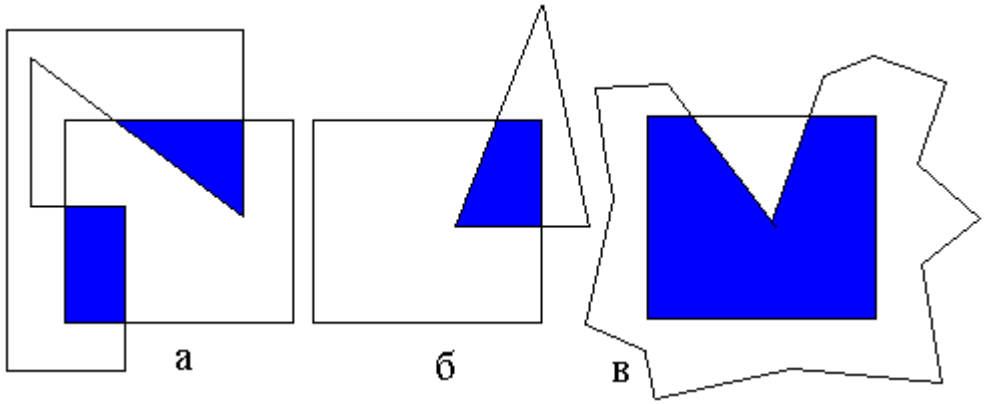
Отсечение и заполнение областей

Во многих приложениях области задаются многоугольниками, вершины которых хранятся в прикладной базе данных.

Многоугольники должны сначала подвергнуться видовому преобразованию, пройти процесс ***отсечения***, затем должны быть переведены в систему координат устройства, затем ***закрашены***.

Алгоритм, который отсекает многоугольник, должен уметь обрабатывать самые различные случаи.



Алгоритм Сазерленда-Ходгмана (Хонджмена)

Основная идея алгоритма состоит в том, что отсечь многоугольник относительно одной прямой или плоскости очень легко. Исходный и каждый из промежуточных многоугольников отсекается последовательно относительно одной прямой.

Исходный многоугольник задается списком вершин $P_1,\ldots,P_n$,который порождает список его ребер P_1P_2,P_2P_3 ,$P_{n-1}P_n,P_nP_1$.

Результатом работы алгоритма является список вершин многоугольника, у которого все вершины лежат по видимую сторону от очередной отсекающей плоскости. Поскольку каждая сторона многоугольника отсекается независимо от других, то достаточно рассмотреть только возможные ситуации расположения одного отрезка относительно одной отсекающей плоскости.

Алгоритм (S.H.CLIP) "обходит" вокруг многоугольника P_n к P_1 , проверяя на каждом шаге отношение между вершинами и отсекающими точками.

Нуль, одна или две вершины добавляются в выходной список вершин, определяющий усеченный многоугольник.

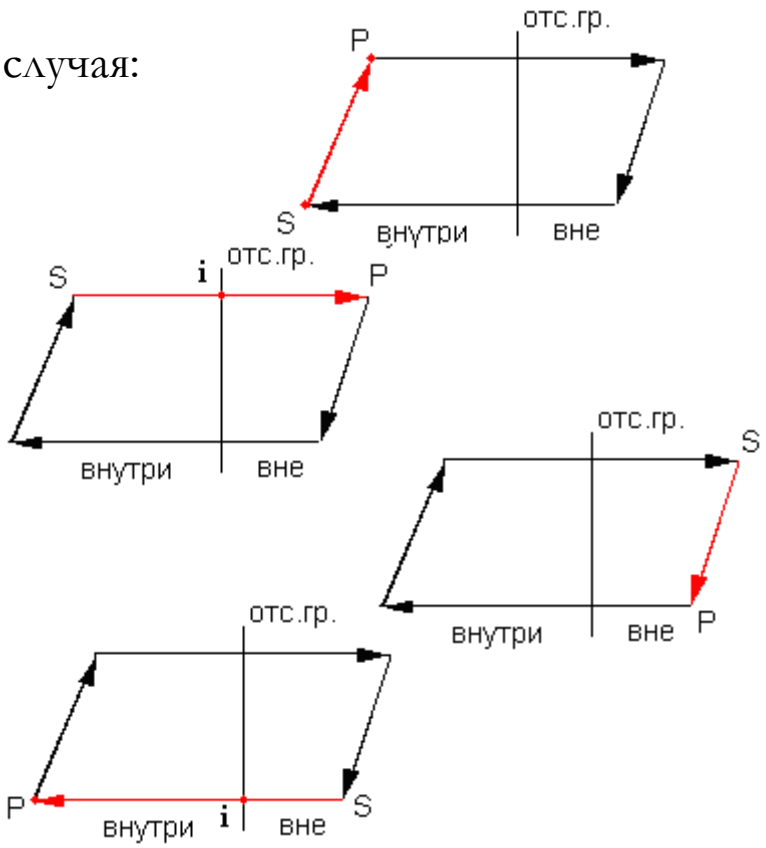
Необходимо проанализировать четыре возможных случая:

Случай 1. Ребро многоугольника лежит внутри - к выходному списку добавляется вершина P.

Случай 2. В качестве вершины выводится точка "i". Начальная точка была выведена в случае 1.

Случай 3. . Обе вершины за пределами и ни одна из них не выводится.

Случай 4. К выходному списку добавляется две точки. Точка пересечения "i" и вершина P.

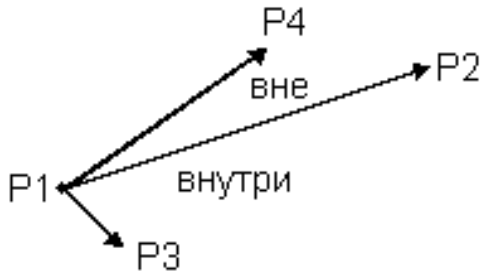


Как определить находится точка внутри или вне границы?

Проверка основана на векторном произведении вектора, проведенного из точки P_1 в P_2 на вектор, соединяющий точку P_1 с исследуемой точкой.

Если направление совпадает с положительным направлением оси Z, точка лежит слева (и следовательно вне), если же она направлена противоположно оси Z, точка находится внутри.

Для использования этого алгоритма требуются выпуклые отсекающие области. Однако во многих приложениях (удаление невидимых линий и т.д.) требуется уметь отсекать и по не выпуклым областям. Это способен делать более мощный, но и более сложный алгоритм Вейлера-Азертона



Векторным произведением двух векторов $V=(V_x,V_y)$ и $W(W_x,W_y)$ принадлежащим плоскости (x,y) является вектор, значение Z-компоненты которого есть $V_xW_y-V_yW_x$. Если эта величина отрицательна, проверяемая точка лежит внутри. Если положительна, то вне.

Заполнение областей

Область представляет собой группу примыкающих друг к другу связанных пикселей. Во многих интерактивных растровых приложениях пользователь определяет области. Эти области должны быть заполнены пикселями, имеющими заданное значение. В некоторых случаях требуется заполнить область узором.

Пользователь схематически очеркивает область, затем выбирает цвет и указывает на произвольную точку области. Компьютер закрашивает область.

Области, заданные своими внутренними значениями, называются ***внутренне-определенными***. Алгоритмы, которые работают с такими областями называются внутренне-заполняющими алгоритмами.

Области, задаваемые границей, ***гранично-определенными***. Алгоритмы - гранично-заполняющими.

В методах растровой развертки пытаются определить в порядке сканирования строк, лежит ли точка внутри многоугольника или контура.

В методах затравочного заполнения предполагается, что известна некоторая точка внутри замкнутого контура. Ищутся точки соседние с затравочной и расположенные внутри контура.

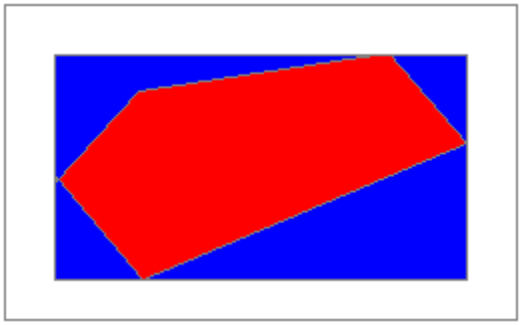
Если соседняя точка вне контура, значит обнаружена граница контура. Если внутри, то она становится новой затравочной.

Методы растровой развертки

Многие замкнутые контуры являются простыми многоугольниками. Если контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками.

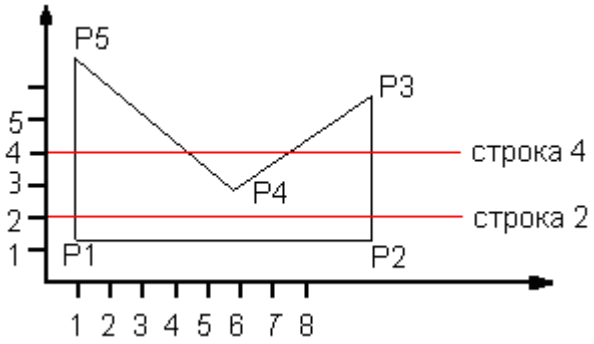
Простейший метод заполнения многоугольника состоит в проверке на принадлежность каждого пиксела в растре. Такой метод чрезмерно расточителен.

Затраты можно уменьшить, если вычислить наименьший прямоугольник, содержащий внутри себя многоугольник.

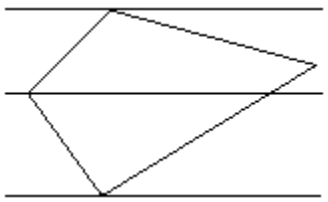


Алгоритмы построчного сканирования

Можно разработать более эффективный способ, если воспользоваться тем фактом, что соседние пикселы имеют одинаковые характеристики (кроме пикселов граничных ребер). Это свойство называется пространственной когерентностью. Характеристики пикселов на данной строке изменяются только там, где ребро многоугольника пересекает строку . Эти пересечения делят строку на части.



Строка 2 пересекает многоугольник в точках X=1 и X=8. Получаем три области: две вне, одна внутри. Строка 4 делится на пять областей.
Точки пересечения необходимо отсортировать в возрастающей последовательности.



Дополнительная трудность возникает при пересечении сканирующей строки и многоугольника точно по вершине. Правильный результат можно получить, учитывая точку пересечения в вершине два раза, если она является точкой локального минимума или максимума, или учитывая ее один раз в противном случае.

Простой алгоритм с упорядоченным списком ребер

- Определить для каждого ребра многоугольника точки пересечения со сканирующими строками, проведенными через середину интервалов (можно использовать алгоритм Брезенхема или ЦДА). Горизонтальные ребра игнорируются. Занести каждое значение (X,Y+1/2) в список.

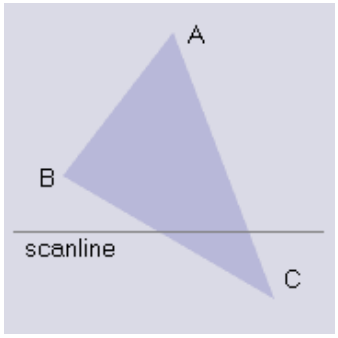
- Отсортировать список по строкам и по возрастанию в строке. Преобразовать эти данные в растровую форму:

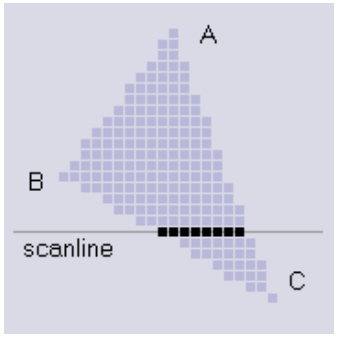
  - Выделить из списка пары элементов (X_1,Y_1) и (X_2,Y_2). Структура списка гарантирует, что $Y=Y_1=Y_2$ и $X_1<=X_2$.
  - Активировать на сканирующей строке пикселы для целых значений X таких, что $X_1<=X+1/2<=X_2$.

Закраска треугольника

Возьмем любой треугольник. Его изображение на экране - набор горизонтальных отрезков, причем из-за того, что треугольник - фигура выпуклая, каждой строке экрана соответствует не более одного отрезка.

Поэтому достаточно пройтись по всем строкам экрана, с которыми пересекается треугольник (то есть, от минимального до максимального значения y для вершин треугольника), и нарисовать соответствующие горизонтальные отрезки.





Отсортируем вершины так, чтобы вершина A была верхней, C - нижней, тогда у нас min_y = A.y, max_y = C.y, и нам надо пройтись по всем линиям от min_y до max_y.

Рассмотрим какую-то линию sy, A.y <= sy <= C.y. Если sy < B.y, то она пересекает стороны AB и AC; если sy >= B.y - то стороны BC и AC. Мы знаем координаты всех вершин, поэтому мы можем написать уравнения сторон и найти пересечение нужной стороны с прямой y = sy. Получим два конца отрезка. Так как мы не знаем, какой из них левый, а какой правый, сравним их координаты по x и обменяем значения, если надо.

Рисуем этот отрезок, повторяем процедуру для каждой строки - треугольник нарисован.

Остановимся более подробно на нахождении пересечения прямой y = sy (текущей строки) и стороны треугольника, например AB.
Напишем уравнение прямой AB в форме x = k*y+b:

x = A.x+(y-A.y)*(B.x-A.x)/(B.y-A.y)

Подставляем сюда известное для текущей прямой значение y = sy:

x = A.x+(sy-A.y)*(B.x-A.x)/(B.y-A.y)

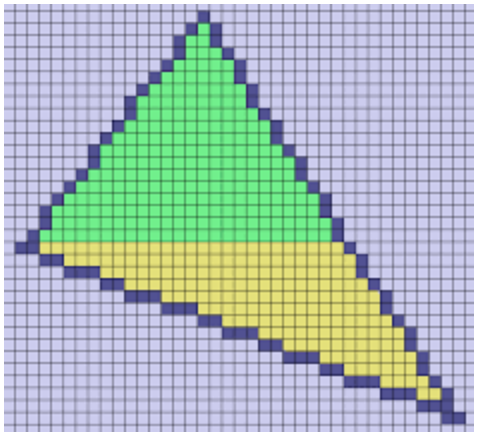
Вот, в общем-то, и все. Для других сторон пересечение ищется совершенно точно так же.

Пример кода:

```
// ...
// здесь сортируем вершины (A,B,C)
// ...
 for (sy = A.y; sy <= C.y; sy++)
    {
      x1 = A.x + (sy - A.y) * (C.x - A.x) / (C.y - A.y);
      if (sy < B.y)
                x2 = A.x + (sy - A.y) * (B.x - A.x) / (B.y - A.y);
        else
                x2 = B.x + (sy - B.y) * (C.x - B.x) / (C.y - B.y);
     if (x1 > x2) { tmp = x1; x1 = x2; x2 = tmp; }
     drawHorizontalLine(sy, x1, x2);
    }
 // ...
```

Надо, правда, защититься от случая, когда B.y = C.y - в этом случае произойдет попытка деления на ноль (только в этом, потому как, если C.y = A.y, то треугольник пустой и рисовать его не стоит, или можно рисовать горизонтальную линию; а если B.y = A.y, то sy >= A.y и до деления на B.y - A.y не дойдет). Код изменится совсем чуть-чуть:

```
// ...
// здесь сортируем вершины (A,B,C)
// ...
for (sy = A.y; sy <= C.y; sy++)
{
   x1 = A.x + (sy - A.y) * (C.x - A.x) / (C.y - A.y);
   if (sy < B.y)
            x2 = A.x + (sy - A.y) * (B.x - A.x) / (B.y - A.y);
    else {
               if (C.y == B.y) x2 = B.x;
                else x2 = B.x + (sy - B.y) * (C.x - B.x) / (C.y - B.y);
            }
   if (x1 > x2) { tmp = x1; x1 = x2; x2 = tmp; }
    drawHorizontalLine(sy, x1, x2);
}
// ...
```



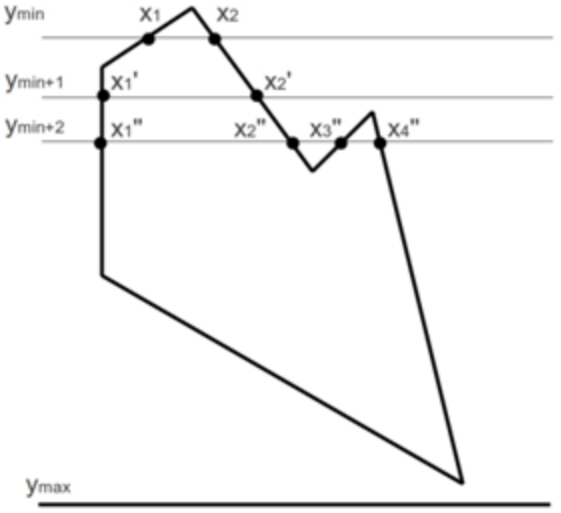
Алгоритмы со списком рёберных точек

Этот алгоритм подходит только для тех случаев, когда закрашиваемая область может быть задана в виде многоугольника.

Будем предполагать, что каждое ребро многоугольника задаётся координатами его концов x1,y1 и x2,y2

Будем обрабатывать только те рёбра, которые пересекаются с текущей линией развёртки.

Найдем y_min и y_max



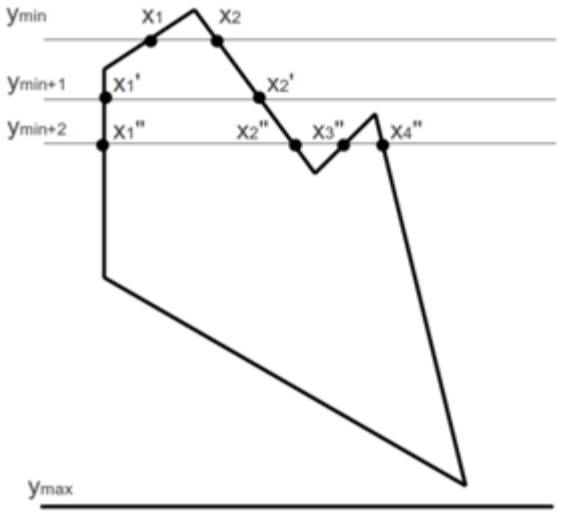
Далее от y_min до y_max находим точки пересечения с ребрами x1 и x2.

Если точек более 2 или менее 2 рассматриваем этот случай

Упорядочиваем по возрастанию.

Заполняем полученный отрезок.

Переходим к следующей строке.



Алгоритмы заполнения с затравкой

Может быть два типа областей- **четырехсвязные** и **восьмисвязные**:

4-х - каждый пиксел достижим из любого другого пиксела этой области при перемещении: влево, вправо, вверх, вниз.

8-ми - еще и по диагоналям. (Это может приводить к неожиданным результатам.)

Простой рекурсивный алгоритм

```
void PixelFill (int x, int y, int BolderColor, int color)
{
   int c=getpixel(x,y);

   if (( c!=BolderColor) && (c!=color))
   {
      putpixel (x,y,color);
      PixelFill (x-1, y, BolderColor, color);
      PixelFill (x+1, y, BolderColor, color);
      PixelFill (x, y-1, BolderColor, color);
      PixelFill (x, y+1, BolderColor, color);
   }
}
```

Процедура проста, но существенно рекурсивна, а наличие многих уровней вложенности требует времени и может вызвать переполнение памяти.

Построчный алгоритм заполнения с затравкой

Размер стека минимизируется за счет хранения только одного затравочного пиксела для любого непрерывного интервала на строке.

Область может быть как выпуклой, так и не выпуклой, а также может содержать дыры.

В области внешней и примыкающей к области не должно быть пикселов с цветом, которым область или многоугольник заполняется.

Алгоритм

1. Затравочный пиксел на интервале извлекается из стека, содержащего затравочные символы.
2. Интервал с затравочным пикселом заполняется влево и право от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница.
3. В переменных $X_{лев}$ и $X_{прав}$ заполняются крайний левый и крайний правый пикселы интервала.
4. В диапазоне $X_{лев}<=X<=X_{прав}$ проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется есть ли в них еще незаполненные пикселы. Если есть (т.е. не все пикселы граничные, или уже заполненные), то в указанном диапазоне крайний правый пиксел в каждом интервале отмечается как затравочный и помещается в стэк.

При инициализации в стек помещается единственный затравочный пиксел, работа заканчивается при опустошении стека.

Рассмотрим версию одного из самых популярных алгоритмов, когда для заданной точки (X,Y) определяется и заполняется максимальный отрезок $[X_1,X_2]$, содержащий эту точку и лежащий внутри области.

После этого в поисках еще не заполненных пикселов проверяются отрезки, лежащие выше и ниже. Если такие отрезки находятся, то функция рекурсивно вызывается для их обработки.

Ниже приведен текст и пример работы этого алгоритма. При запуске программы сначала прорисуются границы областей. Необходимо нажать любую кнопку на клавиатуре. После чего вся фигура заполнится, кроме внутренних областей.

```
int LineFill (int x, int y, int dir, int PrevXl, int PrevXr)
{
   int xl=x,  xr=x, c;
   // Find line segment
   do c=getpixel(--xl, y);    while ((c!=BolderColor) && (c!=Color));
   do c=getpixel(++xr, y);  while ((c!=BolderColor) && (c!=Color));
    xl++; xr--;
    line (xl, y, xr, y); // fill segment
    // Fill adjacent segments in the same direction
    for (x=xl; x<=xr; x++)
            { c=getpixel (x, y+dir);
              if ((c!=BolderColor) && (c!=Color)) x=LineFill (x, y+dir, dir, xl, xr);
            }
      for (x=xl; x<PrevXl; x++)
            { c=getpixel (x, y-dir);
              if ((c!=BolderColor) && (c!=Color)) x=LineFill (x, y-dir,-dir,xl,xr);
            }
       for (x=PrevXr; x<xr; x++)
            { c=getpixel (x, y-dir);
              if ((c!=BolderColor) && (c!=Color)) x=LineFill (x, y-dir, -dir, xl, xr);
            }
     return xr;
}
```

```
int BolderColor=WHITE;
int Color=GREEN;

void Fill (int x, int y) {  LineFill (x, y, 1, x, x);  }

void main( )
{
    circle (320,200,140);
    circle (260,200,40);
    circle (380,200,40);
    getch();
    setcolor(Color);
    Fill (320,300);
   getch();
}
```