Машинная графика
Computer Graphics

Лекция 5.

«Заполнение многоугольников»

План лекции

- Алгоритм с упорядоченным списком рёбер
- Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (САР)
- Алгоритм последовательных инверсий
- Простой алгоритм заполнения с затравкой
- Построчный алгоритм заполнения с затравкой

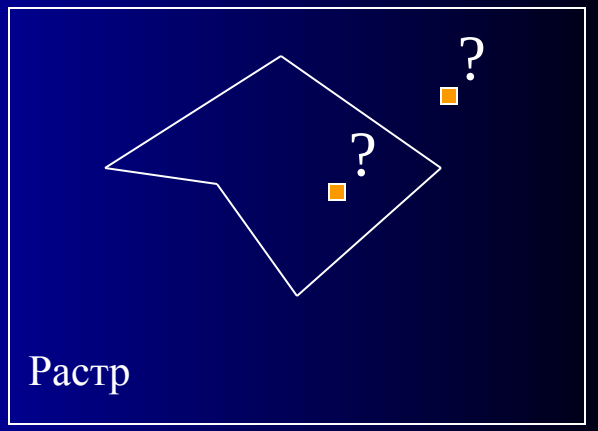
Простейший вариант заполнения многоугольников

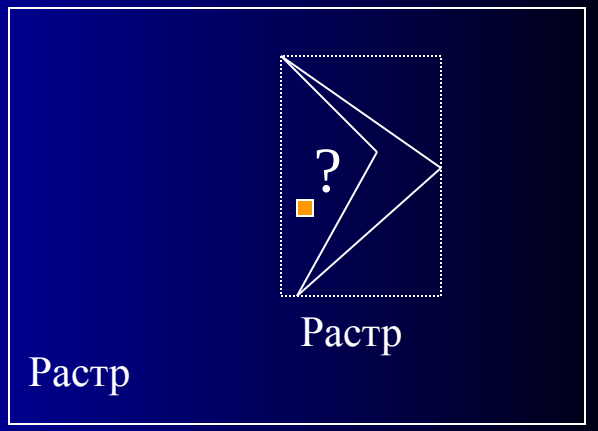
Многие замкнутые контуры являются простыми многоугольниками. В крайнем случае, любой контур теоретически может быть представлен как многоугольник с единичными сторонами.

Простейший метод заполнения многоугольника состоит в проверке каждого пикселя растра на принадлежность внутренней области многоугольника. Эффективность данного подхода весьма невысока.

Увеличить эффективность данного способа можно определив сначала описывающий многоугольник.

Но данный вариант так же далёк от совершенства.





Закраска фигур – 2 подхода

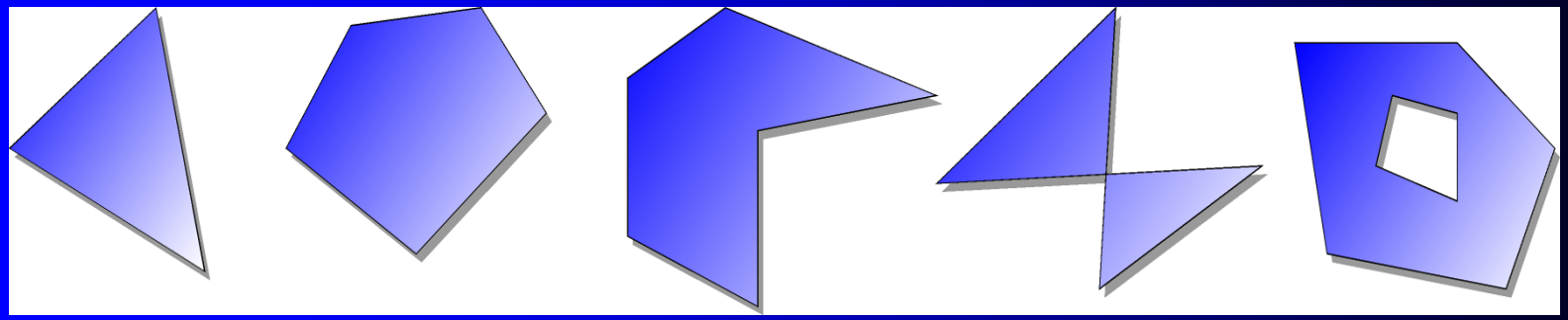
- Затравочное заполнение (Flood Filling)
- Растровая развёртка или сканирование строк (Scan lines)

Оба указанных подхода демонстрируют одну из уникальных способностей растрового графического устройства – представление сплошных областей.

Для векторных устройств – отображение сплошных областей – нетривиальная задача. Затравочное заполнение для её решения использоваться не могло по определению.

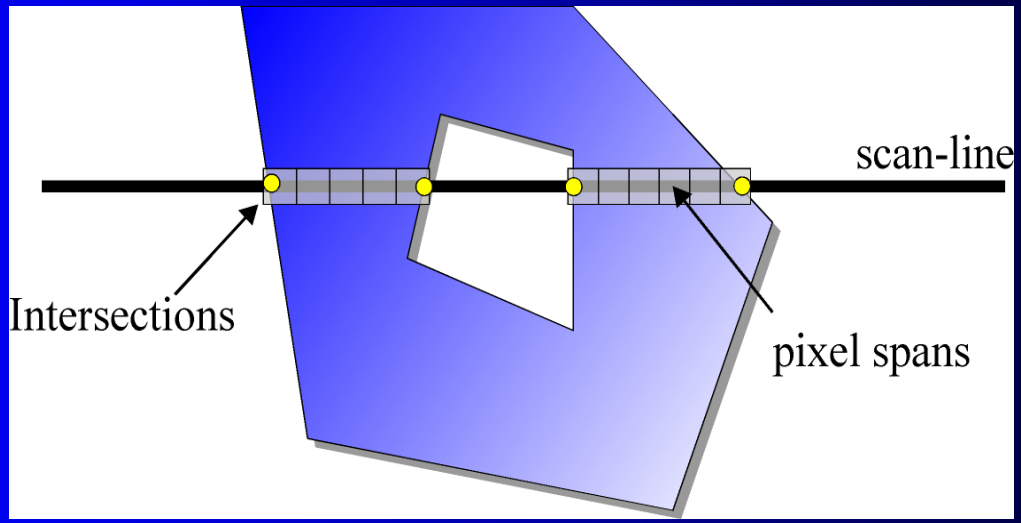
Разновидности вариантов алгоритмов закраски многоугольников

- Существует большоё множество вариантов алгоритмов растеризации сплошных многоугольников. Каждый из них использует особенности тех многоугольников, которые отображает:
  - некоторые алгоритмы работают только с *треугольными полигонами*
  - другие требуют, что бы многоугольники были выпуклыми и не самопересекающимися, некоторые не допускают наличие отверстий в многоугольниках.



Растровая развёртка многоугольников

- *Растровая развёртка многоугольника* – классический алгоритм, заполнения основан на предположении, что любое горизонтальное сечение контура многоугольника состоит из четного числа точек.

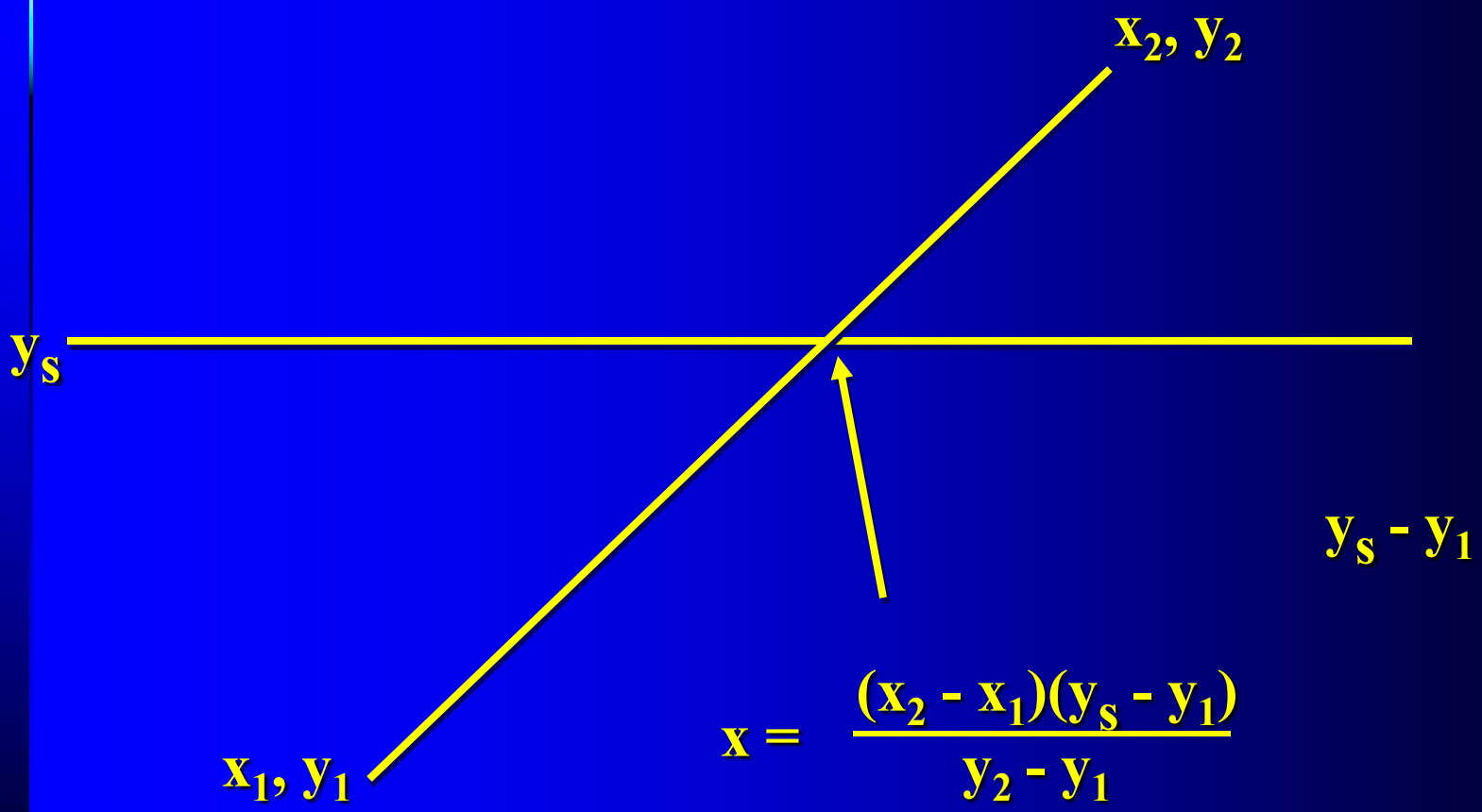


- Указанный способ является ядром *scan-line rendering algorithm*, используемого во многих коммерческих продуктах, как, например: Renderman and 3D Studio MAX.

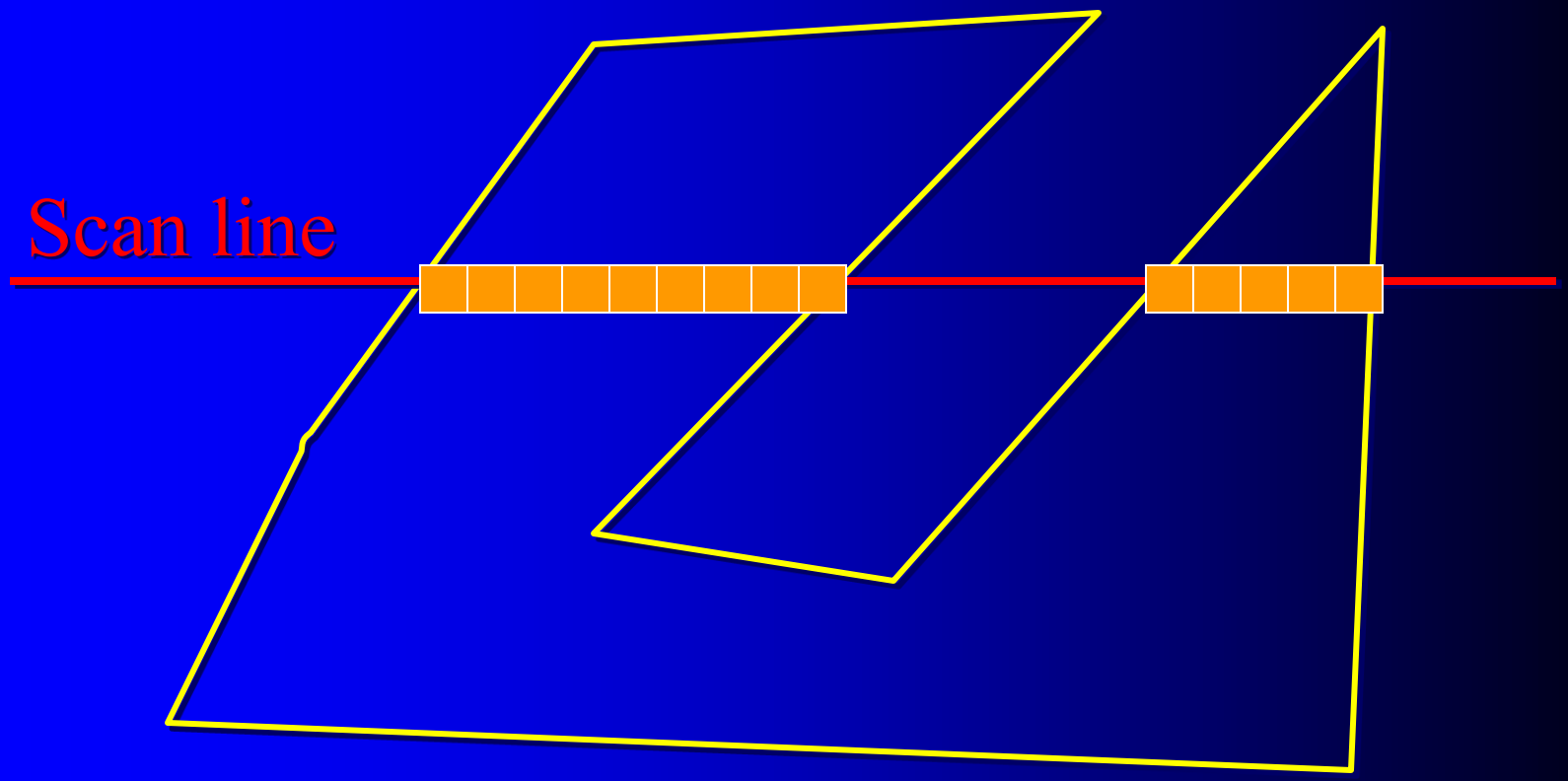
Растровая развёртка многоугольников, общая последовательность действий

- Приведение многоугольника к экранным координатам (в том числе и масштабирование если нужно)
- Для каждой сканирующей строки определяются *точки пересечения* со сторонами полигона
- Вычисляются промежутки строк, соответствующих внутренним частям многоугольника
- Промежутки закрашиваются требуемым цветом.

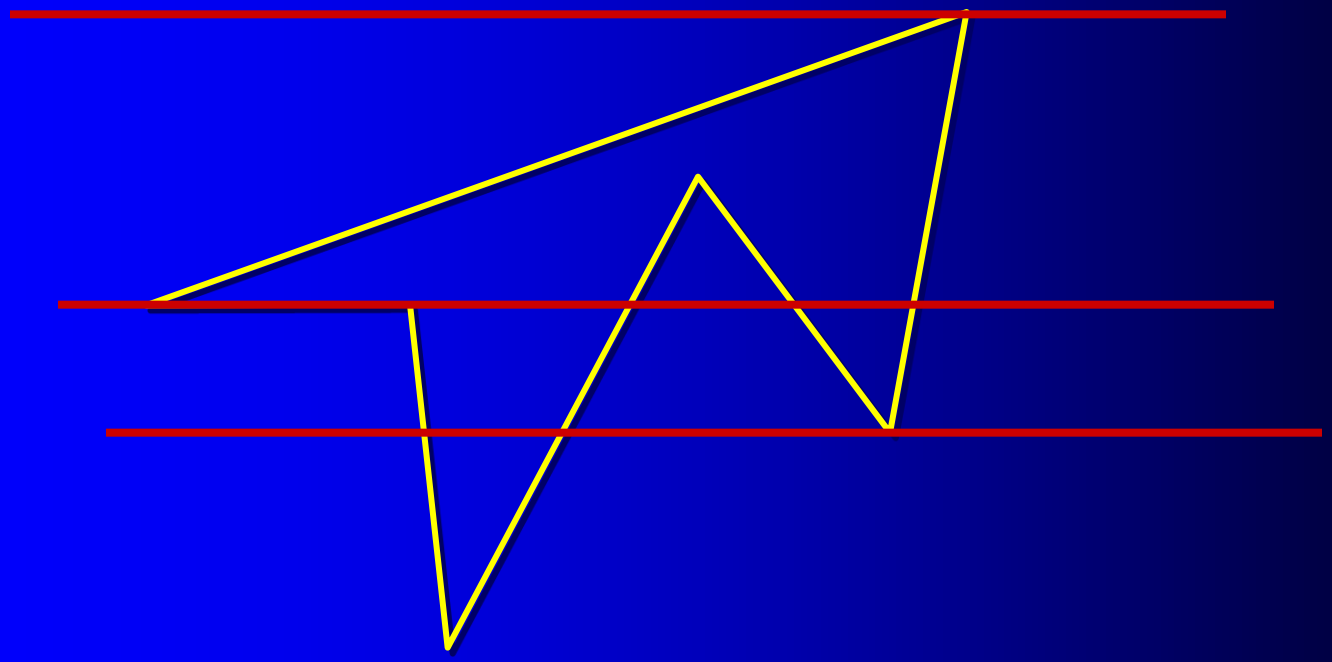
Вычисление точек пересечения



Растровая развёртка многоугольников



Растровая развёртка многоугольников – нюансы

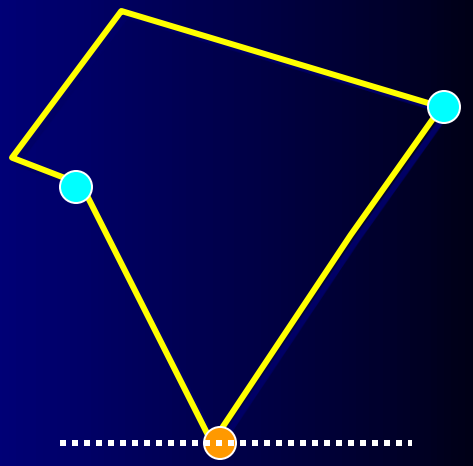


Исключительные случаи: отдельно стоящая вершина, горизонтальное ребро, двойное пересечение

Растровая развёртка многоугольников – обработка исключительных случаев

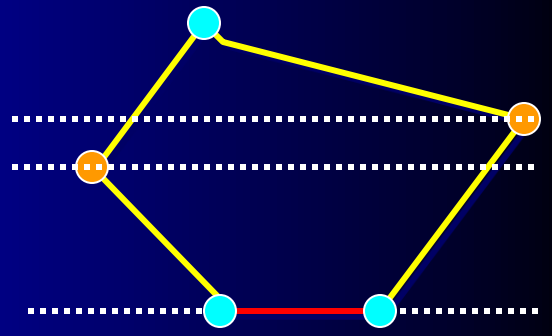
Исключительные случаи:

- горизонтальные рёбра игнорируются,
- отдельно стоящая вершина учитывается два раза при условии, что она – локальный экстремум, и один раз если она им не является.



Проверка вершины на локальный экстремум:

- проверяем оставшиеся концы рёбер, смыкающихся в данной вершине, если у обоих рёбер у координаты концов больше, чем у проверяемой вершины – то вершина лок. минимум, если меньше - то лок. максимум. В оставшемся случае – вершина не является локальным экстремумом.



Алгоритм с упорядоченным списком рёбер

Базируясь на рассмотренных выше принципах были разработаны алгоритмы, называемые алгоритмами с упорядоченными списками рёбер. Они все используют сортировку и их эффективность, как правило, зависит от эффективности метода сортировки.

Простейший вариант алгоритма:

- Для каждого ребра определяются точки пересечения
- Горизонтальные рёбра игнорируются
- Каждое пересечение заносится в список. Список сортируется по строкам (т.е. по y координате), затем в пределах каждой строки – сортировка по x координате.
- Из списка выбираются пары элементов, образующих закрашиваемый интервал.

Недостаток: большой список точек для сортировки.

Алгоритм с упорядоченным списком рёбер

Разделим один список на несколько – для каждой строки будет свой отдельный. Тогда алгоритм будет состоять из трёх шагов.

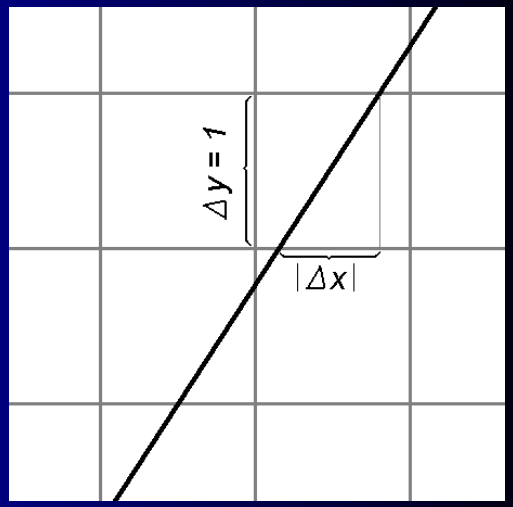
Шаг I. Растеризуем все негоризонтальные ребра многоугольника, помещая полученные точки в списки: каждому значению $y = y_{min}, y_{min}+1, \ldots, y_{max}$ сопоставляется список x-координат всех пикселей из растеризации, находящихся на горизонтали y (здесь y_{min} и y_{max} – минимальная и максимальная y-координаты пикселей в растровом изображении многоугольника).

Формально эту процедуру можно записать следующим образом:

Алгоритм с упорядоченным списком рёбер

```
Для каждого ребра (x1, y1) – (x2, y2)
{
    y=int(y1+1); //Округление до большего
    dx = (x2 – X1)/(y2 – y1);
    x = X1+dx*(y2 – y1);

    while(y <= y2)
    {
      PutToList(x, y);//поместить x в список,
                      //соответствующий данному y
      y++;
      x += Δx;
     }
}
```



где dx – смещение по *x* при движении вдоль ребра, соответствующее увеличению *y*-координаты на 1

Алгоритм с упорядоченным списком рёбер

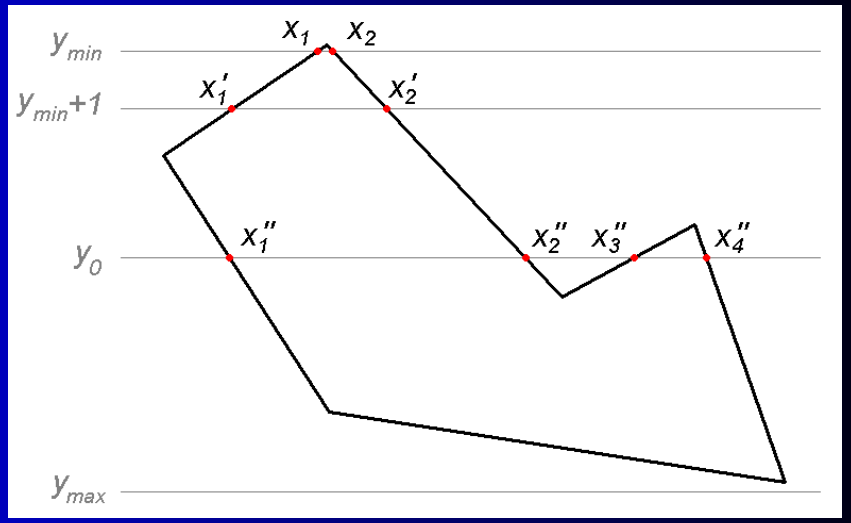
Шаг II.

Для каждого y упорядочим список по возрастанию x.

Шаг III.

Для каждого y закрашиваются все промежутки вида $[x_{2i-1}, x_{2i}]$.

Следует отметить, что в каждом списке будет содержаться чётное количество элементов.



y_{min}: x_1, x_2
$y_{min}+1$: x_1', x_2'
…
…
y_0: x_1'', x_2'', x_3'', x_4''
…
y_{max}: –

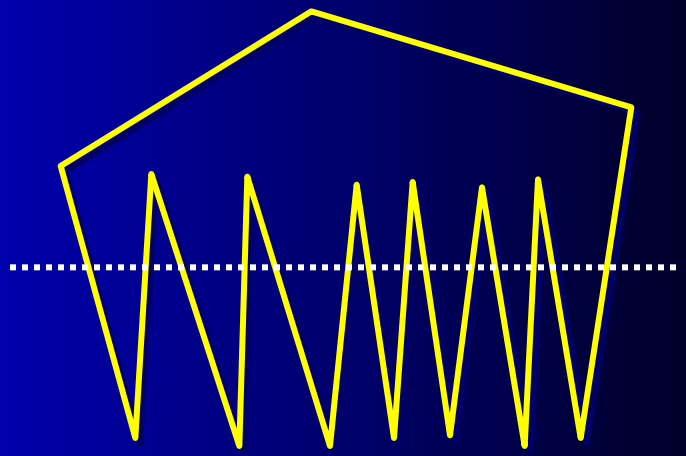
Алгоритм с упорядоченным списком рёбер

Недостатки:

Несмотря на упрощение сортировки, в данном алгоритме либо ограничено число пересечений ребра с текущей сканирующей строкой, так как выделяемая память для каждой у группы будет одинакова.

Либо если число пересечений невелико, а размер выделяемой памяти предусмотрен большим, то слишком много памяти будет выделяться «в пустую».

Что будет в данном случае?



Выход:
использование динамических структур данных

Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (САР)

Вместо хранения в памяти точки пересечения контура с каждой строкой растра, организуем организуем список «активных» ребер (САР), в котором будем хранить информацию обо всех ребрах многоугольника, пересекаемых текущей строкой.

Удобство списка в том, что при переходе к новой строке не требуется его полностью переформировывать. Достаточно лишь удалить из него «закончившиеся» ребра (то есть ребра из САР, чей нижний конец оказался выше нового значения *y*) и добавить вновь появившиеся.

Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (САР)

Формальное описание алгоритма:

Этап подготовки. Для каждого ребра создадим структуру данных:

$y=int(y_1+1)$ – начальная точка ребра по у координате,

x_0- х координата точки пересечения ребра с наивысшей сканирующей строкой.

dx–смещение по *x* при движении вдоль ребра, соответствующее увеличению *y*-координаты на 1,

dy-число пересекаемых ребром строк.

Ребро многоугольника заносится в соответствующую y-группу.

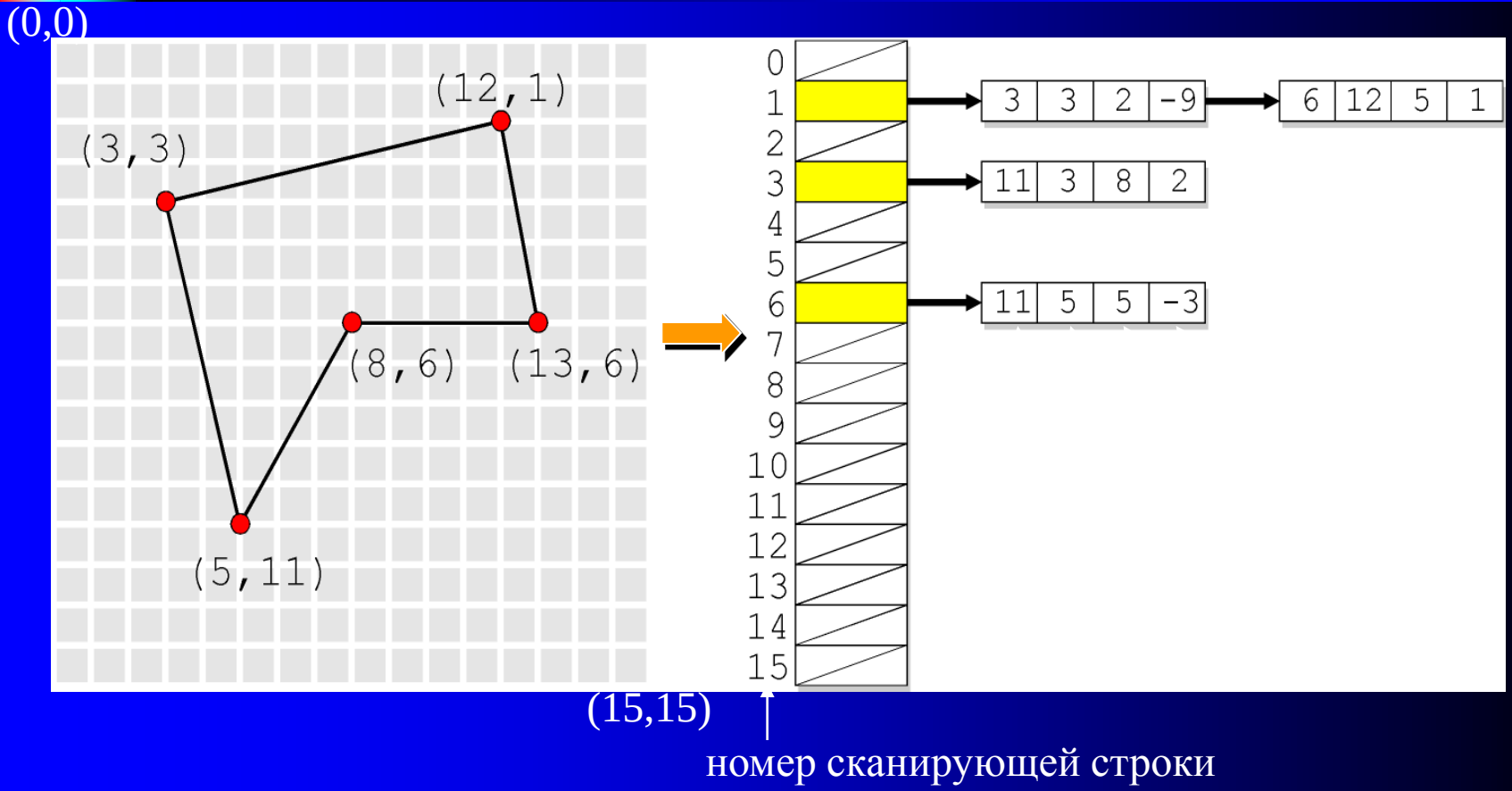
Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (САР)

Формальное описание алгоритма:

Этап отрисовки. Для каждой сканирующей строки проверить y-группу на наличие рёбер. Если таковые есть, то они заносятся в САР. Далее:

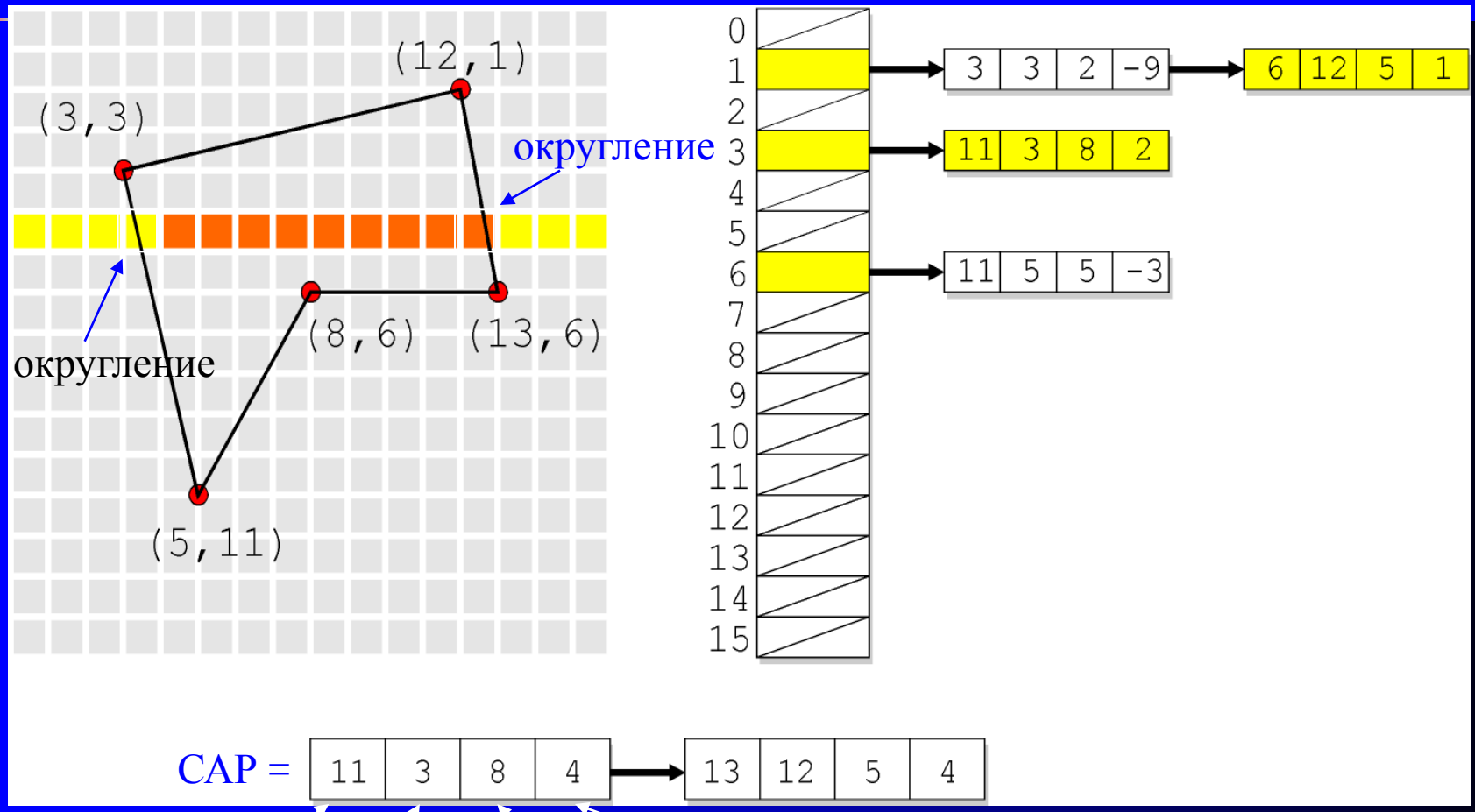
- Отсортировать x координаты точек пересечения рёбер САР с текущей строкой.
- Выделить пары пересечений из отсортированного по x списка.
- Закрасить соответствующие интервалы.
- Для всех рёбер в САР уменьшить dy на 1.
- Проверить dy для всех рёбер из САР. Если dy<0, то исключить его из САР.
- Вычислить новые координаты x точек пересечения.
- Перейти к следующей сканирующей строке.

Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (CAP)



Прим: линия (8,6) → (13,6) не учитывается, согласно правилам

Алгоритм с упорядоченным списком рёбер, использующий список активных рёбер (CAP)



Растровая развёртка многоугольников

Преимущество приведенных выше алгоритмов состоит в том, что операции вывода на экран (относительно «медленные» во многих системах) для каждого пикселя выполняются не более одного раза.

Недостатком является использование динамических структур данных (списков), что сильно усложняет код и требует дополнительной памяти.

Однако для определённых систем использование динамических структур данных нежелательно (вследствие ограниченности ресурса памяти или отсутствия удобных средств разработки программ), в то время как замедление работы из-за частого обращения к видеопамяти некритично.

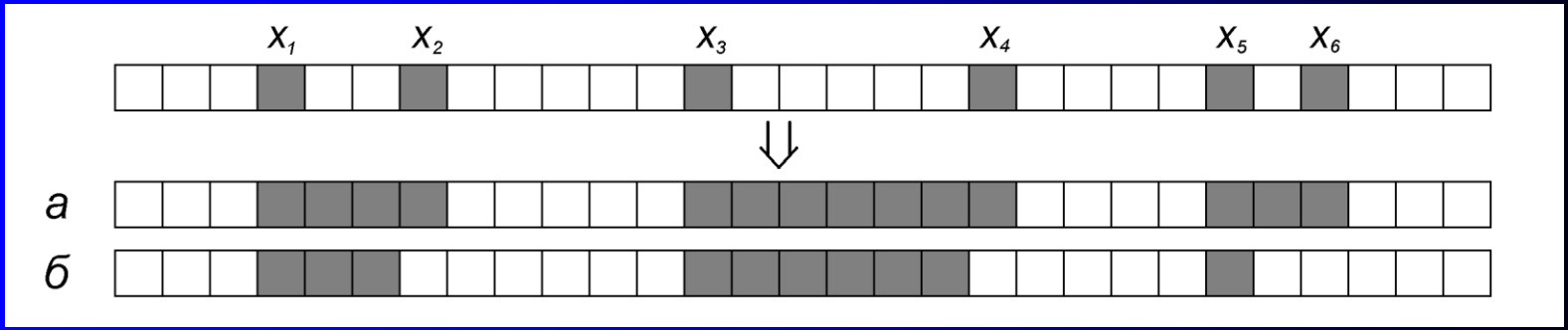
К ним относятся, например, мобильные телефоны и другие портативные устройства, имеющие графический дисплей.

Растровая развёртка многоугольников инверсией (операцией XOR)

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Этот оригинальный алгоритм использует свойства операции XOR.

Пусть контур многоугольника растеризован и выведен на экран. Тогда его закрашивание сводится к заполнению в каждой строке растра всех промежутков вида $[x_{2i-1}, x_{2i}]$, где через x_k обозначены x-координаты «включенных» пикселей в данной строке, упорядоченные по возрастанию.



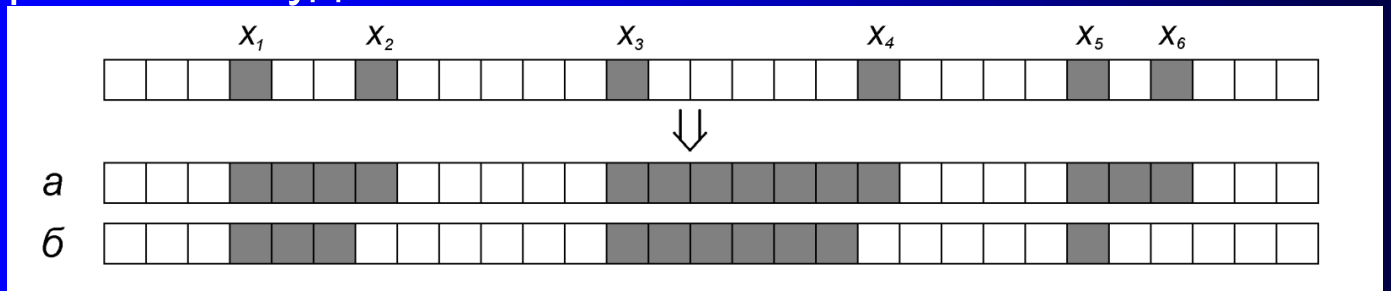
Растровая развёртка многоугольников инверсией (операцией XOR)

Через *I(x,y)* обозначим состояние пикселя с координатами (*x*, *y*): *I(x,y)=1*, если пиксель «включен» и *I(x,y)=0* в противном случае. Нетрудно убедиться, что последовательное выполнение операции

$$I(x+1,y) = I(x+1,y) \ \text{XOR} \ I(x,y)$$

для *x* = 1, 2, 3, …, *X*–1 (где *X* – горизонтальный размер растра) приведет к требуемому результату с той лишь разницей, что последний пиксель в каждом промежутке закрашен не будет.

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0



Растровая развёртка многоугольников инверсией (операцией XOR)

Эта небольшая неточность в большинстве случаев некритична и визуально незаметна. Формально алгоритм записывается так:

```
Растеризовать контур многоугольника и вывести его на экран;
    for(y = 1; y <= Y; y++)
        for(x = 1; x <= X - 1; x++)
I(x + 1, y) = I(x + 1, y) XOR I(x, y);
```

Если же требуется получить результат, соответствующий рис. а, то можно либо после закрашивания повторно вывести на экран контур многоугольника, либо воспользоваться следующей модификацией приведенного алгоритма (называемой «XOR с флагом»):

Растровая развёртка многоугольников инверсией (операцией XOR)

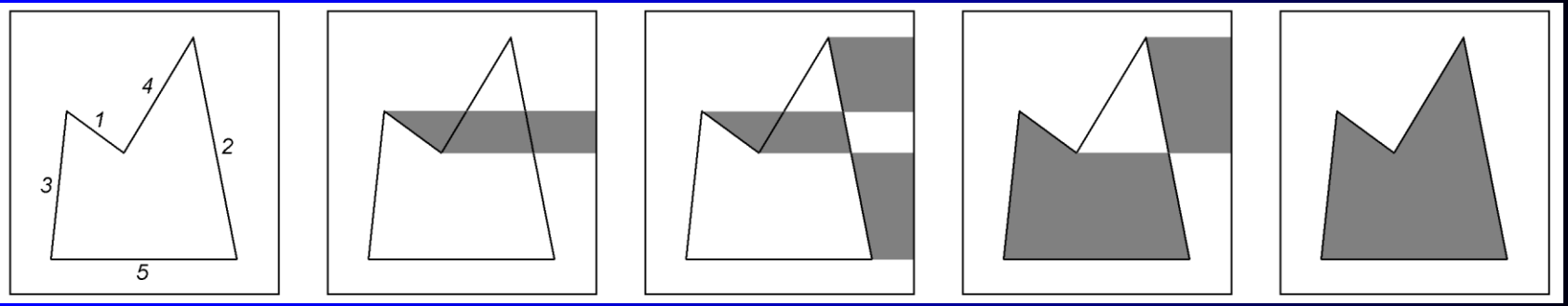
Алгоритм «XOR с флагом»:

```
Растеризовать контур многоугольника и вывести его
на экран;
for(y = 1; y <= Y; y++)
 {
   flag = 0;
     for(x = 1; x <= X; x++)
      {
        if( I(x, y) == 1 ) flag = 1 - flag;
         if( flag == 1 ) I(x, y) = 1;
       }
  }
```

Достоинством алгоритмов XOR является их предельная простота. Недостаток – невозможность работы при наличии посторонних изображений на экране.

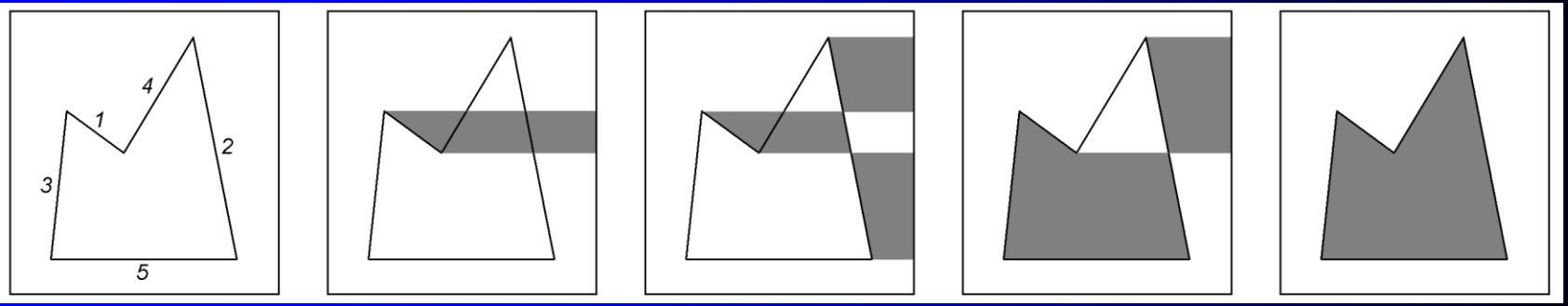
Растровая развёртка многоугольников инверсией (операцией XOR)

Алгоритм заключается в инвертировании цвета всех пикселей, расположенных левее *i*-го ребра, производимом последовательно для *i* = 1, 2, ..., N (порядок нумерации ребер не имеет значения). Горизонтальные ребра при этом игнорируются. Как видно из рис., в результате закрашенными окажутся все внутренние пиксели многоугольника и только они.



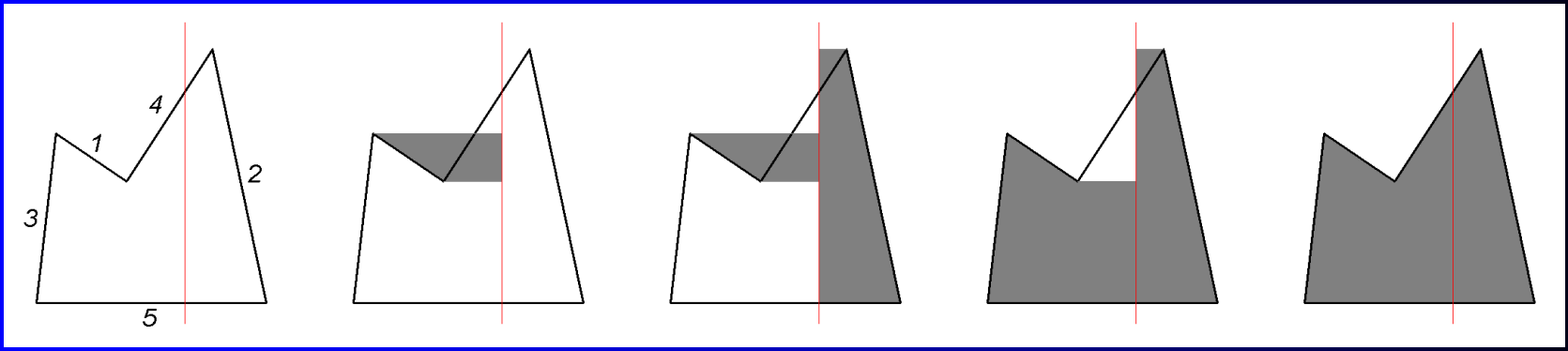
Растровая развёртка многоугольников инверсией (операцией XOR)

К достоинствам данного алгоритма можно отнести его простоту и оригинальность, а так же отсутствие дополнительных структур данных. Недостатком является необходимость выполнения большого числа операций с пикселями (до N операций с каждым пикселем), в том числе и вне многоугольника. В частности, чем больше расстояние между многоугольником и правой границей экрана, тем больше будет совершено «лишних» операций.

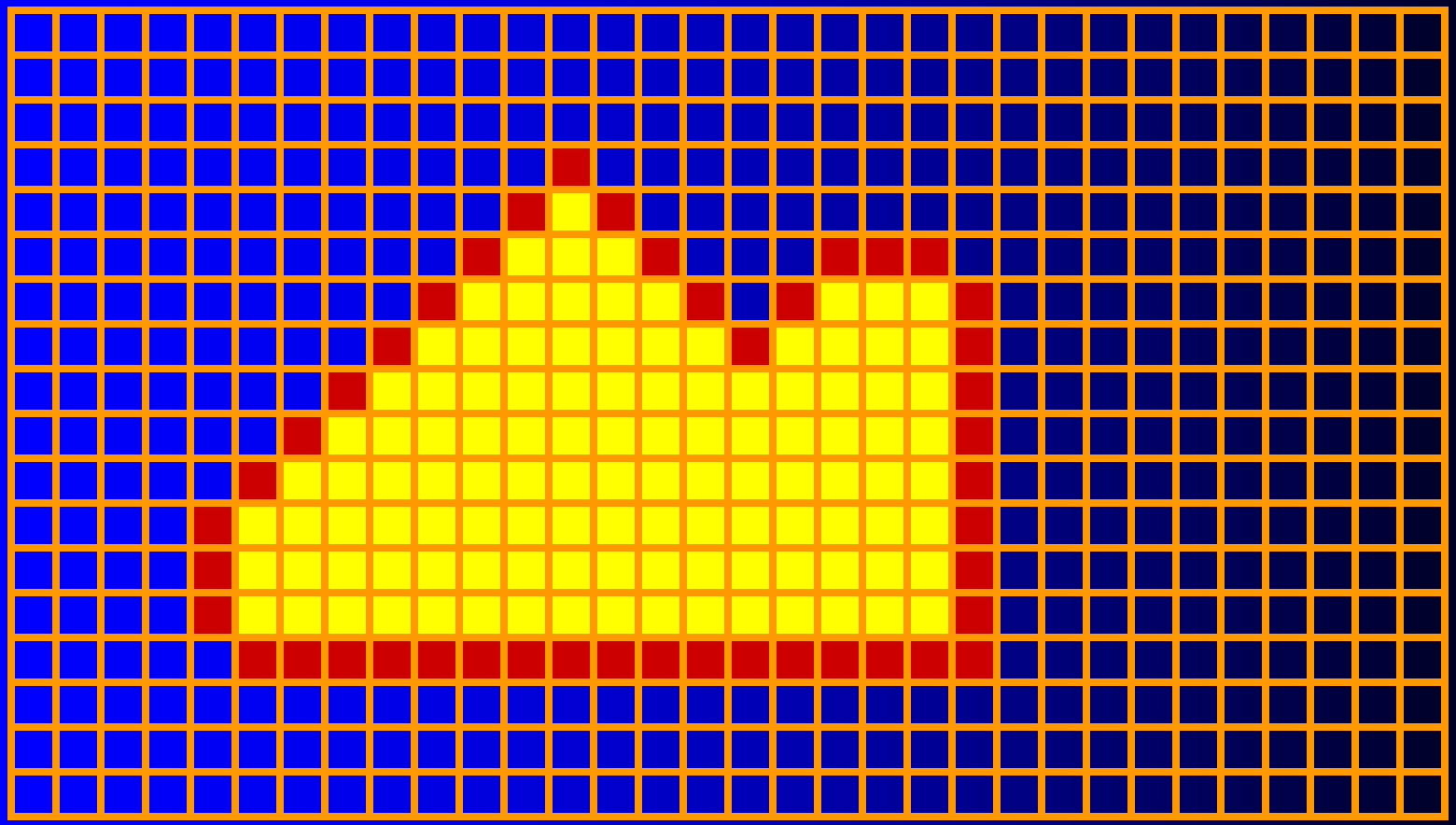


Растровая развёртка многоугольников инверсией (операцией XOR)

От этого недостатка свободна модификация данного алгоритма – «**XOR-2 с перегородкой**». Идея ее заключается в том, чтобы инвертировать область не между ребром и правой границей экрана, а между ребром и вертикальной прямой («перегородкой»), мысленно проведенной в любом удобном месте – например, пересекающей многоугольник.



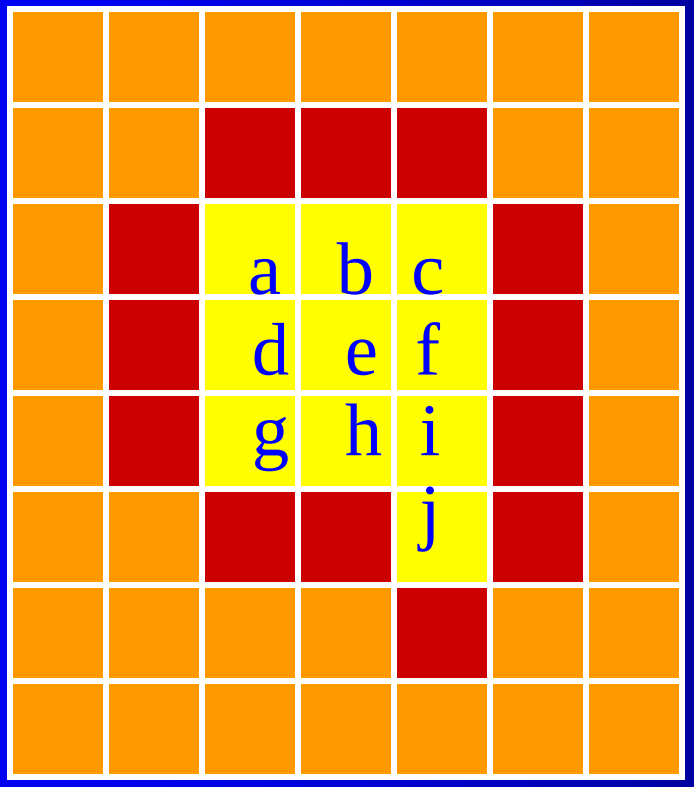
Затравочное заполнение (Flood Filling)

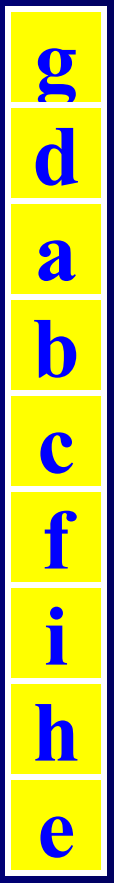


Простейший алгоритм заливки с затравкой

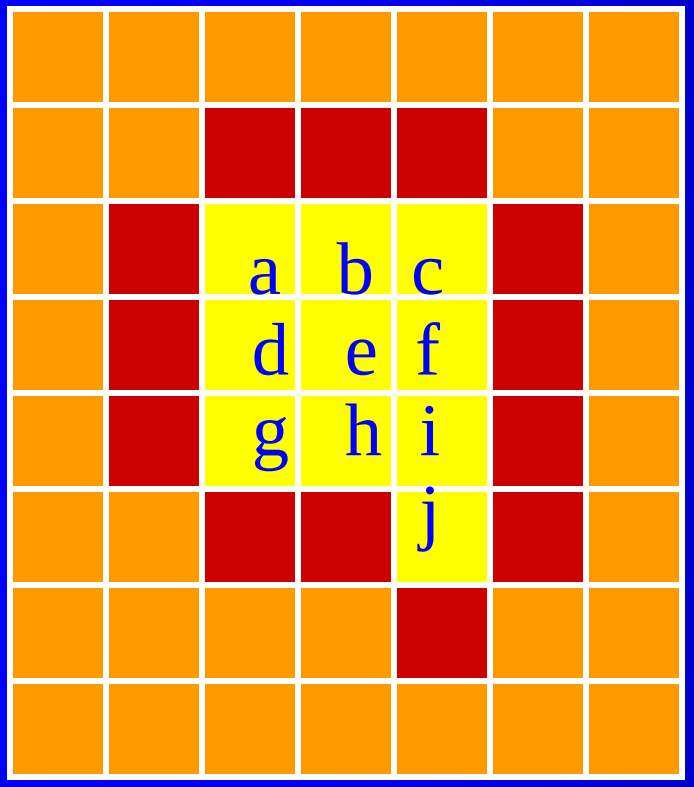
```
void fill(pixel me)
  { pixel tmp; // временная переменная
    { colour(me);
      for  (tmp = each me-neighbour)
       if (!coloured(tmp)) then fill(tmp);
    }
  }
```

Содержимое стека



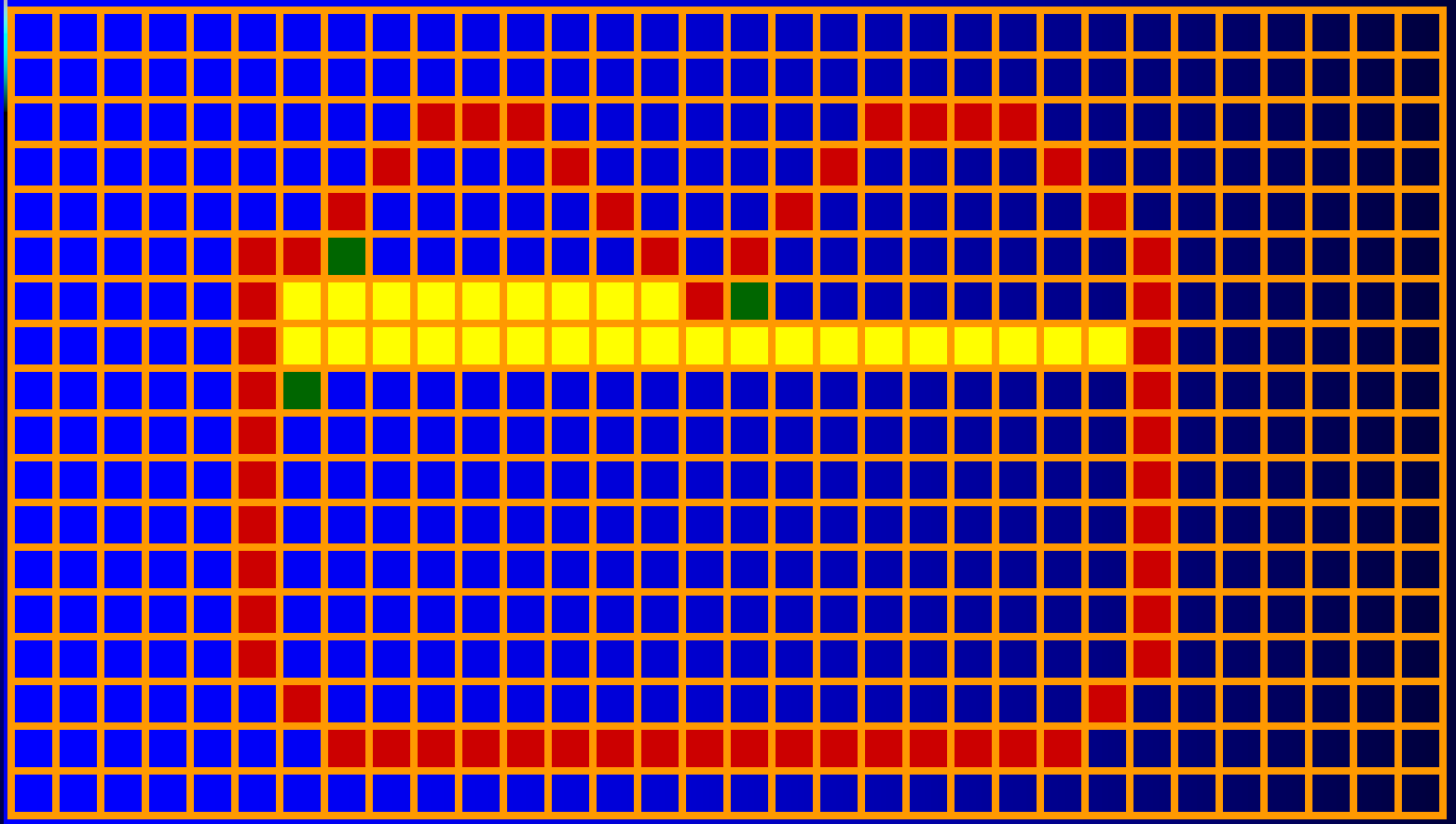


Более эффективный вариант

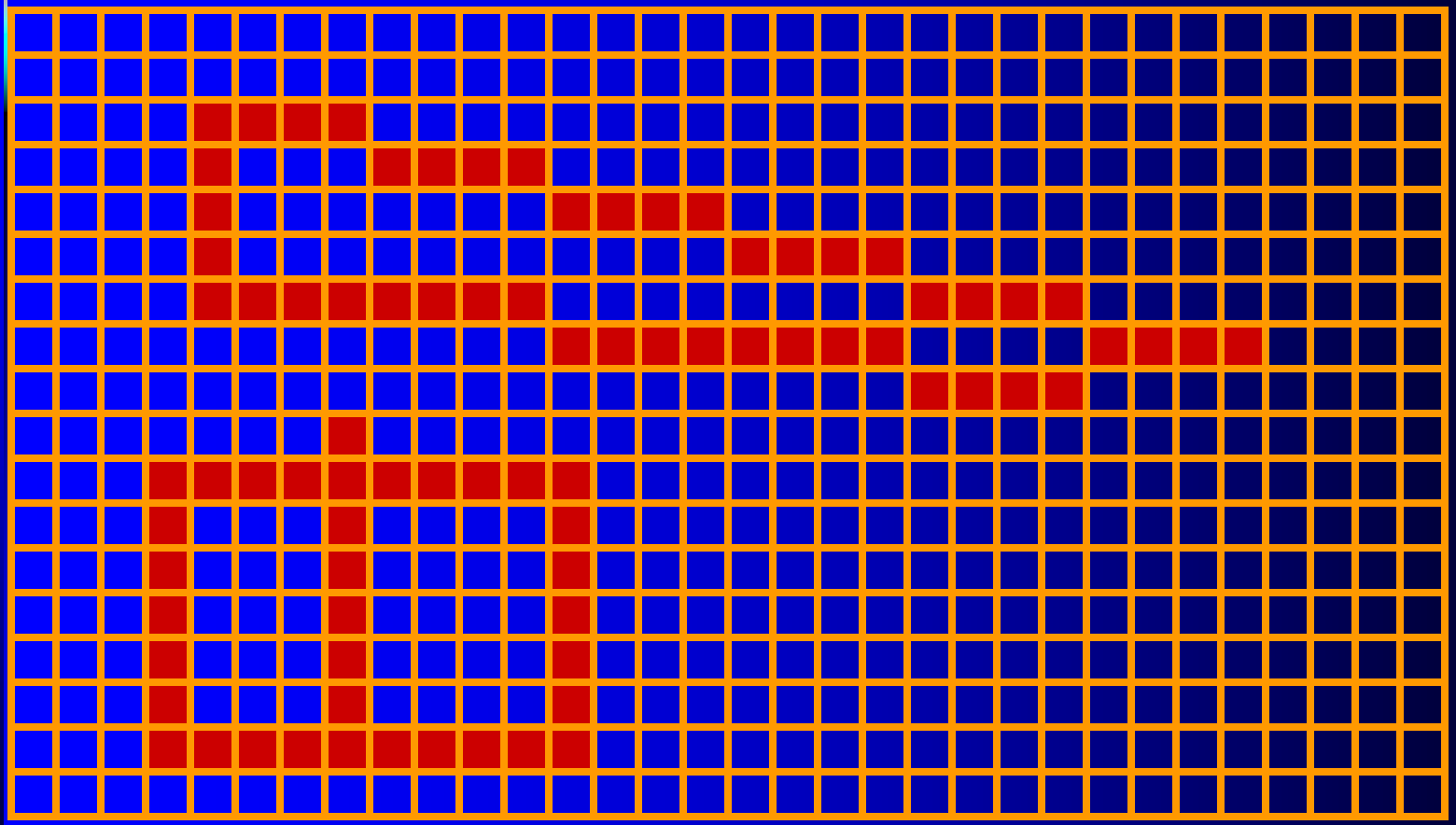


g d h a e i b f j c

Построчный алгоритм заполнения с затравкой

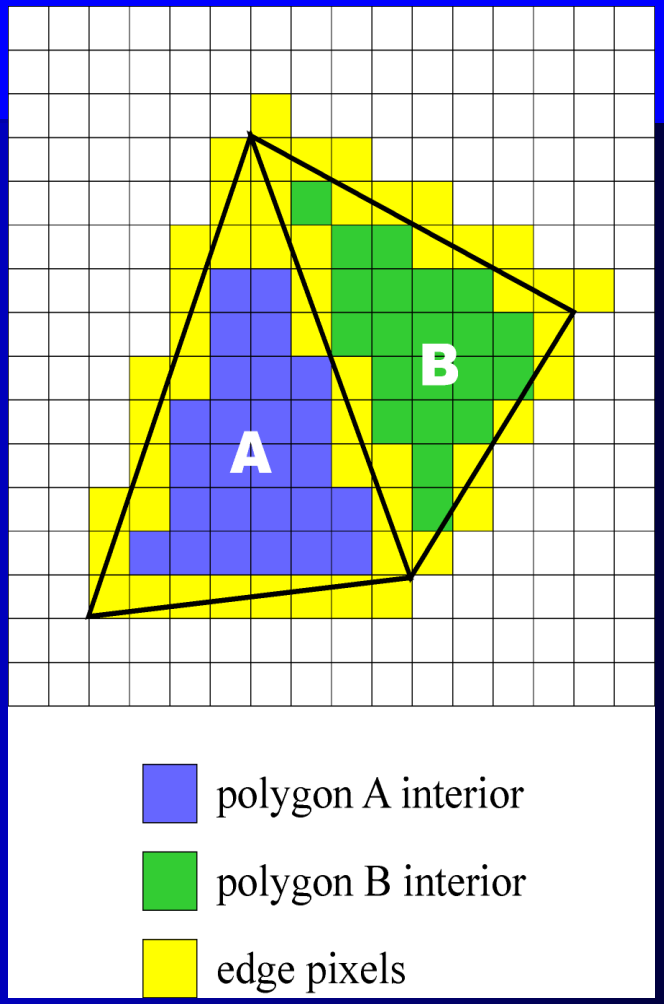


Неизбежные проблемы

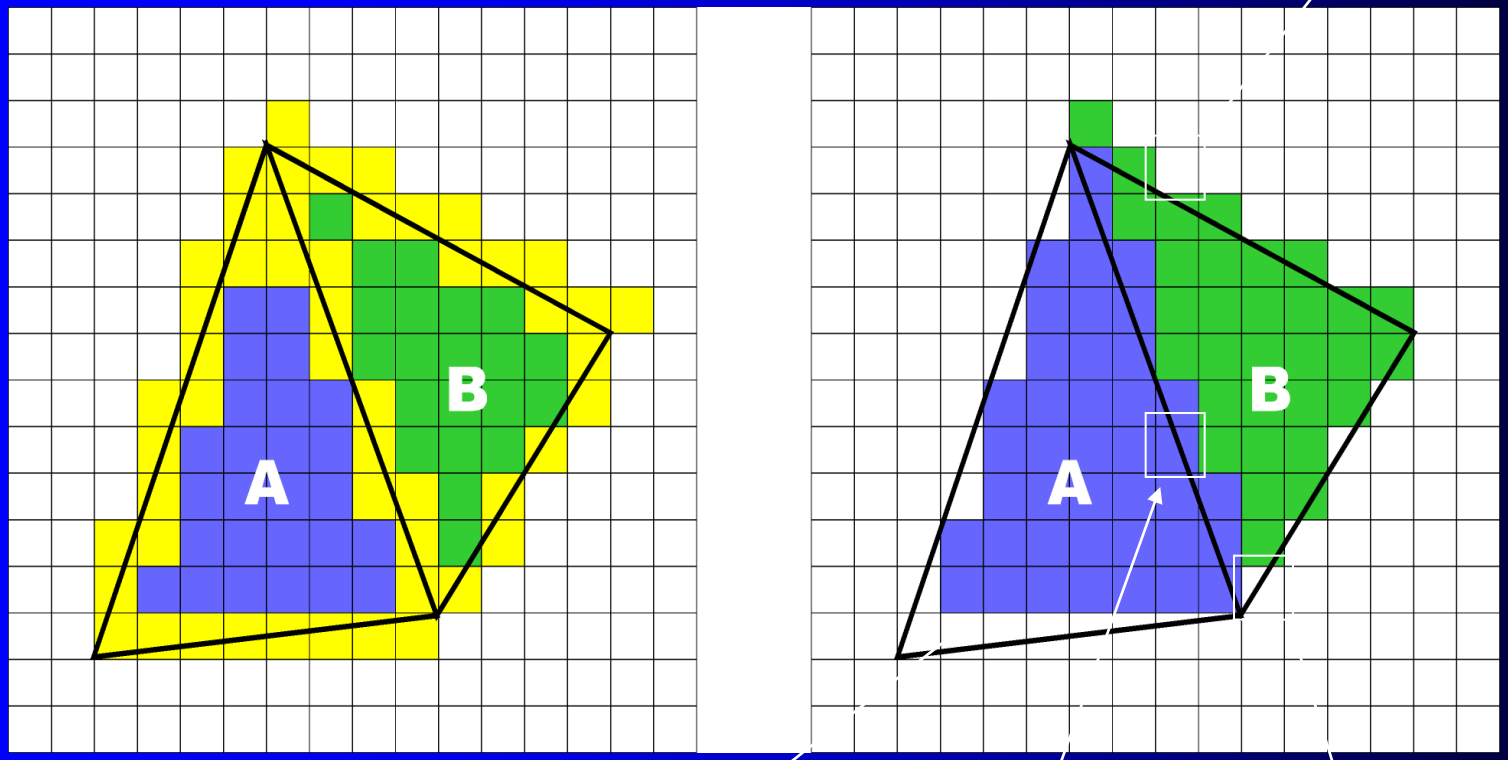


Растеризация многоугольников

- Существуют определённые нюансы по растеризации смежных многоугольников – округление значений границ.
- Плохой результат:
  - Если многоугольники будут делить граничные пиксели
  - Особенно плохо, если многоугольники будут полупрозрачными
  - Оставлять границу между ними так же недопустимо.
- Следует убедиться, что полигоны которые имею общее ребро, не будут иметь общих пикселей.



Растеризация многоугольников



Aha!