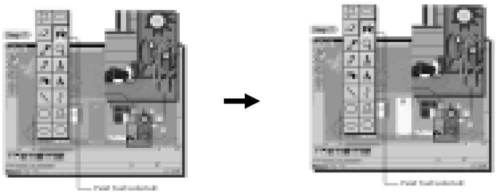
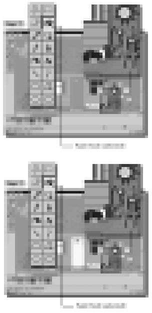
Flood-fill



Flood-Fill

- Used in interactive paint systems.
- The user specify a seed by pointing to the interior of the region to initiate a flood operation



Recursive Flood-Fill

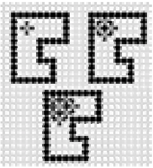
- Fill a image-space region with some intensity (color) value
- How to define the region?
- Fill Until vs. Fill While
- 4-connectivity vs. 8-connectivity

Flood-Fill from Seed

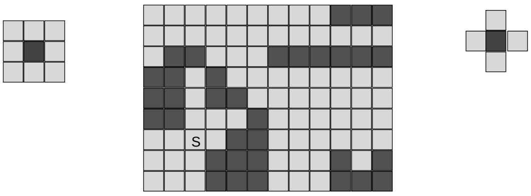
- Start from the seed and floods the region until a boundary is met.

A simple recursive algorithm can be used:

```
void floodFill(int x, int y, int fill, int old)
{
    if ((x < 0) || (x >= width)) return;
    if ((y < 0) || (y >= height)) return;
    if (getPixel(x, y) == old) {
           setPixel(fill, x, y);
           floodFill(x+1, y, fill, old);
           floodFill(x, y+1, fill, old);
           floodFill(x-1, y, fill, old);
           floodFill(x, y-1, fill, old);
    }
}
```



8-connected vs. 4-connected



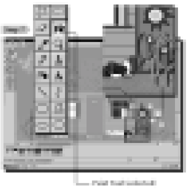
An 8-connected flood is able to flood through corners that a 4-connected flood cannot.

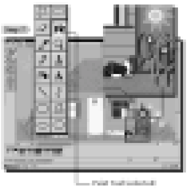
Recursive Flood-Fill Algorithm

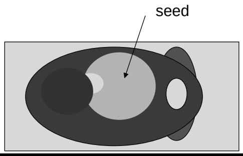
The algorithm is very simple, however it is:

• highly recursive - requiring a huge number of procedural calls;
• can cause recursion stack to overflow
• no mechanism to determine whether the visited pixels have been tested before

Fill Until vs. Fill While







Flood Until

```
void floodUntil(int x, int y, int n_color, int
B_color)
{
    if ((x < 0) || (x >= width)) return;
    if ((y < 0) || (y >= height)) return;
    c = getPixel(x, y);
    if (c != n_color && c != B_color) {
           setPixel(new_color, x, y);
           floodFill(x+1, y, n_color B_color);
           floodFill(x, y+1, n_color B_color);
           floodFill(x-1, y, n_color B_color);
           floodFill(x, y-1, n_color B_color);
    }
}
```

Flood While

```
void floodWhile(int x, int y, int n_color, int
old)
{
    if ((x < 0) || (x >= width)) return;
    if ((y < 0) || (y >= height)) return;
    if (getPixel(x, y) == old) {
           setPixel(fill, x, y);
           floodFill(x+1, y, n_color, old);
           floodFill(x, y+1, n_color, old);
           floodFill(x-1, y, n_color, old);
           floodFill(x, y-1, n_color, old);
    }
}
```

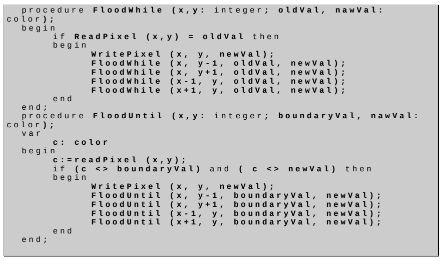
With global variables

```
void floodWhile(int x, int y)
{
    if ((x < 0) || (x >= width)) return;
    if ((y < 0) || (y >= height)) return;
    if (getPixel(x, y) == old_color) {
           setPixel(n_color, x, y);
           floodFill(x+1, y);
           floodFill(x, y+1);
           floodFill(x-1, y);
           floodFill(x, y-1);
    }
}
```

Use a stack

```
Queue q = Ø
Add the seed to q
While(!q.empty()) {
  P = q.pop();
  For (x = P’s neighboring pixels) {
    If (getPixel(x) == old) {
      setPixel(x, new_color);
      q.push(x);
    }
  }
}
```

While vs. Until



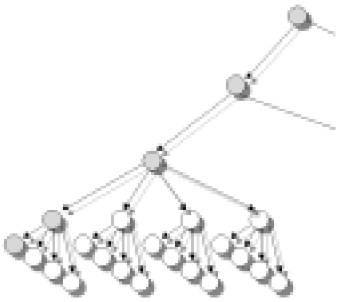
```
   procedure FloodWhile (x,y: integer; oldVal, nawVal:
color);
   begin
        if ReadPixel (x,y) = oldVal then
        begin
              WritePixel (x, y, newVal);
              FloodWhile (x, y-1, oldVal, newVal);
              FloodWhile (x, y+1, oldVal, newVal);
              FloodWhile (x-1, y, oldVal, newVal);
              FloodWhile (x+1, y, oldVal, newVal);
        end
   end;
   procedure FloodUntil (x,y: integer; boundaryVal, nawVal:
color);
   var
        c: color
   begin
        c:=readPixel (x,y);
        if (c <> boundaryVal) and ( c <> newVal) then
        begin
              WritePixel (x, y, newVal);
              FloodUntil (x, y-1, boundaryVal, newVal);
              FloodUntil (x, y+1, boundaryVal, newVal);
              FloodUntil (x-1, y, boundaryVal, newVal);
              FloodUntil (x+1, y, boundaryVal, newVal);
        end
   end;
```

Serial Recursion is Depth-First

So the fill algorithm will continue in one direction until a boundary is reached.

It will then change directions momentarily and attempt to continue back in the original direction.

Potential problem of stack overflow. How to avoid it?



Breath-first Traversal

```
Queue q = Ø
Add the seed to q
While(!q.empty()) {
  P = q.removefirst();
  For (x = P’s neighboring pixels) {
    If (getPixel(x) == old) {
      setPixel(x, fill);
      q.insert(x);
    }
  }
}
```

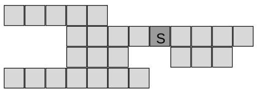
Recursive Flood-Fill Algorithm

- Can also have an “until” version, defining region by boundary
- Recursive flood-fill is somewhat blind and many pixels may be retested several times
- Tag a pixel with a direction and avoid redundant calls…
- Row coherence can improve performance dramatically

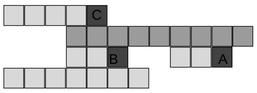
Row Coherence

```
Push address of seed pixel onto stack
while(stack is not empty)
{
  Pop the stack to provide next seed
  Fill in the run defined by the seed
  In the adjacent rows (above and below)
           find the reachable interior runs, and
            push the address of their rightmost pixels
}
```

Floodfill in runs

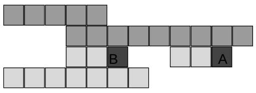


Floodfill in runs



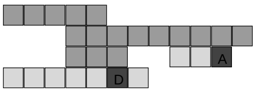
C
B
A
Stack

Floodfill in runs



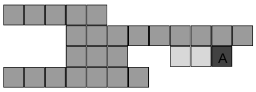
B
A
Stack

Floodfill in runs



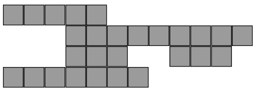
D
A
Stack

Floodfill in runs



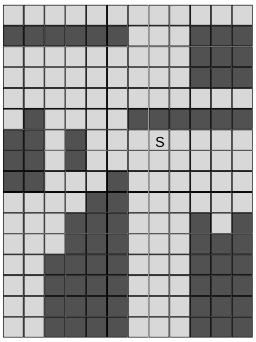
A
Stack

Floodfill in runs

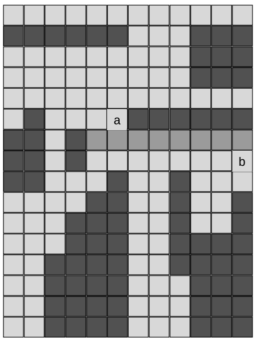


Stack

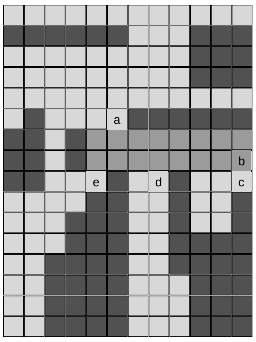
Row Coherence



Row Coherence

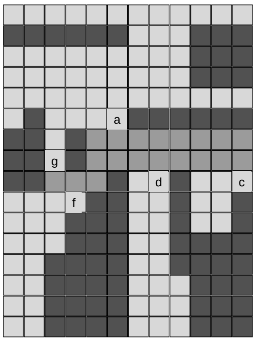


Row Coherence



Row Coherence

The Stack then contains: a,c,d,f,g



```
Flood Fill Algorithm

procedure Fill ( x, y : integer; oldVal, newVal: color);
begin
     push (x, y);
     while stack is not empty
     begin
           pop (x, y);
           open_up = FALSE;
           open_down = FALSE;
           while color [x--, y] == oldVal;
                             /* move to most left pixel */
                 while color [x++,y] == oldVal
                 begin
                       color [x,y] = newVal;
                       if open_up == FALSE
                       begin
                             if color [x, y+1] == oldVal
                             begin
                                   push (x, y+1);
                                   open_up = TRUE;
                             end;
                       end
                       else if color [x, y+1] <> oldVal
                             open_up = FALSE;
                       if open_down == FALSE
                       begin
                             if color [x, y-1] == oldVal
                             begin
                                   push (x, y-1);
                                   open_down = TRUE;
                             end;
                       end
                       else if color [x, y-1] <> oldVal
                             open_doun = FALSE;
                 end;
           end;
     end;
end;
```