Заполнение областей

При выводе изображений часто встречается необходимость вывести на экран не только отдельные линии, но и целые фигуры, внутренняя область которых закрашена произвольным цветом. Алгоритмы, предназначенные для вывода таких областей, называются алгоритмами заполнения (заливки) областей.

Области могут задаваться одним из двух способов:

1. Заданы вершины многоугольника (в таком случае говорят об алгоритмах растеризации многоугольников);
2. Область задана цветом своей границы и внутренней точкой.

Второй способ задания области является более универсальным, т.к. с его помощью можно задать область произвольной формы. Алгоритм заполнения такой области называется заполнением с затравкой. Именно с этого алгоритма мы и начнём рассмотрение методов заполнения областей.

Заполнение с затравкой

Исходными данными для этого алгоритма являются цвет границы области и точка, принадлежащая этой области (т.н. затравочный пиксел).

Для реализации метода заполнения с затравкой нам потребуется структура данных под названием стек. Стек поддерживает две основные операции: поместить элемент в стек и извлечь элемент из стека. При программной реализации самым простым способом создание стека является использование одномерного массива.

Сам алгоритм включает в себя следующие шаги:

1. Поместить затравочный пиксел в стек;
2. Извлечь пиксел из стека;
3. Присвоить пикселу требуемое значение (цвет внутренней области);
4. Каждый окрестный пиксел добавить в стек, если он
    4.1. Не является граничным;
    4.2. Не обработан ранее (т.е. его цвет отличается от цвета границы или цвета внутренней области);
5. Если стек не пуст, перейти к шагу 2.6

При обходе внутренних пикселов может рассматриваться как 4-связная, так и 8-связная окрестность.

Алгоритмы растеризации многоугольников

Пусть задан произвольный многоугольник с вершинами $P_1, P_2, \dots, P_N, P_1$ и требуется отобразить его на экране вместе со всеми внутренними точками. Для удобства будем предполагать, что каждое ребро многоугольника задаётся координатами его концов x_1, y_1 и x_2, y_2 (при этом $y_2 \geq y_1$). Также условимся, что координата x возрастает при движении слева направо, а координата y при движении сверху вниз.

Подавляющее большинство алгоритмов растеризации многоугольников основаны на следующем предположении: любая секущая прямая пересекает границу многоугольника чётное число раз. Это утверждение неверно только в двух случаях:

- Когда секущая прямая содержит ребро;
- Когда секущая прямая содержит вершину, а смежные рёбра лежат по одну сторону от секущей прямой.

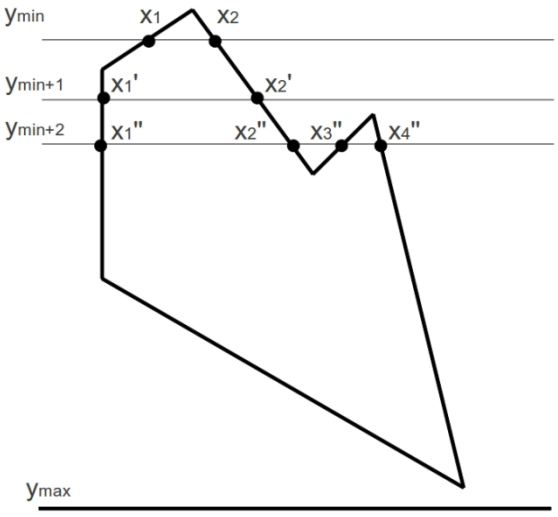
Эти два случая довольно легко обнаружить, поэтому при рассмотрении алгоритмов растеризации многоугольников будем считать, что приведённое выше предположение всегда верно.

Обработка исключительных ситуаций

В первом случае, когда секущая прямая содержит в себе ребро, для борьбы с исключительными ситуациями достаточно вывести только концы этого ребра.

Для исключения остальных случаев можно поступить следующим образом. При растеризации каждого ребра многоугольника будем выводить его нижний конец $(x_2; y_2)$, а верхний конец выведем с помощью операции $I[x_1; y_1] := I[x_1; y_1]\ XOR\ 1$. Это приведёт к тому, что верхние и нижние концы ребёр, попавшие в одну и ту же точку, не будут выведены.

Алгоритм со списком ребёрных точек



Алгоритм, основанный на работе со списком рёберных точек, состоит из трёх основных этапов:

1. На первом этапе растеризуются все негоризонтальные рёбра многоугольника. Для каждого значения y составляется список -координат, закрашенных при растеризации. Например, для рассматриваемого многоугольника мы получим следующие значения (при движении по часовой стрелке):

y_{min}	$x_2; x_1$
y_{min+1}	$x_2'; x_1'$
y_{min+2}	$x_2''; x_3''; x_4''; x_1''$
y_{max}	

2. На втором этапе для каждого значения y списки упорядочиваются по возрастанию. После этого списки будут выглядеть следующим образом:

y_{min}	$x_1; x_2$
y_{min+1}	$x_1'; x_2'$
y_{min+2}	$x_1''; x_2''; x_3''; x_4''$
y_{max}	

3. На третьем этапе для каждого y заполняются все полученные отрезки.

Алгоритм со списком активных рёбер

Алгоритм со списком активных рёбер – это модификация алгоритма со списком рёберных точек. Основное отличие от предыдущего алгоритма в том, что мы на каждом шаге обрабатываем только одну строку.

Для этого организуется специальный список активных рёбер, в котором содержится информация обо всех рёбрах, пересекаемых текущей строкой. Для того чтобы реализовать алгоритм, мы будем для каждого ребра хранить следующие значения:

- $y = [y_1]$
- $dx = \frac{x_2 - x_1}{y_2 - y_1}$
- $x = x_1 + dx * (y - y_1)$

Все такие структуры, описывающие рёбра, мы поместим в список, который назовём $YList$. Все записи в $YList$ мы упорядочим по возрастанию y. Введём ещё один список – список активных рёбер $AList$. Теперь мы можем записать весь алгоритм целиком:

```
AList:=<пусто>;
y:=YList[1].y;
начало цикла
 Добавить в AList рёбра из YList, где ребро.y=y
 Удалить из YList рёбра, у которых ребро.y=y
 Закрасить промежутки между двумя закрашенными пикселами в строке y
 y:=y+1
 для всех рёбер из AList
 begin
  if y>ребро.y2 then
   удалить ребро из AList
  else
  begin
   ребро.x:=ребро.x+ребро.dx
   пока у ребра слева x-координата>ребро.x
    поменять местами рёбра в AList
  end;
 end;
```

пока в AList есть записи

Внутренний цикл пока используется для сохранения упорядоченности рёбер в AList по координате x.

Преимущества рассмотренных алгоритмов растеризации многоугольников состоит в том, что каждый пиксел мы закрашиваем не более одного раза. Это очень важно в системах, где графическая подсистема работает медленно по сравнению со скоростью работы процессора и ОЗУ. К недостаткам этих методов можно отнести их сложность и расход памяти.

Алгоритмы XOR

Построчная XOR-обработка

Этот метод растеризации многоугольников основан на свойствах логической операции исключающего ИЛИ (XOR).

a	b	$a\ XOR\ b$
0	0	0
0	1	1
1	0	1
1	1	0

Этот алгоритм, как и алгоритм со списком рёберных точек, начинается с растеризации границ. После того, как границы построены, закрашивание сводится к заполнению в каждой строке промежутков между двумя закрашенными точками.

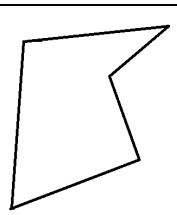
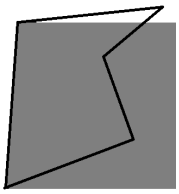
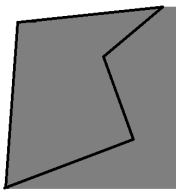
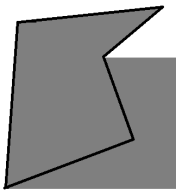
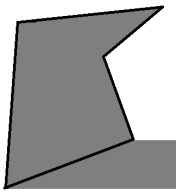
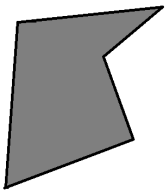
Представим изображение в виде бинарного массива I. Договоримся, что $I[x;y]=1$, когда пиксел закрашен, и $I[x;y]=0$, когда пиксел не закрашен. Легко заметить, что применение операции $I[x+1,y]=I[x;y]XOR\ I[x+1;y]$ ко всем пикселам в строке приведёт к почти идеальному результату. В результате этой операции будут закрашены все промежутки, но последний пиксел в каждом промежутке не будет закрашен. В большинстве случаев эта погрешность несущественна и незаметна, но если требуется получить точный результат, можно после завершения алгоритма повторно вывести на экран границы, или воспользоваться небольшой модификацией этого алгоритма.

```
for y:=1 to YMax do
begin
 fl:=false;
 for x:=1 to XMax do
 begin
  if I[x,y]=1 then
   fl:=not fl;
  if fl then
   I[x,y]:=1;
 end;
end;
```

Преимущество этого алгоритма в его предельной простоте и высокой скорости. Недостаток в том, что алгоритм не может работать, при наличии посторонних изображений.

Алгоритм XOR для граней

Метод XOR для граней описывается следующим простым алгоритмом: Для каждого ребра в многоугольнике инвертируются цвета всех пикселов, расположенных правее этого ребра. При этом порядок обхода рёбер не имеет значения. В таблице приведены шаги этого алгоритма (движение по часовой стрелке):

1	2	3	4	5	6
					

Недостаток этого алгоритма – высокие временные затраты, так как некоторые пикселы обрабатываются более одного раза. Кроме того, чем больше расстояние от изображения до правой границы области экрана, тем больше будет совершено лишних операций.

Алгоритм XOR для граней с перегородкой

От некоторых из недостатков свободен модифицированный вариант алгоритма XOR для граней. Он называется алгоритм XOR для граней с перегородкой. Его идея заключается в том, чтобы инвертировать область не между ребром и границей экрана, а между ребром и специальной вертикальной линий (т.н. перегородкой). Чаще всего перегородка проводится так, чтобы она пересекала многоугольник. Шаги работы алгоритма приведены в таблице.

1	2	3	4	5	6