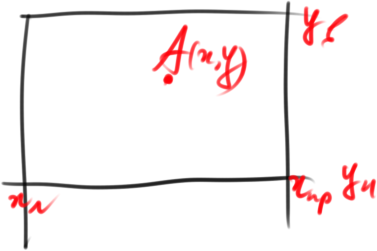
09.04.14-----------------------------------------------------------------------

**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>=xл, 1 иначе. T2=0, если х<=хпр, 1 иначе. Т3 = 0, если у>=упр, 1 иначе. Т4 = 0, если у<=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!=0 => отрезок является тривиально невидимым.

Вычисляя суммы и сравнивая с нулём, можно идентифицировать отрезок.

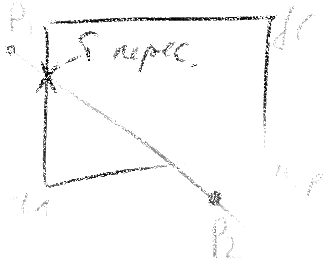
Y=kx+b, k!=infinity. .

*–* ордината точки пересечения с левой границей, – с правой.

. Соответственно, – точка спересечения с нижней границей, – с верхней. М – тангенс угла наклона.

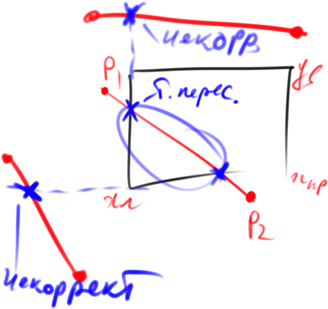
Прямая выпуклый многоугольник пересекает в двух точках (через вершину – они совпадают).

Простой алгоритм:

Точка пересечения проверяется на корректность.

16.04.14-----------------------------------------------------------------------

Простой алгоритм отсечения:



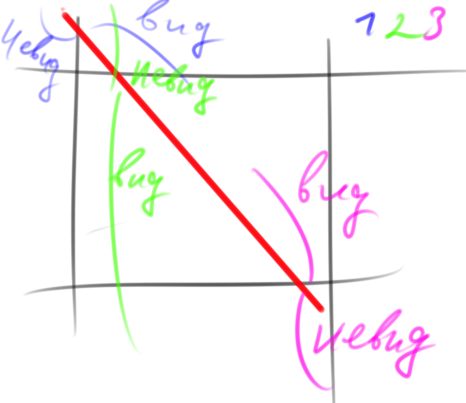
Точка пересечения проверяется на корректность.  
Изначально флаг установлен Flag=0;

1. Ввод исходных данных: хл, хпр, ун, ув; р1,р2
2. Вычисление Т1,Т2 (кодов концов отрезка)
3. Проверка полной видимости (S1=0 и S2=0; )
4. Проверка полей невидимости (  
   //к 5 пункту приходим с частично видимым отрезком
5. Проверка видимости 1й вершины: Если С1=0, то Р1 заносится в результат; i=1 (остается найти вторую вершину – будь то Р2 или точка пересечения отрезка с границей)
6. Проверка видимости 2й вершины: если С2=0, то Р2 заносится в результат (аналогично)
7. Поиск очередной точки пересечения (i-й)
   1. Определение наличия пересечения отрезка с левой границей (Pi.x < Xл)
   2. Нахождение точки пересечения (ординаты) yt= m\*(Pi.x – Xл) + Pi.y
   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
   4. Определение наличия пересечения отрезка с правой границей (Pi.x > Xпр)
   5. Нахождения точки пересечения yt = m\*(Pi.x – Xпр) + Pi.y
   6. Проверка корректности; занесение точки в результат.
   7. Определение наличия пересечения отрезка с нижней границей (Pi.y < Yн)
   8. Нахождение точки пересечения xt = (Pi.y-Yн)/m + Pi.x
   9. Проверка корректности пересечения Xл<= xt <=Xпр; занесение точки в результат при корректности; i=i+1; поиск i-го пересечения (i<3)
   10. Определение наличия пересечения с верхней границей
   11. Нахождение пересечения
   12. Проверка пересечения на корректность; точка заносится в результат.  
       //в 7.13 приходим если ни одного пересечения не найдено
   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 Т1,Т2, S1,S2, произведений Pr (логическое произведение кодов концов: Т11Т21 + Т12Т22 + Т13Т23 + Т14Т24) 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 А //"О"кно ^

Иначе:

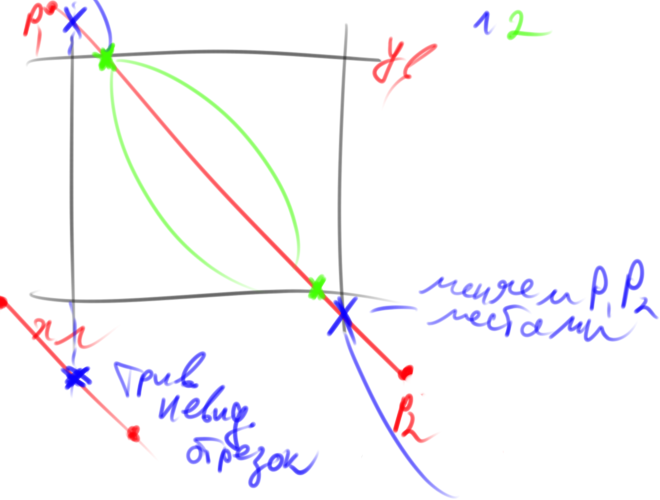
15. Если i<=2

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. Р1.х = (Oi - Pi.y)/m + P1.x; гото 17



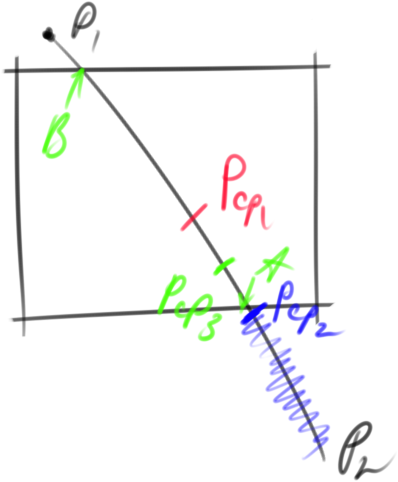
**Алгоритм отсечения средней точкой**

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 или . 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 или . Нахождение точек пересечения отрезка со стороной отсекателя.



В качестве текущей всегда рассматриваем первую точку и ищем точку пересечения отрезка со стороной отсекателя. Точка пересечения A – наиболее удалённая от вершины Р1, но ещё видимая. Находим середину отрезка Р1Р2 (Рср1) и проверяем видимость\невидимость от-резка Рс1Р2. В данном случае отрезок не является невидимым – ищем его середину Рср2. Отрезок Рср2Р2 – невидимый, отбрасываем его; наиболее удалённая видимая от Р1 точка расположена на отрезке Рср1Рср2. Ищем нужную точку методом половинного деления, с точностью . 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, .

Алгоритм:

1. ввод исходных данных Хл, Хпр, Ун, Ув, Р1, Р2

2. вычисление кодов концов отрезка, Т1 и Т2

3. вычисление сумм кодов концов С1 и С2

4. проверка отрезка на полную видимость, С1=0 и С2=0; если видимый - высвечиваем и заканчиваем

5. если не полностьювидимый - проверка на невидимость; если невидимый - заканчиваем

если же частично видимый - пошло-поехало

6. запоминание Р1 (Т=Р1)

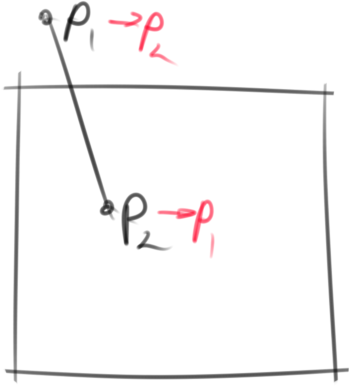
7. определить номер искомой точки пересечения i=1

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

Р1 = P2; P2 = T



10. если S2 != 0 , то ищется точка пересечения:

10.1. если |P1-P2|>E то

ищется средняя точка Рср = (Р1+Р2)/2

Pm = P1

P1 = Рср

определение невидимости Р1Р2, предварительно вычислить код Р1

если Р1Р2 невидим, то Р1 = Pm (вернуть Р1 на прежнее место)

P2 = Рср

(иначе работа продолжается с Р1Р2)

Иначе, если же |P1-P2|<=E, то

Res[i] = P2

P1 = P2

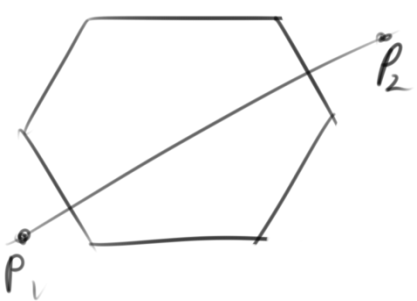
P2 = T

i+=1

переход к 8.

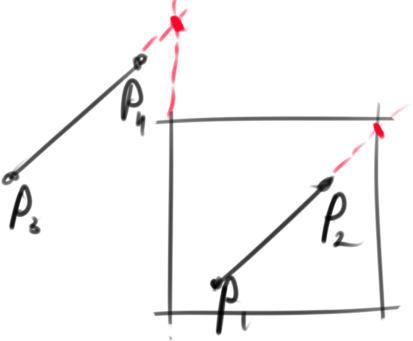
**Отсечение отрезка произвольным выпуклым отсекателем**

Алгоритм Кируса-Бека.



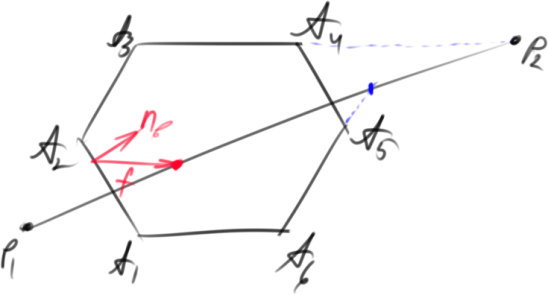
*.* Здесь t – параметр для параметрической формы задания отрезка. Де факто, имеем два уравнения: y(t) = y1 +…, x(t) = x1 + ….

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.

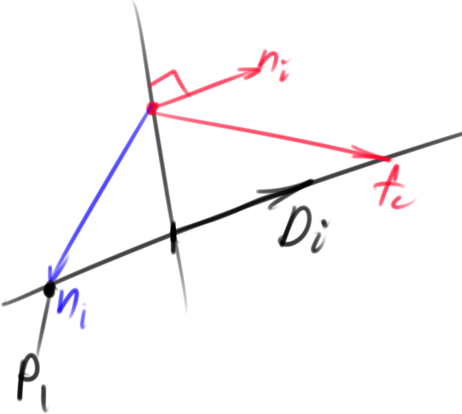
Прямая пересекает выпуклый многоугольник только в двух точках.



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

*. //fi –* просматриваем очередную i-ю сторону. В качестве произвольной f удобно задавать вершину отсекателя.

Вектор директрисы, направления отрезка - ;

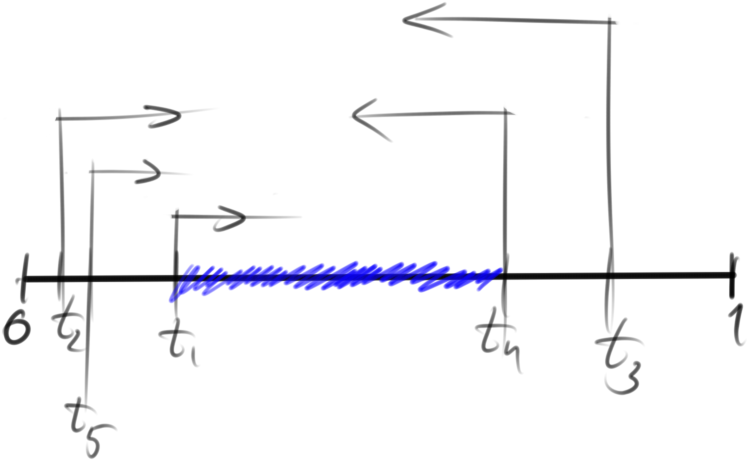
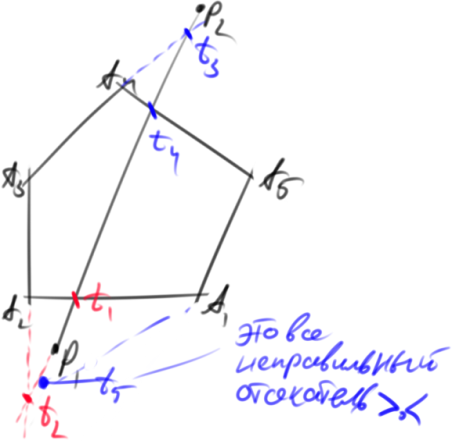


Имеем сумму скаляров: , отсюда . Анализируем знаменатель на равенство 0: может быть Д=0 – отрезок вырождается в точку, которую анализируем на видимость. Знаменатель = 0, когда нормаль перпендикулярна директрисе, а значит сам отрезок паралеллен стороне отсекателя.

Видимость точки Р1 определяется с помощью скалярного произведения вектора внутренней нормали n на вектор первой вершины W, . <0 – Р1 невидима, >0 – видима, =0 – лежит на границе.

*–* 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 Р1 (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.

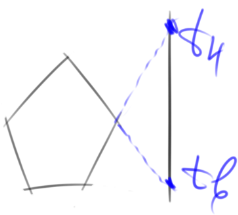
Вообще, отрезок может пересекаться с продолжениями всех сторон отсекателя, а нам надо выбрать всего две точки.



Имеем .

Т1, Т2, Т5 – расположены ближе к началу отрезка, Т3,Т4 – 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.  
 – точки, расположенные ближе к началу отрезка, аналогично .

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 Р1,Р2,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\_внi; 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

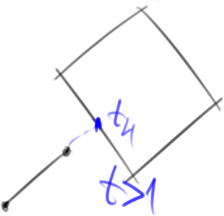
11. i++; continue;

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

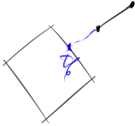
гото 12 //отрезок невидимый

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 tН <= tВ, то

20. отрисовать отрезок P(tн) до P(tв)

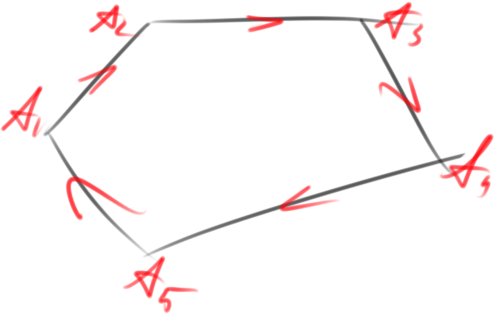
иначе

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).

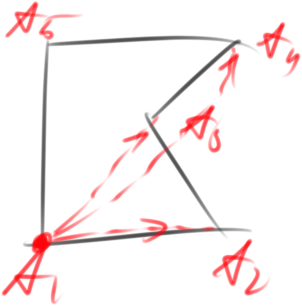
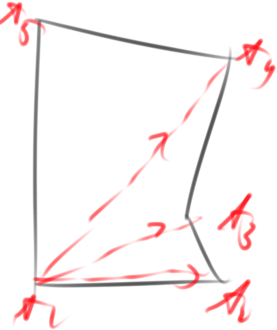
1. Если все значения нулевые, то многоугольник вырожденный.

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.

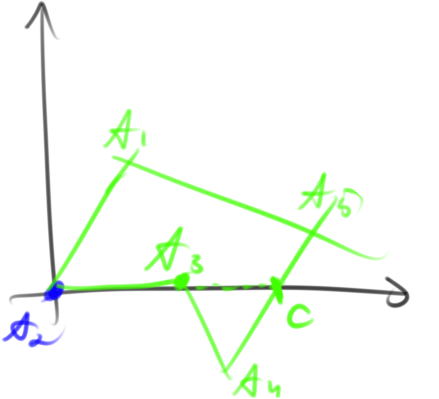
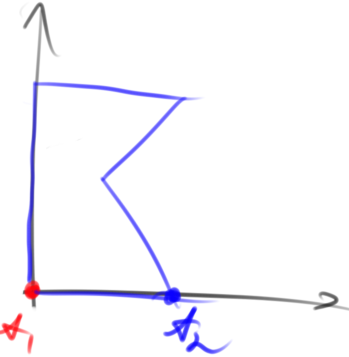
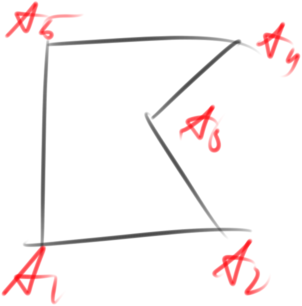
Второй способ – одна вершина берётся в качестве базовой. Количество вычислений может оказаться меньше ^, или больше.



Выбирается базовая вершина и рассматриваются векторы из неё в остальные. А1А2 \* А1А3 > 0. А1А3 \* А1А4 <0. Однако во втором случае векторные произведения из А1 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 Х.

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: С А5 А1 А2; второй снова анализируется на выпуклость.

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

 ax \* nx + ay \* ny = 0;

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

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

ах и ау – проекции.



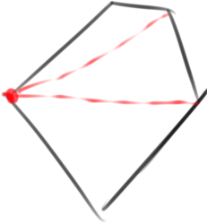
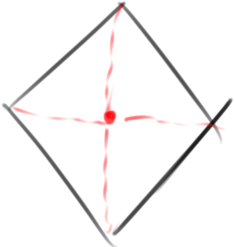
Наконец, в выпуклых многоугольниках легко првоерить внутренность нормали – соединив произвольную точку ребра с любой другой вершиной. , тогда нормаль внутренняя, иначе внешняя (ппроекции надо умножить на -1).

Триангуляция

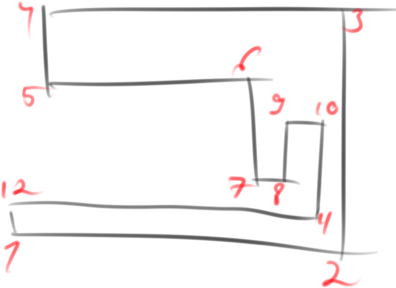
Легко триангулировать выпуклый многоугольник.

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.

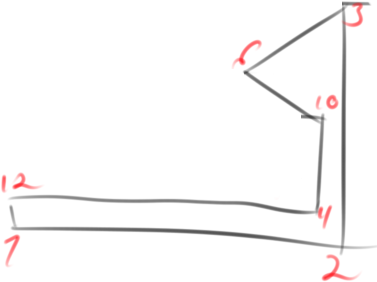


1. найти невыпуклую вершину – в которой векторное произведение смежных векторов будет отрицательным (обход против часовой). Для этой вершины ищем такую диагональ, которая бы лежала внутри многоугольника и пересекала только стороны, смежные с вершиной из которой исходит.

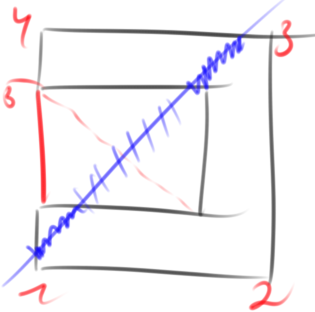
*^ первая невыпуклая – А6. Если провести диагональ 6-8, то она лежит внутри многоугольника и не пересекает другие стороны прямоугольника, несмежные с 6.*

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...*

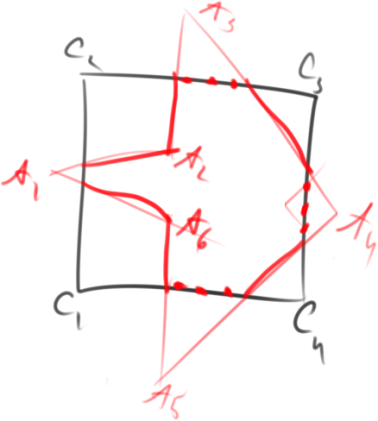


Дополнение многоугольника до выпуклого – находим первую невыпуклую вершину (А5), начинаем проводить диагонали до других вершин, пока не получим выпуклый многоугольник.

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

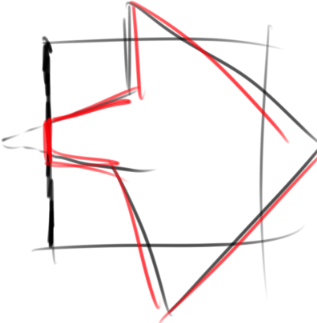
**Специальные алгоритмы отсечения**



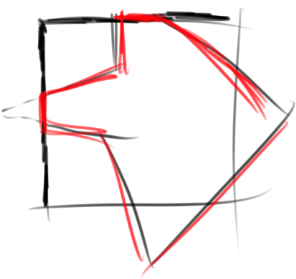
Необходимо дополнять результат отсечения соответствующими частями рёбер отсекателя.

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.

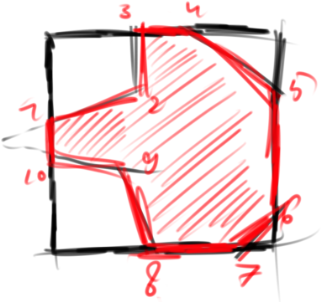
Вершина А1 невидима – результат не попадает. А2 видима, надо найти точку пересечения и занести в результат. 2-3 видма (относительно РЕБРА С1С2), как и А3,А4,А5,А6. 6-1 частично видима – находим точку пересечения. Отсекаем по левой стороне.



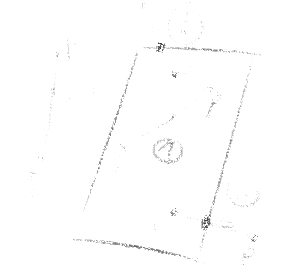
Рассматриваем теперь С2-С3 – А3 невидима, повторяем процесс.



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

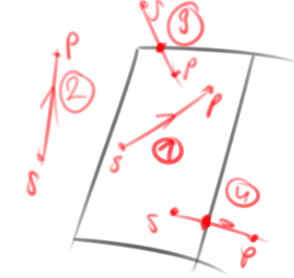
2 Обе вершины невидимы, отрезок целиком невидим. В результат заносится 0 точек

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)

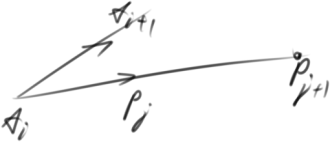
а) использовать скалярное произведение PjAi \* n, >=0 точка видима, <0 невидима



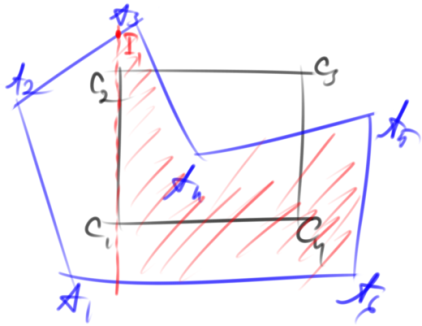
б) использовать пробную функцию (на основе уравнения прямой) F =ax + by + c



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



2. находить точки пересечения



С – отсекатель, А – 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.

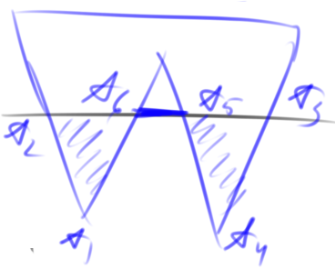
P(t)=Q(s): имеем систему . Предварительно нужно убедиться в непараллельности прямых – точка пересечения должна существовать. Это определяется с помощью видимости концов ребра многоугольника – если видимость разная, то точка пересечения есть.

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.

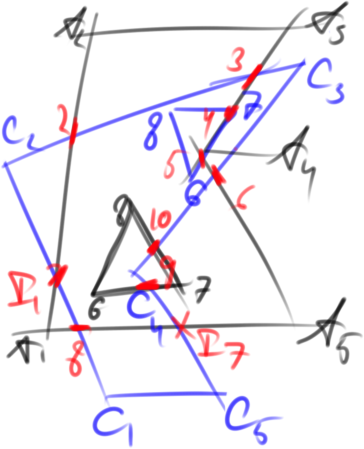
Работа алгоритма сводится к работе с двунаправленными циклическими списками. Решить задачу в общем случае достаточно просто, однако массу неудобств доставляют частные случаи. Проблемы возникают из-за того, что границу отсекателя мы относим к видимой части.

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.

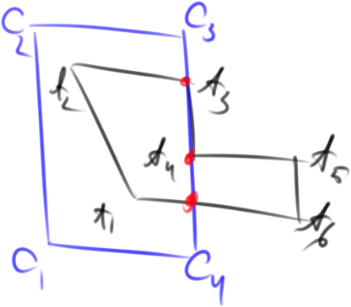
Имеем: I1 I2 (->3) I3 (->1) I4 (->4) C8 C6 I5 I6 I10 A8 A6 I9 I7 I8 I1 //указаны только первые переходы из списка в список!!

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:



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.