

Если есть вариант
сдавать не курсову - сдавайте
не ешь!

P.S. У меня 3.
Билеты идеальные.
по вопросам: @oolonge

1. Задача синтеза сложного динамического изображения. Этапы синтеза изображения. Последовательность и основное содержание.

Методы - математические, алгоритмические.

Средства - технические, программные.

Уровни алгоритмов

1. Нижний (точка, отрезок, эллипс).
2. Средний (плоские изображения, основанные на примитивах)
3. Верхний (трехмерная графика, обработка модели).

При синтезе нужно учесть:

Параметры системы

1. Система координат.
2. Положение картинной плоскости, размер окна обзора (x, y, z наблюдателя, последняя ось - направление взгляда).
3. Источник света (x, y, z света, интенсивность, цвет).
4. Характеристики окружающей среды (коэффициент пропускания света).
5. Частота обновления (обычно 30Гц).

Для моментов времени, отстающих на величину T , должна обеспечиваться возможность вычисления координат объекта.

Параметры поверхности

1. Уравнение поверхности (x, y, z поверхности).
2. Цвет.
3. оптические свойства (коэффициент отражения, преломления и т.д.).
4. Для динамических объектов - уравнения воздействия (Уп).

Параметры системы и поверхности могут быть (и чаще всего) динамически связаны. Пример: отражение от поверхности.

Этапы синтеза изображения

1. Разработка трёхмерной математической модели.
2. Задание положения наблюдателя, картинной плоскости, размеров окна вывода, значений управляющих сигналов.
3. Определение операторов, определяющих пространственное перемещение объектов.
4. Преобразование координат объектов в координаты наблюдателя (относительно наблюдателя).
5. Отсечение объектов по границам пирамиды отсечения (пирамида от камеры).
6. Вычисление двумерных проекций объектов на картинную плоскость.
7. Удаление невидимых линий и поверхностей относительно наблюдателя. Закрашивание и затенение видимых объектов.
8. Вывод полученного изображения на экран.

Какие исходные данные нужно задать при решении задачи синтеза изображения?

Формальное описание объектов, источники света, характеристики окружающей среды, **!оптические свойства!**, частота обновления кадров

1

2. Преобразования на плоскости. Вывод расчетных соотношений.

Матрицы преобразований.

При линейном преобразовании координаты преобразованной точки линейно зависят от координат исходной: $x_1 = Ax + By + C$, $y_1 = Dx + Ey + F$. A..F – параметры, однозначно определяющие преобразование.

Аффинные преобразования – при которых плоскость не вырождается в линию или точку, сохраняется параллельность прямых, всегда есть обратное преобразование.

Матричные преобразования: $(x_1, y_1, 1) = (x, y, 1) \cdot M_{pr}$.

Перенос: два параметра - dx, dy

$$\begin{aligned} x_1 &= x + dx \\ y_1 &= y + dy \end{aligned}$$

$$M_{pr} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{pmatrix}$$

Масштабирование: 4 параметра - $M(x_m, y_m)$ – центр масштабирования, k_x, k_y – коэффициенты. $k_x = k_y \Rightarrow$ однородное

$$M_{pr} = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= k_x x + (1-k_x)x_m & | & |k| > 1 \Rightarrow \text{рисунок увеличивается} \\ y_1 &= k_y y + (1-k_y)y_m & | & 0 < |k| < 1 \Rightarrow \text{рисунок уменьшается} \end{aligned}$$

! Вывод: 1. Сместили x, y на $(-x_m, -y_m)$: $x' = x - x_m$, $y' = y - y_m$

2. Масштабируем: $(x', y', 1) M_{pr} = (k_x x', k_y y', 1)$.

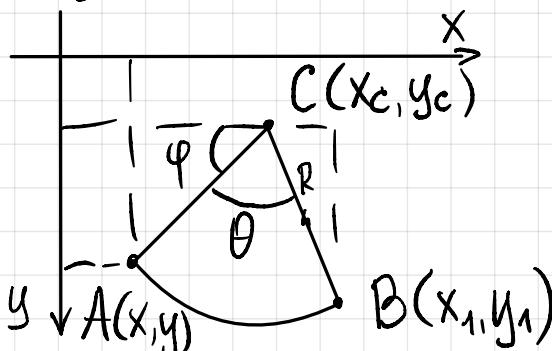
3. Сместили обратно на (x_m, y_m) : $(k_x x' + x_m, k_y y' + y_m, 1)$

$$x_1 = k_x x' + x_m = k_x(x - x_m) + x_m = k_x x + (1 - k_x)x_m, \quad y_1 \text{ – аналогично}$$

Поворот (против часовой): 3 параметра - x_c, y_c – центр поворота, θ – угол поворота

$$M_{pr} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= x_c + (x - x_c) \cos \theta + (y - y_c) \sin \theta \\ y_1 &= y_c - (x - x_c) \sin \theta + (y - y_c) \cos \theta \end{aligned}$$

Вывод:



$$\begin{aligned} x_1 &= x_c + R \cos(\pi - (\varphi + \theta)) = x_c - R \cos(\varphi + \theta) = \\ &= x_c - R \cos \varphi \cos \theta + R \sin \varphi \sin \theta = \\ &= x_c - (x_c - x) \cos \theta + (y - y_c) \sin \theta \\ y_1 &= y_c + R \sin(\pi - (\varphi + \theta)) = y_c + R \sin(\varphi + \theta) = \\ &= y_c + R \sin \varphi \cos \theta + R \cos \varphi \sin \theta = \\ &= y_c + (y - y_c) \cos \theta + (x_c - x) \sin \theta = \\ &= y_c - (x - x_c) \sin \theta + (y - y_c) \cos \theta. \end{aligned}$$

Коммутативность - независимость результата преобразований от порядка, в котором они происходят, коммутативные операции:

1. перенос-перенос.
2. поворот-поворот.
3. масштабирование - масштабирование
4. однородное масштабирование - поворот

Аддитивные операции: перенос, поворот

$$M_{\text{пер-пер}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx_1 + dx_2 & dy_1 + dy_2 & 1 \end{pmatrix}$$

$$M_{\text{поб-поб}} = \begin{pmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Мультипликативные: масштабирование

$$M_{\text{масш-масш}} = \begin{pmatrix} k_{x_1} \circ k_{x_2} & 0 & 0 \\ 0 & k_{y_1} \circ k_{y_2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Почему в преобразовании на плоскости участвуют две координаты, а матрица 3x3?

Не все проводимые нами преобразования на плоскости линейны, мы реализуем также афинные преобразования. К примеру, сдвиг (перенос). Нелинейное преобразование на плоскости таким образом становится линейным в трёхмерном пространстве. Количество координат, необходимое для представления точек, всегда на одну больше, чем размерность пространства, в котором эти координаты используются.

3. Требования, предъявляемые к алгоритмам вычерчивания отрезков.
Пошаговый алгоритм разложения отрезка в растр. Разложение в растр по методу цифрового дифференциального анализатора
Процесс нахождения пикселей, аппроксимирующих заданные отрезок, называется разложением в растр.

- 1. Отрезки должны начинаться и заканчивать в заданных точках.**
Отрезки должны выглядеть как прямые. Выполнено не может быть в полной мере из-за особенностей строения дисплеев, можно лишь добиться визуального восприятия. (Применяя методы устранения ступенчатости)
- 2. Интенсивность вдоль отрезка должна быть постоянной и не зависеть от длины и наклона.** (удовлетворяют также только горизонтальные, вертикальные и наклоненные под углом в 45° отрезки. Однако вертикальные и горизонтальные отрезки по сравнению с отрезками, расположенными под 45°, будут выглядеть ярче, так как расстояние между соседними пикселями у них меньше, чем у наклонных отрезков.) Обеспечение постоянной интенсивности вдоль отрезка требует высвечивания очередного пикселя интенсивностью, зависящей от расстояния между пикселями, вычисление которого производится с извлечением квадратного корня и умножения, что существенно замедляет работу.
- 3. Быстродействие алгоритмов.** Данное требование сводится к оптимизации работы с арифметикой. (Использование целочисленных данных)

ЦДА

аппроксимируем длину отрезка
if $\text{abs}(x_2 - x_1) \geq \text{abs}(y_2 - y_1)$ **then**
 Длина = $\text{abs}(x_2 - x_1)$
else
 Длина = $\text{abs}(y_2 - y_1)$
end if

полагаем большее из приращений dx или dy равным единице раstra
dx = $\text{abs}(x_2 - x_1)/\text{Длина}$
dy = $\text{abs}(y_2 - y_1)/\text{Длина}$

дробную часть округляем
использование знаковой функции делает алгоритм пригодным для всех квадрантов

```
x = x1 + 0.5*Sign(dx)
y = y1 + 0.5*Sign(dy)
i = 0
while (i <= Длина)
    Plot (Integer(x),
          Integer(y))
    x += dx
    y += dy
    i += 1
end while
finish
```

3

Ч. Алгоритмы Брезенхема разложения отрезков в растр. Простой алгоритм Брезенхема. Целочисленный алгоритм Брезенхема. Общий

В алгоритме Брезенхема появляется понятие "ошибки" — расстояние между действительным положением отрезка и ближайшим пикселим раstra. Ошибка считается через приращение dy/dx и в алгоритме требуется проверять только её знак. Так как проверяется только знак ошибки, изначально она принимается как $e = -1/2$. Если угловой коэффициент отрезка больше или равен $1/2$, то величина ошибки в следующей точке раstra может быть вычислена как: $e = e + m$, где m — угловой коэффициент. в зависимости от полученного знака ошибки мы принимает решение, закрашивать верхний или нижний пиксели.

Если 0 - можно любой.

целочисленный

Увеличивает быстродействие первого варианта, убирает плавающую арифметику. Так как важен только знак ошибки, производится преобразование: $E = 2e * dx$ (в таком виде работает только в первом октанте) при таком умножении у нас получается другая e , но это не важно, так как нас интересует только знак, а разница в $2e$ его не меняет.

Чтобы работал во всех квадратах, добавить в проверку номер квадранта и угловой коэффициент. Когда абсолютная величина углового коэффициента больше 1, у постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины x . Выбор постоянно изменяющейся (на + 1 или - 1) координаты зависит от квадранта.

простой

```
x = x1  
y = y1  
dx = x2 - x1  
dy = y2 - y1  
инициализация e с поправкой на половину пикселя  
e = dy/dx - 1/2  
начало основного цикла  
for i = 1 to dx  
    Plot (x, y)  
    if (e >= 0)  
        y += 1, e -= 1  
    end if  
    x += 1, e += dy/dx (по сути тангенс угла наклона, приращение)  
next i  
finish
```

целочисленный

```
x = x1  
y = y1  
dx = x2 - x1  
dy = y2 - y1  
инициализация E с поправкой на половину пикселя (сократилось при умножении на 2 dx)  
E = 2*dy - dx  
for i = 1 to dx  
    Plot (x, y)  
    while (E >= 0)  
        y += 1  
        E -= 2 * dx // 1 * 2dx = 2dx  
    end while  
    x += 1  
    E += 2 * dy // опять таки dy/dx * 2dx = 2dy  
next i  
finish
```

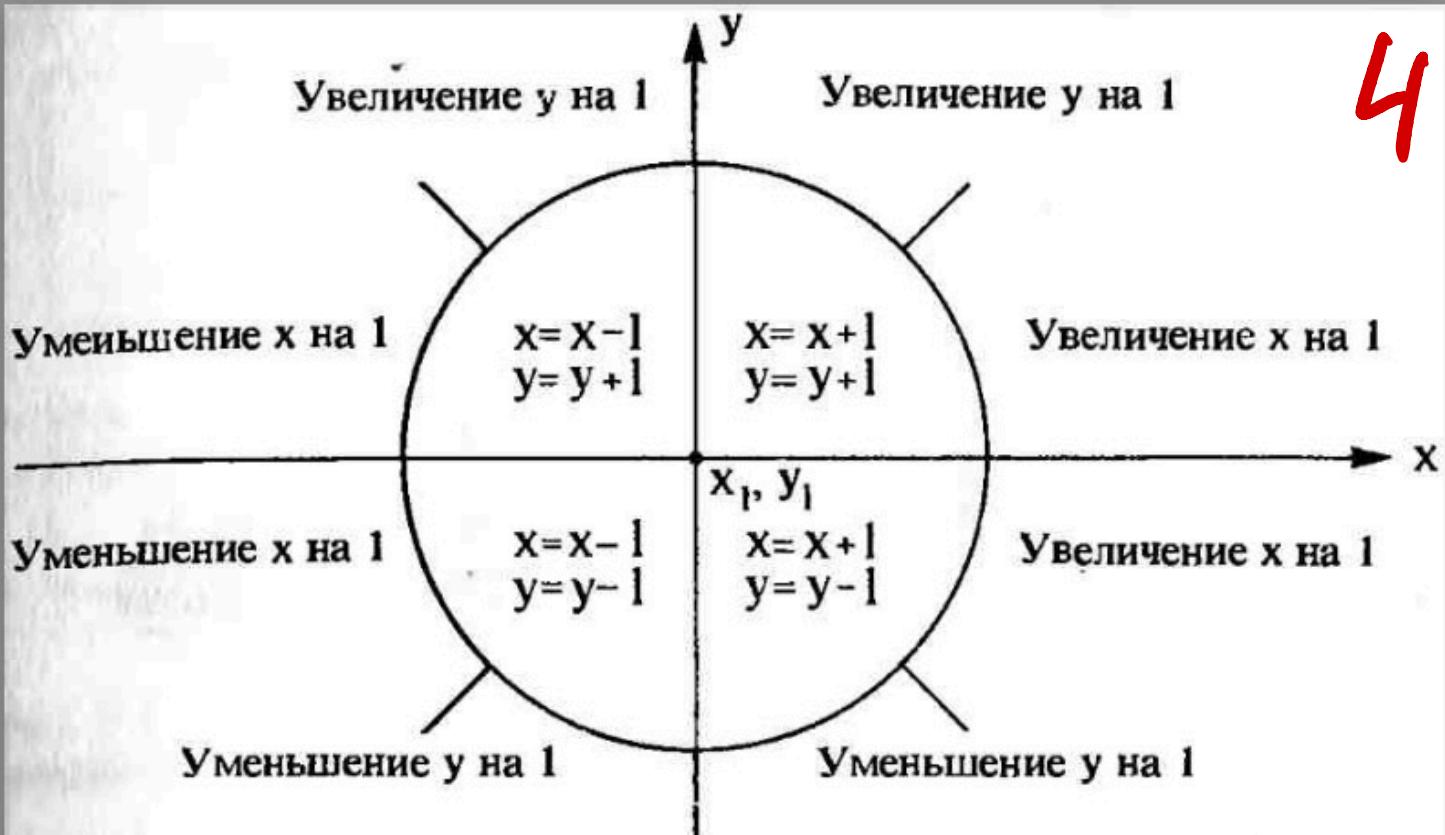


Рис. 2.8. Разбор случаев для обобщенного алгоритма Брезенхема.

$x = x_1$

$y = y_1$

$dx = \text{abs}(x_2 - x_1)$

$dy = \text{abs}(y_2 - y_1)$

$s1 = \text{Sign}(x_2 - x_1)$

$s2 = \text{Sign}(y_2 - y_1)$

обмен значений dx и dy в зависимости от углового коэффициента наклона отрезка

if $dy > dx$ ($dy/dx > 0$) the

$\text{tmp} = dx$

$dx = dy$

$dy = \text{tmp}$

$swap = 1$

else

$swap = 0$

end if

инициализация E с поправкой на половину пикселя (сократилось при умножении на 2 dx)

$E = 2*dy - dx$

for $i = 1$ to dx

 Plot (x, y)

 while $(E \geq 0)$

 if $swap = 1$ then

$x += s1$

 else

$y += s2$

 end if

$e -= 2dx$

 end while

 if $swap = 1$ then

$y += s2$

 else

$x += s1$

 end if

$e += 2dy$

next i

finish

5. Основы методов устранения ступенчатости. Алгоритм Брезенхема с устранением ступенчатости. Алгоритм Ву

Считаем площадь пикселя, входящего в фигуру и красим с соотносимым значением интенсивности. Допустим, у нас есть прямая с тангенсом угла наклона m ($0 \leq m \leq 1$). Если что прямую всегда можно привести к этому виду. При пересечении может быть задействовано как 1, так и 2 пикселя. При одном пикселе $S = y_i + m/2$. При двух $S_1 = 1 - ((1 - y_i)^2)/2m$, $S_2 = (y_i - 1 + m)^2/2m$.

Введем $w = 1 - m$ и посчитаем ошибку $E = e + w = m - 1/2 + 1 - m = 1/2$, $0 \leq E \leq 1$. Теперь E - мера площади той части пикселя, которая находится внутри многоугольника, т.е. $y_i + m/2$.

Изначальное значение ошибки, как показано выше, $1/2$, поэтому инт. первого пикселя = $1/2$ макс. инт.

Более реалистичное значение для первого пикселя дает перемещение оператора активирования пикселя на другое место. Модифицированный алгоритм:
отрезок из (x_1, y_1) в (x_2, y_2) . I - число доступных уровней интенсивности.

Инициализация: $x = x_1$, $y = y_1$, $dx = x_2 - x_1$,
 $dy = y_2 - y_1$, $m = I * dy/dx$, $w = I - m$, $E = I/2$

Plot(x, y, m/2)

while ($x < x_2$)

if ($E < w$) **then**

$x += 1$

$E += m$

else

$x += 1$

$y += 1$

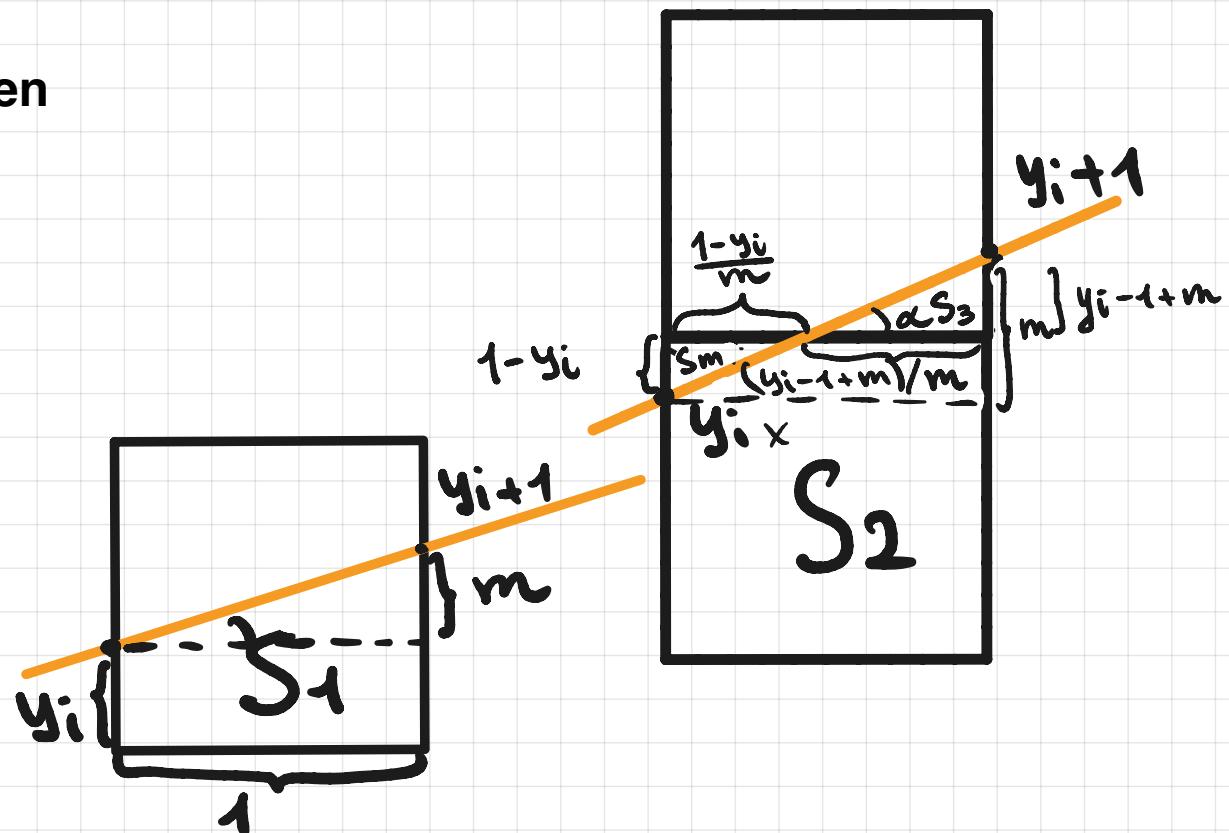
$E -= w$

endif

Plot(x, y, E)

end while

finish



5

Алгоритм Ву устраняет ступенчатость тоже посредством варьирования интенсивности. Но в нём интенсивность зависит от расстояния для идеальной линии и всегда подсвечивает два пикселя-претендента пропорционально их близости. Горизонтальные и вертикальные линии не требуют сглаживания и обрабатываются отдельно. Суммарная интенсивность пикселей $I_1 + I_2 = I_{\text{const}}$. Дробная часть $d_1 = u_{ii} - [u_{ii}]$ – расстояние от нижнего пикселя u_{i1} до точки расположенной на идеальном отрезке. $d_2 = 1 - d_1$. $I_{i1} = I_{\text{const}} * d_2$, $I_{i2} = I_{\text{const}} * d_1$. т. к. $d_1 + d_2 = 1$, то $I_{i1} + I_{i2} = I_{\text{const}} * (d_1 + d_2) = I_{\text{const}}$. Для определения расстояния до пикселя вводится дополнительная переменная которая на каждой итерации цикла увеличивается на величину, равную тангенсу угла наклона прямой. Дробная часть этой переменной определяет расстояние от центра пикселя до идеальной прямой.

6. Построение плоских кривых. Выбор шага изменения аргумента.

Алгоритм построения эллипса и окружности по методу средней точки

Для построения окружности можно использовать её уравнение $(x-x_0)^2 + (y-y_0)^2 = R^2$ или $x = x_0 + R\cos(t)$, $y = y_0 + R\sin(t)$, где x_0, y_0 – координаты центра, R – радиус, t – параметр ($0 \leq t \leq 2\pi$). Учитывая тот факт, что шаг изменения аргумента должен составить величину $t = 1/R$, мы можем рассчитать для каждого значения параметра t значения координат соответствующих точек окружности и затем соединить их отрезками прямых. Это требует большого числа вычислений, от чего неэффективно.

Выбор шага. При достаточно большом радиусе, 2 соседние точки должны быть выбраны так, чтобы величина угла в радианах была не менее $1/R$.

$$\sin\theta = \frac{\Delta z}{R}; \lim_{\theta \rightarrow 0} \frac{\sin\theta}{\theta} = 1; \theta = \frac{\Delta z}{R} = \frac{1}{R}$$

$\Delta z = 1$, потому что изображение растровое и значение принимается за один пиксель.

При генерации окружности достаточно построить $1/8$ окружности и отразить получившуюся дугу. Для эллипса нужно $1/4$ дуги.

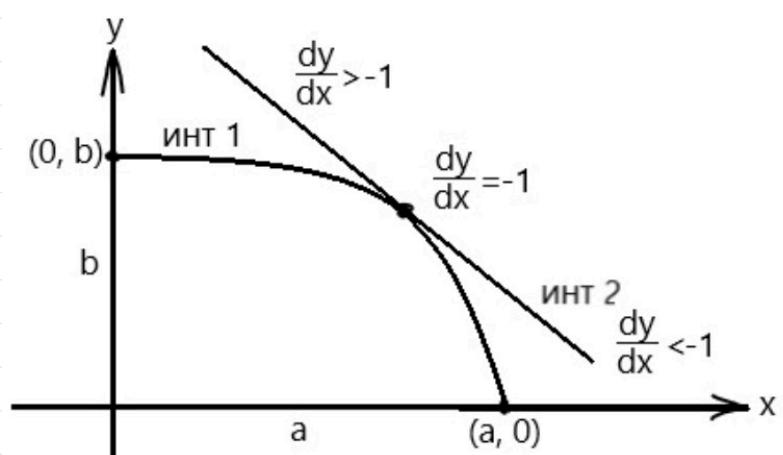
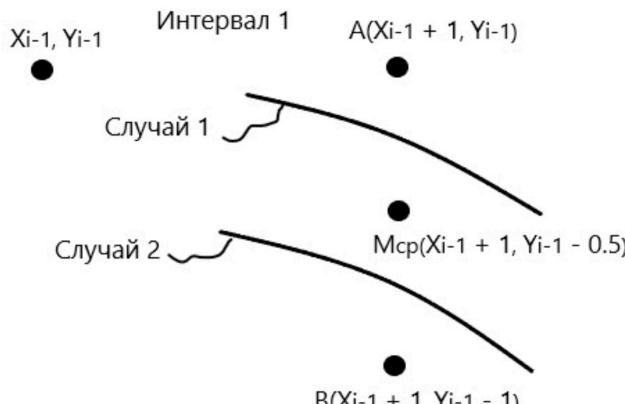
Алгоритм построения по методу средней точки.

Уравнение эллипса: $b^2x^2 + a^2y^2 - a^2b^2 = 0$. На каждом этапе стоит выбор между двумя вертикальными или горизонтальными пикселями. Между ними берется средняя точка и подставляется в уравнение пробной функции (в нашем случае эллипса). если $f(x, y) < 0$ – точка внутри эллипса, иначе – на нём или вне. Пока $dy/dx > -1$ выбираем между двумя вертикальными пикселями, когда < -1 – между горизонтальными.

Пусть пиксель, выбранный на предыдущем шаге, имеет координаты (X_{i-1}, Y_{i-1}) . Средняя точка M будет иметь координаты $(X_{i-1} + 1, Y_{i-1} - 0.5)$. Значение пробной функции для средней точки будет иметь следующее значение:

$$f_{pr} = b^2(x_{i-1} + 1)^2 + a^2(y_{i-1} - 0.5)^2 - a^2b^2$$

Рассмотрим интервал 1:



Это значение можно использовать для принятия решения, но каждый раз его считать нецелесообразно, потому что большое число операций. Для уменьшения можно использовать итерационную схему вычислений, которая будет использовать вычисленное значение пробной функции из предыдущего шага. Координаты средней точки на предыдущем шаге имеют вид (x_{i-1} , $y_{i-1} + 0,5$). Разность значений пробной функции для двух соседних шагов:

$$\Delta f = \left[b^2(x_{i-1} + 1)^2 + a^2 \left(y_{i-1} - \frac{1}{2} \right)^2 - a^2 b^2 \right] - \left[b^2(x_{i-1})^2 + a^2 \left(y_{i-1} - \frac{1}{2} \right)^2 - a^2 b^2 \right]$$

$$\Delta f = 2b^2 x_{i-1} + b^2$$

если провести вычисления по следующей схеме, можно, ограничиться сложением вместо умножения:

$$dx = dx + bd; \quad df = df + b2 + dx, \text{ где } b2 = b^2, bd = 2b^2.$$

Значения $b2$, bd вычисляются один раз в начале работы алгоритма. В случае выбора т.В ранее вычисленное значение пробной функции надо скорректировать.

$$df = \left[b^2(x_{i-1})^2 + a^2 \left(y_{i-1} - 1 - \frac{1}{2} \right)^2 - a^2 b^2 \right] - \left[b^2(x_{i-1})^2 + a^2 \left(y_{i-1} - \frac{1}{2} \right)^2 - a^2 b^2 \right]$$

$$df = -2a^2 y_{i-1}$$

Переход от первого случая выбора пикселов ко второму случаю. Искомая точка удовлетворяет равенству $dY/dX = -1$. Дифференцируем функцию эллипса: $d(b^2 X^2 + a^2 Y^2 - a^2 b^2)/dX = 0$; $2b^2 x + 2a^2 Y (dY/dX) = 0$; $dy/dx = -b^2 x / a^2 y$. \Rightarrow в точке, где $dy/dx = -1$: $2b^2 x = 2a^2 y$ (двойки удобнее для алгоритма, потому что в таком виде вычисляются по ходу выполнения программы). После этого надо скорректировать пробную функцию, чтобы она отражала расположение средней точки между двумя горизонтальными, а не вертикальными пикселями, в итоге имеем:

$$3(a^2 - b^2)/4 - (b^2 x_{i-1} + a^2 y_{i-1})$$

Поскольку в алгоритме ранее вычисляются значения $2b^2 x_{i-1}$ и $2a^2 y_{i-1}$, то полученное

выражение лучше записать как:

$$df = 3(a^2 + b^2)/4 - (2b^2 x_{i-1} + 2a^2 y_{i-1})/2$$

Три случая: выбираем между горизонтальными пикселями, между вертикальными или переходим на новый интервал. С вертикальными: Два случая: ближе к точке А либо к точке В. Если ближе к В (нижней) то мы должны скорректировать. (следует вопрос зачем корректировать?) \rightarrow потому что если мы выбираем пиксель В, мы уже не можем использовать его как пробную функцию предыдущего шага (мы используем разницу между предыдущей и текущей пробными функциями).

```

    while b2 * x < a2 * y do
        if f > 0 then
            x = x + 1
            y = y - 1
            f = f + bd * x + b2 - ad * y
        else
            x = x + 1
            f = f + bd * x + b2
            f = f + 3 / 4 * (a2 + b2) - (a2 * y + b2 * x)

    while y >= 0 do
        if f > 0 then
            y = y - 1; x = x + 1; f = f + ad * y + a2 - bd * x
        else
            y = y - 1; f = f + ad * y + a2

```

Ответ на вопрос, “когда происходит корректировка пробной функции”

7. Основные расчетные соотношения и алгоритм Брезенхема для генерации окружности.

Для генерации окружности достаточно сгенерировать лишь первый октаант, а остальное получить отражениями полученной картинки. Для вывода алгоритма рассмотрим первую четверть окружности с центром в начале координат. Генерируем окружность от точки $(0, R)$ по часовой стрелке. При такой генерации существуют только три возможности выбрать следующий пиксель, наилучшим образом приближающий окружность: правый, диагональный и нижний. Алгоритм выбирает пиксель с минимальным квадратом расстояния между одним из этих пикселей и окружностью. Вычисления можно упростить, если заметить, что в окрестности точки (x_i, y_i) возможны только пять типов пересечений окружности и сетки растра:

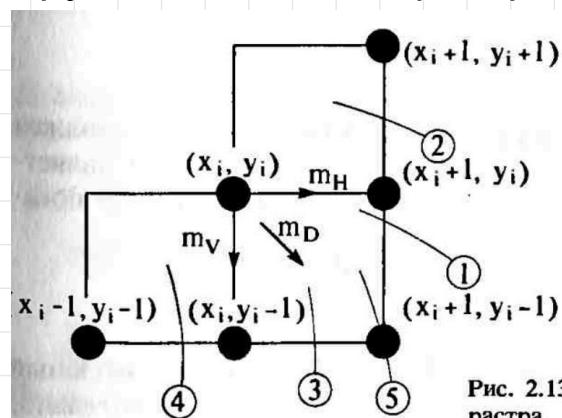


Рис. 2.13
растра.

Разность между квадратами расстояний от центра окружности до диагонального пикселя $(x_i + 1, y_i - 1)$ и от центра до точки на окружности R^2 равна:

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Как и в алгоритме Брезенхема для отрезка, для выбора соответствующего пикселя желательно использовать только знак ошибки, а не ее величину.

При $\Delta_i < 0$, нам подходят только кривые 1 и 2:

чтобы определиться, выбрать m_D или m_H , вычислим:

$$\Gamma \cdot \Delta_1 = |(x+1)^2 + y^2 - R^2| - |(x+1)^2 + (y-1)^2 - R^2| = 2D + 2y - 1,$$

если заметить, что первый модуль ≥ 0 , а второй < 0

если $\Delta_1 > 0$, то должны выбрать диагональный пиксель, иначе горизонтальный

При $\Delta_i > 0$, нам подходят только кривые 3 и 4:

для выбора между m_D и m_V вычислим:

$$\Delta_2 = |(x+1)^2 + (y-1)^2 - R^2| - |x^2 + (y-1)^2 - R^2| = 2D - 2x - 1,$$

если заметить, что первый модуль ≥ 0 , а второй < 0 .

если $\Delta_2 > 0$, то выбираем вертикальный, иначе диагональный.

Значение $\Delta_i = 0$ всегда диагональный (5 случай).

Алгоритм, если будет время:

- 1) Ввод исходных данных R (радиус окружности) и при необходимости X_c, Y_c (координаты центра окружности)
- 2) Задание начальных значений текущих координат пикселя $X=0, Y=R$, параметра $D=2(1-R)$, установка конечного значения ординаты пикселя $Y_k=0$
- 3) Высвечивание текущего пикселя (X, Y)
- 4) Проверка окончания работы: если $Y < Y_k$, то переход к п.11
- 5) Анал значения параметра D : если $D < 0$, то переход к п.6; если $D = 0$, то переход к п.9, иначе переход к п.7
- 6) Вычисление параметра $D_1 = 2D+2Y-1$ и анализ полученного значения: если $D_1 < 0$, то переход к п.8; Если $D_1 > 0$, то переход к п.9
- 7) Вычисление параметра $D_2 = 2D-2X-1$ и анализ полученного значения; если $D_2 < 0$, то переход к п.9, если $D_2 > 0$, то переход к п.10
- 8) Вычисление новых значений X и D (горизонтальный шаг):
 $X=X+1; D=D+2X+1$. Переход к п.3
- 9) Вычисление новых значений X, Y, D (диагональный шаг): $X=X+1; Y=Y-1; D=D+2(X-Y+1)$. Переход к п.3
- 10) Вычисление новых значений Y и D (вертикальный шаг):
 $Y=Y-1; D=D-2Y+1$. Переход к п.3
- 11) Конец

8 Растворная развертка сплошных областей. Алгоритм с упорядоченным списком ребер

Растворная развертка – генерация (закраска) областей на основе простых описаний рёбер или вершин.

Особенность алгоритма: ребра многоугольника упорядочиваются по наивысшей сканирующей строке их пересекающей. Наивысшая сканирующая строка - первая строка, пересекающая ребро.

Эффективность алгоритмов с упорядоченным списком рёбер зависит от эффективности сортировки в порядке сканирования точек пересечений рёбер многоугольника со сканирующими строками.

Простой алгоритм с упорядоченным списком ребер

Для каждого ребра многоугольника определяются точки пересечения со сканирующими строками. Каждое пересечение заносится в список. Список сортируется по двум координатам.

1. Нахождение всех точек пересечения всех сканирующих строк с ребрами многоугольника.
2. Найденные точки пересечения сохраняем в массиве или списке.
3. Точки упорядочиваем, выполняя сортировку по убыванию значений координаты y .
4. Сортируем точки пересечения, расположенные на одной сканирующей строке. Упорядочиваем их по возрастанию абсциссы.
5. Массив точек пересечения, расположенных на одной сканирующей строке, разбиваем на пары и производим закраску пикселей внутри этих интервалов. На этом этапе нужно обработать экстремумы. В случае прохождение очередной сканирующей строки через вершину многоугольника в массиве будут найдены точки с одинаковым значением абсциссы.

Простая версия алгоритма крайне неэффективна: несколько сортировок + сортируем все точки пересечения

Более эффективный вариант предполагает предварительное выделение Y -групп: групп, чьё количество соотносится с числом сканирующих строк. вместо сортировки по Y , теперь мы заносим абсциссу точки пересечения в уже выделенный статический массив для соответствующей ординаты.

Таким образом, первая сортировка отсутствует

Самый эффективный вариант предполагает: Вместо того, чтобы хранить точки пересечения контура со всеми сканирующими строками, можно хранить только точки пересечения контура с текущей сканирующей строкой – Список Активных Ребер.

Подготовка данных:

1. Используя сканирующие строки определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую ребром.
2. Занести ребро многоугольника в у-группу, соответствующую этой сканирующей строке.
3. Сохранить в связном списке значения: начальное значение координат x точек пересечения, dy - число сканирующих строк, пересекаемых ребром многоугольника, и dx - шаг приращения по x при переходе от одной сканирующей строки к другой.

Заполнение:

1. Для каждой сканирующей строки проверить соответствующую у-группу на наличие новых ребер. Новые ребра добавить в САР.
2. Отсортировать координаты x точек пересечения из САР в порядке возрастания; т. е. x_1 , предшествует x_2 , если $x_1 \leq x_2$.
3. Выделить пары точек пересечений из отсортированного по x списка. Активировать на сканирующей строке у пиксели для целых значений x , таких, что $x_1 \leq x + 1/2 \leq x_2$. Для каждого ребра из САР уменьшить dy на 1. Если $dy < 0$, то исключить данное ребро из САР. Вычислить новое значение координат x точек пересечения $X_{\text{нов}} = X_{\text{стар}} + dx$. Перейти к следующей сканирующей строке.



x_i - точка пересечения с ребром $A_i A_j$
 Δy - количество пересечений на ребре
 Δx - шаг приращения по x при переходе от одной сканирующей строки к другой.

9. Заполнение многоугольников. Алгоритмы заполнения по ребрам, с перегородкой, со списком ребер и флагом.

Алгоритм заполнения по рёбрам

Введем соглашение:

- Пиксель может иметь либо цвет фона, либо цвет закраски.
- Считаем, что эти 2 цвета дополняют друг друга (инвертируют).
- Для каждой сканирующей строки, пересекающей ребро многоугольника, дополнить все пиксели, расположенные правее точки пересечения.

В самом простом алгоритме производится закрашивание вплоть до правой границы. В модифицированном в качестве правой границы используется вертикальная линия, проходящая через самую правую вершину многоугольника (X_{\max}).

Ход действий: алгоритм проходится по всем рёбрам (без сортировок!), инвертируя цвет всех пикселов до перегородки. Если пиксель изменил свой цвет четное количество раз — получим изначальный цвет (фона). Иначе инвертированный.

Оценка эффективности: 1. Многократное считывание цвета одного пикселя 2. Конкретный пиксель меняет цвет столько раз, сколько ребер расположено левее него. 3. Обрабатываем пиксели не только внутри, но и снаружи. Вердикт: неэффективный!

Алгоритм заполнения с перегородкой

Вводится вертикальная перегородка, которая выбирается, например, посередине фигуры. Теперь, если точка пересечения сканирующей строки с ребром многоугольника находится правее перегородки — инвертируем все пиксели до неё влево, иначе — вправо. Таким образом мы уменьшаем количество, но не избавляемся от обрабатываемых внешних пикселей. Если выбрать её неправильно, можно уменьшить эффективность алгоритма. Оценка эффективности: чуть лучше, те же проблемы + для выпуклых многоугольников не даёт существенного выигрыша.

Алгоритм заполнения со списком рёбер и флагом

Для этого алгоритма информация о ребрах хранится в виде массива или списка. Вводится флаг — признак нахождения пикселя внутри или вне многоугольника. Шаги алгоритма: 1. Очерчивание границ многоуг.

2. Заполнение многоуг. Отдельно задается цвет границы и цвет заполнения. В этом алгоритме точка пересечения с границей определяется не геометрически, а графически, анализируя цвет очередного пикселя

Алгоритм:

9

1. Очерчивание границ многоугольника
2. Цикл по всем строкам пересекающим ребра многоугольника
 - 2.1 флаг=ложь
 - 2.2 цикл по всем пикселям от $x=0$ до x_{max} (абсцисса правой границы прямоугольной объемлющей оболочки или максимальное значение абсциссы вершин многоугольника)
 - 2.2.1 Если цвет (x,y) =цвет границы, то инвертировать значение флага
 - 2.2.2 Если флаг=истина, цвет (x,y) =цвет заполнения, иначе цвет (x,y) =цвет фона
 - 2.3 Конец цикла по x
3. Конец цикла по y

Оценка эффективности по времени

Критерии быстродействия для алгоритмов заполнения:

1. Кол-во изменений цвета каждого пикселя
2. Кол-во считываий цвета каждого пикселя
3. Кол-во обрабатываемых пикселей (обрабатываем пиксели вне заполняемой области или нет).

Опираясь первыми 2 пунктами можно сказать, что алгоритм достаточно быстродействующий

Этот алгоритм считается таким же быстрым, как и алгоритм с упорядоченным списком ребер, так как каждый пиксель обрабатывается только 1 раз. Считается, что при аппаратной реализации он будет самым быстродействующим из рассмотренных алгоритмов. В случае заполнения такого многоугольника по времени заполнения алгоритм со списком ребер и флагом и алгоритм с упорядоченным списком ребер дадут близкие характеристики.

10. Алгоритм заполнения с затравкой, простой алгоритм заполнения с затравкой.

Алгоритм затравочного заполнения.

Выбираем произвольную область заполнения, могут быть любые области, ограниченные кривой. Задаём затравочную точку (обязательно должна быть расположена внутри области). **Идея заполнения в следующем:** проверяясь соседние с затравочной точкой точки. Если не граница и не закрашенный - теперь они затравочные. **Область можно задать:** путем задания цвета границы (гранитно-определенная) и путем задания цвета пикселей внутри области (внутренне определенная). **Область может быть:** 4-х связная и 8-ми связная (4 и 8 направлений для движения точек соответственно).

Простой алгоритм затравочного заполнения:

- 1. Здание исходных данных: Цвет границы и координаты затравочного пикселя(x,y).**
- 2. Очертить границы заполняемой области.**
- 3. Занесение затравочного пиксела в стек.**
- 4. Пока стек не пуст выполнить следующие действия:**
 - 4.1 Извлечь пиксель из стека.**
 - 4.2 Закрасить пиксель(x, y).**
 - 4.3 Анализ 4-х соседних пикселей. (x + 1, y), (x, y + 1), (x - 1, y), (x, y - 1)**

Если цвет соседнего пикселя не равен цвету границе и цвет этого пикселя не равен цвету заполнения, то текущий пиксель помещаем в стек.

Анализ пикселя на примере (x - 1, y):

Если (x - 1, y) != цвет_границы && (x - 1, y) != цвет_заполнения, то: добавить (x - 1, y) в стек затравочных пикселей.

Быстродействие алгоритмов заполнения зависит от:

- Кол-ва изменений цвета каждого пикселя
- Кол-ва считываний цвета каждого пикселя
- Кол-ва обрабатываемых пикселей

В простом алгоритме:

- Каждый пиксель области меняет цвет один раз
- С каждого пикселя области цвет считывается ~4 раза (кроме пикселей, примыкающих к граничным – там считываний меньше)
- Анализируются только пиксели закрашиваемой области и граничные (с них считывается цвет), пиксели внешней области не анализируются.

Данный алгоритм неэффективен, потому что заносится очень много затравочных пикселей + требуется большой объем памяти для их хранения + некоторые пиксели могут заносится в стек вплоть до 3-4 раз

10

11. Алгоритмы заполнения с затравкой.

Построчный алгоритм заполнения с затравкой

В построчном алгоритме затравочного заполнения мы помещаем в стек один затравочный пиксель для непрерывного интервала пикселей.

Алгоритм

1. Ввод исходных данных (информация о границах области, затравочный пиксель, цвет границы, цвет заполнения)
2. Вычерчивание границы области
3. Занесение затравочного пикселя в стек
4. Пока стек не пуст, выполнять следующие действия:
 - 4.1 Извлечение пикселя из стека (x, y)
 - 4.2 Закраска пикселей текущей строки (y) влево и вправо от затравочного до встречи с граничным пикселием:

Пока цвет(x, y) <> цвет границы:
цвет(x, y) = цвет закраски
Если движемся влево:
 $x = x - 1$
Иначе:
 $x = x + 1$

4.3 Если цвет(x, y) = цвет границы, то $хл = x + 1$ ($хп = x - 1$)

4.4 Поиск новых затравочных пикселей в интервале $хл \leq x \leq хп$
на двух соседних строках по отношению к текущей ($ув = y + 1, ун = y - 1$)

Непрерывный интервал пикселей - группа примыкающих пикселей, расположенных на одной сканирующей строке, ещё не закрашенных и не являющаяся граничными, но ограниченная закрашенными или граничными пикселями. Пример: [граница|П|П|П|П|закрашенный], для этой группы в стек заносим только правый П.

Начальный цвет пикселей области

1. Пиксели не могут иметь цвет границы
2. Пиксели могут иметь цвет закраски, но в ограниченном количестве. Может быть и в неограниченном, но важно, как они распределены в области.
3. Любой другой цвет они могут иметь

Анализ эффективности

1. Цвет каждого пикселя анализируется 3 раза у всех пикселей, кроме пикселей, примыкающих к граничным пикселям сверху и снизу, а также у первого затравочного пикселя - там 2 считывания

11

12. Двумерное отсечение. Простой алгоритм отсечения отрезка

- **Отсечение** - операция удаления изображения за пределами выделенной области, называемой окном. Отсечение может проводиться как на плоскости, так и в пространстве
- **Регулярным (стандартным) отсекателем** на плоскости является прямоугольник со сторонами, параллельными осям экрана или плоскости.
- **Произвольный отсекатель** - обычно многоугольник, который может быть выпуклым/невыпуклым, а также иметь отверстия
- **Видимый объект** - целиком лежащий в области
- **Невидимый объект** - не лежащий в области
- **Частично видимый объект** - частично лежащий в пределах отсекателя

Код точки P_i : T_{ij} , где j ре бокта

1001	1000	1010	$T_{i,0} = 1$, если $x < x_l$
0001	0000	0010	$T_{i,1} = 1$, если $x > x_u$
0101	0100	0110	$T_{i,2} = 1$, если $y < y_l$ $T_{i,3} = 1$, если $y > y_u$

$$S_i = \sum_{j=0}^3 T_{ij} \quad \text{и} \quad P = \sum_{j=0}^3 T_{ij} \cdot T_{2j}$$

$S_i \neq 0 \Rightarrow$ точка невидима

$S_i = 0 \Rightarrow$ точка видима

$P \neq 0 \Rightarrow$ отрезок лежит по одному из сторон окна (невидим)

$(S_1 = 0) \wedge (S_2 = 0) \Rightarrow$ отрезок полностью видим

$((S_1 = 0) \wedge (S_2 \neq 0)) \vee ((S_1 \neq 0) \wedge (S_2 = 0)) \Rightarrow$ отрезок частично видим

Суть алгоритма:

$m = dy / dx$, тангенс

1) Если отрезок полностью видим, отрисовываем

2) Если отрезок тривиально невидим ($P \neq 0$), то ничего не

отрисовываем

3) Если хотя бы один из концов отрезка видим ($S = 0$), то сохраняем его как начало отрезка, а второй конец кладем в рабочую переменную Q , иначе - первый конец в переменную Q

4) Находим точку пересечения рабочей точки с невидимой гранью отсекателя: $X = m * (Y_{\text{грани}} - Q_y) + Q_x$, $Y = 1/m * (X_{\text{грани}} - Q_x) + Q_y$

5) Заносим точку в отрисовываемый отрезок, если концы не видимы, то повторяем пункт 3, кладем второй конец в переменную Q

6) Если нашли две точки, отрисовываем получившийся отрезок

12

13. Отсечение. Алгоритм Сазерленда-Коэна отсечения отрезка

Алгоритм Сазерленда-Коэна предусматривает нахождение точек пересечения отрезка со сторонами окна прямоугольной формы. Однако здесь не производится проверка корректности найденных точек пересечения, как в простом алгоритме.

Найденная точка пересечения разбивает отрезок на **две части**: полностью невидимую относительно очередной стороны отсекателя и видимую часть.

Невидимой будет та часть отрезка, которая заключена от вершины отрезка, невидимой относительно текущей стороны окна, до точки пересечения. Этот факт используется в алгоритме для определения части отрезка, подлежащей отсечению.

В алгоритме предусматривается предварительный анализ сумм кодов концов отрезка и попарных логических произведений этих кодов с целью определения тривиально видимых и тривиально невидимых отрезков.

Если отрезок не является ни тривиально видимым, ни тривиально невидимым, то производится его отсечение. При этом предполагается, что невидимой относительно каждого ребра должна быть первая вершина отрезка. Поэтому в случае необходимости вершины меняются местами.

Можно заметить, что одновременно обе вершины не могут быть невидимыми относительно одного ребра отсекателя, так как этот факт позволил бы на предварительном этапе отбросить отрезок как тривиально невидимый.

Алгоритм:

1. Ввод данных отсекателя (x_l, x_p, y_n, y_v), P_1, P_2 . Параметры отсекателя заносим в массив O .

2. Формирование признака

$flag = 0$

Если $P_1.x = P_2.x$, то $flag = -1$

Иначе $m = dY/dX$, если $m = 0$, то $flag = 1$

3. Цикл отсечения по всем сторонам отсекателя по I от 0 до 4

- 3.1. Обращение к функции определения видимости отрезка

- 3.2. Если отрезок видимый, то переход к отрисовке

Если отрезок невидимый, переход к концу алгоритма

- 3.3. Если $T_1[i] = T_2[i]$, переход на следующую итерацию цикла

- 3.4. Если $T_1[i] = 0$, то обмен местами P_1, P_2

- 3.5. Если $flag <> -1$

Если $i < 2$, то

$$P_1.y = m * (O[i] - P_1.x) + P_1.y$$

$$P_1.x = O[i]$$

Иначе

$$P_1.x = 1/m * (O[i] - P_1.y) + P_1.x$$

Иначе:

$$P_1.y = O[i]$$

- 3.6. Конец цикла

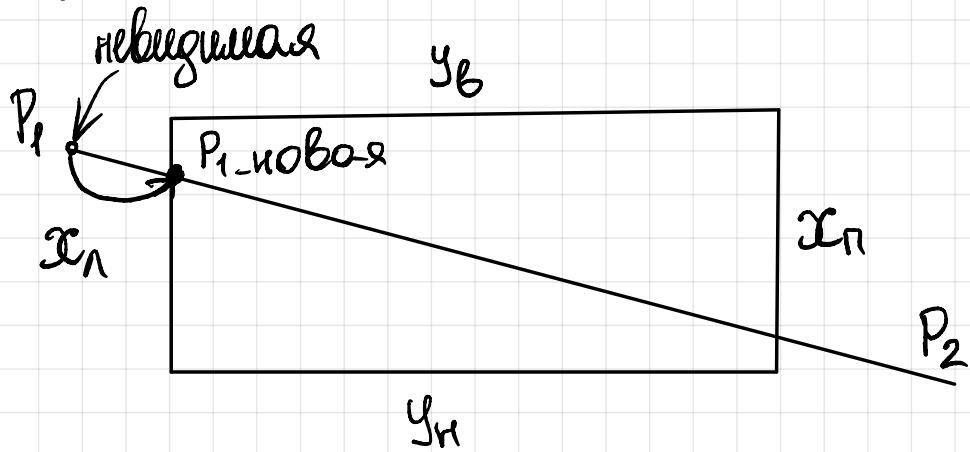
4. Изображение отрезка P_1, P_2

Почему в случае $T1[i] == T2[i]$ уходим на следующий шаг?

Если одноименные биты концов отрезка равны, то мы считаем, что они могут равняться только нулю, т. е. обе вершины располагаются по видимую сторону от текущей границы отсекателя, точки пересечения нет, нужно переходить на следующий шаг. Мы считаем, что они не могут одновременно равняться единице, т. к. такое возможно только если отрезок тривиально невидим, а такие отрезки отбрасываются гораздо раньше, следовательно дойти до этого этапа не могут.

Чем данный алгоритм отличается от простого?

В простом алгоритме после нахождения точки пересечения отрезка с границей отсекателя проводится проверка на корректность найденного пересечения. Если пересечение корректно, то считается, что одна точка видимой части отрезка найдена и дальше надо находить вторую точку видимой части отрезка. В данном алгоритме, каждый раз находя точку пересечения отрезка со стороной отсекателя, мы никаких проверок на корректность не проводится, просто отбрасываем невидимую часть отрезка. Другими словами, мы невидимую вершину P_1 , после нахождения точки пересечения, перемещаем в найденную точку пересечения.

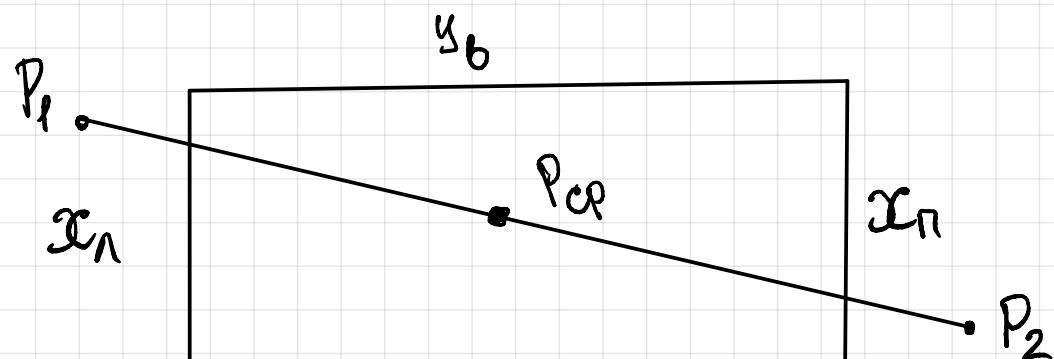


14. Отсечение. Алгоритм разбиения средней точкой при отсечении отрезка.

- Отличие в том, что точка пересечения находится не аналитически, а численным методом (делением отрезка пополам)
- $P_{cp} = (P_1 + P_2) / 2$
- Необходимо задавать точность epsilon (не меньше $\sqrt{2}$)

Нахождение точки пересечения отрезка с границей отсекателя, можно рассматривать как нахождение самой удаленной, но еще видимой точки. Можем сразу определить интервал, где лежит исходный корень.

- Ищем P_{cp} , если часть отрезка невидима - то отбрасываем, таким образом находим 1 точку пересечения
- После того, как нашли точку, на первую вершину ставим точку пересечения, на вторую - оставшуюся вершину. Аналогично находим вторую точку пересечения.
- Целесообразно проанализировать видимость второй точки. Если она видима, то одна вершина найдена. Значит надо искать только второе пересечение.



псюдокоде:

- Ввод данных $P_1, P_2, x_l, x_{pr}, y_l, y_{pr}, \epsilon$
- $i = 1$
- Вычисление кодов отрезка и их сумм (T_1, T_2, S_1, S_2)
- Проверка полной видимости: если $S_1 == S_2 == 0$, то переход к п. 17
- Вычисление произведения кодов концов отрезка, проверка полной невидимости: если $r != 0$, то переход в конец
- $R = P_1$
- Если $i > 2$, то вычислить произведение кодов концов (R); если $r != 0$, то отрезок невидимый, переход к п. 3
- Если $S_2 = 0$, то одна точка пересечения найдена; $P_1 = P_2, P_2 = R; i = i + 1$; переход к п. 3
- Если $|P_2 - P_1| > \epsilon$, то

$$P_{cp} = (P_1 + P_2) / 2, T = P_1, P_1 = P_{cp}$$

Определение кодов конца отрезка и вычисление логического произведения

Если $R != 0$: $P_1 = T; P_2 = P_{cp};$ повторить пункт 9

10. Одна вершина найдена. $P_1 = P_2; P_2 = R; i = i + 1;$ переход к п. 3

11. Конец

Почему алгоритм имеет право на жизнь?

Середину отрезка находим делением на 2, а это быстро в ЭВМ.

Что такое отсечение?

Отсечение - операция удаления изображения за пределами выделенной области, которая называется окном или отсекателем

Надо ли особым образом обрабатывать вертикальные и горизонтальные отрезки?

В данном алгоритме не требуется, т. к. не работаем с тангенсом отрезка, в отличии от алгоритма Сазерленда-Коэна.

Можем ли мы придумать вариант расположения отрезка, что придется делить его пополам для получения вырожденного отрезка (точки) (куров момент)?

Подобное может произойти только если эта точка окажется под 45 градусов от угла отсекателя.

Что такое тривиально невидимый отрезок?

Такой отрезок, у которого при анализе его кодов концов, произведение этих кодов не равно 0, то есть оба конца отрезка находятся по одну сторону от отсекателя.

15. Отсечение. Алгоритм Кируса-Бека отсечения отрезка

Отсечение - это операция удаления изображения за пределами выделенной области, называемой окном

Рассматриваем произвольный отсекатель, удобно использовать параметрическую формулу задания отрезка:

$P(t) = P_1 + (P_2 - P_1) * t$, где $0 \leq t \leq 1$. Границы параметра указаны, чтобы показать, что это отрезок, а не прямая.

Для каждой из четырех границ прямоугольного отсекателя видим, что значения параметра выходят за допустимые значения:

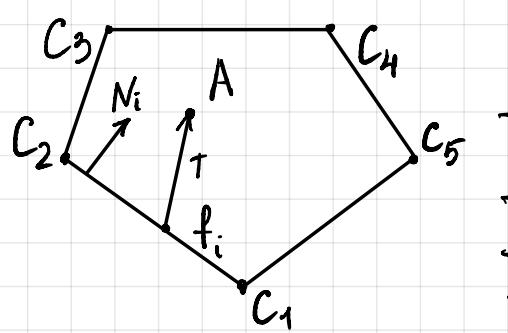
$t < 0$ - для нижней и левой границ

$t > 1$ - для верхней и правой границ

Для полностью видимых отрезков точки пересечения не соответствуют допустимому значению параметра. Для полностью невидимых выполняются те же самые условия.

Возьмём произвольный выпуклый отсекатель:

Видимость т. А:



$f_i \in C_1 C_2, N_i$ - вектор внутренней нормали

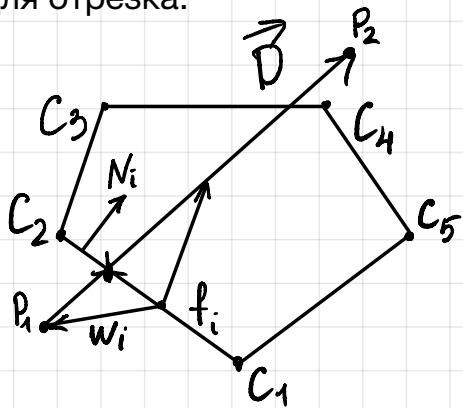
$$T = N_i \cdot (A - f_i).$$

$T > 0 \Rightarrow$ по видимую сторону отсекателя

$T = 0 \Rightarrow$ на границе отсекателя

$T < 0 \Rightarrow$ по невидимую сторону отсекателя

Для отрезка:



$$T = N_i (P_1 + (P_2 - P_1)t - f_i) = N_i (D t + W_i) = 0$$

$T > 0 \Rightarrow$ точка отрезка видима отн. отсекателя

$T = 0 \Rightarrow$ точка лежит на отсекателе

$T < 0 \Rightarrow$ точка невидима относительно отсекателя

$D = P_2 - P_1$ - вектор направления отрезка
(директрисса)

$W_i = P_1 - f_i$ - вектор, соединяющий
произвольную точку f_i с первой вершиной

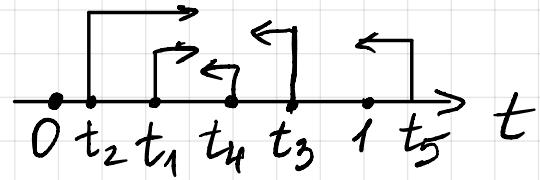
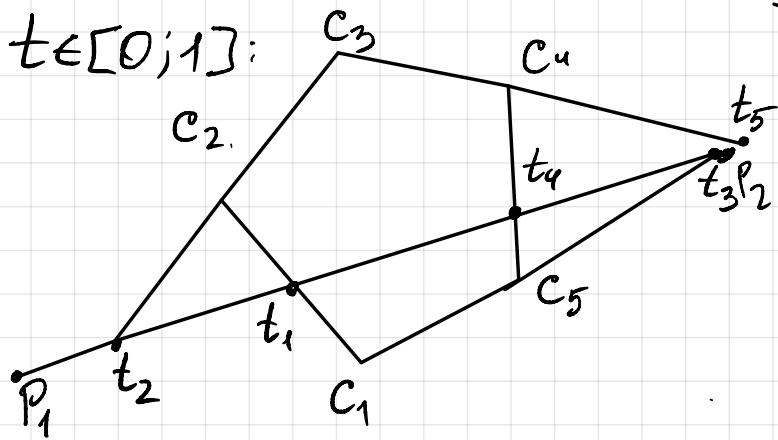
$$t(N_i D) + W_i N_i = 0, W_i N_i = W_{CK}, N_i D = D_{CK}$$

$$t = -\frac{W_{CK}}{D_{CK}}, D=0 \Rightarrow \text{вырожден}, D_{CK}=0 - \text{параметр текущей строке}$$

Если $W_{CK} \geq 0$, отрезок лежит по видимую сторону

Если $W_{CK} < 0$, отрезок по невидимую сторону. можно перейти к

след. узел



t_i - точки пересечения

Найденные точки пересечения можно разбить на две группы:

- 1) Ближе к началу отрезка (определяют начало видимой части)
Критерий: $D_{ck} > 0$
- 2) Ближе к концу отрезка (определяют конец видимой части)
Критерий: $D_{ck} < 0$

Нужно выбрать $\max(\text{начальные})$ и $\min(\text{конечные})$

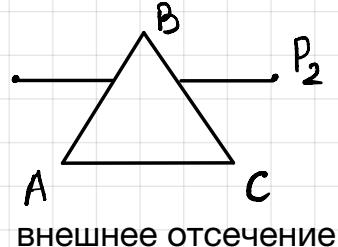
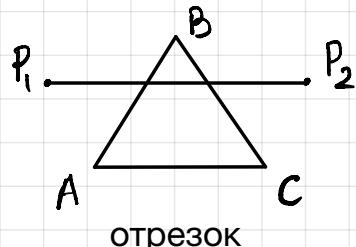
Алгоритм

1. Ввод $P_1, P_2, K, C(K)$
2. $t_h = 0, t_b = 1$
3. $D = P_2 - P_1$
4. Проверка отсекателя на выпуклость
5. Цикл отсечения (по i от 1 до K)
 - 5.1. $W_i = P_1 - f_i$
 - 5.2. Вычисление D_{ck}, W_{ck}
 - 5.3. Если $D_{ck} = 0$, проверить, если $W_{ck} < 0$, то отрезок невидим
 - 5.4. $t = -W_{ck}/D_{ck}$
 - 5.5. Если $D_{ck} > 0$, то проверить если $t < 0$, то отрезок невидим, иначе $t_h = \max(t, t_h)$
 - 5.6. Если $D_{ck} \leq 0$, то проверить если $t > 1$, то отрезок невидим, иначе $t_b = \min(t, t_b)$
- 5.7. Конец цикла
6. Если $t_h \leq t_b$, то изобразить отрезок (t_h, t_b)
7. Конец

P.S. точек пересечения не больше количества рёбер

Простых способов определения полностью видимых/невидимых отрезков быть не может, они распознаются уже по ходу алгоритма

16. Внутреннее и внешнее отсечение. Определение выпуклости многоугольника; определение нормали; разбиение невыпуклых многоугольников. Триангуляция многоугольников.



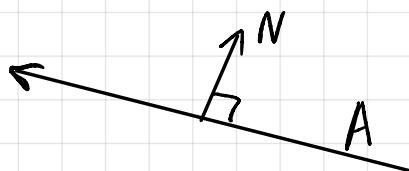
Определение выпуклости многоугольника

Факт выпуклости или невыпуклости можно установить путем вычисления векторных произведений его смежных сторон и анализа знаков этих произведений: все знаки = 0 - многоугольник вырождается в отрезок, есть числа разных знаков - многоугольник не выпуклый, все знаки неотрицательные - многоугольник выпуклый, а внутренние нормали ориентированы влево от его контура; все знаки неположительные - многоугольник выпуклый, а его внутренние нормали ориентированы вправо от его контура.

Формула: $\text{sign}(x_1 * y_2 - x_2 * y_1)$

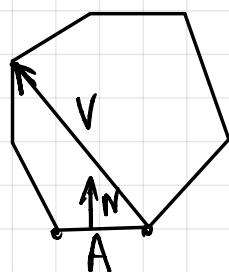
Определение нормали

Для определения нормали можно воспользоваться фактом равенства нулю скалярного произведения вектора отрезка на вектор-нормаль



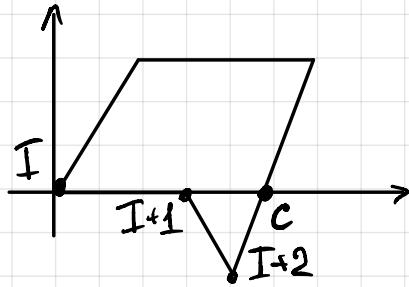
$$Ax \cdot Nx + Ay \cdot Ny = 0$$

$$Nx = -(Ay \cdot Ny) / Ax, Ny = 1$$



Для многоугольника: нашли нормаль, нужно определить, внутреннюю или внешнюю
Для этого необходимо вычислить скалярное произведение вектора N и вектора V , начало которого в произвольной точке прямой A (обычно в начале), а конец - в произвольной вершине многоугольника, такой, что вектор V не совпадает со стороной A . Если скалярное > 0 , нашли внутреннюю нормаль. Если < 0 - внешнюю. Векторы внутренней нормали ориентированы влево от направления обхода, при условии, что обходили против часовой.

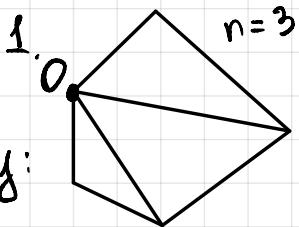
Разбиение невыпуклых многоугольников



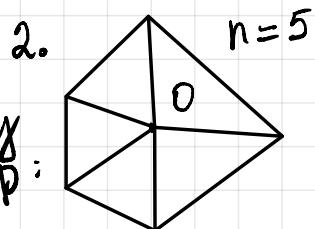
- Если многоугольник оказался невыпуклым в $I + 2$ вершине:
- 1) Перенесём многоугольник так, чтобы вершина I имела координаты $(0, 0)$
 - 2) Повернём его так, чтобы вершина $I+1$ лежала на положительной стороне оси абсцисс.
 - 3) Найти точки пересечения сторон с осью абсцисс.
 - 4) Из найденных точек выбрать ближайшую к началу СК. в нашем случае - единственная, С.
 - 5) В первый многоугольник занести вершины начиная с $I + 1$, заканчивая С.
 - 6) Во второй многоугольник занести все вершины, не попавшие в первый.
 - 7) Каждый многоугольник проверить на выпуклость

Триангуляция многоугольников - разбиение на треугольники.

Выпуклый:



через вершину:

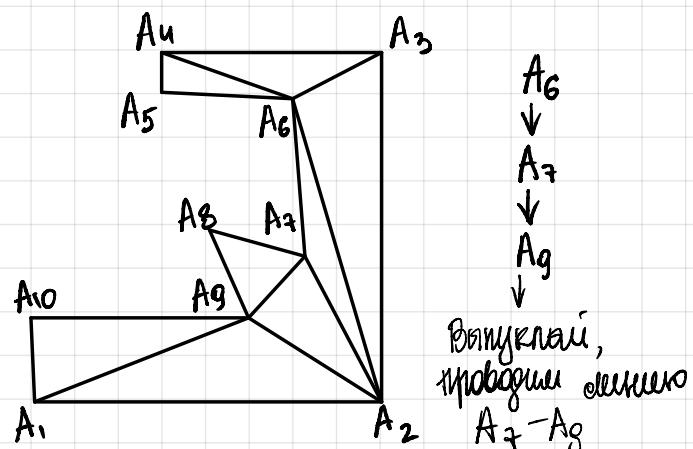


через центр:

Невыпуклый:

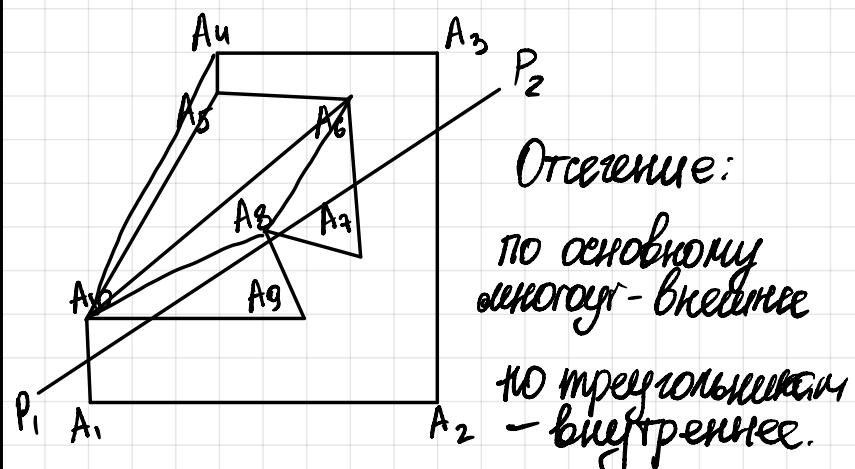
1. Разбить на выпуклые

Ищем невыпуклую вершину, из нее проводим диагональ, содержащую внутри исходной фигуры и проходящую только через ребра из тех вершин, которые она соединяет. Повторяем процесс, пока не разобьем фигуру на выпуклые многоугольники, после чего разбиваем их на треугольники.



2. Дополнить до выпуклого:

1. Ищем вершину, для которой соединение следующей и предыдущей даст ребро вне многоугольника, пока не получим выпуклый



17. Отсечение многоугольников. Алгоритм Сазерленда Ходжмена.

Алгоритм работает с произвольным выпуклым отсекателем. Входной многоугольник может быть как выпуклым, так и невыпуклым.

Воспользоваться простым алгоритмом отсечения отрезков не получится, потому что на выходе будет лишь совокупность сегментов ребер исходного многоугольника.

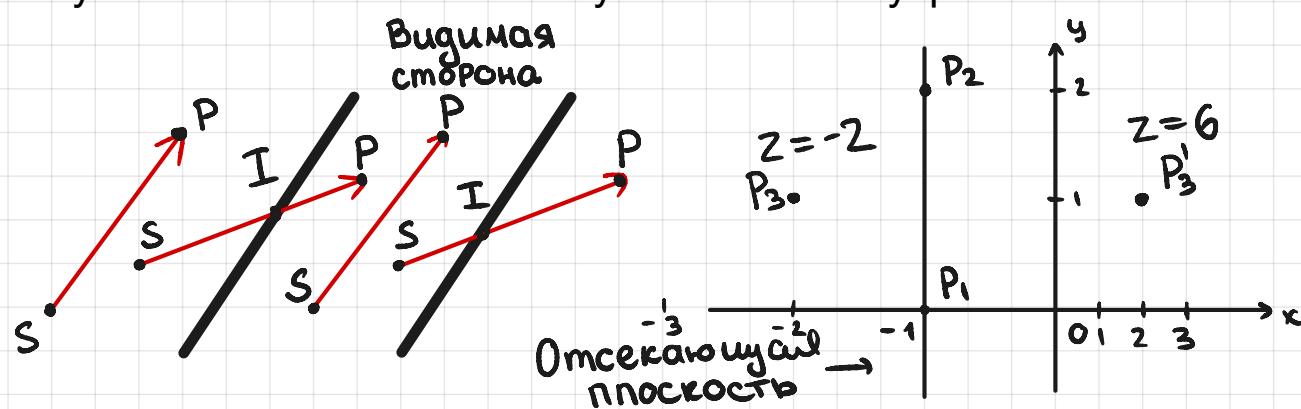
Идея: выполнять отсечение последовательно относительно каждой границы отсекателя. Результат отсечения после очередной итерации используется на следующей. **В результат попадают:** 1) те вершины входного многоугольника, которые видны относительно текущей границы отсекателя; 2) точки пересечения рёбер входного многоугольника с прямой, проходящей через текущую границу отсекателя.

Рассматривается **четыре расположения ребер** многоугольника при обходе:

1. **полностью невидимые относительно границы:** обе вершины невидимы и в результат не заносятся
2. **частично видимые и входят в отсекатель:** начальная вершина невидима, конечная видима. Найти точку пересечения, занести в результат её и конечную точку ребра.
3. **Полностью видимые относительно границы:** обе должны быть занесены (но начальная уже должна быть занесена на предыдущем шаге)
4. **Частично видимые и выходят из отсекателя:** начальная видима, конечная нет. Начальная будет занесена на предыдущем шаге, надо лишь найти точку пересечения ребра с прямой и занести её.

Перед анализом может оказаться, что отсекаемый многоугольник полностью невидим: если нам осталось проанализировать последнее ребро а результат пуст, то исходный многоугольник невидим относительно текущей границы, а значит и для всего отсекателя — завершаем.

Недостаток алгоритма: в результирующих многоугольниках иногда появляются ложные ребра — ребра, которые не формируют границу плоского многоугольника и их не должно быть. Появляются, когда получается несколько многоугольников внутри области отсекателя.



Определить видимость относительно стороны отсекателя можно несколькими способами:

1. Используя произведение вектора внутренней нормали на вектор, соединяющий точку ребра отсекателя с исследуемой точкой (с последующей оценкой знака)
2. Использование пробной функции из уравнения прямой ($Ax+By+C$), проходящей через ребро отсекателя. Подставив координаты исследуемой точки в пробную функцию мы определим знак, после чего сравним его со знаком пробной функции точки, положение которой мы знаем (лучше не писать)
3. Проверка знака координаты z у векторного произведения двух векторов, лежащих в одной плоскости. Берутся две точки P_1 и P_2 из плоскости и интересующая нас точка P_3 . Получается плоскость, на которой лежат вектора P_1P_2 и P_1P_3 . Если считать её плоскостью xy , то у векторного произведения $P_1P_3 \times P_1P_2$ не нулевой будет только $z = (x_3-x_1)(y_2-y_1)-(y_3-y_1)(x_2-x_1)$. $>0 = 0 <0$ – значит точка справа, на или слева прямой соответственно

Чтобы определить **факт пересечения** ребром многоугольника прямой, проходящей через границу отсекателя, нужно удостовериться, что видимость первой точки ребра относительно границы отсекателя отличается от видимости второй точки. После установления этого факта можно применить любой из существующих алгоритмов определения точки пересечения. Например, Кируса-Бека или разбиения средней точкой.

В нашем случае это будет наиболее удобно выполнить, используя параметрическую форму задания:

$P(t) = P_1 + (P_2 - P_1)t, 0 \leq t \leq 1$ – ребро отсекаемого многоугольника

$Q(s) = Q_1 + (Q_2 - Q_1)s$ – граница отсекателя

Заметим, что мы ищем точку пересечения ребра отсекаемого многоугольника с прямой, проходящей через границу отсекателя. При этом легко заметить, что в таком случае мы не накладываем ограничения на параметр s .

Таким образом, чтобы найти требующуюся точку пересечения будем иметь:

$P(t) = Q(s)$

То есть получаем систему двух уравнений с двумя неизвестными, решаем его и получаем значение параметра соответствующей точки пересечения. После этого вычисляются x и y координаты точки пересечения.

18. Отсечение многоугольников невыпуклыми областями.

Алгоритм Вейлера-Азертона.

Позволяет производить отсечение произвольных многоугольников произвольными отсекателями, причём отверстия могут быть и там и там. Можно выполнять как внутреннее, так и внешнее отсечение. В результате новых ребер не появляется. Рёбра отсеченного многоугольника совпадают либо с участками ребер исходного либо многоугольника, либо отсекателя.

Направление обхода: внешние по часовой, внутренние — против часовой (таким образом, внутр. часть всегда справа).

Работа сводится к направленными кольцевыми списками.

Инициализация:

- Для каждой границы каждого многоугольника формируются двунаправленные списки. Общее количество списков определяется количеством внутренних границ (отверстий) + 2 (мин.).
- Найти точки пересечения отсекателя с многоугольником: решается полным перебором всех рёбер. Используем параметрическую форму задания отрезков. если $0 \leq t \leq 1$, от пересекаются рёбра, — иначе пересекаются прямые лежащие на ребрах.
- Найденные точки пересечения добавить на соответствующие места в ранее сформированные списки границ (сразу в два списка, так как принадлежит двум).

*Имеет смысл установить связь между элементами списка, содержащих информацию об одной и той же точке пересечения. (в итоге три связи в списке)

- Точки пересечения надо разделить на: точки входа (в которой ребро отсекаемого многоугольника входит внутрь отсекателя) и точки выхода (ребро отсекаемого многоугольника выходит из отсекателя наружу).

- Для получения внутреннего многоугольника, надо начинать просмотр с точки входа, для внешнего — с точки выхода.

Способ распознания точек входа и выхода происходит с помощью векторного произведения вектора стороны отсекателя. Если оно > 0 , то это точка входа. иначе - это точка выхода.

Крайние ситуации: Если число точек входа нечетно, то одна из точек является точкой касания (а не пересечения). Если точка пересечения совпадает с началом ребра, то это пересечение. С концом - касание.

Так же крайним случаем является ситуация, когда **границы отсекаемого многоугольника и отсекателя не пересекаются**.

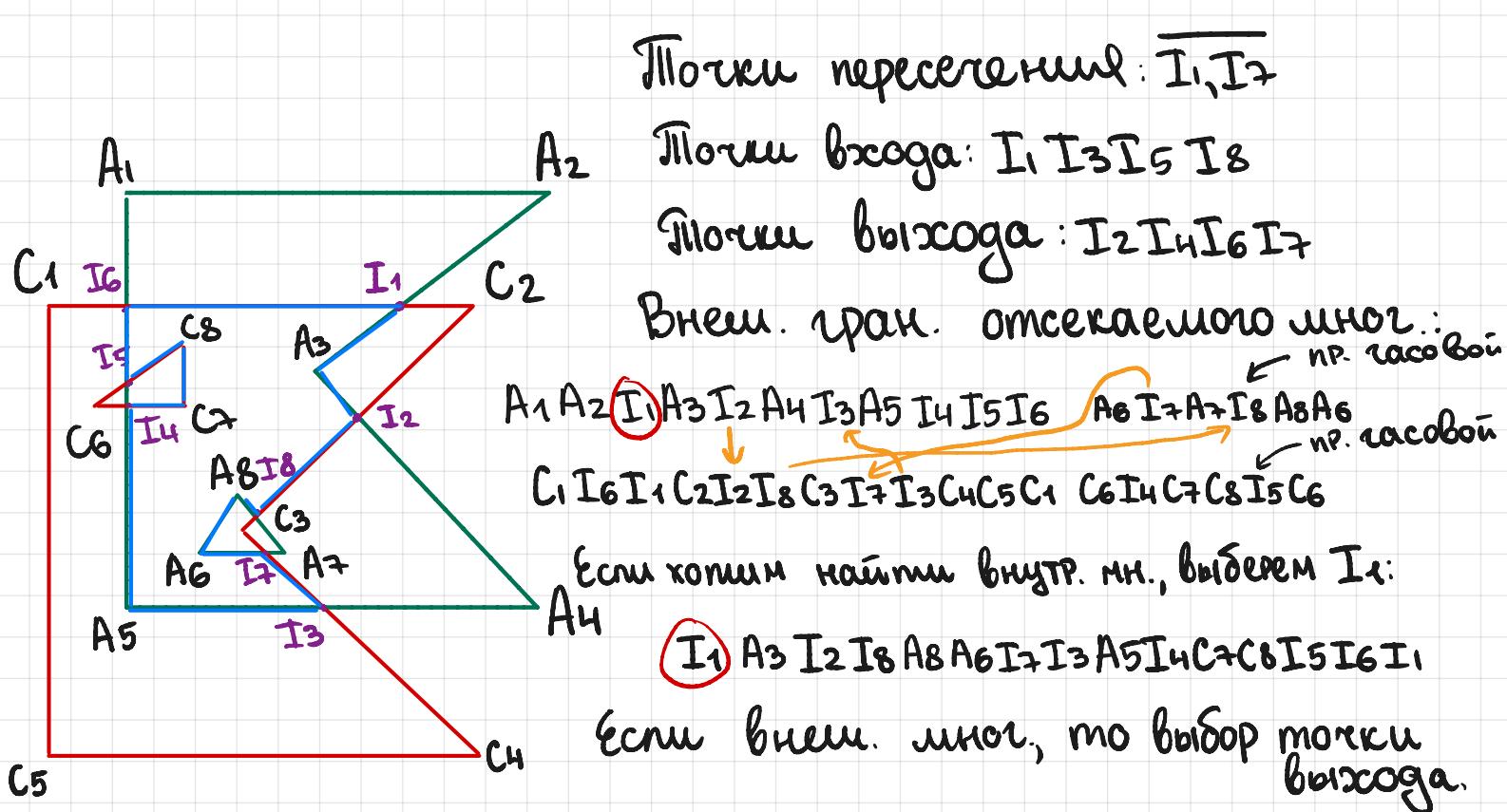
Алгоритм Сазеленда-Ходжмена справлялся с ним легко, здесь все не так просто.

Геометрически задача сводится к определению принадлежности пикселей заданной области. Мы испускаем луч из точки одного многоугольника и считаем количество пересечений с рёбрами другого.

- Если для обоих многоугольников получилось четное число — отсекаемый многоугольник внешний.
- Если для отсекателя четное, а для многоугольника нечетное, то отсекатель объемлет многоугольник.
- Если у отсекателя нечетное, а у отсекаемого четное, то многоугольник объемлет отсекатель.

Самый неприятный случай: вершина отсекаемого многоугольника расположена на ребре отсекателя. Та же проблема, что и в первом случае, надо обрабатывать кучу случаев при нахождении пересечений.

Чем простая модель отличается от глобальной? Простая модель не учитывает отражения от других объектов. **Какие отражения знаете?** **Какие из них в глобальной модели?** Есть диффузное отражение и зеркальное. В глобальной оба. **Какое отражение от других тел учитывает глобальная модель?** Зеркальное. **Чем создаётся интенсивность света?** Источником. **Типы многоугольников? И почему такие?** Цель? Внешние, внутренние, пересекающиеся, полностью охватывающие, когда с окном связаны несколько многоугольников Выделяем именно такие для того, чтобы понять, что нам изобразить в окне (каким цветом высвечивать) **Внешние:** Закрашиваем окно цветом фона **Внутренний один:** Закрашиваем цветом фона и выполняем растровую развертку для внутреннего многоугольник



19. Модели трёхмерных объектов. Требования, предъявляемые к объектам.

Модель является отображением формы и размеров объектов. Основное назначение модели - правильно отображать форму и модели объекта. В основном используется три вида моделей:

1. Каркасная (проволочная) модель:

- Простейший вид модели. Задаётся информацией о вершинах и ребрах.
- Не всегда правильно передаёт представление об объекте. (Пример с тессерактом: не понятно, где отверстия, какие грани между собой связаны).

2. Поверхностная модель

- Поверхность может описываться либо аналитически, либо другим способом. Например, хранить библиотеку поверхностей (хранить уравнения и задавать им коэффициенты).
- Отсутствует информация, о том, с какой стороны поверхности находится материал, а с какой - пустота.
- Если решаем вопрос моделирования трехмерных объектов, то алгоритм для нас сойдет, потому что нам все равно, где и что с какой стороны находится.

3. Объемная модель

- По факту это поверхность модель + информация, где находится материал. Проще всего это сделать путем указания внутренней нормали. В графике, как правило, мы работаем с поверхностными моделями. В более простом случае можем работать с каркасными моделями, а потом, заполучив результат, считать, что на этот каркас натянут материал

Требования к 3D-моделям.

1. Модель не должна противоречить исходному объекту.
2. Модель должна допускать возможность конструирования тела целиком (мощность модели).
3. Модель должна позволять вычисление геометрических характеристик тела.
4. Модель должна позволять производить расчеты.

Требования к 3D-моделям от Курова (лично требует):

1. Компактность - определяется количеством информации, которое необходимо задать и хранить для представления модели.
2. Не всегда компактность является основным параметром. Имеет смысл пойти на некую

избыточность в плане представления модели, если эта избыточность позволяет быстрее выполнять операции при использовании модели, то есть не вычислять каждый раз некоторые параметры, которые будут нужны при моделировании.

3. Желательно, чтобы каждая модель могла дополняться при расширении области применения данной модели, дополняться новыми свойствами.

Свойства объемных моделей

- 1) однородность (тело заполнено изнутри)
- 2) конечность (каждое тело занимает конечный объем)
- 3) жесткость (сплошное тело должно сохранять форму независимо в пространстве)

Требования к программам геометрического моделирования

- 1) Согласованность операций (любые операции проводимые над сплошным телом должны приводить к сплошным телам)
- 2) Возможность описания (любое тело должно представляться в машинном виде)
- 3) Непротиворечивость информации (любая точка пространства должна принадлежать только одному телу для любой точки можно сформулировать принадлежит ли она какому телу или нет)
- 4) Компактность модели (определяется количеством информации)
- 5) Открытость модели (разрабатываемая модель должна иметь применения при работе с разными алгоритмами)

20. Операции преобразования в трёхмерном пространстве.

Матрицы преобразований

Афинное преобразование - отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся - в пересекающиеся, скрещивающиеся - в скрещивающиеся, а также существует обратное преобразование.

- n -мерный объект отображается в n -мерный
- сохраняется параллельность линий и плоскостей
- сохраняются пропорции параллельных объектов

$$(x', y', z', 1) = (x, y, z, 1) M_{pr}.$$

Перенос

$$M_{pr} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}, \quad \begin{aligned} x_1 &= x + dx \\ y_1 &= y + dy \\ z_1 &= z + dz \end{aligned}$$

Поворот

Вокруг Ох

$$M_{pr} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= x \\ y_1 &= y_c + (y - y_c)\cos\theta - (z - z_c)\sin\theta \\ z_1 &= z_c + (y - y_c)\sin\theta + (z - z_c)\cos\theta \end{aligned}$$

Вокруг Оу

$$M_{pr} = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= x_c + (x - x_c)\cos\theta - (z - z_c)\sin\theta \\ y_1 &= y \\ z_1 &= z_c + (x - x_c)\sin\theta + (z - z_c)\cos\theta \end{aligned}$$

Вокруг Oz

$$M_{pr} = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= x_c + (x - x_c)\cos\theta - (y - y_c)\sin\theta \\ y_1 &= y_c + (x - x_c)\sin\theta + (y - y_c)\cos\theta \\ z_1 &= z \end{aligned}$$

Масштабирование

$$M_{pr} = \begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} x_1 &= k_x x + (1 - k_x)x_m \\ y_1 &= k_y y + (1 - k_y)y_m \\ z_1 &= k_z z + (1 - k_z)z_m. \end{aligned}$$

Проектирование на плоскость $z = 0$

$$M_{pr} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\frac{1}{c} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \begin{aligned} &\text{с - фокусное расстояние} \\ x_1 &= x \\ y_1 &= y \\ z_1 &= z \\ w_1 &= -\frac{z}{c} + w \end{aligned}$$

Перспективное преобразование - преобразование одного трёхмерного пространства в другое. Весь правый столбец матрицы отвечает за **20** проецирование на разные плоскости, т. е. для получения проекции на ось X, нужно было бы поместить $-1/C$ в первую строку. Проецируем на одну плоскость - одноточечное проецирование, дальше - двухточечные проекции и тд. **Являются перспективным преобразованием, не афинным!**

Коммутативность афинных преобразований

Перенос-перенос

Масштаб-масштаб

Поворот-поворот

Однородное масштабирование-поворот

Все остальные пары не коммутативны!

Для доп. вопросов: матрица 4×4 так как сдвиг - нелинейное (но афинное) преобразование, для выполнения которого требуется наличие свободного члена.

Для поворота/масштабирования относительно произвольной точки необходимо

1. Сделать перенос, совмещающий центр преобразования и начало координат
2. Выполнить преобразование (умножить на матрицу)
3. Выполнить обратный перенос для п. 1

21. Трехмерное отсечение. Виды отсекателей. Вычисление кодов концов отрезка для каждого типа отсекателей.

Алгоритм отсечения отрезков средней точкой.

Отсечение - операция удаления части изображения, которая находится за пределами некоторой заданной области, называемой отсекателем.

Для трехмерного отсечения наиболее распространенными формами отсекателей являются: прямоугольный параллелепипед и усеченная пирамида видимости (6 граней).

Видимый объект - объект, находящийся целиком внутри отсекателя.

Невидимый объект - объект, находящийся целиком вне отсекателя

Частично видимый объект - объект, часть которого находится внутри отсекателя, а часть - вне отсекателя.

При трехмерном отсечении используется **6-битовый код**:

$T_1 = 1$, если $x < x_{hl}$, 0 – иначе

$T_2 = 1$, если $x > x_{hp}$, 0 – иначе

$T_3 = 1$, если $y < y_{un}$, 0 – иначе

$T_4 = 1$, если $y > y_{uv}$, 0 – иначе

$T_5 = 1$, если $z > z_{zb}$, 0 - иначе

$T_6 = 1$, если $z < z_{hn}$, 0 - иначе

Если $code1 == 0$ и $code2 == 0$:

отрезок полностью видимый

Если $code1 \& code2 != 0$:

отрезок полностью не видим.

Если $code1 \& code2 == 0$:

Отрезок может оказаться как и частично видимым, так и полностью невидимым. Необходимо проверять пересечения отрезка с гранями отсекателя.

Для **усеченной пирамиды видимости** используются **пробные функции**, каждая из которых описывает одну грань. каждая грань задана плоскостью, которая через нее проходит. Пробная функция грани выводится из уравнения прямой на плоскости XZ, несущей проекцию этой грани.

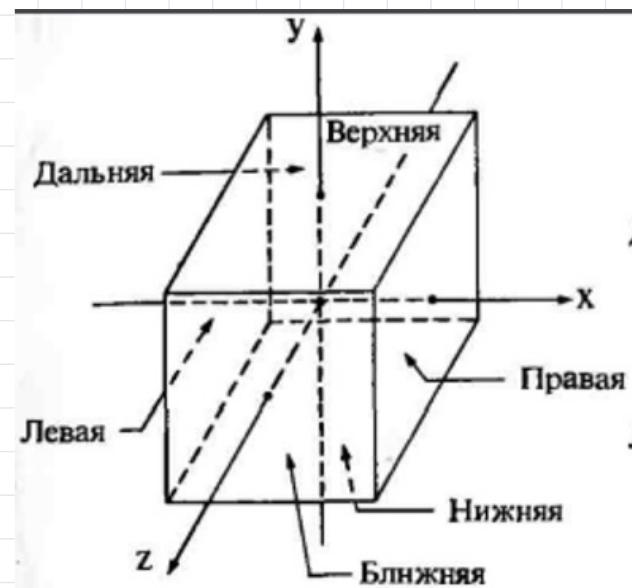
ZЦП - Центр проекции.

Уравнение прямой на плоскости xz, несущей проекцию правой грани отсекателя, имеет вид: $x = (z - Z_{цп}) \cdot x_p / (Z_g - Z_{цп}) = Z\alpha_1 + \alpha_2$, $\alpha_1 = x_p / (Z_g - Z_{цп}), \alpha_2 = -\alpha_1 Z_{цп}$

2. Перенесем все слагаемые в левую часть:

$$f_n = x - Z\alpha_1 - \alpha_2$$

3. При подстановке вместо x и z координат проверяемой точки в результате выражения получим значение пробной функции в этой точке:



a. Параллелепипед

$$f_n = x - z\alpha_1 - \alpha_2 \begin{cases} > 0, \text{ если } P \text{ СПРАВА от плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ < 0, \text{ если } P \text{ СЛЕВА} \end{cases}$$

21

Это для правой грани!

$$f_n = z\beta_1 - \beta_2, \text{ где } \beta_1 = (z_g - z_{цп}), \text{ а } \beta_2 = -\beta_1 z_{цп}.$$

$$f_B = y - z\gamma_1 - \gamma_2, \text{ где } \gamma_1 = \frac{y_B}{z_g - z_{цп}} \text{ и } \gamma_2 = -\gamma_1 z_{цп}.$$

$$f_H = y - z\delta_1 - \delta_2, \text{ где } \delta_1 = \frac{y_H}{z_g - z_{цп}}, \text{ а } \delta_2 = -\delta_1 z_{цп}.$$

$$f_\delta = z - z\delta; \quad f_g = z - z_g.$$

Так же стоит отметить, что при $z > z_{цп}$ возможна неопределенность, при которой точка будет одновременно правее правой и левее левой грани. Происходит это потому, что в точке центра проекций все грани пересекаются, а за ней инвертируются.

Чтобы это решить, Лианг и Барский предложили способ устранения неопределённости путём инвертирования первых четырёх бит кода

Алгоритм разбиения средней точкой заключается в поиске точки пересечения с помощью двоичного поиска. Вариант алгоритма для трех измерений отличается от двумерного варианта только размерностями массивов Ткод, Окно, и немного измененными подпрограммами Конец и Логическое с учетом трех измерений:

подпрограмма вычисления кода концевой точки отрезка относительно трехмерной усеченной пирамиды

subroutine Конец(P, Окно; Ткод, Сумма)

P_x, P_y, P_z — координаты x, y и z точки P
Окно — массив 1 × 7, содержащий координаты (x_д, x_п, y_н, y_в, z_б, z_л, z_{цп}) левой, правой, нижней, верхней, ближней, задней сторон окна и центра проекции

Ткод — массив 1 × 6, содержащий код концевой точки

Сумма — сумма элементов Ткод

вычисление α₁, α₂, β₁, β₂, γ₁, γ₂, δ₁, δ₂

$$\alpha_1 = x_p / (z_d - z_{цп})$$

$$\alpha_2 = -\alpha_1 z_{цп}$$

$$\beta_1 = x_l / (z_n - z_{цп})$$

$$\beta_2 = -\beta_1 z_{цп}$$

$$\gamma_1 = y_v / (z_d - z_{цп})$$

$$\gamma_2 = -\gamma_1 z_{цп}$$

$$\delta_1 = y_h / (z_d - z_{цп})$$

$$\delta_2 = -\delta_1 z_{цп}$$

определение кода концевой точки

if P_x - P_zβ₁ - β₂ < 0 then Ткод(6) = 1 else Ткод(6) = 0

if P_x - P_zα₁ - α₂ > 0 then Ткод(5) = 1 else Ткод(5) = 0

if P_y - P_zδ₁ - δ₂ < 0 then Ткод(4) = 1 else Ткод(4) = 0

if P_y - P_zγ₁ - γ₂ > 0 then Ткод(3) = 1 else Ткод(3) = 0

if P_z - z_б > 0 then Ткод(2) = 1 else Ткод(2) = 0

if P_z - z_д < 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



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

Так же, как и в двумерном варианте алгоритма, в трехмерном может рассматриваться произвольный выпуклый отсекатель. Использовать будем все то же параметрическое уравнение: $P(t) = P_1 + (P_2 - P_1)t$, где $0 \leq t \leq 1$ (уравнение прямой), но теперь все точки и векторы будут иметь три компоненты.

Видимость точек определяется относительно граней отсекателя, а не сторон, как в двумерном алгоритме. Способ тот же. Определение видимости точки P относительно грани $A_1B_1C_1D_1$:

- N_i - вектор внутренней нормали к i -й грани отсекателя
- f - произвольная точка любой грани отсекателя. Для удобства можно использовать любые угловые точки граней

Видимость точки относительно выпуклого

трехмерного отсекателя. Для определения

видимости вычислим скалярное

произведение двух векторов: N_i и $(P - f)$.

Второй вектор - это вектор соединяющий произвольную точку грани отсекателя и рассматриваемую точку.

Скалярное произведение для трехмерных

векторов можно рассчитать по формуле:

$$a \cdot b = ax * bx + ay * by + az * bz$$

Если произведение: > 0 - точка видима, < 0 -

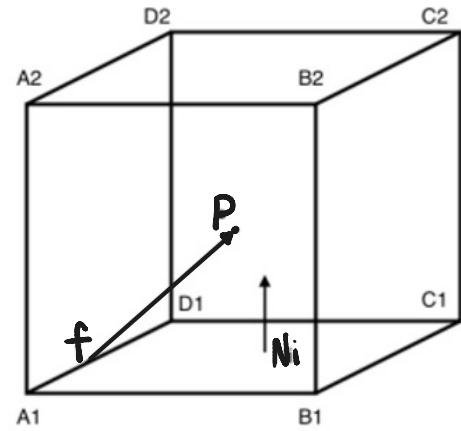
точка невидима, $= 0$ - точка лежит на грани.

Для отсечения целого отрезка будем анализировать вектор

направления отрезка(директрису) $D = P_2 - P_1$

Также для определения точек пересечения с гранями отсекателя нам потребуется значение параметра t , которое считается по аналогии с двумерным алгоритмом отсечения $t = -W_{ck} / D_{ck}$, где

- W_{ck} - скалярное произведение вектора $W_i = (P - f)$ на вектор нормали i -й грани N_i
- D_{ck} - скалярное произведение директрисы на вектор нормали i -й грани N_i



Частные случаи

- $D = 0$ - отрезок вырождается в точку
- $D_{ck} = N_i * D = 0$ - отрезок параллелен i -й грани отсекателя
- $W_{ck} = N_i * W_i \geq 0$ - отрезок расположен по видимую сторону отсекателя
- $W_{ck} = N_i * W_i < 0$ - отрезок расположен по невидимую сторону отсекателя, и тогда отрезок является полностью невидимым

Нахождение точки пересечения

Анализируя знак D_{CK} можно сказать о положении точки пересечения относительно начала и конца отрезка. Если:

- $D_{CK} > 0$ - точка расположена ближе к началу отрезка
- $D_{CK} < 0$ - точка расположена ближе к концу отрезка
- $D_{CK} = 0$ - точка расположена посередине отрезка

Соответственно в случае, если $D_{CK} > 0$, текущее значение t выбирается, как максимальное между текущим и вычисленным $t = -W_{CK} / D_{CK}$. Иначе же значение t выбирается, как минимальное между текущим и вычисленным.

- $D_{CK} > 0 \rightarrow t_h = \max(t_h, t)$

После нахождения точек пересечения необходимо выполнить проверку $t_h \leq t_b$, чтобы убедиться в том, что начало видимой части не расположено за концом отрезка

Алгоритм:

1. Ввод $P_1, P_2, K, C(K)$

2. $t_h = 0, t_b = 1$

3. $D = P_2 - P_1$

4. проверка отсекателя на выпуклость

5. Цикл отсечения (по i от 1 до K)

 a. $W_i = P_1 - f_i$

 b. Вычисление D_{CK}, W_{CK}

 c. Если $D_{CK} = 0$, и если $W_{CK} < 0$, то отрезок невидим

 d. $t = -W_{CK} / D_{CK}$

 e. Если $D_{CK} > 0$, то если $t > 1$, то отрезок невидим, иначе $t_h = \max(t, t_h)$

 f. Если $D_{CK} \leq 0$, то если $t < 0$, то отрезок невидим, иначе $t_b = \min(t, t_b)$

 g. Конец цикла

6. Если $t_h \leq t_b$, то изобразить отрезок (t_h, t_b)

7. Конец

1) **Что такое отсечение?** Отсечение – это операция по удалению невидимых частей изображения по отношению к отсекателю.

2) **Какую задачу позволяет решить алгоритм Кирус?** Отсечение отрезков выпуклым отсекателем.

3) **Что вычисляется для решения этой задачи?** Мы вычисляем нормали к грани, используем параметрическое уравнение отрезка.

4) **Что дальше делаете с этими векторами?** Для каждой грани считаем 2 скалярных произведения.

5) **Каким образом можно установить невидимость отрезка с помощью этого алгоритма?** Если произведение вектора, соединяющего рассматриваемую точку с точкой на грани и нормали < 0 , то отрезок невидим. (точка пересечения находится вне грани)

23. Определение факта выпуклости трехмерных тел.

Разбиение тела на выпуклые многогранники.

Определение факта выпуклости трехмерного тела

1. Перенести тело так, чтобы одна из вершин грани оказалась в начале координат.
2. Повернуть тело относительно начала координат так, чтобы одна сторона из двух смежных выбранной вершине сторон грани совпала с одной из осей координат, например с осью x.
3. Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань легла на координатную плоскость, например на плоскость $z = 0$
4. Для всех вершин тела, не принадлежащих выбранной грани, проверить знаки координаты, которая перпендикулярна этой грани. Здесь это будет координата z

Если эти знаки для всех вершин совпадают или равны нулю, то тело будет выпуклым относительно выбранной грани. Если тело выпукло относительно всех своих граней, то оно считается выпуклым, - в противном случае тело невыпукло.

Если для всех вершин значения координаты, перпендикулярной выбранной грани, равны нулю, то тело вырождено; т.е. оно плоское.

Вектор внутренней нормали к выбранной плоскости, заданный в повернутой системе координат, имеет все нулевые компоненты, кроме той, которая перпендикулярна этой плоскости. Знак этой компоненты для выпуклой грани будет совпадать с ранее найденным знаком.

Для определения искомой ориентации внутренней нормали в исходной системе координат нужно применить к ней только обратное преобразование поворотов.

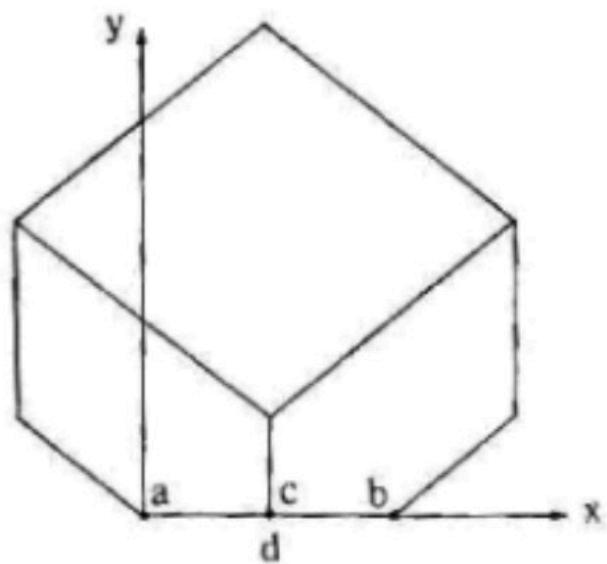
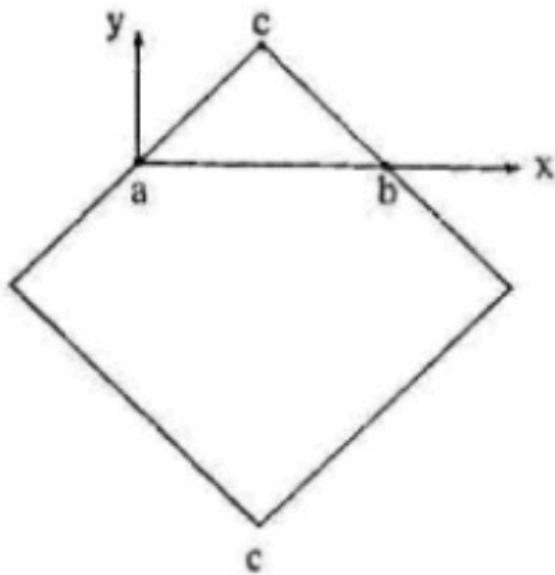
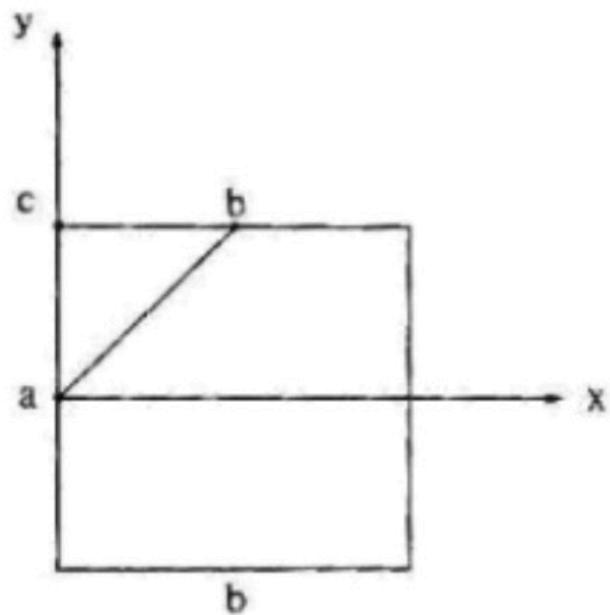
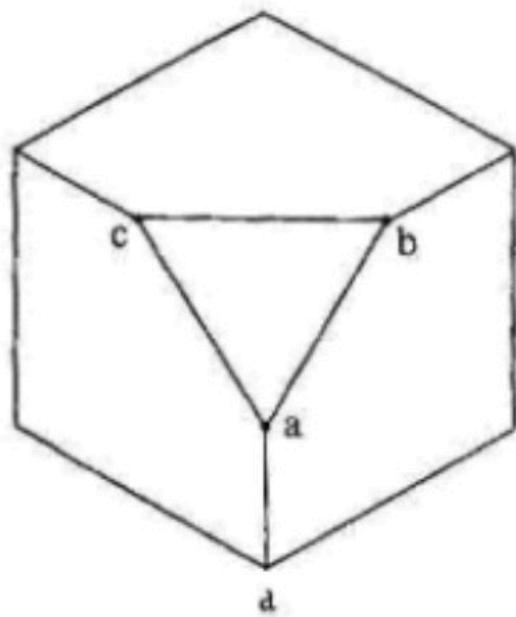
Разбиение тела на выпуклые многогранники

1. Перенести тело так, чтобы одна из вершин выбранной грани совпала с началом координат.
2. Повернуть тело вокруг начала координат так, чтобы одно из инцидентных ему ребер совпало с одной из осей координат, например с осью x
3. Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань совпала с одной из координатных плоскостей, например с плоскостью $z = 0$

4. Проверить знаки координаты, которая перпендикулярна выбранной грани (т.е. координаты z), для всех вершин тела, не лежащих на это **23** грани.

5. Если все эти знаки совпадают или равны нулю, то тело является выпуклым относительно этой грани. В противном случае оно невыпукло; разрезать тело плоскостью, несущей выбранную грань.

Повторить всю процедуру с каждым из вновь образовавшихся тел. Продолжать работу до тех пор, пока все тела не станут выпуклыми.



24. Алгоритм плавающего горизонта.

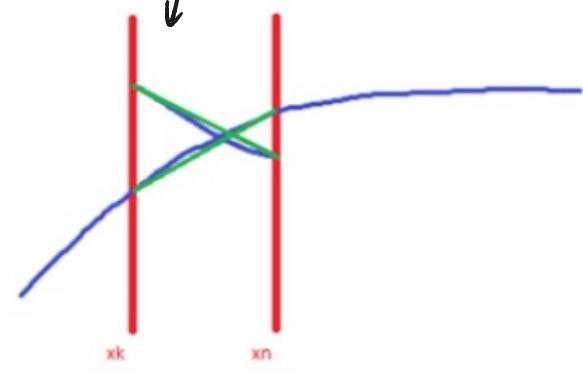
Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде $F(x, y, z) = 0$.

Суть алгоритма: рассматриваемая поверхность рассекается плоскостями, перпендикулярными оси Z. В каждом отсечении получается кривая. Эта кривая описывается уравнением $y=f(x, z=\text{const})$ или $x=Q(y, z=\text{const})$. Полученные кривые можно проецировать на плоскость на плоскость $z=0$ и изобразить видимые части каждой кривой. Анализ начинается от кривой, полученной в ближайшем к наблюдателю сечении. Для определения видимости точек будут использоваться верхний и нижний горизонты - видимые участки кривых с наибольшим и наименьшим значением ординат соответственно. если $y(x, z=\text{const}) > y_{\max}$, то $y_{\max} = y(x, z=\text{const})$ или если $y(x, z=\text{const}) < y_{\min}$, то $y_{\min} = y(x, z=\text{const})$. В обоих случаях точка является видимой. Если оба условия не выполнены - точка невидима.

* Горизонты по факту представляются массивами с минимальными и максимальными значениями ординат

Поиск точек пересечения кривых на интервалах:

green



Пусть зеленым отмечен некоторый горизонт

Т.к. Δx (шаг) достаточно мало, то можно воспользоваться аппроксимацией кривой отрезком и через уравнения прямых найти точку пересечения отрезков.

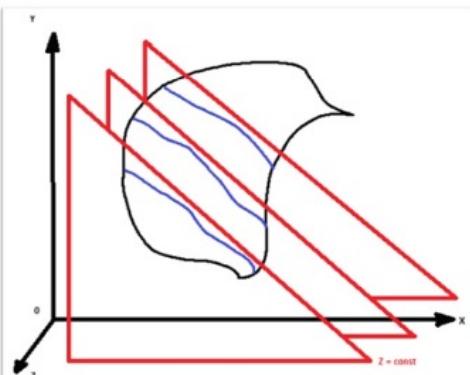
$$m = (y_n - y_k) / (x_n - x_k)$$

- тангенс угла наклона прямой

Далее можем воспользоваться уравнением прямой: $y = m(x - x_k) + y_k$. Приравняем ординаты точки пересечения:

$$m_{\text{пред}}(x - x_k) + y_{\text{пред}} = m_{\text{тек}}(x - x_k) + y_{\text{тек}}$$

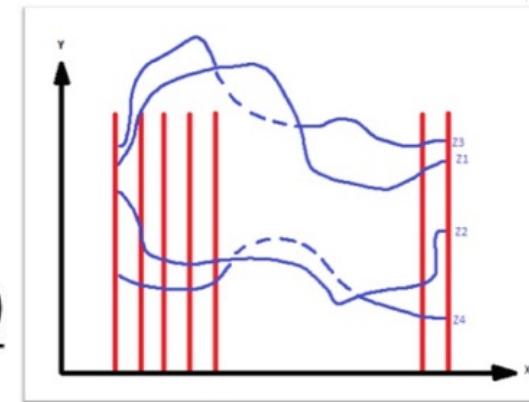
$$x(m_{\text{пред}} - m_{\text{тек}}) = x_k(m_{\text{пред}} - m_{\text{тек}}) + (y_{\text{тек}} - y_{\text{пред}})$$



$$x = x_k + \frac{y_{\text{тек}} - y_{\text{пред}}}{m_{\text{пред}} - m_{\text{тек}}}$$

$$x = x_k - \frac{y_{\text{тек}} - y_{\text{пред}}}{m_{\text{тек}} - m_{\text{пред}}}$$

$$x = x_k - \frac{dx(y_{\text{тек}} - y_{\text{пред}})}{dy_{\text{тек}} - dy_{\text{пред}}}$$



Таким образом, если текущая точка кривой невидима, то изображается участок кривой от предыдущей точки до найденной точки пересечения. Если текущая точка кривой видима, то изображается участок кривой от найденной точки пересечения до текущей точки.

24

Улучшение качества изображения поверхности

Для улучшения восприятия картинки добавляют боковые ребра путем соединения начальных и конечных точек каждой кривой друг с другом, например, отрезками.

Алгоритм

Для каждой плоскости $z = \text{const}$:

1. Обработать левое боковое ребро.
2. Для каждой точки, лежащей на кривой из текущей плоскости:
 - a. Если при некотором заданном значении x , соответствующее значение y на кривой больше максимума или меньше минимума по y для всех предыдущих кривых при этом x , то кривая видима (в этой точке). В противном случае она невидима.
 - b. Если на сегменте от предыдущего (X_n) до текущего (X_{n+k}) значения x видимость кривой изменяется, то вычисляется пересечение (X_i)
 - c. Если на участке от X_n до X_{n+k} сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается его кусок от X_n до X_i , если же он стал видимым, то изображается его кусок от X_i до X_{n+k} .
3. Заполнить массивы верхнего и нижнего плавающих горизонтов
4. Обработать правое боковое ребро.

Обработка левого бокового ребра:

Если P_n является первой точкой на первой кривой, то $P_{n-1} = P_n$

Иначе создадим ребро (P_{n-1}, P_n), обновим $P_{n-1} = P_n$. (Координаты ребра внесем в массива верхнего и нижнего горизонтов)

Обработка правого бокового ребра:

Если P_n является последней точкой на первой кривой, то $P_{n-1} = P_n$

Иначе создадим ребро (P_{n-1}, P_n), обновим $P_{n-1} = P_n$. (Координаты ребра внесем в массива верхнего и нижнего горизонтов)

Недостатки алгоритма

- 1) 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).
- 2) Если функция содержит пики (очень острые участки), то алгоритм может давать некорректные результаты при малом шаге. Если для корректного изображения требуется шаг, меньший, чем разрешающая способность экрана, то видимыми могут считаться некоторые невидимые отрезки.
- 3) Алгоритм хранит горизонты и постоянно их обновляет, из-за этого он недостаточно быстродействующий.
- 4) При любом изменении угла обзора происходит полный перерасчет.

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 линий 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.

Сортировка по горизонтали и вертикали (выполняется после главной) - нужна чтобы выяснить, будет ли рассматриваемый объект действительно заслонен объектом, расположенным ближе к точке наблюдения.

Алгоритмы можно классифицировать по:

- способу выбора системы координат
- пространства, в котором они работают

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. Тут получаются точные результаты. Такой подход лучше применять для несложных сцен.

ГРУБАЯ ОЦЕНКА СЛОЖНОСТИ: $O(N^2)$. N - число объектов.

Алгоритмы, работающие в пространстве изображений, имеют дело с системой координат того экрана, на котором объекты визуализируются. Точность вычислений ограничена разрешающей способностью экрана.

ГРУБАЯ ОЦЕНКА СЛОЖНОСТИ: $O(nN)$, N - число объектов, n - число пикселей.

Чисто теоретически, большинство алгоритмов следует реализовывать в объектном пространстве ($N^2 < nN$). Однако на практике это не всегда так, т.к. алгоритмы второго типа более эффективны, потому что там проще воспользоваться когерентностью при растровой реализации (близко расположенные пиксели чаще обладают одинаковыми свойствами)

25

Когерентность сцены - тенденция неизменяемости характеристик сцены в целом.

В каких пространствах мы можем решать её? В объектном и в экранных координатах. Где лучше? Точнее получается в объектном, там мы работаем в мировой системе координат (вещественные числа). Но связанные алгоритмы (Робертс) долго выполнимы. В экранной системе будут быстрее за счёт учёта свойства когерентности (цвет пикселя будет меняться при пересечении одной кривой с другой (нахождение пересечения))

26. Алгоритм Робертса. Основные этапы и математические основы каждого этапа.

Алгоритм Робертса удаляет невидимые линии. Для него необходимо чтобы все изображаемые тела были выпуклыми. Если тело невыпуклое необходимо разбить его на выпуклые составляющие

Основные этапы решения задачи:

- Подготовка исходных данных
- Удаление рёбер (или граней), экранируемых самим телом
- Удаление рёбер, экранируемых другими телами
- Удаление линий пересечения тел, экранируемых самими телами, связанными отношением прокалывания и другими телами

В этом алгоритме выпуклое

многогранное тело должно

представляться набором

пересекающихся плоскостей.

Уравнение произвольной плоскости

имеет вид:

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

Тогда любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскости. Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела (при подстановке координат точки в уравнение результат положителен). Если для очередной грани это условие не выполняется, то соответствующий столбец матрицы надо умножить на -1. Как тестирующую точку можно взять барицентр. Если вдруг необходимо также преобразовать тело, то надо матрицу исходного тела слева умножить на обратную матрицу преобразования: $[VT] = [T]^{-1}[V]$

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

Этап 1. Удаление ребер, экранируемых самим телом

Необходимо определить вектор взгляда наблюдателя, если наблюдатель располагается в +бесконечности на оси Z и смотрит в сторону начала координат: $E = (0, 0, -1, 0)$ = вектор взгляда наблюдателя.

Для определения невидимых граней следует вектор E умножить на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням. Невидимые ребра образуются пересечением невидимых граней. Если тело одно, алгоритм закончен.

Этап 2. Удаление невидимых ребер, экранируемых другими телами

Введем новый вектор (типа), координаты точки в которой находится наблюдатель: $g = (0, 0, 1, 0)$ Исследуем какое-нибудь ребро, чтобы определить его видимость начинаем "пускать лучи", если наблюдатель на +бесконечности, то лучи распространяются параллельно оси Z.

Точка, на этом отрезке, является невидимой, если луч встретил преграду из тела, то есть прошел через него. Если луч располагается по положительную сторону от каждой грани тела, значит он проходит сквозь тело. Для того чтобы найти все невидимые точки рассмотрим наше ребро в параметрической форме: $p(t) = P_1 + (P_2 - P_1) * t$, где P_1 и P_2 начало и конец ребра, а $0 \leq t \leq 1$. Используя его запишем уравнение луча (которое можно считать уравнением плоскости проходящим через луч и отрезок) $Q(t, \alpha) = P(t) + \alpha * g$, где $P(t)$ произвольная точка на ребре, а $\alpha \geq 0$ (что означает что наблюдать находится перед телом, при отрицательных значениях наблюдатель расположен с другой стороны тела) Уравнение луча соединяет точку наблюдателя с произвольной точкой на луче. $H = Q * V, h_j > 0, j=1,n$ (то есть луч располагается по положительную сторону от каждой грани тела, h_j это нормали, а n – количество ребер). Невидимым точкам ребра P_1P_2 соответствуют такие значения параметров t и α , при которых выполняются все неравенства. Система неравенств задает область допустимых решений. Все точки расположенные внутри области определяют координаты невидимых точек отрезка. Нам надо найти в этой области минимальное и максимальное значение параметра t . 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. Целевая функция линейна – производная постоянна, поэтому минимальное и максимальное значения достигаются на границах области.

Этап 3. Удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами

Решения на границе $\alpha = 0$ возникают в случае протыкания. Решается протыкание при помощи запоминания всех точек протыкания и добавлении к сцене отрезков, связывающих эти точки. Отрезки образуются путем соединения каждой точки протыкания со всеми остальными точками протыкания для этой пары объектов. Затем проверяется экранирование этих отрезков данными и другими телами. Видимые отрезки образуют структуру протыкания.

27. Алгоритм Робертса. Формирование матрицы тела.

Удаление нелицевых граней.

Подготовка исходных данных.

Формирование матрицы тела. Для каждого тела сцены должны быть сформирована матрица тела. Пусть V - матрица тела, которая имеет размерность $4 \times n$ где n - кол во граней. Каждый столбец матрицы представляет собой 4 коэффициента уравнения плоскости, проходящей через очередную грань тела. Общий вид уравнения плоскости:

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

Поиск коэффициентов: (решение системы)

$$\begin{aligned} 1. \quad ax_1 + by_1 + cz_1 + d_1 &= \\ &= 0 \sim a'x_1 + b'y_1 + c'z_1 = \\ &= -1, \text{ где } a' = a/d \end{aligned}$$

$$2. \quad ax_2 + by_2 + cz_2 + d_2 = 0$$

$$3. \quad ax_3 + by_3 + cz_3 + d_3 = 0$$

$$4. \quad ax_4 + by_4 + cz_4 + d_4 = 0$$

(Т.к. плоскость однозначно задается тремя точками, не лежащими на одной прямой, то мы можем поделить на $d \neq 0$)

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \Rightarrow [X][C] = [D] \Rightarrow [C] = [X]^{-1}[D]$$

Откуда мы находим координаты вершин соотв. грани, то коэф-ты равны:

$$a = \sum_{i=0}^n (y_i - y_j)(z_i + z_j)$$

$$b = \sum_{i=0}^n (x_i - x_j)(z_i + z_j) \text{ где } j = i+1, \text{ если } i < n, j = 1, \text{ если } i = n$$

$$c = \sum_{i=0}^n (x_i - x_j)(y_i + y_j)$$

Зная уравнение нормали для плоскости $\underline{n} = a\underline{i} + b\underline{j} + c\underline{k}$, находим a, b, c , а

$d = -(ax_1 + by_1 + cz_1)$ для известной точки.

Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. Если для очередной грани это условие не выполняется, то соответствующий столбец матрицы надо умножить на -1 .

Для проверки корректности матрицы тела следует взять точку, расположенную внутри тела. Координаты такой точки можно получить усреднением координат всех вершин тела или взяв полусумму максимальной и минимальной координат. S – вектор координат точки.

Находим произведение вектора S на матрицу тела. В полученном векторе надо найти отрицательные элементы, они соответствуют некорректно сформированным уравнениям плоскостей. Соответствующий столбец надо умножить на -1 .

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

Для получения матрицы преобразованного тела надо матрицу исходного тела слева умножить на обратную матрицу преобразования.

27

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

Удаление ребер, экранируемых самим телом (Удаление нелицевых граней тела)

Должны понимать, что нелицевые ребра расположены в нелицевых гранях тела или по другому говорят, что нелицевые ребра образуется пересечением невидимых нелицевых граней тела.

Для решения задачи надо указать расположение наблюдателя и направление его взгляда. Наблюдатель располагается в +бесконечности на оси Z и смотрит в сторону начала координат. $E = (0, 0, -1, 0)$ = вектор взгляда наблюдателя. С одной стороны, E - вектор взгляда наблюдателя, с другой стороны - это вектор однородных координат точки, расположенной в -бесконечности по оси Z. Для определения невидимых граней следует вектор Е умножить на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням. Невидимые ребра образуются пересечением невидимых граней.

Итоги этапа:

1. Если тело в сцене одно, то решив задачу поставленную на этом этапе мы решим и всю задачу целиком. Работа алгоритма закончится.
2. Решенную задачу можно интерпретировать и по-другому, чтобы определить видимые или невидимые грани достаточно найти скалярное произведение вектора взгляда на вектор внутренней нормали грани. Этот подход используется для построения реалистичных изображений, то есть когда мы должны закрашивать грани тел, поверхностей, в данном случае многогранников

Легко задача решается графически. Сначала надо построить область допустимых решений. Для этого от неравенств переходим к равенствам. Каждое равенство будет определять прямую.

Совокупность пересекающихся прямых даст многоугольник области допустимых решений. В этой области найти минимальные и максимальные значения параметра t . $t \rightarrow \min$ и $t \rightarrow \max$. Целевая функция линейна - производная постоянна, поэтому минимальное и максимальное значения достигаются на границах области

Этап 3. Удаление линий пересечения тел, экранируемых самими телами, связанными отношением прокалывания и другими телами последовательность действий:

- если не было обнаружено точек прокалывания - визуализируем
- формируем все возможные ребра, соединяющие точки прокалывания, для пар тел, связанных отношением прокалывания
- проверить экранирование всех соединяющих ребер обоими телами, связанных отношением прокалывания
- проверить экранирование оставшихся соединяющих ребер всеми прочими телами сцены. оставшиеся отрезки будут видимыми - добавим их к ответу, визуализируем.

1. Зачем ищем точки прокалывания? Если тела взаимно прокалываются, то образуются новые рёбра, которые могут быть экранированы как самими телами, так и другими объектами. Мы эти рёбра должны тоже получить и исследовать.

2. Где и как искать точки прокалывания?

Мы составляем систему и решаем её. Находим область допустимых решений. Точки прокалывания располагаются на оси t при $\alpha = 0$.^{**} Почему они там?^{**} Эти точки прокалывания находятся на основном отрезке. (Вставка от Курова) Точка прокалывания принадлежит не только телу, но и самому отрезку. Уравнение луча получается комбинацией условно говоря уравнения ребра и точки наблюдения. (вставка от Курова закончилась). При α равном 0 получим уравнение самого отрезка. Точка прокалывания ему принадлежит. Какое значение надо найти на оси α ? На оси надо найти максимальное и минимальные значение.

29. Удаление невидимых линий и поверхностей в пространстве изображений. Алгоритм Варнока (разбиение окнами): последовательность действий и основные принципы.

Основная идея алгоритма Варнока:

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. Соответственно в алгоритме Варнока и его вариациях делается попытка извлечь преимущество из того факта, что большие области изображения однородны.

Если непонятно, что изображать в окне, **делим его на подокна**. Принятие решения о моменте окончания деления принимается в зависимости от версии алгоритма. **В простейшей версии окна делятся всякий раз, пока они не пусты**. Окно является пустым, когда все многоугольники являются внешними по отношению к этому окну. До тех пор, пока размер окна $>$ размера пикселя. После этого определяем глубину каждого из рассматриваемых окон и изображаем наиболее близкую к пользователю точку. В более сложных версиях, ставится задача определения что изображать в очередном окне > 1 пикселя. Может работать как с ребрами, так и с поверхностями. Самый простой вариант делить на 4 прямоугольных окна. Дополнительная функция алгоритма — устранение лестничного эффекта: продолжаем делить окно на части, меньше пикселя, в последствии усредняя найденные характеристики.

29

30. Типы многоугольников, анализируемых в алгоритме Варнока.

Методы их идентификации.

Способы расположения многоугольников относительно окна

- Внешний - целиком вне окна
- Внутренний - целиком внутри окна
- Пересекающий - пересекает границу окна
- Охватывающий - окно находится целиком внутри многоугольника
- С окном связано несколько многоугольников

Правила обработки окна

1. Если все многоугольники сцены являются внешними: 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 многоугольник, связанный с окном, является пересекающим – отсечение по границе окна и п. 2 4. Если окно охвачено ровно одним многоугольником: залить окно цветом многоугольника
5. Если с окном связано несколько многоугольников, но ближе всех прочих к наблюдателю расположен охватывающий: заполнить окно цветом охватывающего многоугольника

Методы идентификации

- **внешний** (не для всех, только если с окном связан один многоугольник) - габаритная проверка (с прямоугольной оболочкой)
 $x_{\max} < x_{\min} \text{ || } x_{\max} > x_{\max} \text{ || } y_{\max} < y_{\min} \text{ || } y_{\max} > y_{\min}$ этот тест не распознает внешние многоугольники, которые огибают окно
- **внутренний** - габаритная проверка (с прямоугольной оболочкой)
Многоугольник может быть и выпуклый и невыпуклый.
 $x_{\min} \geq x_{\min} \text{ & } x_{\max} \leq x_{\max} \text{ & } y_{\min} \geq y_{\min} \text{ & } y_{\max} \leq y_{\max}$
- **пересекающий** — тест на пересечение. определяем потенциальное пересечение: подставляем все вершины окна во все уравнения прямых, задающих ребра многоугольника. если знак пробной функции (уравнения прямой) 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.
- **охватывающий** — тест с бесконечной прямой:
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, и прорезание - как одно.

- **несколько многоугольников** — поиск ближайшего охватывающего многоугольника. вычисляем высоту z многоугольника в каждой вершине окна (используем уравнение плоскости). если глубина охватывает многоугольника больше глубин всех других в каждой из вершин, то он расположен ближе остальных. красим в его цвет

30

Возможные оптимизации

- Сортировка многоугольнико по z
- Хранение информации. Например если для окна данный многоугольник охватывающий - запомнить и не проверять на подокнах (подонках).
- Списки охватывающих, пересекающих и внутренних (запоминаем на каком уровне они появились, для обхода по дереву окон пригодится)

В каком пространстве работаем? В пространстве изображения.

Есть ли единая версия алгоритма Варнока? Нет, одной единственной нет.

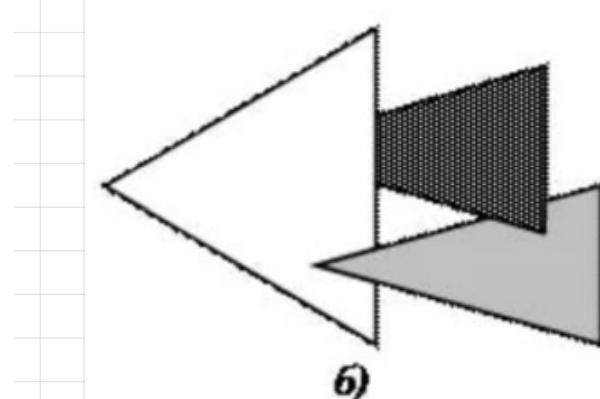
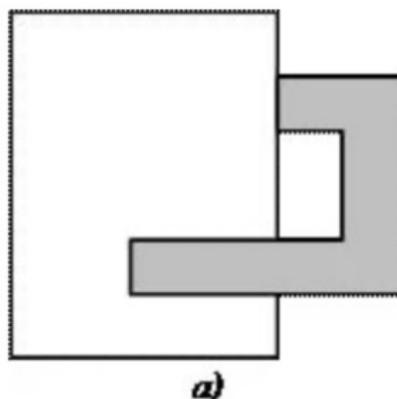
Как вы будете определять какой многоугольник ближайший к наблюдателю?

По координате z .

31. Алгоритм Вейлера Азертона удаления невидимых линий и поверхностей.

Алгоритм – попытка минимизировать количество шагов в алгоритме Варнока путем разбиения окна вдоль границ многоугольника.

1. Предварительная сортировка по глубине (для формирования списка приблизительных приоритетов). если наблюдаем из $z=+\infty$, то первым рассмотрим тот многоугольник, который имеет вершину с максимальной координатой z (ближе всего к нам). Удобно сортировать по Z_{\max}
2. Отсечение по границе ближайшего к наблюдателю многоугольника, называемое сортировкой многоугольников на плоскости, или ху-сортировка (в качестве отсекателя используется копия первого многоугольника из списка приблизительных приоритетов. отсекаться будут все многоугольники в этом списке, включая первый). Используется алгоритм отсечения Вейлера-Азертона. На выходе формируется 2 списка – внутренний (для каждого отсекаемого многоугольника та часть, которая оказывается внутри отсекателя) и внешний (оставшаяся часть)).
3. Удаление многоугольников внутреннего списка, которые экранируются отсекателем (все вершины имеют z_{\max} не больше, чем z_{\min} отсекателя). Если все многоугольники были экранированы – продолжим работу с внешним списком.
4. Если глубина многоугольника из внутреннего списка больше, чем Z_{\min} отсекателя, то такой многоугольник частично экранирует отсекатель. Нужно рекурсивно разделить плоскость, используя многоугольник, нарушивший порядок в качестве отсекателя (нужно использовать копию исходного, а не остаток после предыдущего отсечения). Отсечению подлежат все многоугольники из внутреннего списка, включая предыдущий отсекатель.
5. По окончанию отсечения или рекурсивного разбиения изображаются многоугольники из внутреннего списка (те, которые остались после удаления всех экранируемых на каждом шаге многоугольников – остаются только отсекающие многоугольники)
6. Работа продолжается с внешним списком (шаги 1-5)

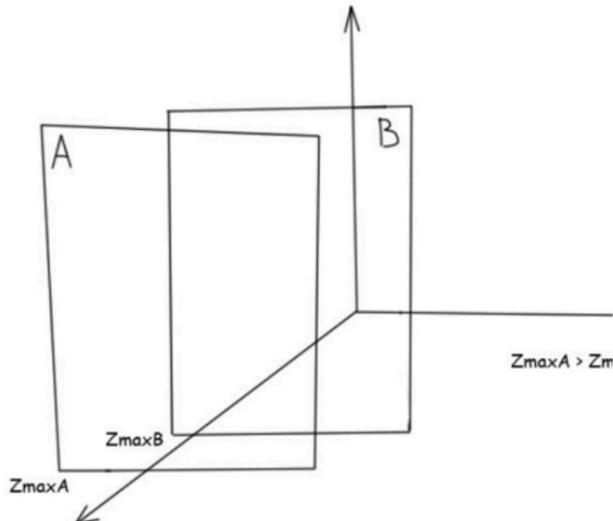


Примечание. Если многоугольник циклически перекрывается с отсекающим (лежит как спереди, так и сзади него), нет необходимости в рекурсивном разбиении, т.к. все лишнее уже было удалено прежде. Для избежания таких циклов храним список многоугольников, по которым мы уже отсекали. если при запросе отсечения по многоугольнику мы нашли его в списке - был найден циклически перекрывающийся многоугольник и разбивать ничего не надо.

Алгоритм отсечения позволяет найти как внутренние, так и внешние отсечения, что важно. Этот алгоритм справляется с циклическим прерыванием многоугольников (часть многоугольников лежит позади другого, а часть спереди). Если многоугольники протыкаются, то придется один из многоугольников разбить на два линией пересечения этих многоугольников.

31

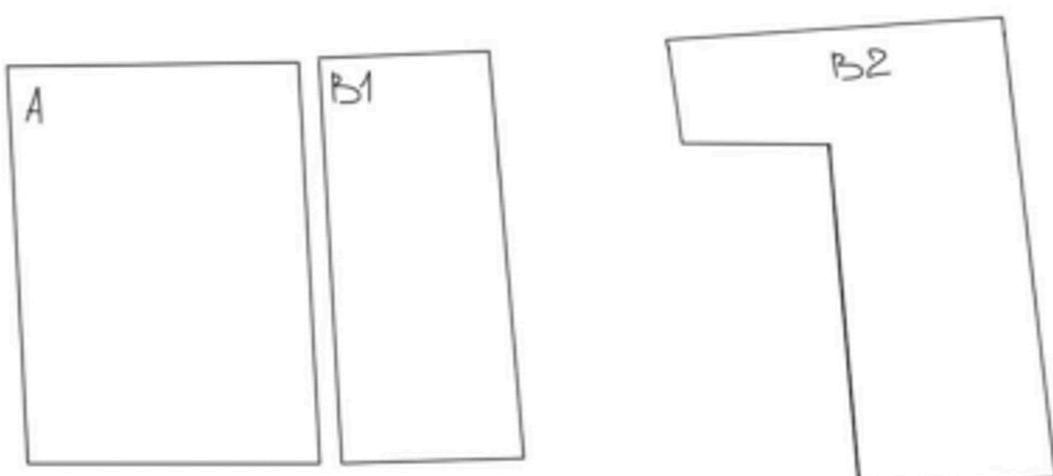
1. Сортировка (A ближе B)



2. Отсечение по границам многоугольника A.

Список внутренних многоугольников: A + какая-то небольшая часть многоугольника B - B1

Список внешних многоугольников: вторая часть многоугольника B - B2



32. Алгоритм, использующий Z-буфер.

- Это один из простейших алгоритмов удаления невидимых поверхностей.

- Работает в пространстве изображений.**

- В этом алгоритме используется идея о буфере кадра.

Буффер кадра используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения.

В данном алгоритме используется два буфера: буффер регенерации и собственно сам z-буффер, куда можно помещать информацию о координате z для каждого пикселя.

В начале z-буффер кладут минимально возможные значения z, а в буффер регенерации кладут пиксели, описывающие фон.

Затем каждый многоугольник приводят к растровому виду, и записывают в буффер регенерации (без упорядочивания)

В процессе подсчета глубины нового пикселя (который надо занести в буффер кадра), сравнивается с тем значением, что лежит в z-буффере.

Если новый пиксель расположен ближе, то он заносится в буффер кадра, при этом происходит корректировка z-буффера: в него заносится глубина нового пикселя. В сущности, алгоритм для каждой точки ищет максимальное значение z для каждой точки (x, y).

Вычисление глубины z

Многоугольник описывается уравнением: $Ax + By + Cz + D = 0$, отсюда получаем:

$z = -(Ax + By + D) / C$ При $C = 0$ - многоугольник для наблюдателя вырождается в линию.

Ясно, что для сканирующей строки $y = \text{const}$. Поэтому, можем рекуррентно считать z' для каждого $x' = x + dx$. $z' - z = -(ax' + d) / c + (ax + d) / c = a(x - x') / c$, откуда $z' = z - (a / c)$, потому что $(x - x') = dx = 1$ (единичный шаг раstra)

Глубина пикселя, являющегося пересечением сканирующей строки с ребром многоугольника

Сначала определяют ребра грани, вершины которых лежат по разные стороны от сканирующей строки, так как только в этом случае сканирующая строка пересекает ребро. Затем из найденных точек пересечения выбирают ближайшую к наблюдателю.

Глубину определяют по соотношению:

$$z_3 = z_2 + (y_3 - y_2) / (y_2 - y_1) * (z_2 - z_1), \text{ где}$$

- $(y_1, z_1), (y_2, z_2)$ - координаты вершин проекции ребра на плоскость YOZ.
- (x_3, z_3) - координаты проекции точки пересечения на ту же плоскость.

Оценка эффективности

Плюсы:

- Сцены могут быть произвольной сложности.
- Не нужна сортировка, как в других алгоритмах.
- Трудоемкость линейно зависит от числа рассматриваемых поверхностей

Недостатки:

- Большой объем памяти (под буфера).
- Трудоемкость устранения лестничного эффекта.
- Трудность реализации эффектов прозрачности.

Псевдокод (алгоритм)

1. Инициализация буфера кадра фоновым значением.
2. Инициализация z-буфера минимальным значением Z.
3. Растворная развертка каждого многоугольника (в произвольном порядке).
4. Вычисление глубины $z = (x, y)$ для каждого пикселя, принадлежащего многоугольнику.
5. Сравнение полученной глубины z со значением z, лежащей в буфере (для пикселя (x, y)). Если полученная глубина больше значения в буфере, то записать атрибут многоугольника в буфер кадра и заменить значение в Z-буфере на полученное значение

Для невыпуклых многогранников, предварительно надо удалить нелицевые грани. Алгоритм так же можно применять для построения сечений поверхностей, в таком случае изменится только операция сравнения:

$$[Z(x, y) > Z_{\text{буф}}(x, y)] \text{ и } (Z(x, y) \leq Z_{\text{разр}}),$$

1. Что хранится для Z-буфера? Это буфер глубины, в нем хранятся координаты z для каждого пикселя.

2. И какого объема он должен быть? Сколько памяти нам потребуется выделять? Вот у вас экран 3 млн ячеек имеет. Какой тип данных будет использоваться? Вещественный.

А вещественный сколько байт 4 или 8? 4.

А почему 4 достаточно? Этого достаточно, чтобы сравнить два значения по Z

3. Значит глубину вы откуда будете вычислять, узнавать? Из уравнения плоскости.

Вот вычисляете, а С оказалось равным нулю. Такое может быть? Может. **И что делать тогда? Глубину не можем вычислить ведь Уравнение плоскости: Ax+By+Cz+D=0.** Для прямоугольника $Z = -(Ax+By+D)/C$, $C \neq 0$. Если $C=0$, то прямоугольник расположен параллельно вектору взгляда – отрисовывать нужно линию или вершину. Используется уравнение ребра; $(z-z1)/(z2-z1) = (y-y1)/(y2-y1)$, если координаты не равны. Если же равны, ребро горизонтально, отрисовывается точка. При $y2 \neq y1$: $z = z1 + (y-y1)/(y2-y1) * (z2-z1)$; При $y2 = y1$: $z = \max(z2, z1)$

4. Где хранится интенсивность? В буфере кадра

33. Алгоритм, использующий список приоритетов.

Алгоритмы, использующие список приоритетов, пытаются получить преимущество предварительной сортировкой по глубине или приоритету. На выходе (после сортировки) получаем окончательный список элементов сцены, упорядоченных по приоритету глубины, основанному на расстоянии от точки наблюдения. Если список окончателен, то никакие два элемента не будут перекрывать друг друга. Эффекты прозрачности можно включить в состав алгоритма путем частичной корректировки содержимого буфера кадра. Такие алгоритмы работают как в пространстве объекта, так и в пространстве изображения.

Основная проблема

- Для простых сцен, список приоритетов мы можем получить сразу.
- Однако, есть сцены, для которых невозможно получить результат простой сортировкой по z.
- Может случиться так, что объекты циклические прерывают друг друга. Для этого придумали метод, который динамически вычисляет новый список приоритетов перед обработкой каждого кадра сцены.

Называется алгоритмом Ньюэла - Ньюэла - Санча.

1. Сформировать предварительный список приоритетов по глубине, используя в качестве ключа сортировки z_{min} , для каждого многоугольника. Первый многоугольник в списке -

с минимальным z_{min} . Он будет лежать дальше всех от точки наблюдения. Обозначим его P, следующий в списке - Q.

2. Для каждого P из списка проверить отношение с Q.

2.1. Если ближайшая вершина P (P_zmax) дальше от точки наблюдения чем самая удаленная

$Q(Q_zmin) \geq P_zmax$, то никакая часть P не может экранировать Q.

Занести P

в буффер кадра.

2.2. Иначе (если $Q_zmin < P_zmax$), то P потенциальное экранирует не только Q, но и любой

другой многоугольника типа Q из списка для которого $Q_zmin < P_zmax$.

Образуется множество { Q }. Однако, P может фактически и не экранировать ни один из этих многоугольников. Если последнее верно, то P заносим в буфер кадра. Для ответа

на этот вопрос, используется серия тестов. Если ответ на любой вопрос положительный, то P не может экранировать { Q }, P заносится в буффер кадра.

Расположены эти **тесты** по возрастанию их вычислительной сложности: Верно ли, что прямоугольные объемлющие оболочки РиО не перекрываются по x?

Верно ли, что прямоугольные оболочки Р и О не перекрываются по y?

Верно ли, что Р целиком лежит по ту сторону плоскости, несущей Q, которая расположена дальше от точки наблюдения (рис. 4.53, а)?

Верно ли, что Q целиком лежит по ту сторону плоскости, несущей Р, которая ближе к точке наблюдения (рис. 4.53, б)?

Верно ли, что проекции P и Q не перекрываются?

Каждый из этих тестов применяется к каждому элементу {Q}. Если ни один из них не дает положительного ответа и не заносит P в буфер кадра, то P может закрывать Q. Поменять P и Q местами, пометив позицию Q в списке. Повторить тесты с новым списком. Это дает положительный результат для сцены с рис. 4.51, b.

При циклическом экранировании

Тут нужно разбиение. Для разбиения многоугольников вдоль линии пересечения несущих этих многоугольников можно воспользоваться алгоритмом Сазерланда - Ходжмена.

Плоскость, несущая Q, используется как секущая. Каждое ребро P отсекается Q (полностью), при этом формируются два новых многоугольника. Для поиска пересечения можно воспользоваться алгоритмом Кируса-Бека. Тот, что слева, имеет приоритет над Q. А справа - наоборот, экранируется Q.

Оценка эффективности

- Может нехватить 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).

Как ещё называют данный алгоритм?

Алгоритм художника. Сначала фон, потом от дальних объектов к ближнем.

Для самого дальнего надо проверить перекрывают ли он те, которые ближе.

Какая простейшая проверка? Больше ли z_{\max} этого многоугольника чем z_{\min} .

Если больше, то может. Тогда надо проверить. Проверяются пятью тестами.

Упорядочены ли эти тесты?

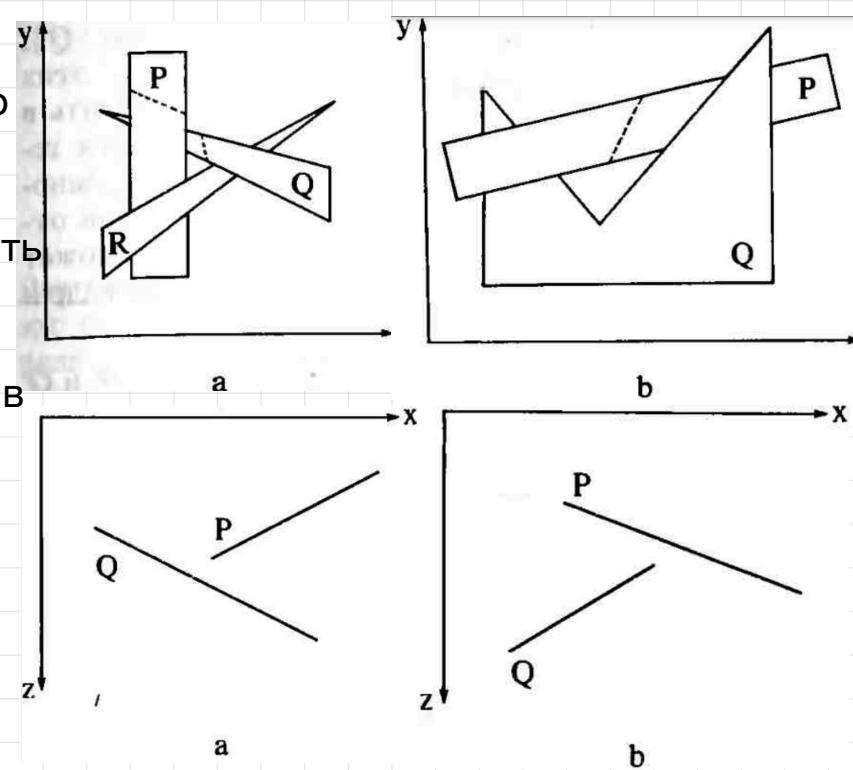
Да. Упорядочены от самого простого к самому сложному. (зависит от количества операций)

Как провести третий и четвертый тест? Как убедиться, что один многоугольник целиком лежит по невидимую сторону плоскости, несущей другой многоугольник?

У нас есть пробная функция (получается из уравнения плоскости), относительно которой проверяем плоскость.

После того, как провели плоскость через один многоугольник и нам надо определить, как расположен многоугольник относительно этой плоскости, для этого нужно подставить из плоскости значения (координаты каждой из вершин многоугольника), которую мы определяем и убедиться в том, что лежит по одну сторону и все знаки одинаковы.

Можно подставить координаты наблюдателя, и если знаки такие же как мы получили до этого, то, значит, многоугольник расположен с той же стороны, что и наблюдатель.



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

Алгоритмы Варнока, Z-буфера и строящего список приоритетов обрабатывают элементы сцены в порядке, который не связан с процессом визуализации. Алгоритмы построчного сканирования обрабатывают сцену в порядке прохождения сканирующей строки. Работают в пространстве изображения

Алгоритм:

1. Подготовка информации а) Для каждого многоугольника определить самую верхнюю сканирующую строку, которую он пересекает. б) Занести многоугольник в группу у, соответствующую этой сканирующей строке с) Запомнить для каждого многоугольника: Δu – число строк, пересекающих этот многоугольник, список рёбер многоугольника, коэффициенты A, B, C, D уравнения плоскости многоугольника, визуальные атрибуты многоугольника

2. Решение задачи удаления невидимых поверхностей а)

Инициализировать буфер кадра дисплея б) Для каждой сканирующей строки 1. Инициализировать буфер кадра размером с одну сканирующую строку, заполнив его фоновым изображением 2. Инициализировать Z-буфер размером с одну сканирующую строку значением Z_{min} 3. Проверить необходимость добавления в список активных многоугольников (САМ) новых многоугольников 4. Если было добавление многоугольника в САМ, то добавить в САР соответствующие рёбра новых многоугольников 5. Если произведено удаление какого-либо элемента из пары рёбер САР, то проверить необходимость удаления всего многоугольника из САМ. Если он остаётся, то проверить необходимость удаления другого ребра из этой пары – если его удалять не нужно, то доукомплектовать пару (добавив недостающее левое или правое ребро) с) **В САР должна храниться следующая информация:** 1. X_l – пересечение левого ребра с текущей сканирующей строкой 2. ΔX_l – приращение X_l в интервале между соседними сканирующими строками 3. ΔY_l – число сканирующих строк, пересекаемых левым ребром 4. X_p , ΔX_p , ΔY_p 5. $\Delta Z_x = -A/C$ для $C \neq 0$ (иначе $\Delta Z_x = 0$) – приращение по Z вдоль сканирующей строки 6. $\Delta Z_y = -B/C$ для $C \neq 0$ (иначе $\Delta Z_y = 0$) – приращение по Z в интервале между соседними сканирующими строками

d) Для каждой пары ребёр многоугольника из САР выполнить: 1. Извлечь 2. Инициализировать Z значением Z_l 3. Для каждого пикселя $X_l \leq X \leq X_p$ вычислить $Z(x, y = \text{const})$. $Z_1 = Z_l, \dots, Z_k = Z_{k-1} + \Delta Z_x$ 4. Если $Z() > Z_{\text{буф}}()$, то $Z_{\text{буф}} = Z$ и занести атрибуты многоугольника в буфер кадра **e) Записать буфер кадра сканирующей строки в буфер кадра дисплея f)**

Скорректировать САР 1. $\Delta Y_l --, \Delta Y_p --$ 2. $X_l = X_l + \Delta X_l, X_p = X_p + \Delta X_p$ 3. $Z_l = Z_l + \Delta Z_x \Delta X + \Delta Z_y$ 4. Если ΔY_l , то удалить соответствующее ребро из списка, пометив положение обоих ребер в списке и породивший многоугольник **g) Скорректировать САМ** 1. $\Delta Y --$ 2. Если ΔY , 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, полученном путём деления сканирующей строки проекциями точек пересечения ребёр

1. Изобразить фон 2. Изобразить атрибуты многоугольника, соответствующие этому отрезку 3. Изобразить атрибуты многоугольника, соответствующие отрезку с $\text{MAX } Z$ 4. $\text{sign}(z_{1m} - z_{2m}) \neq \text{sign}(z_{1k} - z_{2k})$ – разбить интервал точкой пересечения

Решение задачи удаления невидимых поверхностей сводится к выводу видимых отрезков в каждом из интервалов, полученных путем деления скан строки проекциями точек пересечения ребер

1 Изобразить фон 2 изобразить атрибуты мн-ка, соотв этому интервалу 3 изобразить атрибуты мн-ка, соотв отрезку с $\text{max } z$ 4 $\text{sign}(z_{1n} - z_{2n}) * \text{sign}(z_{1k} - z_{2k})$

В связи с чем появились алгоритм построчного сканирования? Использование z-буфера занимает много памяти. Мы его сводим к хранению одной строки. А сколько памяти z-буфер? Много по каким меркам? Зависит от кол-во пикселей в окне ($1920 * 1080 \sim 2\text{-3 лимона}$). Храним (глубину) вещественные значения в буфере (с точностью float (высокая не нужна потому что нам нужно не само значение, а сравнивать значения)). $2\text{-3 лимона} * 4\text{-8 байт} = 8\text{-24 лимонов байт} = .8\text{-12 Мбайт}$. Не очень много по современным меркам. Но вот в 1994 году.... 170 Мб - очень много. Определяете $\text{delta}_z(x, y)$. Делите на с. А если с = 0? Что означает с = 0? Как для неё определить глубину? Уравнение плоскости: $Ax + By + Cz + D = 0$. Для прямоугольника $Z = -(Ax + By + D)/C$, $C \neq 0$. Если $C = 0$, то прямоугольник расположен параллельно вектору взгляда – отрисовывать нужно линию или вершину. Используется уравнение ребра; $(z - z_1)/(z_2 - z_1) = (y - y_1)/(y_2 - y_1)$, если координаты не равны. Если же равны, ребро горизонтально, отрисовывается точка. При $y_2 \neq y_1$ $z = z_1 + (y - y_1)/(y_2 - y_1) * (z_2 - z_1)$ При $y_2 = y_1$ $z = \text{max}(z_2, z_1)$

35. Алгоритм определения видимых поверхностей путем трассировки лучей.

Трассировка лучей - метод грубой силы (метод, не учитывающий специфику обрабатываемого объекта). Главная идея: наблюдатель видит объект непосредством испускаемого неким источником света, который падает на этот объект и затем как то видит этот объект. Из огромного числа лучей, выпущенных источником, лишь небольшая часть дойдет до наблюдателя, поэтому отслеживать лучи таким образом неэффективно. Аппель предложил трассировать (то есть отслеживать) лучи в обратном направлении, т.е. от наблюдателя к объекту

— Предполагается, что сцена преобразована в пространство изображений. — Считается, что точка зрения или наблюдатель находится бесконечности на положительной полуоси z, поэтому все лучи параллельны оси z. — Каждый луч проходит через центр пикселя на растре до сцены. — Траектория каждого луча отслеживается, чтобы определить, какие именно объекты пересекаются сданным лучом. Необходимо проверить пересечение каждого объекта с каждым лучом. — Пересечение с z_{max} представляет видимую поверхность для данного пикселя

Случай, если точка расположена не в бесконечности (перспективная проекция)

— Предполагается, что наблюдатель так же находится на OZ. — Растр перпендикулярен OZ. — Задача состоит в том, чтобы построить одноточечную центральную проекцию на картинную плоскость.

Определение пересечений

Понятно, что нужно вычислять множество точек пересечений (около 75 - 95% всего времени). Поэтому, для ускорения процесса, необходимо иметь важные критерии, чтобы исключить из процесса заведомо лишние объекты. Одно из решений - погружение объектов в выпуклую "оболочку". Например сферическую, или в форме параллелепипеда. Таким образом, можем проверить пересечения луча с объектом (оболочкой). Если луч не пересекает оболочки, то объект можно откинуть, и не искать пересечения луча и этого объекта.

Сферический тест.

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

Пусть прямая задана как $P_1(x_1, y_1, z_1)$ и $P_2(x_2, y_2, z_2)$.

$$\begin{aligned} P(t) &= P_1 + (P_2 - P_1)t \quad \text{с компонентами:} \\ x &= x_1 + (x_2 - x_1)t = x_1 + at \\ y &= \dots = y_1 + bt; \quad z = z_1 + (z_2 - z_1)t = z_1 + ct. \end{aligned}$$

Тогда минимальное расстояние d от этой прямой до точки $P_0(x_0, y_0, z_0)$ равно:

$$d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2$$

где параметр t , определяющий ближайшую точку $P(t)$ равен:

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

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

Габаритный тест

Тест при прямоугольной оболочке

Удобно провести преобразование, которое совместит трассирующий луч с OZ. Тогда достаточно будет исследовать x_{min} , x_{max} , y_{min} и y_{max} . Если они меняются в каждой паре, то зафиксируем возможное пересечение.

- Выполнение габаритного теста с прямоугольной оболочкой в трехмерном пространстве требует большого объема вычислений.
- Следует проверить пересечение по меньшей мере с тремя боксонаностями плоскостями (я не понял о чем речь)
- Более медленный, чем со сферической оболочкой.

Алгоритм

1. Создание списка объектов.

- Полное описание объекта (тип, пов-ть, хар-ки)
- Описание сферической оболочки.
- Флаг прямоугольной оболочки (если нужны габаритные тесты)

2. Для каждого луча выполнить:

2.1. Для каждого объекта выполнить тест со сферич. оболочкой.

Если объект проходит тест, то заносим его в список активных объектов. 2.2. Если список пуст, изобразить пиксель цветом фона. Иначе, перенос и поворот луча для совмещения с OZ (запомнить преобразование) 3. Для каждого активного объекта:

3.1. Если поднят флаг прямоугольной оболочки, преобразовать эту оболочку в СК луча и выполнить соответствующий тест.

3.2. Преобразовать объект в СК луча, определить его пересечение с лучом.

Занести все пересечения в список пересечений.

4. Если список пуст, то ставим пиксель цветом фона. Иначе, определить z_{max} для списка пересечений.

5. Вычислить обратное преобразование. Определяем с его помощью т.

пересечения в исходной СК. Нарисовать пиксель с атрибутами объекта и модели освещения.

// Пятый пункт для определения видимости не нужен.

Возможные модификации

1. Кластерные группы пространственно связанных объектов

Вводим сферические оболочки для групп (наборов) связанных между собой объектов. (например: корзина внутри которой находятся котики). Такие сферические оболочки называются сферическим кластером - такой кластер охватывает два и более объектов. После всех манипуляций, просто обрабатываем все в иерархическом порядке. Если луч не пересекает сферический кластер, то выкидываем весь кластер. Если пересекает, то рекурсивно выполняется, пока не будут обработаны все объекты. 2. Упорядочение по приоритету. Используется для сокращения числа объектов, для которых вычисляются пересечения. После рассмотрения всех объектов, преобразованный список пересеченных объектов упорядочивается по приоритету глубины. Для определения приоритетного порядка можно использовать центры сферических оболочек или z_{max} (или z_{min}) прямоугольных оболочек. Далее, определяют список активных объектов (по идеям алгоритма со списком приоритетов), далее так же трассировка как в обычном алгоритме, только уже для активных объектов.

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

Что нужно учитывать, при построении реалистических изображений:

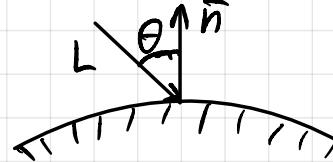
1. Оптические свойства поверхностей.
2. Воспроизводить рисунок на поверхности.
3. Воспроизводить неровности.
4. Поверхности отбрасывают тени (учитывать).
5. Восприятие окружающего мира человеческим глазом.

Физические и психологические факторы

Из опытов известно, что чувствительность глаза к яркости света изменяется по логарифмическому закону. Пределы чувствительности к яркости чрезвычайно широки, порядка 10^{10} , однако глаз не в состоянии одновременно воспринять весь этот диапазон. Глаз реагирует на гораздо меньший диапазон значений относительно яркости, распределённый вокруг уровня адаптации к освещённости. Чувствительность к относительной яркости имеет порядок 100-150. Скорость адаптации к яркости неодинакова для различных частей сетчатки, но тем не менее очень высока. Экстремумы диапазона относительной яркости воспринимаются соответственно как чёрный и белый.

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

Ещё одним свойством глаза, имеющим значение для машинной графики, является то, что границы областей постоянной интенсивности кажутся более яркими, в результате чего области с постоянной интенсивностью воспринимаются, как имеющие переменную интенсивность. Это явление называется эффектом полос Маха. Эффект полос Маха наблюдается, когда резко изменяется наклон кривой интенсивности. Если кривая интенсивности вогнута, то в этом месте поверхность кажется светлее, если выпукла - темнее.



Простая модель освещения

Отражаемый свет может быть диффузным и зеркальным. Диффузный свет - как бы проникает под поверхность, а потом расходится по всем направлениям.

$$I = I_u \cdot k_p \cdot \cos\theta, \quad 0 \leq \theta \leq \frac{\pi}{2}$$

На объекты реальных сцен падает рассеянный свет:

$$I = I_p \cdot k_p + I_u \cdot k_p \cdot \cos \theta$$

Опытным путём можно достичь большей реалистичности при линейной зависимости затухания интенсивности:

$$I = I_p k_p + \frac{I_d k_d \cos \theta}{d+K}$$

K - произвольная постоянная, выбираемая на основе эксперимента.
 d - расстояние от центра проекции до объекта

Зеркальное отражение света

Зависит от угла падения, длины волны и свойств вещества

Формула для простой модели освещения:

$$I_3 = I_u w(\theta, \lambda) \cos \alpha$$



где w - функция отражения, представляющая отношение зеркально отраженного света к падающему и зависящая от угла падения тета и длины волны лямбда.

n - степень, аппроксимирующая пространственное распределение зеркально отраженного света.

Формула (функция закраски) применяется для расчета интенсивности или тона точек объекта или пикселов изображения:

$$I = I_p k_p + \frac{I_u}{d+K} \cdot (k_g \cos \theta + k_3 \cos^n \alpha)$$

Если имеется несколько источников света, то интенсивности от отдельных источников суммируются, m - количество источников

$$I = I_p k_p + \sum_{j=1}^m \frac{I_{uj}}{d+K} \cdot (k_g \cos \theta_j + k_3 \cos^n \alpha_j).$$

$$\text{P.S. } \cos \theta = \frac{n}{|n|} \cdot \frac{L}{|L|} = \bar{n} \cdot \bar{L}, \quad \cos \alpha = \frac{R}{|R|} \cdot \frac{S}{|S|} = \bar{R} \cdot \bar{S}.$$

37. Построение реалистических изображений. Метод Гуро закраски поверхностей (получение сглаженного изображения).

Бывает три возможных типа закрашивания:

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

Простая закраска

В данном алгоритме закрашиваем все одним уровнем интенсивности (одним цветом). Алгоритм быстрый, но качество не очень.

Простая закраска используется при выполнении трех условий.

1. Предполагается, что источник находится в бесконечности.

Падающие лучи параллельны друг другу. Это влияет на расчёт диффузной составляющей, т.к. она зависит от угла падения. Для всех точек угол падения одинаковый -> диффузная составляющая одна и та же.

2. Предполагается, что наблюдатель находится в бесконечности

Вектор наблюдения взгляда и вектора зеркального отражения будут для всех точек одинаковыми, значит везде будет одинаковая зеркальная составляющая (интенсивность)

3. Закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности.

ЭТО САМОЕ ВАЖНОЕ ТРЕБОВАНИЕ! Например, можно рассмотреть сферу. Аппроксимировали ее гранями, каждую грань закрасили своим уровнем интенсивности (грани будут разного цвета). Глаз будет видеть резкий переход, который будет выглядеть как ребро. Какие ребра могут быть у сферической поверхности??? Сильное искажение изображения.

- Плюсы: быстрый и простой
- Минусы: возникают ребра

Закраска Гуро

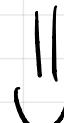
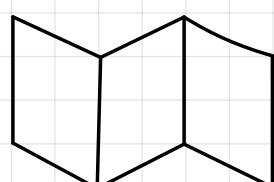
Сглаживание на основе интерполяции интенсивности.

Гуро предложил биполярную (биполяр очка) интерполяцию интенсивности. Границы рассматриваются отдельно. Вычисляем нормали вершины грани. Зная нормаль, вычисляем интенсивность каждой вершины и выполняем первую линейную интерполяцию вдоль ребер. (Линейную чтобы минимизировать вычисления).

Вторая линейная интерполяция выполняется когда вычисляем интенсивности пикселей, расположенных на сканирующей строке. Качество изображения улучшится.

Закраска не предусматривает учет кривизны поверхности. При применении закраски можно потерять ребра и получить плоское изображение. Это может произойти, когда закрашивается смежная грань одним уровнем интенсивности, то есть когда углы граней одинаковые.

Пример - детская книжка-раскладушка. Нормали одинаковые, интенсивности тоже, каждая из 3х граней будет закрашена одинаковым уровнем. Ребер не увидим.

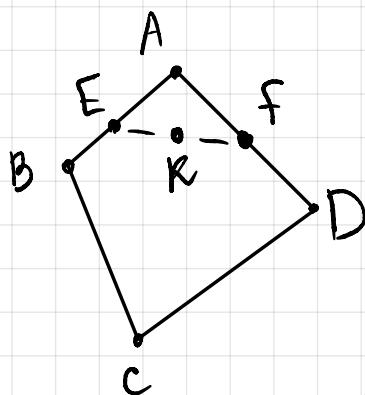


он спросит...

Решение: наарандомить изменения в нормалях либо разбить на более мелкие грани (если речь идет об аппроксимации)

Такая закраска хорошо стягивается с диффузной составляющей поверхности (матовой).

Расчеты:



$$I_E = (1-t)I_A + tI_B, \quad t = \frac{AE}{AB}$$

$$I_F = (1-v)I_A + vI_C, \quad v = \frac{AF}{AC}$$

$$I_K = (1-w)I_E + wI_F, \quad w = \frac{EK}{EF}$$

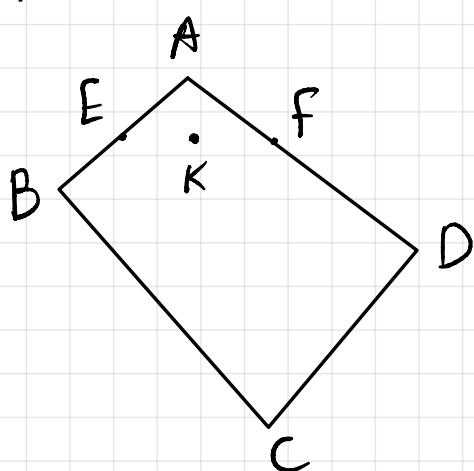
Можно вычислить интенсивность вдоль сканирующей строки инкрементально, то это самое.

38. Построение реалистических изображений. Закраска Фонга (улучшение аппроксимации кривизны поверхности).

Требует больших вычислительных затрат, однако решает многие проблемы методы Гуро. При закраске Гуро вдоль сканирующей строки интерполируется значение интенсивности, а при закраске Фонга - вектор нормали. Далее он используется в модели освещения для вычисления интенсивности пикселя. При этом достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение. В частности лучше выглядят зеркальные блики.

При закраске Фонга аппроксимация кривизны поверхности производится сначала в вершинах многоугольников путём аппроксимации нормали в вершине. После этого билинейной интерполяцией вычисляется нормаль в каждом пикселе. Например, обращаясь к след. рисунку получаем нормаль в K линейной интерполяцией между A и B, в R - между B и C и, наконец, в P - между Q и R.

Алгоритм более чем в 3 раза уступает в производительности методу Гуро, т. к. для каждой точки вычисляем интенсивность из нормали, +3 операции.



$$\bar{n}_E = (1-t)\bar{n}_A + t\bar{n}_B, \quad t = \frac{AE}{AB}$$

$$\bar{n}_F = (1-v)\bar{n}_A + v\bar{n}_C, \quad v = \frac{AF}{AC}$$

$$\bar{n}_K = (1-w)\bar{n}_E + w\bar{n}_F, \quad w = \frac{EK}{EF}$$

38

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

Определение нормали зависит от способа задания поверхности.

1. Если поверхность задана аналитически $F(x, y, z) = 0$, то нормаль вычисляется через градиент в точке $\text{grad}(F(x, y, z))$.

2. Для полигональных моделей нормаль совпадает с нормалью соответствующего полигона и вычисляется через векторное произведение двух ребер. (или задаётся заранее).

$N = N_1 + N_2 + N_3$, где N_1, N_2, N_3 - нормали граней инцидентных вершине в которой ищется нормаль.

$N = v_1^*v_2 + v_2^*v_3 + v_3^*v_1$, где v_1-v_3 - векторы ребер инцидентных вершине в которой ищется нормаль.

Нормаль в точке схода плоскостей полигональной модели равна среднему значению нормалей всех сходящихся в этой точке плоскостей. Если плоскости заданы не аналитически, то нормаль ищется как усредненное векторное произведение всех ребер, сходящихся в вершине. Чем больше площадь грани, тем больше будет вклад векторного произведения в результат.

Вычислять нормаль необходимо до перспективного деления, иначе направление нормали будет искажено, что приведет к неправильному определению интенсивности освещения.

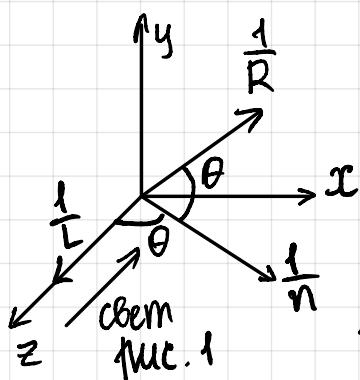
Определение вектора отражения

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.

Существует диффузное и зеркальное отражение. При диффузном отражении свет поглощается поверхностью на которую падает и вновь излучается этой поверхностью по всем направлениям и цвет света совпадает с цветом поверхности. Блеклая, матовая. При зеркальном отражении отраженный свет концентрируется вокруг вектора отражения, то есть это направленное отражение. Цвет зеркально отраженного света совпадает с цветом падающего света.

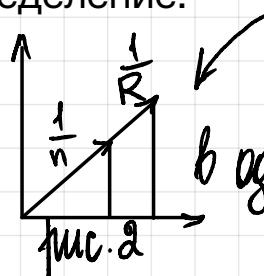
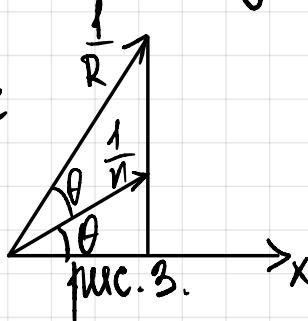
Учесть факт того, что происходит отражение не строго по вектору отражения а вблизи его можно возведением косинуса в степень. Чем выше степень, тем более концентрированное распределение.

1. свет $\parallel O_z$



R - отраженный угл

L - исходный угл



Из подобия на фиг. 2:

$$\frac{\frac{1}{R_x}}{\frac{1}{L_x}} = \frac{\frac{1}{n_x}}{\frac{1}{n_y}}$$

$$\hat{\frac{1}{n_z}} = \cos\theta, \hat{\frac{1}{R_z}} = \cos 2\theta = 2\cos^2\theta - 1 = 2\hat{\frac{1}{n_z^2}} - 1$$

$$\hat{\frac{1}{R_x^2}} + \hat{\frac{1}{R_y^2}} = 1 - \hat{\frac{1}{R_z^2}} = 1 - (2\hat{\frac{1}{n_z^2}} - 1)^2 = 4\hat{\frac{1}{n_z^2}}(1 - \hat{\frac{1}{n_z^2}}) = 4\hat{\frac{1}{n_z^2}}(\hat{\frac{1}{n_x^2}} + \hat{\frac{1}{n_y^2}}).$$

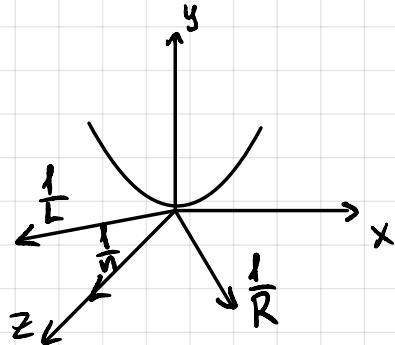
$$\hat{\frac{1}{R_y}} = \frac{\hat{\frac{1}{n_x}}}{\hat{\frac{1}{n_y}}} \Rightarrow \hat{\frac{1}{R_y^2}} + \frac{\hat{n_x^2}}{\hat{n_y^2}} \cdot \hat{\frac{1}{R_y^2}} = 4\hat{\frac{1}{n_z^2}}(\hat{\frac{1}{n_x^2}} + \hat{\frac{1}{n_y^2}})$$

$$\hat{\frac{1}{n_y^2}} \cdot \hat{\frac{1}{R_y^2}} + \hat{\frac{1}{n_x^2}} \cdot \hat{\frac{1}{R_y^2}} = 4\hat{\frac{1}{n_z^2}}(\hat{\frac{1}{n_x^2}} + \hat{\frac{1}{n_y^2}})\hat{\frac{1}{n_y^2}}; \hat{\frac{1}{R_y^2}} = 4 \cdot \hat{\frac{1}{n_z^2}} \cdot \hat{\frac{1}{n_y^2}},$$

$$\hat{\frac{1}{R_y}} = 2\hat{\frac{1}{n_z}} \cdot \hat{\frac{1}{n_y}}$$

$$\hat{\frac{1}{R_x}} = 2\hat{\frac{1}{n_z}} \cdot \hat{\frac{1}{n_x}}$$

2. много источников

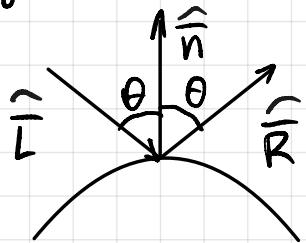


$$\hat{L}_z = -\hat{R}_z; \hat{L}_y = -\hat{R}_y; \hat{L}_x = \hat{R}_x$$

= -домик курова :)

Нормаль поворачивается относительно оси Z, а точка P принимается за начало координат. Тогда плоскость xy, будет касательной к поверхности, а x, y составляющие векторов падения и отражения будут иметь разные знаки, z будет одинаковой. Далее выполняется обратное преобразование и получаются координаты вектора отражения в исходной СК.

3. Через векторное произведение



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

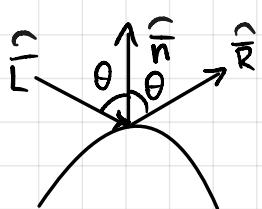
$$\hat{L} \otimes \hat{n} = \begin{vmatrix} i_x & i_y & i_z \\ \bar{L}_x & \bar{L}_y & \bar{L}_z \\ \bar{n}_x & \bar{n}_y & \bar{n}_z \end{vmatrix} = \begin{vmatrix} i_x & i_y & i_z \\ \bar{R}_x & \bar{R}_y & \bar{R}_z \\ \bar{n}_x & \bar{n}_y & \bar{n}_z \end{vmatrix} = \hat{n} \otimes \hat{R}$$

$$\hat{L}_x \hat{\frac{1}{n_x}} + \hat{L}_y \hat{\frac{1}{n_y}} + \hat{L}_z \hat{\frac{1}{n_z}} = \hat{\frac{1}{n_x}} \hat{\frac{1}{R_x}} + \hat{\frac{1}{n_y}} \hat{\frac{1}{R_y}} + \hat{\frac{1}{n_z}} \hat{\frac{1}{R_z}} \quad [(\hat{L}, \hat{n}) = (\hat{n}, \hat{R})]$$

Mam. bug: $N \cdot R = \begin{bmatrix} n_z L_y - n_y L_z \\ n_x L_z - n_z L_x \\ n_y L_x - n_x L_y \\ n_x L_x + n_y L_y + n_z L_z \end{bmatrix}$

бешар, кайфуем

4. Аналитически



$$\hat{\frac{1}{R}} = \alpha \hat{\frac{1}{L}} + \beta \hat{\frac{1}{n}}; \cos\theta = (-\hat{\frac{1}{L}}, \hat{\frac{1}{n}}) = (\hat{\frac{1}{n}}, \hat{\frac{1}{R}})$$

$$(\alpha \hat{\frac{1}{L}} + \beta \hat{\frac{1}{n}}) \hat{\frac{1}{n}} = \alpha (\hat{\frac{1}{L}}, \hat{\frac{1}{n}}) + \beta (\hat{\frac{1}{n}}, \hat{\frac{1}{n}}) = \cos\theta, \beta = \cos\theta(\beta + \alpha)$$

$$(\alpha \hat{\frac{1}{L}} + \beta \hat{\frac{1}{n}})^2 = \dots = \alpha^2 + 2\alpha\beta\cos\theta + \beta^2 = 1$$

Короче такие $\alpha = 1, \beta = 2\cos\theta$.

40. Построение теней при создании реалистических изображений.

Учет теней в алгоритмах удаления невидимых поверхностей.

Если положение источника света и наблюдателя совпадают, то тени учитывать не надо! Положение теней не зависит от наблюдателя, поэтому тени достаточно расчитать один раз. Тень состоит из двух частей: полутиени и полной тени. В КГ обычно используют точечные источники освещения, которые создают только полную тень. Сложность вычислений обычно зависит от положения источников света. Источник света находится в бесконечности. Используется ортогональное проецирование. Источник расположен на конечном расстоянии, но вне поля зрения наблюдателя. Используется перспективная проекция. Источник света в поле зрения наблюдателя. Пространство делят на участки и определяют тень для каждого участка. При построении теней нужно два раза удалить невидимые поверхности, для каждого источника и для наблюдателя. Различают собственные тени и проекционные. Для нахождения таких теней нужно построить проекцию всех нелицевых граней на сцену.

- Центр проекции хранится находится в источнике света.
- Теневые многоугольники заносятся и хранятся в виде контуров (для оптимизации размера)
- Для разных точек наблюдения нет смысла пересчитывать каждый раз, т.к. они изменяются только при изменении источников света.

Учет теней в алгоритмах удаления невидимых поверхностей.

Учет теней и фактуры в наиболее полном виде возможен при построении сцены с использованием алгоритма трассировки лучей в сочетании с глобальной моделью освещения. Построение теней в сочетании с удалением невидимых поверхностей требует дополнительных усилий и использования дополнительных СД.

Модификация z-буфера:

1 Строится сцена положения наблюдателя, совмещенного с положением источника света. Вычисленные значения глубины(зет координата) хранятся в дополнительном теневом зет-буфере, при этом значения интенсивности не учитываются. 2 Строится сцена для истинного положения наблюдателя. При обработке каждого мн-ка его глубина в каждом пикселе сравнивается с глубиной в зет-буфере наблюдателя. Если поверхность видима, то значения координат (x,y,z) из вида наблюдателя преобразуются в значения (xp,yp,zn) на виде из источника.

Модификация Вейлера-Азертона:

1. 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. Т. к. это удвоило бы количество обрабатываемых граней для выпуклого многоугольника из-за того, что нелицевые грани обычно отбрасываются до применения алгоритма удаления невидимых поверхностей.

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. Иначе область вне этого объема окажется затененной и наблюдатель увидит ложные тени. 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.

2. 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.

Тень зависит от положения наблюдателя?

Не зависит. Зависит только от источника света.

Если в сцене только источник света. Будут ли какие-нибудь тени?

Смотря где наблюдатель. Если наблюдатель в одном месте с источником света теней не будет. (ну и расположение объектов)

Какие тени мы разделяем? Собственные (тень самого объекта) и проекционные (тень от другого объекта). Как найти каждую из теней?

Чтобы найти собственную тень нужно воспользоваться первым этапом алгоритма Робертса. А именно определить нелецевые грани, совместить точку источника света с точкой наблюдения.

Чтобы найти проекционную тень нужно найти проекцию невидимых граней на объекты.

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

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

Каким условиям должна удовлетворять грань, чтобы давать видимую тень?

1. Должна быть в тени.
2. Должна быть лицевой для наблюдателя

2 раза применяем Робертса:
сначала для источника света,
потом для наблюдателя.

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

В основных моделях освещения и алгоритмах удаления невидимых линий и поверхностей рассматриваются только непрозрачные поверхности и объекты. Однако существуют и прозрачные объекты, пропускающие свет, например, такие, как стакан, ваза, окно автомобиля, вода. При переходе из одной среды в другую, например из воздуха в воду, световой луч преломляется; поэтому торчащая из воды палка кажется согнутой. Преломление рассчитывается по **закону Снеллиуса**, который утверждает, что падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления связаны формулой

$$\eta_1 \sin \theta_1 = \eta_2 \sin \theta_2$$

где η - показатели двух сред; θ_1 - угол падения;
 θ_2 - угол преломления.

Существует два варианта аппроксимации - **линейная и нелинейная**
Линейная

Прозрачные многоугольники или поверхности помечаются, и если видимая грань прозрачна, то в буфер кадра записывается линейная комбинация двух ближайших поверхностей. При этом интенсивность

$$I = t I_1 + (1-t) I_2; 0 \leq t \leq 1$$

где I_1 - видимая поверхность, I_2 поверхность, расположенная непосредственно за ней, t коэффициент прозрачности I_1 . Если поверхность совершенно прозрачна, то $t = 0$, а если непрозрачна, то $t = 1$. Если I_2 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 Ньюэла - Ньюэла - Санча, тогда I_2 будет соответствовать значению,енному в буфер кадра, а I_1 - 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**:

$$t = t_{\min} + (t_{\max} - t_{\min}) \left[1 - (1 - |n_z|^P) \right]$$

где t_{\min} и t_{\max} - минимальная и максимальная прозрачность объекта, n_z есть z - составляющая единичной нормали к поверхности, P - коэффициент степени прозрачности, t - прозрачность пикселя или точки объекта.

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.

Общий вид формулы расчёта интенсивности:

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

, где индексы a , d , s , t обозначают рассеянный, диффузный, зеркальный и пропущенный свет. В большинстве моделей предполагается, что к постоянная и 1 интенсивность преломленного света определяется по закону Снеллиуса.

42. Учет фактуры при создании реалистических изображений.

Под фактурой, или текстурой, в компьютерной графике понимается детализация строения поверхности. Как правило, рассматривают два вида детализации. В первом случае на гладкую поверхность объекта наносят заданный рисунок (узор). В этом случае поверхность остается гладкой. Во втором случае ставится задача создания неровностей на поверхности, т.е. создания шероховатой поверхности.

1. Нанесение рисунка на поверхность (текстурирование)

1.1. Проективные текстуры

Подразумевает наличия проективных текстур фиксированных размеров (рисунков), которые хранятся в некоторой библиотеке текстур в памяти ЭВМ. Очевидная проблема - большой расход памяти на хранение данных текстур.

Можем искать отображение рисунка (текстуры) на 3D-поверхность путем проецирования (параллельным переносом/цилиндрическим/сферическим методами). В случае, если размер поверхности больше чем размер рисунка, узор рисунка обычно повторяется (иногда нормальные люди называют это паттерном но дед таких слов не знает).

1.2. Процедурные текстуры

Текстуры, описываемые математическими формулами. Такие текстуры не занимают место в видеопамяти, и вычисляются на лету (пиксельными шейдерами но возможно за это дадут пизды). Ярким примером от Курова являются годовые кольца на срубе дерева - концентрические окружности. Преимуществом процедурных текстур является неограниченный уровень детализированности (отсутствие пикселизации).

2. Неровности

Для того, чтобы поверхность казалась шероховатой, можно оцифровать фотографию нерегулярной фактуры и отобразить ее на поверхность. Однако при этом неровности будут восприниматься как нарисованные на гладкой поверхности, т.е. изображение будет не полностью реалистичным. Это связано с тем, что в векторе нормали к реальной шероховатой поверхности, а следовательно, в направлении вектора отражения, есть небольшая случайная составляющая. Этот факт и лег в основу способа моделирования неровностей на отображаемой поверхности. Пусть $Q(X, Y)$ – уравнение поверхности, т.е. функция Q позволяет для каждой точки поверхности определить ее третью координату Z .

В произвольной точке поверхности частные производные по направлениям X, Y лежат в плоскости, касательной к поверхности в этой точке. Нормаль в точке поверхности может определяться векторным произведением этих производных:

$$N = [Q'_x, Q'_y]$$

Для создания шероховатой поверхности можно создать новую поверхность путем внесения возмущения в направлении нормали в точках поверхности. Пусть $P(X, Y)$ – функция возмущения, тогда радиус-вектор точки на новой поверхности будет определяться из

$$Q_n(x, y) = Q(x, y) + \frac{P(x, y)N}{|N|}$$

Нормаль к новой возмущенной поверхности будет иметь вид:

$$N_n = [Q'_{nx}, Q'_{ny}]$$

Частные производные Q'_{nx} , Q'_{ny} вычисляются из следующих выражений:

$$Q'_{nx} = Q'_x + \frac{P'_x(x, y)N}{|N|} + \frac{P(x, y)N_x}{|N|}$$

$$Q'_{ny} = Q'_y + \frac{P'_y(x, y)N}{|N|} + \frac{P(x, y)N_y}{|N|}$$

Последними слагаемыми можно пренебречь, так как функция возмущения $P(X, Y)$ – очень маленькая величина. Теперь можно записать выражение для вычисления нормали к возмущенной поверхности:

$$\begin{aligned} N_n = & [Q'_{nx} + Q'_{ny}] + \\ & + P'_y(x, y)[Q'_x, \frac{N}{|N|}] + P'_x(x, y)[\frac{N}{|N|}, Q'_y] + \\ & + P'_x(x, y) \otimes P'_y(x, y)[N, (\frac{N}{|N|})^2] \end{aligned}$$

Учитывая, что первое слагаемое представляет собой нормаль к исходной поверхности, а последнее слагаемое равно нулю как векторное произведение коллинеарных векторов, окончательно получим:

$$N_n = N + P'_y(x, y)[Q'_x, \frac{N}{|N|}] + P'_x(x, y)[\frac{N}{|N|}, Q'_y].$$

43. Глобальная модель освещения с трассировкой лучей

Модель предназначена для изучения свойств объекта, выполнение каких-то вычислений и т.д. Есть простая (локальная) модель освещения, которая включает в себя:

- Диффузную составляющую отражения;
- Зеркальную составляющую;
- Рассеянное освещение;

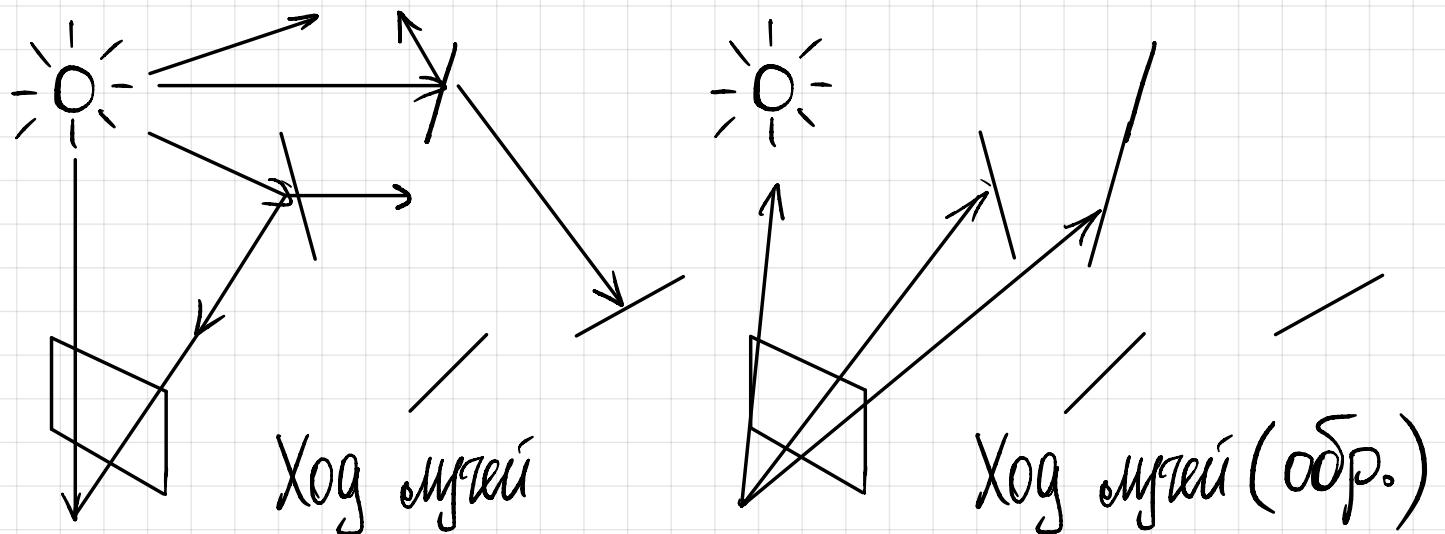
В простой модели освещения мы учитываем цвет, попадающий в рассматриваемую точку от источника света.

Глобальная модель будет содержать в себе те же составляющие, но учитывается еще интенсивность, зеркально отраженный свет от других поверхностей (зеркальное отражение - направленное отражение и его просто учитывать, диффузное трудно, поскольку распространяется по всем направлениям) и пропускание света прозрачными поверхностями.

$$I = I_p K_p + I_i K_d + I_i K_z + I_z K_z + I_{pr} K_{pr}$$

- I_p - интенсивность рассеянного освещения: с помощью коэффициента мы учитываем какие либо потери, значения лежат от 0 до 1;
- I_i - интенсивность источника;
- K_d - коэффициент диффузного отражения;
- K_z - коэффициента зеркального отражения;
- $I_z K_z$ - добавляем отраженность от других поверхностей, учитывая долю;
- I_{pr} - интенсивность преломленного луча;
- K_{pr} - коэффициент пропускания.

Модель является расширением локальной модели распространения света. Основана на том факте, что из множества лучей, испускаемых источником, на формирование изображения влияют лишь те лучи, которые попадают в объектив аппарата или на сетчатку глаза человека. Свет достигает наблюдателя, если он отражается от поверхности, преломляется или проходит через нее. Из каждой точки источника света исходит бесконечное количество лучей, причём большинство их никогда не достигает точки зрения. Большая часть лучей не попадает на сетчатку или в камеру, поэтому не влияет на формирование изображения. Если изменить ход лучей на противоположный, то в создании изображения примут участие только те лучи, которые "испускает" центр проецирования с учётом рамки отсечения в картинной плоскости. Эти лучи называются приведёнными.



44. Алгоритм трассировки лучей с использованием глобальной модели освещения.

Модель освещения предназначена для расчёта интенсивности отражённого к наблюдателю света в каждой точке изображения. Модель может быть глобальной или локальной.

- Локальная модель - учитывается только свет от источников и ориентация поверхности;
- Глобальная модель - учитывается ещё и свет, отражённый от других поверхностей или пропущенный через них.

Глобальная модель освещения является частью алгоритмов выделения видимых поверхностей путём трассировки лучей.

Дерево лучей

В глобальной модели освещения предполагается, что падающий луч v в точке Q отражается в направлении r и проходит сквозь поверхность в направлении p . Таким образом, в точке образуется ещё два луча, для которых нужно найти все пересечения с объектами. Процесс повторяется до тех пор, пока не останется ни одного пересечения.

Пусть v - падающий луч, r - отражённый, а p - преломлённый.

Пропущенный луч высчитывается по закону Снеллиуса:

$$r = v' + \alpha n, \quad p = k_f(n+v') - n$$
$$\psi' = \frac{v}{|v \cdot n|}, \quad k_f = (k_n^2 \cdot |v'|^2 - |v'+n|^2)^{-0.5}$$

Здесь k_n - отношение показателей преломления, n - вектор единичной нормали в направлении падающего луча. По итогу, для расчёта результирующей интенсивности луча нужно обойти дерево в обратном направлении, учитывая ослабление луча при каждом отражении. В теории, глубина дерева может быть любой, но его построение можно прерывать при достижении определённого уровня или при исчерпании запаса памяти.

44

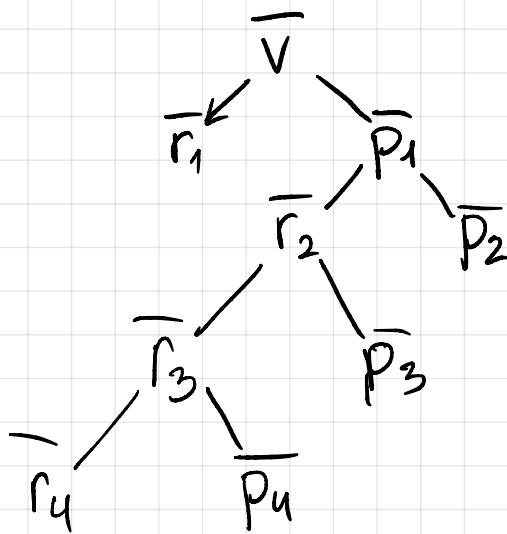
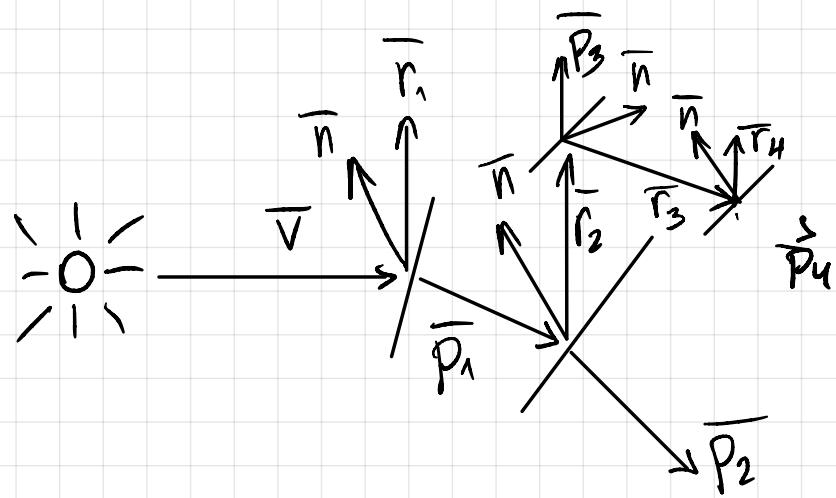
$$I = I_p k_p + I_u k_g \cos \theta + I_u k_z \cos^2 \alpha + I_z k_z + I_{np} k_{np}$$

и - источник,

р - рассеянное освещение,

з - отражение,

пр - преломление



45. Определение направления преломленного луча.

В более сложных моделях освещения нам придется учитывать пропускающие свойства поверхностей, т.е. прозрачность этих поверхностей, что приводит к наличию преломленного луча. Для получения вектора преломленного луча воспользуемся законом преломления Снеллиуса, который утверждает, что падающий и преломленной лучи, а также вектор нормали лежат в одной плоскости, а углы падения и преломления связаны следующей формулой:

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

Как и в предыдущем случае, будем считать, что вектор преломления представляет собой линейную комбинацию вектора падающего света и вектора нормали, т.е.

$$T = \alpha L + \beta N$$

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

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

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

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 (3)

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

Равенство (3) 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 (4)

$$\cos \theta T = \alpha \cos \theta L - \beta$$

Возведя (0) в квадрат и произведя замену $\eta = \eta_1/\eta_2$, получим, после замены синуса на косинус получим (5)
 $\sin(\theta T)^2 = \eta^2 \sin(\theta L)^2$

Вычитая из (3) выражение (5), получаем

$$a^2(1 - \cos(\theta L)^2) = \eta^2(1 - \cos(\theta L)^2)$$

Из последнего уравнения получаем $\alpha = \eta$
(второй корень не подходит по физическому смыслу). Подставив полученное значение альфа в (5), получим уравнение

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

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

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

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

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

В последнем выражении под знаком квадратного корня может стоять отрицательная величина

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

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