбранная гран легла на координатную плоскость (z=0, например)
4. Для всех вершин тела, не принадлежащих выбранной грани, проверить знаки координаты, которая перпендикулярна этой грани (z)
    - Все знаки совпадают или равны 0-тело выпукло относительно этой грани (для выпуклости нужна выпуклость относительно всех)
    - Все равны 0 – тело вырождено (плоское)

Вектор внутренней нормали к выбранной плоскости, заданный в повернутой СК, имеет все компоненты 0, кроме той, которая перпендикулярна этой плоскости. Знак этой компоненты для выпуклой грани=найденному знаку. Для определения исходной ориентации внутренней нормали нужно применить к ней обратное преобразование поворотов.

Разбиение на выпуклые многогранники

Кирус-Бек работает только с выпуклыми отсекателями. Задачу разрезания простого невыпуклого тела на составляющие его выпуклые тела можно решить путем обобщения метода переносов и поворотов из двумерного случая. Предполагается, что тело представляет собой многогранник с плоскими гранями.

Для каждой грани тела:

1. Перенести тело так, чтобы одна из вершин грани оказалась в начале координат
2. Повернуть тело относительно начала координат так, чтобы одна из двух смежных выбранной вершине сторон грани совпала с одной из осей координат (x, например)
3. Повернуть тело вокруг выбранной оси координат так, чтобы выбранная гран легла на координатную плоскость (z=0, например)
4. Для всех вершин тела, не лежащих на рассматриваемой грани, определить знаки координаты, которая перпендикулярна выбранной плоскости (z).

a. Если знаки для всех вершин одинаковы (или равны 0), то тело выпуклое относительно рассматриваемой грани. Иначе нужно разрезать тело плоскостью, несущей рассматриваемую грань. Рассматриваемую процедуру применяем к каждой из полученных частей. Продолжать работу до тех пор, пока все тела не станут выпуклыми

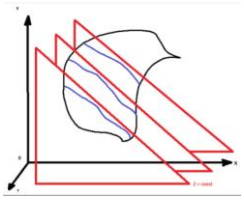
24. Алгоритм плавающего горизонта.

Общее

Алгоритм решает задачу представления трехмерной поверхности, заданной неявным уравнением F(x,y,z)=0 путем изображения совокупности кривых, что требует решения так называемой центральной задачи графики-удаления невидимых линий поверхности (алгоритм неполноценный-решает важную, но частную задачу)

Наблюдатель располагается на положительной части оси z и смотрит на начало координат.

Рассматриваемая поверхность с некоторым шагом рассекается плоскостями, перпендикулярными оси Z. В каждом отсечении получается кривая, которая описывается уравнением y=f(x, z=const) или x=Q(y, z=const).

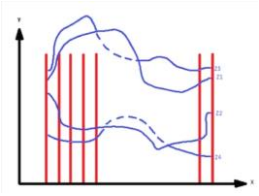


Полученные кривые можно проецировать на плоскость xoy (начиная с ближайшего к наблюдателю сечения) и изобразить видимые части каждой кривой.

Кривые в 1 и 2 сечении всегда видимы (вторая выше или ниже) (если совпадают - одна кривая). Начиная с третьей кривой нужно решать задачу удаления невидимых линий, которая сводится к определению видимости точек очередной кривой (с помощью нахождения мин/макс)

Определение видимости точек кривой

Поставленная задача решается в пространстве изображения (скорее попеременно). Точка видима, если она выше самой верхней/ниже самой нижней. Нужно вычислить функцию в очередной точке x, расположенной на кривой. Она видима, если y(x) > ymax(xi) или y(x) < ymin(xi) и тогда высвечивается, иначе – невидима (расположена между 2 горизонтами)



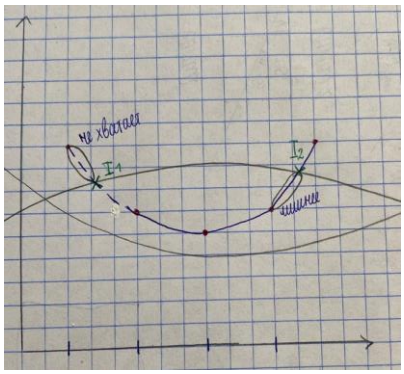
Отсюда и название: мы решаем задачу определения видимости с использованием двух массивов, называемых верхним и нижним горизонтами. Верхний – участки кривых, образующих массив с наибольшими значениями ординат ymax(xi), нижний – ymin(xi). Требуется хранить горизонты и поддерживать их (обновлять при необходимости)

Поддержание горизонта сводится к вычислению значения y(x,z=const) и сравнению: если y(x,z=const)>y_max, то y_max=y(x,z=const), иначе если y(x,z=const)<y_min, то y_min (x)=y(x,z=const).

На интервалах

Обычно шаг dx=1, но его можно брать больше, если нужно ускорить вычисления/кривые довольно гладкие.

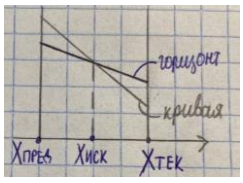
Обрабатывая новую кривую, находим ее точки. Если первая видима, то рисуем. Руководствуемся правилом: если очередная точка невидима, то мы её не соединяем с предыдущей точкой, иначе-соединяем



Получаем 2 ошибочных случая: видимый участок кривой объявляется невидимым, и наоборот. Меняем правило: если текущая точка видима, а предыдущая нет – найти точку пересечения и изобразить от пересечения до текущей. Аналогично во втором.

Точка пересечения

Для упрощения вычислений, с учётом того, что dx достаточно мало, а также предполагая, что поверхности (и получаемые кривые) достаточно гладкие, на текущем интервале будем аппроксимировать кривую отрезком.



Уравнение прямой Y=M(X-Xk)+Yk

Мгор=(Yгор.тек-Yгор.пред)/dX

Мкр=(Yкр.тек-Yкр.пред)/dX

Точка пересечения принадлежит кривой и горизонту:

Yпер=Мгор(X-Хпред)+Yгор.пред

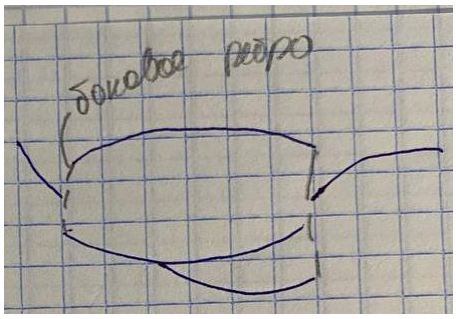
Yпер=Мкр(X-Хпред)+Yкр.пред

Мгор(X-Хпред)+Yгор.пред= Мкр(X-Хпред)+Yкр.пред

X=Хпред + (Yкр.пред- Yгор.пред)/ (Мгор-Мкр)=Хпред+dX(Yкр.пред-Yгор.пред)/((Yгор.тек-Yгор.пред)- Yкр.тек-Yкр.пред))= Хпред+dX(Yкр.пред-Yгор.пред)/(dYгор-dYкр))

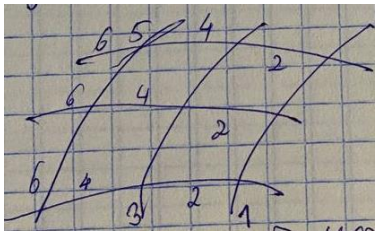
Улучшение качества изображения поверхности

Если мы поворачиваем кривые, то можно столкнуться со следующей ситуацией:

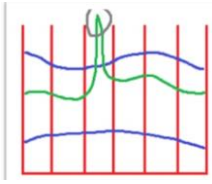


Человеку надо подключить фантазию, чтобы увидеть поверхность. Для решения этой проблемы нужно начальные и конечные точки каждой кривой соединить друг с другом, например, отрезками. Указанные рёбра можно соответственно назвать левым боковым и правым боковым. Для этого руководствуемся правилом: если текущая точка первая, то соединяем с первой предыдущей, и аналогично для последней

Сетка: Можно улучшить дополнительным рассечением поверхности плоскостями, перпендикулярной ещё одной оси координат, но построение должно происходить одновременно в каждой серии.



Недостатки алгоритма



1. В результате вычисления значения функции с указанным шагом не будет вычислено значение функции в месте, указанном на рисунке серым цветом, и пик будет потерян. Шаг изменения x выбран неверно, и мы не сможем правильно восстановить функцию.
2. Алгоритм не может отображать поверхности, для которых y = f(x, z) не является функцией (то есть, можно найти несколько значений y для этих x и z).
3. Алгоритм хранит горизонты и постоянно их обновляет, из-за этого он недостаточно быстродействующий.
4. При любом изменении угла обзора происходит полный перерасчет.

Алгоритм

Для каждой плоскости z=const

1. Обработать левое боковое ребро.
2. Для каждой точки текущей кривой P:
   2.1 Определить видимость точки P
   2.2 Если текущий сегмент видим:

2.2.1 Изобразить его полностью.

2.3 Если видимость изменилась:

2.3.1 Найти точку пересечения кривой с горизонтом.

2.3.2 Если текущая точка невидима: Изобразить участок от пересечения до точки

2.3.3 Если текущая точка видима: Изобразить участок от точки до пересечения

2.4 обновить массивы верхнего и нижнего горизонтов

3. Обработать правое боковое ребро,

Обработка боковых рёбер:

Q - переменная, хранящая крайнюю точку предыдущей кривой, P - первая (для левого бокового ребра) или последняя (для правого бокового ребра) точка текущей кривой

[Если текущая кривая не первая: Изобразить ребро (P; Q)] Q = P

Вопросы

- В чём основная идея алгоритма? - свести решение трёхмерной задачи к решению ряда двухмерных. Для этого исходная поверхность пересекается с рядом параллельных секущих плоскостей вида x = const , y = const или z = const. Плоскости обрабатываются в порядке удаления от точки обзора, для каждой плоскости получившаяся кривая изображается с помощью набора точек, соединяемых прямыми линиями. Во время этого анализа отрезки проверяются на видимость и меняют массивы горизонтов в случае видимости.
- Всегда ли в вашей программе шаг по оси x больше 1? (всегда ли надо искать пересечения)

Шаг можно задать произвольно. Поиск пересечений осуществляется в том случае, если видимость концов отрезка различна. В этом случае на отрезке имеется точка, по прохождении которой видимость точек отрезка меняется. Для её нахождения можно пройти по отрезку в от начальной точки до отрезка, и определив первую точку, имеющую отличную от начала отрезка видимость

25. Задача удаления невидимых линий и поверхностей. Ее значение в машинной графике. Классификация алгоритмов по способу выбора системы координат (объектное пространство, пространство изображений).

Задача удаления невидимых линий и поверхностей

Является одной из наиболее сложных в компьютерной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линии ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

- В компьютерной графике нет единого алгоритма для решения этой задачи. Поэтому существует куча алгоритмов для разных областей.
- Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок в обоих показателях.
- Все алгоритмы удаления невидимых линей включают в себя сортировку.

Про сортировки

- Главная сортировка (первая) ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения.
  Основная идея этой сортировки, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения.
- После этого остается провести сортировку по горизонтали и вертикали, чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения.
- Эффективность алгоритмов в большей мере зависит от эффективности сортировок
- Для повышения эффективности сортировок используется также когерентность сцены - тенденция неизменяемости характеристик сцены в малом (тогда алгоритмы похожи на растровую развертку).

Классификация алгоритмов по способу выбора системы координат

Алгоритмы, работающие в **объектном пространстве**, имеют дело с СК, в которой описаны эти объекты. Применяются, когда нужна высокая точность (ограничена лишь точностью вычислений), для несложных сцен. ~O(n^2). n - число объектов.

Алгоритмы, работающие в **пространстве изображений**, имеют дело с СК того экрана, на котором объекты визуализируются. Точность вычислений ограничена разрешающей способностью экрана. ~ O(nN), n - число объектов, N - число пикселей.

Здесь легче применять когерентность при растровой реализации (близко расположенные пиксели чаще обладают одинаковыми свойствами), поэтому он эффективней, несмотря на то, что N=прим(512)^2>n

Алгоритмы, формирующие список приоритетов, работают попеременно в обеих СК.

26. Алгоритм Робертса. Основные этапы и математические основы каждого этапа.
27. Алгоритм Робертса. Формирование матрицы тела. Удаление нелицевых граней.
28. Алгоритм Робертса. Удаление отрезков, экранируемых другими телами.

1. Первый алгоритм для удаления невидимых линий (ребер),
2. работает в объектном пространстве,
3. с выпуклыми многогранниками
4. в виде проволочной модели.
5. O (число тел^2)

Основные этапы решения задачи:

0. (предварительный) формирование исходных данных
1. Удаление невидимых линий, экранируемых самим телом (лицо/спина). Если в сцене только 1 тело, то конец.
2. Удаление невидимых линий, экранируемых другими телами сцены
3. Удаление невидимых частей новых ребер, появившихся при протыкании тел.

Этап 0. Формирование исходных данных

Для каждого тела нужно сформировать матрицу тела V(4xn), n-число граней, состоящую из коэффициентов уравнений плоскостей, проходящих через его грани.

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

Матрица должна быть сформирована корректно: любая точка внутри тела должна быть расположена по положительную сторону по отношению к каждой грани тела. Координаты точки внутри тела можно получить усреднением координат всех вершин тела или усреднением min и max. Находим произведение вектора координат полученной усредненной точки на матрицу тела. В результирующем векторе отрицательные элементы соответствуют столбцам, для которых это не выполняется, их нужно домножить на -1.

Пример: прямоугольный параллелепипед с центром в начале координат: хл=-1, хп=1, ун=-1,ув=1, zд=-1, zб=1. Матрица

$$[V] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}, [S] = [0 \quad 0 \quad 0 \quad 1], [S][V] = [1 \quad \mathbf{-1} \quad 1 \quad \mathbf{-1} \quad 1 \quad \mathbf{-1}]$$

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

Зачем: плоскость (x-y+z-1=0) = плоскость (-x+y-z+1=0). Формируем пробную функцию, которая для одной и той же точки даст разные знаки fпроб=x-y+1-1 и вторая. Поэтому просто по знаку нельзя было бы судить о положении точки.

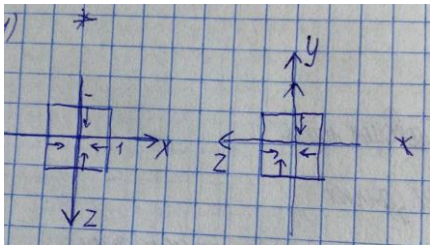
Чтобы преобразовать тело, нужно его матрицу умножить слева на обратную матрицу преобразований

$$[VT] = [T]^{-1}[V]$$

Преобразованная точка лежит внутри преобразованного тела

Этап 1. Удаление невидимых линий, экранируемых самим телом

Зритель находится в бесконечности на положительной полуоси Z и смотрит на начало координат. [E] = [0, 0, -1, 0] = вектор взгляда наблюдателя и образ точки в -бесконечности на оси Z., то есть любой точки (x, y, -бск).



Эта точка лежит по отрицательную сторону для тех граней, которые являются не лицевыми. Для нахождения невидимых граней нужно вектор взгляда умножить на матрицу тела. Тогда в полученном векторе отрицательные компоненты соответствуют невидимым граням, а невидимые ребра образованы пересечением невидимых граней.

$[E][V] = [0 \quad 0 \quad 0 \quad 0 \quad -1 \quad 1]$ Левая, правая, верхняя, нижняя – на границе видимости/невидимости, задняя невидима, передняя видима. Нелицевых отрезков нет

Этап 2. Удаление невидимых линий, экранируемых другими телами сцены

Для сравнения отрезка P1P2 с телом удобно использовать параметрическое представление отрезка P(t)=P1+(P2-P1)t, 0<=t<=1.

Точка отрезка невидима, если луч, соединяющий ее с точкой наблюдения, на своём пути встречает преграду в виде другого тела, т.е. проходит через него (тогда часть луча внутри тела, и видима относительно каждой грани). Этот луч не || оси z, так как точка в бск

Формируется параметрический отрезок от точки P(t) до точки наблюдения g[0,0,1,0]: Q(t, a)= P1+(P2-P1)t +ag, , 0<=t<=1, a>=0 (иначе наблюдатель по другую сторону от тела. Тела, экранирующие P(t) между отрезком и наблюдателем). Здесь t-точка на отрезке P(t), a-точка на луче. Фактически Q-плоскость в трехмерном пространстве, проходящая через отрезок и луч, (a, t) - точка на этой плоскости

Таким образом, для определения части отрезка, которая экранируется телом, необходимо найти уравнение луча, умножить его на матрицу тела, и найти такие (a, t), при которых в полученном векторе все +.

Система неравенств [H]=[Q][V]=[P1][V]+[d][V]t+[g][V]a, hj>0, j=1..n

hj=Pj+qjt+wja>0, j=1..n, 0<=t<=1, a>=0

hj=0-граничное условие

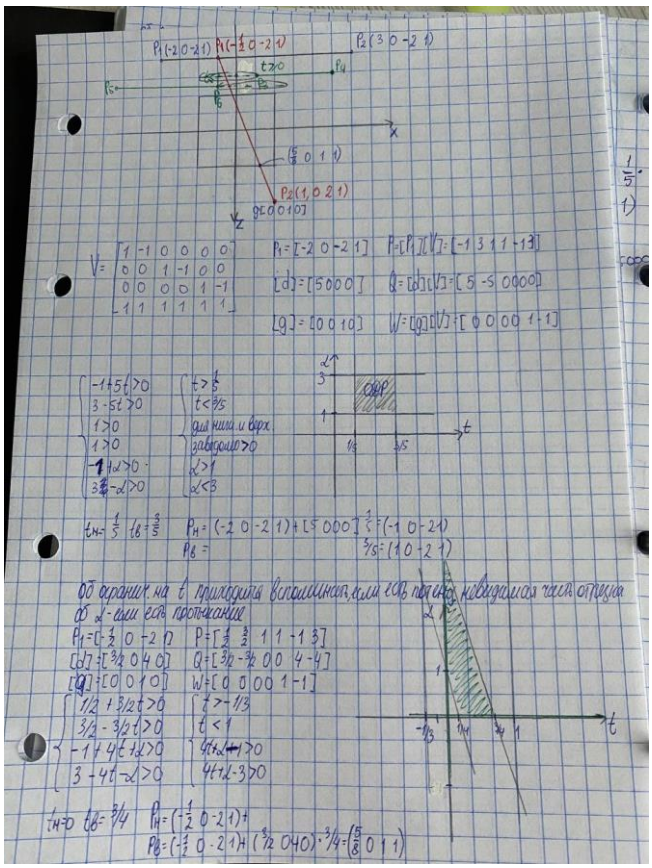
Система неравенств задаёт область допустимых решений. Все точки, расположенные внутри области, дают координаты невидимых точек отрезка. Необходимо найти в этой области минимальное и максимальное значения параметра t - задает координаты начала и конца невидимой части отрезка и подставить в уравнение отрезка для нахождения точек.

Об ограничениях на t приходится вспоминать, если есть потенциально невидимая часть отрезка, об ограничениях на a – если имеет место протыкание (находим точки протыкания на оси a как min и max, так как они принадлежат и грани, и отрезку, а отрезок при a=0). Если в системе уравнения, которые выполняются при любых значениях – заведомо виден относительно этих граней.

Задача легко решается графически: строим область допустимых решений в координатах (t, a) и находим минимальное и максимальное значения для t.

Задача сложно решается программно: всего уравнений n + 3 (где n - количество граней, 3-количество ограничений, накладываемых на t и al: al >= 0, 0 <= t <= 1). Неизвестных меньше чем уравнений, поэтому следует воспользоваться следующей процедурой:

1. Сформировать все возможные (n+2)(n+3)/2 системы по 2 уравнения
2. Решить очередное уравнение
3. Полученные корни подставить во все другие уравнения.
4. Если полученные корни являются решениями всех уравнений, то они принимают участие в нахождении tmax, tmin



Этап 3. Удаление невидимых частей новых ребер, появившихся при протыкании тел

Если тела связаны отношением взаимного протыкания, то образуются новые ребра, соответствующие линиям пересечения тел. Необходимо найти их путём соединения точек протыкания между собой. Нет смысла рассматривать ребра, соединяющие невидимые точки протыкания.

Вновь полученные рёбра надо проверить на экранирование самими телами, связанными отношением протыкания, оставшиеся видимые части ребер надо проверить на экранирование другими телами сцены.

Видимые отрезки образуют структуру протыкания.

29. Удаление невидимых линий и поверхностей в пространстве изображений. Алгоритм Варнока (разбиение окнами): последовательность действий и основные принципы.
30. Типы многоугольников, анализируемых в алгоритме Варнока. Методы их идентификации.

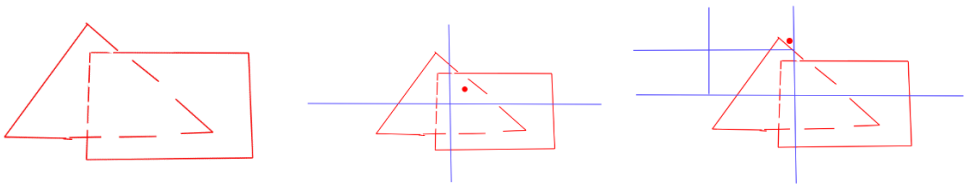
1. Алгоритм работает в пространстве изображения. может работать как с ребрами (ячейка пуста, если в ней нет ребер), так и с поверхностями (анализ поверхностей и заливка нужным цветом).
2. Человеком тратится малое количество времени на обработку тех областей, которые содержат мало информации, большая часть времени и труда затрачивается на области с высоким информационным содержимым.
3. Делается попытка извлечь преимущество когерентности изображения: смежные пиксели вдоль обеих осей x и y имеют тенденцию к однородности.

Идея алгоритма

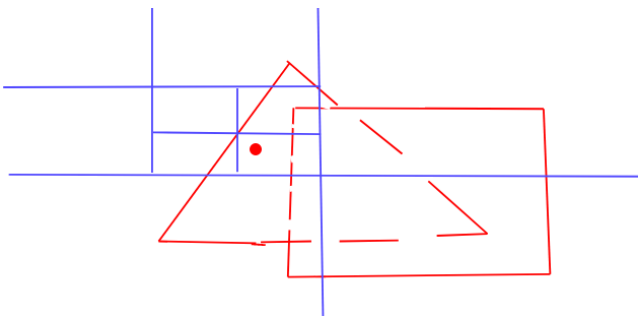
1. Начальное окно имеет размеры экрана. Основная задача состоит в том, чтобы отобразить элементы которые являются видимыми. Необходимо найти ответ на вопрос "Что изображать в очередном окне?".
2. В пространстве изображения рассматривается окно и решается вопрос: оно пусто или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты, пока содержимое окна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация в нем усредняется, и результат изображается с одинаковой интенсивностью или цветом
3. Единой версии алгоритма не существует: зависит от метода разбиения окна и деталей критерия того, что содержимое окна достаточно простое.
4. Оригинальный и самый простой способ - использовать прямоугольные окна, каждый раз делить непустые окна на 4 одинаковых, пока не дойдем до 1 пиксела log2(разрешение) шагов). Тогда определяем глубину каждого из рассматриваемых многоугольников в этой точке. Изображаем точку многоугольника наиболее близкого к пользователю. Окно является пустым, когда все многоугольники сцены являются внешними по отношению к этому окну.
5. В более сложных версиях ставится задача определения, что изображать в очередном окне, для размера окна больше 1 пикселя. Вейлер и Азертон делят на полигональные подокна.
6. Для устранения лестничного эффекта можно продолжить деление на части, меньшие размера пикселя, и усреднить найденные характеристики. Например, в пикселе была найдена часть многоугольника. Разобьем пиксель на 4 части и выясним, что многоугольник лежит только в одной четверти (в прочих пусто). значит, необходимо закрасить пиксель с интенсивностью, равной 1/4.

Пример

Имеется прямоугольник и треугольник.



Отсекатель - окно. Закрашиваем вершину треугольника.



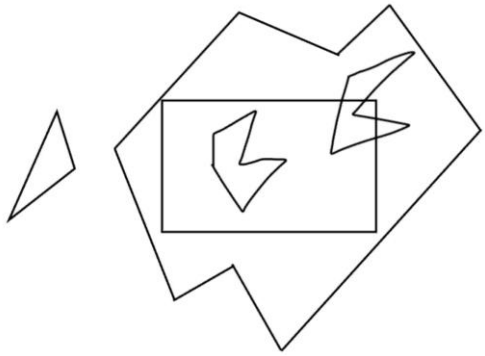
Классификация многоугольников, рассматриваемых в алгоритме Варнока

Способы расположения многоугольников относительно окна:

1. внешний - целиком вне окна
2. внутренний - целиком внутри окна

3. пересекающий - пересекает границу окна
4. охватывающий - окно находится целиком внутри многоугольника

с окном связано несколько многоугольников (внутренние, пересекающие, охватывающие)



правила обработки окна

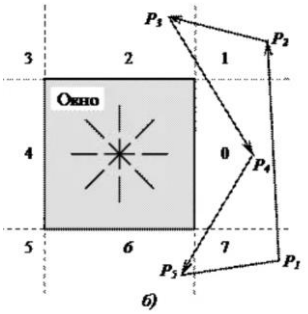
1. Если все многоугольники - внешние: окно закрасить цветом фона
2. Если внутри окна только один многоугольник: площадь окна вне него заполняется фоновым цветом, а сам многоугольник - своим цветом (растровая развертка).
3. Если только 1 многоугольник пересекает окно – отсечение по границе окна и п. 2
4. Если окно охвачено только 1 многоугольником и в окне нет других многоугольников: залить окно цветом многоугольника
5. Если найден хотя бы один охватывающий и он ближе других к точке наблюдения: заполнить окно цветом охватывающего многоугольника. (ключ к удалению невидимых поверхностей)

В противном случае – разбиение.

методы определения расположения

1. Габаритная проверка – внутренний и внешний (не все - огибающие)
   В случае прямоугольных окон, произвольный многоугольник с прямоугольной оболочкой

- внутренний - x_min >= x_left & x_max <= x_right & y_min >= y_down & y_max <= y_up.
- внешний (не все) x_max < x_left || x_min > x_right | y_max < y_down | y_min > y_up



2. Пробная функция – пересекающий или (внешний/охватывающий)

Для прямой, проходящей через ребро отсекателя P1P2: Fпроб=y-mx-b для невертикальных, Fпроб=x-x1 для вертикальных

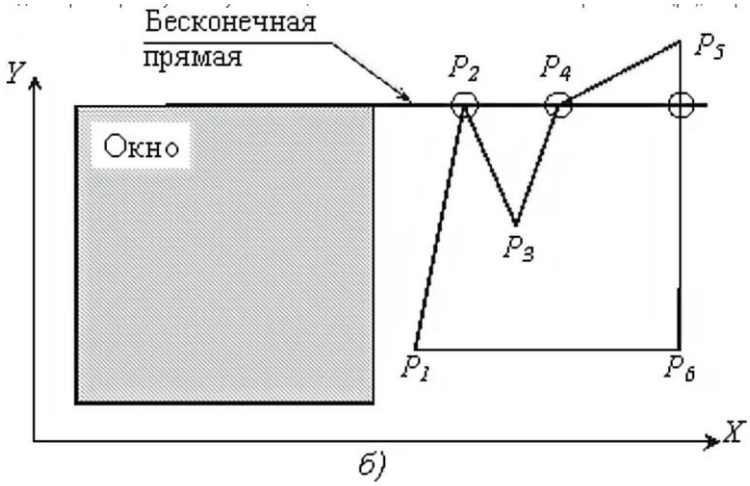
подставляем все вершины окна во все уравнения прямых, задающих ребра многоугольника. если знак пробной функции не зависит от выбора вершины окна - нет

точек пересечения с ней. если пересечение возможно, оно находится и проверяется на принадлежность конкретно ребру, а не его продолжению.

Если ни одно из ребер не пересекает окна, то многоугольник внешний или охватывающий.

3. тест с бесконечной прямой - внешние и охватывающие

проводим луч из любой точки окна (например, из угла в бесконечность). считаем число пересечений луча с заданным многоугольником. если число четное (или 0) - внешний, иначе - охватывающий. Если луч проходит через вершину многоугольника, анализ: считаем касание как 2 пересечения (p2), и протыкание - как одно (p4), или меняем угол прямой

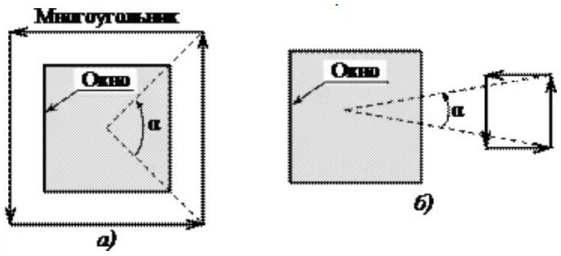


4. тест с подсчетом угла - внешние и охватывающие

   обход по ребрам многоугольника по/против часовой стрелки. суммируем углы, образованные лучами, начинающимися в любой точке окна (например, в середине) и проходящими через концы ребра (с учетом направления). анализ полученной суммы:

сумма = 0: многоугольник внешний

сумма = +-360*n - охватывает окно n раз (многоугольник без пересечений – только 1 раз)



Необязательна точность вычисления каждого угла: можно считать только целые октанты (углы по 45), покрытые отдельными углами. Нумеруем октанты от 0 до 7 справа против чс. Число целых октантов, покрытых углом равен разности номеров октантов, в которых лежат концы ребер.

Для каждого ребра считаем da. Если da > 4, то da -= 8, Если da <- 4, то da += 8, Если da = 0, то ребро многоугольника расщепляется ребром окна, и процесс повторяется с парой полученных отрезков.

анализ: сумма разностей октантов = 0 - внешний, = +-8n - охватывающий

Иерархическое использование 4 методов дает дополнительный выигрыш.

Если используется простейший алгоритм Варнока, то не надо определять внутренние или пересекающие (разбиением превращаются во внешние или охватывающие). Первая проверка выявляет внешние, но частный случай не решает. Нужен более мощный способ определения пустоты окна или охвата многоугольником, на более ранних этапах, чем 1 пиксель:

1. Габаритный тест – пустые и с 1 внутренним многоугольником сразу изображаются.

2. Простой тест на пересечение окна одним многоугольником – отсекается и отображается

3. Более сложные тесты для внешних и охватывающих – дополнительные пустые окна и окна, охватываемые одним многоугольником изображаются

Далее –попытка обнаружить ближайший к точке наблюдения охватывающий многоугольник или разбиение окна (и первый вариант откладывается до достижения уровня пикселов). В любом случае придется проводить тест глубины.

5. Тест глубины - поиск ближайшего охватывающего многоугольника.

Сравниваем глубину z плоскостей, несущих многоугольники, в угловых точках окна. Охватывающий многоугольник с наибольшей глубиной экранирует все остальные в этом окне – заполняем цветом многоугольника (достаточно, но не необходимо (у)).

Глубину несущей плоскости в углу окна можно вычислить с помощью ее уравнения (см. разд. 4.3 и пример 4.3). Например, если уравнение этой плоскости имеет вид

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

а координаты угловой точки окна равны (x_w, y_w), то значение

$$z = -(d + ax_w + by_w)/c \qquad c \neq 0$$

Оптимизации

Предполагалось, что все многоугольники следует сравнивать с каждым окном. Проводим сортировку по приоритету глубины z (ближайшей к наблюдателю вершины многоугольника). Первый – ближайший к наблюдателю.

Если многоугольник является внешним или охватывающим по отношению к текущему окну, то то же самое – для его подокон, его можно не рассматривать для них. Храним 3 списка: охватывающих (обрабатывается первым для нахождения ближайшего к наблюдателю), (пересекающих и внутренних) (не экранируются ли они все охватывающим), внешних (игнорируется). По ходу многоугольники включаются или удаляются из списков, а также запоминается шаг разбиения, на котором многоугольник впервые попал в этот список (надо при обходе по дереву из подокон).

31. Алгоритм Вейлера-Азертона удаления невидимых линий и поверхностей.

1. Удаление невидимых линий и поверхностей
2. В пространстве объекта.
3. Многоугольники плоские – трёхмерная задача сводится к двухмерной.
4. Базируется на их отсечении многоугольника, который позволяет провести как внешнее, так и внутреннее отсечение.
5. Этот алгоритм справляется с циклическим перекрытием многоугольников (часть лежит позади другого, а часть спереди).

6. Если многоугольники протыкаются, то придется один из многоугольников разбить на два линией пересечения этих многоугольников.
7. Цель – минимизировать количество шагов в алгоритме разбиения типа алгоритма Варнока путем разрезания окна вдоль границ многоугольника.

Алгоритм
состоит из 4 шагов

1. Предварительная сортировка по глубине.
   Формируется список приблизительных приоритетов. Наблюдатель на +бск z, многоугольник с максимальной zmax – ближайший и загораживает остальные, используется в качестве отсекателя.

2. Отсечение многоугольников сцены (включая отсекатель) по границам отсекателя («сортировка на плоскости»).
   С этого момента ведется работа с проекциями на плоскость XOY. Формируются 2 списка – внутренний (части отсекаемых внутри отсекателя) и внешний (вне).

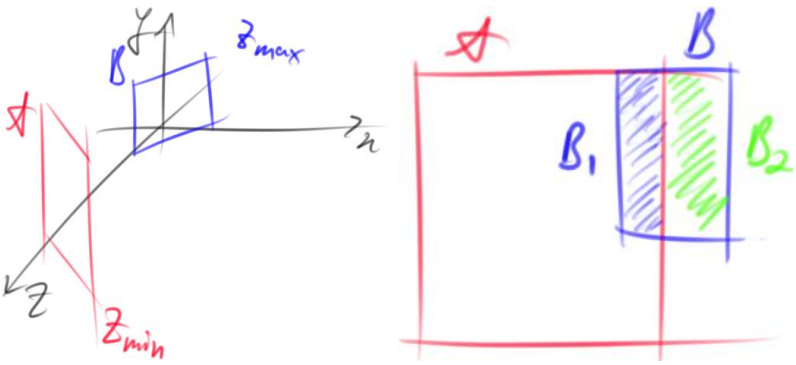
3. Удаление многоугольников, экранируемых ближайшим к наблюдателю многоугольником.
   Для каждого многоугольника внутреннего списка: если его zmax <= zmin отсекающего, то они экранируются им и удаляются. Если все внутренние были удалены, то работа продолжается с внешним списком. В противном случае нужен 4 пункт.

4. Рекурсивное разбиение плоскости OXY с помощью многоугольника, нарушившего порядок, в качестве отсекателя
   Берется копия исходного многоугольника, а не отсеченная часть. Отсечению подлежат многоугольники из внутреннего списка, включая предыдущий отсекатель.

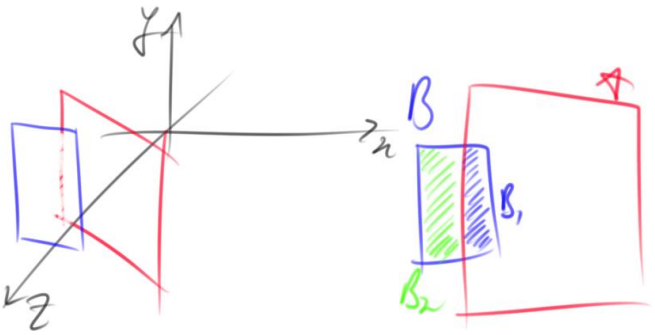
Изображаются многоугольники из внутреннего списка (остаются только отсекающие многоугольники) и работа продолжается с внешним списком

Примеры

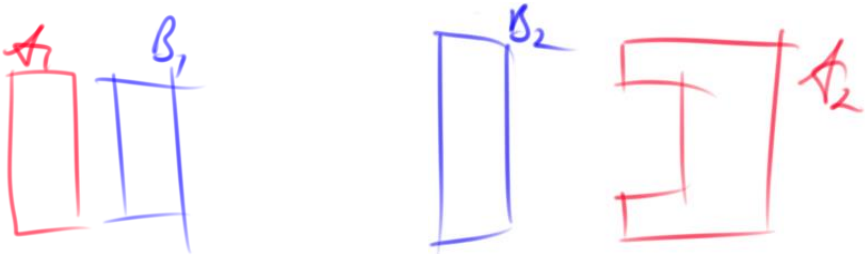
1. *внутренние: A, B1; внешние: B2. ZmaxB1 < ZMinA => изображается A.*



2. *A – отсекающий. Внутренние: A, B1; внешние: B2.*
   ZmaxB > ZminA => A изображать сразу нельзя, нужен 4 пункт:



Список внутренних прямоугольников: A1, B1; внешних – A2 и B2



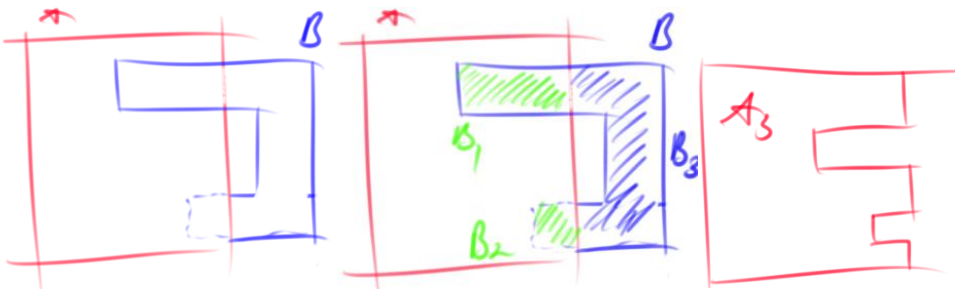
A2 и B2 являются внешними по отношению друг к другу; изображаются оба

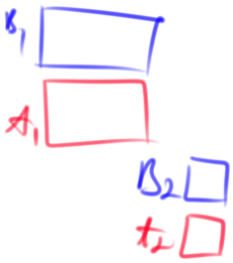
ZmaxA1<=ZminB1, изображается многоугольник B1.

Алгоритм достаточно просто справляется с циклически перекрывающимся многоугольником (лежит то впереди, то сзади). В рекурсивном разбиении необходимости нет. Вся закрытая часть уже удалена на предыдущем этапе, теперь достаточно отсечь исходный по границам циклического.

Чтобы избежать рекурсивного разбиения, создается список многоугольников, которые уже использовались в качестве отсекателей. Если при рекурсивном разбиении отсекающий вновь туда попадает, значит обнаружен циклически перекрывающийся многоугольник

3. Пусть ZmaxA > ZmaxB. Отсечение по границам A, список внутренних: A, B1, B2, список внешних: B3.
   A3 и B3 являются внешними по отношению друг к другу, изображаются оба.
   4 Отсекаем по грани B: внутренние A1 B1 A2 B2 внутренние; внешний: A3





Z_A1max<=ZB1min - изображается B1.

Z_B2max <= ZA2min - изображается A2

вопрос

32. Алгоритм, использующий Z-буфер.

- Один из простейших алгоритмов удаления невидимых поверхностей
- работает в пространстве изображения.

Используются 2 буфера:

Буфер кадра (регенерации) используется для запоминания атрибутов (интенсивности/цвета) каждого пикселя в пространстве изображения.

Z-буфер (буфер глубины), используемый для запоминания координаты z каждого видимого пиксела.

Алгоритм:
В качестве предварительного шага, там, где это целесообразно (для выпуклых многогранников) удалить нелицевые грани (см. алгоритм Робертса)

1. Заполнить буфер кадра фоновым значением цвета.
2. Заполнить Z - буфер минимальным значением Z.
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке
   Для каждой точки многоугольника (x, y) вычислить ее глубину Z(x, y)
   Сравнить глубину Z(x, y) со значением Zбуф(x, y). Если Z(x, y)>Zбуф(x, y), то Zбуф(x, y)= Z(x, y) и занести в буфер кадра на соответствующую позицию цвет текущего многоугольника. Иначе – ничего.

   Здесь можно построить сечение поверхности плоскостью z = const. Для этого изменяется операция сравнения: если (Z(x, y) > Z_буф(x, y)) И (Z(x, y) <= Z_сек.плоск)

Достоинства
1. Простота. Сцены могут быть любой сложности
2. Элементы сцены не нужно предварительно сортировать по глубине
3. Фиксированный размер окна - линейный рост времени работы алгоритма
4. позволяет построить сечение плоскостью z = const

Недостатки
1. Большой объём памяти
   Из-за занесения пикселов в произвольном порядке - трудоёмкость реализации:

2. эффектов прозрачности и просвечивания. Для построения прозрачных поверхностей можно применять модифицированный вариант Z - буфера ALFA - буфер. Если рассматриваются монохромные, почти прозрачные поверхности, то достаточно наряду с Z - буфером иметь ещё один буфер с информацией о текущей интенсивности пикселя.

3. устранения лестничного эффекта. Осуществить префильтрацию сложно, легче постфильтрация (усреднение подпикселов) - считаем в изображении с более высокой разрешающей способностью, потом усредняем.

вычисление глубины z

Уравнение плоскости, несущей грань многоугольника: Ax+By+Cz+D=0, откуда $Z = -\frac{Ax+By+D}{C}$, C! =0. Если C==0, то плоскость расположена параллельно oz – отрисовывать нужно линию или вершину. Для сканирующей строки y=const. Поэтому глубина пиксела на этой строке, у которого x1=x+deltaX, $\Delta x = 1$ (шаг растра) изменяется так:

$$z1 - z = -\frac{Ax1 + By + D}{C} + \frac{Ax + By + D}{C} = -\frac{A}{C}(x1 - x) = -\frac{A}{C}; \quad \Delta z = -\frac{A}{c}$$

Z1=Z – A/C

$\Delta Z(x, y) = -\frac{A}{c}\Delta X - \frac{B}{C}\Delta y.$

Глубина пикселя, являющегося пересечением сканирующей строки с ребром многоугольника

Сначала определяют ребра грани, вершины которых лежат по разные стороны от сканирующей строки. Затем из найденных точек пересечения выбирают ближайшую к наблюдателю.

Глубину определяют по соотношению (видимо, если C=0) :

z3 = z2 + (y3 - y2) / (y2 - y1) * (z2 - z1), где

(y1, z1), (y2, z2) - координаты вершин проекции ребра на плоскость YOZ.

(x3, z3) - координаты проекции точки пересечения на ту же плоскость.

33. Алгоритм, использующий список приоритетов.

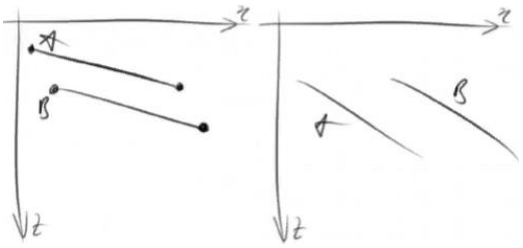
(алгоритм художника). Работает как в пространстве объекта, так и в пространстве изображения.

Основная идея – объекты отрисовываются, начиная с наиболее удалённых от наблюдателя, до ближайших.

Основная задача: перед изображением объекта определить, может ли очередной многоугольник загораживать многоугольники, отображаемые в дальнейшем. Если не может загораживать другие, то он отрисовывается; если может, то проводятся расчёты.

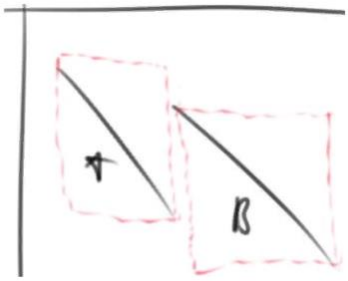
Алгоритм

1. отсортировать многоугольники сцены по возрастанию глубины (по возрастанию Zmin многоугольника). Первый – самый дальний от наблюдателя
2. проверка возможности изображения очередного многоугольника на основе сравнения глубин текущего A со следующим B. если ZmaxA < ZminB, то заносим A в буфер кадра и переходим к обработке следующего многоугольника. Иначе A потенциально экранирует B и любой другие многоугольник типа B из списка. Тем самым образуется множество {B}

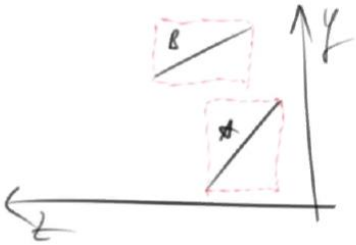


3. Чтобы это выяснить, проводится серия тестов, расположены по возрастанию вычислительной сложности, каждый формулируется как вопрос. Если ответ положительный – A можно отображать без проведения последующих тестов.

3.1 Верно ли, что прямоугольные объемлющие оболочки A и B не пересекаются по оси X?



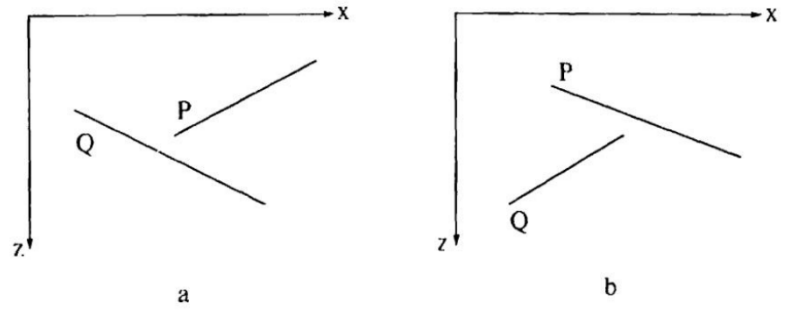
3.2 – " -, по оси У?



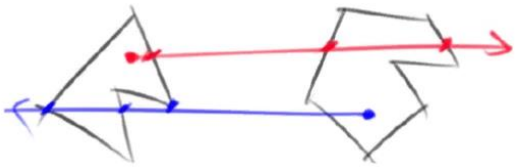
3.3 A целиком лежит по ту сторону плоскости, несущей B, которая дальше находится от наблюдателя (по невидимую сторону плоскости B)?

Задается уравнение плоскости через B: Ax+By+Cz+D=0 (по координатам вершин) и формируется пробная функция f=ax+by+cz+d. Подставить координаты всех вершин A в пробную функцию – одинаков ли знак (многоугольник лежит по одну сторону); сравнить знак со знаком в точке, положение которой относительно B известно.

3.4 B лежит по ту сторону плоскости A, которая ближе находится к наблюдателю?

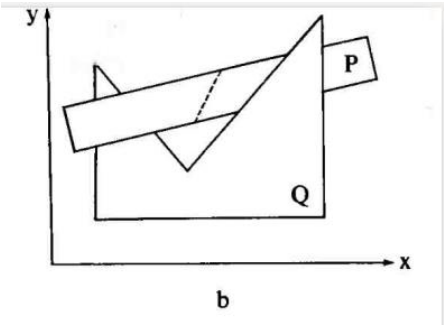


3.5 проекции A и B не пересекаются? (алгоритм с лучом; проверка на одинаковость чётности)



Каждый из этих тестов применяется к каждому элементу {B}. Если ни на один из тестов не получен положительный ответ, то A может закрывать B. Многоугольники следует поменять местами в списке и зафиксировать факт обмена. Выполнить проверки заново. Если во втором случае также не получено положительного ответа, значит обнаружена ситуация циклического экранирования. В этом случае A надо разбить плоскостью, несущей B удалить A из списка, а его части – занести. Затем тесты повторяются для нового списка. Этот шаг предотвращает зацикливание алгоритма.

При циклическом экранировании (пример)



Алгоритмом отсечения многоугольников Сазерленда-Ходжмена разбиваем многоугольники вдоль линии пересечения плоскостей, несущих эти многоугольники.

Плоскость с Q – секущая. Каждое ребро P отсекается плоскостью Q с помощью алгоритма отсечения Кируса-Бека.

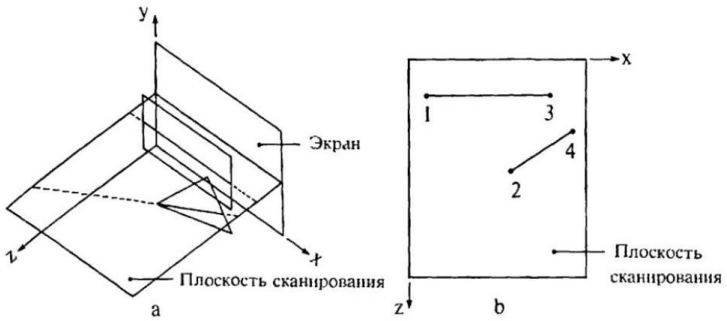
Оценка эффективности

Может не хватить мощности ЭВМ для динамического вычисления.

В большинстве случаев, картинка статична. Можно предварительно вычислять некоторые более общие приоритетные характеристики (алгоритм Шумейкера, там вообще шок какой-то).

34. Алгоритм построчного сканирования, использующий Z-буфер. Интервальные методы построчного сканирования (основные предпосылки).

1. Удаление невидимых поверхностей в пространстве изображения.
2. Элементы сцены обрабатываются в порядке прохождения скан-плоскости, ее пересечение с трехмерной сцены дает окно высотой в 1 скан-строку и шириной во весь экран, где и решается задача удаления невидимых поверхностей.
3. Буфер размером в одну сканирующую строку. Поочередно рассматриваются сканирующие строки, для очередной строки рассматриваются все многоугольники (что не эффективно, поэтому п. 4), в конце буфер выводится на экран
4. Можно использовать обобщение методов растровой развертки: применяется групповая сортировка по y, список активных многоугольников и список активных ребер



Содержимое САР

Для каждой пары ребер многоугольника, которые пересекаются сканирующей строкой

Xл, Xп – пересечения левого и правого ребра с текущей сканирующей строкой

ΔXл, ΔXп – приращение Xл и Xп в интервале между соседними сканирующими строками

ΔYл, ΔYп – число сканирующих строк, пересекаемых левым и правым ребром

Zл-глубина многоугольника в левом ребре

ΔZx = -A/C для C!=0 (иначе ΔZx=0) – приращение по Z вдоль сканирующей строки

ΔZy = -B/C для C!=0 (иначе ΔZy=0) – приращение по Z в интервале между соседними сканирующими строками

Пары активных ребер многоугольника заносятся в произвольном порядке, в пределах одной пары упорядочены слева направо, для одного многоугольника может оказаться более 1 пары активных ребер (невыпуклый)

Алгоритм

Подготовка информации:

1. Для каждого многоугольника определить самую верхнюю сканирующую строку, которую он пересекает.
2. Занести многоугольник в группу y, соответствующую этой сканирующей строке
3. Запомнить для каждого многоугольника (в связном списке): Δy – число строк, пересекающих этот многоугольник, список ребер многоугольника, коэффициенты A, B, C, D уравнения плоскости многоугольника, визуальные атрибуты многоугольника

Решение задачи удаления невидимых поверхностей

1. Инициализировать буфер кадра дисплея
2. Для каждой сканирующей строки:
   2.1. Инициализировать буфер кадра размером с одну сканирующую строку, заполнив его фоновым изображением
   2.2. Инициализировать Z-буфер размером с одну сканирующую строку значением Zmin
   2.3. Проверить появление в группе у сканирующей строки новых многоугольников, добавить все новые многоугольники в список активных многоугольников (САМ)
   2.4. Если было добавление многоугольника в САМ, то добавить все пары активных ребер новых многоугольников в САР (сформировать информацию).
   2.5. Если произведено удаление какого-либо элемента из пары ребер САР, то проверить необходимость удаления всего многоугольника из САМ. Если он остается, то укомплектовать пару активных ребер этого многоугольника в САР) (если удалилось 1 - добавить 1, если 2 - добавить 2). В противном случае удалить и второй элемент этой пары из САР
   2.6. Для каждой пары ребер из САР (в произвольном порядке):
      2.6.1. Извлечь пару из САР
      2.6.2. Инициализировать Z значением Zл
      2.6.3. Для каждого пикселя Xл ≤ X ≤ Xп: вычислить Z(x, y=const). Сводится к вычислению приращения Z1=Zл, ..., Zk = Zk-1+ ΔZx
      Если Z(x, y)>Zбуф(x, y), то Zбуф=Z и занести атрибуты многоугольника в буфер кадра
   2.7. Записать буфер кадра сканирующей строки в буфер кадра дисплея
   2.8. Скорректировать САР: для каждой пары
      2.8.1. ΔYл-=1, ΔYп-=1
      2.8.2. Если ΔYл < 0 или ΔYп < 0, то удалить ребро из списка, пометив положение обоих ребер в списке и породивший их многоугольник
      2.8.3. Xл=Xл + ΔXл, Xп=Xп + ΔXп
      2.8.4. Zл=Zл + ΔZxΔX + ΔZy*ΔY(=1)

   2.9. Скорректировать САМ
      2.9.1. ΔY-=1
      2.9.2. Если ΔY < 0 (значит удалялось и ребро), то удалить многоугольник из САМ

Интервальные методы построчного сканирования

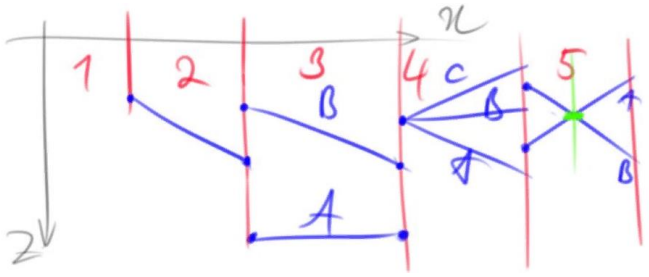
В алгоритме построчного сканирования с использованием Z-буфера глубина многоугольника вычисляется для каждого пикселя на сканирующей строке. Количество вычислений можно сократить, если использовать понятие интервалов. Решение задачи удаления невидимых поверхностей сводится к выбору видимых отрезков в каждом интервале, полученных путем деления сканирующей строки проекциями точек пересечения ребер

1. Пустой интервал. Изобразить фон

2. Один многоугольник, изобразить его атрибуты

3, 4. несколько многоугольников, но один лежит ближе других, изобразить атрибуты многоугольника с MAX Z

5. sign (z1н-z2н)!=sign (z1k-z2k) – условие пересечения. разбить интервал точкой пересечения



35. Алгоритм определения видимых поверхностей путем трассировки лучей.

Трассировка лучей - метод грубой силы (не учитывает специфику обрабатываемой сцены, например, когерентность, которую использовали остальные).

Прямая трассировка: некоторый источник света испускает свет во все стороны, он попадает на объект и затем каким-то путем отражений доходит до наблюдателя. (неэффективно - только небольшая часть дойдет до наблюдателя)

Обратная трассировка (Аппель): отслеживать лучи в обратном направлении, от наблюдателя к объекту (источником считается камера)

Задача решается в пространстве изображений –лучи, испускаемые камерой, проходят через очередной пиксель экрана. Используется матрица пикселей, каждый пиксель является источником луча. Задача сводится к поиску пересечений лучей и поверхностей. Необходимо проверить пересечение каждого объекта с каждым лучом и найти ближайший пересекаемый многоугольник.

Простая трассировка: наблюдатель на +бск z, все лучи || z, Каждый луч проходит через пиксель на растре до сцены.

Трассировка с учетом перспективы: точка расположена не в бесконечности, растр также перпендикулярен OZ, нужно построить одноточечную центральную проекцию на картинную плоскость

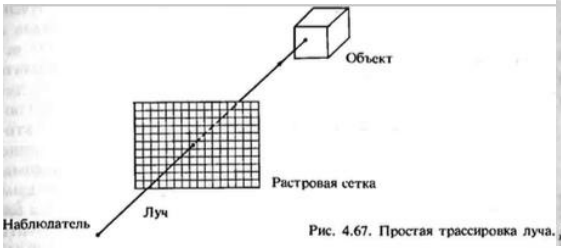


Рис. 4.67. Простая трассировка луча.

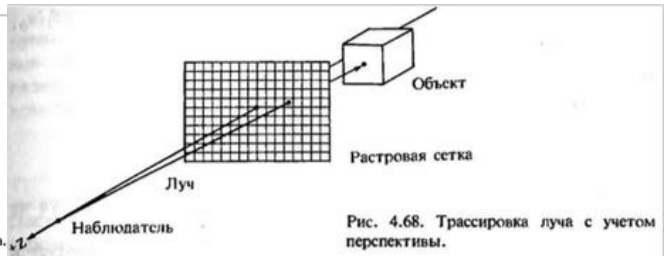


Рис. 4.68. Трассировка луча с учетом перспективы.

Определение пересечений

Нужно вычислять множество пересечений (около 75 - 95% всего времени). Поэтому для ускорения процесса нужно исключить из процесса заведомо лишние объекты.

Одно из решений - погружение объектов в выпуклую "оболочку". Например, сферическую (чаще) или в форме параллелепипеда. Если луч не пересекает оболочки, то объект можно откинуть, и не искать пересечения луча и этого объекта.

Сферический тест

самый простой, но может оказаться неэффективным. Если расстояние от центра сферической оболочки до луча превосходит радиус этой сферы, то луч не пересекает оболочки.

Используем параметрическое уравнение прямой, проходящей через P1(x1, y1, z1) и P2(x2, y2, z2).

$$P(t) = P_1 + (P_2 - P_1)t$$

с компонентами

$$x = x_1 + (x_2 - x_1)t = x_1 + at$$
$$y = y_1 + (y_2 - y_1)t = y_1 + bt$$
$$z = z_1 + (z_2 - z_1)t = z_1 + ct$$

D^2=(x1+at− xc)^2 +(y1+bt− yc)^2+(z1+ct−zc)^2
Находим первую производную по t, приравниваем к 0 – условие минимума, выражаем tmin

$$t = -\frac{a(x_1 - x_0) + b(y_1 - y_0) + c(z_1 - z_0)}{a^2 + b^2 + c^2}$$

Если $d^2 > R^2$, где R — радиус сферической оболочки, то луч не может пересечься с объектом.

Габаритный тест (параллелепипед)

Много вычислений в 3d (проверить пересечение по меньшей мере с тремя бесконечными плоскостями), поэтому сводим тест к сравнению знаков. Переносами и поворотами совмещаем трассирующий луч с OZ, и то же преобразование применяем к оболочке. Луч пересекает оболочку, если в новой СК знаки xmin!=xmax и ymin !=ymax (оболочка охватывает центр), тогда пересечение расположено на x=y=0

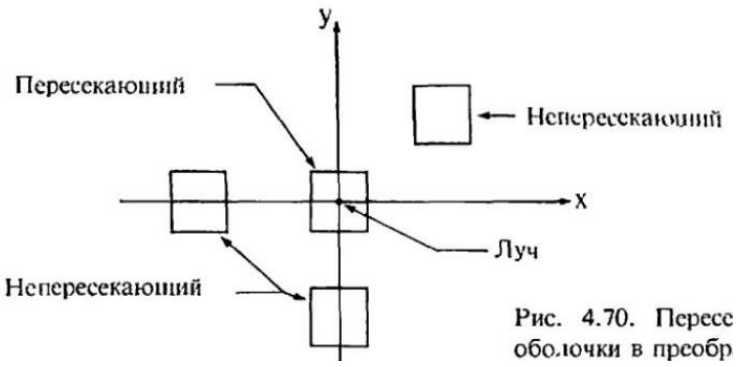


Рис. 4.70. Пересе
оболочки в преобр

Алгоритм

I Создание списка объектов.

- Полное описание объекта (тип, поверхность, характеристики)
- Описание сферической оболочки (центр и радиус)
- Флаг прямоугольной оболочки (=1, если нужны габаритные тесты с прямоугольной оболочкой. Например, для сферы он не нужен)
- Описание прямоугольной оболочки если нужно (xmin, xmax, ymin, ymax, zmin, zmax)

II Для каждого трассируемого луча:

1. Для каждого объекта выполнить тест со сферической оболочкой. Если луч пересекает ее, то занести объект в САО (список активных объектов)
2. Если САО пуст, изобразить пиксель цветом фона. Иначе находим преобразования, совмещающие трассируемый луч с осью Z и запоминаем это преобразование
3. Для каждого объекта из САО:
   3.1. Если флаг прямоугольной оболочки = 1, преобразовать эту оболочку в СК луча и выполнить габаритный тест. Если пересечения нет, то перейти к следующему объекту.
   3.2. преобразовать объект в СК луча, определить его пересечения с лучом, если они есть. Занести все пересечения в список пересечений.
4. Если список пересечений пуст, то ставим пиксель цветом фона. Иначе определить zmax для списка пересечений, вычислить обратное преобразование и с его помощью – точку пересечения в исходной СК. Нарисовать пиксель с атрибутами пересеченного объекта и соответствующей модели освещения.

(для простых непрозрачных поверхностей не нужно определять пересечение в исходной СК, если в алгоритм модели освещения не нужно включать свойства или ориентацию поверхности)

Возможные модификации

1. Кластерные группы пространственно-связанных объектов: вводим сферические оболочки для групп (наборов) связанных между с собой объектов, обрабатываем все в иерархическом порядке.

2. Упорядочение по приоритету (например, центр сферической оболочки или zmax). САР (по идеям алгоритма со списком приоритетов), далее обычная трассировка для активных объектов.

36. Построение реалистических изображений. Физические и психологические факторы, учитываемые при создании реалистичных изображений. Простая модель освещения.

Сводится к расчету интенсивности либо всей грани, либо отдельных пикселов.

Поверхность видим, если она отражает или пропускает свет и частично отражает

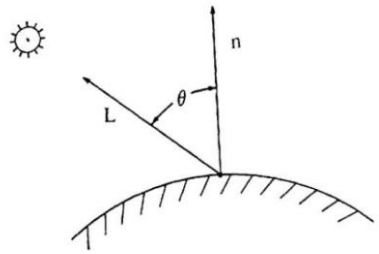
Простая модель освещения

Диффузное, зеркальное и рассеянное отражение

Световая энергия, падающая на поверхность, может быть частично или полностью поглощена (в тепло), отражена или пропущена. Количество энергии на эти цели зависит от длины волны цвета. Объект можно увидеть, если он отражает или пропускает свет, а если все поглощает – абсолютно черное тело. Цвет объекта определяется поглощаемыми длинами волн

Свет, отраженный от объекта, может быть диффузным (происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. Такой свет рассеивается равномерно по всем направлениям - положение наблюдателя не имеет значения) и зеркальным (Происходит от внешней поверхности объекта)

Диффузное отражение



Поверхность поглощает всю падающую энергию, а затем как источник света ее отдает внутренними слоями. Равномерное по всем направлениям. Цвет отраженного цвета совпадает с цветом материала.

Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта**: I = I_l * k_d * cos θ, 0 <= θ <= pi / 2**, где I - интенсивность отраженного света, I_l - интенсивность точечного источника, k_d - коэффициент диффузного отражения (0<= k_d<=1, зависит от длины и волны света, но в простых моделях освещения считается постоянным.), θ - угол между нормалью к поверхности и направлением света (если >pi/2, значит источник расположен за объектом)

Рассеянное отражение

При таком подходе поверхность выглядит блеклой и матовой, но на объекты реальных сцен падает еще и рассеянный свет, отраженный от окружения. Для упрощения он считается константой k_ a: **I = I_a * k_a + I_l * k_d * cos θ**, где I_a - интенсивность рассеяного света, k_a - коэффициент диффузного отражения рассеянного света. (тоже 0<= k_a<=1)

Расстояние

Из физики – квадрат расстояния, но здесь приходится подругому длля реалистичности. Мы считаем, что мы в бесконечности, но тогда интенсивность 0. Если же мы на конечном расстоянии, то если 2 поверхноссти на небольшом расстоянии от источника, их иинтенсивности сидьно отличаются, что не соответсвует нашему восприятию. Поэтому решили учитывать не квадрат, а 1 степень. При этом считают, что ближайшая поверхность освещена максимально, и дальше по убыванию. При центральном проецированиии – в центре.

Чтобы учитывать расстояние до объекта, опытном путем достигли большей реалистичности при линейной зависимости затухания интенсивности. **I = I_a * k_a + I_l * k_d * cos θ / (d + K)**, где K - произвольная постоянная, выбираемая на основе эксперимента, d - расстояние от центра проекции до объекта.

Если наблюдатель в бесконечности, то d определяется положением ближайшего к нему объекта. Для цветных поверхностей модель освещения применяется к каждому из 3 цветов.

Зеркальное отражение света

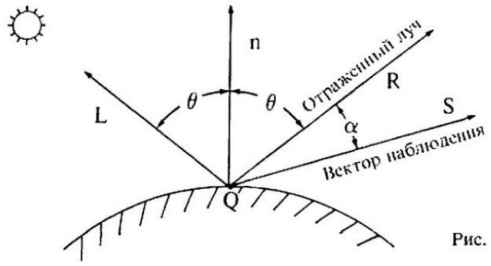


Рис.

Зависит от угла между вектором отражения и вектором наблюдения, является направленным. Нужно, чтобы вектор отражения попал в глаз. Ничего неидеально, поэтому часть энергии рассеивается, но часть все же попадает – а сколько попадает зависит от степени косинуса (а в итоге от свойств вещества)

Интенсивность зависит от угла падения, длины волны и свойств вещества. Такое отражение является направленным.

Угол отражения от идеальной отражающей поверхности равен углу падения, в любом другом положении наблюдатель не видит зеркально отражённый свет. Значит, что $\alpha = 0$ и $S = R$ (S - вектор наблюдателя, R - вектор отражения). Если поверхность неидеально, то чем шероховатей, тем шире распределение.

Благодаря зеркальному отражению на блестящих объектах появляются блики (цвета луча, а не объекта, так как отражается от поверхности), которые перемещаются вместе с наблюдателем.

Формула Фонга для простой модели освещения

$I_s=I_l*w(i, \lambda)*\cos^n(\alpha)$, где $w(i, \lambda)$ - кривая отражения, представляющая отношение зеркально отраженного света к падающему как функцию угла падения i и длины волны λ; n - степень, аппроксимирующая пространственное распределение зеркально отраженного света (чем больше n, тем сфокусированней распределение (блестящие поверхности металлов)). Обычно w - сложная функция, заменяют константой ks, которая определяется экспериментально.

Объединяя с формулой рассеянного света и диффузного отражения: **$I = I_a * k_a + (k_d * \cos\theta + k_s* \cos^n(\alpha))*(I_l / (d + K))$**. Формула называется функцией закраски и применяется для расчета интенсивности или тона точек объекта или пикселов изображения.

Если имеется несколько источников света, то интенсивности суммируются. (аддитивная модель)

$I = I_a * k_a + суммапоj1\text{-}m[(k_d * \cos\theta_j + k_s* \cos^n(\alpha_j))*(I_l_j / (d + K))]$, m - количество источников света.

Применяя формулу скалярного произведения двух векторов, запишем

$$\cos\theta = \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} = \hat{\mathbf{n}}\cdot\hat{\mathbf{L}}$$ где последние- единичные векторы соответственно нормали к поверхности и направления к источнику.

$$\cos\alpha = \frac{\mathbf{R}\cdot\mathbf{S}}{|\mathbf{R}||\mathbf{S}|} = \hat{\mathbf{R}}\cdot\hat{\mathbf{S}}$$

где последние-единичные векторы, определяющие направления отраженного луча и наблюдения. Следовательно, модель освещения для одного источника определяется как

$$I = I_a k_a + \frac{I_t}{d+K}\left[k_d\left(\hat{n}\cdot\hat{L}\right) + k_s\left(\hat{R}\cdot\hat{S}\right)^n\right]$$

Фоновое освещение: обычно есть фоновое освещение (отражение от стен, пола, ночью – луна, абсолютно темно – безумная ночь.

Для простой модели надо уметь рассчитывать нормаль к поверхности и вектор отражения. Косинус – скалярное произведение единичных векторов ->используют единичный вектор.

Что нужно учитывать, при построении реалистических изображений:

1. Оптические свойства поверхностей.
2. Воспроизводить рисунок на поверхности.
3. Воспроизводить неровности.
4. Повернхости отбрасывают тени (учитывать).
5. Восприятие окружающего мира человеческим глазом.

Физические и психологические факторы, учитываемые при создании реалистичных изображений.

Свет, т.е. электромагнитная энергия, после взаимодействия с окружающей средой попадает в глаз, где в результате физических и химических реакций вырабатываются электроимпульсы, воспринимаемые мозгом.

Чувствительность глаза к яркости света изменяется по логарифмическому закону. Пределы чувствительности к яркости чрезвычайно широки, но глаз реагирует на гораздо меньший диапазон значений яркости, распределённый вокруг уровня адаптации к освещённости. Скорость адаптации к яркости неодинакова, но очень высока. Экстремумы диапазона относительной яркости воспринимаются соответственно как чёрный и белый.

1. Явление одновременного контраста.

Глаз приспосабливается к "средней" яркости обозреваемой сцены; поэтому область с постоянной яркостью (интенсивностью) на тёмном фоне кажется ярче или светлее, чем на светлом фоне.

2. Эффект полос Маха

Границы областей постоянной интенсивности кажутся более яркими, в результате чего области с постоянной интенсивностью воспринимаются, как имеющие переменную интенсивность. Эффект полос Маха наблюдается, когда резко изменяется наклон кривой интенсивности. Если кривая интенсивности вогнута, то в этом месте поверхность кажется светлее, если выпукла - темнее.

3. Свойства, связанные с мышлением

Рисунок лестницы, вечно идущей вниз построен так, что мы считаем замкнутой фигуру, которая замкнутой быть не может.

37. Построение реалистических изображений. Метод Гуро закраски поверхностей (получение сглаженного изображения).
38. Построение реалистических изображений. Закраска Фонга (улучшение аппроксимации кривизны поверхности).

Бывает три возможных типа закрашивания:

1. Простая закраска
2. Закраска по Гуро
3. Закраска по Фонгу

Простая закраска.
Для каждой грани используется по 1 нормали, закрашиваем все одним уровнем интенсивности. Используется при выполнении трех условий.

1. Источник находится в бесконечности. Диффузная составляющая зависит от угла падения, все лучи || друг другу-> диффузная составляющая одинакова.
2. Наблюдатель находится в бесконечности. Зеркальная составляющая зависит от вектора наблюдения, все лучи || (угол между отражением и наблюдением равен), одинакова
   Если 1 или 2 не выполняются, то можно просто усреднить
3. Самое важное: закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности. (Сферу аппроксимировали гранями, получили резкий переход, который будет выглядеть как ребро)

Быстро и просто, но возникают ребра. Надо проводить сглаживание на границе многоугольников, аппроксимирующих поверхность.

Формулы
Для Гуро пишем I, для Фонгу - n

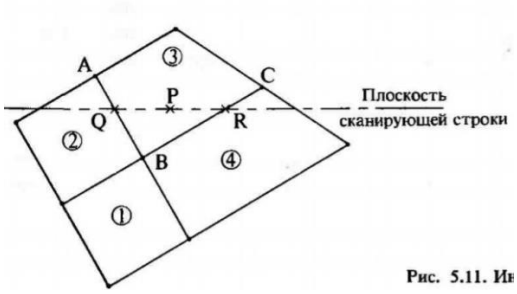


Рис. 5.11. Ив

P=интерполяция (Q, R,) Q-точка пересечения ребра со скан-строкой=интерполяция (A, B)

$$I_Q = uI_A + (1-u)I_B \quad 0 \le u \le 1$$

где u = BQ/BA. Аналогично для получения интенсивности R линейно интерполируются интенсивности в вершинах B и C, т.е.

$$I_R = wI_B + (1-w)I_C \quad 0 \le w \le 1$$

где w = CR/CB. Наконец, линейной интерполяцией по строке между Q и R находится интенсивность P, т.е.

$$I_P = tI_Q + (1-t)I_R \quad 0 \le t \le 1$$

где t = RP/RQ.

Можно повысить эффективность, рассматривая вдоль скан-строки

Значения интенсивности вдоль сканирующей строки можно вычислять инкрементально. Для двух пикселов в t_1 и t_2 на сканирующей строке

$$I_{P_2} = t_2 I_Q + (1 - t_2) I_R$$

$$I_{P_1} = t_1 I_Q + (1 - t_1) I_R$$

Вычитая, получим, что вдоль строки

$$I_{P_2} = I_{P_1} + (I_Q - I_R)(t_2 - t_1) = I_{P_1} + \Delta I\, \Delta t$$

Закраска по Гуро

Биполярная (то есть дважды) интерполяция интенсивности.

Интенсивность линейно зависит от интенсивности на концах

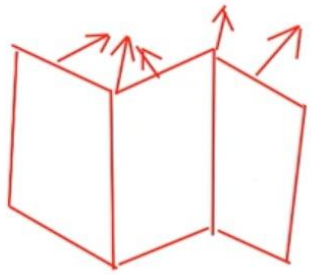
Рассматриваем отдельную грань, вычисляем нормали к вершинам грани, интенсивность каждой вершины, выполняем первую интерполяцию вдоль рёбер. (Линейную - чтобы минимизировать вычисления, можно и другую). Вторая линейная интерполяция выполняется, когда мы вычисляем интенсивности пикселей, расположенных на сканирующей строке.

хорошо сочетается с диффузной составляющей поверхности (будет выглядеть матовой), так как форма бликов для зеркального отражения сильно зависит от выбора многоугольников, представляющих поверхность.

+Качество изображения улучшится. Граница между двумя гранями визуально сгладится.

-Закраска не учитывает кривизны поверхности. Ошибки при изображении невыпуклых, закраска может меняться от кадра к кадру, полосы Маха,

Мы пытаемся сгладить ребра, но можно их и потерять ребра и получить плоское изображение, если закрашивается смежная грань одним уровнем интенсивности, то есть когда углы граней одинаковые. Пример - детская книжка-раскладушка. Нормали (вычисленные по нормалям к граням) одинаковые, интенсивности тоже, каждая из 3х граней будет закрашена одинаковым уровнем. Ребер не увидим.



Решение: Либо случайно внести изменения в полученные значения нормали (чтобы разные интенсивности получились), либо, если речь идет об аппроксимации поверхности, разбить на более мелкие грани - еще один шаг аппроксимации.

Закраска Фонга

Фонг предложил интерполировать не интенсивность, а нормали: в пределах грани нормали от точки к точке меняется, что позволяет учитывать криволинейный характер поверхности.

Закраска по Фонгу хорошо сочетается с зеркальной составляющая. Хорошо моделирует блики, возникающие при зеркальном отражении

+Изображение более реалистичное, учитывается криволинейный характер поверхности.

-трудоемкость закраски будет в 3+ раза выше чем Гуро (надо вычислить все 3 составляющие нормали + там мы сразу вычисляем интенсивность, а в здесь - только после получения нормали в точке), виден разрыв производной. Ошибки при изображении невыпуклых, закраска может меняться от кадра к кадру, полосы Маха

Включает как физические, так и психологические процессы.

При построении реалистического изображения необходимо:

1. Учитывать оптические свойства поверхностей
2. Воспроизводить рисунок на поверхности
3. Воспроизводить неровности
4. Учитывать, что поверхности отбрасывают тени.
5. Учитывать восприятие окружающего мира человеческим глазом.

39. Определение нормали к поверхности и вектора отражения (4 способа) в алгоритмах построения реалистических изображений.

Для простой модели надо уметь рассчитывать нормаль к поверхности и вектор отражения. Косинус – скалярное произведение единичных векторов ->используют единичный вектор.

Нормаль

К поверхности определяет ее локальную кривизну, следовательно, и вектор зеркального отражения

1. Если известно аналитическое описание (уравнение) поверхности (ФДП z = f(x, y), НЕЯВ F (x, y, z) = 0, ПАРАМ F(s, t)) - нормаль вычисляется через градиент в точке: grad(F(x, y, z)). Векторное произведение векторов касательных в этой точке, векторы касательных получаем путем вычисления производных.

2. Если задана лишь полигональная аппроксимация/уравнения плоскостей, проходящие через полигональные грани, то нормаль к их общей вершине равна среднему значению нормалей ко всем многоугольникам, сходящимся в этой вершине.
N = (A1 + ... + An)i + (B1 + ... + Bn)j + (C1 + ... + Cn)k, где буквы - коэффициенты уравнений плоскостей.

3. Если не заданы уравнения плоскостей, нормаль ищется путём усреднения (=суммирование) векторного произведения векторов, построенных на всех ребрах, пересекающихся в вершине.
N = v1v2 + v2v3 + v3v1, v1 - v3 векторы ребер инцидентных вершине в которой ищется нормаль.

Требуется найти только внешние нормали. Если ее не нормировать, то величина зависит от количества и площади (длин) элементов. В 2 и 3 могут быть несколько разные результаты.

Если нормаль к поверхности используется для определения интенсивности и для изображения объекта или сцены выполняется перспективное преобразование, то нормаль следует вычислять до перспективного преобразования. Иначе направление нормали будет искажено, что приведет к неправильному определению интенсивности.

Определение вектора отражения.

Основная идея: по закону отражения вектор падающего света, нормаль к поверхности и вектор отражения **лежат в одной плоскости**, причем на этой плоскости угол падения равен углу отражения.

Первый способ

Это решение подходит для случая, когда свет падает вдоль оси Z (удобно с одним точечным источником). Переносят начало СК в точку поверхности, проекция вектора нормали и отражения (на плоскость xy) будут лежать на одной прямой.

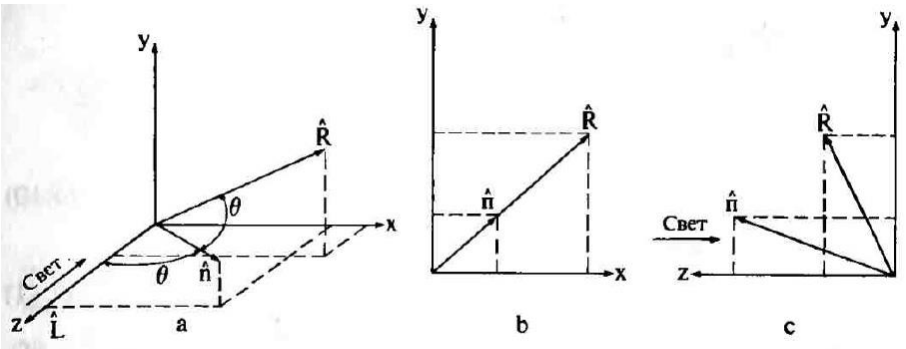


Рис. 5.9. Расчет направления отражения.

Таким образом,

$$\frac{\hat{R}_x}{\hat{R}_y} = \frac{\hat{n}_x}{\hat{n}_y} \tag{5.8}$$

где $\hat{R}_x$, $\hat{R}_y$, $\hat{n}_x$, $\hat{n}_y$ — суть x- и y-составляющие единичных векторов соответственно отражения и нормали.

Обозначим угол между единичным вектором нормали и осью z через θ. Тогда составляющая вектора нормали по оси z есть

$$\hat{n}_z = \cos\theta \qquad 0 \le \theta \le \pi/2$$

Аналогично, угол между единичным вектором отражения и осью z равен 2θ, поэтому

$$\hat{R}_z = \cos 2\theta = 2\cos^2\theta - 1 = 2\hat{n}_z^2 - 1 \tag{5.9}$$

Известно, что

$$R_x^2 + R_y^2 + R_z^2 = 1$$

и

$$\hat{R}_x^2 + \hat{R}_y^2 = 1 - \hat{R}_z^2 = 1 - \cos^2 2\theta$$

или

$$\hat{R}_y^2\left(\frac{\hat{R}_x^2}{\hat{R}_y^2} + 1\right) = 1 - \cos^2 2\theta$$

Используя отношение x- и y-составляющих векторов отражения и нормали (5.8) и соотношение

$$\hat{n}_x^2 + \hat{n}_y^2 + \hat{n}_z^2 = 1$$

получаем

$$\frac{\hat{R}_y^2}{\hat{n}_y^2}(\hat{n}_x^2 + \hat{n}_y^2) = \frac{\hat{R}_y^2}{\hat{n}_y^2}(1 - \hat{n}_z^2) = 1 - \cos^2 2\theta$$

Перепишем правую часть:

$$\frac{\hat{R}_y^2}{\hat{n}_y^2}(1 - \hat{n}_z^2) = 1 - (2\cos^2\theta - 1)^2 = 1 - (2\hat{n}_z^2 - 1)^2 = 4\hat{n}_z^2(1 - \hat{n}_z^2)$$

или

$$\hat{R}_y = 2\hat{n}_z\hat{n}_y \tag{5.10}$$

Из соотношения (5.8) имеем

$$\hat{R}_x = 2\hat{n}_z\hat{n}_x \tag{5.11}$$

Хорош, если источник 1. Иначе куча преобразований для совмещения с oz – долго и сложно.

Второй способ

Если свет не падает по оси z или источников несколько, проще перенести и повернуть нормаль так, чтобы она была параллельна оси Z, а точку P поверхности принять за начало координат. В таком случае, плоскость xy - касательная к поверхности, x и y составляющие единичных векторов падения и отражения будут иметь разные знаки, z составляющая - одинаковая.

Чтобы получить результаты в первоначальной ориентации, надо выполнить обратное преобразование. В перемещенной и повернутой СК:

$$\hat{R}_x = -\hat{L}_x \qquad \hat{R}_y = -\hat{L}_y \qquad \hat{R}_z = \hat{L}_z$$

Метод удобен, когда преобразования реализованы аппаратно или микропрограммно.

Третий способ

Формальная запись. В конце одно из векторных заменяем на скалярное

Условие, что 3 вектора в 1 плоскости, записывается с помощью векторных произведений.

$$\mathbf{n} \otimes \mathbf{L} = \mathbf{R} \otimes \mathbf{n}$$

Векторные произведения совпадают, если равны их x-, y- и z-составляющие:

$$\begin{aligned} -n_z R_y + n_y R_z &= n_z L_y - n_y L_z \\ n_z R_x - n_x R_z &= n_x L_z - n_z L_x \\ -n_y R_x + n_x R_y &= n_y L_x - n_x L_y \end{aligned} \qquad (5.12)$$

Но одно из уравнений всегда линейно зависимо. Дополнительное условие-равенство углов с помощью скалярных произведений

$$\mathbf{n} \cdot \mathbf{L} = \mathbf{n} \cdot \mathbf{R}$$

или

$$n_x R_x + n_y R_y + n_z R_z = n_x L_x + n_y L_y + n_z L_z \qquad (5.13)$$

что дает необходимое добавочное условие. Матричный вид для всех уравнений с тремя неизвестными R_x, R_y и R_z таков:

$$\begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \\ n_x & n_y & n_z \end{bmatrix} \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} = \begin{bmatrix} n_z L_y - n_y L_z \\ n_x L_z - n_z L_x \\ n_y L_x - n_x L_y \\ n_x L_x + n_y L_y + n_z L_z \end{bmatrix}$$

или

$$[N][R] = [B]$$

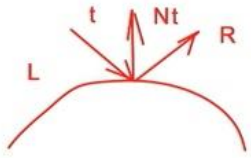
Поскольку матрица $[N]$ не является квадратной, для решения системы нужны особые методы[1]. В частности,

$$[R] = \left[[N]^{\mathrm{T}}[N]\right]^{-1}[N]^{\mathrm{T}}[B]$$

Четвертый способ

ЧИСТО АНАЛИТИЧЕСКИЙ

Угол падения обозначается t, вектор R представляется суммой векторов L и n с некоторыми коэффициентами alpha beta. Далее решается система из двух уравнений и находятся эти коэффициенты



R=al L+bN
cos

cost=(-l,N) cost=(N,R)
N(alL+bN)=cost
(alL+bN)^2=al^2+2al b(-cost+b^2=1
al(-cot)+b=cost b=cost(1+al)
al^2+2al(1+al)cost(-cost) Cos^2t(1+al)^2

al^2(1-c0s^1t)=1-cos^2t al^2=1 al=1 al=-1
al=1 b=2cost

$$\hat{R} = \alpha\hat{L} + \beta\hat{n}.$$

$$\cos t = (-\hat{L}, \hat{n}) = (\hat{n}, \hat{R}).$$

$$\hat{n}(\alpha\hat{L} + \beta\hat{n}) = \alpha(-\cos t) + \beta = \cos t.$$

$$(\alpha\hat{L} + \beta\hat{n})^2 = \alpha^2 + 2\alpha\beta(-\cos t) + \beta^2 = 1.$$

$$\beta = \cos t(1 + \alpha).$$

$$\alpha^2 - 2\alpha\cos^2 t(1 + \alpha) + \cos^2 t(1 + \alpha)^2 = 1.$$

$$\alpha^2 + \cos^2 t(-2\alpha - 2\alpha^2 + 1 + 2\alpha + \alpha^2) = 1.$$

$$\alpha^2 + \cos^2 t(1 - \alpha^2) = 1.$$

$$\alpha^2(1 - \cos^2 t) = 1 - \cos^2 t.$$

$$\alpha = 1.$$

$$\beta = 2\cos t.$$

Не -1 из физических принципов

40. Построение теней при создании реалистических изображений. Учет теней в алгоритмах удаления невидимых поверхностей.

Тени возникают, когда какая-то поверхность не освещается источником в результате того, что на пути света от источника встречается преграда. 1 простейший случай – препятствие – само тело (грани, обращенные к свету препятствуют попаданию света на задние по отношению к источнику света грани). Это собственные тени – тень, возникающая в результате того, что одни грани тела заслоняют другие. Второй вид теней – проекционные – возникающие в силу того, что одни тела препятствуют попаданию света на поверхности других тел.

Решение второй задачи (проекционных теней) – задача инженерной графики – построение проекции грани одного тела на поверхность другого тела.

Еще разделяют полутени (поверхность освещается не всеми источниками: одним из многих – почти полная тень, почти всеми – близка к отсутствию тени) полные тени (свет

ни от одного источника света не попадает в какие-то точки поверхности – освещена только фоном.

Мы строим изображение, ориентируясь на положение наблюдателя – построение тех граней, которые видит наблюдатель. В связи с этим вопрос не просто в нахождении теней, а в нахождении видимых теней. Пользователь может вообще не видеть тени (если вектор взгляда совпадает с вектором падения), иначе – видит.

Нахождение собственных видимых теней – с помощью 1 этапа алгоритма Робертса (нахождение невидимых граней). Дважды: найти грани, находящиеся в тени (грани, невидимые из позиции источника света – совмещаем положение наблюдателя с источником), и видимыми теневыми гранями будут грани, видимые из положения наблюдателя: **невидимы из источника и видимы из наблюдателя**

Повернуть на 45, видима будет 2 видимые грани. Наблюдатель на z, свет на отрицательной x

Главная мысль от Курова с консультации: каждый алгоритм должен делать то, для чего он предназначен. Если совмещать две вещи (тени и удаление невидимых линий, поверхностей), то получается громоздкий и сложный алгоритм.

Все эффекты сами собой учитываются в алгоритме трассировки луче. В других – неудобно – доп буффуры, переменная работа и тд

Построение теней при создании реалистических изображений

- Тени появляются, если положение наблюдателя и источника света не совпадает.
- Тень состоит из двух частей: Полная тень – (центральная, темная, резко очерненная часть) и Полутень (окружающая ее более светлая часть)
- В КГ обычно используют точечные источники освещения, которые создают только полную тень (распределенные источники создают и полутень, но это долго считать).
- Сложность вычислений обычно зависит от положения источников света (проще-в бесконечности: тени с помощью ортогонального проецирования, сложнее-на конечном расстоянии, но вне поля зрения: необходима перспективная проекция, самый трудный – в поле зрения: делить на секторы и считать отдельно для каждого)
- необходимо два раза удалить невидимые поверхности – для положения каждого источника и для положения наблюдателя.
- Различают собственные тени и проекционные. Собственная тень получается, когда сам объект препятствует попаданию света на грани (построение тени аналогично 1 этапу алгоритма Роджерса (удалению нелицевых граней), если точку наблюдения совместить с источником света). Если один объект препятствует попаданию света на другой объект, то возникает проекционная тень. (Чтобы найти такие тени, нужно построить проекцию всех нелицевых граней на сцену. Центр проекции находится в источнике света. Точки пересечения проецируемой грани со всеми другими плоскостями образуют теневые многоугольники заносятся в структуру данных и хранятся в виде контуров, для оптимизации размера)
- Тени зависят от положения источника света и не зависят от положения наблюдателя

про Z буффер и построчного сканирования с z-буфером

Модифицированный алгоритм состоит из двух шагов:

строится сцена из точки наблюдения, совпадающей с источником. Значения z для этого вида хранятся в отдельном теневом z-буфере. Значения интенсивности не рассматриваются;

затем сцена строится из точки, в которой находится наблюдатель. При обработке каждой поверхности или многоугольника его глубина в каждом пикселе сравнивается с глубиной в z-буфере наблюдателя. Если поверхность видима, то значения x, y, z из вида наблюдателя линейно преобразуются в значения x', y', z' на виде из источника. Z’ сравнивается со значением теневого z-буфера при x', y'. Если оно видимо, то оно отображается в буфер кадра в точке x, y без изменений. Если нет, то точка находится в тени и изображается согласно соответствующему правилу расчета интенсивности с учетом затенения, а значение в z-буфере наблюдателя заменяется на z'.

Про Алгоритм Вейлера-Азертона

из двух шагов.

На первом шаге с помощью алгоритма удаления невидимых поверхностей выбираются освещенные, т. е. видимые из положения источника грани. Для повышения эффективности в памяти хранятся именно они, а не невидимые грани (иначе придется хранить и все нелицевые). Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где они приписываются к своим прототипам в качестве многоугольников детализации поверхности.
Чтобы не получить ложных теней сцену надо рассматривать в пределах видимого или отсекающего объема, определенного положением источника. Это ограничение требует также, чтобы источник не находился в пределах сцены, т. к. в этом случае не существует перспективного или аксонометрического преобразования, которое бы охватывало всю сцену.

На втором шаге объединенные данные о многоугольниках обрабатываются из положения наблюдателя. Если какая-то часть не освещена, применяется соответствующее правило затенения. Если источников несколько, то используется несколько наборов освещенных граней.

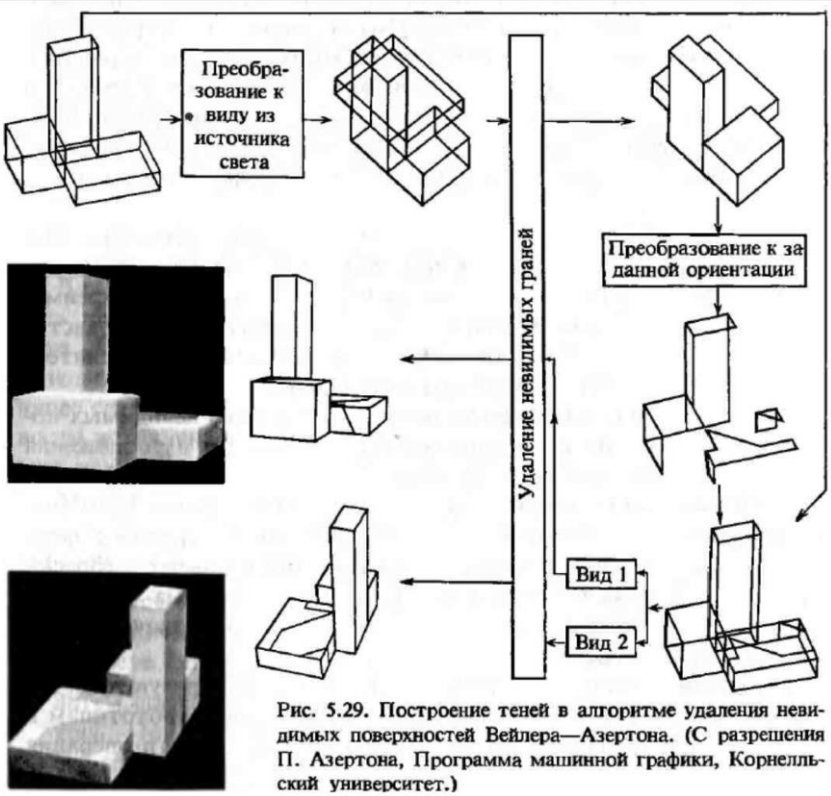


Рис. 5.29. Построение теней в алгоритме удаления невидимых поверхностей Вейлера—Азертона. (С разрешения П. Азертона, Программа машинной графики, Корнелльский университет.)

про трассировку

1) (как и в предыдущем) трассировкой луча от точки наблюдения через плоскость проекции определяются видимые точки сцены

2) луч трассируется от видимой точки до источника света. Если между ними есть какой-нибудь объект, то она оказывается в тени

Наиболее полного отображения тени можно добиться используя алгоритм трассировки лучей вместе с глобальной моделью отражения.

- Когда вы будете строить сцену. Вас какие тени будут интересовать?

Тени, которые видит наблюдатель. Найти их можно используя алгоритмом Робертса. Нас будут интересовать грани, дающие видимую тень.

- Каким условиям должна удовлетворять грань, чтобы давать видимую тень?

Должна быть в тени. Должна быть лицевой для наблюдателя.

Офф-топ

Задачу по Робертсу даёт. Рисует тебе фигурку объёмную, надо составить матрицу тела, определить освещенные и теневые поверхности. И найти видимую тень.

Построить матрицу тела, нормализовать. Найти собственные тени, умножив вектор от источника к объекту на матрицу тела. Найти нелицевые грани, умножив вектор наблюдения на матрицу тела
Отрицательные - не видно
Положительные столбцы - видно
Нули - вектор параллелен плоскости

Если грань теневая и невидимая, то она в списке невидимых и в списке теневых
Записываешь ее уравнение вида ax + by + cz + d= 0 в ответ

41. Учет прозрачности в модели освещения. Учет прозрачности в алгоритмах удаления невидимых поверхностей.

Закон преломления Снеллиуса

При переходе из одной среды в другую световой луч преломляется. Преломление рассчитывается по закону Снеллиуса: падающий и преломляющий лучи лежат в одной плоскости, и произведение коэффициента оптической плотности (показателя преломления) первой среды на синус угла падения равен произведению коэффициента второй среды на синус угла преломления (нормаль и вектор преломления)

$$\eta_1 \sin\theta = \eta_2 \sin\theta',$$

где η_1 и η_2 — показатели преломления двух сред, θ — угол падения, θ' — угол преломления (рис. 5.24). Ни одно вещество не пропускает весь падающий свет, часть его всегда отражается; это также по-

Чисто математический подход (формальный) – 4 способ расчета угла отражения.

ПИШЕМ 45 ВОПРОС!!!!!!

Пропускание бывает

- зеркальным (направленным) (свойственно прозрачным веществам, если смотреть сквозь такое вещество на объект, то искажения (кроме контурных линий криволинейных поверхностей) не будет)
- диффузным (после пропускания свет рассеивается, такие вещества кажутся матовыми, искажает расположенные за ним объекты).

Если не учитывать преломление, то можно ошибочно изобразить на самом деле невидимый объект, и наоборот – не изобразить видимый.

Длина пути луча в преобразованном объекте меняется-> не совпадают точки выхода луча из объекта; меняется количество поглощенного объектом света, поэтому исходящий луч имеет другую интенсивность.

Для того чтобы устранить влияние преломления, можно либо применять алгоритмы, работающие в пространстве объекта, либо пользоваться специальными преобразованиями между пространствами объекта и изображения. Однако проще включить преломление в алгоритмы построения видимых поверхностей методом трассировки лучей, использующие глобальную модель освещения.

Существует два варианта аппроксимации - линейная и нелинейная.

Линейная

если видимая грань прозрачна, то в буфер кадра записывается линейная комбинация двух ближайших поверхностей. При этом интенсивность

$$I = tI_1 + (1 - t)I_2 \qquad 0 \le t \le 1$$

где I1 - видимая поверхность, I2 - поверхность, расположенная непосредственно за ней, t - коэффициент прозрачности I1. Если поверхность совершенно прозрачна, то t = 0, а если непрозрачна, то t = 1. Если I2 тоже прозрачна, то алгоритм применяется рекуррентно, пока не встретится непрозрачная поверхность или фон. Если многоугольники записываются в буфер кадра в соответствии с приоритетами глубины, как в алгоритме со списком приоритетов (Ньюэла-Санча), тогда I2 будет соответствовать значению, записанному в буфер кадра, а I1 - текущей поверхности.

Нелинейная

Для криволинейных поверхностей, например, таких, как ваза или бутылка, линейной аппроксимации недостаточно, так как вблизи контурных линий прозрачность уменьшается из-за толщины материала. Чтобы точнее изобразить это явление, Кэй предложил несложную нелинейную аппроксимацию на основе z-составляющей нормали к поверхности. В частности, коэффициент прозрачности

$$t = t_{min} + (t_{max} - t_{min})[1 - (1 - |n_z|)^p]$$

где tmin и tmax - минимальная и максимальная прозрачность объекта, nz есть z-составляющая единичной нормали к поверхности, p - коэффициент степени прозрачности, t - прозрачность пиксела или точки объекта.

Учет прозрачности в алгоритмах

Простое пропускание света можно встроить в любой алгоритм удаления невидимых поверхностей, кроме алгоритма с z-буфером.

алгоритм с z-буфером

Чтобы учесть эффект прозрачности, необходимы отдельные буферы прозрачности, интенсивности и весовых коэффициентов, а также пометить прозрачные многоугольники.

1. Для каждого многоугольника:
   1.1. если многоугольник прозрачен, то внести его в список прозрачных многоугольников;
   1.2. если многоугольник непрозрачен и z > z-буфер , то записать его в буфер кадра для непрозрачных многоугольников и скорректировать этот буфер.
2. Для каждого многоугольника из списка прозрачных многоугольников:

2.1. если z >= z-буфер , то прибавить его коэффициент прозрачности к значению, содержащемуся в буфере весовых коэффициентов прозрачности;
2.2. прибавить его интенсивность к значению, содержащемуся в буфере интенсивности прозрачности, в соответствии с правилом

$$I_{bn} = I_{b0}t_{b0} + I_c t_c$$

где Ibn - новое значение интенсивности, Ib0 - старое значение интенсивности, записанное в буфере интенсивности прозрачности, Ic - интенсивность текущего многоугольника, tb0 - старый коэффициент прозрачности из буфера весовых коэффициентов прозрачности, tc - коэффициент прозрачности текущего многоугольника. Таким образом, получается взвешенная сумма интенсивностей всех прозрачных многоугольников, находящихся перед ближайшим непрозрачным многоугольником.

3. Объединим буферы интенсивности для прозрачных и непрозрачных многоугольников в соответствии с правилом

$$I_{fb} = t_{b0}I_{b0} + (1 - t_{b0})I_{fb0}$$

где Ifb - окончательная интенсивность в буфере кадра для непрозрачных многоугольников, а Ifb0 - старое значение интенсивности в этом буфере.

Эту процедуру удобнее использовать в сочетании с алгоритмом построчного сканирования с z-буфером, поскольку для полного алгоритма с z- буфером требуется очень много памяти.

Модель освещения

Для того чтобы включить преломление в модель освещения, нужно при построении видимых поверхностей учитывать не только падающий, но и отраженный и пропущенный свет. Эффективнее всего это выполняется с помощью глобальной модели освещения в сочетании с алгоритмом трассировки лучей для выделения видимых поверхностей.

Обычно рассматриваются только зеркально отраженные и пропущенные лучи, так как диффузное отражение от просвечивающих поверхностей порождает бесконечное количество беспорядочно ориентированных лучей. Поэтому моделируются только прозрачные вещества, для которых формула расчета интенсивности является простым расширением ранее описанных моделей. Ее общий вид:

$$I = k_a I_a + k_d I_d + k_s I_s + k_t I_t$$

где индексы a, d, s, t обозначают рассеянный, диффузный, зеркальный и пропущенный свет. В большинстве моделей предполагается, что kt - постоянная и It - интенсивность преломленного света определяется по закону Снеллиуса.

визуализация внутреннего вида сложных объектов или пространств

Для этого всем многоугольникам поверхности приписываются коэффициенты прозрачности, первоначально равные 1, т. е. они считаются непрозрачными. Можно построить изображение такого объекта с удаленными невидимыми гранями. Затем коэффициенты прозрачности некоторых групп граней заменяются на 0, т. е. они делаются

невидимыми. При новом построении изображения сцены получается внутренний вид этого объекта или пространства.

42. Учет фактуры при создании реалистических изображений.

Пишем курова и дополняем из низа формулками и тд

Фактура(текстура) в компьютерной графике - детализация строения (свойств) поверхности. Как правило, рассматривают 2 вида (направления) детализации:

- Нанесение рисунка на поверхность изображения. В первом случае на гладкую поверхность объекта наносят заданный узор с помощью функции отображения, после этого поверхность все равно остается гладкой.
- Учет неровностей. Во втором случае ставится задача создания неровностей на поверхности, т.е. создания шероховатой поверхности путем внесения возмущений в параметры, задающие поверхность

1 Задача

Может решаться 2 способами: 1) есть файлы с рисунками, нанесенными на плоскую поверхность, надо перенести рисунок из фактурного пространства в 3-хмерное пространство, связанное с поверхностью, причем эта задача может решаться:

Либо нахождение соответствующей точки на поверхности (переход от 2d текстурных координат к 3d координатам объекта, либо наоборот – надо закрасить каждую точку поверхности, мы ищем соответствующую ей точку в другом пространстве и смотрим, какая интенсивность соответствует этой точке в 2-мерном пространстве.

Как правило, используют самое простое – линейное отображение – строим зависимость 3м координат от 2м текстурных координат. Зная для концов интервала соответсвующие координаты, находим коэффициенты отображения, как в одну сторону, так и в другую

2 Задача

Основной способ – внесение некоторых возмущений в вектор нормали (возмушение поверхности по направлению вектора нормали в каждой точке поверхности). Вектор нормали в точке невозмущенной поверхности определяется как векторное произведение касательных векторов. А если поверхность возмущенная – возникает второе слагаемое. Векторное произведение в уже возмущенной – можем пренебречь возмущением. **Само возмущение может быть мало , но производная не мала, поэтому производной не пренебрегаем. Одна штука обнуляется. Вектор нормали в итоге – вектор в исходной +2 слагаемых**

1

Для нанесения рисунка на поверхность необходимо отображение объектного пространства в пространство изображения, а также преобразование из фактурного пространства в объектное:

Если рисунок задан в фактурном пространстве в прямоугольной системе координат (u, w), а поверхность — в другой прямоугольной системе координат (q, f), то для нанесения рисунка на поверхность нужно найти или задать функцию отображения одного пространства на другое, то есть $q = f(u, w)$; $f = g(u, w)$ *или* $u = r(q, f)$; $w = s(q, f)$.

Если для пространства изображения используется растровое устройство, то можно применить два несколько отличающихся метода.

1.1

Первый основан на алгоритме разбиения Кэтмула: разбивает кусок поверхности и соответствующий узор. Когда найден фрагмент, покрывающий центр только одного пиксела, усредненная интенсивность соответствующего фрагмента узора используется для определения интенсивности пиксела. В качестве весовой функции берется отношение площади пикселов фактуры, находящихся внутри фрагмента, к их полной площади. Можно также применять сглаживающий фильтр, например, 2*2

Таким образом, алгоритм разбиения Кэтмула начинает свою работу на куске поверхности в объектном пространстве и развивается **в двух направлениях: в пространстве изображения и фактурном пространстве.**

+: не обязательно знать обратное преобразование из пространства изображения в объектное пространство или глубину (значение z) фрагмента в пространстве изображения.

-: фрагмент может не совпадать с одним пикселом в пространстве изображения.

1.2

Часто глубина (значение z) известна из алгоритма удаления невидимых поверхностей. Для того чтобы найти обратное преобразование, нужно сохранить трехмерное видовое преобразование и преобразование объектного пространства в пространство изображения до проецирования на плоскость картины. При этом в фактурное пространство переводится точная площадь, покрываемая пикселом в пространстве изображения.

Задача состоит в том, чтобы отобразить площадь пиксела из пространства изображения на поверхность в объектном пространстве, а затем — в фактурное пространство. Интенсивность пиксела в пространстве изображения равна средней интенсивности пикселов, покрытых этой площадью в фактурном пространстве. На полученный коэффициент умножается диффузная компонента модели освещения.

Существуют, конечно, и другие, более сложные методы, позволяющие устранять лестничный эффект.

2

Для того чтобы поверхность казалась **шероховатой,** можно оцифровать фотографию нерегулярной фактуры и отобразить ее на поверхность. Однако при этом будет казаться, что неровности нарисованы на гладкой поверхности. Дело в том, что в векторе нормали к настоящей шероховатой поверхности и, следовательно, в направлении отражения есть небольшая случайная составляющая. На этой основе **Блинн разработал метод возмущения нормали для построения неровных поверхностей.**

В любой точке поверхности Q (u, w) частные производные по направлениям u и w, Qu и Qw, лежат в плоскости, касательной к поверхности в этой точке. Нормаль в ней определяется векторным произведением n = Qu x Qw.

Блинн строит новую поверхность, которая выглядит шероховатой, внося в направлении нормали функцию возмущения P(u, w). Таким образом, радиус-вектор точки на новой поверхности есть

Q'(u, w) = Q(u, w) + P(u, w)(n/|n|).

Нормаль к возмущенной поверхности имеет вид

n' = Q'u x Q'w

Частные производные Q'u и Q'w выражаются формулами

$$\mathbf{Q}'_u = \mathbf{Q}_u + P_u \frac{\mathbf{n}}{|\mathbf{n}|} + P\left(\frac{\mathbf{n}}{|\mathbf{n}|}\right)_u$$

$$\mathbf{Q}'_w = \mathbf{Q}_w + P_w \frac{\mathbf{n}}{|\mathbf{n}|} + P\left(\frac{\mathbf{n}}{|\mathbf{n}|}\right)_w$$

Последним членом можно пренебречь, так как P (функция возмущения) очень мала. Поэтому

Q'u = Qu + Pu(n'/|n|)

Q'w = Qw + Pw(n'/|n|).

Возмущенная нормаль имеет вид

n' = Qu x Qw + Pu(n x Qw)/|n| + Pw(Qu x n)/|n| + PuPw(n x n)/(|n|^2).

Первый член — нормаль n к исходной поверхности, а последний равен нулю, поэтому

n' = n + Pu(n x Qw)/|n| + Pw(Qu x n)/|n|,

где два последних члена, приведенные к единичной длине, представляют собой возмущение нормали к поверхности и создают соответствующий эффект в модели освещения.

В качестве P можно использовать почти любую функцию, у которой существуют частные производные. Если узор не определяется аналитически, то функция возмущения записывается как двумерная таблица цветов с индексами u, w. Промежуточные значения вычисляются билинейной интерполяцией табличных величин, а производные Pu и Pw вычисляются методом конечных разностей.

- Она должна быть непрерывной;
- Принимать значения из интервала [0,1];
- Вести себя аналогично равномерно распределенной случайной величине.

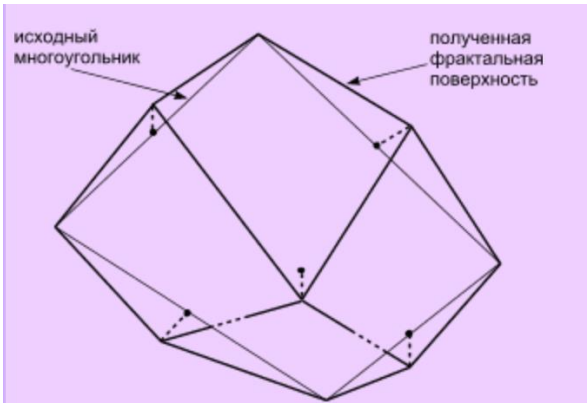
Эффект шероховатости зависит от масштаба изображаемого объекта. Например, если размер объекта увеличится в два раза, то величина вектора нормали возрастет в четыре раза, а его возмущения — только в два. Это приводит к тому, что увеличенный объект кажется более гладким.

Однако масштаб фактуры на перспективном изображении не зависит от перемещения объекта в пространстве по направлению к наблюдателю или от него.

-может появиться лестничный эффект, но необходимо рассчитывать изображение с разрешением, большим, чем у дисплея, а затем отфильтровать или усреднить его и вывести с более низким разрешением экрана.

Один из последних методов построения нерегулярностей основан на фрактальных поверхностях. Фрактальная поверхность состоит из случайно заданных полигональных или биполиномиальных поверхностей.

Для того чтобы получить полигональную фрактальную поверхность, исходный многоугольник рекурсивно разбивается на фрагменты. Для этого можно, например, случайным образом сместить центр и середины сторон многоугольника, причем и исходный, и полученный многоугольники не обязательно должны быть плоскими. Одно из преимуществ фрактальных поверхностей в том, что их можно разбивать бесконечно и получить любой уровень детализации. Он может зависеть от положения наблюдателя: чем ближе точка наблюдения, тем с большей степенью детализации изображается поверхность. Если наблюдатель находится далеко, объем вычислений значительно сокращается. Фрактальная поверхность изображается с помощью любого подходящего алгоритма удаления невидимых поверхностей и любой модели освещения. Однако число разбиений возрастает со скоростью выше линейной, поэтому между количеством разбиений и уровнем детализации должен быть некоторый компромисс. Иначе потребуется слишком много вычислений.



43. Глобальная модель освещения с трассировкой лучей.
44. Алгоритм трассировки лучей с использованием глобальной модели освещения.

Глобальная и локальная модели освещения

Простая (локальная) модель включает в себя:

- Диффузную составляющую отражения
- Зеркальную составляющую
- Рассеянное освещение

В простой модели освещения мы учитываем свет, попадающий в рассматриваемую точку непосредственно от источника света (отраженный), и ориентацию поверхностей.

Глобальная модель будет содержать в себе те же составляющие, но учитывается еще 2: пропускание света прозрачными поверхностями (преломление), и зеркально отраженный свет от других поверхностей (Зеркальное отражение - направленное отражение и его просто учитывать, Диффузное трудно, поскольку распространяется по всем направлениям, обычно учитывается в трассировке не лучей, а путей)

Не путать 2 трассировки: нахождение видимых участков поверхностей и полный вариант (расчет интенсивности, попутно – удаление, учет преломления, теней и фактуры)

Глобальная модель предназначена для изучения свойств объекта, выполнение каких-то вычислений, чтобы рассчитать интенсивность отражённого к наблюдателю света в каждой точке (пикселе) изображения

Благодаря этому глобальная модель освещённости способна воспроизводить эффекты зеркального отражения и преломления лучей (прозрачность и полупрозрачность), а также затенение. Из всех алгоритмов удаления невидимых граней остается только метод трассировки лучей, чьей составной частью она является.

Формула для глобальной модели

Трассируется луч **v**, падающий на поверхность в точке **Q**. В этой точке он отражается в направлении **r** и, если поверхность прозрачна, преломляется в направлении **p**. Здесь **It** - интенсивность света, падающего в точку Q по направлению p. Этот свет преломляется и достигает наблюдателя, находящегося в направлении - v. Аналогично **Is** - интенсивность зеркально отраженного света, падающего в направлении - r и отраженного к наблюдателю в точке Q,

n - нормаль к поверхности в точке Q, **Lj** - направление на j -й источник света, **S и R** - локальные векторы наблюдения и отражения, η - показатель преломления среды, n - степень пространственного распределения Фонга для зеркального отражения. Тогда наблюдаемая интенсивность I выражается формулой

$$I = k_a I_a + k_d \sum_j It_j \left(\hat{n} \cdot \hat{L}_j\right) + k_s \sum_j It_j \left(\hat{S} \cdot \hat{R}_j\right)^n + k_s I_s + k_t I_t$$

где **ka, kd, ks** - коэффициенты рассеянного, диффузного и зеркального отражения, а **kt** - коэффициент пропускания, считаются постоянными, но можно и рассчитать. Две суммы по j - это диффузное и зеркальное отражение от множества источников.

Упрощенно и с 1 источником света

I = IрасКрас + IистКдиф + IистКзерк + IзеркКзерк + IпрКпр.

- Iрас - интенсивность рассеянного освещения; с помощью коэффициента рассеянного отражения Крас мы учитываем какие-либо потери, значения лежат от 0 до 1
- Iист - интенсивность источника;
- Кдиф - коэффициент диффузного отражения;
- Кзерк - коэффициента зеркального отражения.
- IзеркКзерк - добавляем отраженность от других поверхностей, учитывая долю;
- Iпр - интенсивность преломленного луча; Кпр - коэффициента пропускания.

Алгоритм трассировки

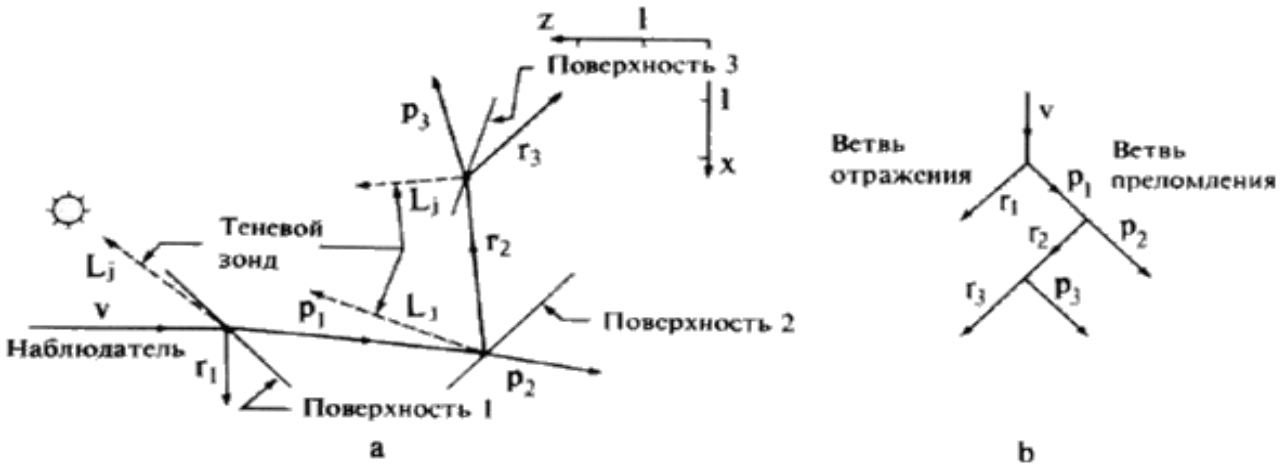
Алгоритм работает рекурсивно. На первом этапе мы строим дерево лучей. Рассматриваем очередной испущенный из глаза наблюдателя луч, проходящий через очередной пиксел экрана, и находим пересечение луча с первой поверхностью на пути его распространения. Точка может породить отраженный луч (по курову – оно всегда есть, в отличие от преломленного) – найти, и определить, есть ли преломленный луч, и если есть – найти характеристик. Порожденные лучи также могут встретиться с повреxностями сцены. Таким образом прослеживаем ход каждого из порождаемых лучей.

Когда прекращаем построение дерева лучей: 1) Луч покинул сцену. 2) Легко догадаться, что каждое новое поколение луча несет меньше энергии (часть отражается, преломляется, поглощается. Поэтому можно это учесть и больше 10 поколений уж точно не стоит рассматривать, лучше 5-7

Используется соглашение: преломленный луч – в правую ветвь, отраженный – в левую. Построив дерево лучей, мы должны ко 2 шагу рекурсии – расчет интенсивности. Интенсивность начинаем рассчитывать от самого нижнего правого узла дерева, поднимаясь к корню, потом и левое поддерево. на каждом шаге рассчитав интенсивность преломленного луча, поднимаемся в предыдущий, учитываем отраженную составляющую и тд. Получаем итоговую интенсивность.

Использование алгоритма трассировки лучей дает не только более реалистичную картину, но сцена также будет более яркой и светлой, так как мы учитываем не только энергию отраженного луча, но и другие составляющие, вносящие серьезный вклад.

В алгоритме удаления невидимых линий трассировка луча продолжалась до первого пересечения с поверхностью. В точке пересечения луч отражается и преломляется, которые так же отслеживаются, пока не останется ни одного пересечения. Таким образом, происходит разветвление алгоритма в виде двоичного дерева: узел - пересечение луча с поверхностью, ветви - правая, порожденная преломлением, и левая - отражением. Ветвь кончается, когда луч покидает сцену.



Направления отраженного и пропущенного лучей рассчитываются по законам геометрической оптики.

- Преломление- Закон Снеллиуса: падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления связаны формулой

$$\eta_1 \sin\theta = \eta_2 \sin\theta',$$

где η_1 и η_2 — показатели преломления двух сред, θ — угол падения, θ' — угол преломления (рис. 5.24). Ни одно вещество не пропускает весь падающий свет, часть его всегда отражается; это также по-

В нашей модели и обозначениях направления r и p таковы

$$r = v' + 2\hat{n}$$

$$p = k_f(\hat{n} + v') - \hat{n}$$

где

$$v' = \frac{v}{|v \cdot \hat{n}|}$$

$$k_f = (k_\eta^2 |v'|^2 - |v' + \hat{n}|^2)^{-1/2}$$

$$k_\eta = \frac{\eta_2}{\eta_1}$$

Здесь k𝔫 - отношение показателей преломления, а $\hat{n}$ - вектор единичной нормали в направлении падающего луча. Если знаменатель kf комплексный, то происходит полное внутреннее отражение, и It, предполагается равным нулю.

- Отражение –вектор падающего света, нормаль к поверхности и вектор отражения лежат в одной плоскости, причем на этой плоскости угол падения равен углу отражения.

Для того чтобы определить интенсивность в каждом пересечении луча с поверхностью, надо пройти дерево в обратном направлении, причем для каждого следующего узла интенсивность ослабляется.

Теоретически дерево трассировки луча может быть бесконечно глубоким. Построение оканчивается, когда все лучи уходят за пределы сцены, но его можно прерывать, когда интенсивность узла падает ниже определенного уровня или когда исчерпан запас памяти.

Чтобы включить в алгоритм тени, надо из каждого пересечения луча с поверхностью направить ко всем источникам Lj теневые зонды. Если на этом направлении между данной точкой и источником лежит другой объект, то точка относительно этого источника лежит в тени и его вклад в локальное диффузное и зеркальное отражение уменьшается. Если поверхность, лежащая на пути луча, непрозрачна, то свет источника вообще не попадает на поверхность, а если тень отбрасывается прозрачной поверхностью, то свет ослабляется в зависимости от ее свойств.

УБИЦА

Тонкости реализации

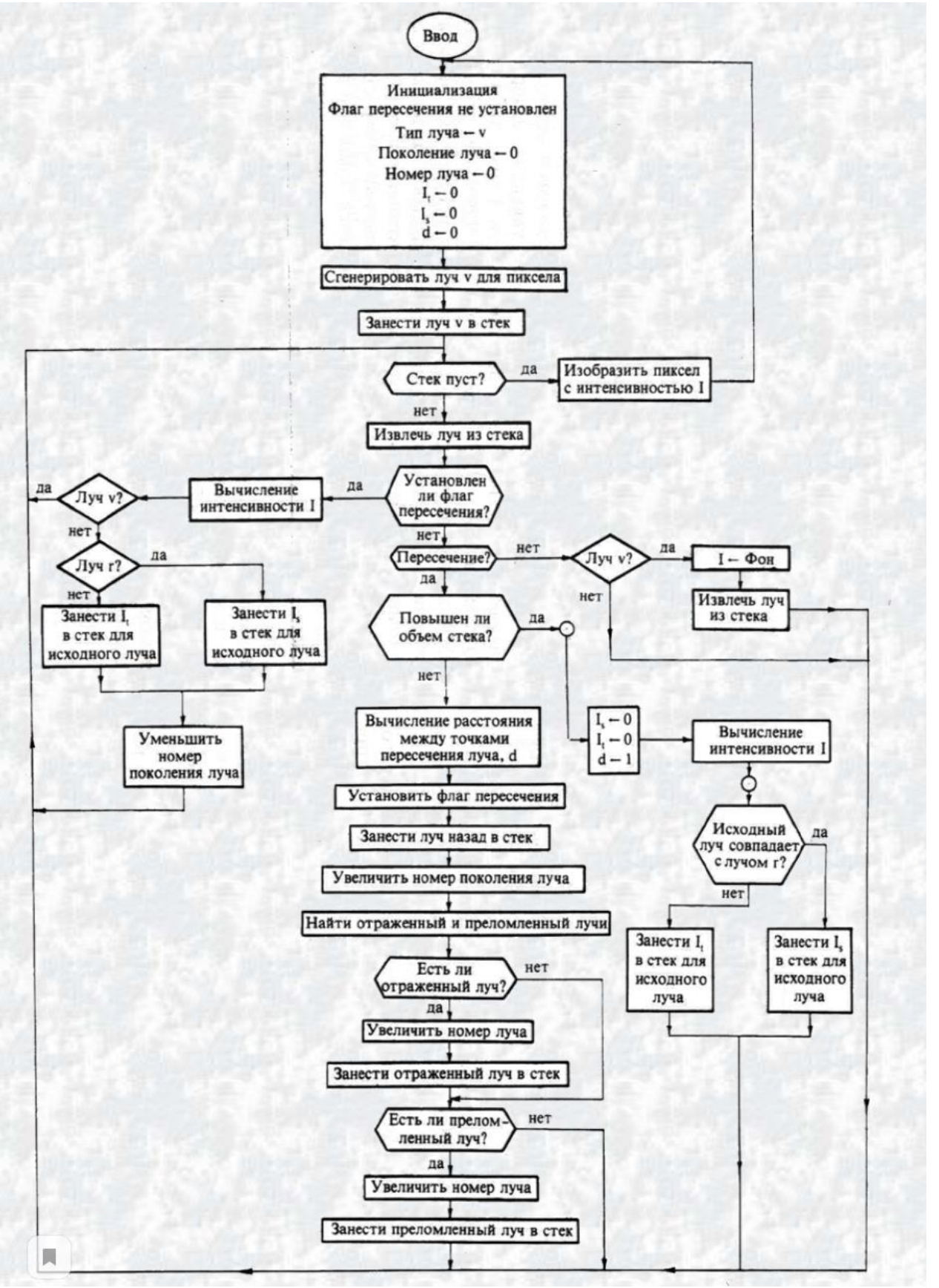
Для передачи данных об отражённом и пропущенном свете применяется стек магазинного типа. Стек содержит в себе в каждый момент только часть дерева (как вариант, самую длинную предполагаемую ветвь) При переполнении стека алгоритм рассчитывает интенсивность исходного луча, используя имеющиеся компоненты в точке падения.

В стеке для каждого луча хранятся следующие данные:

- Идентификатор луча (*уникальный номер*)
- Тип луча (**v** - из глаза, **r** - отражённый, **p** - преломлённый)
- Идентификатор исходного (*породившего данный*) луча
- Тип исходного луча
- Флаг пересечения (*1, если есть пересечения. 0 иначе*)
- Указатель объекта (* Положение объекта, с которым луч пересекается*)
- Координаты пересечения

- Направляющие косинусы луча
- **d** (*расстояние от пересечения исходного до пересечения данного лучей*)
- **It** (*интенсивность пропущенного света в направлении луча*)
- **Is** (*интенсивность зеркально отражённого света в направлении луча*) --

Блок-схема алгоритма трассировки луча с использованием глобальной модели освещения.



45. Определение направления преломленного луча.

При переходе из одной среды в другую световой луч преломляется. Преломление рассчитывается по закону Снеллиуса: падающий и преломляющий лучи лежат в одной плоскости, и произведение коэффициента оптической плотности (показателя преломления) первой среды на синус угла падения равен произведению коэффициента второй среды на синус угла преломления (нормаль и вектор преломления)

$$\eta_1 \sin\theta = \eta_2 \sin\theta',$$

где η_1 и η_2 — показатели преломления двух сред, θ — угол падения, θ' — угол преломления (рис. 5.24). Ни одно вещество не пропускает весь падающий свет, часть его всегда отражается; это также по-

Пусть вектор преломления представляет собой линейную комбинацию вектора падающего света и вектора нормали, т.е.

$$T = \alpha L + \beta N$$ **-2 неизвестных**

Закон Снеллиуса: падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления связаны формулой

$$\eta_1 \sin\theta L = \eta_2 \sin\theta T \quad (0)$$

Поскольку скалярное произведение единичных векторов численно равно косинусу угла между ними, то можно записать следующие два равенства:

$$(-L)N = \cos\theta L \quad (1)$$
$$N(-T) = \cos\theta T \quad (2)$$

- 1 уравнение – косинус угла преломления

2 уравнение – нормировка

Вычислим скалярное произведение единичного вектора преломления самого на себя, которое равно единице:

$$TT = (\alpha L + \beta N)^2 = (\alpha L)^2 + 2\alpha\beta LN + (\beta N)^2 = 1$$ или
$$\alpha^2 + 2\alpha\beta LN + \beta^2 = 1$$

Заменив скалярное произведение вектора падающего света на нормаль через (2), получаем

$$\alpha^2 - 2\alpha\beta \cos\theta L + \beta^2 = 1 \quad (3)$$

Равенство (2) перепишем с учетом исходного выражения, определяющего значение вектора преломления:

$$N(\alpha L + \beta N) = \alpha LN + \beta NN = \alpha LN + \beta = \alpha(-\cos\theta L) + \beta = -\cos\theta T$$

или

$$\cos\theta T = \alpha \cos\theta L - \beta \quad (4)$$

Возведя (0) в квадрат и произведя замену $\eta = \eta_1/\eta_2$, получим после замены синуса на косинус

$$sin^2\theta_T = \eta^2 sin^2\theta_L \rightarrow 1 - cos^2\theta_T = \eta^2(1 - cos^2\theta_L) \rightarrow cos^2\theta_T = 1 - \eta^2(1 - cos^2\theta_L)$$

Возведем теперь в квадрат (4) и, заменив неизвестное значение косинуса угла преломления на известное значение косинуса угла падения, получим:

$$\alpha^2 cos^2\theta_L - 2\alpha\beta cos\theta_L + \beta^2 = 1 - \eta^2(1 - cos^2\theta_L) \quad (6)$$

Вычитая из (3) выражение (6), получаем

$$\alpha^2(1 - cos^2\theta_L) = \eta^2(1 - cos^2\theta_L)$$

Из последнего уравнения получаем α = η (второй корень α = -η не подходит по физическому смыслу). Подставив полученное значение α в (6), получим уравнение

$$\beta^2 - 2\beta\eta \cos\theta L + \eta^2 - 1 = 0$$

Решая это уравнение, получим следующие корни, определяющие неизвестное значение коэффициента

$$\beta_{1,2} = \eta cos\theta_L \pm \sqrt{\eta^2(cos^2\theta_L - 1) + 1}$$

Из двух возможных значений из физического смысла следует взять меньший корень (отрицательный):

$$\beta = \eta cos\theta_L - \sqrt{\eta^2(cos^2\theta_L - 1) + 1}$$

В последнем выражении под знаком квадратного корня может стоять отрицательная величина

$$\eta^2(cos^2\theta_L - 1) + 1 < 0$$

Это будет означать полное внутреннее отражение, т.е. отсутствие преломленного луча при переходе из оптически более плотной среды в оптически менее плотную среду.

Не задумались о том, что чего-то не существует (похоже на отсечение многоугольников с 1 массивом сазерленда- ходжмена)

От курова

$n_1 \sin\gamma_1 = n_2 \sin\gamma_2$

$T = \alpha L + \beta N$

$\cos\gamma_1 = (-L, N) \quad \cos\gamma_2 = (-N, T)$

$(\alpha L + \beta N) N = \cos\gamma_2$

$(\alpha L + \beta N)^2 = 1$

$n_1^2 (1 - \cos^2\gamma_1) = n_2^2 (1 - \cos^2\gamma_2)$

$\cos^2\gamma_2 = n_1^2 (1 - \cos^2\gamma_1) + 1$

$\beta = \alpha \cos\gamma_1 - \cos\gamma_2$

$\alpha^2 = n_1^2$

$\alpha = n_1 \qquad \alpha = -n_1 \qquad n_1 = \frac{n_1}{n_2}$

$\beta = n_1 \cos\gamma_1 - (n_1^2 \cos^2\gamma_1 + 1 - n_1^2)^{\frac{1}{2}}$