Low level graphics software

- Points, lines and pixel-level things.
- A bit of history and some useful techniques

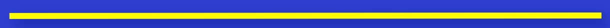
Line and poly-lines

Drawline 0 0 1 0 √

Drawline 1 0 1 1

Drawline 1 1 0 1

Drawline 0 1 0 0



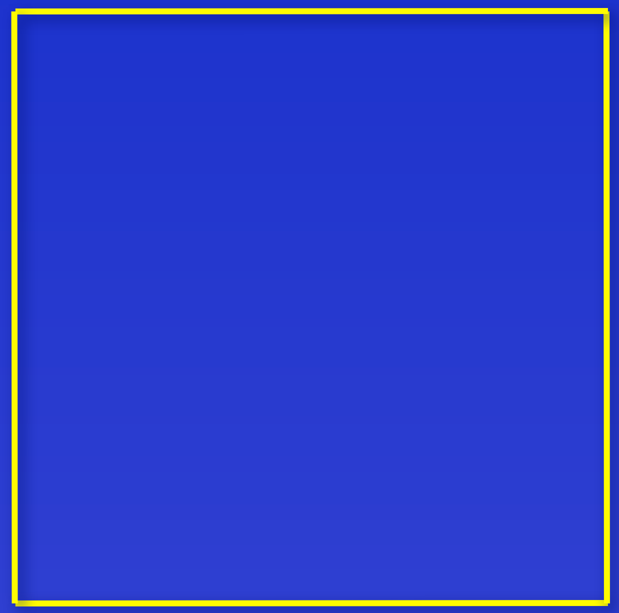
Line and poly-lines

Drawline 0 0 1 0 √

Drawline 1 0 1 1 √

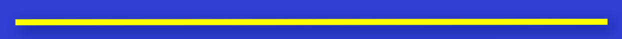
Drawline 1 1 0 1 √

Drawline 0 1 0 0 √



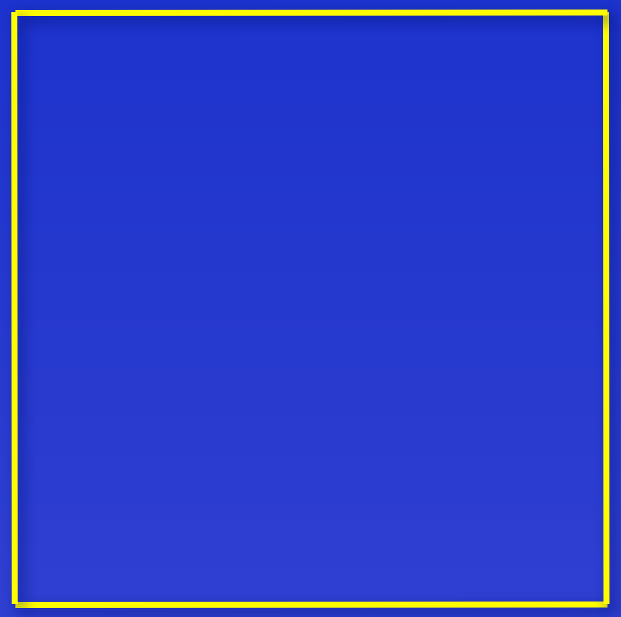
Poly-lines

```
glBegin(GL_LINES);
glVertex2i(0, 0);
glVertex2i(1, 0);    √
glVertex2i(1, 1);
glVertex2i(0, 1);
glVertex2i(0, 0);
```



Poly-lines

```
glBegin(GL_LINES);
glVertex2i(0, 0);
glVertex2i(1, 0);    √
glVertex2i(1, 1);    √
glVertex2i(0, 1);    √
glVertex2i(0, 0);    √
```

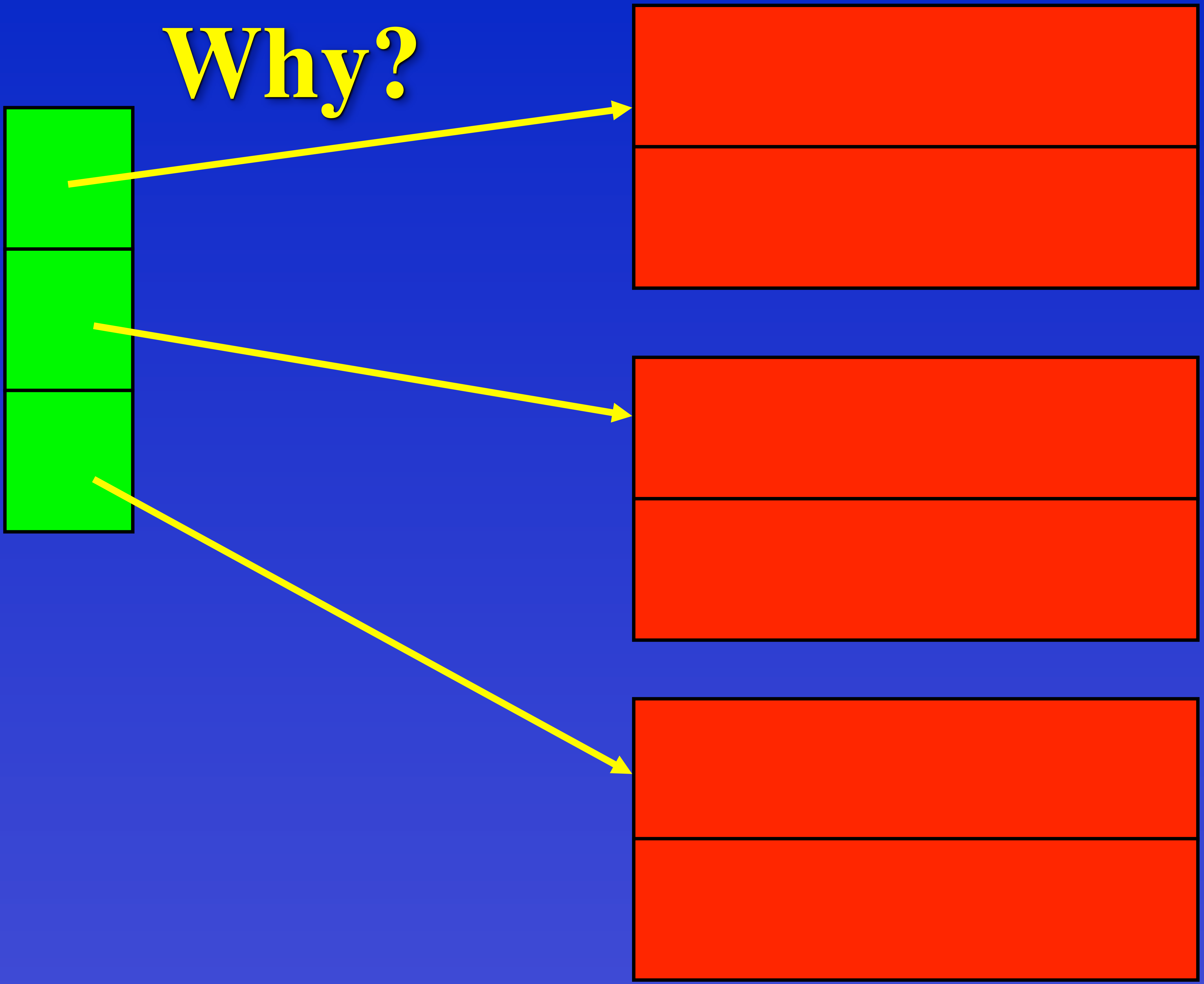


Polygon data structure

```
typedef struct { double x, y; } point;
typedef point *triangle[3];
```

```
Type point = record x, y: real end;
        pstore = ^point;
        triangle = array [1 .. 3] of pstore;
```

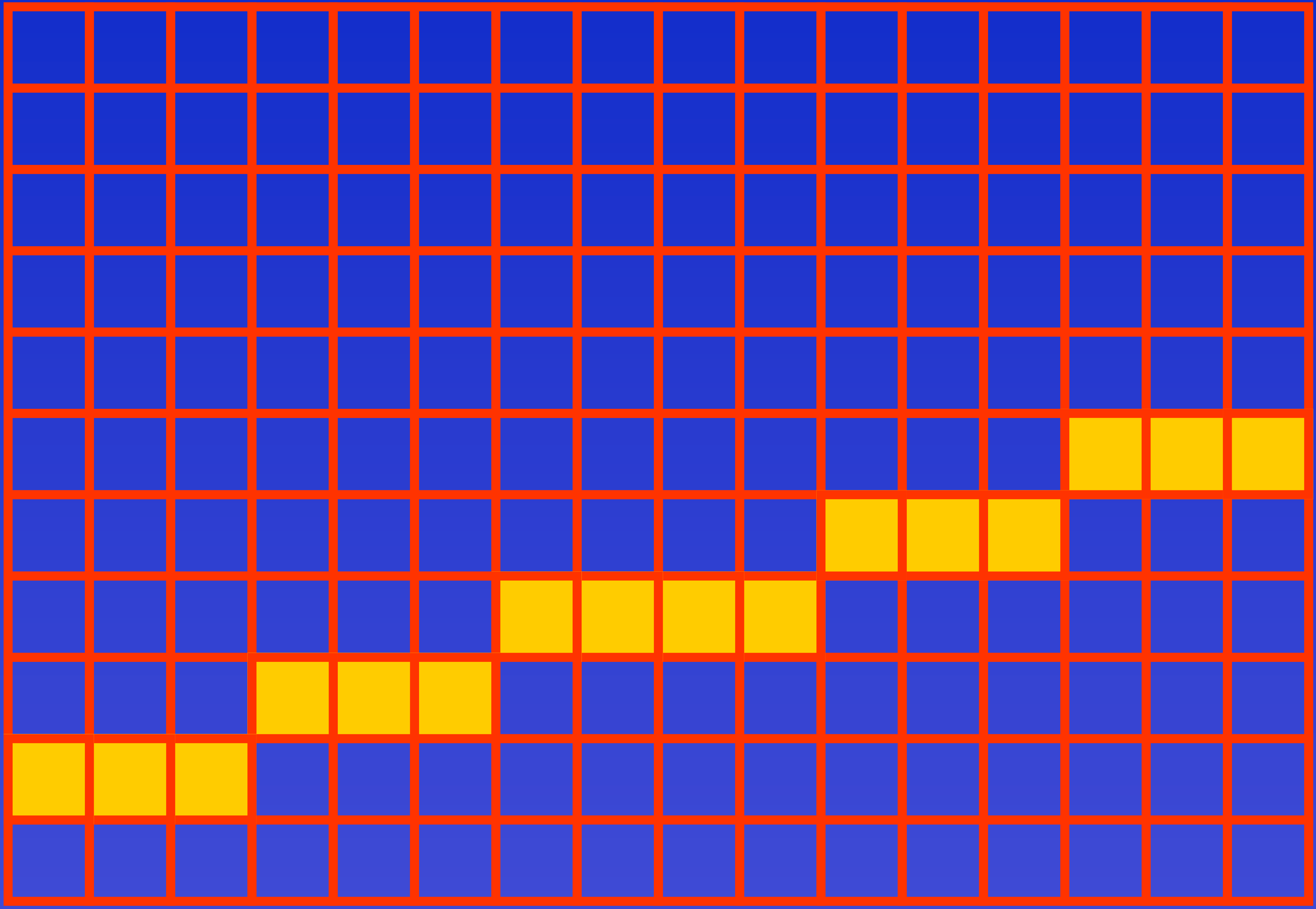
Why?



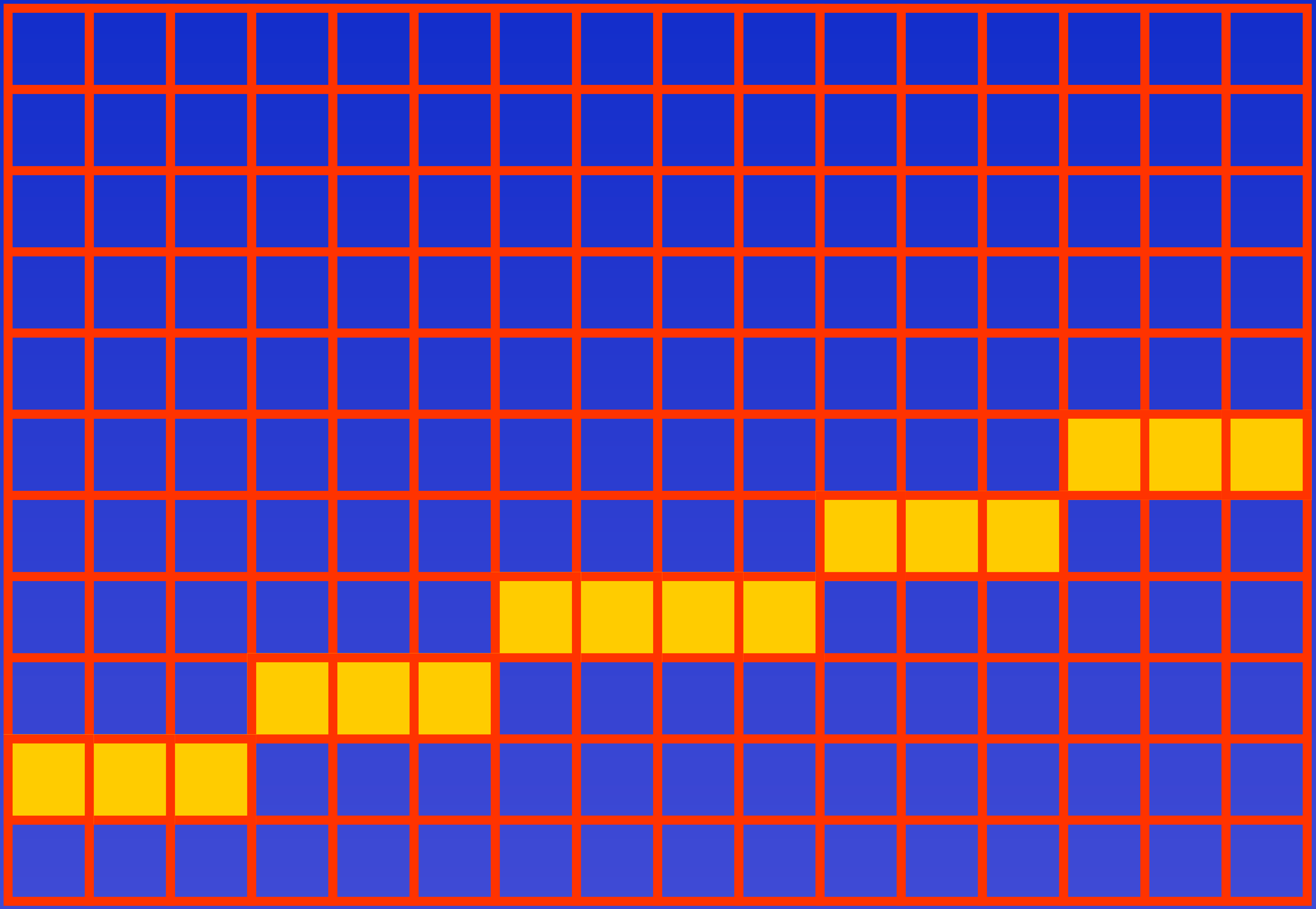
Why not arrays of vertices?

- Pointers are smaller than vertices.
- Each vertex appears only once.
- The same vertex can appear logically in more than one triangle.

Drawing lines at the pixel level



16 x 5 Pixel Example



How Lines are Drawn

- So we increment y every 16/5 steps
- But 16/5 is not a whole number
- How do we choose the best pattern?

Bresenham 1965

- Use relative coordinates
  - solve restricted problem first

- rx >= ry and ry >= 0
- use running error d
- every loop d := d - ry; x := x+1;
- sometimes d := d + rx; y := y+1;

```
d = rx / 2;
incr = rx - ry;
for (i = 1; i <= rx; i++)
{   x = x+1;
    if (d < ry)
    {   y = y+1;
        d = d+incr;
    } else
        d = d - ry;
    pixel[x] [y] = colour;
}
```

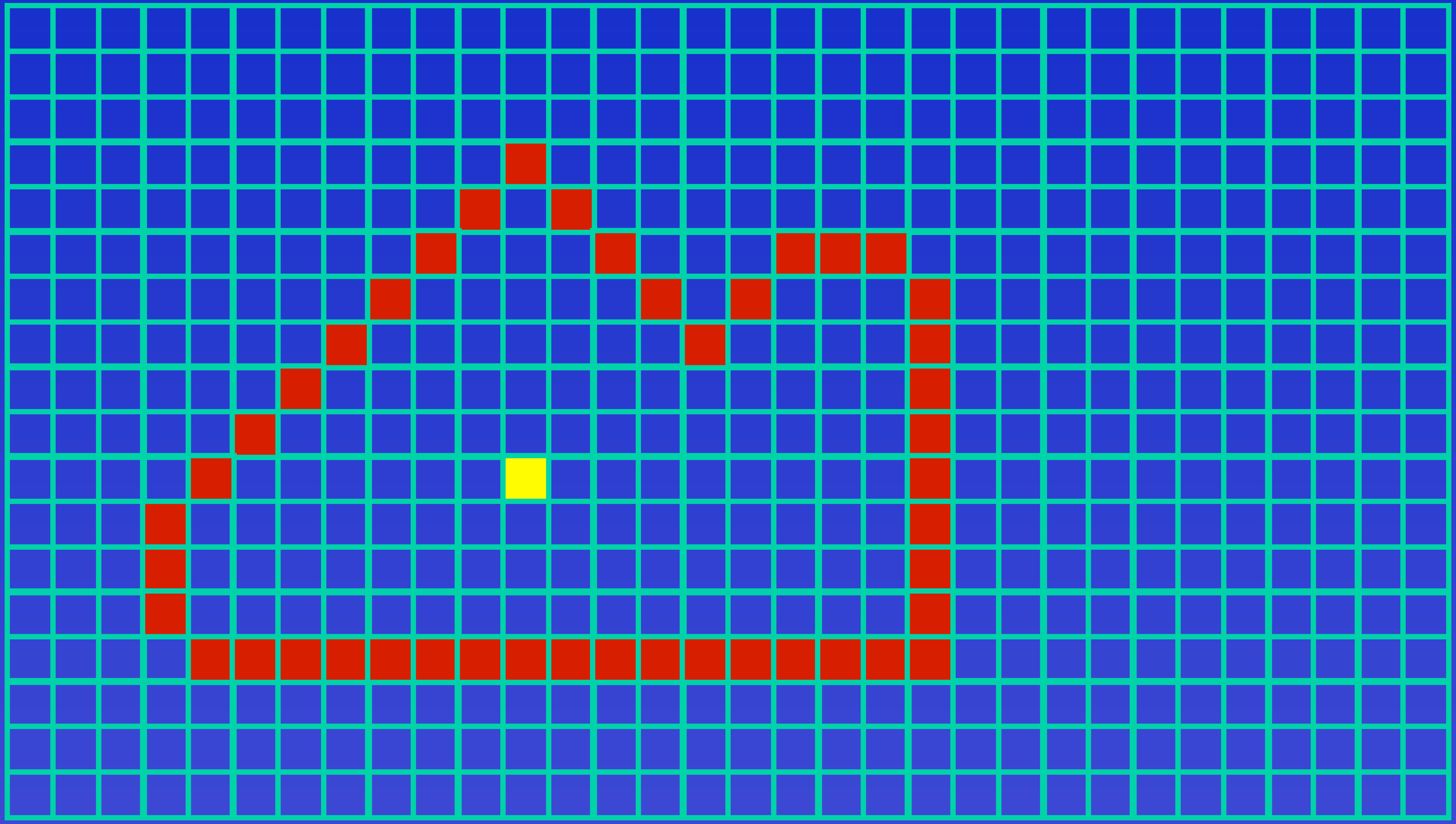
How does it work?

- Basically doing division by repeated subtraction operations.
- d is a running error term. Whenever the error is big enough we do ++y and thus reduce the error.

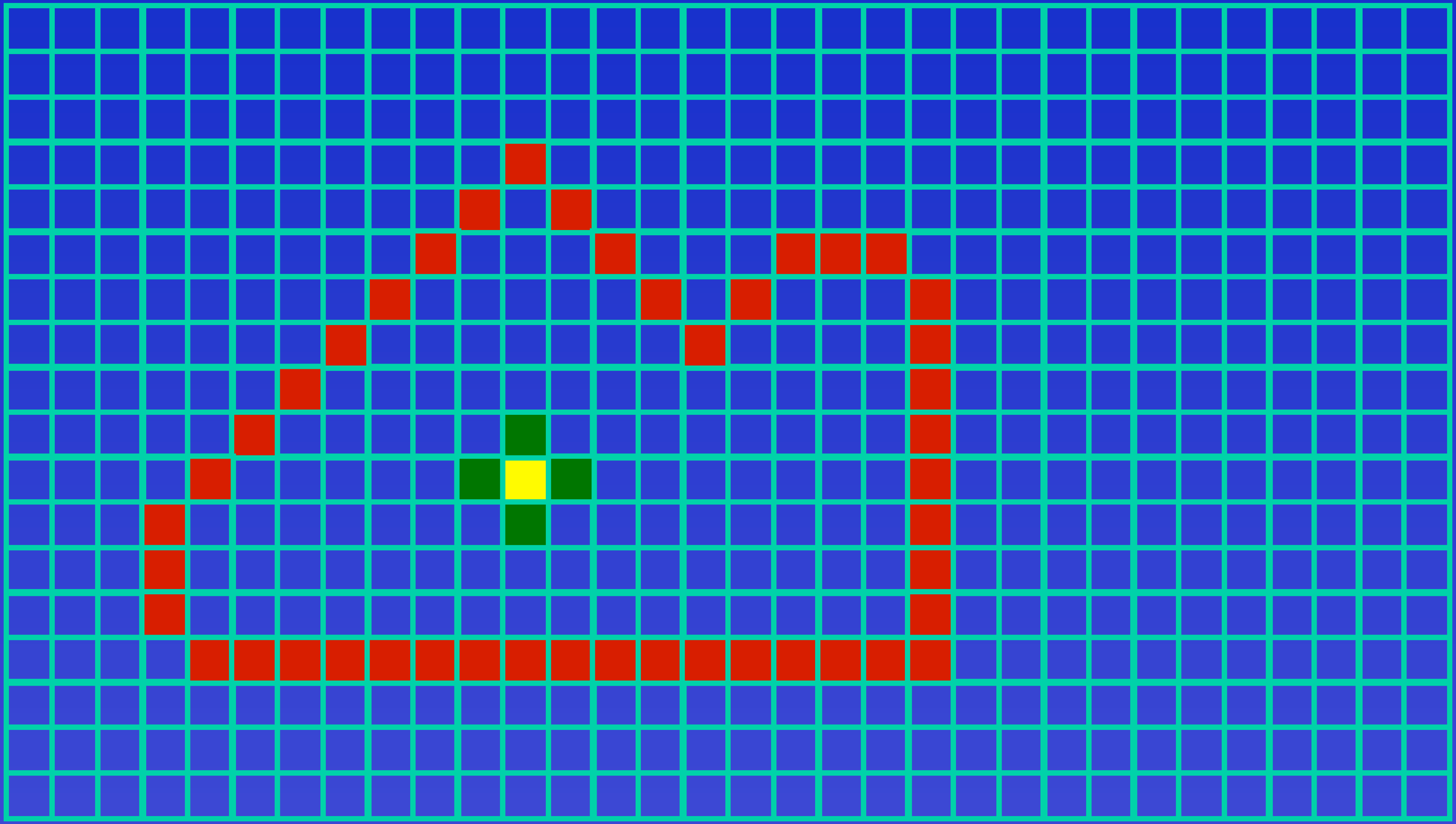
Filled Shapes

- We look at two approaches:
  - Flood Filling
  - Scan lines

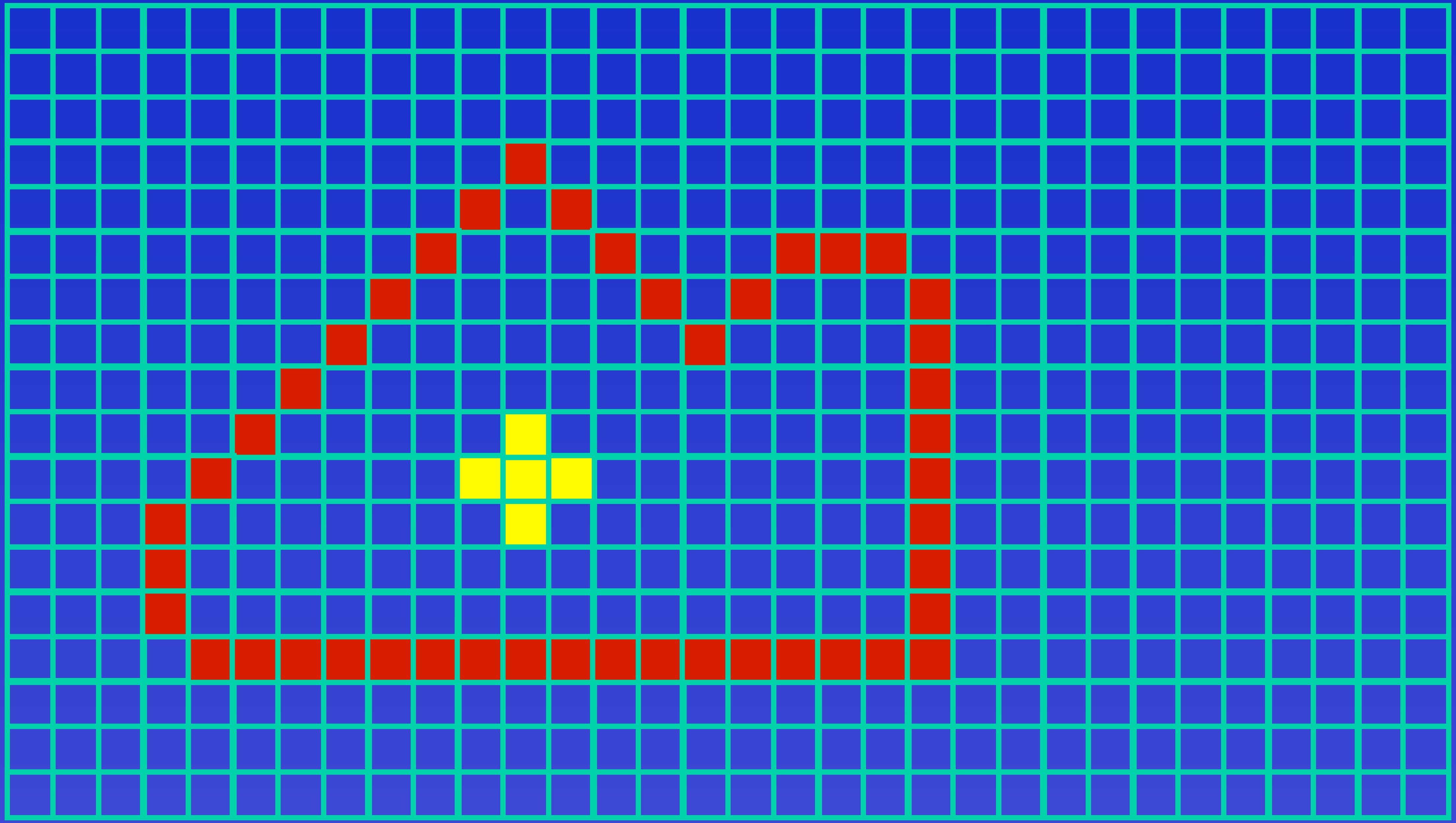
Flood Filling



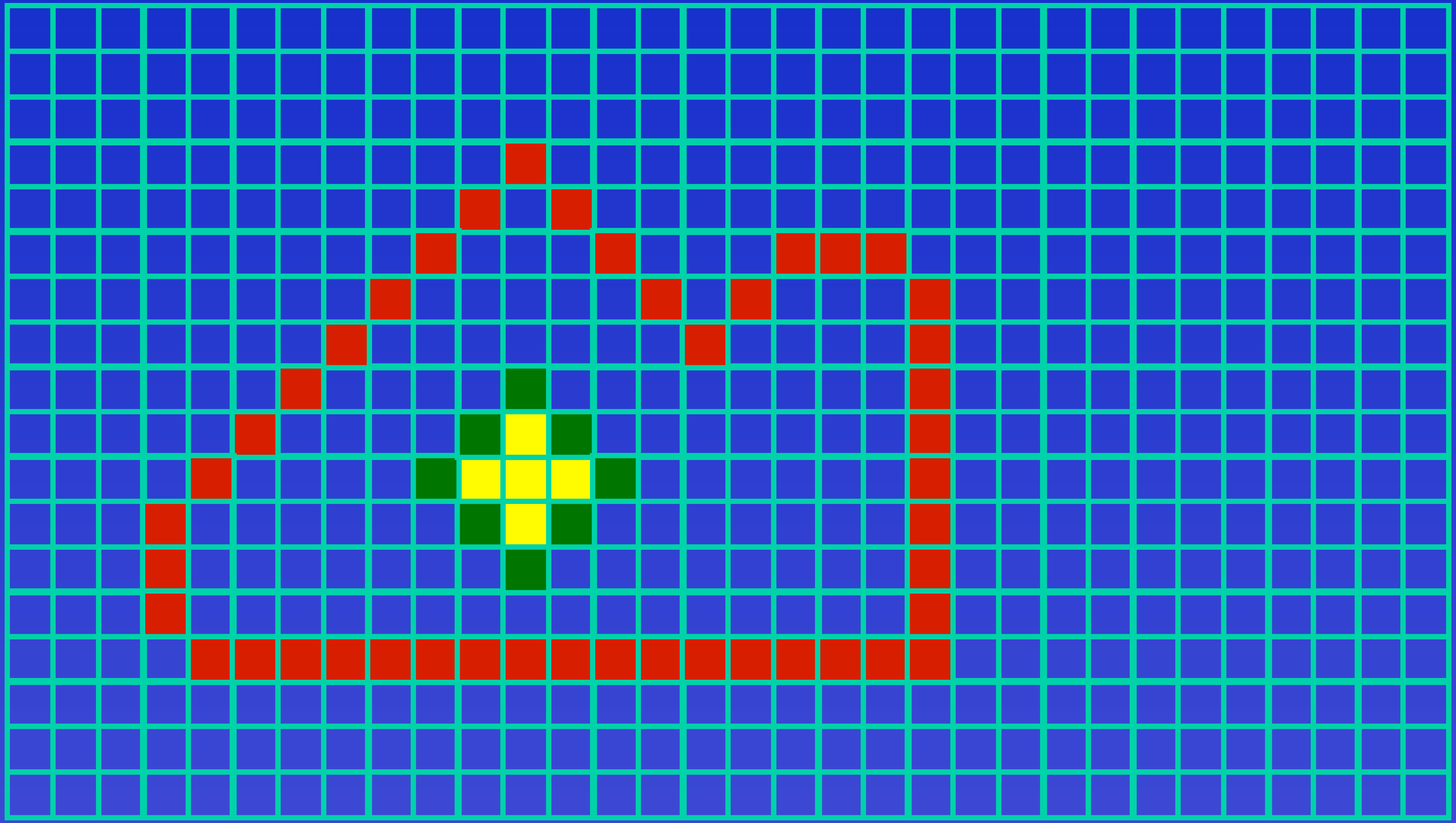
Flood Filling



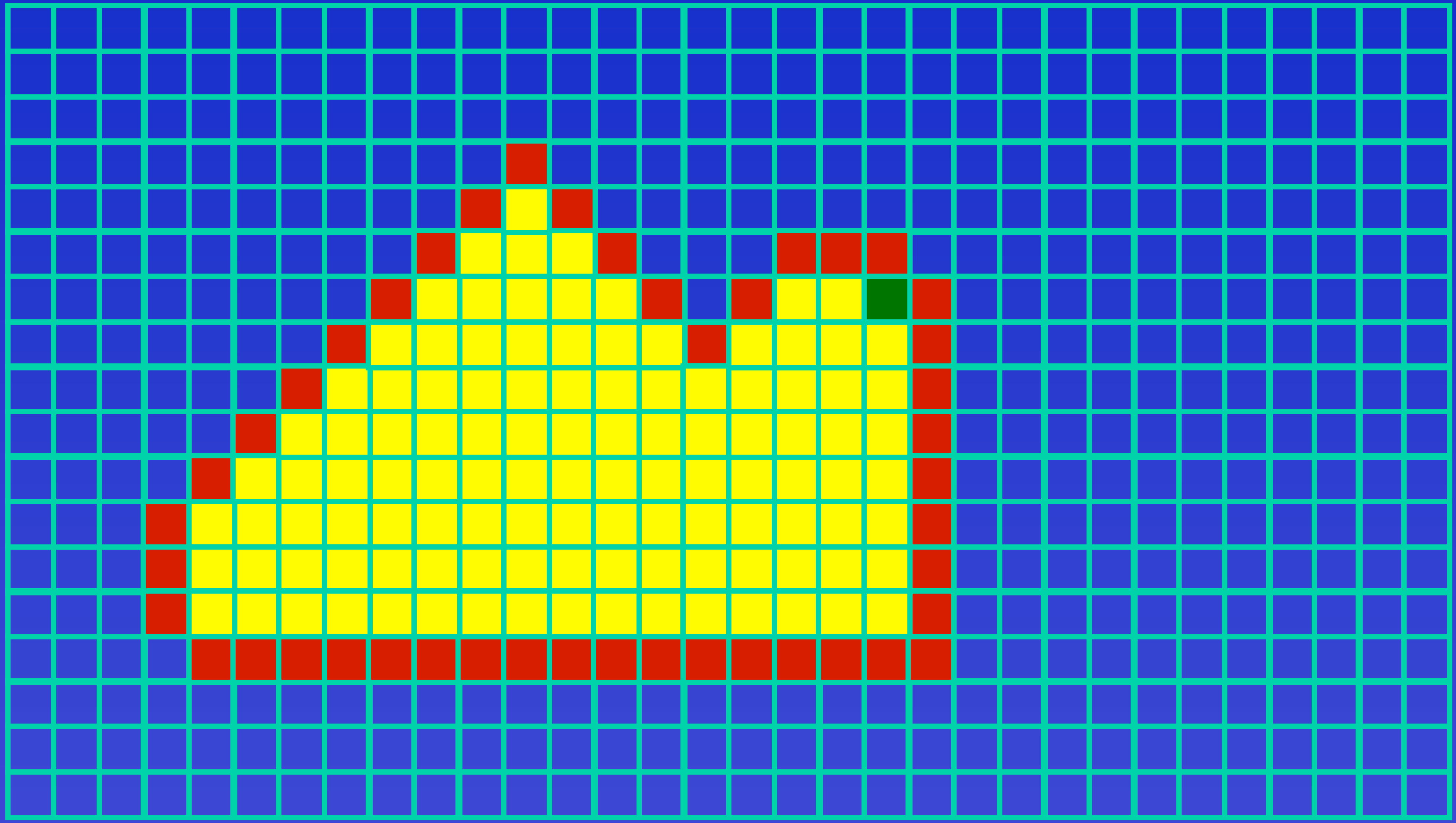
Flood Filling



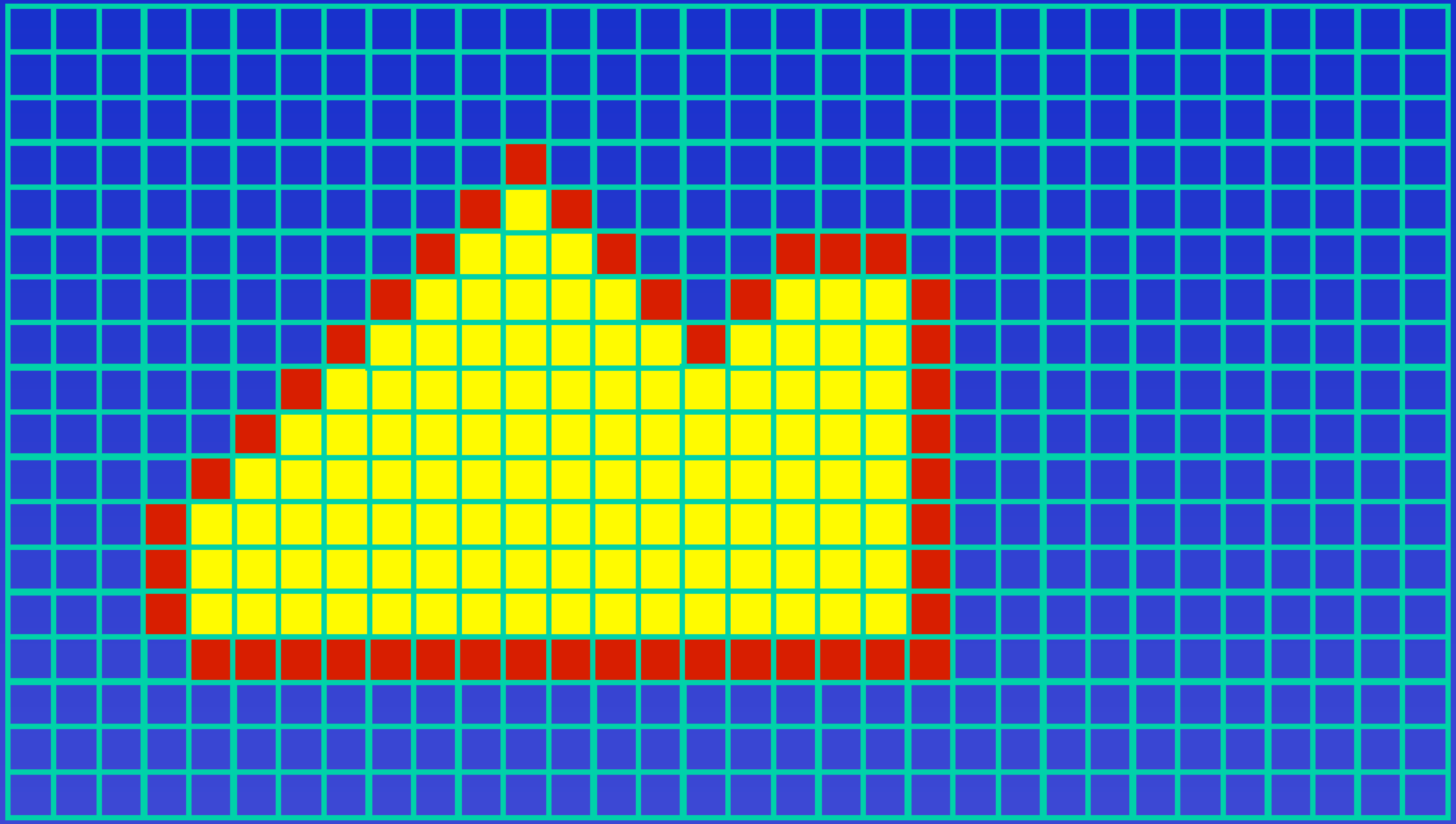
Flood Filling



Flood Filling



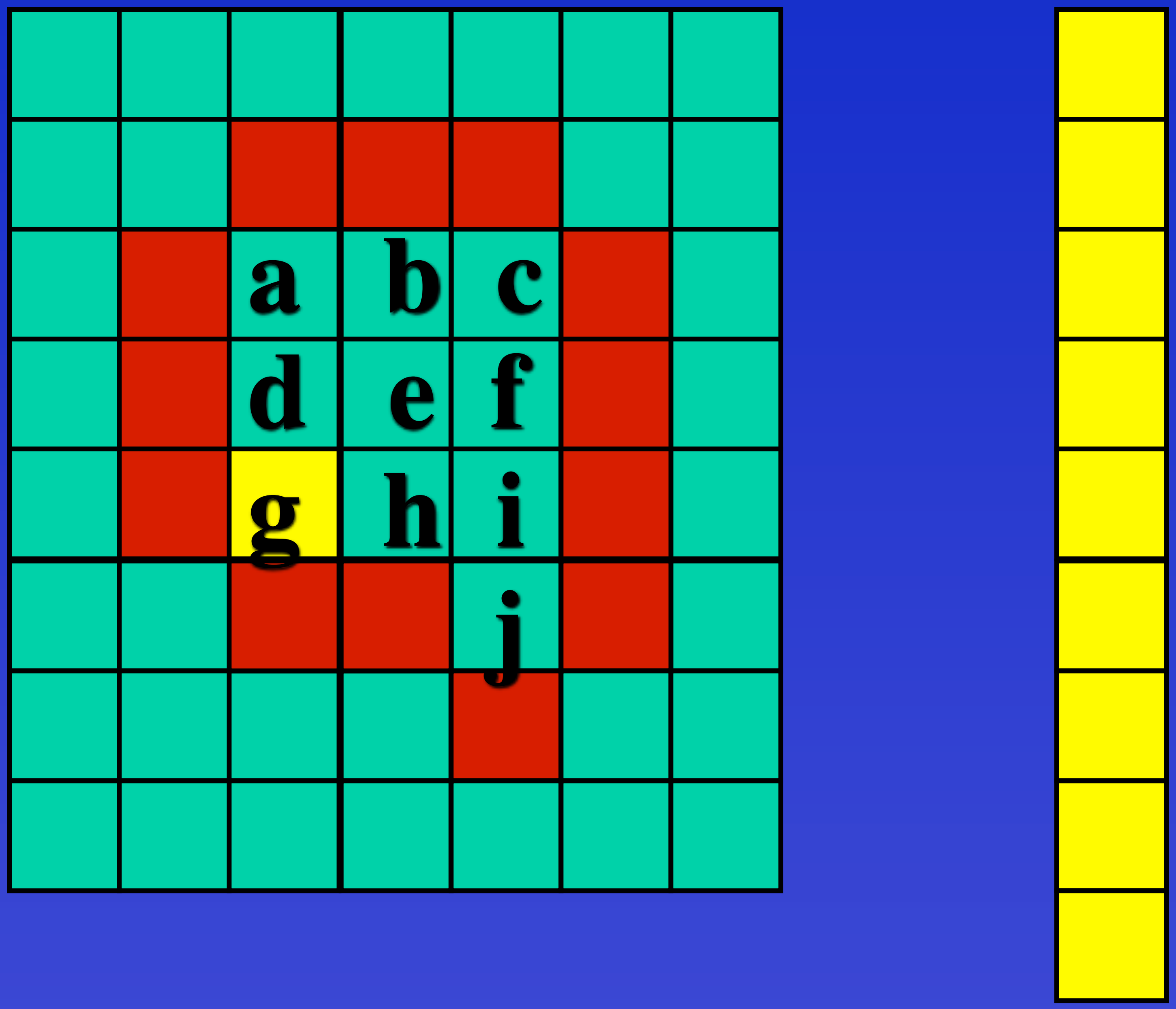
Flood Filling



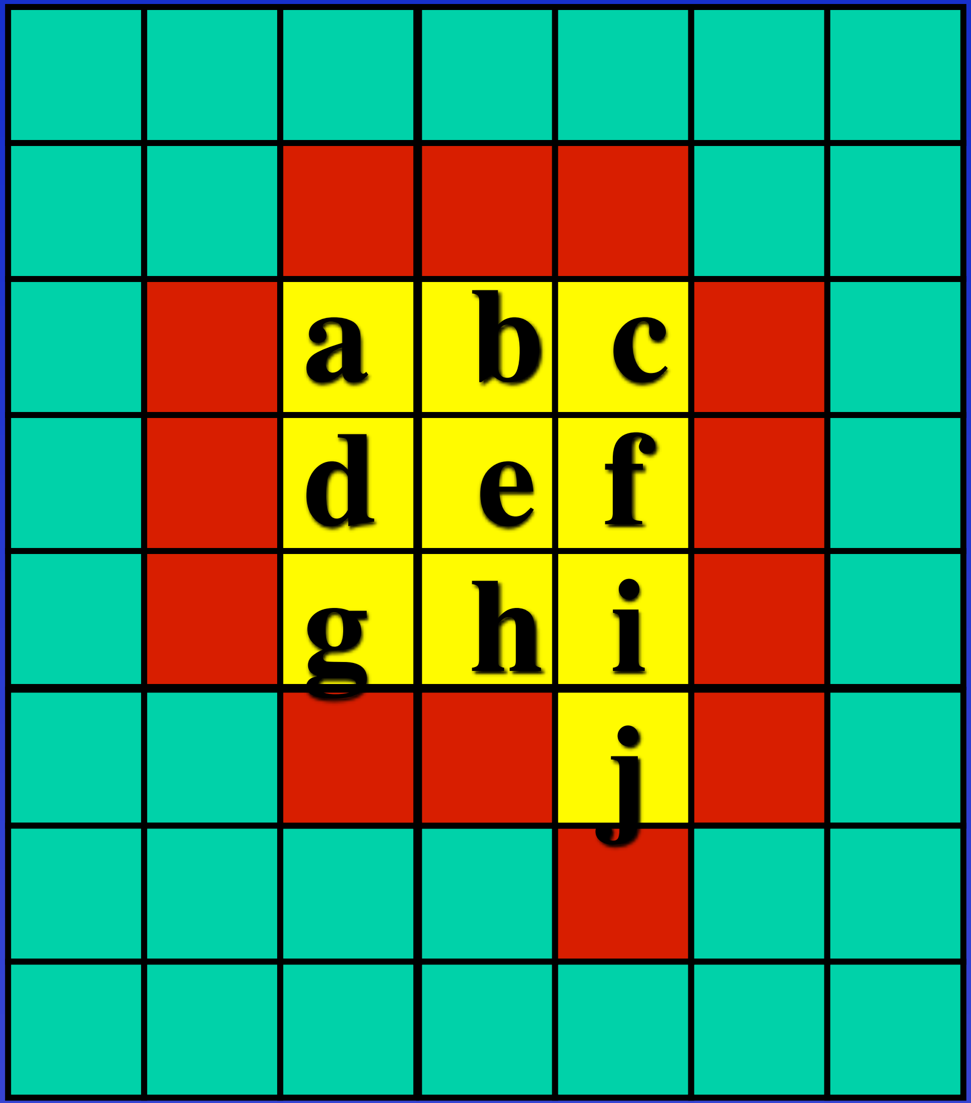
Simple approach

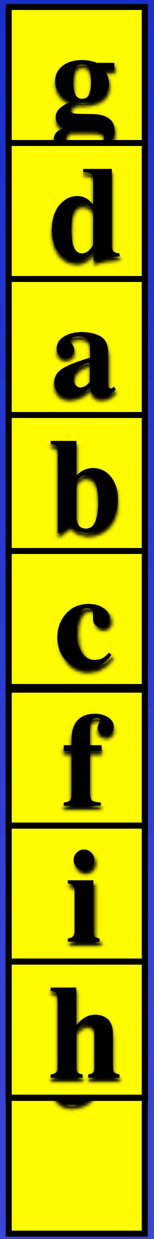
```
void fill(pixel me)
{ pixel tmp;
  colour(me);
  for(tmp = each me-neighbour)
  { if (!coloured(tmp)) then
      fill(tmp);
  }
}
```

Watch the stack

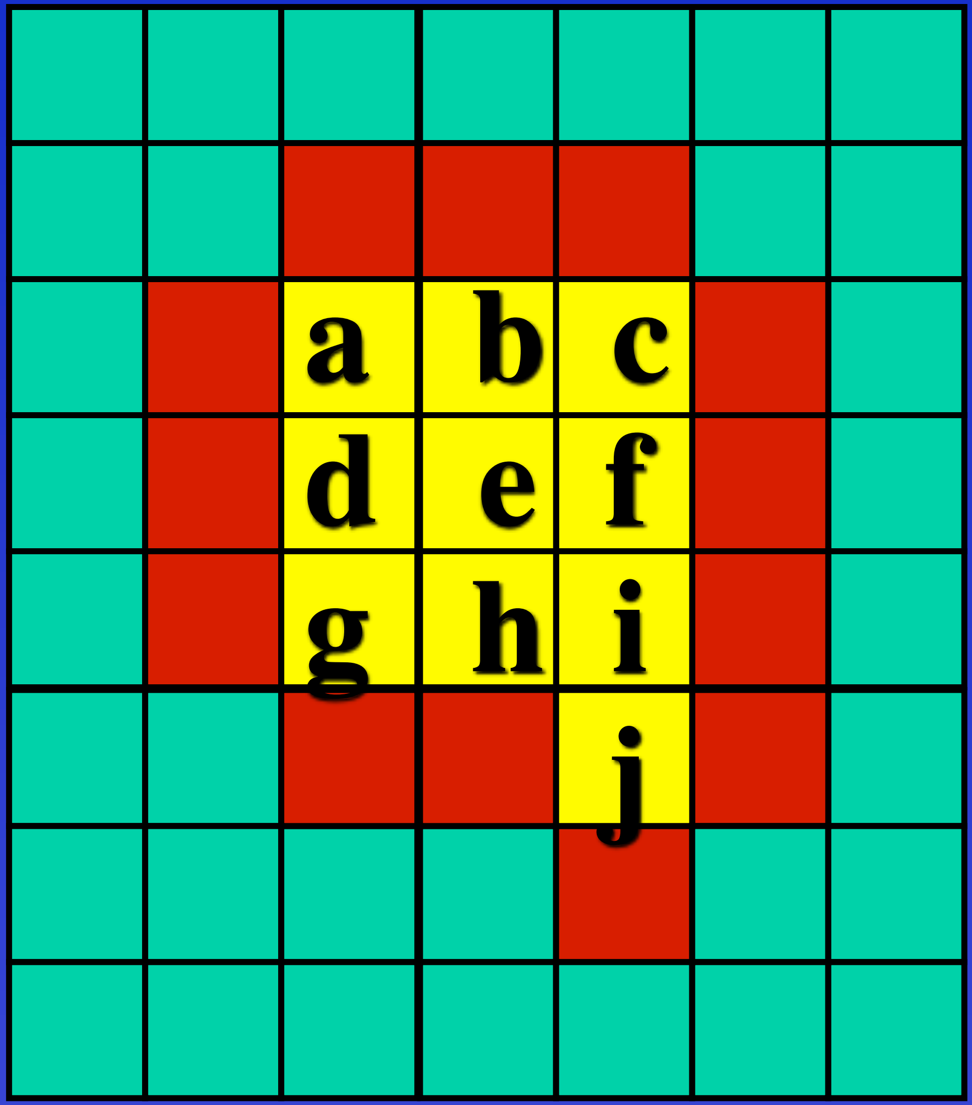


Watch the stack



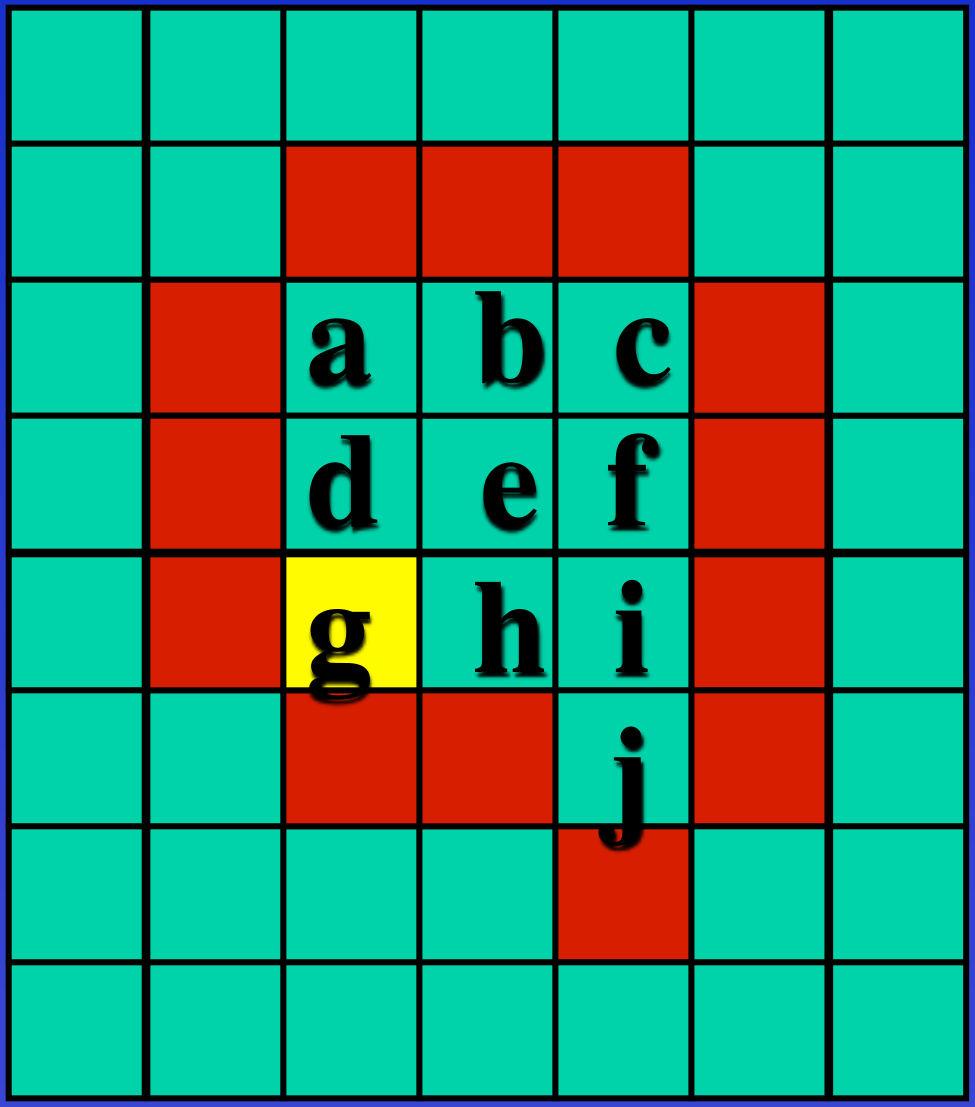


Watch the stack



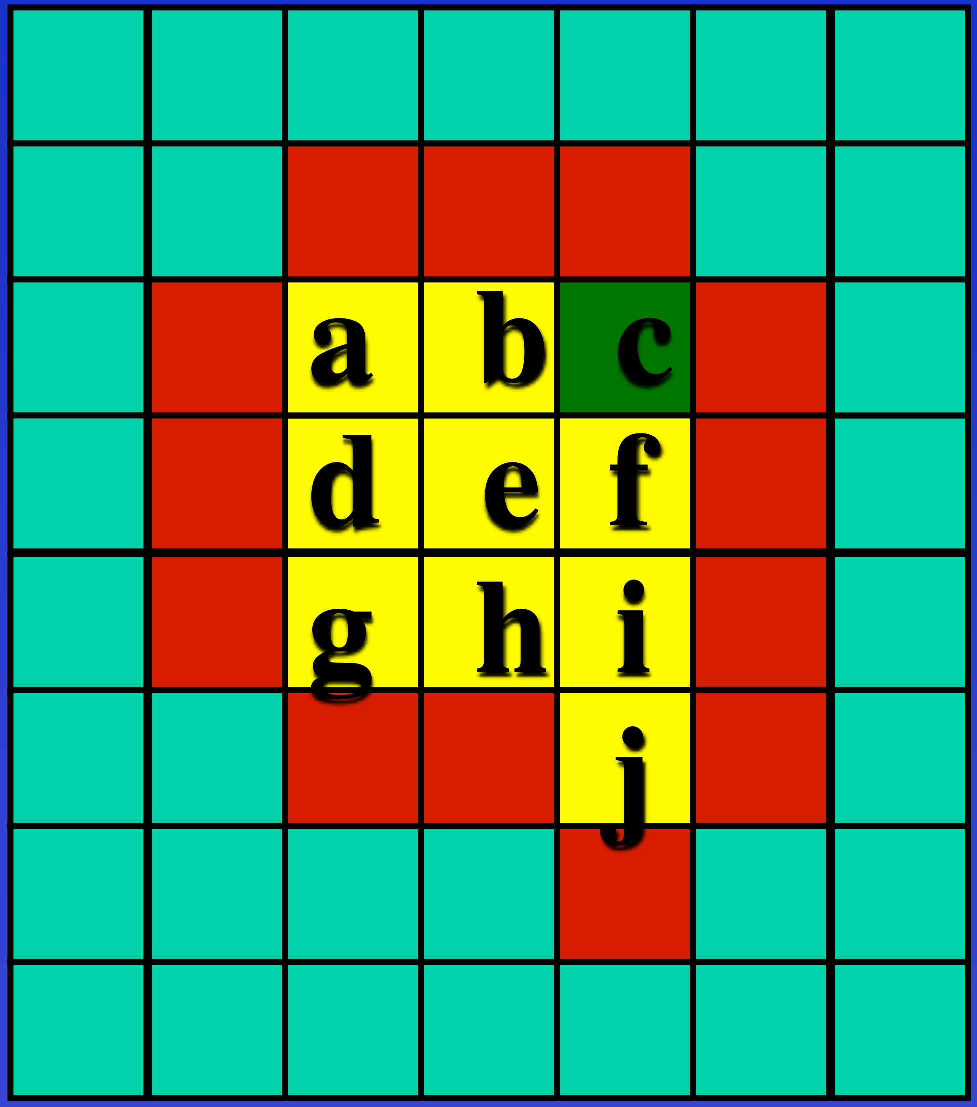
g
d
a
b
c
f
i
h
e

Better with a queue



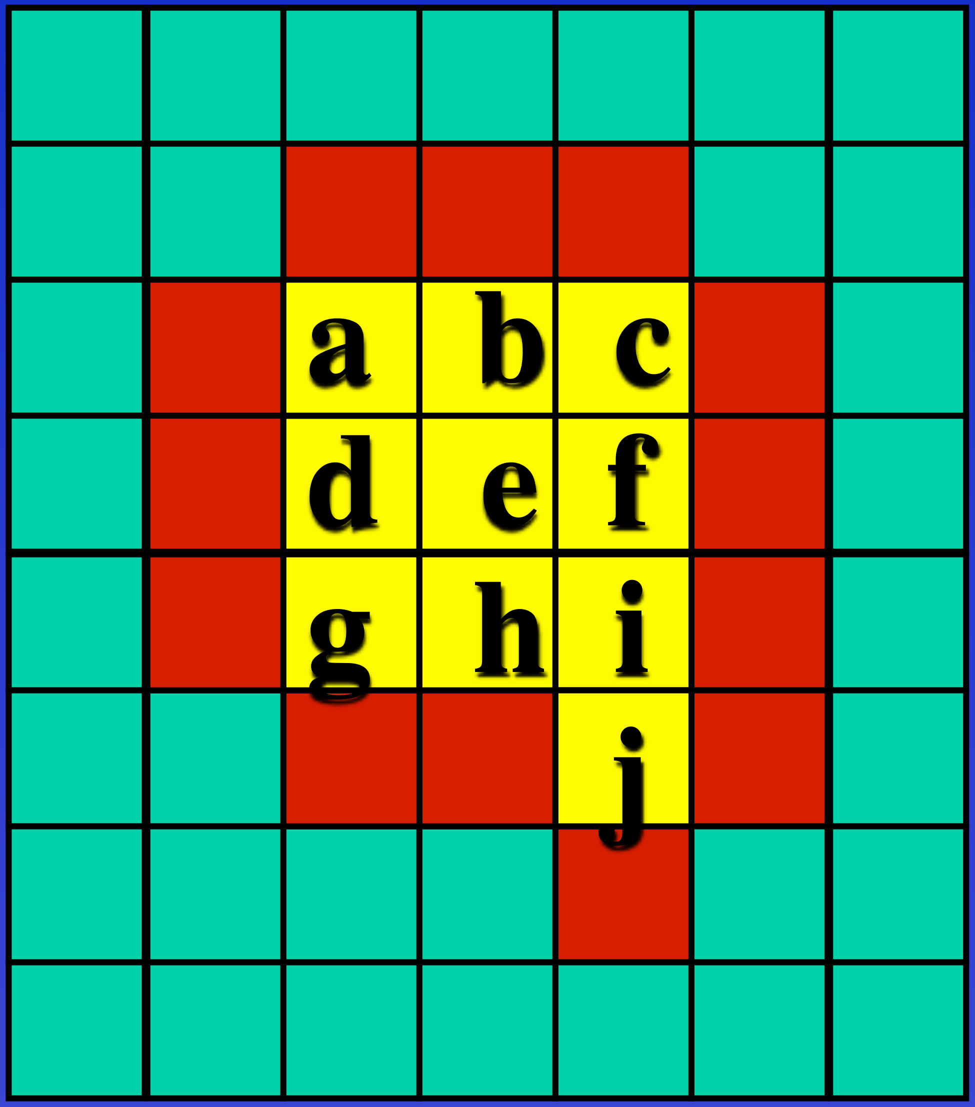
g d h a e i b f j c

Better with a queue



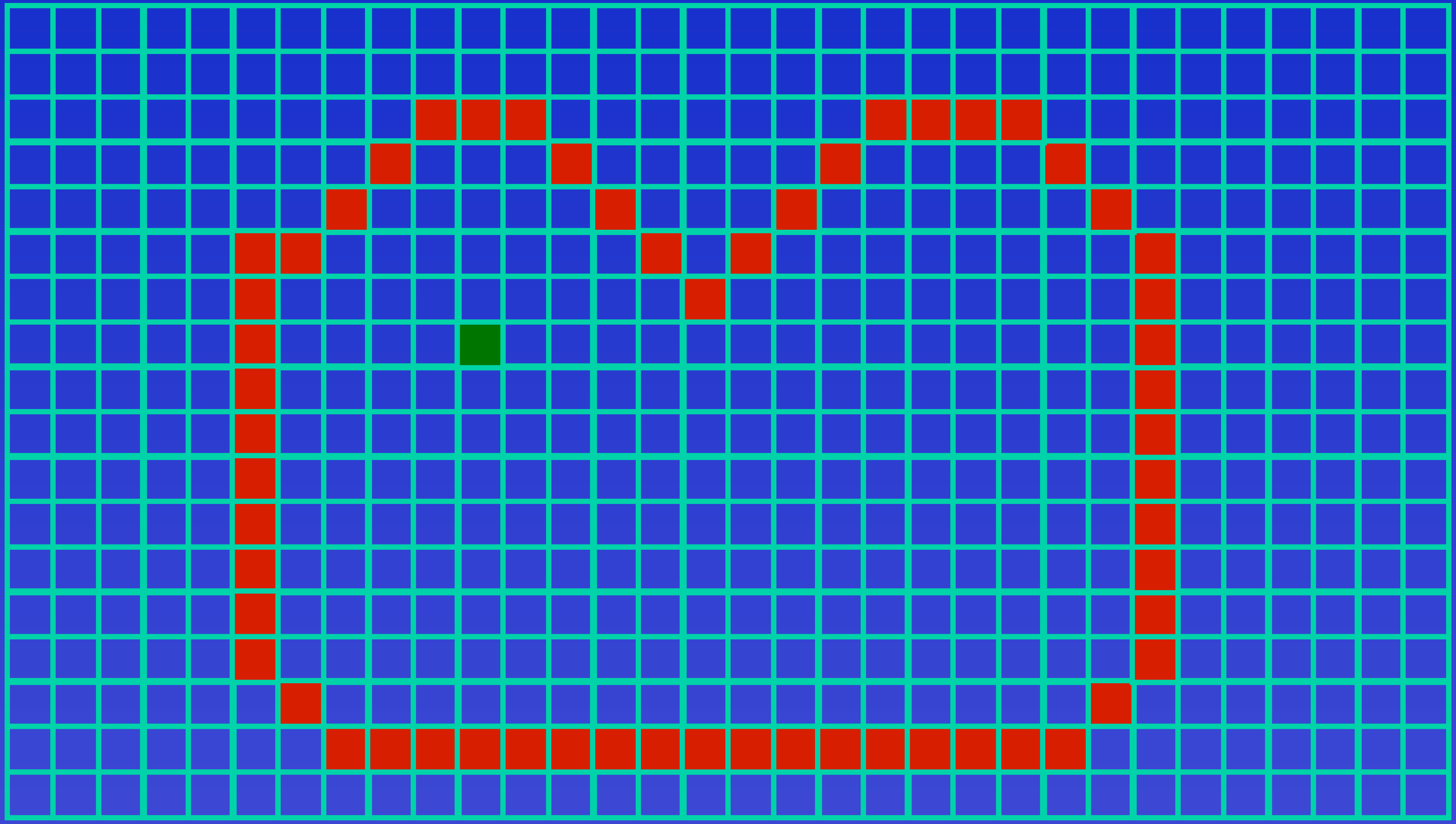
g d h a e i b f j c

Better with a queue

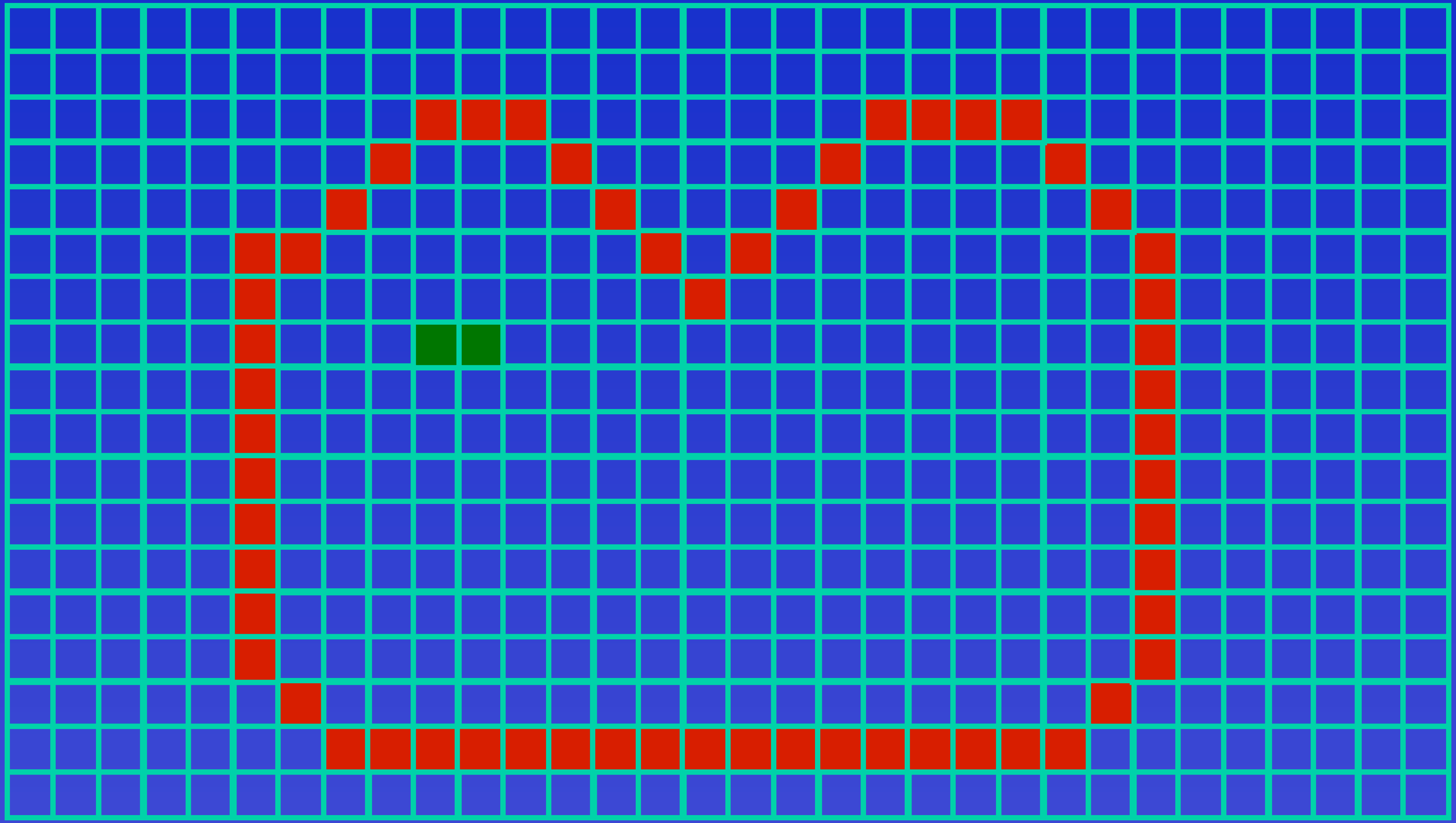


g d h a e i b f j c

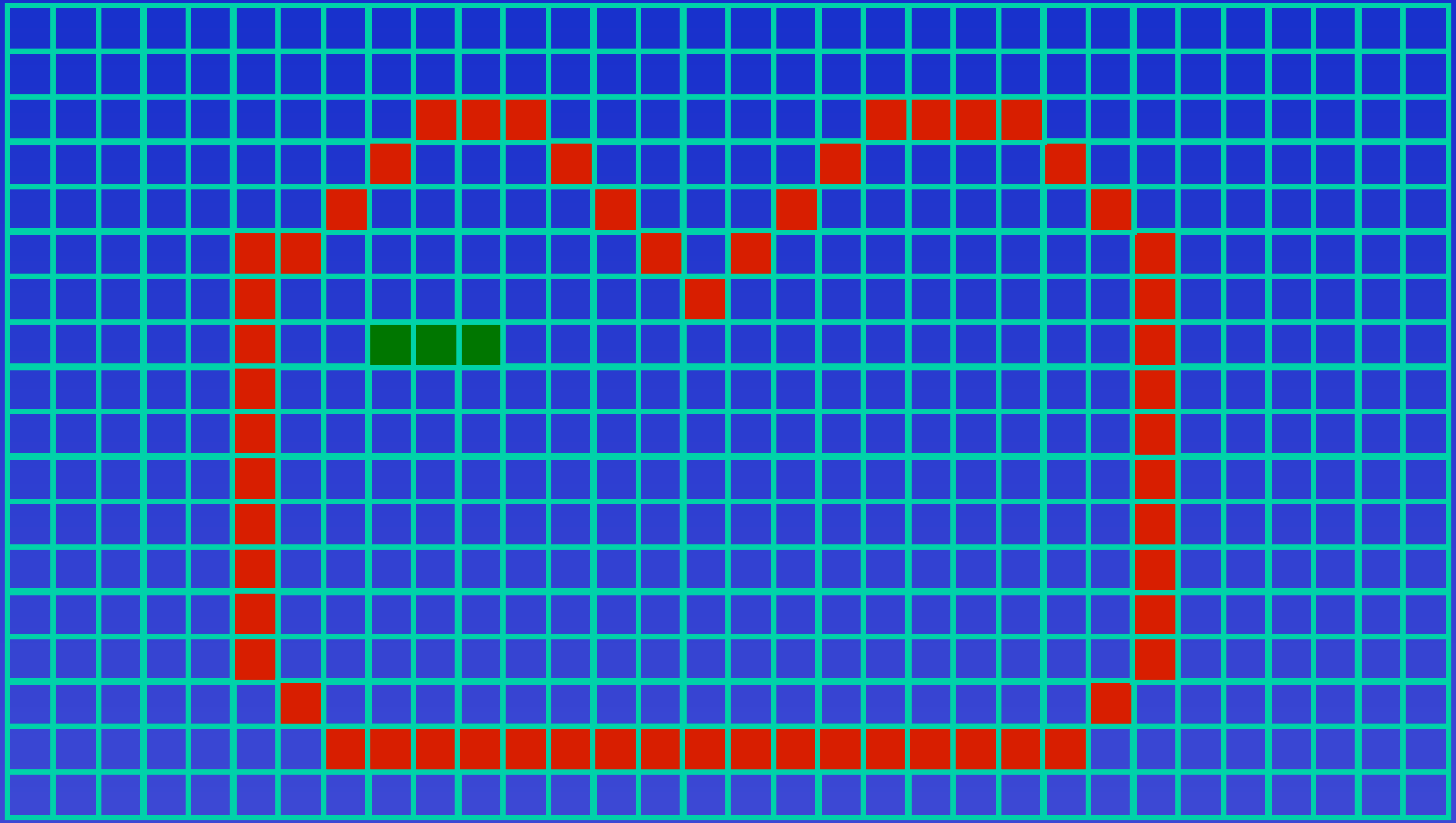
Better yet with runs



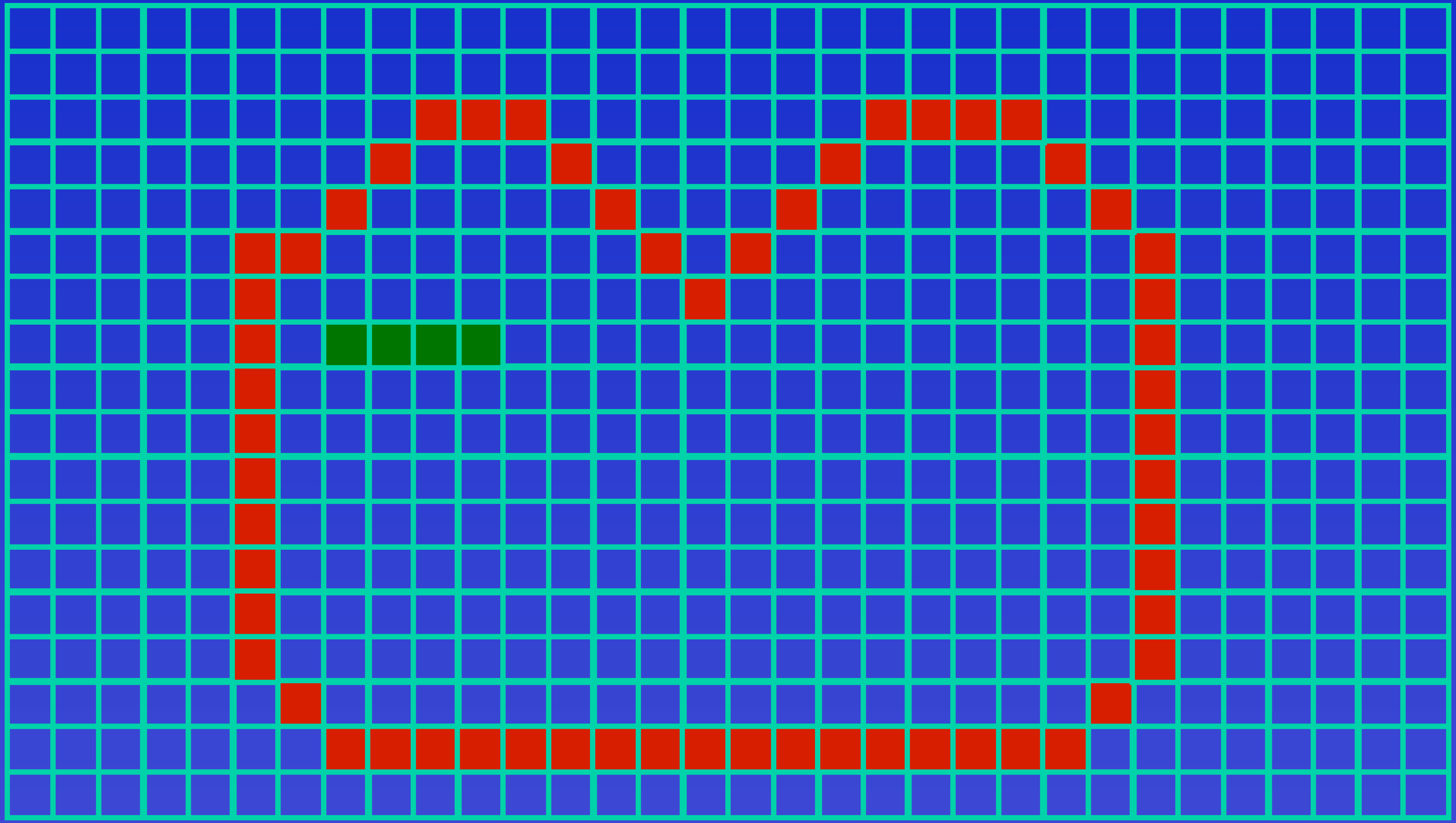
Better yet with runs



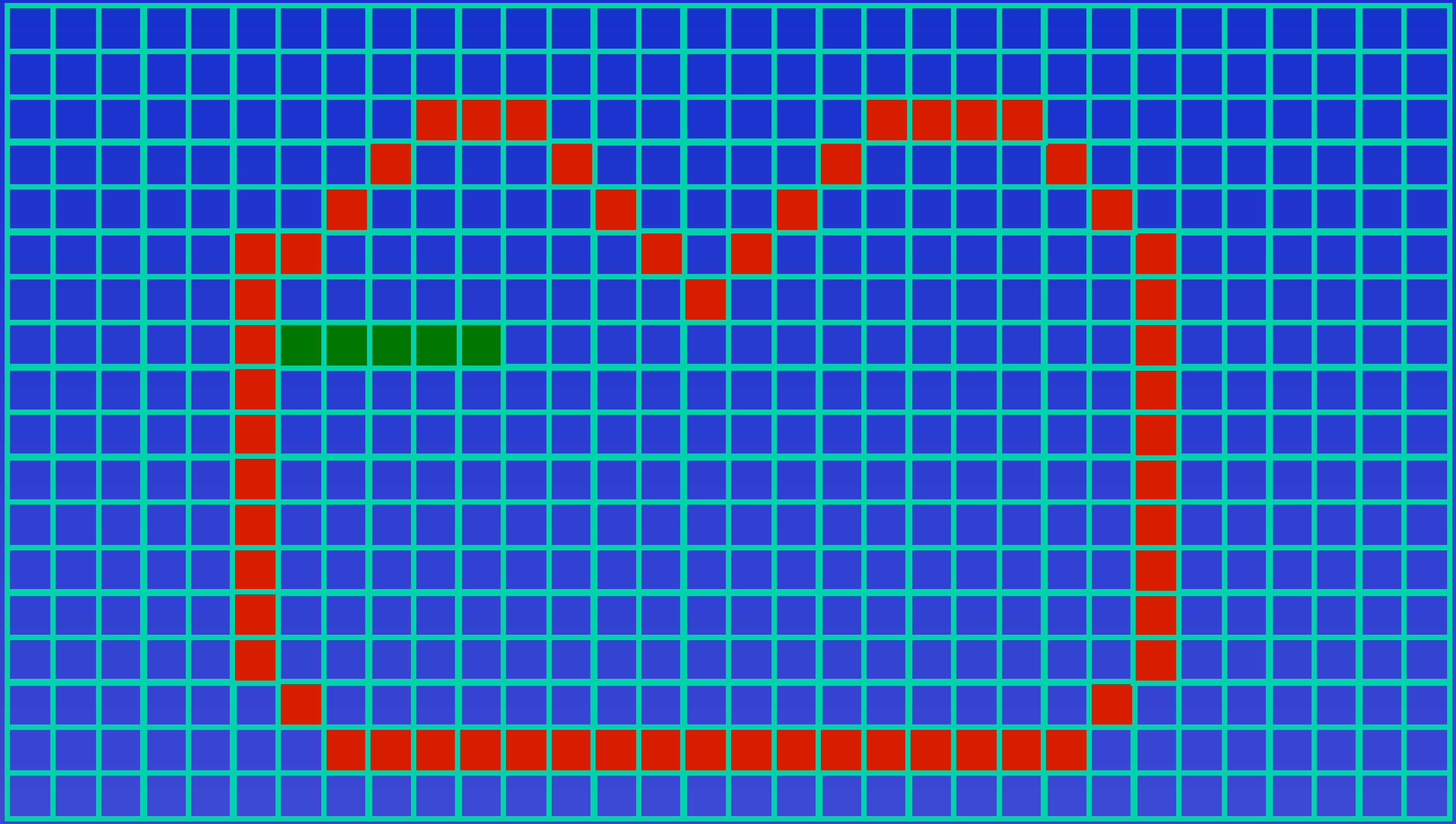
Better yet with runs



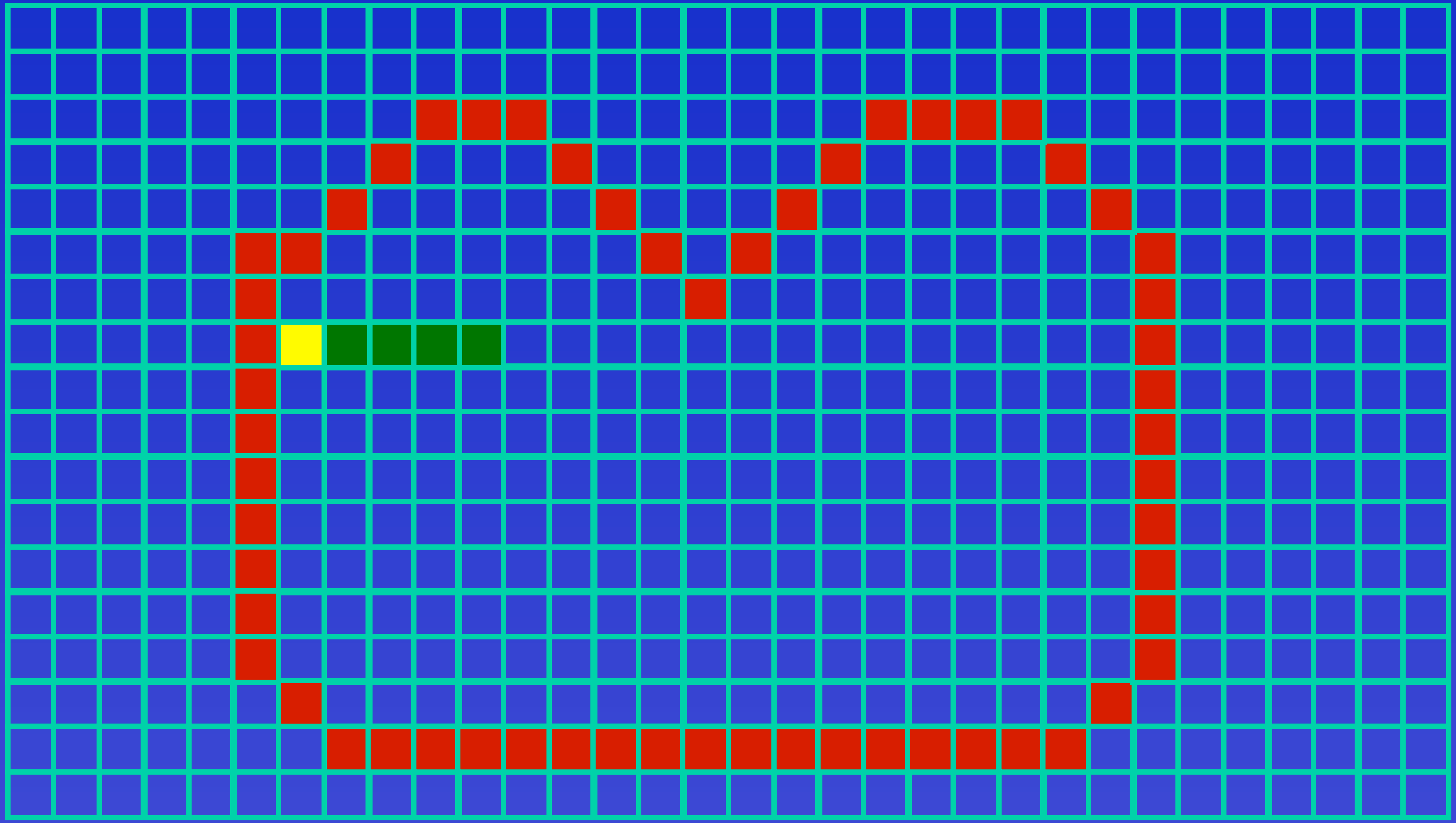
Better yet with runs



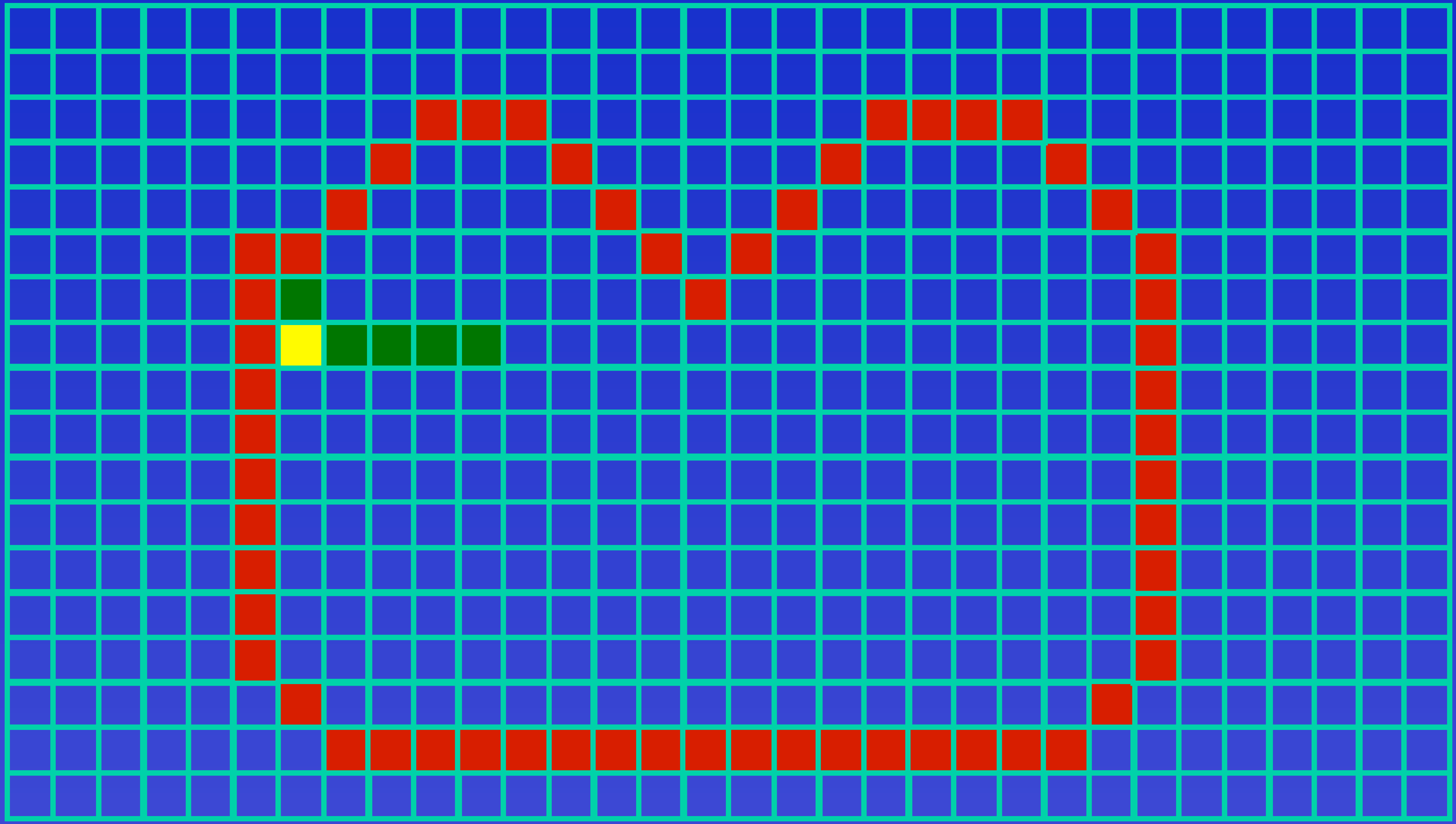
Better yet with runs



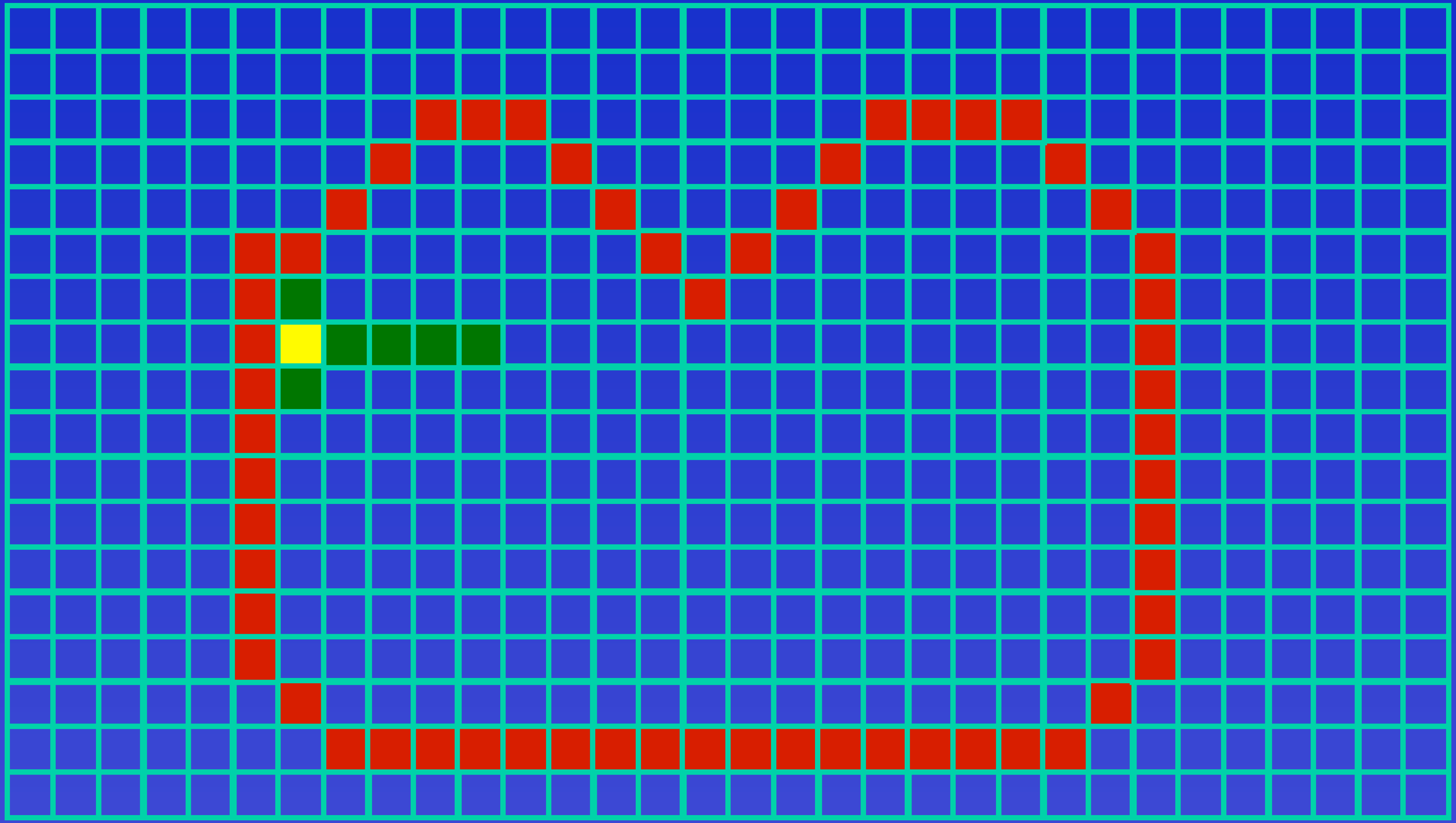
Better yet with runs



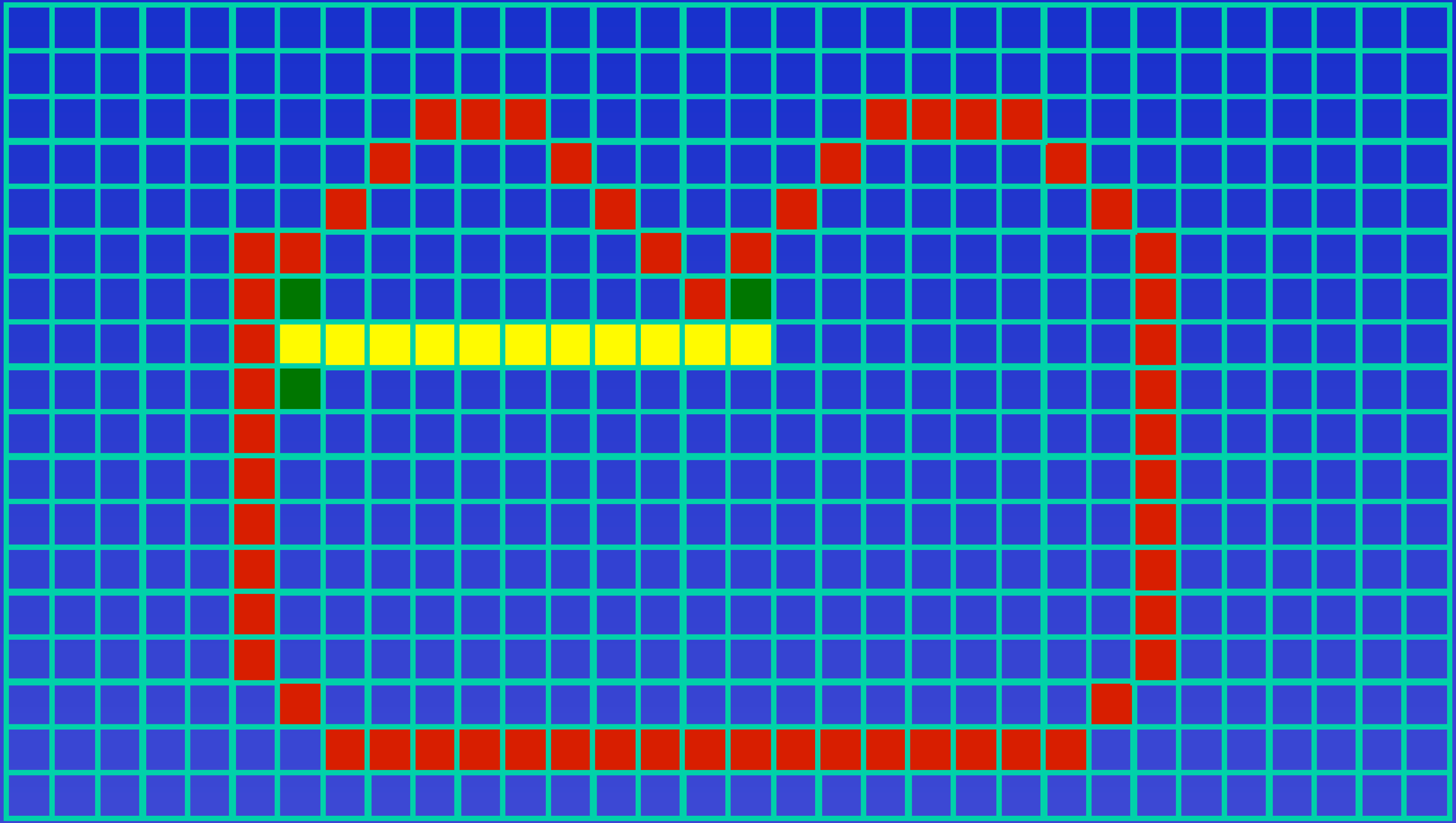
Better yet with runs



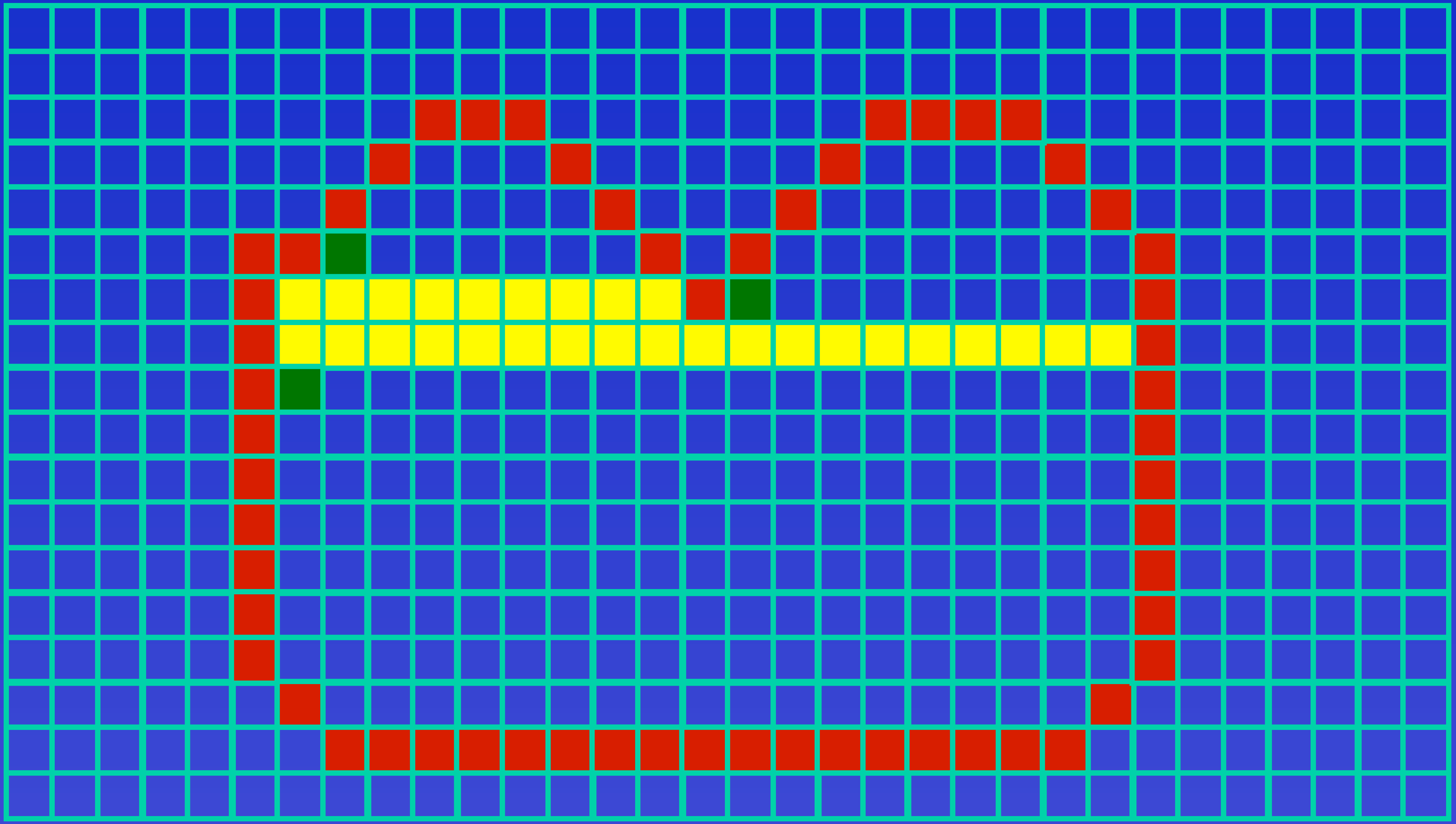
Better yet with runs



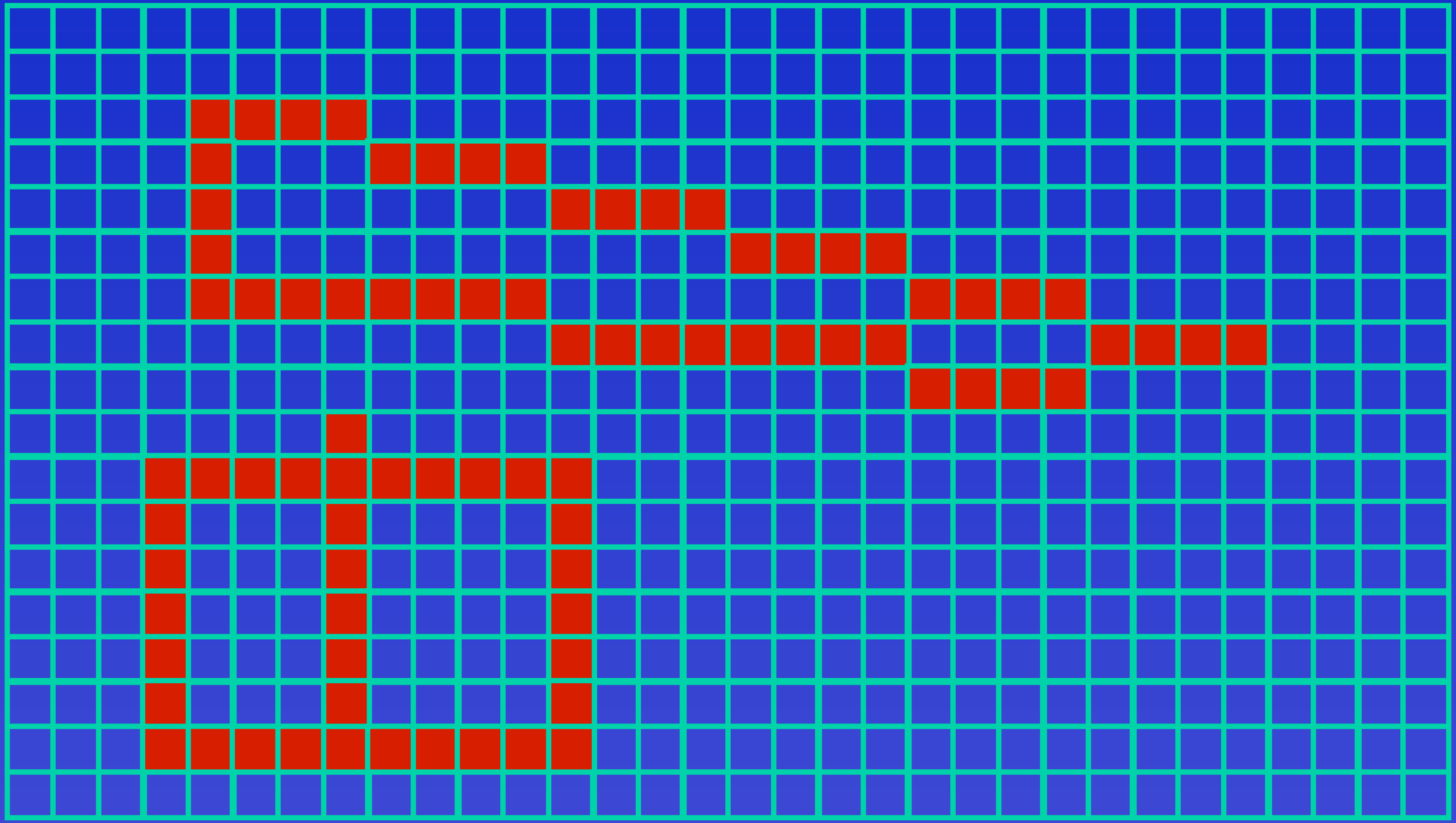
Better yet with runs



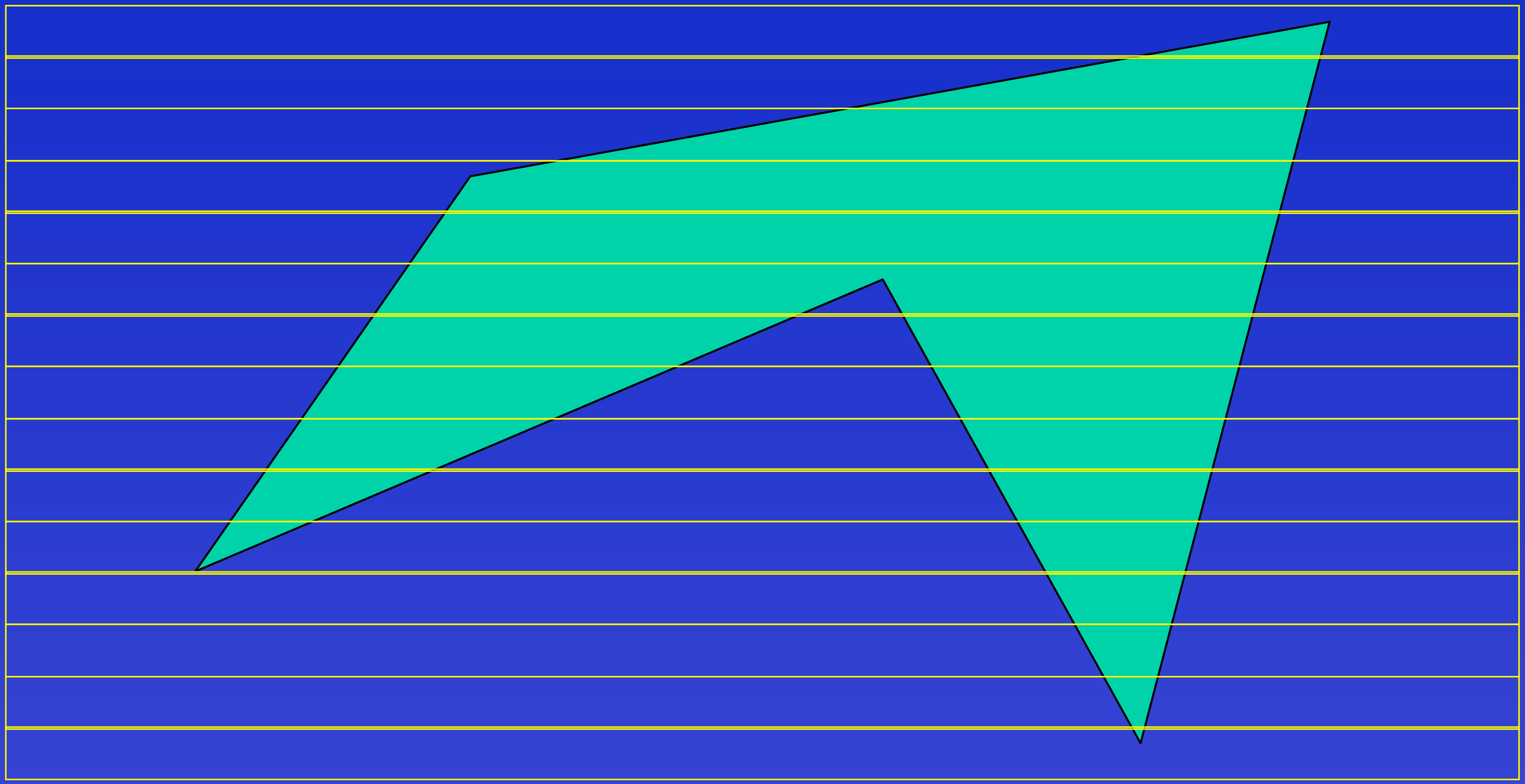
Better yet with runs



Inescapable problems

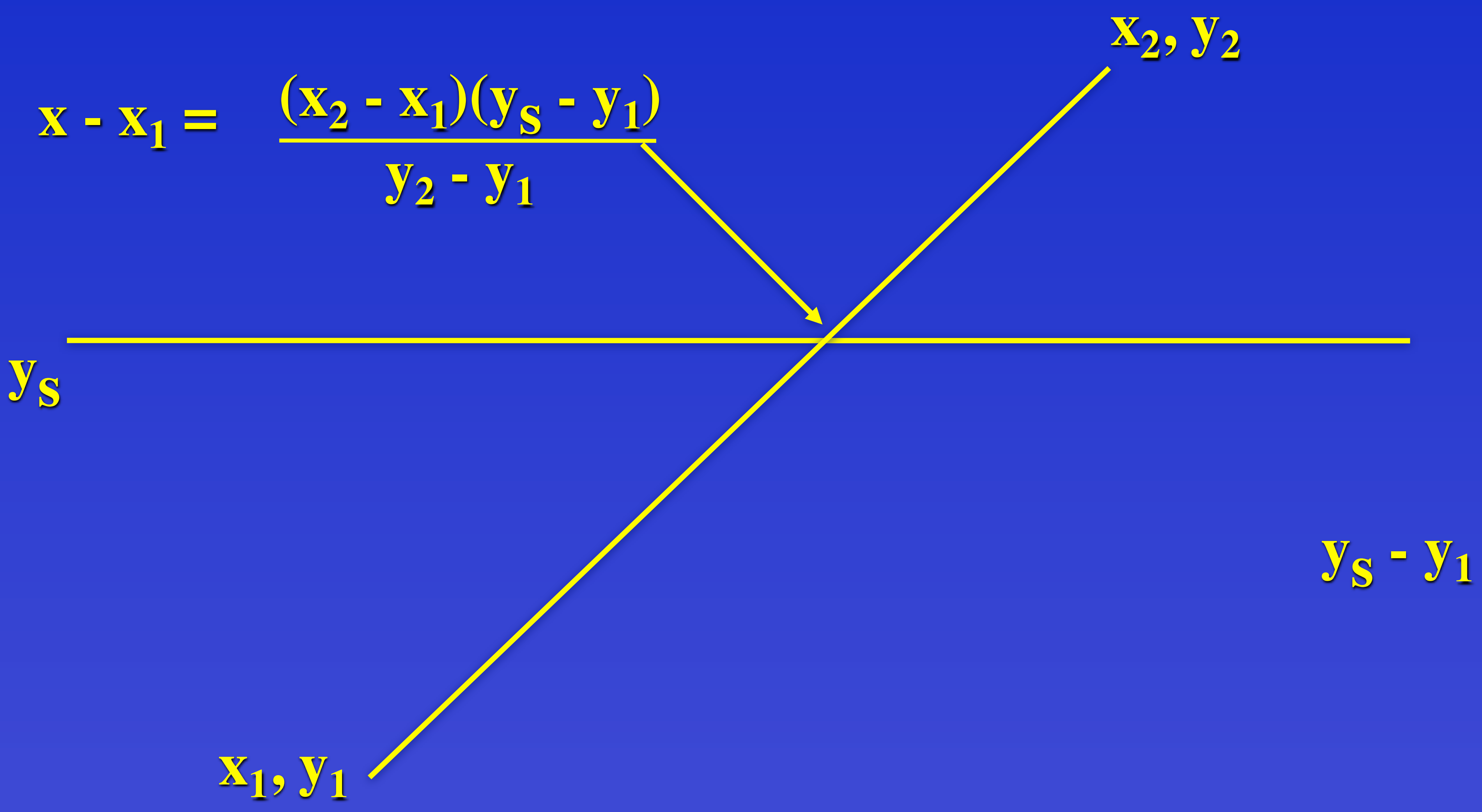


Don’t wait for pixels...



...Scan across raw polygon

Intersection calculation



$$x - x_1 = \frac{(x_2 - x_1)(y_S - y_1)}{y_2 - y_1}$$

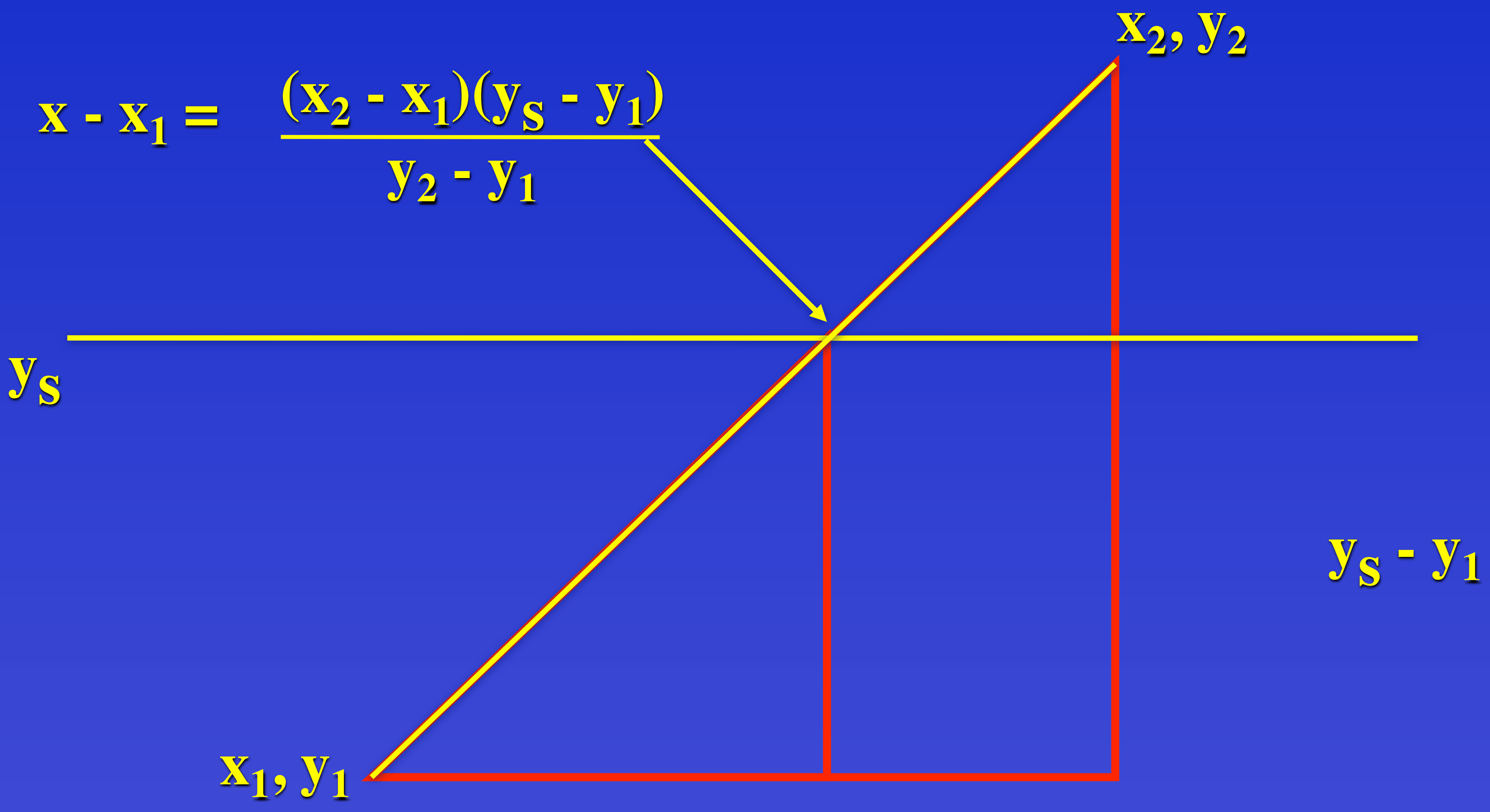
x_2, y_2

y_S

$y_S - y_1$

x_1, y_1

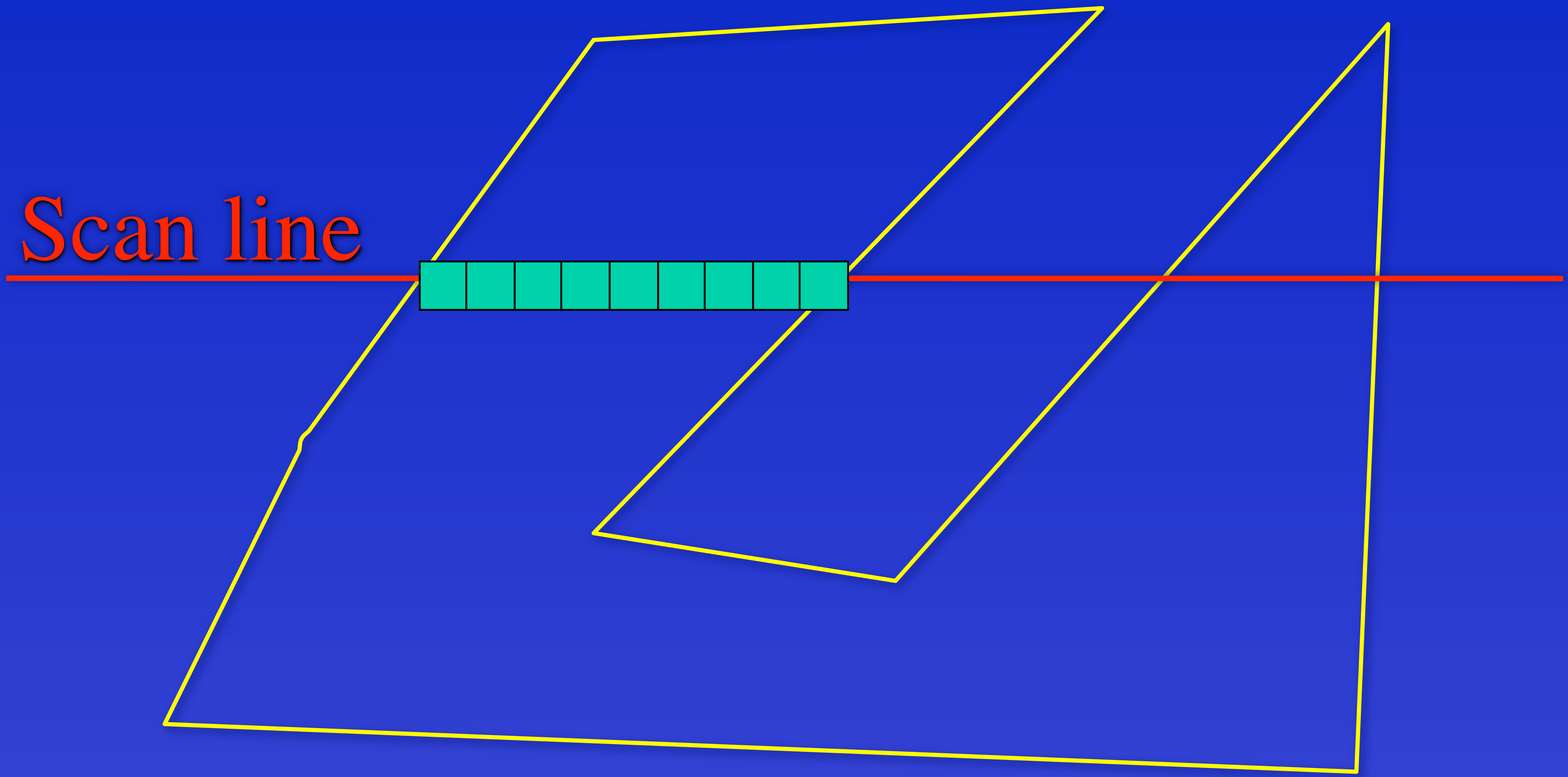
Intersection calculation

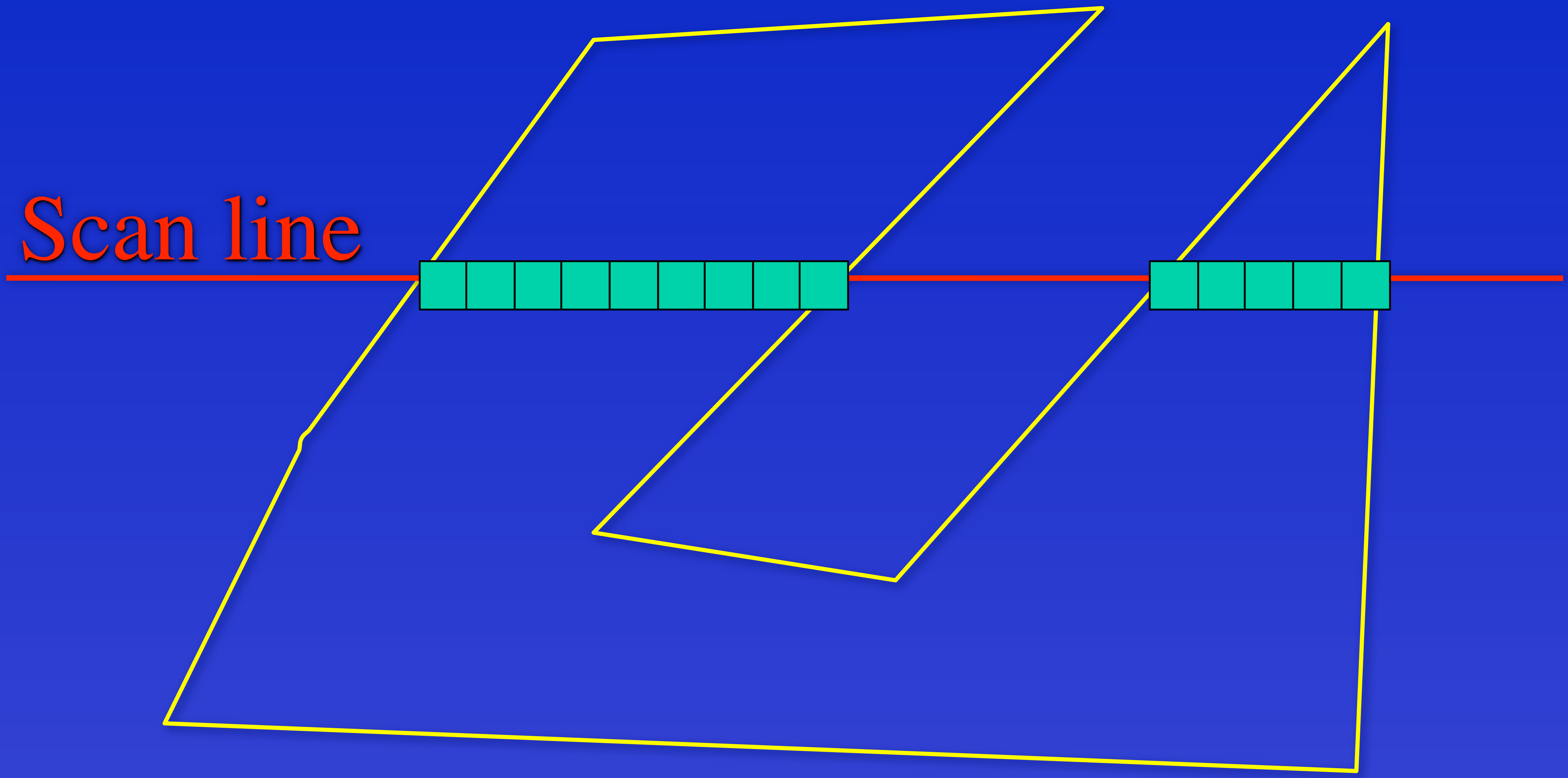


$$x - x_1 = \frac{(x_2 - x_1)(y_S - y_1)}{y_2 - y_1}$$

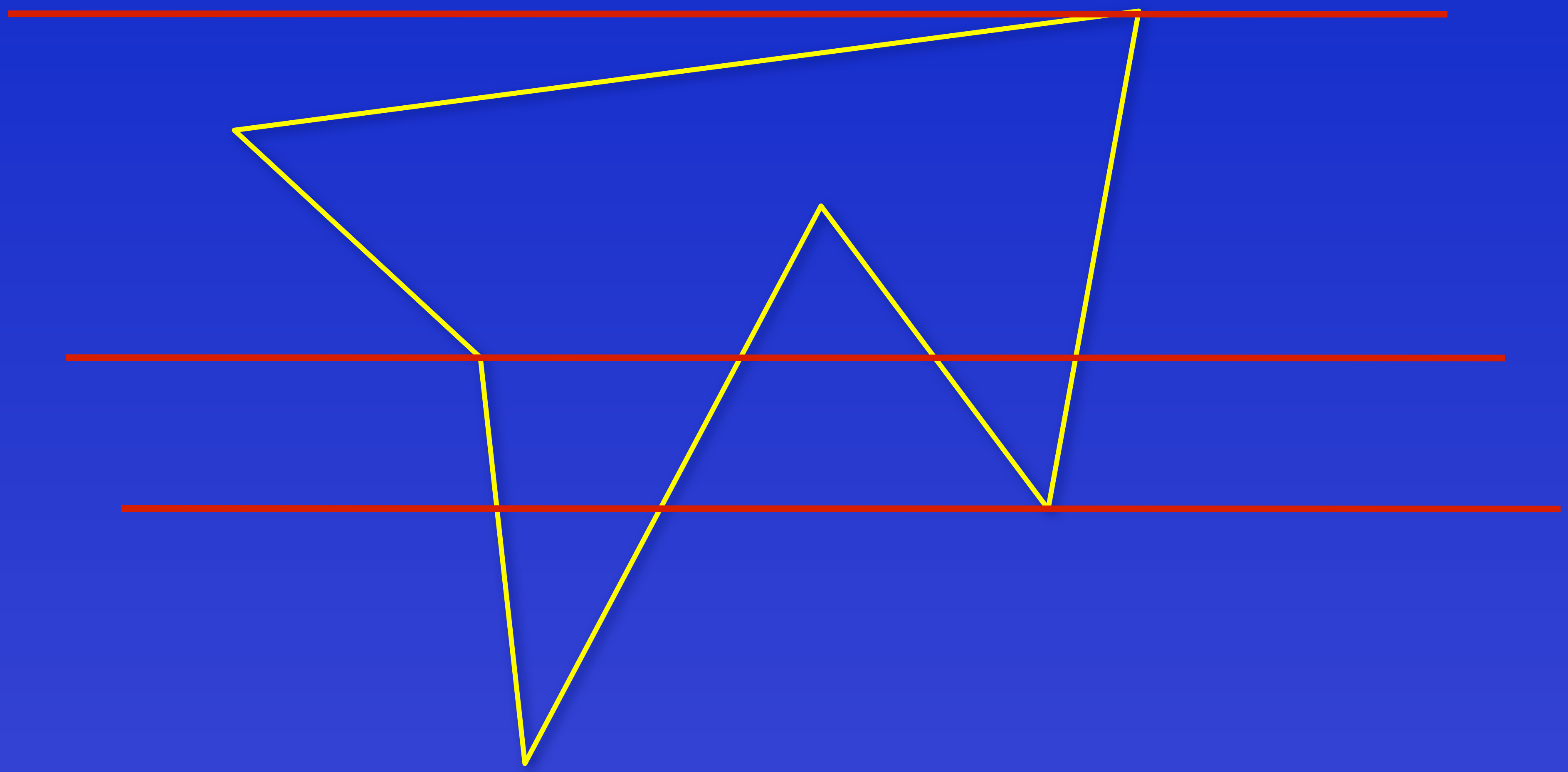
Basic scan-line filler

- Scale polygon to screen coordinates
- For each horizontal line find all intersections with polygon edges
- Draw in alternate line segments

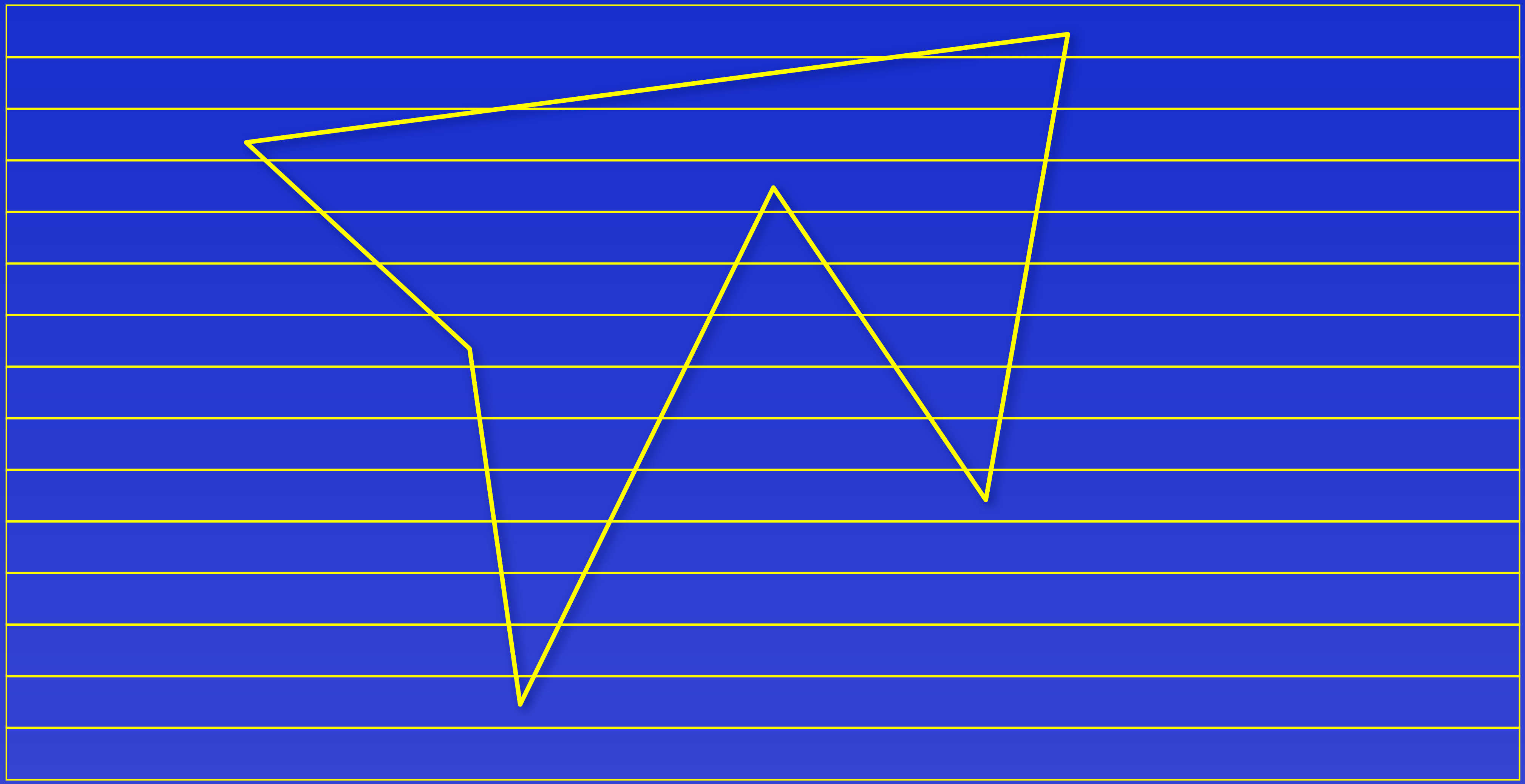

Scan line


Scan line

Double intersection



Offset vertices



Aha!