12.02.14----------

Куров Андрей Владимирович, 517л

Компьютерная графика – совокупность методов и средств преобразования информации в графическую форму и из графической с помощью ЭВМ. Методы – математические и алгоритмы. Средства – технические и программные. Компьютерная (вычислительная) геометрия.

Любой геометрический объект может быть задан несколькими способами. Окружность – три точки; координаты центра и радиус. Параметрическое число геометрического объекта – минимальное количество параметров, задающих этот объект.

Прямая – 3 (Ax+By+C=0)

Прямоугольник – 5 (концы диагонали и угол)

Эллипс – 5 (центр, a,b, угол; или вписывая в прямоугольник)

Об аппроксимации кривых

Рассмотрим окружность, возьмём две точки. Минимальное расстояние между ними после округления должно быть >=1, иначе точки начнут сливаться. Пусть угол между радиусами этих точек равен θ, $lim\left(\frac{\sin\theta}{\theta}\right)=1.\ \sin\theta=\frac{\Delta x}{R}\implies\theta=\frac{1}{R}$. При достаточно большом радиусе кривизны кривой, две соседние точки кривой должны выбираться таким образом, чтобы величина угла (выраженная в радианах) между радиусами этих точек был не меньше чем 1/радиус. Радиус кривизны $R=\frac{(1+y'^2)^{\frac{3}{2}}}{|y''|}$. Если прямая задана параметрически, x(т) и y(т), то $R=\frac{(x'^2+y'^2)^{\frac{3}{2}}}{|x'y''-x''y'|}$. В полярных координатах $R=\frac{\left(\rho^2+\frac{d\rho}{d\phi}^2\right)^{\frac{3}{2}}}{\rho^2+2\left(\frac{d\rho}{d\phi}\right)^2-\rho\frac{d^2\rho}{d\phi^2}}$

Если взять две точки на кривой, провести через M1 касательную, то радиус кривизны - $R=\lim\limits_{M2\to M1}\frac{\overline{M1M2}}{\omega}$, где w – угол между касательной и осью абсцисс.

Для эллипса можно выделить два участка; граничной является точка, в которой $\frac{dy}{dx}=1$. На участке 0..x1 X изменяется с шагом 1 и вычисляется У, на участке x1..x2 У изменяется с шагом 1 и вычисляется X.

Преобразование изображений на плоскости.

Отличие фрагментов: 1. Местоположение и радиус, 2. Пропорции, 3. Ориентация.

При линейном преобразовании координаты преобразованной точки линейно зависят от координат исходной: x1=Ax+By+C, y1=Dx+Ey+F. A..Ф – параметры, однозначно определяющие преобразование.

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

Для описания преобразований часто используется матричная форма. Однако в ней нельзя описать, например, перенос: приходится использовать однородные координаты [X,Y,W], где w – масштабный коэффициент. В двумерной графике W=1. Декартовы координаты $x=\frac{X}{W},y=\frac{Y}{W}$.

Линейное преобразование в матричной форме: $(x1\ y1\ 1)=(x\ y\ 1)\begin{pmatrix}A & D & 0\\ B & E & 0\\ C & F & 1\end{pmatrix}$. Любое сколь угодно сложное преобразование может быть описано совокупностью трёх – переноса, масштабирования и поворота.

1. **Перенос на плоскости**: задается смещениями по X и У. X1=x+dx, y1=y+dy.

Матрица переноса $\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0.\\ dx & dy & 1\end{matrix}$

19.02.14----------

2. **Масштабирование** – изменение пропорций\размеров изображения. Задается четырьмя параметрами: координаты центра масштабирования, коэффициент по X, коэффициент по У. Все параметры имеют вещественный тип. Центр может располагаться в любой плоскости, проходящей через плоскость экрана.

Однородное масштабирование – коэффициенты одинаковые, неоднородные – с разными коэффициентами.

Центр Xm,Ym, точка X,Y, новая точка X1,Y1, коэффициенты Kx,Ky,

X1 – Xm = Kx (X-Xm)

Y1 – Ym = Ky (Y-Ym)

Таки образом, К.Новая = К.Центра + Коэффициент*(К.Старая – К.Центра)

В матричной форме, для случая Xm=Ym=0:

$M_{\text{масшт}} = \begin{matrix} Kx & 0 & 0 \\ 0 & Ky & 0 \\ 0 & 0 & 1 \end{matrix}$. В общем случае – вначале мы проводим ПЕРЕНОС, чтобы центр масштабирования совпал с центром коодинат, затем провести масштабирование, затем обратный перенос: $\mathrm{M}_{\text{переноса в 0}} * \mathrm{M}_{\text{масштабирования}} * \mathrm{M}_{\text{переноса обратно}}$.

Если коэффициент меньше 1 – изображение уменьшается, точки перемещаются ближе к центру масштабирования. Если больше – увеличиваются, точки перемещаются дальше. Если коэффициент меньше 0 – изображение заодно отзеркаливается.

3. **Поворот** – задается тремя параметрами: центр поворота, и углом поворота.

Центр Xc,Yc, точка X,Y, новая X1,Y1, O – угол на который поворачиваем, Ф – исходный угол относительно оси абсцисс. X1 =

$$Xc + R * Cos\left(180^o - (\phi + \theta)\right) =$$
$$Xc - R * Cos(\phi + \theta) =$$
$$Xc - R * \cos\phi\cos\theta + R * \sin\phi\sin\theta =$$
$$Xc - (Xc - X)\cos\theta + (Y - Yc)\sin\theta$$

$X1 = Xc + (X - Xc)\cos\theta + (Y - Yc)\sin\theta$. Аналогичными преобразованиями,

$Y1 = Yc + (X - Xc)\cos\theta - (Y - Yc)\sin\theta$.

В матричной форме: поворот относительно начала координат $\mathrm{M}_{\text{пов}} = \begin{matrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{matrix}$.

Определитель =1 – для любого поворота существует обратный поворот.

Основные свойства преобразований

Коммутативность – независимость результата преобразований от порядка, в котором они происходят. (X1 Y1 1) = (X Y 1)*M1*M2 – если M1 можно умножить на M2, то в ОБЩЕМ случае, M2 умножить на M1 нельзя – в общем случае, преобразование коммутативностью не обладает.

Коммутативные преобразования:

перенос	Перенос
масштаб	Масштаб
поворот	Поворот
Однородное масштабирование	Поворот

Доказательство коммутативности производится через перемножение матриц.

Аддитивные и мультипликативные операции – типа сложение и умножение.

Аддитивные: перенос-перенос и поворот-поворот. Чтобы найти итоговую матрицу такого поворота, достаточно сложить аргументы.

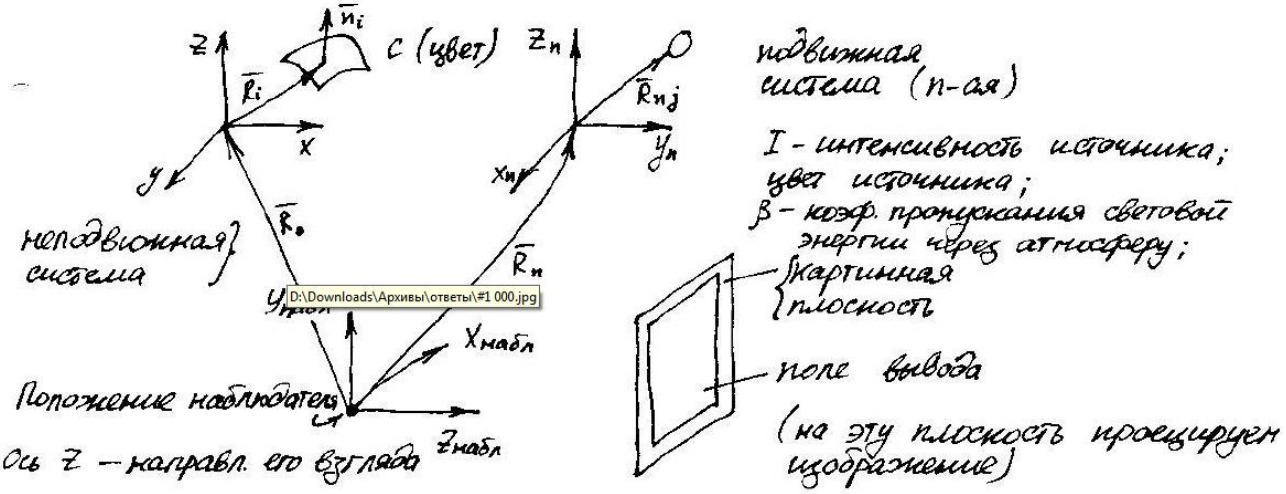
$$\mathrm{M}_{\text{пер1}}\mathrm{M}_{\text{пер2}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx1 + dx2 & dy1 + dy2 & 1 \end{pmatrix}$$

$$M_{пов1}M_{пов2} = \begin{pmatrix} \cos\theta_1 + \theta_2 & -\sin\theta_1 + \theta_2 & 0 \\ \sin\theta_1 + \theta_2 & \cos\theta_1 + \theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$kx1 * kx2$$

Мультипликативные: перемножить аргументы, $M_{масшт1}M_{масшт2}$

Синтез сложного динамического реалистичного изображения

Постановка задачи.



Зададим *подвижную* систему координат (У вверх). В ней зададим поверхность, задав вектор Ri некой центральной точки поверхности, и ориентацию поверхности, задав вектор нормали Ni этой точки. Поверхность обладает цветом Ci.

Поскольку изображение является реалистичным, необходимо передать различные оптические эффекты – отражение света (диффузное и зеркальное), его пропуск, затенение. Задаётся коэффициент зеркального Кзi и диффузного Кдиффi отражения, коэффициент пропускания Кпрi.

Далее задаётся *подвижная* система координат, связанная с неким объектом (радиусвектор Подi). Вектор Rпод идёт из центра неподвижной в центр подвижной системы.

Третья система координат связывается с наблюдателем. Начало её совмещается с положением точки наблюдения, Z направлена по линии взгляда, и изображение строится на картинной плоскости Хнабл:Унабл.

Также присутствуют источники освещения Iисti. Они задаются координатами Хисti, Уисti, Zисti, цветом (интенсивностью каждой из трех составляющих)

Также необходимы условия среды – коэффициент пропускания.

Для каждого динамического объекта требуется указать управляющие операторы, на основе которых может быть вычислено его положение для дискретных моментов времени, разделенных интервалами $\Delta T = \frac{1}{f}. f_{min} = 25{:}30$ гц.

Этапы синтеза

I. *Разработка (трехмерной) математической модели синтезируемой визуальной обстановки. Представление поверхности: Аналитическое описание или полигональная модель или т.п.*
II. *Определение положения наблюдателя в 3мерном пространстве, направления линии взгляда и положения экрана, размеров окна обзора, значения управляющих сигналов (для динамических объектов).*
III. *Формирование операторов, осуществляющих пространственное перемещение динамических моделируемых объектов*
IV. *Преобразование координат объектов в неподвижной системе в координаты в системе наблюдателя.*

V. *Отсечение объектов сцены (в том числе ФРАГМЕНТОВ объектов) в пределах пирамиды видимости (удаление того, что находится за пределами видимой области).*
VI. *Нахождение двумерных перспективных проекций объектов сцены на картинную плоскость.*
VII. *Исключение невидимых объектов сцены при заданном положении наблюдателя. Закрашивание и затенение видимых участков сцены.*
VIII. *Вывод полученного полутонового изображения на экран растрового дисплея.*

26.02.14----------

Алгоритмы нижнего уровня -> Алгоритмы среднего уровня (построение плоских изображений) -> Алгоритмы верхнего уровня (удаление невидимого, построение реалистичного изображения)

Растровая графика

Растровая развертка отрезка (разложение в растр)

Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок. Простейшие случаи – горизонтальный, вертикальный и под 45°. В большинстве случаев проявляется лестничный (ступенчатый) эффект.

Общие требования:

1. Отрезок должен выглядеть как отрезок прямой, начинаться и заканчиваться в заданных точках
2. Интенсивность (яркость) вдоль отрезка должна быть постоянной. Отрезки, имеющие разные углы наклона, должны быть одной интенсивности. Восприятие человека зависит не только от интенсивности свечения объекта, но и от расстояния между светящимися объектами //чтобы удовлетворить этому требованию, надо высвечивать точки с переменной интенсивностью от расстояния – потребует дополнительных вычислений, без особой нужды не используется
3. Алгоритмы (особенно нижнего уровня) должны работать быстро

Все алгоритмы имеют пошаговый характер – на очередном шаге высвечиваем пиксель, и производим вычисления, используемые в следующем шаге.

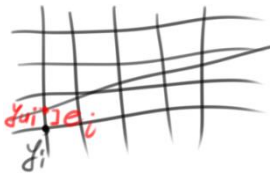
Алгоритм цифрового дифференциального анализатора

Если прямая задана каноническим уравнением $Ax + By + C = 0$, то производная $\frac{dy}{dx} = const$. Если заданы начальная и конечная точки x1y1 и x2y2, то $\frac{dy}{dx} = \frac{y2-y1}{x2-x1}$; $y_{i+1} = y_i + \Delta y = y_i + \frac{dy}{dx}\Delta x$. $\Delta x = 1$, а Δy – высчитывается и округляется. Однако $\Delta x = 1$ следует выбирать не всегда – вначале нужно ответить на вопрос, как наклонён отрезок. Если угол наклона <45, то |dX|=1, если > то |dY|=1

1. Начало
2. Ввод xн, yн, xк, yк
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку и переход на конец
4. $x_{тек} = x_н$, $y_{тек} = y_к$
5. $dx = x_k - x_n$, $dy = y_k - y_n$
6. Если |dx|>|dy|, то L=|dx| иначе L=|dy|
7. $dx = \frac{dx}{L}$, $dy = \frac{dy}{L}$
8. Построение отрезка (цикл по i=1 до L+1)
   a. Высвечивание точки E(xt),E(yt)
   b. Вычисление координат следующей точки xt += dx; yt += dy;
9. Конец

Недостатки – работает с целочисленной арифметикой (координаты текущей точки). Работает медленнее за счет операции округления.

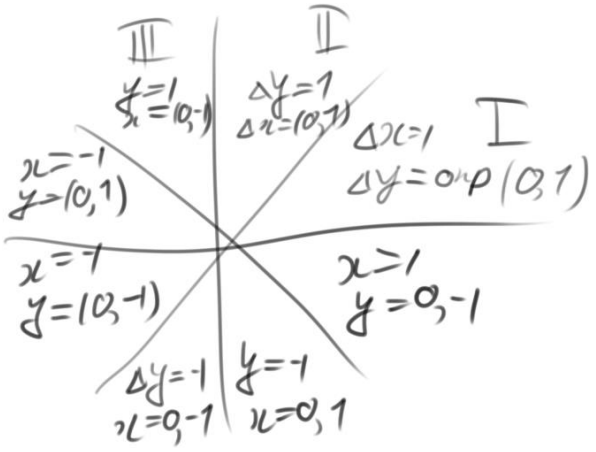
Алгоритмы Брезенхема



Ошибка ei – величина изменяющаяся на каждом шаге , расстояние между точкой самого отрезка и точкой, аппроксимирующей его на очередном шаге. $0 \le e_i \le 1$, если e<0.5, то ордината пикселя не меняется, $y_{i+1} = y_i$. Если >0.5, то выбирается верхний пиксель. $y_{i+1} = y_i + 1$. Начальное значение ошибки e=m, где м – тангенс угла наклона, а $e_{i+1} = e_i + m = y_{иi+1} - y_{i+1} = y_{иi} + m - y_{i+1}$. $y_и$ – здесь ордината идеального отрезка, в вещественных а не целых координатах.

На i+1 шаге $\left(y_{i+1} = y_i, \text{если } e < \frac{1}{2}\right)$ и $\left(y_{i+1} = y_i + 1, \text{если } e \ge \frac{1}{2}\right)$. В таком случае вычисляем предварительное $e_{i+1} = y_{иi+1} - y_{иi} + m - (y_i + 1) = y_{иi} - y_i + m - 1 = e_{i+1} - 1$. Если на очередном шаге мы корректируем значение y, то мы должны также скорректировать и ошибку.

Допускается начальное e=m-0.5 и дальше сравнивать ошибку с нулем, а не с 0.5



$$sx = sign(\Delta x);\ sy = sign(\Delta y)$$

$$sign(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$$

Алгоритм Брезенхема:

1. Начало
2. Ввод хн, ун, хк, ук
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку хн,ун и переход на конец
4. $x_{\text{тек}} = x_{\text{н}},\ y_{\text{тек}} = y_{\text{к}}$
5. $dx = x_k - x_n, dy = y_k - y_n$
6. Sx=sign(dx), sy=sign(dy)
7. Dx=|dx|, dy=|dy|
8. Если dx>dy то обмен=0, иначе обмен=1 {t=dy; dy=dx; dx=t}
9. $m = \frac{dy}{dx}$.
10. Цикл построения отрезка (по i=1 to dx+1)
11. Конец

05.03.14----------

Алгоритм Брезенхема:

1. Начало
2. Ввод хн, ун, хк, ук
3. Проверка отрезка на вырожденность: если вырожденный то высветить точку хн,ун и переход на конец
4. $x_{тек} = x_н,\ y_{тек} = y_к$
5. $dx = x_к - x_н,\ \ dy = y_к - y_н$
6. Sx=sign(dx), sy=sign(dy)
7. dx=|dx|, dy=|dy|
8. Если dx>dy то обмен=0, иначе обмен=1 {t=dy; dy=dx; dx=t}
9. $m = \frac{dy}{dx};\quad e = m - 0.5$ **(1)**. //тангенс угла наклона
10. Цикл построения отрезка (по i=1 to dx+1)
    a. Высвечивание точки (xt,yt)
    b. Если (e>=0) {
       если (обмен=0) {
         yt=yt+sy
       } иначе {
         xt=xt+sx
       }
       e=e-1 **(2)**
    } иначе {
       если (обмен=0) {
         xt=xt+sx
       } иначе {
         yt=yt+sy
       }
       e=e+m **(3)**
    }
11. Конец

Недостатки: не все переменные являются переменными целого типа (e,m - действительные).

Чтобы перейти к алгоритму, работающему ТОЛЬКО с целыми:

$e = \frac{dy}{dx} - \frac{1}{2};\ \Rightarrow\ 2dxe = 2dy - dx;\ \Rightarrow\ \bar{E} = 2dy - dx$ (1). Тогда в цикле в (2) будет e=e-2dx; в (3) e=e+2dy; не нужно вычислять m в 9м шаге.

Брезенхем предложил простейший алгоритм сглаживания; алгоритм Брезенхема с устранением ступенчатости.

Используется при отображении ребёр многоугольника, который закрашивается. Идея состоит в сглаживании резких переходов от ступени к ступени. Сглаживание основывается на том, что каждый пиксель высвечивается со своим уровнем интенсивности. Уровень выбирается пропорционально площади части пикселя. 1 пиксель – квадрат с единичной стороной, а не математическая точка.

Так как интенсивность I~Si площади, то

1) отрезок связан (покрывает) на i шаге с одним пикселем. Обозначим Yi расстояние по вертикали от точки пересечения отрезка с пикселем, до левой нижней границы пикселя. Обозначим тангенс угла наклона отрезка через m, тогда Si=Sпр+Sтр = Yi*1 + 1*m/2 = Yi + m/2

2) отрезок покрывает на I шаге два пикселя. Yi – расстояние от нижней границы до пересечения с отрезком. Площадь нижней части S1 = Sпикс – Sтр1 = $1 - \frac{(1-Yi)^2}{2m}$. Площадь части второго пикселя S2 = $\frac{(m-1+Y_i)^2}{2m}$. Складывая площади, S1+S2 = Yi + m/2

1) $Si \quad Si+1$ На очередном шаге Si = Yi + m/2; S(i+1)=Sпр1 + Sпр2 + Sтр = Si + m

2) Si $Si+1$ X S(i+1) = Si + m, но высвечивается верхний пиксель, нижний – не высвечивается, и его площадь не учитывается. S = S+m; если $y_{i+1} = y_i$, то выражение корректируется, если $y_{i+1} = y_i + 1$, то S=S-1 (вычитаем площадь нижнего пикселя)

В данном случае в качестве ошибки можно рассматривать ei=Si (интенсивность пропорциональна ошибке) – однако её нельзя будет скорректировать через -0.5. За пороговый уровень можно взять другое значение. Обозначим w=1-m, e = e + w = m-0.5 + 1-m = 0.5. I=Imax/2 – начальный пиксель всегда высвечивается в половинной интенсивности.

W – пороговое значение. Если e>=w, то $y_{i+1} = y_i + 1$ (e=e-w), иначе Y(i+1)=Yi. Чтобы внутри цикла не приходилось постоянно умножать на интенсивность, сделать это можно один раз в начале работы алгоритма (m=Imax*m, e=Imax*e, w=Imax-m).

При реализации алгоритма без сглаживания исходные данные – начальные и конечные координаты. Здесь же добавится количество уровней интенсивности, либо же максимальный уровень.

Алгоритм Брезенхема для построения окружности

$$(x - x_c)^2 + (y - y_c) = R^2$$

Будем считать, что центр находится в начале координат, xc=yc=0. У направлена вверх, X вправо.

Окружность – симметричная фигура, можно построить половину или четверть и отражать и поворачивать. Y=A(x) (отрисовка четвертькруга от (0,R) до (R,0)) – в первой четверти монотонно убывающая. На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

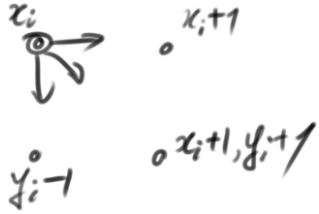
Критерий $\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$ – местный аналог ошибки. Разность квадратов (расстояния от центра окружности до диагонального пикселя) и (расстояния от центра окружности до самой окружности).
Если >0, то диагональный пиксель лежит вне окружности. Выбирается диагональный или вертикальный пиксель для отрисовки.
Если =0, то диагональный пиксель лежит ровно на окружности. Выбирается диагональный.
Если <0, то диагональный пиксель лежит внутри окружности. Выбирается диагональный или горизонтальный.

12.03.14----------

Алгоритм Брезенхема для построения окружности

$$(x - x_c)^2 + (y - y_c) = R^2$$

Будем считать, что центр находится в начале координат, xc=yc=0. У направлена вверх, X вправо.



Окружность – симметричная фигура, можно построить половину или четверть и отражать и поворачивать. Y=A(x) (отрисовка четвертькруга от (0,R) до (R,0)) – в первой четверти монотонно убывающая. На очередном шаге возможен переход из (xi,yi) в (xi+1,yi), (xi,yi-1), (xi+1,yi-1).

Критерий $\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$ – местный аналог ошибки. Разность квадратов (расстояния от центра окружности до диагонального пикселя) и (расстояния от центра окружности до самой окружности).

$$\delta_1 = \left|(x_i + 1)^2 + y_i^2 - R^2\right| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

Если $\Delta < 0$, то диагональный пиксель лежит внутри окружности. Выбирается диагональный.

Случай1: первый модуль >=0, второй <0. $\delta = (x_i + 1)^2 + y_i^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2y_i - 1 = 2\Delta + 2y_i - 1$

Случай2: первый модуль <=0, второй <0, выбор пикселя очевиден, $(x_i + 1, y_i)$. $\delta = -(x_i + 1)^2 - y_i^2 + R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2 = -2y_i + 1$, где 0<=yi<=R

Если $\Delta = 0$, то диагональный пиксель лежит ровно на окружности. Выбирается диагональный.

Если $\Delta > 0$, то диагональный пиксель лежит вне окружности. Выбирается диагональный или вертикальный пиксель для отрисовки.

$\delta_2 = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - \left|x_i^2 + (y_i - 1)^2 - R^2\right|$. Если <0, то выбирается диагональный, если =0 – любой из двух, >0 – вертикальный.

Случай4: бельта2 =0, выбор пикселя (xi,yi-1) очевиден, но надо проверить знак бельта2. $(x_i + 1)^2 + (y_i - 1)^2 - R^2 > 0$; $x_i^2 + (y_i - 1)^2 - R^2 > 0$; $\delta_2 = (x_i + 1)^2 - x_i^2 = 2x_i + 1 > 0$.

Для случая $\Delta = 0$ (случай5) выбор очевиден, но надо проверить знаки бельта1 и бельта2 на предмет отсутствия противоречий. $\delta_1 = (xi + 1)^2 + yi^2 - r\hat{}2| - |(xi + 1)^2 + (yi - 1)^2 - r^2|(== 0)$ $\delta_1 > 0$;

$$\delta_2 = |(xi + 1)^2 + (yi - 1)^2 - r^2|(== 0) - |xi^2 + (yi - 1)^2 - R^2|\ \delta_2 < 0$$

Δ_{i+1} надо выражать через x_{i+1} и y_{i+1}

Горизонтальный шаг: $x_{i+1} = x_i + 1$; $y_{i+1} = y_i$; $\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 = (x_i + 2)^2 + (y_i - 1)^2 - R^2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_i + 3 = \Delta_i + 2x_{i+1} + 1$

Диагональный шаг: $x_{i+1} = x_i + 1$;

Вертикальный шаг: $x_{i+1} = x_i$; $y_{i+1} = y_i - 1$; $\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 = \cdots = \Delta_i - 2y_{i+1} + 1$

Начальное значение Δ $(0, R)$: $\Delta = (0 + 1)^2 + (R - 1)^2 - R^2 = 1 - 2R + 1 = 2(1 - R)$

Алгоритм Брезенхема для построения эллипса

$\frac{(x-xc)^2}{a^2}+\frac{(y-yc)^2}{b^2}=1$. Можно использовать операцию переноса центра, тогда $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$. Учесть ненулевые координаты центра можно с помощью

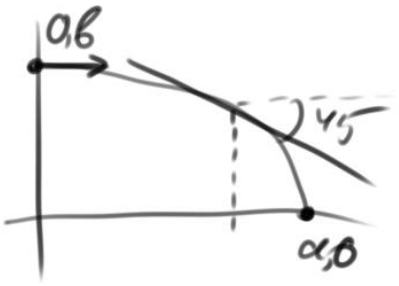
$$xэ1 = xc + x, yэ1 = yc + y$$

$$xэ2 = xc - x, yэ2 = yc + y$$

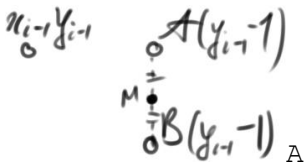
$$xэ3 = xc - x, yэ3 = yc - y$$

$$xэ4 = xc + x, yэ4 = yc - y$$

$$\begin{cases} x(t) = xc + a * \cos t \\ y(t) = yc + b * \sin t; \ \ t \in [0,2\pi) \end{cases}$$



В этой точке $\frac{dy}{dx}=-1$. На первом интервале – единичное приращение по x, приращение по y определяется. На втором – наоборот.



1 интервал: A и B – альтернативные пиксели, M – точка между ними. Зная, как проходит дуга эллипса относительно средней точки, можно выбрать пиксель A или B.

Случай1: средняя точка внутри эллипса. Дуга эллипса проходит между точкой A и M => выбрать для отрисовки нужно пиксель A.

Случай2: средняя точка вне эллипса. Дуга эллипса проходит между точкой M и B => выбрать нужно пиксель B.

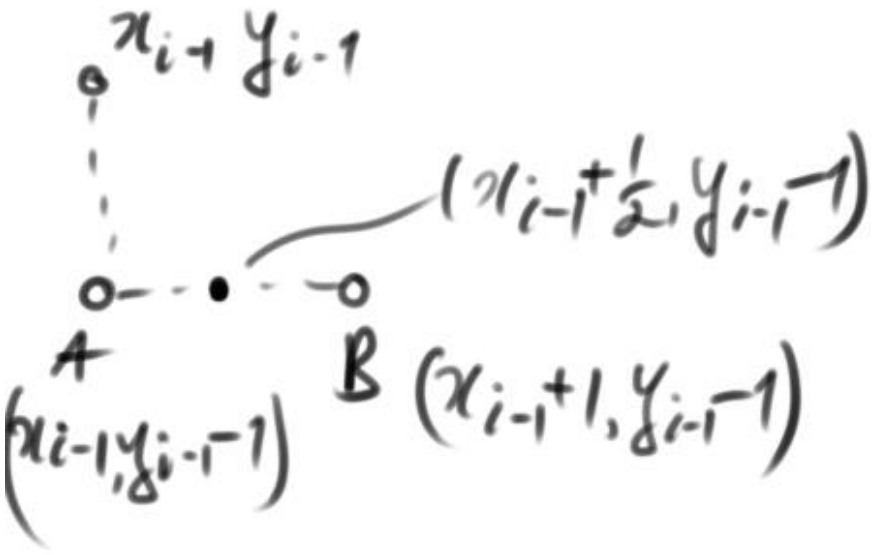
Таким образом, осталось определить положение точки M относительно эллипса. $b^2x^2+a^2y^2-a^2b^2=0$. Значения XY обращающие в равенство – принадлежат эллипсу.

Пробная функция: $f=b^2x^2+a^2y^2-a^2b^2=\begin{cases}=0, \text{если точка принадлежит} \\ >0, \text{лежит вне} \\ <0, \text{лежит внутри}\end{cases}$ Вычислим f_i через предыдущее значение. Изменение $df=f_i-f_{i-1}=b^2(x_{i-1}+1)^2+a^2\left(y_{i-1}-\frac{1}{2}\right)^2-a^2b^2-\left[b^2x_{i-1}^2+a^2\left(y_{i-1}-\frac{1}{2}\right)^2-a^2b^2\right]=2b^2x_{i-1}+b^2=b^2(2x_{i-1}+1)$

В итоге можно отказаться не только от возведения в квадрат, но и от умножения. В итоге $dx=dx+bd;\ df=df+b^2+dx$, где b2=b*b, bd=2b2

Если на очередном шаге была выбрана нижняя точка B, то необходимо скорректировать значение пробной функции, поскольку на последующем шаге вычисления должны проводиться для точки M1, лежащей ниже этой B.

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



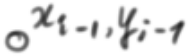
2 интервал:

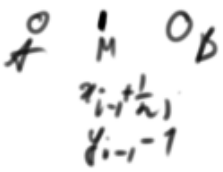
Случай 1: точка M лежит вне эллипса (эллипс проходит между A и M), выбираем A.

Случай2: M лежит внутри эллипса (эллипс проходит между M и B), выбираем B.

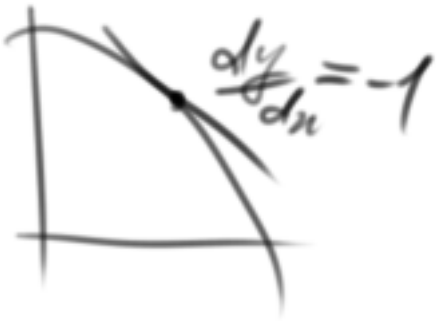
$$df_i = b^2\left(x_{i-1} + \frac{1}{2}\right)^2 + a^2(y_{i-1} - 1)^2 - a^2b^2 - \left[b^2\left(x_{i-1} + \frac{1}{2}\right)^2 + a^2\left(y_{i-1}^2\right) - a^2b^2\right] = -2a^2 y_{i-1} + a^2$$

$$dy = dy - ad;\ \ df = df + a2 - dy, \qquad a2 = a * a, ad = 2a2$$





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

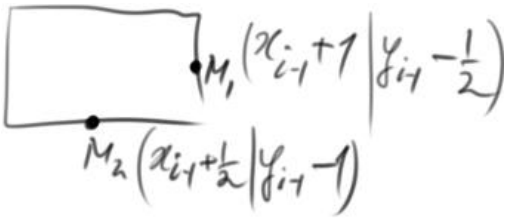


$b^2x^2 + a^2y^2 - a^2b^2 = 0;\ \ 2b^2xdx + 2a^2ydy = 0;\ \ \frac{dy}{dx} = -\frac{b^2x}{a^2y} = -1;\ \ a^2y = b^2x.$

$b^2x^2 + a^2y^2 - a^2b^2 = 0.$ A4y2=b4x2 (1), отсюда

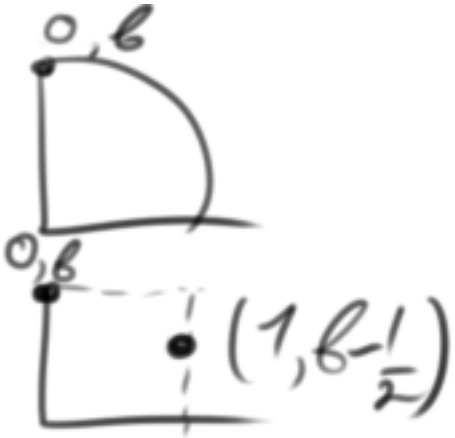
$a^2y^2 = a^2b^2 - b^2x\text{^}2.$ $a^2(a^2b^2 - b^2x^2) = b^4x^2 \Rightarrow\ a^4b^2 - a^2b^2x^2 = b^4x^2 \Rightarrow\ x^2(b^4 + a^2b^2) = a^4b^2$. Отсюда имеем $x = \frac{a^2b}{b\sqrt{a^2+b^2}} = \frac{a^2}{\sqrt{a^2+b^2}}$, т.е. $x \in \left[0, \frac{a^2}{\sqrt{a^2+b^2}}\right]$

Необходимо также сделать вторую коррекцию на границе интервалов. Сделаем её однократно, в отличие от предыдущей, которая выполнялась двукратно.



$$\Delta f_{\text{кор}} = b^2\left(x_{(i-1)}+\frac{1}{2}\right)^2 + a^2(y_{i-1}-1)^2 - a^2b^2 - \left[b^2\left(x_{(i-1)}+1\right)^2 + a^2\left(y_{i-1}-\frac{1}{2}\right)^2 - a^2b^2\right]$$
$$= -b^2x_{i-1} - \frac{3}{4}b^2 - a^2y_{i-1} + \frac{3a^2}{4} = \frac{3}{4}(a^2-b^2) - (b^2x_{i-1} + a^2y_{i-1})$$

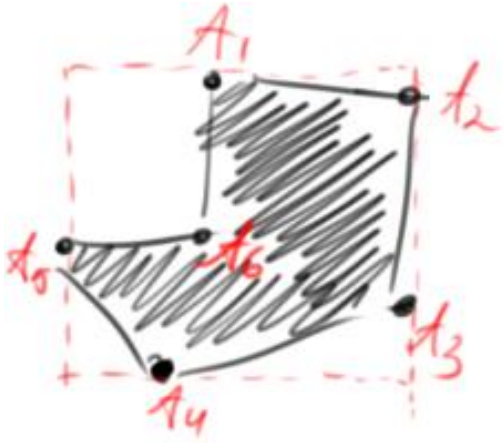
Наконец надо вычислить функцию в начальной точке.



$$A_{\text{пр}} = b^2 * 1^2 + a^2\left(b-\frac{1}{2}\right)^2 - a^2b^2 =..= b^2 - a^2b + \frac{a^2}{4}$$

Растровая развёртка сплошных областей (заливка).

Генерация сплошных областей на базе простых описаний вершин или рёбер многоугольника.



1 границы можно не отрисовывать; принадлежат внутренней области

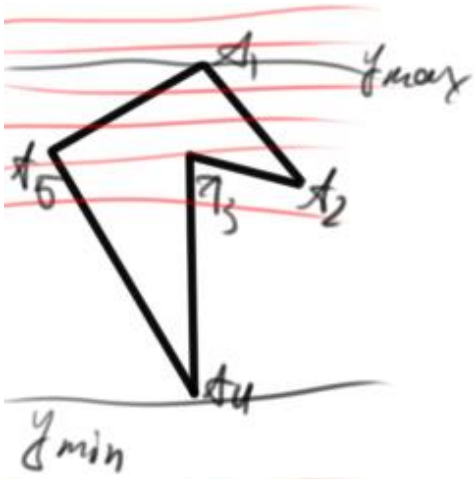
2 Алгоритм, справляющийся с произвольным многоугольником (выпуклые/невыпуклые; внутри могут содержаться границы

Фактически, задача сводится к определению принадлежности каждого пикселя внутренней области, ограниченной прямоугольник. Это определяется путём «запуска вектора». Если он пересекает прямоугольник дважды – это закрашенная область, единожды – вершина.

Условия: xmin <= x <= xmax && ymin <= y <= ymax.

Если область, которую нужно закрасить, значительно меньше прямоугольника граничных координат, то КПД простейшего алгоритма чрезвычайно мал.

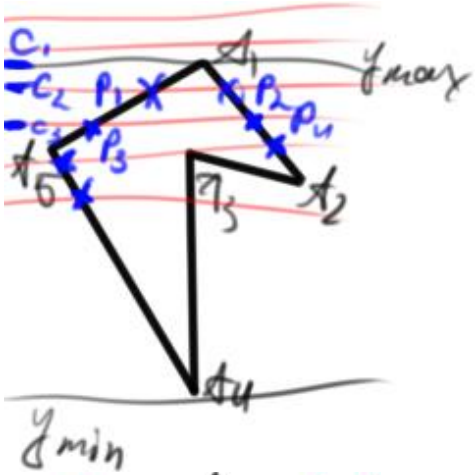
Все алгоритмы заполнения можно разделить на две группы – растровые и затравочные.



В растровом алгоритме для каждой строки каждый раз решается задача определения, лежит пиксель внутри многоугольника или нет. Рассматриваются сканирующие строки, пересекающие многоугольник.

Для пикселей каждой сканирующей строки определяется их принадлежность многоугольнику.

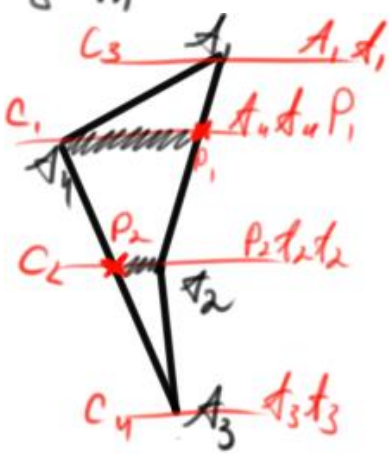
Уменьшить трудоёмкость задачи можно если принять во внимание факт того, что пиксели, расположенные друг рядом с другом, сохраняют неизменным «свойство принадлежности».



C1: [A1..A1]

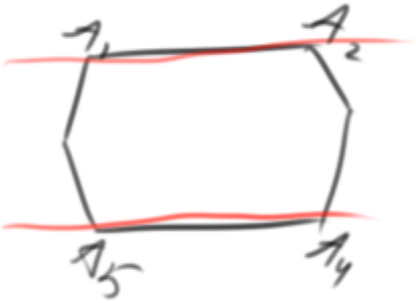
C2: [P1..P2]

C4: [P3..A3] [A3..P4]



Используя подход, связанный с учетом когерентности пикселей (одинаковые свойства рядом расположенных пикселей), проблема возникает только для строк, проходящих ровно через вершины многоугольника. Проблема: для одних строк количество пересечений надо принимать ==2, а для других ==1

Если в примере выше взять ребра A1A5 и A1A2, то y1>y5 && y1>y2. Аналогично, y4<y5, y4<y3. А вот для пятой вершины, y5<y1 и y5>y4 – точка A5 не является точкой экстремума; в то время как A1 и A



Вторая проблема – с горизонтальными рёбрами. Ребро лежит на сканирующей строке – огромное число «пересечений». Решение – выкинуть ребро. Тогда сканирующая строка всёравно будет пересекаться с А1.. но уже как с концами других рёбер – это не значит, что они не будут закрашены.

Два тонких момента – горизонтальные ребра (можно не рассматривать) и учет сканирующих строк, проходящих ровно через вершины многоугольника.

Рассмотрим конкретные алгоритмы.

Алгоритм заполнения с упорядоченным списком ребер

Активное ребро – ребро, которое в данный момент рассматривается. Ребро, пересекаемое текущей сканирующей строкой. Список активных ребер – список, хранящий информацию об активных рёбрах.

Алгоритм для нахождения точек персечения (прошлая лекция) использовать нельзя – отрезок представлен в виде совокупности ступенек, а значит вместо одной точки пересечения может получиться несколько, и результат заполнения будет определяться четностью\нечетностью количества точек. Необходимо сочетать два подхода – разлагая отрезок в растр надо получать одну точку пересечения для сканирующей строки. $x_A, \Delta x, \Delta y$ – информация об отрезке.

dX – фактически котангенс угла наклона. $\Delta x = ctg\ \alpha = \frac{xk-xn}{yk-yn}$. $x_A = x_A + \Delta x;\ \Delta y = \Delta y - 1$. Когда dY <0 – отрезок обработан.

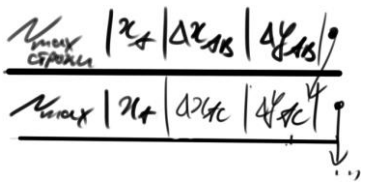
Простой алгоритм:

1. найти пересечение всех отрезков со всеми сканирующими строками (x1,y1), ... (xк,yк).
2. найденные точки пересечения упорядочиваем по убыванию координаты У
3. сортировка точек пересечения, принадлежащих одной (каждой) сканирующей строке по возрастанию X. (x1,y)...(xн,y), x1<=x2<=...xн
4. упорядоченный список для каждой сканирующей строки разбить на пары и закрасить все пиксели, расположенные в интервале, ограниченном очередной парой.

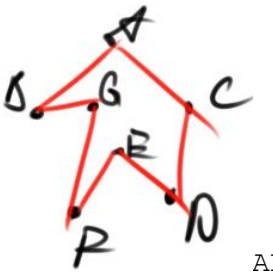
Более эффективный вариант – необходимо избавиться от сортировки всего множества точек пересечения. Для хранения точек в простейшем случае можно использовать массив, и разбить его на n участков, по количеству сканирующих строк.

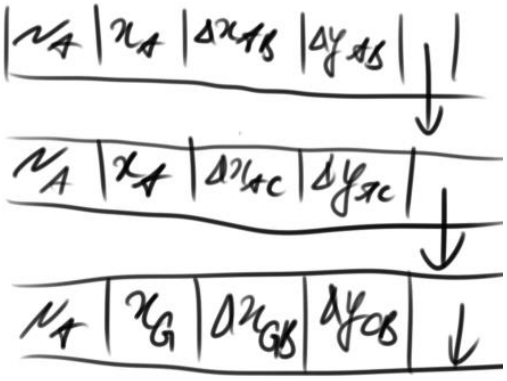
1. находимые точки пересечения сразу заносятся в нужную группу
2. упорядочить содержимое каждой y-группе по возрастанию X
3. образовать пары внутри каждой y-группы и закрасить все пиксели в интервалах.

При этом использовать массивы крайне невыгодно – при статике будет перерасход памяти, при динамике требуется копирование элементов при перевыделении. Можно использовать однонаправленные списки.



Информацию о ребрах в список надо заносить в порядке убывания максимального значения координаты У ребра.





AB, AC, GB, GF, CD, EF, ED.

Если N текущее > N первого элемента списка, то список активных ребер пуст, многоугольник «ещё не начался». Если N текущее <= N первого элемента списка, то строка пересекает многоугольник (САР не пуст).

Список просматривается до тех пор, пока выполняется условие N тек. <= N очередного элемента списка. При просмотре списка

1) выбираем Xij абсциссы точек с текущей строкой.
2) корректируем поля элемента списка: $Xij = Xij + \Delta_{ij}$; $\Delta y_{ij} = \Delta y_{ij} - 1$. Если dy<0, то элемент списка удаляется из списка. Удалив обработанное ребро, избавляемся от необходимости просмотра уже просмотренных ребер.
3) обработка ведется пока список не опустеет

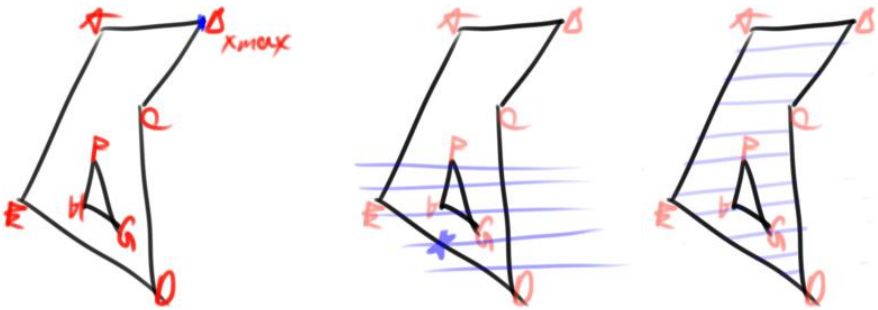
Сформированный список абсцисс точек пересечения ребер с очередной сканирующей строкой надо опять отсортировать по возрастанию и закрасить пиксели, расположенные внутри каждого интервала.

Алгоритм заполнения принято оценивать количеством операций, связанных с изменением цвета пикселя (и получения информации о его цвете), и самим количеством пикселей. В данном алгоритме, обрабатываются только те пиксели, которые лежат внутри области, И каждый пиксель обрабатывается по одному разу. Следовательно, с точки зрения отрисовки данный алгоритм является одним из наиболее быстродействующих. Тем не менее, для некоторых фигур алгоритм становится значительно более накладным («расческа»).

Алгоритм заполнения по ребрам

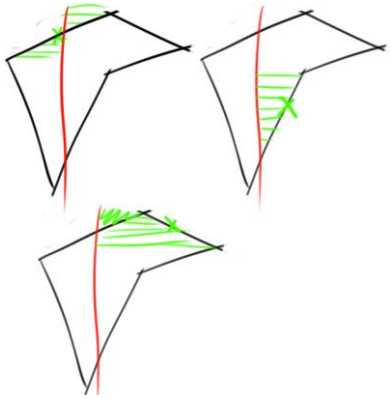
Ребра никак не упорядочиваются, сортировка не происходит. В этом алгоритме ребра могут обрабатываться в произвольном порядке. Алгоритм прост, но требует бОльших временных затрат, поскольку некоторые пиксели могут обрабатываться несколько раз.

Задается цвет фона и цвет закраски; цвет закраски = !(цвет фона), цвет фона = !(цвет закраски) (инверсия/дополнение). Суть алгоритма: для каждой сканирующей строки, пересекающей ребро многоугольника, дополнить (проинвертировать) все пиксели, расположенные правее точки пересечения.



Количество обработок пикселя определяется количеством ребер, расположенных левее этого пикселя.

Алгоритм заполнения с перегородкой: для каждой сканирующей строки пересекающей ребро, дополнить все пиксели, расположенные правее точки пересечения, но левее перегородки (если x < x_пер). Затем дополнить все пиксели, расположенные левее точки пересечения, но правее перегородки (если x > x_пер).



Алгоритм со списком ребер и флагом

Флаг – признак принадлежности точки внутренней или внешней области многоугольника. =true , если точка расположена внутри. Алгоритм двушаговый:

1) очертить границы закрашиваемого многоугольника
2) закрасить ограниченный многоугольник

02.04.14----------

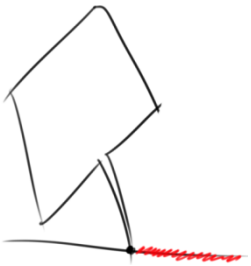
Алгоритм со списком ребер и флагом

Флаг – признак принадлежности точки внутренней или внешней области многоугольника. =true , если точка расположена внутри. Алгоритм двушаговый:

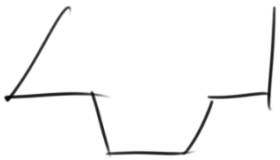
1) очертить границы закрашиваемого многоугольника
2) закрасить ограниченный многоугольник

Цикл по всем сканирующим строкам (по y от Ymax до Ymin)
    Flag = false
    Цикл по всем пикселям сканирующей строки (по x от 0 до Xmax)
        Если цвет(x,y) = цвет границ, то flag=!flag; продолжаем
        Если флаг=истина, то цвет(x,y) = цвет закраски; иначе цвет(x,y) = цвет фона
    Конец цикла по X
Конец цикла по У

Возникающие проблемы:



На стадии отрисовки границ, два ребра могут сливаться в одно. Как вариант – отрисовывать границу с использованием инверсного цвета пера. Либо же, первое ребро отрисовывается, при отрисовке последующих – если точку надо поставить в ту точку, которая уже подсвечена, то поставить её рядом.



Также извечный вопрос строки, проходящей через вершины.

Также, использовать обычный способ отрисовки отрезков нельзя – может возникнуть разбиение; точек пересечения со сканирующей строкой может оказаться несколько.

Скорость работы алгоритма закраски определяется количеством операций обращений к пикселю. И первый и последний алгоритмы обрабатывают каждый пиксель единожды, НО – нужно и смотреть, сколько обрабатывается пикселей. В некоторых алгоритмах анализируются пиксели, лежащие не только внутри обрабатываемой области.

Алгоритмы затравочного заполнения

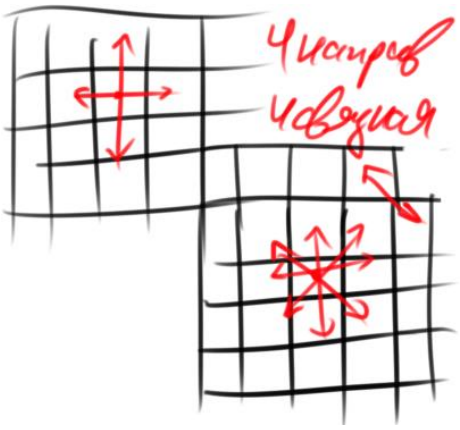
1.должна быть задана область, подлежащая заполнению (гранично-определённая, или внутренне-определённая)

2.должен быть задан затравочный пиксель – точка, заведомо лежащая внутри области

Способы задания области.

1. Гранично-определённая => задана граница области, т.е. известен цвет границы

2. Внутренне-определённая область => все пиксели, принадлежащие области, имеют один и тот же цвет – этот цвет должен быть известен. Закрашиваемые области могут быть четырёхсвязными и восьмисвязными.Связность можно определить количеством направлений движения для достижения точки области.

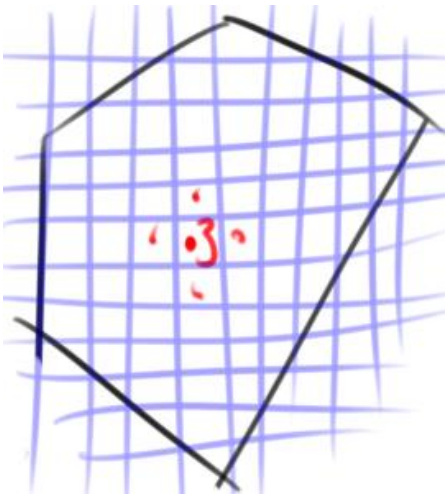


Граница является 8связной; у 8связной граница является 4связной.

Алгоритмы можно разделить на алгоритмы для гранично-определённых областей, и внутренне-определённых областей. Также для 4связных областей и для 8связных областей, причем 8связный алгоритм справляется и с 4связной.

Простой алгоритм заполнения с затравкой

В алгоритмах заполнения очень удобно и эффективно использовать стек для хранения затравочных пикселей.



3(x,y); (x,y+1); (x+1,y); (x,y-1); (x-1; y)

1.Задаются граница, цвет границы, координаты затравочного пикселя

2.занесение затравочного пикселя в стек

3.пока стек не пуст, выполнить следующие действия:

- извлечь пиксель из стека (координаты)

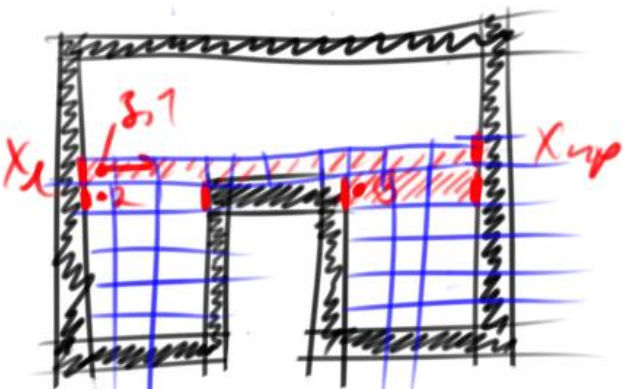
- закрасить извлеченный пиксель (возможно, с проверкой на цвет – нужно ли закрашивать или он уже)

- проанализировать цвет 4-х соседних пикселей ^. Если пиксель не является граничным, то поместить его в стек

Алгоритм достаточно прост в реализации и понимании. Недостатки: перерасход памяти – в стек заносится любой незакрашенный неграничный пиксель, т.е. в стеке побывает каждый пиксель области. В наихудшем случае, пиксель может быть занесён в стек аж четыре раза.

Построчный алгоритм заполнения с затравкой

Для улучшения характеристик алгоритма, в стек надо заносить не каждый незакрашенный пиксель, а один на группу незакрашенных. Под группой будем понимать пиксели в одной строке, образующие непрерывный интервал. Это группа примыкающих друг к другу пикселей, ещё не закрашенных, ограниченных уже закрашенными\граничными пикселями. В стек заносится самый левый или самый правый пиксель.



Анализируем пиксель – закрашиваем – анализируем пиксели сверху и снизу. Каждый пиксель анализируется, а затем может быть рассмотрен ещё как и верхняя и нижняя граница – рассматривается три раза.

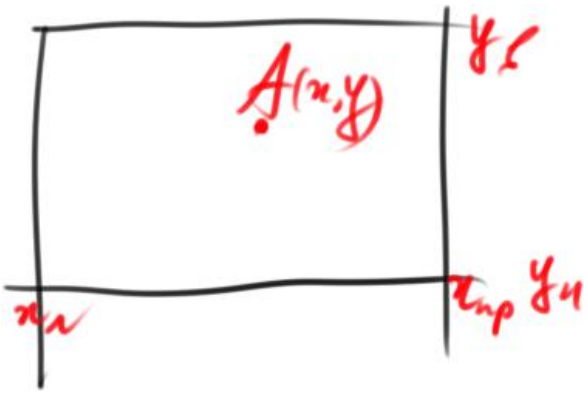
```
1. Если стек пуст
2. то конец
3. иначе: извлечение пикселя (x,y) из стека
4. цвет(x,y) = цвет закраски
5. tx = x, запоминаем абсицссу
   //два однотипных куска, обеспечивают закраску пикселей слева и справа, с
   запоминанием границы
6. двигаемся влево, x=x-1
7. если цвет(x,y) != цвету границы
 8. цвет(x,y) = цвет закраски; гото 6
9. иначе: Xleft = X+1
10. x=tx
11. двигаемся вправо, x=x+1
12. если цвет(x,y) != цвету границы
13. цвет(x,y) = цвет закраски; гото 11
14. иначе: Xright = X-1
   //два однотипных куска, обеспечивают закраску пикселей слева и справа, с
   запоминанием границы
15. x = Xleft
16. y = y+1 рассматриваем строку сверху
17. если x <= Xright (иначе гото
18. то Fl=0 //признак нахождения
19. Если цвет(x,y) != цвету закраски (иначе гото 25)
20. То если цвет(x,y) != цвету границы (иначе гото 25)
21. То если x<=хпр (иначе гото 25) (иначе гото 25)
22. То если Fl=0
 23.    То FL=1
24. X=X+1; гото 19

   № по журналу mod 4:
   1 -> с упорядоченным списком
   2 -> по ребрам
   3 -> с перегородкой
   0 -> с флагом
   Реализовать алгоритм: нарисовать область; закрасить
```

09.04.14----------

Алгоритмы отсечения

Отсечение (внешнее) – операция удаления части изображения, находящегося за пределами некоторой заданной области (отсекателя). Удаление части области внутри отсекателя – стирание.

Отсечение может происходить на плоскости и в 3Д-пространстве. Для его выполнения необходимо задать отсекатель – регулярный или нерегулярный. Регулярные: прямоугольник на плоскости, стороны параллельны осям; параллелепипед в пространстве, грани параллельны плоскостям. Нерегулярные: выпуклые и невыпуклые многоугольники; четырёхгранная усеченная пирамида (правильная).



Границы относят к внутренностям области. Точка – лежит либо внутри, либо снаружи. Точка видима, если находится внутри отсекателя и невидима иначе.

Отрезок – полностью невидимый (целиком лежит за пределами отсекателя), полностью видимый (полностью лежит внутри отсекателя), частично видимый (часть отрезка в пределах, часть за). Отрезок полностью видим, если обе вершины расположены в пределах, как определить видимость точки известно. Если обе вершины НЕВИДИМЫ, то отрезок может быть как ПОЛНОСТЬЮ невидимым, так и ЧАСТИЧНО невидимым.

Используются коды, обозначающие положение точки. T1 = 0, если x>=хл, 1 иначе. T2=0, если x<=хпр, 1 иначе. T3 = 0, если y>=упр, 1 иначе. T4 = 0, если y<=ув, 1 иначе.

Если сумма кодов равна 0, то точка видима, если !=0 – невидима.

(S1=0) & (s2=0) => отрезок видимый. Если одна из сумм !=0, то частично видимый. Если обе суммы !=0, то ситуация неопределена – толи частично, толи полностью невидимый.

$$p = \sum_{i=1}^{4} (T1iT2i)$$

P!=0 => отрезок является тривиально невидимым.

Вычисляя суммы и сравнивая с нулём, можно идентифицировать отрезок.

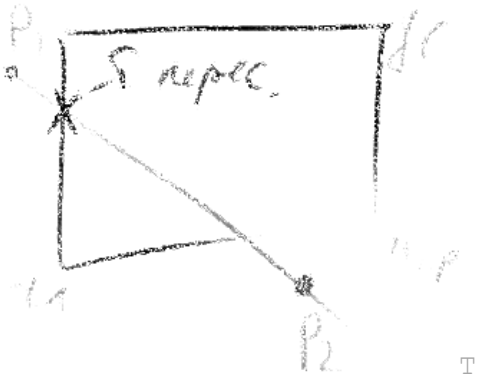
Y=kx+b, k!=infinity. $y = m(x - x1) + y1, m \neq \infty$.

$y = m(x_л - x_1) + y_1$ – ордината точки пересечения с левой границей, $y = m(x_{пр} - x1) + y_1$ – с правой.

$x = \frac{1}{m}(y - y_1) + x_1,\ m \neq 0$. Соответственно, $x = \frac{1}{m}(y_н - y_1) + x_1$ – точка спересечения с нижней границей, $x = \frac{1}{m}(y_в - y_1) + x_1$ – с верхней. М – тангенс угла наклона.

Прямая выпуклый многоугольник пересекает в двух точках (через вершину – они совпадают).

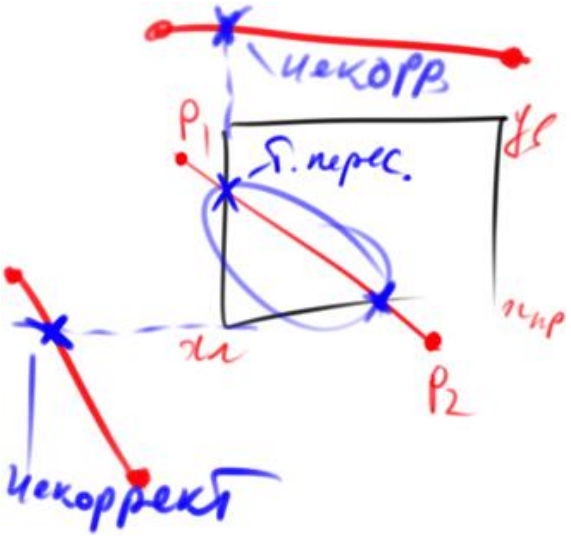
Простой алгоритм:



Точка пересечения проверяется на корректность.

16.04.14----------

Простой алгоритм отсечения:



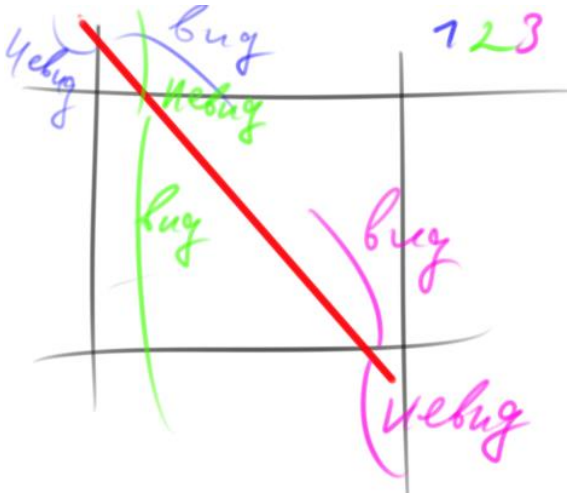
Точка пересечения проверяется на корректность.
Изначально флаг установлен Flag=0;

1. Ввод исходных данных: хл, хпр, ун, ув; p1,p2
2. Вычисление T1,T2 (кодов концов отрезка)
3. Проверка полной видимости (S1=0 и S2=0; $S_1 = \sum i = 1\,4\,(T_{1i})$; $S_2 = \sum i = 1\,4\,(T_{2i})$)
4. Проверка полей невидимости ($p = \sum i = 1\,4\,(T_{1i}T_{2i})$
   //к 5 пункту приходим с частично видимым отрезком
5. Проверка видимости 1й вершины: Если C1=0, то P1 заносится в результат; i=1 (остается найти вторую вершину – будь то P2 или точка пересечения отрезка с границей)
6. Проверка видимости 2й вершины: если C2=0, то P2 заносится в результат (аналогично)
7. Поиск очередной точки пересечения (i-й)
   7.1. Определение наличия пересечения отрезка с левой границей (Pi.x < Хл)
   7.2. Нахождение точки пересечения (ординаты) yt= m*(Pi.x – Хл) + Pi.y
   7.3. Проверка корректности пересечения Yн<= yt <=Yк (точка пересечения отрезка с границей лежит за пределами отсекателя); занесение точки в результат. i=i+1; повторяем пункт 7 если i<3
   7.4. Определение наличия пересечения отрезка с правой границей (Pi.x > Хпр)
   7.5. Нахождения точки пересечения yt = m*(Pi.x – Хпр) + Pi.y
   7.6. Проверка корректности; занесение точки в результат.
   7.7. Определение наличия пересечения отрезка с нижней границей (Pi.y < Yн)
   7.8. Нахождение точки пересечения xt = (Pi.y-Yн)/m + Pi.x
   7.9. Проверка корректности пересечения Хл<= xt <=Хпр; занесение точки в результат при корректности; i=i+1; поиск i-го пересечения (i<3)
   7.10. Определение наличия пересечения с верхней границей
   7.11. Нахождение пересечения
   7.12. Проверка пересечения на корректность; точка заносится в результат.
   //в 7.13 приходим если ни одного пересечения не найдено
   7.13. Признак видимости отрезка устанавливается Flag=-1.
8. Если Flag==0 то высвечиваем видимую часть отрезка по результату. Если Flag==-1, то отрезок является невидимым.

Недостаток алгоритма – мы ищем точку пересечения, потом проверяем на корректность; если точка некорректна то вычисления выполнялись зря.

Алгоритм Сазерленда-Коэна.

Алгоритм отсечения, основанный на разбиении отрезка сторонами отсекателя. При работе с отрезок, оный разбивается на два – на отрезок, лежащий в невидимой области, и отрезок, лежащий в видимой.



Найдя точку пересечения торезка со стороной, необходимо отбросить невидимую часть. Невидимые отрезки обычно определяются с помощью логического произведения кодов концов. Однако если мы искали точку пересечения с очередной границей, то мы её искали для случая когда она действительно существует;отрезок может закончиться раньше чем дойдёт до пересечения. Значит есть вершина, лежащая в пределах отскателя.

Чтобы легко можно было распознать невидимую часть отрезка, нужно при определении пересечения определить, какая из двух вершин невидимая. Когда точка пересечения найдена, то невидимая вершина де-факто перемещается в эту точку.

Алгоритм схемой:

1. начало //заводим "окно" с четыремя элементами для T#i - левое правое нижнее верхнее; O(Хл, Хпр, Yн, Yв)

2. ввод исходных данных P1,P2

3. формирование признака расположения отрезка: Fl=-1 вертикальный, 0 горизонтальный, 1 общего положения

4. вычисление тангенса угла наклона m (для невертикальных отрезков); m= (P2.y - P1.y)/(P2.x - P1.x)

5. цикл отсечения отрезка по четырём границам; i=1..4

6. определение T1,T2, S1,S2, произведений Pr (логическое произведение кодов концов: T11T21 + T12T22 + T13T23 + T14T24) и признака видимости Vis; Vis= -1 отрезок невидимый, 1 видимый, 0 частично видимый

7. Если

Vid=-1 :

8. Конец

Vid=1 :

9. Высвечивание отрезка; гото 8

Vid=0 :

10. Если

T1i = T2i : //в этой точке алгоритма, одноименные разряды кодов могут равняться только 0 для того чтобы алгоритм продолжался

11. Конец цикла отсечения (по i) 5; гото 9.

иначе :

12. Если T2i=1 :

13. t = P1; P1 = P2; P2 = t; //меняем вершины местами

14. Если Fl=-1 : //в данной точке алгоритма, если отрезок вертикальный, то он как минимум частично выдимый.

17. P1.y = Oi; гото A //"O"кно ^

Иначе:

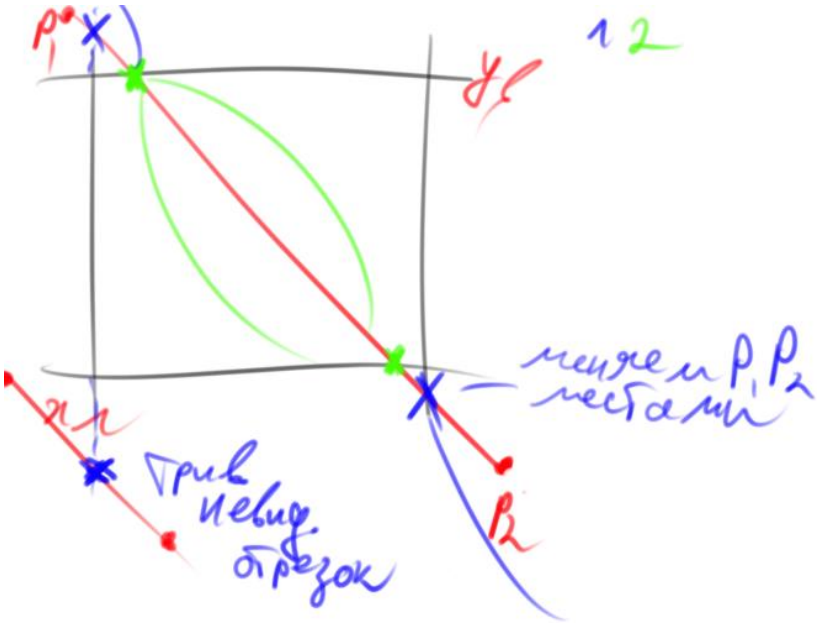
15. Если i<=2

16. P1.y = m*(Oi - Pi.x) + Pi.y; P1.x = Oi;

(A) гото 11 //к хвосту цикла - проверить нужно ли искать остальные точки, или выйти из цикла

иначе : //на 3 или 4 шагах, и отрезок невертикальный - ищем нижнюю\верхнюю границу

18. P1.x = (Oi - Pi.y)/m + P1.x; гото 17



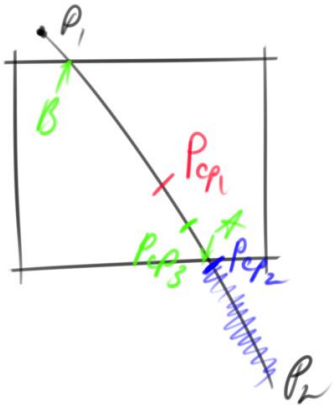
Алгоритм отсечения средней точкой

Основное отличие – точка пересечения находится не аналитически. Уравнение, фактически, решается численно – методом деления отрезка пополам. Деление на 2 – сдвиг, выполняется достаточно быстро, посему алгоритм имеет право на существование. Точность $\varepsilon = \sqrt{2}$ или $|dx| \leq 1; |dy| \leq 1$. Нахождение точек пересечения отрезка со стороной отсекателя.

23.04.14----------

Алгоритм отсечения средней точкой

Основное отличие – точка пересечения находится не аналитически. Уравнение, фактически, решается численно – методом деления отрезка пополам. Деление на 2 – сдвиг, выполняется достаточно быстро, посему алгоритм имеет право на существование. Точность $\varepsilon = \sqrt{2}$ или $|dx| \le 1; |dy| \le 1$. Нахождение точек пересечения отрезка со стороной отсекателя.



В качестве текущей всегда рассматриваем первую точку и ищем точку пересечения отрезка со стороной отсекателя. Точка пересечения А – наиболее удалённая от вершины P1, но ещё видимая. Находим середину отрезка P1P2 (Pcp1) и проверяем видимость\невидимость от-резка Pc1P2. В данном случае отрезок не является невидимым – ищем его середину Pcp2. Отрезок Pcp2P2 – невидимый, отбрасываем его; наиболее удалённая видимая от P1 точка расположена на отрезке Pcp1Pcp2. Ищем нужную точку методом половинного деления, с точностью $\varepsilon = \sqrt{2}$. Можно проверять, не стала ли одна из проекций отрезка (постоянно уменьшающегося) <=1. (возможно – нужно проверять что ОБЕ стали <=1, у нас ведь может быть ситуация когда dy=1 и dx=100 для почти параллельного отрезка..)

В общем, процесс деления повторяется пока не окажется что |P1-P2|<=eps. После нахождения А, аналогично находится Б. В данном случае, при половинном делении мы ориентируемся не на значение функции, а на приснопамятный код точки, $S = \sum(1..4)(T_i)$.

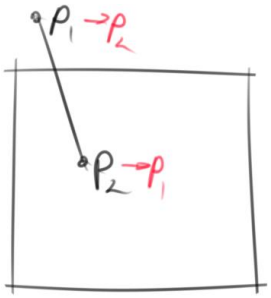
Алгоритм:

1. ввод исходных данных Хл, Хпр, Ун, Ув, P1, P2
2. вычисление кодов концов отрезка, T1 и T2
3. вычисление сумм кодов концов C1 и C2
4. проверка отрезка на полную видимость, C1=0 и C2=0; если видимый - высвечиваем и заканчиваем
5. если не полностьювидимый - проверка на невидимость; если невидимый - заканчиваем

   если же частично видимый - пошло-поехало
6. запоминание P1 (T=P1)
7. определить номер искомой точки пересечения i=1
8. если i>2, то анализируем полученный отрезок на невидимость (произведение кодов концов, если равно нулю - то видим)
9. проверка S2 = 0. Если S2=0, то Res[i] = P2;

   i+=1

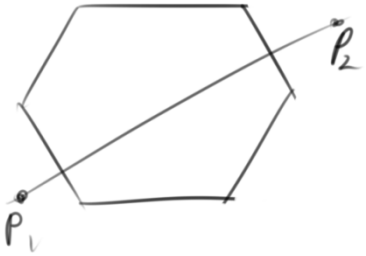
   P1 = P2; P2 = T



10. если S2 != 0 , то ищется точка пересечения:
    10.1. если |P1-P2|>E то
            ищется средняя точка Pcp = (P1+P2)/2
            Pm = P1
            P1 = Pcp
            определение невидимости P1P2, предварительно вычислить код P1
            если P1P2 невидим, то P1 = Pm (вернуть P1 на прежнее место)
                                   P2 = Pcp
            (иначе работа продолжается с P1P2)
        Иначе, если же |P1-P2|<=E, то
            Res[i] = P2
            P1 = P2
            P2 = T
            i+=1
            переход к 8.

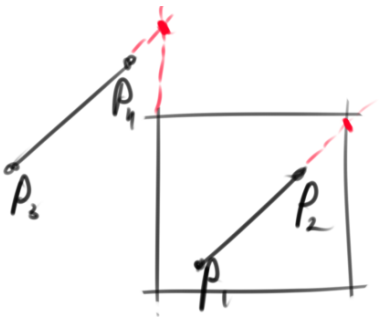
Отсечение отрезка произвольным выпуклым отсекателем

Алгоритм Кируса-Бека.



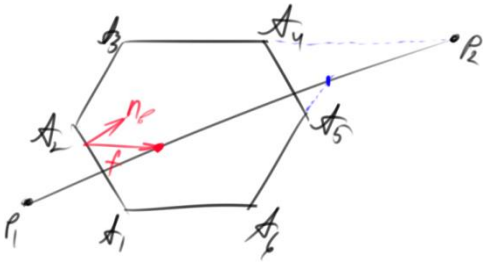
$P(t) = P1 + (P2 - P1)t,\ 0 \le t \le 1$. Здесь t – параметр для параметрической формы задания отрезка. Де факто, имеем два уравнения: y(t) = y1 +…, x(t) = x1 + ….

При попытке найти точки пересечения для полностьювидимого отрезка, можно получить точку, находящуюся за пределами отсекателя – не проходит проверка на корректность. Раньше мы видимые отрезки отсеивали, а здесь нет. Для полностью невидимого отрезка можно получить аналогичную картину.



Простыми средствами отсеять эти отрезки невозможно, если полная видимость\невидимость и обнаруживается, то только в процессе работы.

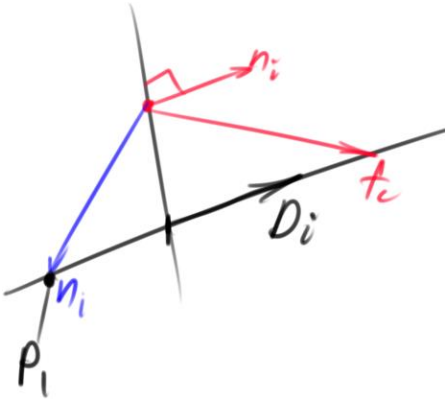
Прямая пересекает выпуклый многоугольник только в двух точках.



Используем скалярное произведение векторов – вектора нормали к стороне, и вектора от точки на ребре к точке на отрезке (красные).

$$\bar{n}_i(\overline{P1 + (P2 - P1)t - f_i} = \begin{cases} > 0, \text{точка видима}, -\frac{\pi}{2} < \theta < \frac{\pi}{2} \\ = 0, \text{точка на границе отсекателя } (\theta = \pm\frac{\pi}{2}. \\ < 0, \text{точка невидима}, \frac{\pi}{2} < \theta < \frac{3\pi}{2} \end{cases}$$ //fi - просматриваем очередную i-ю сторону. В качестве произвольной f удобно задавать вершину отсекателя.

Вектор директрисы, направления отрезка - $D = \overline{P2 - P1}$; $\bar{W}_i = \overline{P1 - f_i}$

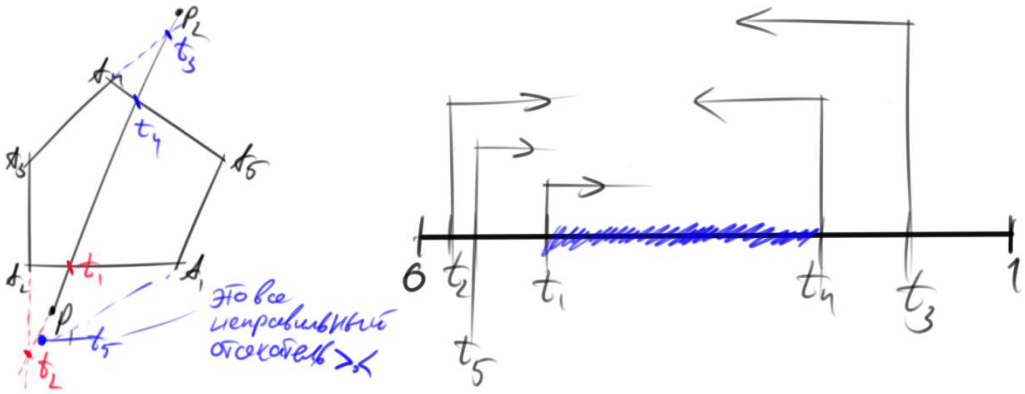


Имеем сумму скаляров: $\bar{n}_i\bar{W}_i + \bar{n}\bar{D}_i t = 0$, отсюда $t = \frac{-\bar{n}_i\bar{W}_i}{\bar{n}\bar{D}_i}$. Анализируем знаменатель на равенство 0: может быть Д=0 – отрезок вырождается в точку, которую анализируем на видимость. Знаменатель = 0, когда нормаль перпендикулярна директрисе, а значит сам отрезок паралеллен стороне отсекателя.

Видимость точки P1 определяется с помощью скалярного произведения вектора внутренней нормали n на вектор первой вершины W, $\bar{n}_i\bar{W}_i$. <0 – P1 невидима, >0 – видима, =0 – лежит на границе.

$\bar{n}_i\bar{D}_i = 0$ – отрезок параллелен стороне отсекателя, нужно проверить его видимость. Отрезок может лежать по видимую или по невидимую сторону. Сделать это можно, проверив видимость любой его точки, например P1 (спомощью nW). Если относительно текущей границы отрезок видим, то переходим к следующему шагу – ищем точки пересечения отрезка с рёбрами. Если же отрезок невидим отноеительно текущей границы, то он невидим и относительно остального многоугольника.

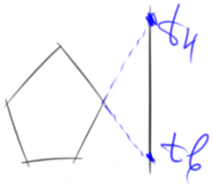
Вообще, отрезок может пересекаться с продолжениями всех сторон отсекателя, а нам надо выбрать всего две точки.



Имеем $t1 \le t \le t4$.

T1, T2, T5 – расположены ближе к началу отрезка, T3,T4 – ближе к концу. Среди точек пересечения, одни лежат ближе к началу отрезка, другие ближе к концу, достаточно очевидный факт. Из «начальных» точек надо выбрать максимальное значение – нижняя граница видимости. Из «конечных» - минимальное, нижняя граница. $t_н = \max t_i, t_i$ – точки, расположенные ближе к началу отрезка, аналогично $t_в = \min t_j$.

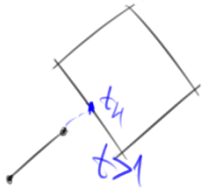
Тем не менее существует частный случай, когда отрезок не может быть распознан как невидимый.

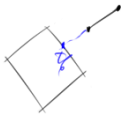


Нужна проверка на то что tн <= tв – если нет, то отрезок заведомо невидим.

1. начало
2. ввод P1,P2,n вершин, A[n] массив вершин

//анализ н-угольника на выпуклость

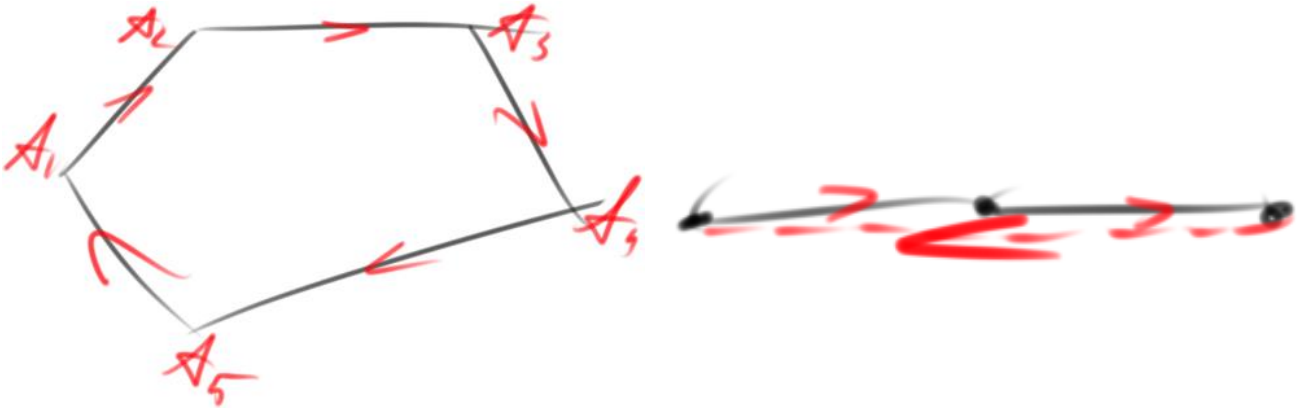
3. вычисление D = (P2-P1)\bar
4. изначально считаем что весь отрезок видимый, tн=0, tв=1
5. цикл отсечения по всем сторонам отсекателя, i=1..n
   6. вычислить нормаль n_вni; задание точки f_i (можно взять очередную вершину)
   7. вычислить вектор Wi = (P1-fi)\bar
   8. вычисление скалярных произведений Dsk=nD и Wsk=nWi
   9. если Dsk==0, то //вектор параллелен стороне
      10. если Wsk>0, то
         11. i++; continue;
      иначе //отрезок невидим
         12. конец
   иначе //можно поделить
      13. t = -Wsk / Dsk
      14. если Dsk > 0 //проверяем расположение - "видимая часть отрезка" начинается за границей пересечения 

         15. если t>1, то //точка пересечения относится к нижней границе, но при этом парамтр >1
            гото 12 //отрезок невидимый
         иначе
            16. tn = max(tn, ti);
               гото 11
      иначе
         17. если t<0, то //"видимая часть отрезка" за границей 
            гото 12 //отрезок невидимый
         иначе
            18. tв = min(tв, ti)
               гото 11
19. если tH <= tB, то
   20. отрисовать отрезок P(tн) до P(tв)
иначе
   гото 12 //отрезок невидимый

30.04.14----------

Определение факта выпуклости многоугольника (в алгоритме Кируса-Бека):

Первый способ основан на вычислении векторных произведений.



Стороны многоугольника рассматриваются как векторы; в каждой вершине вычисляется векторное произведение смежных векторов и анализируется его знак (направление результирующего вектора).

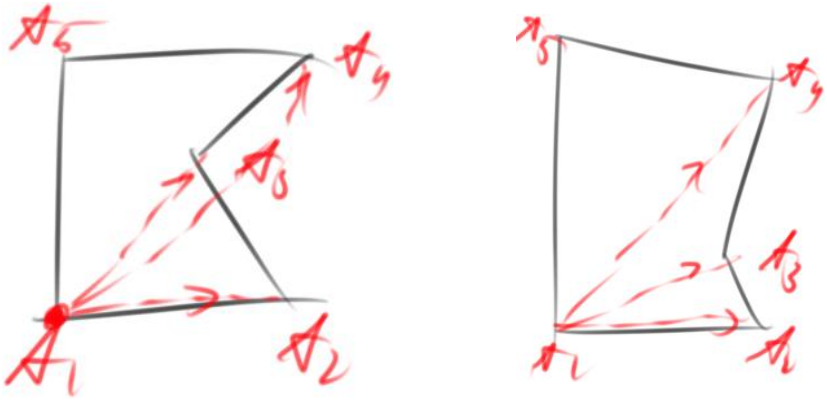
1. Если все значения нулевые, то многоугольник вырожденный.

2. Если все знаки неположительные, то многоугольник выпуклый; однако какие-то два вектора лежат на одной прямой. Его внутренняя область лежит «справа от направления обхода»; можно определить вектор внутренней нормали.

3. Если все знаки неотрицательные, то -''-. Внуренняя область расположена слева от направления обхода.

4. Если есть противоположные знаки, то многоугольник невыпуклый – алгоритм Кируса-Бека применять нельзя; надо либо разбить либо дополнить до выпуклого.

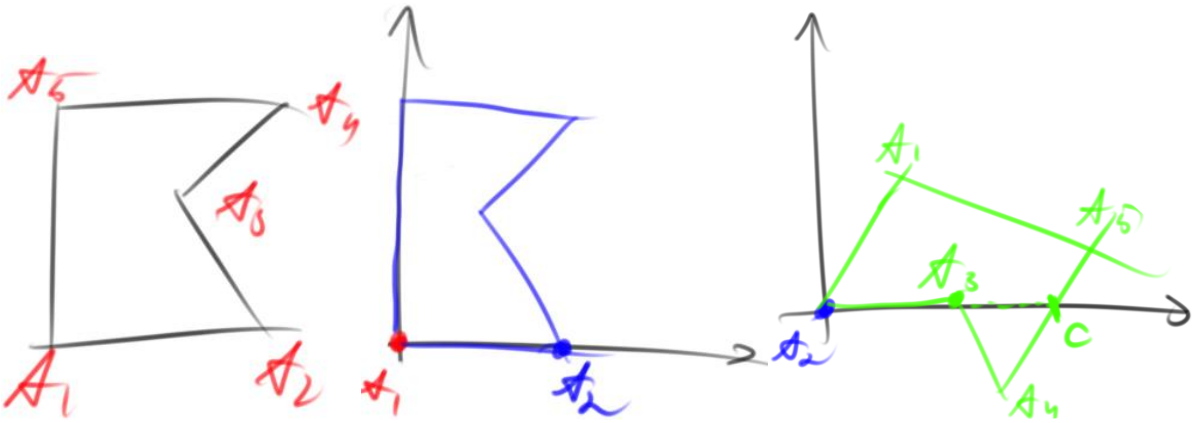
Второй способ – одна вершина берётся в качестве базовой. Количество вычислений может оказаться меньше ^, или больше.



Выбирается базовая вершина и рассматриваются векторы из неё в остальные. A1A2 * A1A3 > 0. A1A3 * A1A4 <0. Однако во втором случае векторные произведения из A1 ВСЕ положительные – в худшем случае понадобится последовательно проверять все вершины, факториальная сложность.

Третий способ – наиболее общий, но он достаточно громоздкий. Используются переносы и повороты. Тем не менее, позволяет найти ответ аж на три задачи. Если многоугольник не является выпулкым, то его надо разбить на выпуклые части.

Очередная вершина – iя. Следующая вершина – i+1я. Все остальные вершины условно называем i+2ми.



1. осуществляется перенос всего многоугольника так, чтобы i-я вершина располагалась в начале координат.

2. выполняется поворот вокруг НК так, чтобы i+1я вершина оказалась бы на положительной полуоси X.

3. вычислить значение ординат всех i+2х вершин.

4.1 если все знаки положительные, то многоугольник выпуклый относительно текущей стороны – что не значит быть в принципе выпуклым. I-й вершиной выбирается следующая; процесс повторяется.

4.2 Если знаки разные, то многоугольник невыпуклый.

5. найти точку пересечения многоугольника (ближашую к началу координат) с осью абсцисс. В один многоугольник попадут вершины, начиная с i+1й до найденной точки пересечения (^ A3 A4 C). Во второй многоугольник попадут все вершины, не попавшие в первый: C A5 A1 A2; второй снова анализируется на выпуклость.

6. Определяем нормаль – знак Y проекции совпадает со знаком любой из i+2х вершин.

ax * nx + ay * ny = 0; $n_x = -\frac{a_y}{a_x} n_y,\ n_y = 1,\ a_x \neq 0$

Если ay = 0 (отрезок горизонтальный), то ny = 1, nx=0

Если ax = 0 (отрезок вертикальный), то ny=0, nx=1

ax и ay – проекции.

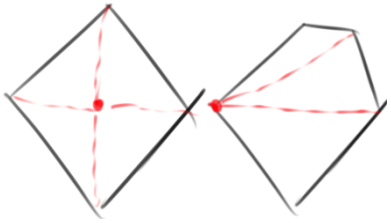
Наконец, в выпуклых многоугольниках легко првоерить внутренность нормали – соединив произвольную точку ребра с любой другой вершиной. $\bar{n}_i * \overline{\overline{(f_i A_j)}} > 0$, тогда нормаль внутренняя, иначе внешняя (ппроекции надо умножить на -1).

Триангуляция

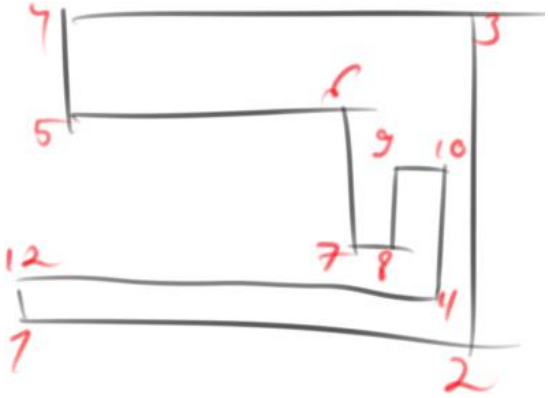
Легко триангулировать выпуклый многоугольник.

А) произвольная точка внутри многоугольника соединяется с вершинами.

Б) Другой способ – взять одну из вершин базовой, соединить с остальными вершинами многоугольника



Хуже обстоит дело с невыпуклым многоугольником.

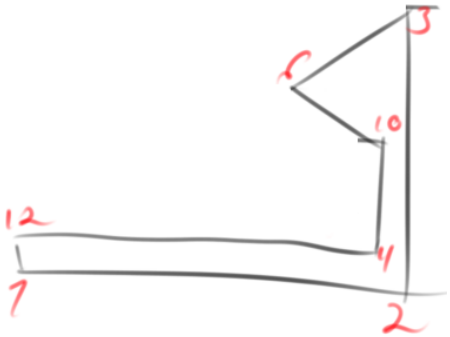


1. найти невыпуклую вершину – в которой векторное произведение смежных векторов будет отрицательным (обход против часовой). Для этой вершины ищем такую диагональ, которая бы лежала внутри многоугольника и пересекала только стороны, смежные с вершиной из которой исходит.

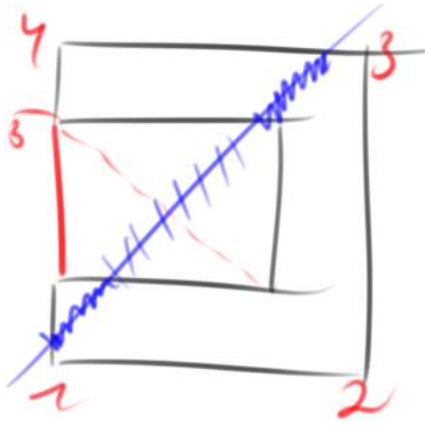
^ первая невыпуклая – А6. Если провести диагональ 6-8, то она лежит внутри многоугольника и не пересекает другие стороны прямоугольника, несмежные с 6.

Чтобы уменьшить количество анализируемых вершин, продолжаем анализ с текущей вершины (*рассматривать начинае 6, для которой была найдена диагональ*).

Далее соединяем 6-9, получаем снова треугольник. 6-10 – снова треугольник. Дальше диагонали не применимы вплоть до 6-3 – верхний остаётся выпуклым, разбиваем нижнюю часть.



Анализируем А10. 10-12 и 10-11 нельзя, можно 10-2. Процесс повторяется...

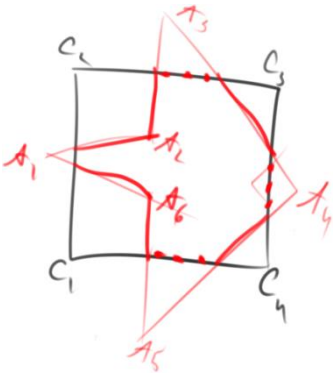


Дополнение многоугольника до выпуклого – находим первую невыпуклую вершину (A5), начинаем проводить диагонали до других вершин, пока не получим выпуклый многоугольник.

Выполняется внутреннее и внешнее отсечение. Внешнее отсечение здесь выполняется по границам дополняющих многоугольников (^ - *отсекаем по большому прямоугольнику; затем внешне по 5-7-8 треугольнику, потом внешне по 5-6-7).*

В общем случае, оптимальнее использовать идеи из алгоритмов

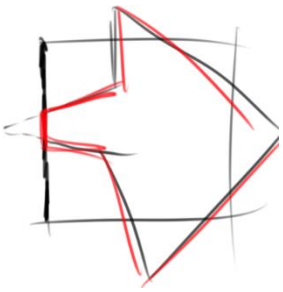
Специальные алгоритмы отсечения



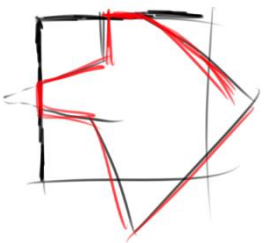
Необходимо дополнять результат отсечения соответствующими частями рёбер отсекателя.

Алгоритм Сазерленда – Ходжмена. Отсечение произвольного многоугольника (без отверстий) ВЫПУКЛЫМ отсекателем. Отсечение выполняется последовательно каждой стороной отсекателя. Результат, полученный на очередном шаге, рассматривается как исходный для следующего шага.

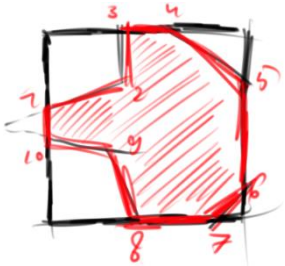
Вершина A1 невидима – результат не попадает. A2 видима, надо найти точку пересечения и занести в результат. 2-3 видма (относительно РЕБРА C1C2), как и A3,A4,A5,A6. 6-1 частично видима – находим точку пересечения. Отсекаем по левой стороне.



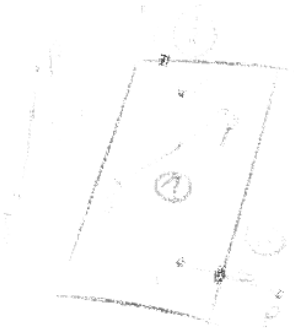
Рассматриваем теперь C2-C3 – A3 невидима, повторяем процесс.



Повторяем ещё дважды – получаем многоугольник, в котором вершин больше чем в исходном.



Возможные варианты взаимного расположения.



1 обе вершины отрезка S P видимы, отрезок целиком видим. В результат заносятся точки С и П

2 Обе вершины невидимы, отрезок целиком невидим. В результат заносится 0 точек

3 S невидима, P видима, отрезок частично видим. В результат заносится точка пересечения (надо найти) и П.

4 Наоборот. С и точка пересечения.

05.07.14----------

Алгоритм Сазерленда – Ходжмена. Возможные варианты взаимного расположения.

1 обе вершины отрезка S P видимы, отрезок целиком видим. В результат заносятся точки С и П

2 Обе вершины невидимы, отрезок целиком невидим. В результат заносится 0 точек

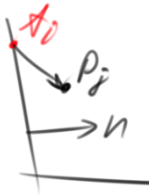
3 S невидима, P видима, отрезок частично видим. В результат заносится точка пересечения (надо найти) и П.

4 Наоборот. С и точка пересечения.

Алгоритм:

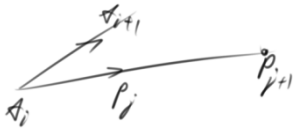
1. определить видимость точки (вершины рёбер)

а) использовать скалярное произведение PjAi * n, >=0 точка видима, <0 невидима

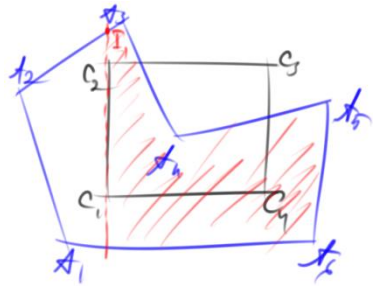


б) использовать пробную функцию (на основе уравнения прямой) F =ax + by + c

в) использование векторного произведения AiPj * AiAi+1, >=0 точка Pj видима



2. находить точки пересечения



C – отсекатель, A – многоугольник.

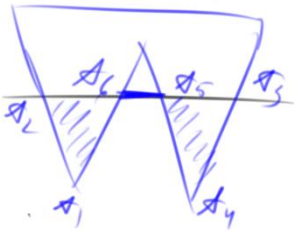
В данном алгоритме рассматривается конкретный вариант нахождения: ищется точка пересечения прямой, проходящей через ребро отсекателя, с ребром отсекаемого многоугольника. Поскольку отрезки имеют произвольное расположение, удобно использовать параметрическую форму записи: P(t) = P1 + (P2-P1)t, где 0<=t<=1 – ребро многоугольника, и Q(s) = Q1 + (Q2-Q1)t – ребро отсекателя.

P(t)=Q(s): имеем систему $\begin{cases} P1.x + (P2.x - P1.x) * t = Q1.x + (Q2.x - Q1.x)s; \\ P1.y + (P2.y - P1.y) * t = Q1.y + (Q2.y - Q1.y)s; \end{cases}$. Предварительно нужно убедиться в непараллельности прямых – точка пересечения должна существовать. Это определяется с помощью видимости концов ребра многоугольника – если видимость разная, то точка пересечения есть.

Начальная вершина очередного ребра является одновременно и конечной вершиной для предыдущего ребра. Эта вершина анализируется (и заносится в результат если видима) на предыдущем шаге.

Данный алгоритм имеет недостаток – можно столкнуться с ситуацией построения «ложных ребер» (I2I3).

Мы работаем с массивом вершин, вершины обходятся последовательно – ложным будет ребро, которое обходится два раза.



Для удобства можно продублировать первую вершину в качестве n+1й для простоты организации цикла отрисовки.

Алгоритм Вейлера-Азертона

Наконец рассмотрим алгоритм, позволяющий произвольный многоугольник отсекать произвольным отсекателем. Этот алгоритм позволяет на своей базе построить один из алгоритмов удаления невидимых поверхностей.

Особенности:
И отсекаемый многоугольник и отсекатель – произвольные многоугольники. Могут быть невыпуклыми и вдобавок могут содержать отверстия.
Алгоритм позволяет выполнить как внутреннее (рисуем всё что внутри отсекателя) так и внешнее отсечение (рисуем всё что снаружи).
В результате отсечения получаются многоугольники, ребра которых являются либо ребрами исходного многоугольника, либо ребрами отсекателя – никаких новых рёбер в результате отсечения не получается.

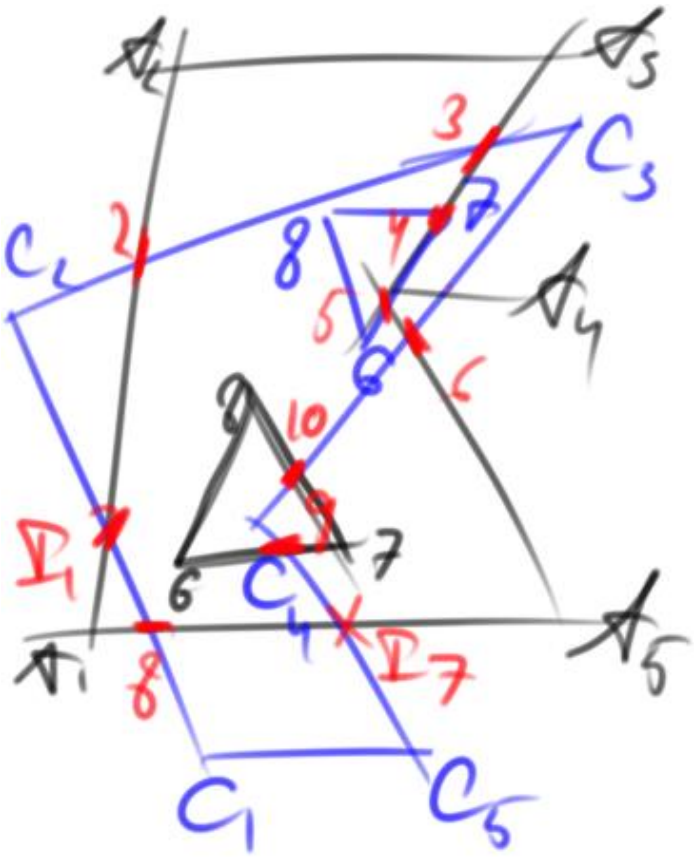
Работа алгоритма сводится к работе с двунаправленными циклическими списками. Решить задачу в общем случае достаточно просто, однако массу неудобств доставляют частные случаи. Проблемы возникают из-за того, что границу отсекателя мы относим к видимой части.

Контуры многоугольников должны задаваться определенным образом: внешняя граница КАЖДОГО многоугольника обходится по часовой стрелке, а внутренние границы – против часовой стрелки, чтобы внутренняя область всегда лежала по правую сторону от направления обхода.

Для реализации отсечения надо найти все точки пересечения границ многоугольников.

1. найти все точки пересечения рёбер отскекаемого с ребрами отсекателя

2. найденные точки разбиваются на две группы: точки входа и точки выхода. Точка входа – если ребро отсекаемого многоугольника входит внутрь отсекателя (с учётом обхода часовой стрелки) и выхода, если выходит из отсекателя. Для получения внутренних многоугольников движение надо начинать с точки входа.



Точки входа: I1 I3 I5 I7 I10

Точки выхода: I2 I4 I6 I8 I9

Построим списки, содержащие и исходные вершины и точки пересечения.

A1 I1 I2 A2 A3 I3 I4 A4 I6 I6 A5 I7 I8 A1 – циклический список обхода внешней грани многоугольника по часовой стрелке.

A6 I9 A7 I 10 A8 A6 – циклический список обхода внутренней грани многоугольника против часовой стрелки.

C1 I8 I1 C2 I2 I3 C3 I6 I10 C4 I9 I7 C5 C1 – обход внешней грани отсекателя по часовой стрелке.

C6 I5 C7 I4 C8 C6 – внутренняя грань отсекателя против часовой.

Одноименные точки пересечения в разных списках лучше соединить связями – можно будет без лишних операций переходить от одного списка к другому.

Для нахождения внутренних многоугольников движение начинаем с очередной точки входа, причём просмотр начинается со списка отсекаемого многоугольника. Просматриваемые вершины заносятся в список вершин результирующего многоугольника.

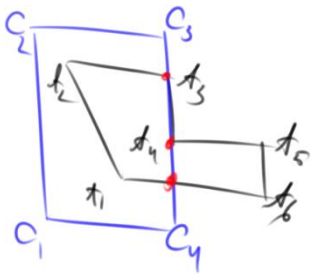
Имеем: I1 I2 (->3) I3 (->1) I4 (->4) C8 C6 I5 I6 I10 A8 A6 I9 I7 I8 I1 //указаны только первые переходы из списка в список!!

Движение заканчивается когда мы вернулись в стартовую точку (И1). Все точки входа вошли в список – внутренний многоугольник один. Если бы остались «лишние» точки входа, то процесс бы повторился для них.

Чтобы находить внешние многоугольники, движение надо начинать с очередной точки выхода. Списки границ отсекателя надо просматривать В ОБРАТНОМ направлении.

I2 A2 A3 I3 (->3) I2 ; I4 A4 I5 C6 C8 I4 ; I6 A5 I7 I9 A7 I10 I6 ; I8 A1 I1 I8

Частные случаи:



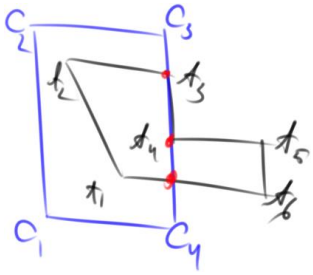
A3 – точка касания, не учитывается.

Мы ищем пересечения с границей отсекателя – однако оная относится к видимой части. Истиная граница ^ проходит на пиксель правее от ребра C3C4 – в A3 пересечения не будет.

Точки пересечения делятся на входы и выходы с помощью векторного произведения вектора стороны отсекаемого на вектор стороны отсекателя. Положительное – вход, отрицательное – выход.

14.05.14----------

Частные случаи:



А3 – точка касания, не учитывается.

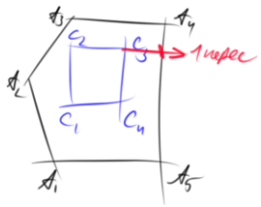
Мы ищем пересечения с границей отсекателя – однако оная относится к видимой части. Истиная граница ^ проходит на пиксель правее от ребра С3С4 – в А3 пересечения не будет.

Точки пересечения делятся на входы и выходы с помощью векторного произведения вектора стороны отсекаемого на вектор стороны отсекателя. Положительное – вход, отрицательное – выход.

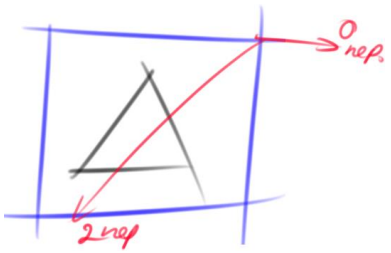
Точек пересечения может не быть.

Тест с лучом:

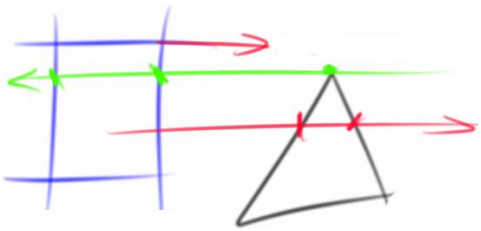
Из произвольной точки отсекателя испускается луч (например из вершины). Основной критерий – чётное или нечётное количество пересечений.



1)



2)

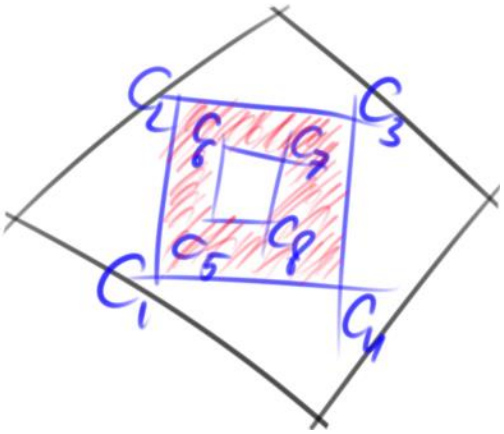


3)

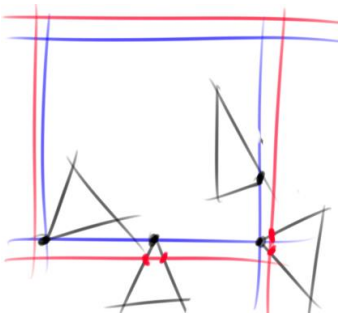
Если нечётное – отсекатель является обхватываемым. Если чётное – не обязательно обхватывающий. Нужно выпустить дополнительный луч из многоугольника в противоположную сторону.
Если второе число пересечений будет чётным – то многоугольники внешние друг к другу; в результате отсечения видимых многоугольников нет.
Если второе число пересечений нечётное – то отсекатель целиком охватывает многоугольник; в результате отсечения многоугольник полностью видим.

Если же луч проходит через вершину (касается) или же проходит сквозь всю сторону – ситуация неопределена, нужно взять другой луч (смещённый вверх или вниз).

^ в 1м случае, отсекатель расположен внутри многоугольника. У внутреннего многоугольника внешние границы совпадают с границами отсекателя. У внешнего многоугольника внешня граница совпадает с границей многоугольника; внутренняя граница совпадает с границей отсекателя; отсекатель «проделывает отверстие» в многоугольнике.



Внешняя граница отсекателя, попавшая внутрь многоугольника, является внешней границей внутреннего многоугольника, и внутренней границей внешнего многоугольника. Если граница отверстия попала внутрь многоугольника, то она проделывает отверстие во внутреннем многоугольнике.



В частном случае, когда происходит пересечение только в одной точке, отсекатель нужно расширить или уменьшить на один пиксель – точек пересечения станет несколько.

Трёхмерная графика

Используется модель трёхмерного объекта. Применяется машинное представление, дающее информацию о форме и размерах объекта: каркасное, поверхностное или объёмное.

Каркасная: проста при представлении, но не даёт полной информации о форме объекта.

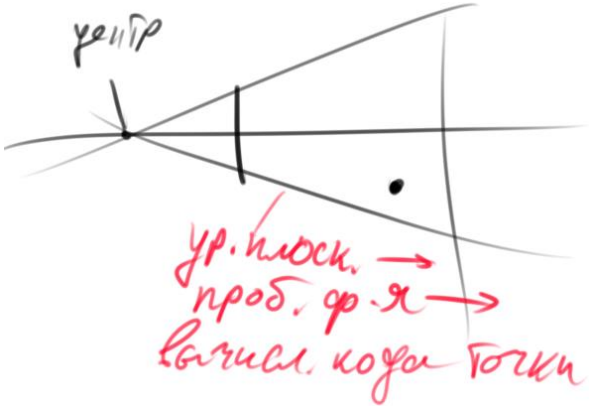
Поверхностные: не содержат информации о том, где находится материал, а где – пустота (по какую сторону поверхности).

Объёмные: добавляют к поверхности модели информацию о раположении материала. Материал обычно указывается направлением внутренней нормали.

Требования, предъявляются к математической и к программистской модели.

Преобразования в трёхмерном пространстве требуют матрицы 4x4, чтобы была возможной реализация матрицы переноса.

Трехмерное отсечение: прямоугольным параллелепипедом или усеченной 4гранной пирамидой.



Определение факта выпуклости трёхмерного тела (основанный на переносах и поворотах.

Задача удаления невидимых линий и плоскостей (граней)

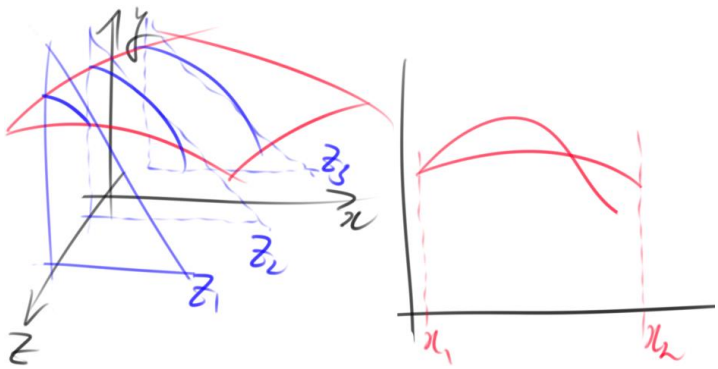
Осложняется тем, что необходимо учитывать положение наблюдателя.

В силу сложности, а так же из-за разных задач (статика\динамика), одного единого метода для всех задач не существует. Рассматривают пространство объектов (в мировой системе координат) и пространство изображения (экранная система координат).

В основу многих алгоритмов заложена сортировка объектов по «глубине» – расстоянию от наблюдателя. Трудоёмкость алгоритмов в объектном пространстве (N объектов) – N^2. Задача же в пространстве изображения (N объектов и M пикселей) – трудоёмкость N*M. Таким образом, приоритет определяется тем, чего больше – пикселей или объектов.

Алгоритм плавающего горизонта

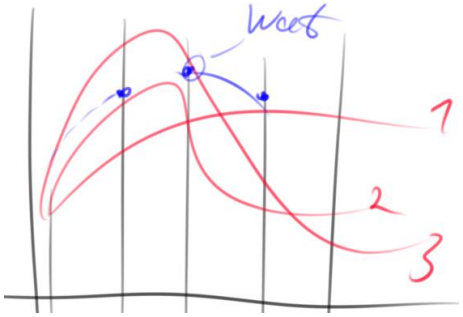
Предназначен для изображения поверхностей, задаваемых неявным уравнением F(x,y,z)=0. В общем случае считается, что наблюдатель расположен на оси Z в бесконечности. Поверхность рассекается плоскостями, параллельными OXY



Ближайшая к наблюдателю кривая видима всегда. Далее в каждом сечении имеем уравнение y = f(x, z=const). Для очередной кривой вычисляем Yj(x_i) – она будет видима, если она расположена выше самой верхней точки кривой j-1, или ниже самой нижней её точки. $y_j(x_i) > y_{\max(x_i)}$ или $y_j(x_i) < y_{min}(x_i)$. Таким образом, нижний горизонт – массив минимальных значений функции, верхний – массив максимальных значений.

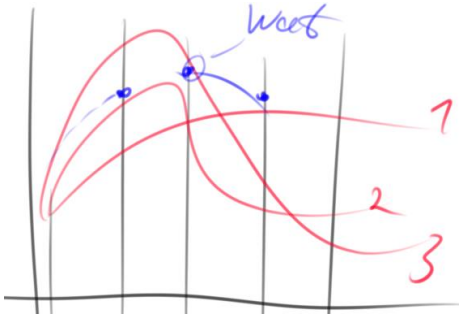
Если мы можем вычислить значение функции в каждой точке, то алгоритм достаточно просто выполняется. Очередная точка текущей кривой является видимой, если её ордината больше текущего максимума (по предыдущим кривым) или меньше текущего минимума.

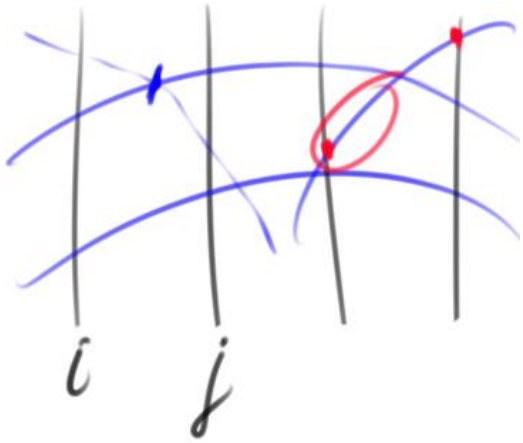
Вычисления могут производиться не с шагом=1 по oX. Если очередная точка является невидимой, то она не соединяется с предыдущей. Наоборот, если видима, то соединяется. Для упрощения вычислений можно использовать линейную аппроксимации.



21.05.14----------

Вычисления могут производиться не с шагом=1 по oX. Если очередная точка является невидимой, то она не соединяется с предыдущей. Наоборот, если видима, то соединяется. Для упрощения вычислений можно использовать линейную аппроксимации.





Требуется искать точки пересечения кривых. Для облегчения вычислений, можно использовать линейную аппроксимацию кривых.

Y_тек = Y_тек(xi) + m_тек(x-xi)

Y_пред = Y_пред(xi) + m_пред(x-xi)

M_тек = (Y_max(xj) – Y_тек(xi)) / (xj – xi)

M_пред = (Y_пред(xj) – Y_пред(xi)) / (xj – xi)

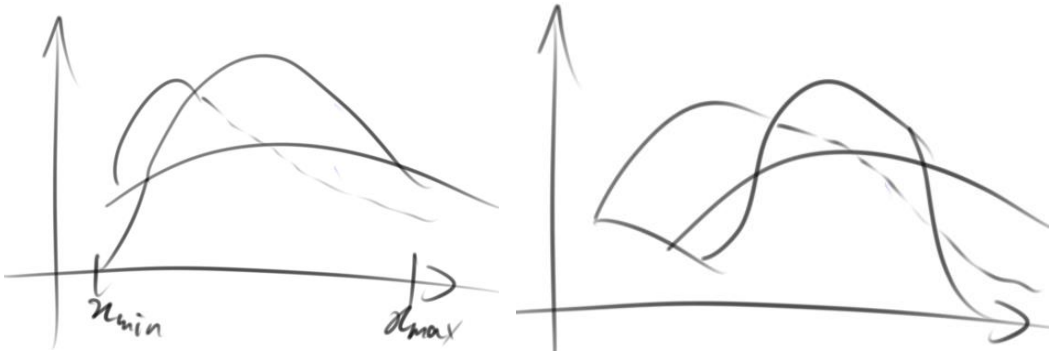
Y_тек(xi) + m_тек(x-xi) = Y_пред(xi) + m_пред(x-xi)

M – тангенс угла наклона одной прямой и другой. Текущее – для рассматриваемой кривой; предыдущая кривая - либо верхний либо нижний горизонт, в зависимости от того где расположена текущая.

Определяем x из упред и утек:

$$X = \frac{Y_{pred(xi)} - Y_{tek(xi)}}{m_{tek} - m_{pred}} + xi; \quad \Delta x = \frac{Y_{pred(xi)} - Y_{tek(xi)}}{\frac{Y_{tek(xj)} - Y_{tek(xi)}}{xj - xi} - \frac{Y_{pred(xj)} - Y_{pred(xi)}}{xj - xi}} = \frac{(Y_{pred(xi)} - Y_{tek(xi)}) * dx}{\Delta Y_{tek} - \Delta Y_{pred}},$$

dx=xj-xi



После поворота будет ->

1.Если точка P является первой на текущей кривой, то:

- если эта кривая первая, то запомнить текущую точку Q=P
- если кривая не первая, то соединить точку P с точкой Q, заменить Q=P

Последовательность действий при реализации алгоритма плавающего горизонта:

1. обработать левое боковое ребро (^)
2. для каждой точки каждой кривой выполнить:

2.1 если значение Y(xi) для этой точки больше Y_max(xi), т.е. больше значения в массиве верхнего горизонта, или меньше Y_min(xi), т.е. меньше значения в массиве нижнего горизонта

текущая точка кривой является видимой

иначе

текущая точка является невидимой

2.2 если видимость кривой на очередном интервале изменяется

найти точку пересечения текущей кривой с линией верхнего\нижнего горизонта

2.3 если на очередном интервале кривая видима

отрисовывается весь участок

если текущая точка видима, а предыдущая невидима

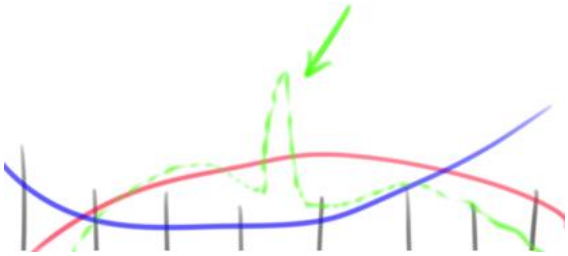
отрисовывается участок от пересечения до текущей

если текущая невидима, а предудущая видима

отрисовывается участок от предыдуще до пересечения

3. изменить содержимое текущих элементов верхнего и нижнего горизонтов (если текущая больше верхнего или меньше нижнего)
4. обработать правое боковое ребро

Выбор шага дискретизации зависит от характеристик функции (кривой) – второй производной. При большом шаге можно потерять кусок поверхности.



В целом, данный алгоритм не справляется с общим случаем задачи об отсечении невидимых частей.

Алгоритм Z-буфера

Помимо буфера кадра, используется буфер, хранящий информацию о координате Z каждого пикселя (ближайшго на текущий момент времени объекта, связанного с этим пикселем). С высокой точностью хранить Й не обязательно – достаточно лишь различать поверхности.

Идейно алгоритм прост – не требуется никаких сортировок (выигрыш по времени), а с позиции удобства программисту – объекты сцены можно обрабатывать в произвольном порядке. Тем не менее, в Й-буфере не учитывается эффект прозрачности или сглаживания.

1. проинициализировать буфер кадра фоновым значением
2. проинициализировать й-буфер минимальным значением глубины
3. преобразовать в растровую форму каждый многоугольник сцены в произвольном порядке - определение принадлежности пикселей экрана внутренней области многоугла.

Для каждой точки многоугольника (ХУ) вычислить её глубину (Й)

сравнить значения Й_буф(ху) и Й(ху). если Й(ХУ) > Й_буф(ху), то

занести в Й-буфер текущее значение Й(xy); занести в буфер кадра в соответствующую позицию цвет текущего многоугольника

4. если (Й(xy) > Й_буф(xy)) И (Й(xy) <= Й_сек.плоск) то можно построить разрез.

Помимо отсутствия сортировок, алгоритм Й-буфера позволяет построить и разрез объекта.

Уравнение плоскости: Ax+By+Cz+D=0. Для прямоугольника $Z = -\frac{Ax+By+D}{C}$, C!=0. Если C==0, то прямоугольник расположен параллельно вектору взгляда – отрисовывать нужно линию или вершину. Используется уравнение ребра; $\frac{z-z1}{z2-z1} = \frac{y-y1}{y2-y1}$, если координаты не равны. Если же равны, ребро горизонтально, отрисовывается точка.

$$\Delta x = 1;$$

$$z2 - z1 = -\frac{Ax2 + By + D}{C} + \frac{Ax1 + By + D}{C} = -\frac{A}{C}(x2 - x1) = -\frac{A}{C};\ \ \Delta z = -\frac{A}{c}$$

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

Алгоритм со списком приоритетов (алгоритм художника)

Основная идея – объекты отрисовываются «издали – вблизь», начиная с наиболее удалённых от наблюдателя, до ближайших. Основная задача: перед изображением объекта определить, может ли очередной прямоугольник загораживать прямоугольники, отображаемые в дальнейшем. Если не может загораживать другие, то он отрисовывается; если может, то проводятся расчёты.

28.05.14----------

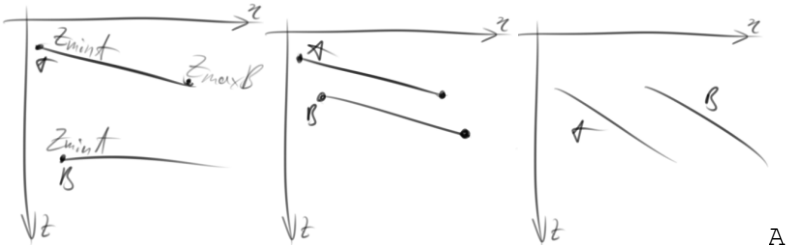
Алгоритм со списком приоритетов (алгоритм художника)

Основная идея – объекты отрисовываются «издали – вблизь», начиная с наиболее удалённых от наблюдателя, до ближайших. Основная задача: перед изображением объекта определить, может ли очередной прямоугольник загораживать прямоугольники, отображаемые в дальнейшем. Если не может загораживать другие, то он отрисовывается; если может, то проводятся расчёты.

Построение сцены начинается с многоугольников, находящихся на наибольшем расстоянии от наблюдателя – требуется отсортировать объекты сцены по возрастанию координаты Й. Если в данный момент отображается очередной многоугольник, то надо проверить, не может ли он загораживать какие-либо другие многоугольники.

1 отсортировать многоугольники сцены по возрастанию глубины (координаты Й). Первым в списке будет многоугольник на наибольшем расстоянии от наблюдателя; сортировка по Zmin многоугольника.

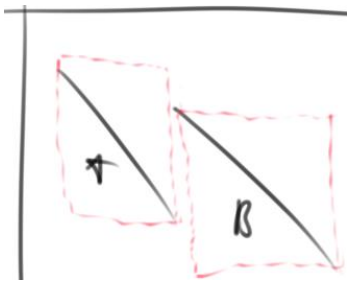
2 проверка возможности изображения очередного многоугольника на основе сравнения глубин; сравнение текущего со следующим



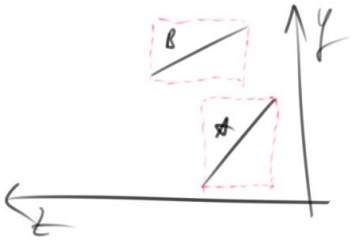
A не может экранировать B, если ЙмахА < ЙминВ

3 Проверка тестов; каждый формулируется как вопрос. Положительный – А можно отображать без проведния последующих тестов.

3.1 Верно ли, что прямоугольные объемлющие оболочки А и В не пересекаются по оси Х?



3.2 – '' –, по оси У?



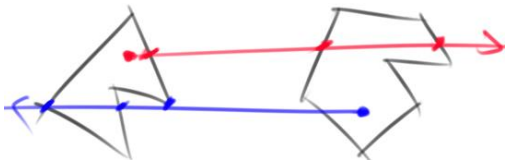
3.3 Верно ли, что А лежит по ту сторону плоскости, несущей В, которая дальше находится от наблюдателя (по невидимую сторону плоскости В)

Задается уравнение плоскости через В: Ax+By+Cz+D=0 (по координатам вершин) и формируется пробная функция f=ax+by+cz+d. Подставить координаты всех вершин А в

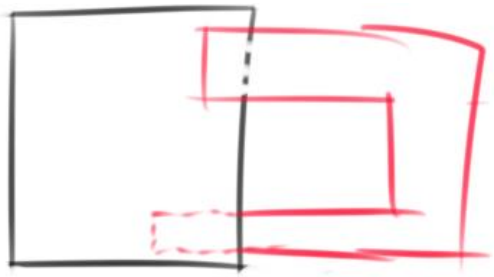
пробную функцию – одинаков ли знак (многоугольник лежит по одну сторону); сравнить знак со знаком в точке, положение которой относительно B известно.

3.4 Верно ли, что B лежит по ту сторону плоскости A, которая ближе находится к наблюдателю? //идет в дополнение к 3.3

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



Если ни на один из тестов не получен положительный ответ, то многоугольники следует поменять местами в списке и зафиксировать факт обмена. Выполнить проверки заново. Если во втором случае также не получено положительного ответа, то менять прямоугольники смысла нет; либо сами многоугольники пересекаются, либо пересекаются плоскости, их несущие. В этом случае один из прямоугольников надо разбить надвое плоскостью, несущей другой многоугольник.



Алгоритм Вейлера-Азертона

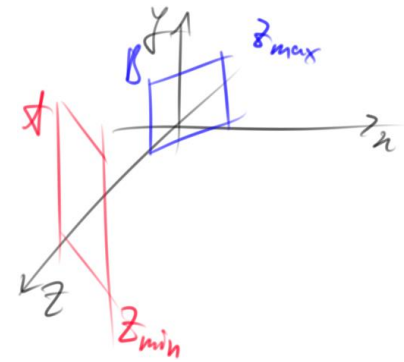
Этот алгоритм удаления невидимых плоскостей и граней целиком и полностью базируется на их алгоритме отсечения граней. Многоугольники плоские – трёхмерная задача сводится к двухмерной.

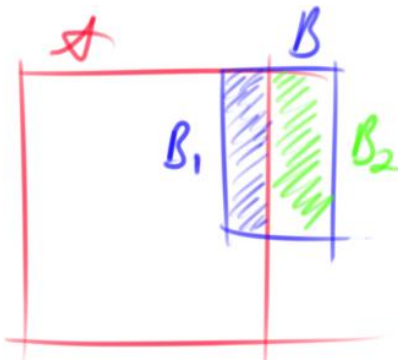
1 Отсортировать прямоугольники по глубине; ближайший загораживает остальные и используется в качестве отсекателя.

2 Отсечение многоугольников сцены по границам отсекателя («сортировка» на плоскости).

3 Удаление многоугольников, экранируемых ближайшим к наблюдателю многоугольником.

4 Рекурсивное подразбиение многоугольников (в случае необходимости) для устранения всех неопределённостей.

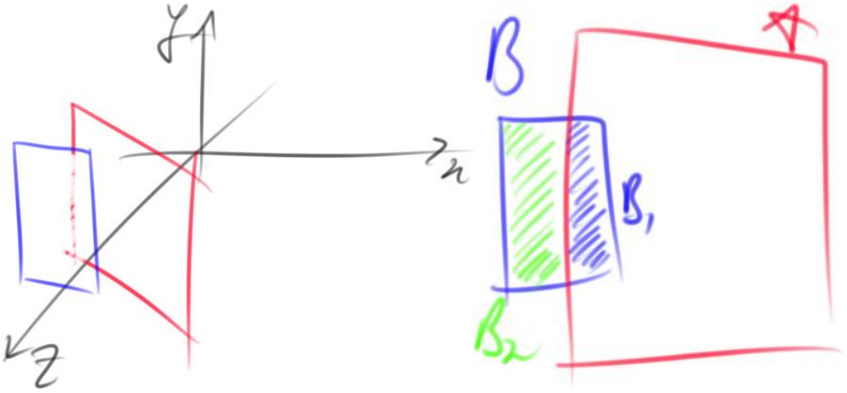




//с пункта 2 работа вдётся с проекциями многоугольника на плоскасть XOУ

Список внутренних многоугольников: A, B1; список внешних: B2

ZmaxB1 < ZMinA => изображается A. Если это условие не выполняется, то возникают неопределенности.

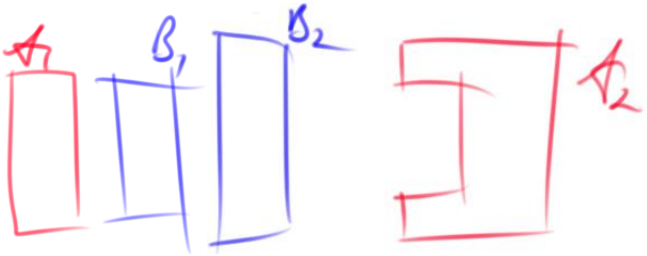


A берётся в качестве отсекателя. Список нутренних: A, B1; внешних: B2.

ZmaxB > ZminA => A изображать сразу нельзя, нужен 4 пункт.

4 Отсечение по границам многоугольника B.

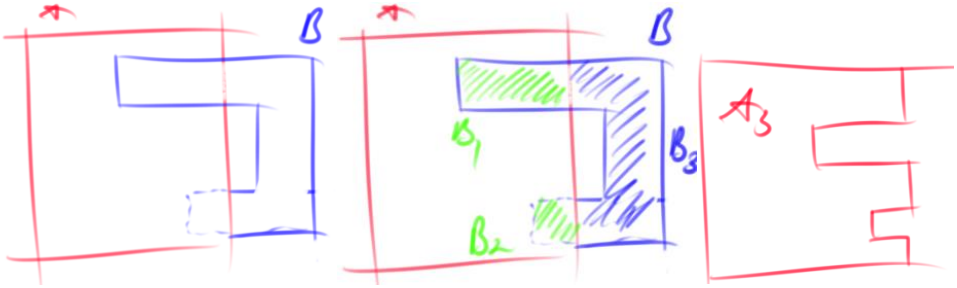
Список внутренних прямоугольников: A1, B1; внешних – A2 и B2



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

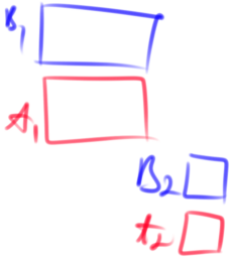
Выполнить проверку: сравнить глубины многоугольников во всех их вершинах. Z_B1i > Z_A1i, i=1..4; изображается многоугольник B1.

Алгоритм достаточно просто справляется с циклическим перекрыванием многоугольников:



Пусть ЙmaxA > ЙmaxB. Отсечение по границам A, список внутрених: , B1, B2, список внешних: B3. A3 и B3 являются внешними по отноению друг к другу, изображаются оба.

4 Отсекаем по грани B: A1 B1 A2 B2 внутренние; внешний: A3 ^



Z_B1i > Z_A1i, изображается B1. Z_A2j > Z_B2j, изображается A2

19.06.14----------

Алгоритм Робертса удаления невидимых рёбер

Работает с выпуклыми телами.

Этапы:

0. подготовка исходных данных

I. удаление рёбер, экранируемых самим телом

//работа может закончиться уже на этом этапе

II. нахождение и удаление рёбер, которые экранируются другими объектами сцены

//наиболее интересный вопрос с позиции математики

III. удаление рёбер новых рёбер (возникающих если объекты проходят друг сквозь друга), экранируемых самии телами, и другими объектами сцены

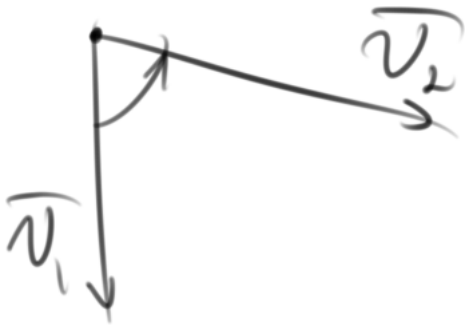
0 Подготовка данных.

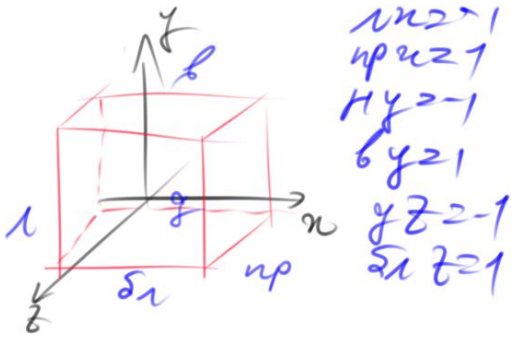
Каждое тело сцены описывается матрицей тела: 4xN – 4 параметра плоскости, n многогранников.

$$[V] = \begin{matrix} a1 & a2 & \dots & an \\ b1 & b2 & & bn \\ c1 & c2 & & cn \\ d1 & d2 & & dn \end{matrix},$$ ax+by+cz+d=0. Каждый столбец – уравнение плоскосоти, проходящей через грань. Решаем систему уравнений ax_i + by_i + cz_i + d=0, i=1..4

Уравнение можно пронормировать, поделив на d, a'x+b'y+c'z+1=0;

В тоже время, можно умножить векторно V1 x V2:





$$V = \begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{matrix}$$. Далее необходимо проверить корректность задания матрицы. Любая точка, лежащая внутри тела, должна располагаться по «положительную» сторону от любой его грани. Если это будет не так, то соответствующий столбец нужно умножить на -1. Нужно взять произвольую точку внутри тела, сформировать вектор однородных координат и умножить на матрицу; в качестве точки можно взять полусумму максимальных и минимальных координат. В нашем случае пробная точка P имеет координаты [P] = [0 0 0 1]

[P][V] = [1 -1 1 -1 1 -1] – там где -1, нужно умножить соотетствующие столбцы на -1; получим $$V = \begin{matrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}.$$

Если над телом совершаются преобразования (переносы-масштабирования), то можно заново не составлять матрицу преобразованного тела.

Пусть V – матрица исходного, VT – матрица преобразованного, T – матрица преобразований, B – матрица координат вершин тела, BT – матрица координат вершин преобразованного тела. [BT] = [B][T]; [B][V] = [D] – в Д ненулевой элемент будет с точностью до коэффициента показывать расстояние до данной плоскости.

[BT][VT] = [D] – для двух преобразований, перенос и поворот. Тогда можно [B][V] = [BT][VT];

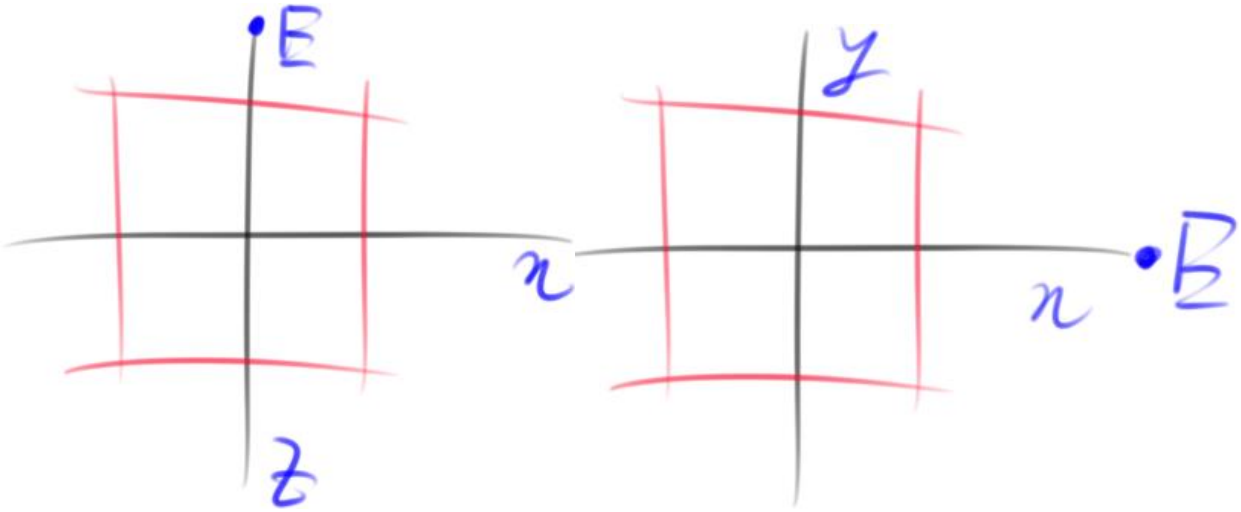
[B][V] = [B][T][VT]; умножая на обратную B, получаем [V] = [T][VT]; умножая на обратную T, получаем $[VT] = [T]^{-1}[V]$.

Подготовительный этап завершён

I Удаление рёбер, экранируемых самим телом.

Нужно определить вектор взгляда [E]=[0 0 -1 0] – наблюдатель находит на положительной оси Z и смотрит к центру. Данную запись можно рассматривать как вектор взгляда, и как однородные кординаты некоторой точки, расположенной в – бесконечности оси Z.

Любая точка внутри тела расположена по положительную сторону любой грани. Рассматривая проекции,



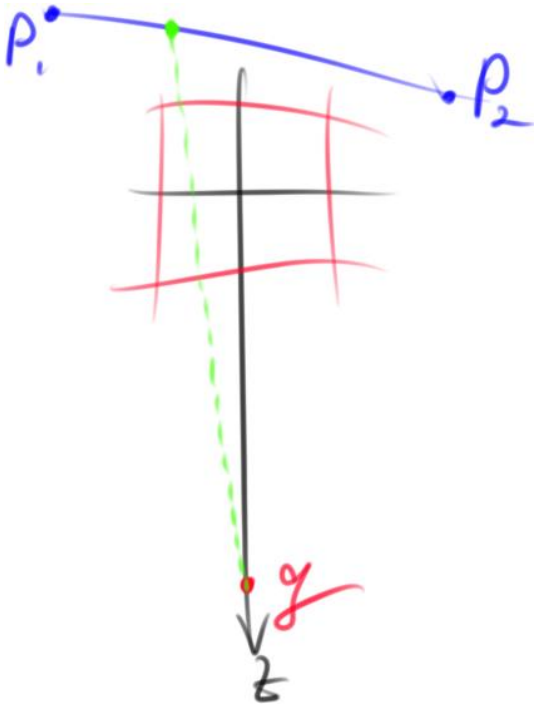
[E][V] = [0 0 0 0 **-1** 1]. Для дальней грани получили отрицательный результат, точка лежит по отрицательную сторону дальней грани – невидимой. Только для невидимых граней наблюдательная точка E будет лежать по отрицательную сторону.

Таким образом, вектор однородных координат пробной точки (вектора взгляда) нужно умножить на матрицу тела и найти отрицательные компоненты – они будут соответствовать невидимым граням. Нули можно интерпретировать: ребро является граничным случаем между видимостью и невидимостью.

В простом случае, можно вычислить внутреннюю нормаль каждой грани и посмотреть, как она ориентирована относительно вектора взгляда. Если сонаправлена – то грань видима, если противоположна – то невидима, перпендикулярна – погранченое состояние.

II Удаление рёбер, экранируемых другими телами.

Вначале на содержательном уровне сформулируем условие, при котором тело загораживает ребро. G – вектор координат точки, в которой находится наблюдатель, [g] = [0 0 1 0]. Проведём луч из произвольной точки в точку наблюдения. Если он встречает препятствие (проходит через тело), то точка невидима, а луч расположен по положительную сторону от каждой грани тела. //зелёный луч идёт параллельно, если точка расположена на бесконечности.



Проверить луч на расположенность по положительную сторону от каждой грани тела. Запишем уравнение луча через уравнение отрезка; P(t) = P1 + (P2-P1)t, 0<=t<=1 – уравнение анализируемого отрезка.

Отсюда уравнение луча Q(t,alpha) = P(t) + alpha*g = P1 + d*t + alpha*g , где d=p2-p1, и alpha>=0 (тело находится между исследуемой точкой и точкой наблюдения). По сути, Q(t,alpha) будетуравнением плоскости, проходящей через точки. Альфа служит своего рода «расстоянием» от g до точки.

[H]=[Q][V]; $h_j > 0,\ j = \overline{1,n}$. Луч должен располагаться по положительную сторону каждой грани тела; если все hj больше нуля, то это условие выполняется.

[H] = [P1][V] + t[d][V] + apha[g][V]; отсюда $h_j = p_j + g_j t + \alpha w_j,\ h_j > 0$.

Здесь [Q] = [d][V]; [W] = [g][V].

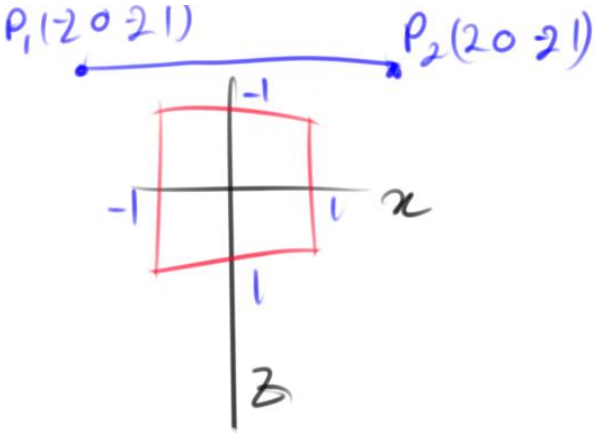
Имеем систему неравенств в общем случае: $\begin{matrix} a11x1 + a12x2 + \cdots + a1nxn > b1 \\ a21x1 + a22x2 + \cdots + a2nxn > b2 \\ \dots \\ am1x1 + am2x2 + \cdots + amnxn > bm \end{matrix}$, m ограничений, n неизвестных.

В нашем случае, n ограничений, 2 неизвестных.

Зададим целевую функцию: $\sum_{i=1}^{n} c1x1 \to min$. Целевая функция линейна, производная (которая может применяться для поиска минимума) равна нулю. Здесь C – некоторые коэффициенты. Стационарных точек здесь нет, минимумы надо искать на границах области. От ^ неравенств надо перейти к равенствам.

В данной ситуации функция, задающая ограничение, представляет собой прямую, определяемую переменными t,alpha; условия – неравенства. Точки, лежащие на прямой, обращают запись в равенство, точки лежащие по ту ли иную сторону прямой – в неравенства.

Рассмотрим пример.



[P1] = [-2 0 -2 1], [d] = [4 0 0 0]; [g] = [0 0 1 0];

[P1][V] = [P] = [-1 3 1 1 -1 3]

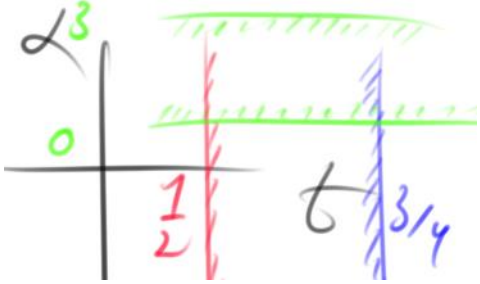
[d][V] = [Q] = [4 -4 0 0 0 0]

[g][V] = [W] = [0 0 0 0 1 -1]

Получаем упомянутые ранее ограничения $\begin{matrix} -1+4t>0 \\ 3-4t>0 \\ 1>0 \\ 1>0 \\ -1+\alpha>0 \\ 3-\alpha>0 \end{matrix}$.

1,2 – ближ-дальние грани, 3,4 – ниж-верхние грани (отрезок лежит между нижней и верхней гранями), 5,6 – лев-правые грани

От неравенств перейдём к равенствам – получим уравнения границ. В этой прямоугольной области все точки удовлетворяют неравенству и являются невидимыми.



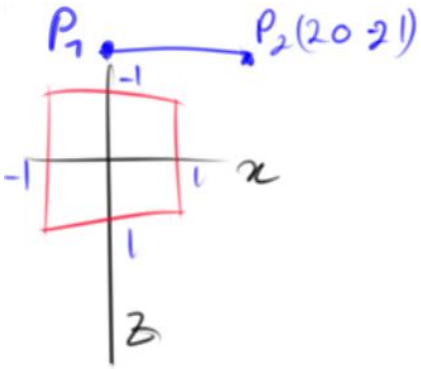
Целевая функция линейна – решения находим на найденных границах. $t \to min$. Выбираем t=const и сдвигаем её, пока не выйдем за область допустимых решений, $t_{min} = \frac{1}{4}$. Аналогично с другой границей $t_{max} = \frac{3}{4}$. Обозначим точками P3 и P4 границы невидимой области отрезка:

P3 = (-2 0 -2 1) + ¼(4 0 0 0) = (-1 0 -2 1)

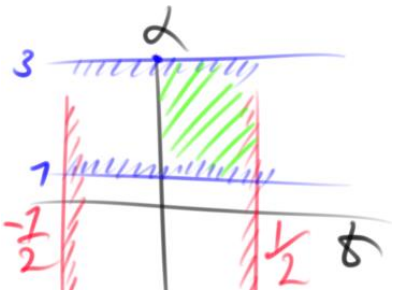
P4 = (-2 0 -2 1) + ¾(4 0 0 0) = (1 0 -2 1).

В общем случае, может понадобиться учитывать ограничения параметров: t[0..1], alpha>0.

Начало исходного отрезка передвинем в середину: [P1]=[0 0 -2 1], [d]=[2 0 0 0].



Получаем [P] = [1 1 1 1 -1 3]; [Q] = [2 -2 0 0 0 0]; [W] = [0 0 0 0 1 -1].

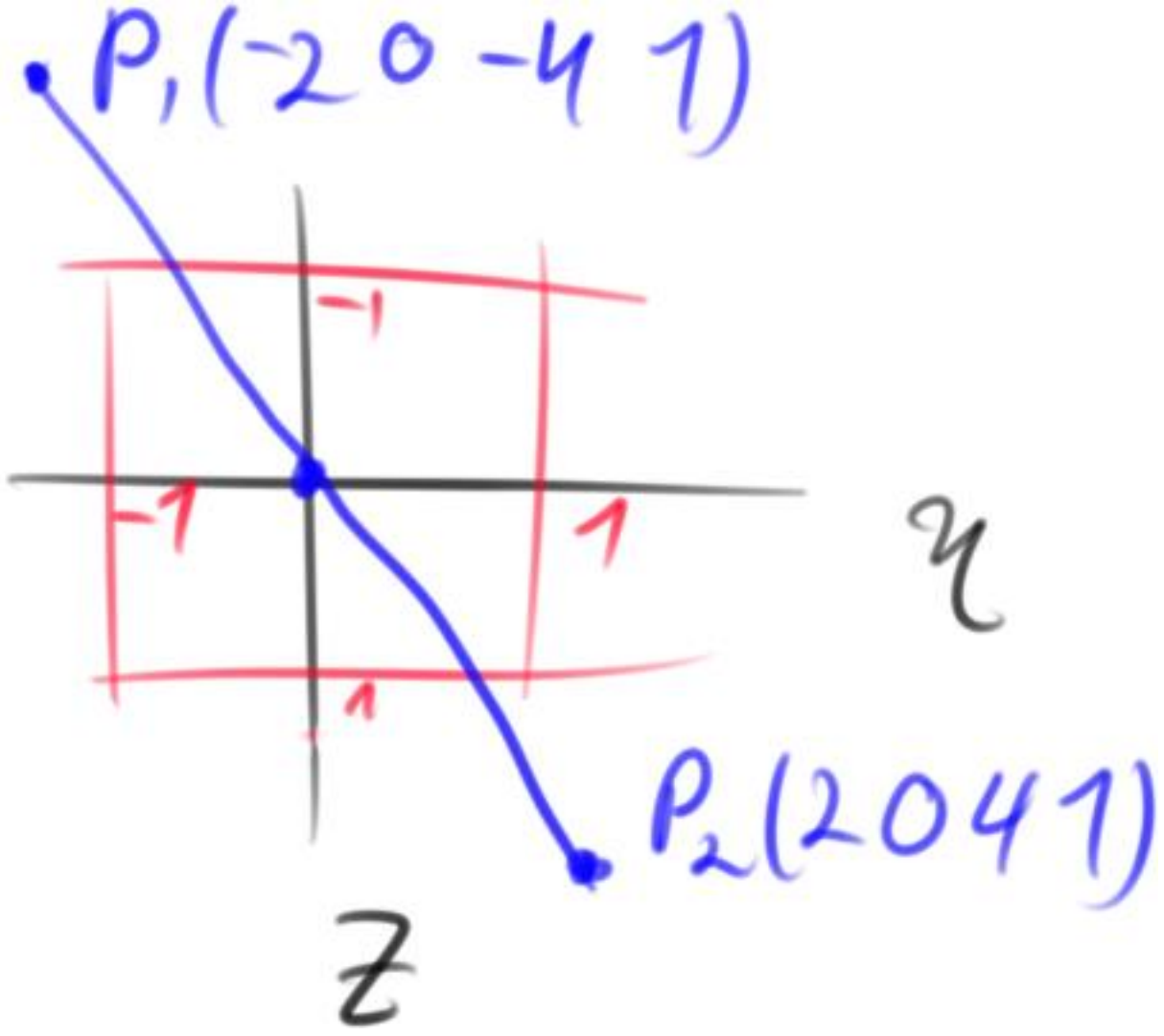


Отсюда $\begin{matrix} 1+2t>0 \\ 1-2t>0 \\ 1>0 \\ 1>0 \\ -1+\alpha>0 \\ 3-\alpha>0 \end{matrix}$;

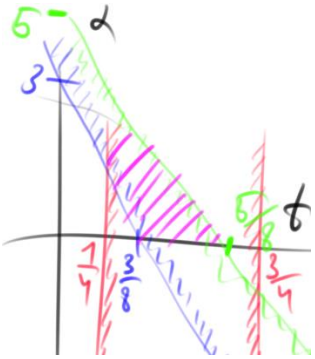
Ограничение параметра альфа нужно в том случае, если отрезок проходит сквозь объект.

III.1 нахождение точек протыкания

Нас интересует видимые части рёбер – если связать ребром две точки протыкания, видимым оно не будет.




```
[p1] = [-2 0 -4 1];  [d] = [4 0 8 0];  [g] = [0 0 1 0]
[P] = [-1 3 1 1 -3 5]
[Q] = [4 -4 0 0 8 -8]
[W] = [0 0 0 0 1 -1]
```



Имеем $\begin{matrix} -1+4t>0 \\ 3-4t>0 \\ 1>0 \\ 1>0 \\ -3+8t+\alpha>0 \\ 5-8t-\alpha>0 \end{matrix}$.

Отсюда $t_{min}=\frac{1}{4}, t_{max}=\frac{5}{8}$.

Точки протыкания – пересечения прямой с телом, при alpha!=0 мы бы получли точку на другой прямой. В этом случае получим $t_{min}=\frac{1}{4}$ – начало невидимой части, $t_{max}=\frac{3}{8}$ – конец невидимой части.

Замечания.

На ЭВМ используется перебор: n уравнений, 3 ограничения; n+3 уравнения для 2 неизвестных. Составляются все возможные комбинации систем по 2 уравнения. Составили очередную комбинацию, решили – убеждаемся, что полученное решение является решением оставшимся уравнений, необходимо подставить. Если полученное решение является решением всей системы, то оно рассматривается как канидат на поиск tmin и tmax.

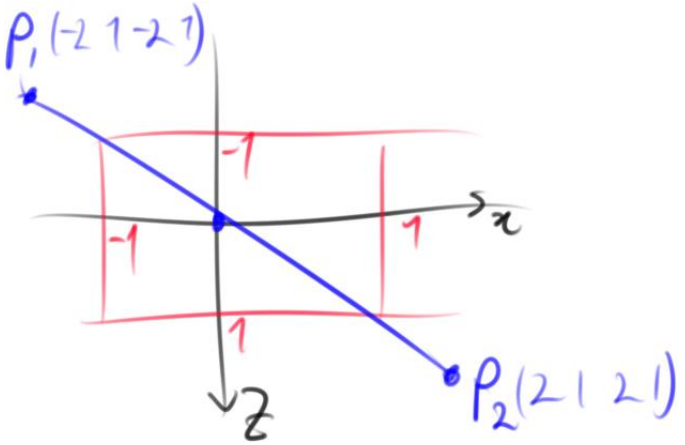
По формуле арифметической прогрессии: n+2 системы для первого уравнения, 1 для последнего; $S_n=\frac{a1+an}{2}*n=\frac{n+2+1}{2}*(n+2)=\frac{(n+3)(n+2)}{2}$.

Сокращение трудоёмкости

Даная процедура достаточно трудоёмка.

Можно попробовать определить заведомо-невидимые рёбра – обе вершины лежат позади(на) невидимой грани. Тем не менее, грань конечна, а плоскость бесконечна – уравнение плоскости использовать нельзя.

Можно попытаться найти целиком видимые рёбра – обе вершины лежат на(перед) видимой грани; на плоскости или по отрицательную её сторону, при этом сама плоскость должна быть видимой. Наблюдатель также должен находиться по отрицательную сторону плоскости.



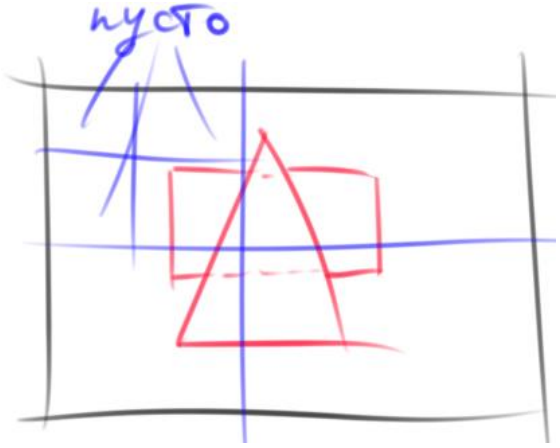

```
[p1] = [-2 1 -2 1];  [d] = [4 0 4 0];  [g] = [0 0 1 0]
[P] = [-1 3 2 0 -1 3]
[Q] = [4 -4 0 0 4 -4]
[W] = [0 0 0 0 1 -1]
```

Pj<=0 (перемножили вектор координат начальной точки на матрицу тела); pj+qj <= 0 – в этих случаях вершина видима. Если Wj<=0, то плоскость видима.

Анализируем компоненты первого вектора – нас интересуют неположительные значения. P1=-1 : <0, P1+Q1 > 0; не интересует. P2, P3>0. P4=0, P4+Q4=0; вдобавок к этому W4=0. Для четвёртой грани выполняются все условия. Это верхняя грань; отрезок расположен на верхней граи – он всегда видим.

Алгоритм Варнока

В то время как алгоритм Робертса работает в пространстве объектов, остальные алгоритмы работают в пространстве изображений.



Экран представляется в виде окна. Если в текущем окне непонятно, что изображать, то оно делится на четыре части, процесс повторяется.

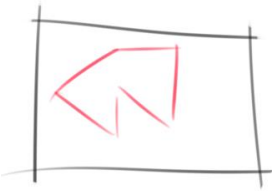
Единой версии алгоритма не существует. В простейшем варианте – окно делится на части каждый раз, если оно не пусто. Деление производится до тех пор, пока не получится окна размером 1x1 пиксель. Для окна размером в 1 пиксель мы можем однозначно сказать, что отрисовать надо элемент ближайшего к наблюдателю многоугольника (при размере 2 пикселя может возникнуть неоднозначность – многоугольники могут протыкаться). Окно является пустым, если все многоугольники являются внешними по отношению к этому окну.

В более сложных вариантах алгоритма предпринимается попытка ответа на вопрос на более ранних стадиях, не доходя до окна в 1 пиксель.

Необходимо выполнять анализ взаимных расположений многоугольника и окна.

Классификации многоугольников, рассматриваемых в алгоритме; анализ возможных случаев взаимного расположения:

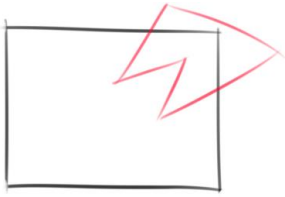
1) Один внутренний многоугольник.



а) окно закрасить цветом фона

б) выполнить растровую развёртку многоугольника

2) один пересекающий многоугольник



а) окно закрасить цветом фона

б) выполнить отсечение многоугольника по границам окна

в) растровая развёртка результата отсечения

3) все многоугольники внешние

Окно закрашивается цветом фона.

4) один охватывающий многоугольник (по размеру окна или больше)

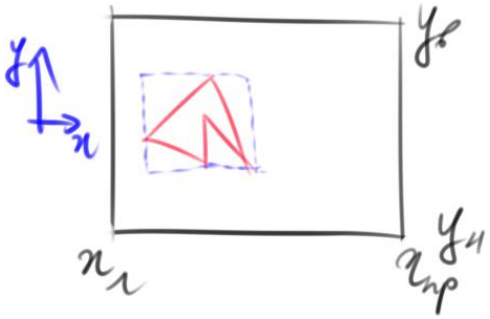
Окно закрашивается цветом многоугольника.

5) несколько многоугольников, охватывающий ближе всего

-''-

Идентификация многоугольников.

1) внутренний

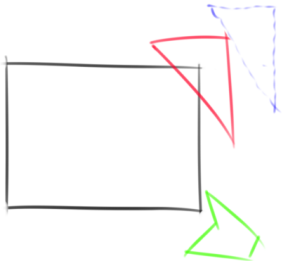


Используется объемлющая оболочка. (xmin >= xL) ^ (xmax <= xR) ^ (ymin >= yD) ^ (ymax <= yU).

2) часть внешних многоугольников

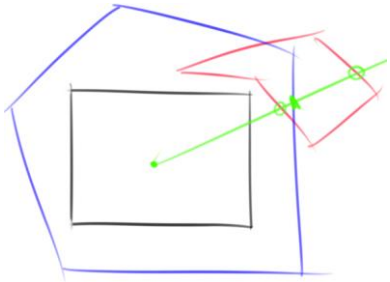
(xmax < xL) v (xmin > xR) v (ymax < yD) v (ymin > yU). Если оболочка пересекается с окном, то такая простая проверка не сработает.

3) пересекающие многоугольники



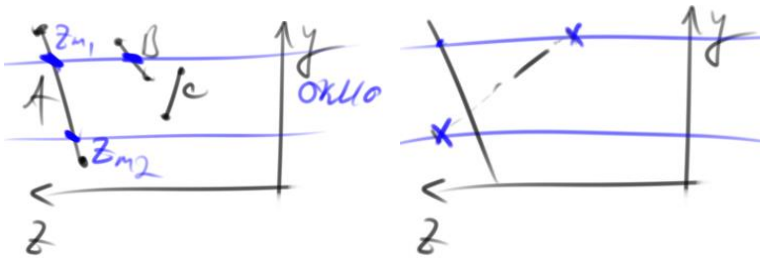
Можно использовать пробную функцию F_пр = ax + by + c; для каждой стороны анализировать многоугольники. Если для всех вершин окна знак один и тот же, то они лежат по одну сторону многоугольника, пересекающим не является (синий). В противном случае ищутся точки пересечения; определяется их «корректность», лежат они на сторонах многоугольника, или на продолжениях многоугольника.

4) часть внешних и все охватывающие



Используется тест с лучом; испускается из центра и считается количество пересечений. Чётное – многоугольник внешний, нечётное – охватывающий. Необходмо анализировать вариант, когда луч проходит через вершину внешнего прямоугольника – нужно взять на луче две точки, соседние с точкой пересечения. Если их принадлежность многоугольнику будет одинаковая, то это точка касания, если принадлежность разная – точка пересечения.

Тест проверки расположения охватывающих: достаточное условие



Вычислить глубины (Z-коорд) всех многоугольников в вершине окна. Если Zохв_i = max(j) Zij; i=1..4, j=1..m. Условие достаточное, но необходимое – может не выполняться.

Рассмотренный на лекции алгоритм Вейлера-Азертона является обобщением алгоритма Варнока. В варноке происходит много разбиений окна на подокна – форма и расположения окна не связаны с формой и расположением многоугольников. В алгоритме ВА мы берём за отсекатель один из многоугольник и отсекаем по его границам.

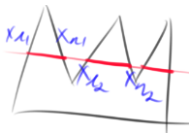
Алгоритм Z-буфера (разбираться самому).

Буфер можно выбрать размером в одну сканирующую строку. Поочередно рассматриваются сканирующие строки, для очередной строки рассматриваются все многоугольники; больше объём исходных данных.

Может использоваться аналог алгоритма заполнения с активным ребром, однако также необходимо использовать активные грани.

Задание исходных данных:

- для каждого ребра каждого многоугольника:
  - x верш; dX; dY
  - z верш; dZ_dx(-A/C) - изменение глубины при проходе по строке; dZ_dy(-B/C) - изменение глубины при проходе по разным строкам. отсюда dZ=dZ_dy*1 + dZ*dX
  - образовать пары "пределов" по x



- для каждого многоугольника:
  - вычислить ymax, ymin и количество строк

Для каждой сканирующей строки выполнить следующие действия:

1. инициализировать й-буфер размером в одну строку
2. инициализировать буфер кадра, соответствующий текущей строке, фоном цвета
3. проверить список многоугольников и добавить при необходимости очередной многоугольник в список активных многоугольников (САМ) на текущей строке
4. если многоугольник добавлен в САМ, то к списку активных рёбер (САР) добавить активные рёбра многоугольника. Для каждого активного ребра сформировать информацию: Хл, dXл, dYл; Хпр, dXпр, dYпр; dZхл, dZул, dZхп, dZуп; Zл, Zпр
5. в произвольном порядке обработать пары левых и правых рёбер в интервале Хл <= X <= Хпр вычислять глубину каждой точки z(x,y) и сравнивать с глубиной z_буф(x,y).
   - если z(x,y) > z_буф(x,y), то цвет(x,y) := цвет тек. многоугольника
6. скорректировать список активных ребер: dYл -= 1; dYпр -=1.

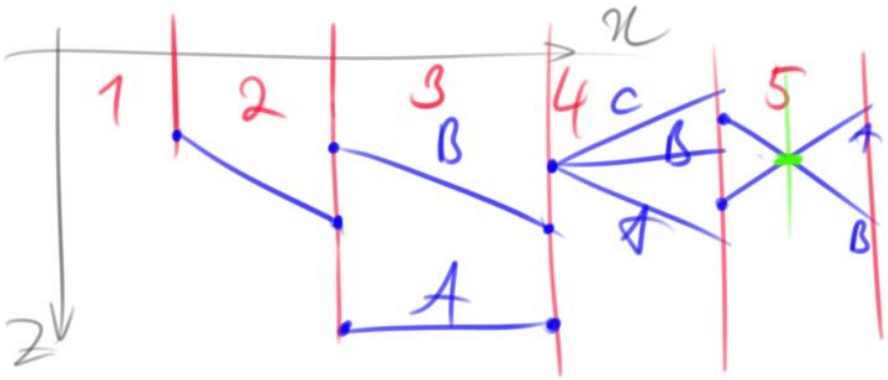
Если dYл < 0 или dYпр < 0, то удалить ребро из САР.

Если dY >=0, то Хл += дX; Хп += дX; Z += dZx*dx + dZy*dy.

Если ребро удаляется из САР, то проверить необходимость удаления многоугольника из САМ (dYмн<0). dYмн -= 1.

Если многоугольник не удаляется из САМ, а ребро удалено, то укомплектовать пару рёбер (если удалилось одно - добавить одно, если два - добавить два).

Идеи интервальных методов построчного сканирования



0 – пустой интервал

1 – один многоугольник

2 – несколько многоугольников, но один лежит ближе других. ZA1 > ZB1, ZA2 > ZB2

3 – ZA1 < ZB1 < ZC1

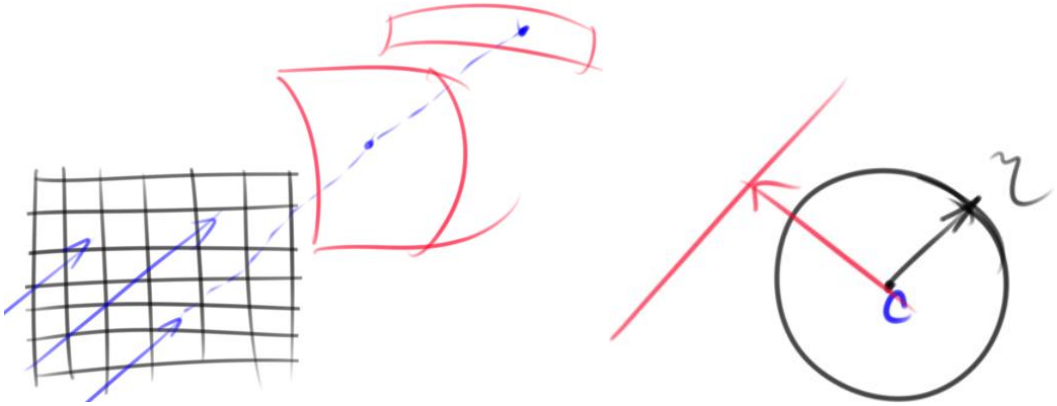
4 – ZA2 > ZB2, ZA2 > ZB2

Алгоритм определения видимых поверхностей методом трассировки лучей

Рассматриваем простейший вариант для решения задачи о видимых поверхностей. Используются принципы оптики.

Источник света испускает лучи света во все стороны, в камеру попадают отражённые лучи, благодаря которым видно поверхность. Прямая трассировка

Используют обратную трассировку – источником считается камера. Задача решается в задаче изображений – рассматриваем лучи, которые испускаются камерой и проходят через очередной пиксель экрана. Используется матрица пикселей, каждый пиксель является источником луча. Задача сводится к поиску пересечений лучей и поверхностей.



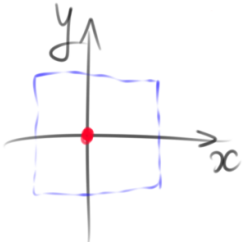
Можно использовать оболочки, хотя они и не дают однозначных результатов. Чаще всего используется сферическая оболочка.

Для луча P(t) = P1 + (P2-P1)t, t>=0. X(t) = x1 + at; Y(t) = y1 + bt; Z(t) = z1 + ct; a=x2-x1, b=y2-y1, c=z2-z1.

$d^2 = (x1 + at - x_c)^2 + (y_1 + bt - y_c)^2 + (z_1 + ct - z_c)^2$.

$\frac{dd^2}{dt} = 2(x1 + at - x_c)a + 2b(y1 + bt - y_c) + 2c(z1 + ct - z_c)$. Приравниваем производную нулю – получаем минимум, минимальное расстояние от центра до луча. Выразим t, $t_{min} = -\frac{a(x1-x_c)+b(y1-y_c)+c(z1-z_c)}{a^2+b^2+c^2}$. $d^2(t_{min}) \le r^2$ – пересечение есть.

Можно использовать и прямоугольную оболочку – но там сложнее найти точку пересечения с гранью. Для упрощения удобно выполнить преобразование, в результате которого трассируемый луч будет совмещён с осью Z



Пересечение есть, если оболочка охватывает начало координат.

Поверхность можно задать уравнением второго порядка:

$$Q(x,y,z) = a1x^2 + a2y^2 + a3z^2 + a4xy + a5 + xz + a6yz + a7x + a8y + a9z + a10 = 0$$

В результате преобразований, т.к. в точке пересечения x=y=0, получаем уравнение

$$Q'(z) = Az^2 + Bz + C = 0;\ z1,2 = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Из которого можно определить точки пересечения.

Алгоритм:

I подготовка исходных данных:

- создать список объектов, содержащий следующую информацию: описание поверхности объекта (тип объекта) и цвет; описание сферической оболочки (центр, радиус)

- при использовании прямоугольной оболочки: x,y,z_max,min

II для каждого трассирующего луча:

1. выполнить тест со сферической оболочкой. Если пересечение есть, то занести объект в список активных.

2. если САО пуст, то текущий пиксель (лучевой) высвечиваем цветом фона.

3. если САО не пуст, то находим преобразования, совмещающие трассируемый луч с осью Z и запоминаем это преобразование

4. для каждого объекта из списка активных выполнить:

4.1. если используется прямоугольная оболочка, то преобразовать её к новой системе координат. Для преобразованной оболочки выполнить тест: если пересечение есть, то заносим объект в САО

4.2. если не используется, то преобразовать очередной объект (из САО) в новую систему координат

4.3. находим пересечение луча с каждым активным объектом и заносим в список пересечений

4.4. для списка пересечений находим ближайшее к наблюдателю (Zmax)

4.5. если список пересечений пуст, отображаем пиксель фоновым цветом

4.6. если пересечение есть, то отобразить пиксель цветом ближайшего объекта (не используя модель освещения. Используя: вычислить положение точки пересечения в исходной системе координат по обратным преобразованиям; вычислить интенсивность и высветить пиксель.