Erik Wohlrab
Jeremy Kescher

Flood fill

...is an algorithm which searches an area of connected nodes. If the connected nodes have a target value (which is to be replaced by another value), the flood fill algorithm will start “flooding” all nodes in every direction (therefore spreading that new value).

As long as the nodes are not interrupted and have the same value, the flood fill algorithm will spread out over them.

This is made possible through the algorithms recursive structure (although the example below is still iterative):

```
while (pixels.Count > 0)
{
    Point a = pixels.Dequeue();
    if (a.X < bmp.Width && a.X > 0 &&
            a.Y < bmp.Height && a.Y > 0)
    {
        if (bmp.GetPixel(a.X, a.Y) == targetColor)
        {
            bmp.SetPixel(a.X, a.Y, replacementColor);
            pixels.Enqueue(new Point(a.X - 1, a.Y));
            pixels.Enqueue(new Point(a.X + 1, a.Y));
            pixels.Enqueue(new Point(a.X, a.Y - 1));
            pixels.Enqueue(new Point(a.X, a.Y + 1));
            pictureBox1.Refresh();
        }
    }
}
```

If the target value is given the algorithm will spread to new nodes.

Scanline fill

...uses the same principle of spreading replacement values. If the target value is given, but in contrast to flood fill, which spreads in every direction at once it only inspects the neighboring lines (previous and next). Through this the number of recursive lookups is reduced, which essentially halves the amount of time taken to fill an area.

Uses of fill

Filling algorithms are commonly used when a delimited area has to be filled with e.g. a different color (Paint programs). They are also used with pathfinding in games or some variants even in real-life route calculation to find the shortest route to a destination.

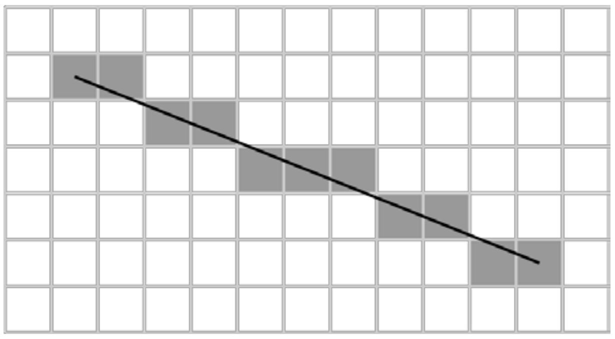
Fills are best described using animations. To see how it works, visit:

https://kescher.at/fill-anim

Erik Wohlrab
Jeremy Kescher

Bresenham’s line algorithm

...is an algorithm that takes two arbitrary points and draws a line between them. To do this, it calculates intermediate points between the starting point and ending points. This makes the algorithm suitable for drawing lines on low-resolution screens or grids, such as in programs like https://kescher.at/multicube ;)



The algorithm is essentially a for loop which has a percentage attached to it:

```
for (double percentage = 0.0; percentage <= 1.0; percentage += step)
{
    // ...
}
```

and each iteration, it calculates the two coordinates of a new intermediate point:

```
Xᵢ = X₁ + percentage * (X₂ - X₁)
Yᵢ = Y₁ + percentage * (Y₂ - Y₁)
```

step depends on the accuracy needed for the line draw. If the line is 100 [pixels] long, then step should be 1 / 100. If the line is 1000 pixels long, it should be 1 / 1000. Thus, another formula that has to be calculated just once before the loop arises:

```
dx = XB - XA
dy = YB - YA
step = sqrt(dx² + dy²)
```

Doing all this, you have everything you need to draw an accurate line between two points.

Uses

As the fill algorithms previously, its main purpose is found in painting programs. However, you can also find it in a few games where it is can be used for grid-based collision detection, mostly shooters (2D, but there allegedly are some 3D implementations as well).

All programs and other materials for this presentation can be easily found at:

https://kescher.at/lines-and-fills