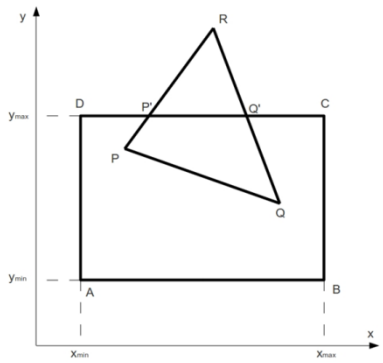
# Алгоритмы отсечения

При генерации изображений могут получаться фигуры произвольной формы и размеров. Зачастую устройства вывода не могут отобразить сгенерированные изображения целиком. Также иногда возникают ситуации, когда необходимо задать область изображения на экране и выводить изображения только внутри этой области.

## Отсечение отрезков

Самой простой задачей отсечения является задача отсечения отрезков. Сформулируем её на конкретном примере. Будем считать, что область вывода задаётся прямоугольником $ABCD$. Рассмотрим пример, и в качестве отсекаемой фигуры возьмём треугольник $PRQ$.




Процесс отсечения должен выполняться полностью автоматически. Т.е. для отрисовки треугольника $PRQ$ должны выполниться только команды отрисовки отрезков $PQ; PP'; Q'Q$. При этом координаты точек $P'; Q'$ заранее не известны.

На практике возможно большое количество взаимных расположений точек отрезка и области вывода. Это разнообразие делает операцию отсечения весьма нетривиальной с алгоритмической точки зрения. Для решения этих задач созданы алгоритмы отсечения.

## Алгоритм Сазерленда-Коэна

Весьма интересен алгоритм отсечения, предложенный Сазерлендом и Коэном. Суть алгоритма заключается в том, что концам отрезка присваивается четырёхбитный код: $b_0, b_1, b_2, b_3$. Этот четырёхбитный код содержит информацию о положении точки относительно области вывода. На практике возможны 9 комбинаций:

| | | |
|---|---|---|
| 1001 | 0001 | 0101 |
| 1000 | 0000 | 0100 |
| 1010 | 0010 | 0110 |

Поясним эти коды:

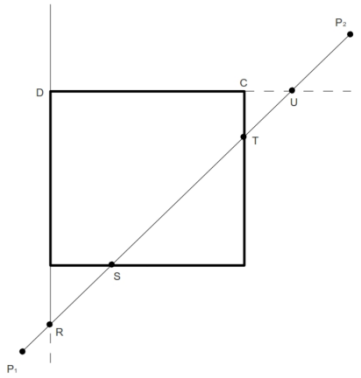
- $b_0 = 0$, если $x \geq x_{min}$;
- $b_0 = 1$, если $x < x_{min}$;
- $b_1 = 0$, если $x \leq x_{max}$;
- $b_1 = 1$, если $x > x_{max}$;

- $b_2 = 0$, если $y \geq y_{min}$;
- $b_2 = 1$, если $y < y_{min}$;
- $b_3 = 0$, если $y \leq y_{max}$;
- $b_3 = 1$, если $y > y_{max}$;

После того, как коды получены, возможны следующие варианты:

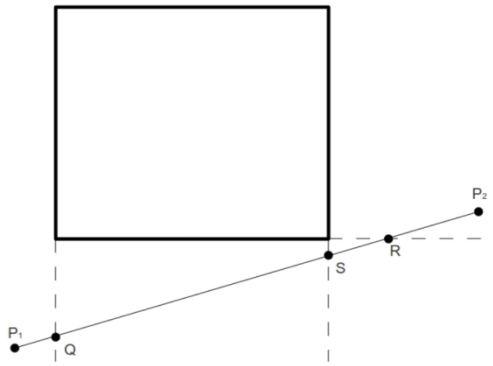
- Коды содержат только $0$, а значит отрезок целиком лежит внутри окна и должен быть отрисован целиком;
- Коды содержат единичный бит в одной и той же позиции, а значит, отрезок лежит за пределами окна и не будет отрисован;
- Во всех остальных случаях в окне лежит только часть отрезка, и это значит, что есть необходимость в отсечении.

Первые два случая легко проверяются с помощью побитовых логических операций. Наибольший интерес представляет именно третий случай. Рассмотрим его на небольшом конкретном примере.




Если код любого из концов содержит единичный бит, то либо $P_1$, либо $P_2$ перемещается из-за пределов окна к одной из его границ (или к её продолжению). Т.е. точка $P_1$ перемещается в точку $R$, а $P_2$ в точку $U$. Для новых точек мы вновь вычисляем четырёхбитные коды. В нашем случае концы отрезка по-прежнему лежат вне окна, т.е. нам понадобится ещё одно перемещение: точка $R$ переместиться в точку $S$, а точка $U$ переместиться в точку $T$. Получается, что процесс отсечения является итеративным. Так как на каждом шаге уменьшается расстояние между концами отрезка, мы можем утверждать, что алгоритм сойдётся. В итоге будет получен отрезок (в нашем случае $(S; T)$, который и надо отобразить).

Рассмотрим работу этого алгоритма на ещё одном примере.



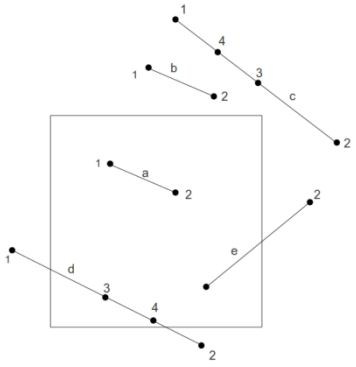

Расположение отрезка $(P_1; P_2)$ не соответствует ни условиям полной видимости, ни условиям полной невидимости, поэтому в этом случае тоже прибегнуть к операции переноса точек.

| Код начала отрезка | Код конца отрезка | Перенос | |
|---|---|---|---|
| | | Что переносится | Куда переносится |
| $P_1(1010)$ | $P_2(0100)$ | $P_1$ | $Q$ |
| $Q(0010)$ | $P_2(0100)$ | $P_2$ | $R$ |
| $Q(0010)$ | $R(0100)$ | $R$ | $S$ |
| $Q(0010)$ | $S(0010)$ | – | – |

После выполненных переносов коды концов отрезка удовлетворяют второму условию, т.е. отрезок целиком не будет выводиться на экран.

## Алгоритм средней точки

В алгоритме Сазерленда-Коэна поиск точки пересечения отрезка с границей окна может занять несколько итераций. Этого можно избежать, если реализовать поиск точки пересечения с помощью двоичного поиска. Эта идея была предложена Спруллом и Сазерлендом. Программная реализация этого алгоритма медленнее, чем алгоритм Сазерленда-Коэна, но аппаратная реализация быстрее, так как основные операции алгоритма весьма эффективно реализуются в аппаратуре.




Этот метод также использует четырёхбитный код, описанный выше. С помощью этих кодов легко выявляются случаи тривиальной видимости $(a)$ и тривиальной невидимости $(b)$. Остальные (нетривиальные) отрезки разбиваются на две равные части, и проверки выполняются для двух вновь полученных отрезков. Деление и проверки будут выполняться до тех пор, пока не будет обнаружено пересечение со стороной окна, или пока отрезок не выродится в точку. После того, как отрезок выродился, мы определяем его видимость и в зависимости от результата либо отрисовываем току, либо нет.

Рассмотрим отрезок $c$. Очевидно, что этот отрезок целиком лежит вне границ окна, но он не может быть отвергнут сразу. Именно поэтому мы выполняем половинные деления и исключаем некоторые части отрезка. Деление заканчивается в тот момент, когда отрезок превращается в точку. Убедившись, что полученная точка невидима, мы делаем вывод, что и весь отрезок не должен быть отрисован.

Отрезок $d$ тоже не определяется тривиально. Его деление на две половины (точкой $3$) приводит к одинаковым результатам для обеих половин. Отрезок $(3;2)$ разбиваем пополам точкой $4$. Теоретически можно уже отрисовать отрезок $(3;4)$, но это не очень эффективно. Правильнее запомнить точку $4$, как наиболее удалённую от точки $1$ видимую точку, а отрезок $(4;2)$ продолжить разбивать до пересечения с нижней границей окна. Полученная точка и будет считаться наиболее удалённой от точки $1$ видимой точкой. Аналогично обрабатывается отрезок $3;1$, и находится видимая точка, наиболее удалённая от точки $2$. Потом между двумя этими точками и происходит отрисовка.

Для отрезков типа $d$ необходимо выполнять два двоичных поиска видимых точек, наиболее удалённых от концов отрезка. Для отрезков типа $e$ необходимость в одном из двоичных поисков отпадает.

## Алгоритм Кируса-Бека

В работе алгоритма Кируса-Бека используется параметрическое представление отрезка: $P_s(t) = P_1 + (P_2 - P_1) * t; t \in [0;1]$. Данный алгоритм позволяет выполнять отсечение не только прямоугольным окном, но и любым выпуклым многоугольником.

Рассмотрим отдельное ребро $E_i$ отсекающего многоугольника. Ориентируем нормаль к этому ребру во внешнюю сторону отсекающего многоугольника. Также будем считать, что отсекающий многоугольник обходится против часовой стрелки. Тогда, если ребро – это вектор $\overrightarrow{P_{E_{i_1}}P_{E_{i_2}}}$, то нормаль $N_{E_i}$ будет пропорциональна $(y_{E_{i_2}} - y_{E_{i_1}}; x_{E_{i_1}} - x_{E_{i_2}})$.

Обозначим прямую, на которой лежит ребро через $L_i$. Тогда область, отсекаемая прямой $L_i$, соответствует точкам $P$, для которых скалярное произведение векторов $(P - P_{E_i})$ и $N_{E_i}$ больше $0$ ($P_{E_i}$ - любая точка на ребре $E_i$). Точка пересечения прямой, на которой лежит отрезок, с отсекающей прямой $L_i$ находится из уравнения $\left((P_s(t) - P_{E_i}), N_{E_i}\right) = 0$. Решая это уравнение, получаем: $t = \frac{\left((P_1 - P_{E_i}), N_{E_i}\right)}{((P_2 - P_1), N_{E_i})}$ [1].

Для описываемого алгоритма также важно в каком направлении (внутрь окна или из него) проходит точка при движении по отрезку от $P_1$ к $P_2$. Это определяется знаком $((P_2 - P_1), N_{E_i})$. Будем обозначать такие точки пересечения как:

| | |
|---|---|
| Потенциально входящие | $\left((P_2 - P_1), N_{E_i}\right) < 0$ |
| Потенциально выходящие | $\left((P_2 - P_1), N_{E_i}\right) > 0$ |

После того, как рассчитаны координаты $t$ для всех возможных пересечений с прямыми $L_i$, следует выбрать максимальную координату из потенциально входящих и минимальную из потенциально выходящих $(t_{\text{ВхМакс}}; t_{\text{ВыхМин}})$. Если прямая, на которой лежит отрезок $P_1P_2$, пересекает

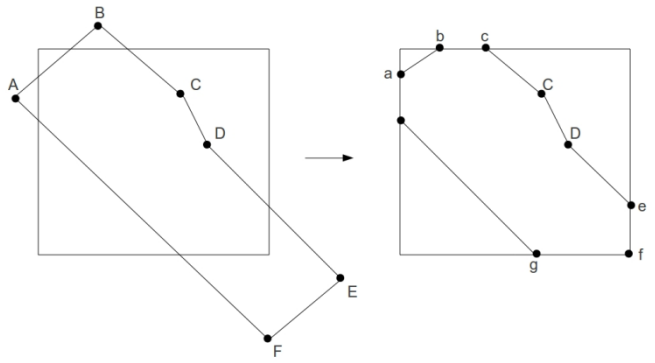
[1] Если $\left((P_2 - P_1), N_{E_i}\right) = 0$, это означает, что отсекаемый отрезок параллелен $L_i$, т.е. невозможно указать единственную

отсекающий многоугольник, то $t_{\text{ВхМакс}} < t_{\text{ВыхМин}}$. В этом случае, если пересечение $[t_1, t_2] = [t_{\text{ВхМакс}}, t_{\text{ВыхМин}}] \cap [0,1]$ непусто, то $P_s(t_1)P_s(t_2)$ и будет искомым отсечённым отрезком, в противном случае отрезок полностью лежит вне отсекаемой области. Запишем алгоритм более формально:

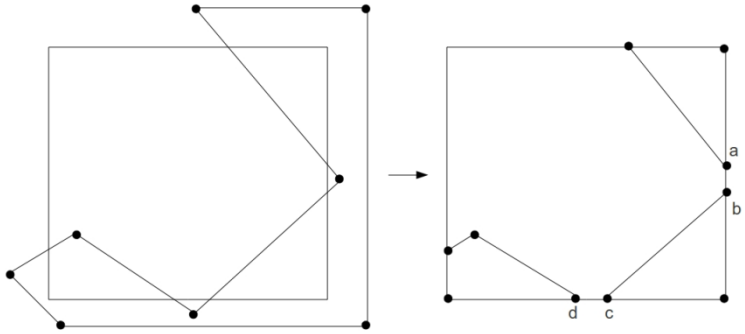
вычислить $N_{E_i}$ и взять $P_{E_i} = P_{E_{i_1}}$ для каждого ребра
отсечь отрезок $P_1P_2$
begin
 $D$:=$P_2 - P_1$;
 $t_{\text{вх}}$:=0;
 $t_{\text{вых}}$:=1;
 для каждого ребра $E_i$
 begin
  $dp$:=$(D, N_{E_i})$;
  if $dp <> 0$ then
  begin
   $t$:=$-\frac{((P_1 - P_{E_i}), N_{E_i})}{dp}$;
   if $dp < 0$ then
    if $t > t_{\text{вх}}$ then
     $t_{\text{вх}}$:=$t$;
   else
    if $t < t_{\text{вых}}$ then
     $t_{\text{вых}}$:=$t$;
  end;
 end;
 if ($t_{\text{вх}} < t_{\text{вых}}$) then
  отрисовать отрезок $P_s(t_{\text{вх}})P_s(t_{\text{вых}})$
 else
  ничего не отрисовывать
end;

## Отсечение многоугольников

Для отсечения многоугольников можно использовать алгоритмы отсечения отрезков, но только в том случае, когда многоугольник – это контурное изображение, в этом случае отсечённый многоугольник воспринимается, как совокупность отрезков. Но если многоугольники воспринимаются, как сплошные области, необходимо, чтобы замкнутость сохранялась и у результата: т.е. $(b; c), (e; f), (f; g), (h; a)$ должны быть добавлены к отсечённому многоугольнику.




Большие сложности также возникают, когда в результате отсечения получается несколько несвязных областей.




Если отрезки $(a;b),(d;c)$ будут включены в отсечённый многоугольник, то эти отрезки будут отрисованы цветом многоугольника, что противоречит ожидаемому результату.

Одним из наиболее распространённых алгоритмов отсечения многоугольников, является алгоритм Сазерленда-Ходжмена.

## Алгоритм Сазерленда-Ходжмена

Алгоритм Сазерленда-Ходжмена предназначен для отсечения многоугольника прямоугольным окном. Алгоритм основан на отсечении частей многоугольника полуплоскостями, составляющими окно вывода. Для получения результата необходимо отсечь части многоугольника, лежащие вне каждой полуплоскости.

Будем считать, что многоугольник задан списком своих вершин: $P_1, P_2, \dots, P_N$. От списка вершин легко перейти к списку рёбер $\overrightarrow{P_1P_2}, \dots, \overrightarrow{P_{N-1}P_N}, \overrightarrow{P_NP_1}$. Относительно произвольной полуплоскости ребро $\overrightarrow{P_k, P_{k+1}}$ может находиться в одном из следующих положений:

- Целиком внутри полуплоскости;
- Целиком вне полуплоскости;
- Входить в полуплоскость;
- Выходить из полуплоскости.

Сформулируем алгоритм, который выведет координаты концов видимой части рёбер после их отсечения полуплоскостью.

```
отсечь P[1]…P[N] относительно полуплоскости,  заданной границей L
begin
 for i:=1 to N do
 begin
  j:=j+1;
  if j>N then
   j:=1;
  if (P[i];P[j]) пересекает L then
  begin
   I:= точка пересечения (P[i];P[j]) с L;
   вывести I;
  end;
  if P[j] видима then
   вывести P[j];
 end;
end;
```

Вершины отсечённого многоугольника будут выведены в порядке обхода. После применения этого алгоритма ко всем $4$ полуплоскостям получится многоугольник, отсечённый относительно области видимости. Этот алгоритм легко можно обобщить для любого выпуклого многоугольника, достаточно только представить окно в виде пересечения полуплоскостей.

Очевидным недостатком этого алгоритма является некорректная обработка ситуаций, когда в результате отсечения получается несколько изолированных многоугольников. В результате работы алгоритма будут получены не только изолированные многоугольники, но и отрезки, связывающие их.