Challenge #1 - Bresenham Algorithm

Computer Graphics

Presented by:

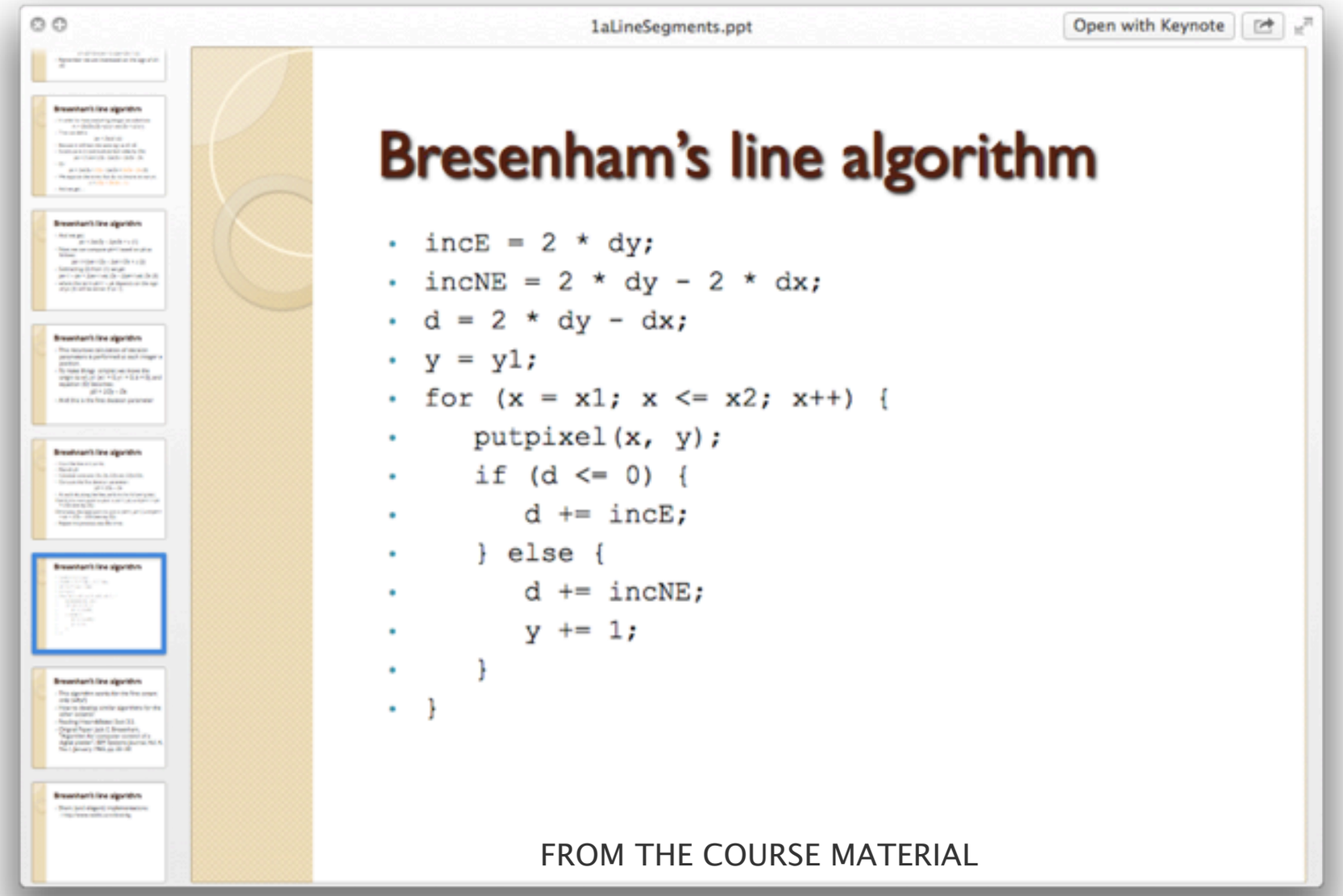
Santiago Zubieta / Jose Cortes

Teacher:

Helmuth Trefftz

Universidad EAFIT

1. Line Algorithm

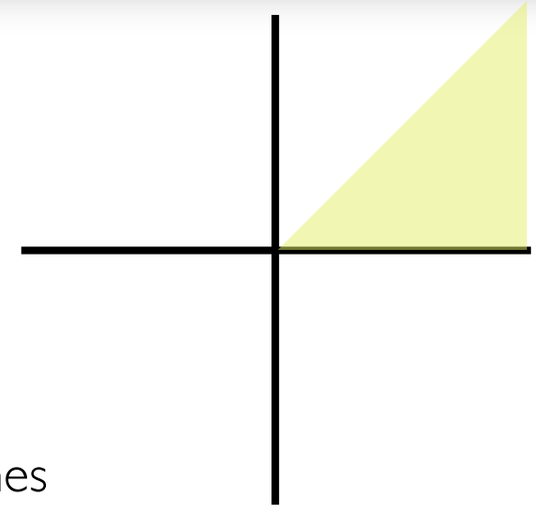


Considerations:

-Only works in 1st Octant

Properties of 1st:

- **ΔX**, **ΔY** are positive
- Slope is $0 \leq m \leq 1$
- X constantly increases, Y only sometimes



How to generalize for all octants?

Treat them as if they were the 1st

Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX** **and never recalculate it again, leave as it is here**

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

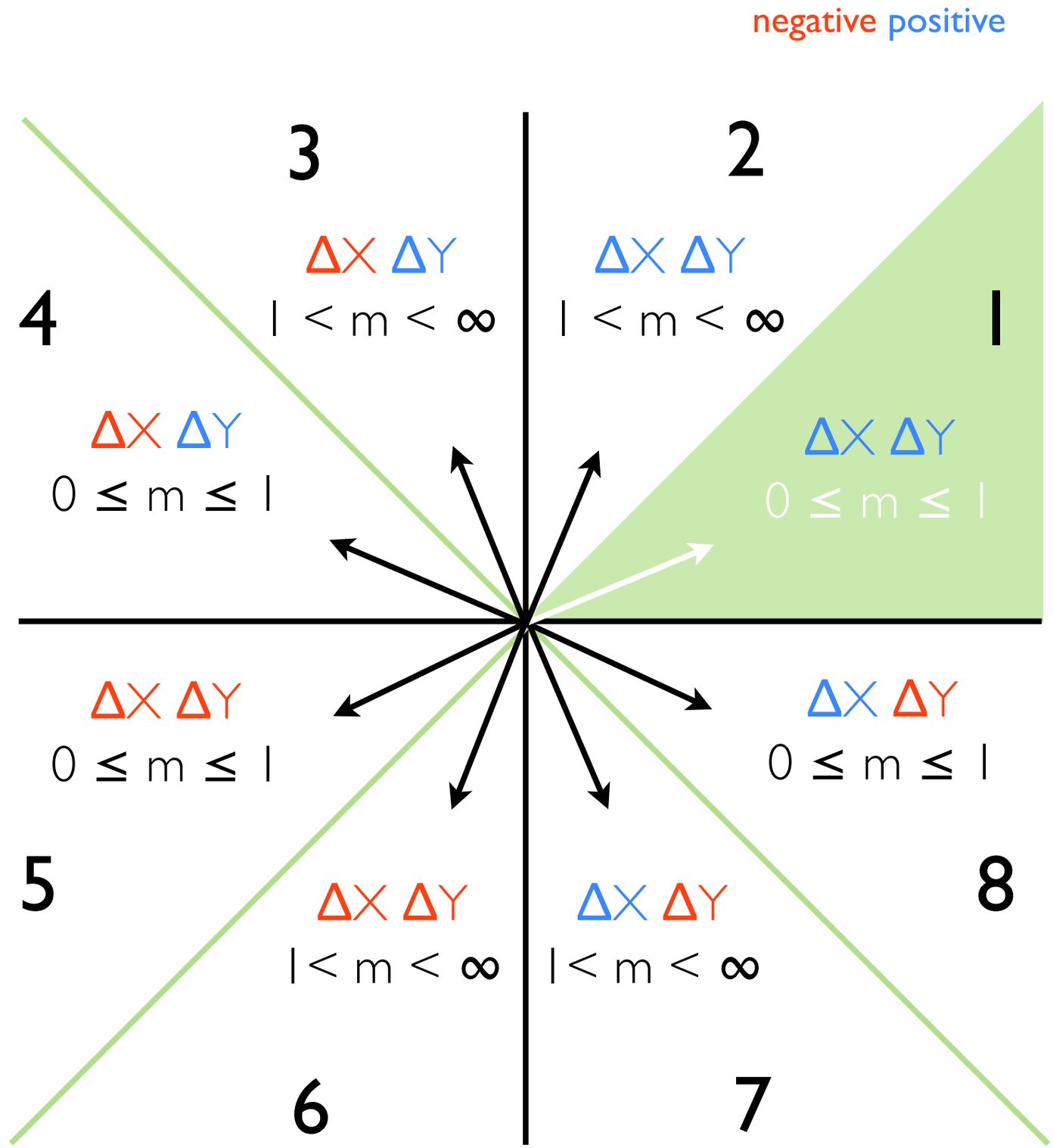
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \le m \le 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX**
and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

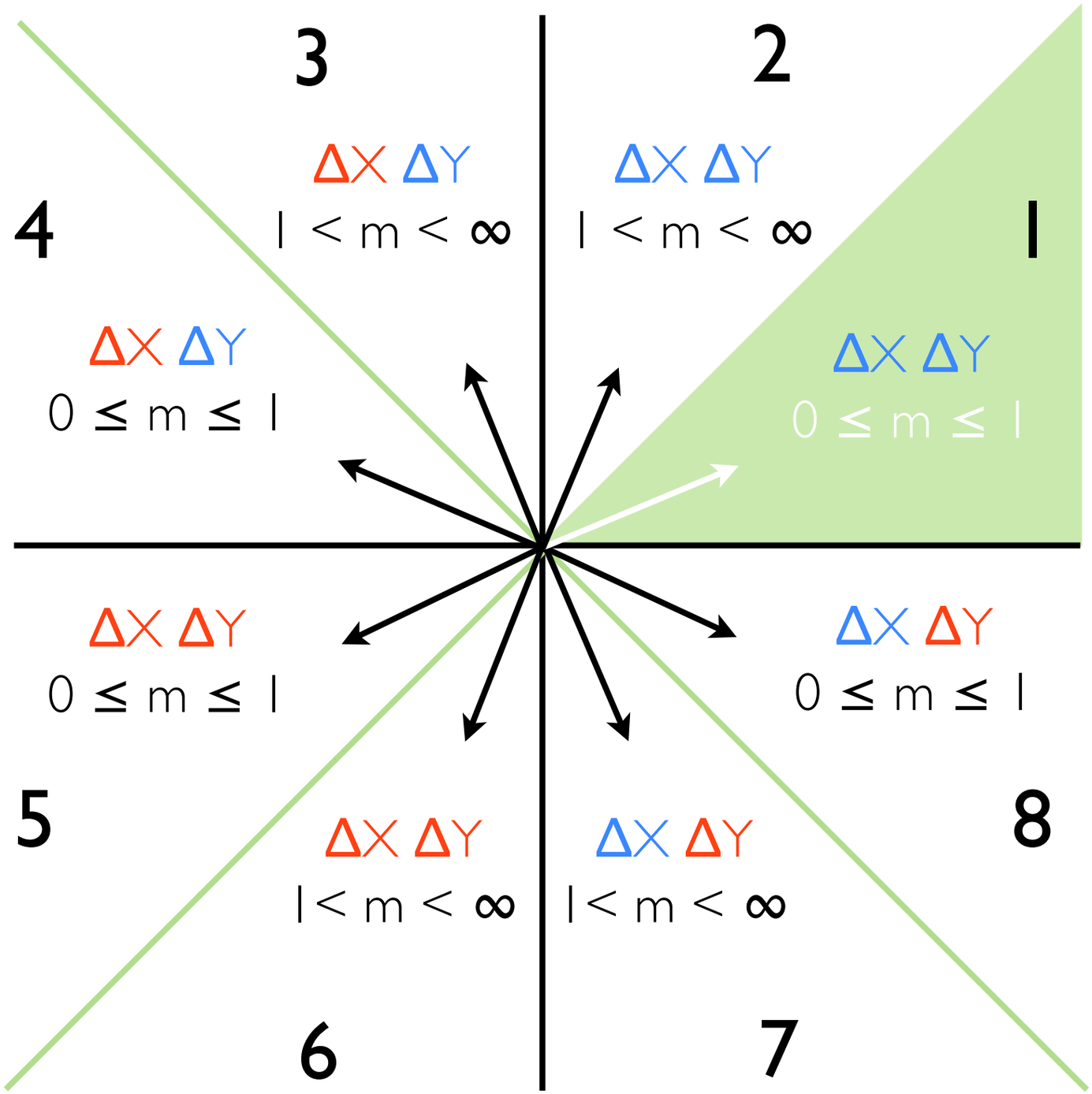
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \le m \le 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX** and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

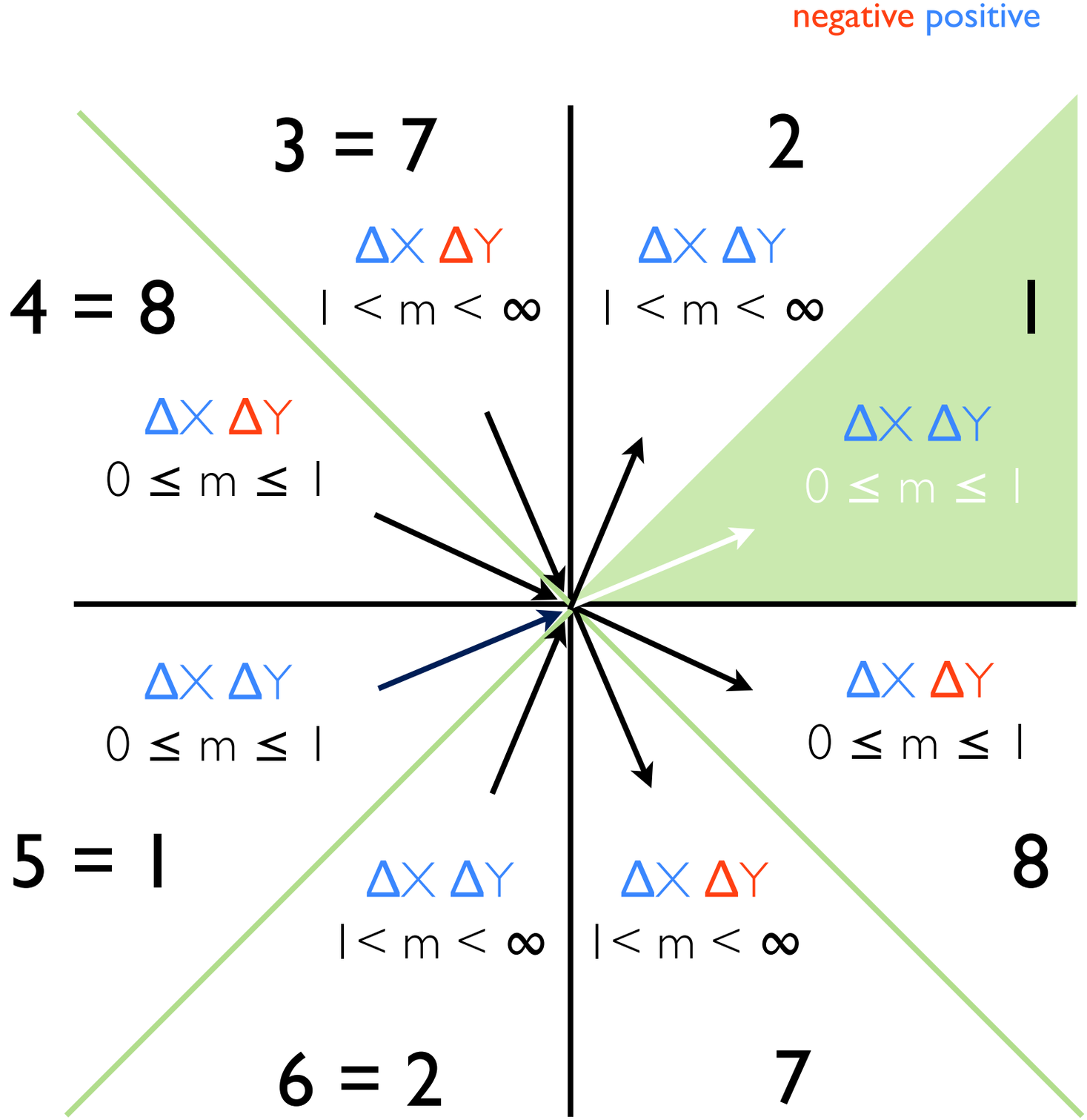
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \le m \le 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX**
and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

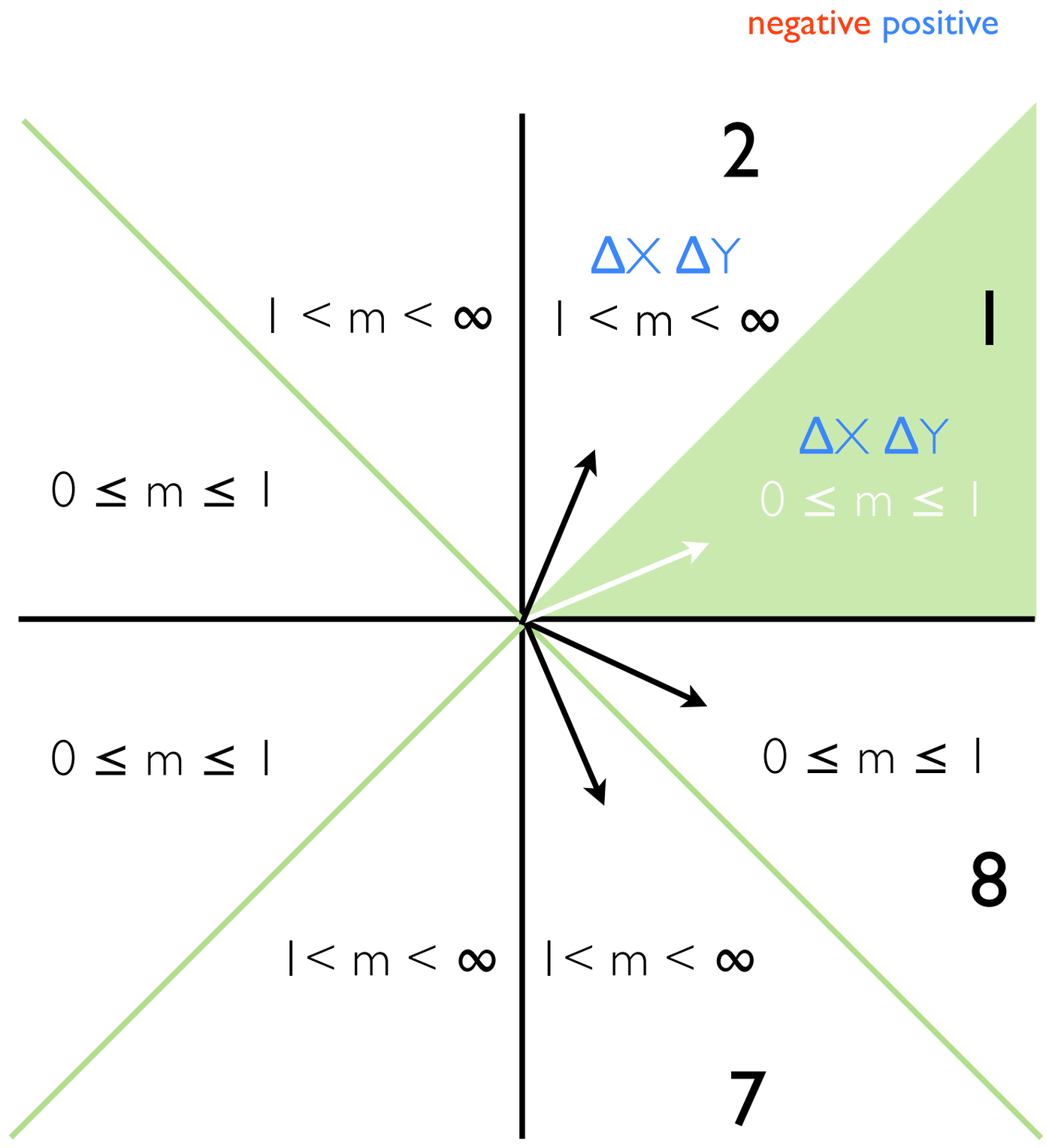
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \leq m \leq 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX**
and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

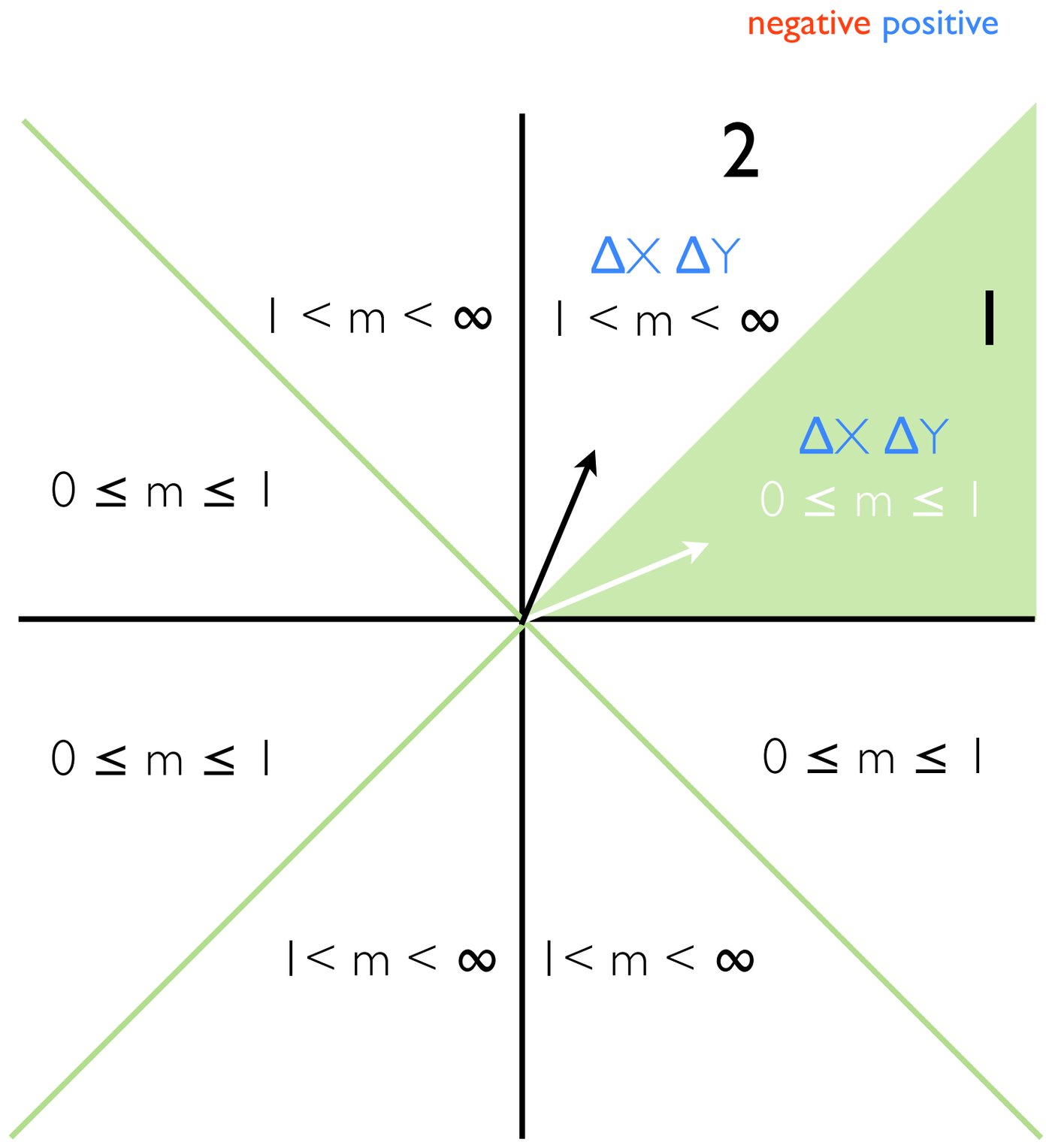
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \leq m \leq 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX**
and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

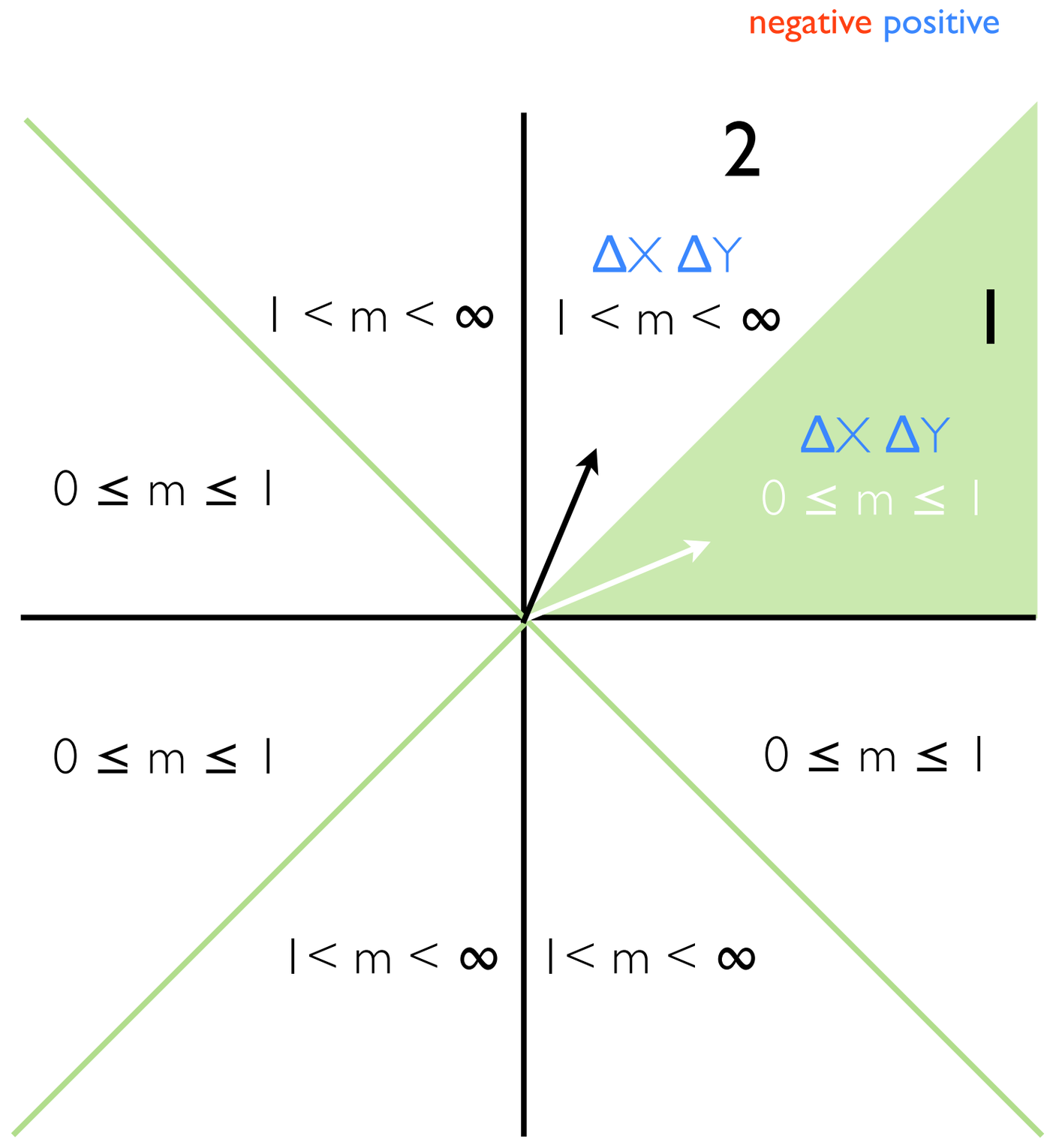
ΔY: **ΔY = abs(ΔY)**
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \le m \le 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.



Fixing Line Algorithm

m: calculate with original, unchanged **ΔY** and **ΔX**
and never recalculate it again, leave as it is here

ΔX: $x_2 < x_1$: change the orientation
swap starting and ending points:
x_2 with x_1, y_2 with y_1, **ΔX = abs(ΔX)**
This will cause the ΔX to become ΔX, 1st condition.

Octant conversions: 3 to 7, 4 to 8, 5 to 1, 6 to 2

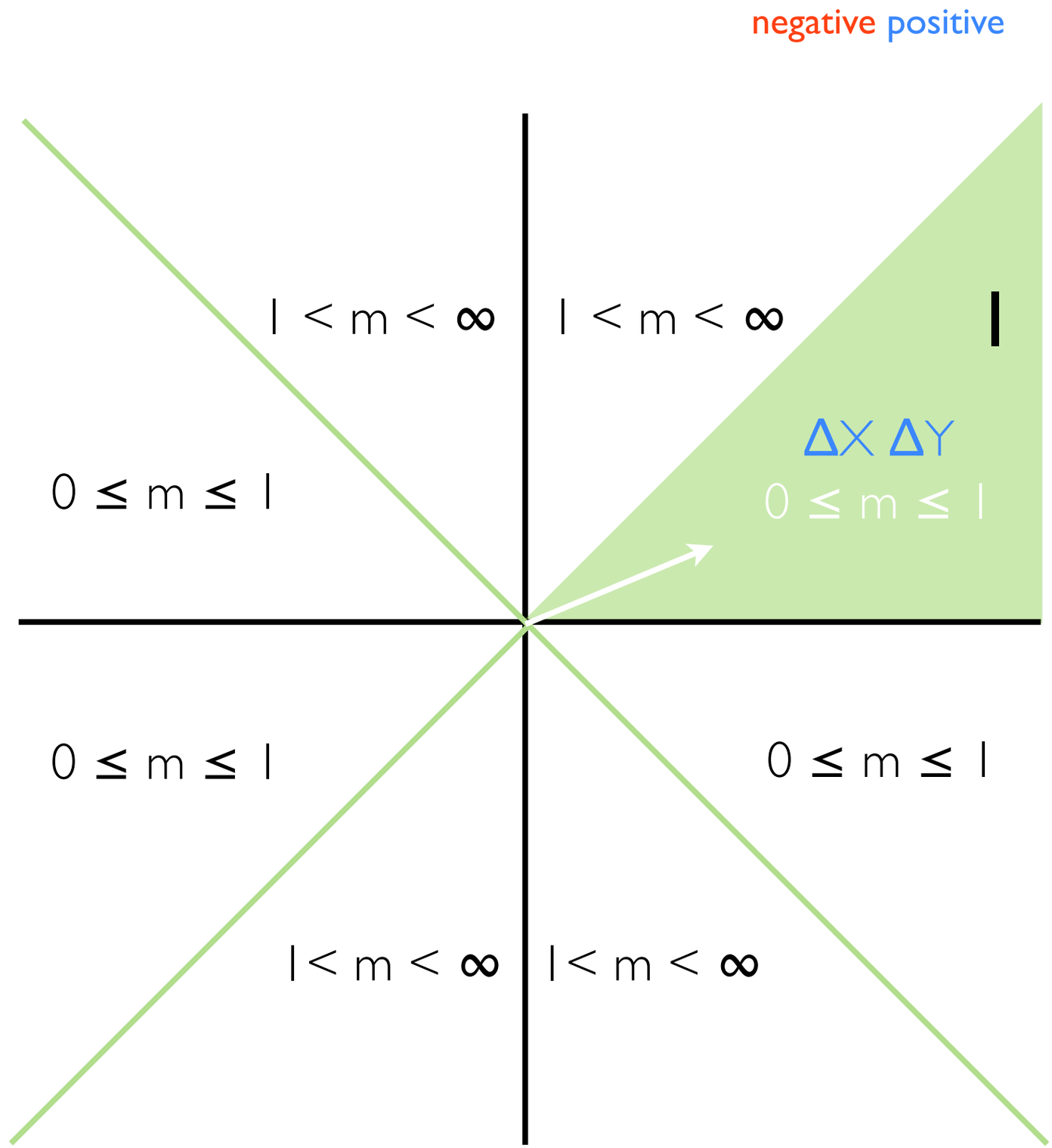
ΔY: ΔY = abs(ΔY)
This will cause the ΔY to become ΔY, 2nd condition.

Octant conversions: 7 to 2, 8 to 1

$1 < m < \infty$: make it $0 \leq m \leq 1$
Invert the coordinates:
x_2 with y_2, x_1 with y_1,
Store original m, because remembering it was > 1, will help swapping back the pixels when they are drawn
This will cause the placement of the points to represent a line with the slope in the acceptable range, fulfilling the 3rd condition

Octant conversions: 2 to 1

Now all lines are as in the 1st octant.

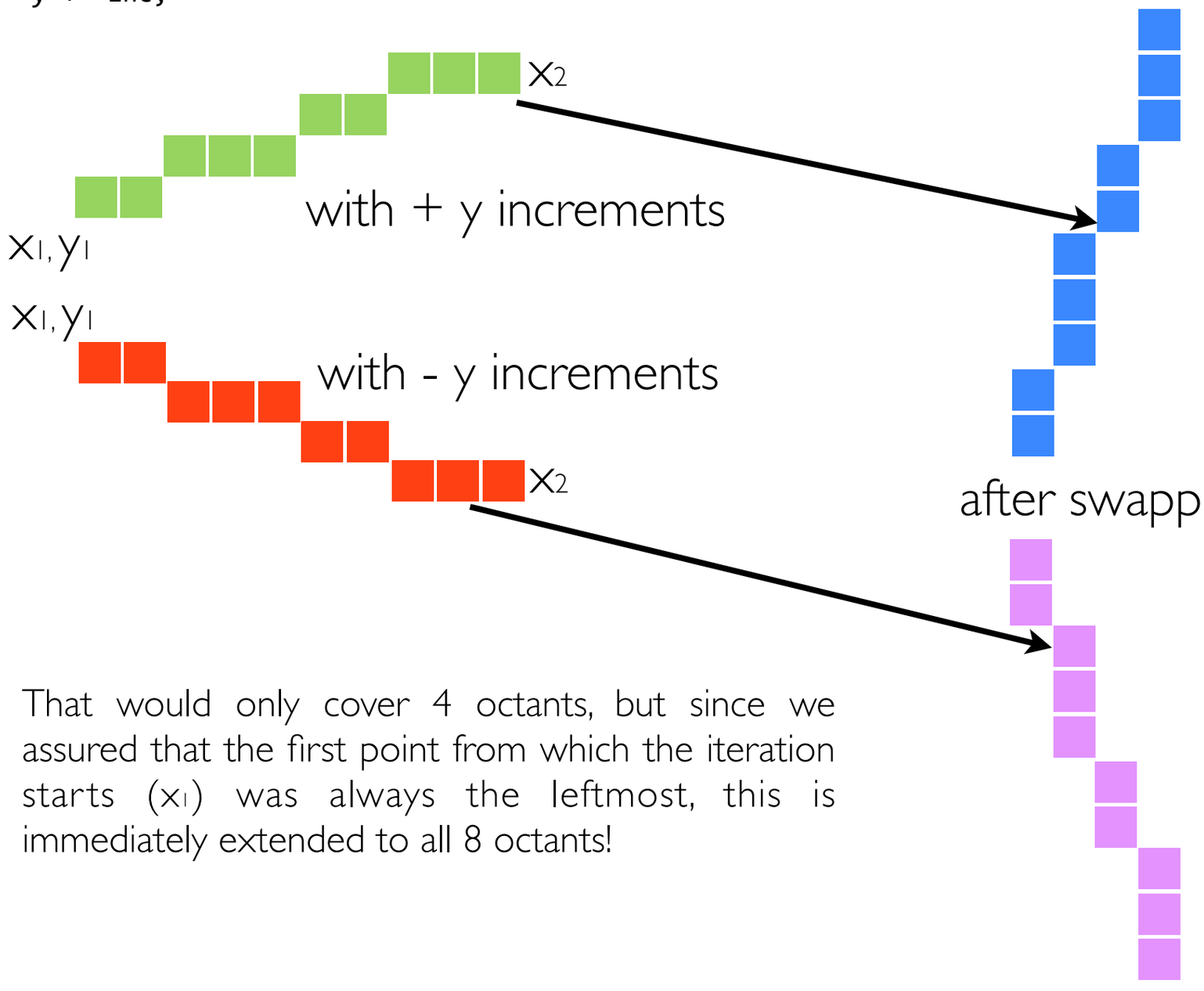


Fixing Line Algorithm

```
incE = 2 * dy;
incNE = 2 * dy - 2 * dx;
d = 2 * dy - dx;
y = y1;                          /1
for (x = x1; x <= x2; x++) {
   putpixel(x, y);/3
   if (d <= 0) {
      d += incE;
   } else {
      d += incNE;
      y += 1;/2
   }
}
```

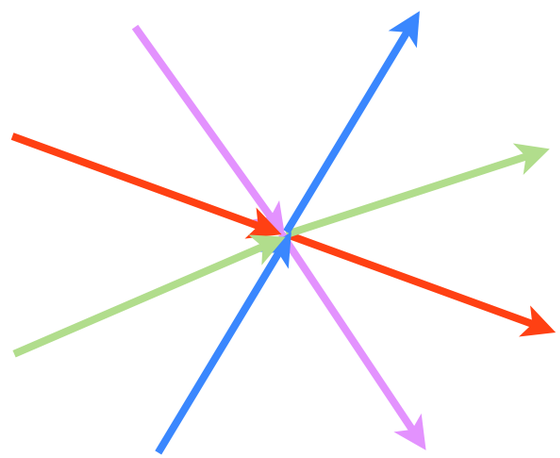
1. constantly grows in X (from x_1 to x_2)
2. **only sometimes grows in Y** (no limit, just a increment)
   2.1. if original **m < 0,** increment can be negative and still work

```
int inc = (m < 0)? -1 : 1;
...
y += inc;
```



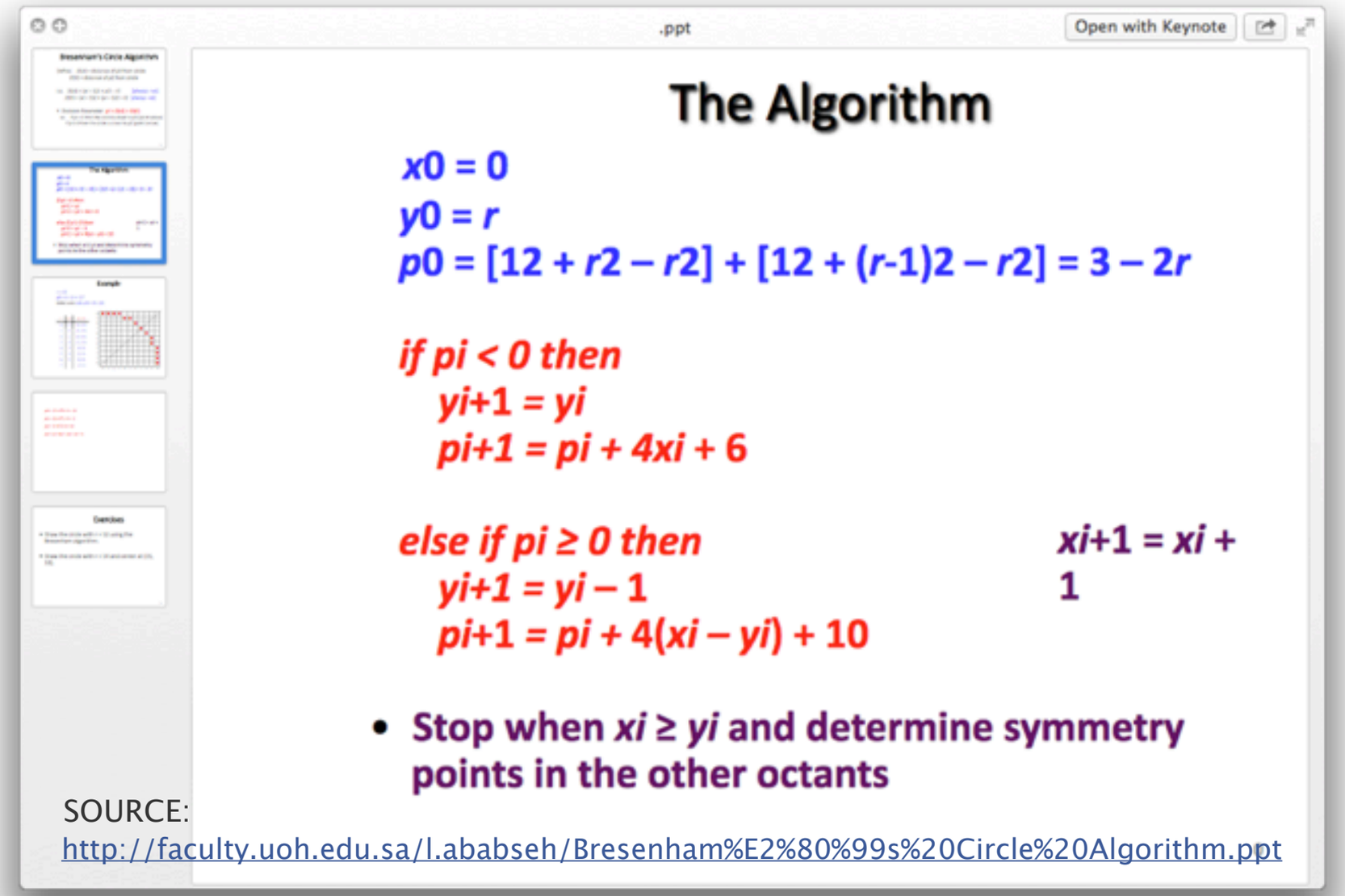
3. if original **abs(m) > 1**, swap back the coordinates

```
if(abs(m) > 1){
    putpixel(y,x);
}
else{
    putpixel(x,y);
}
```



That would only cover 4 octants, but since we assured that the first point from which the iteration starts (x_1) was always the leftmost, this is immediately extended to all 8 octants!

II. Circle Algorithm

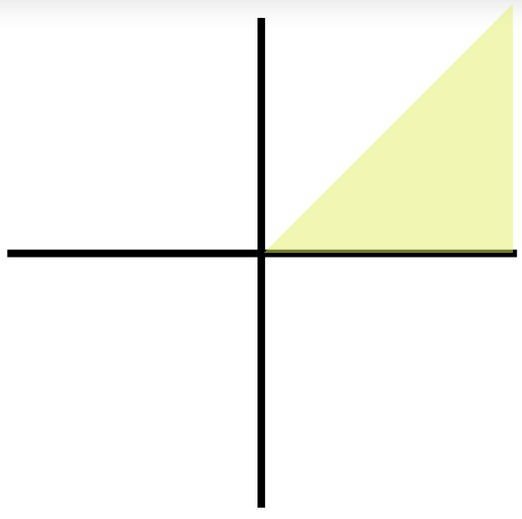


Considerations:

-Only works in 1st Octant

Properties of Octants:

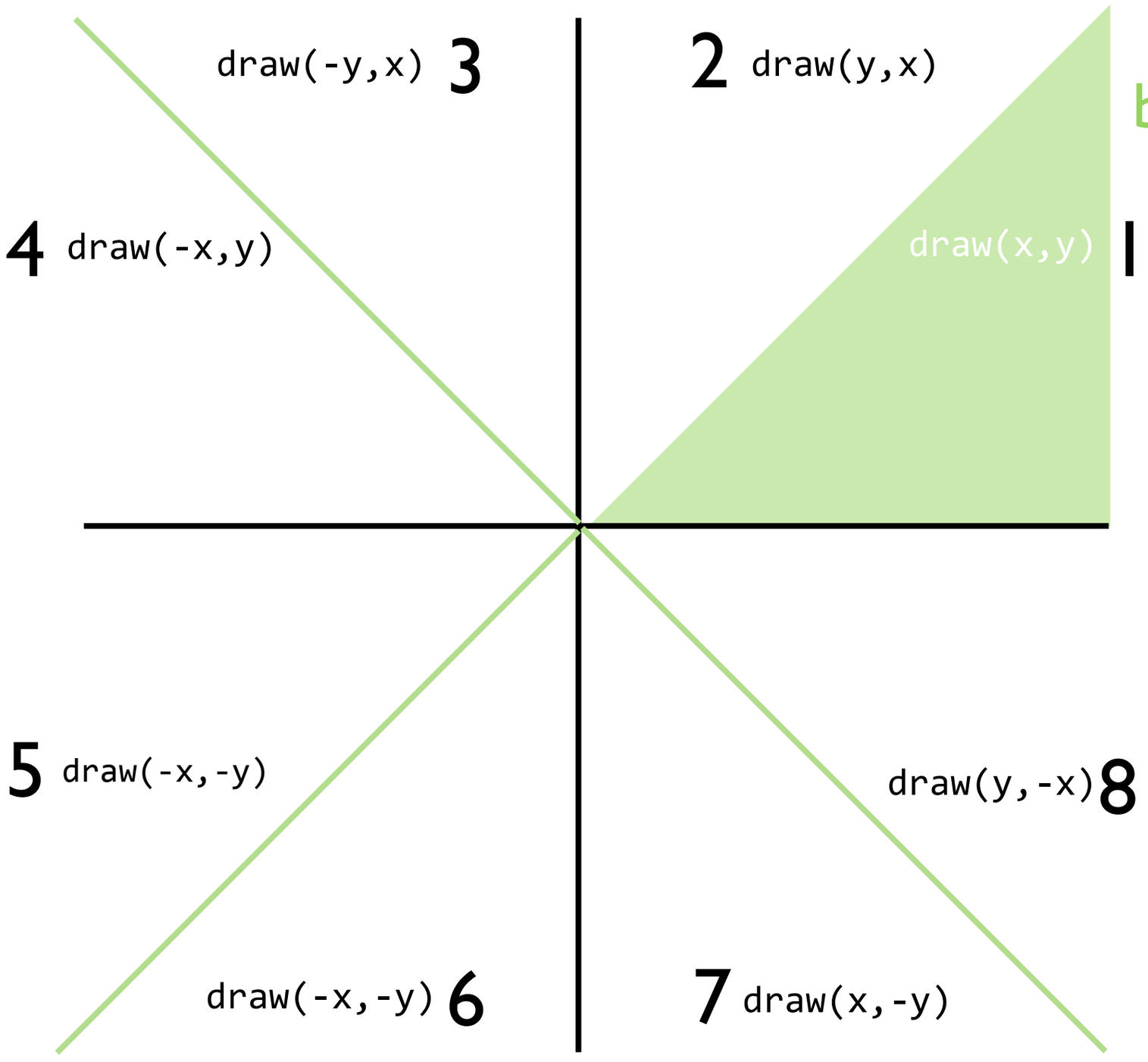
-All are symmetrical!



How to generalize for all octants?

Use the symmetry of the circle.

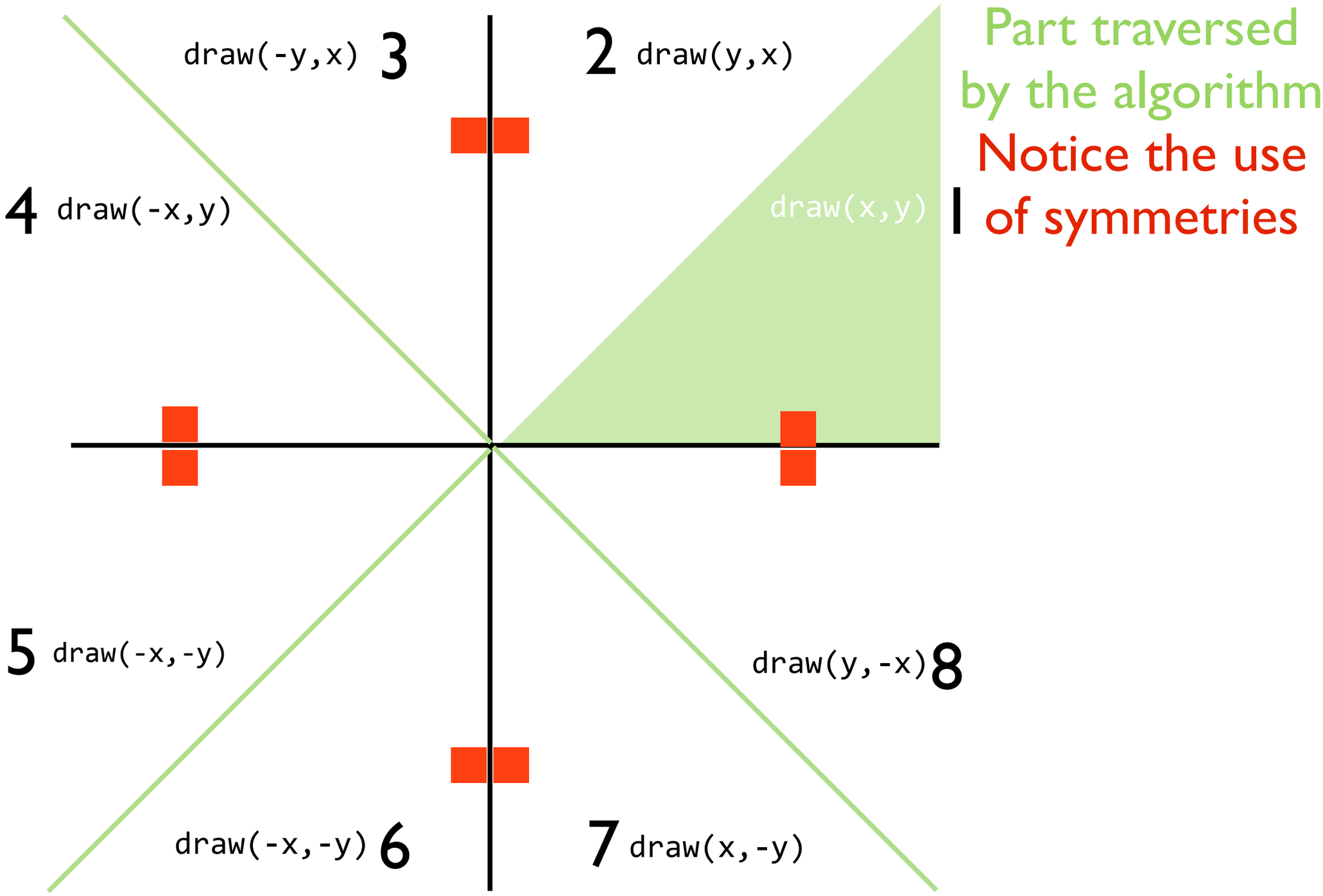
Fixing Circle Algorithm



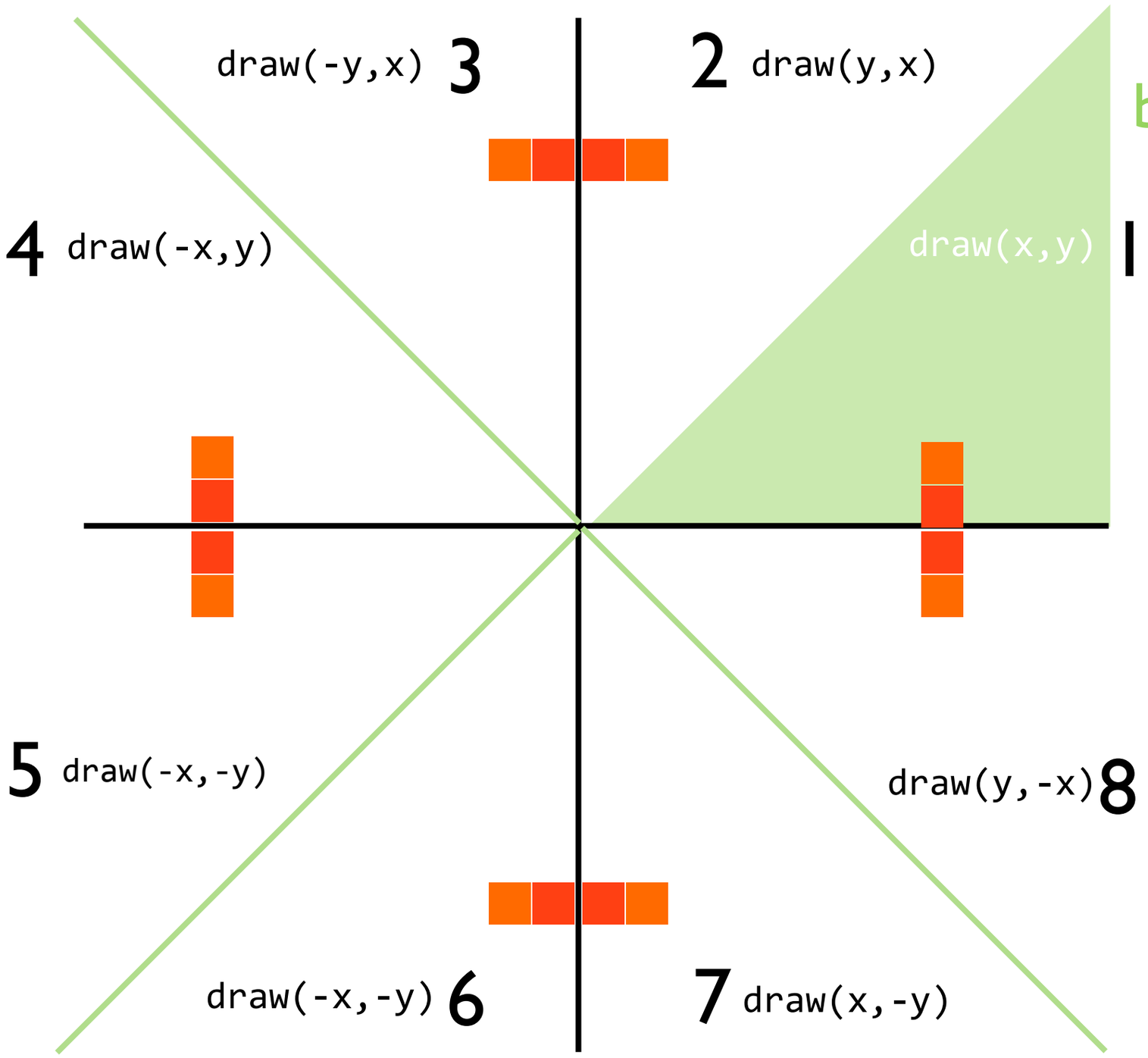
Part traversed by the algorithm

Notice the use of symmetries

Fixing Circle Algorithm



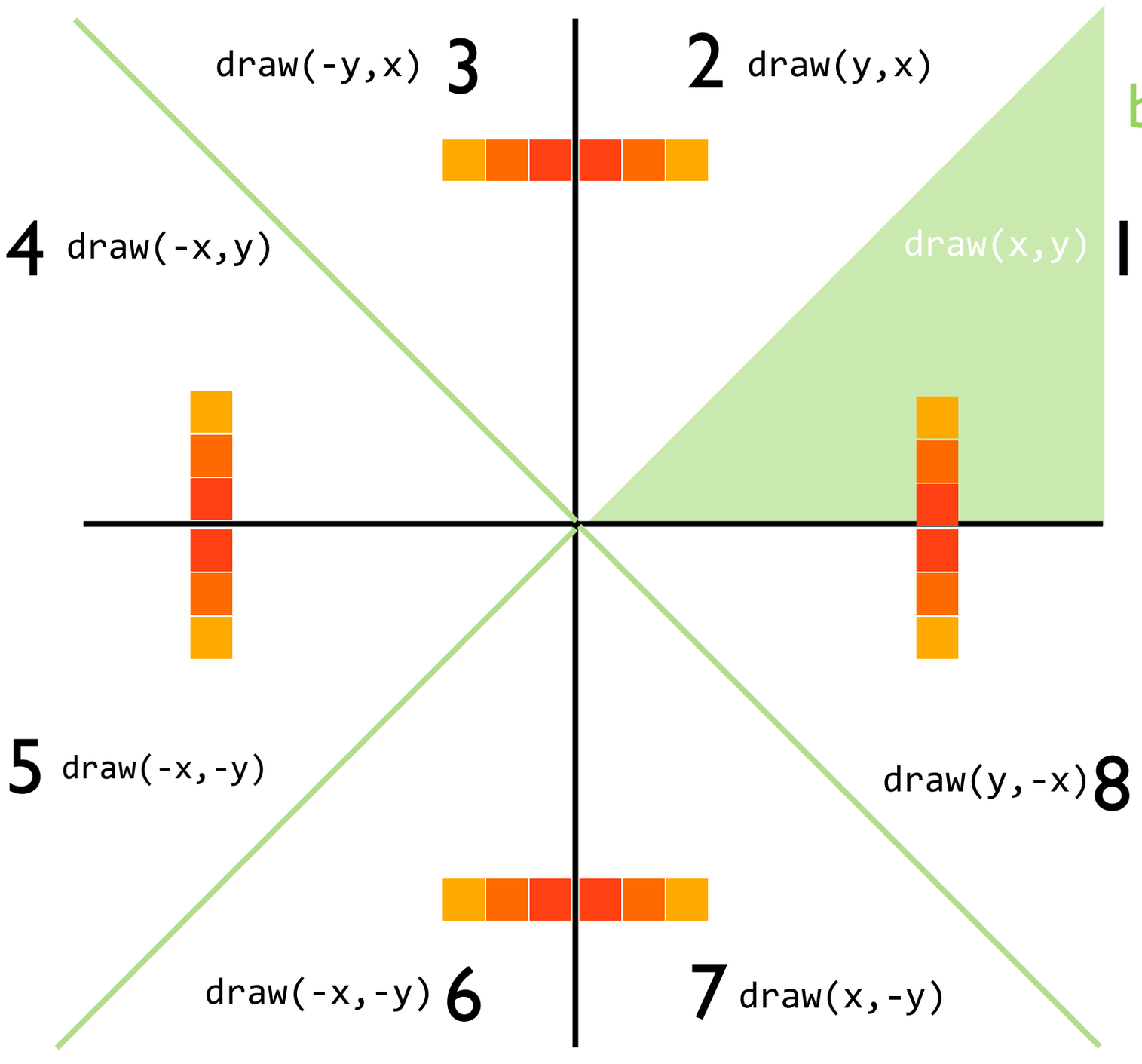
Fixing Circle Algorithm



Part traversed by the algorithm

Notice the use of symmetries

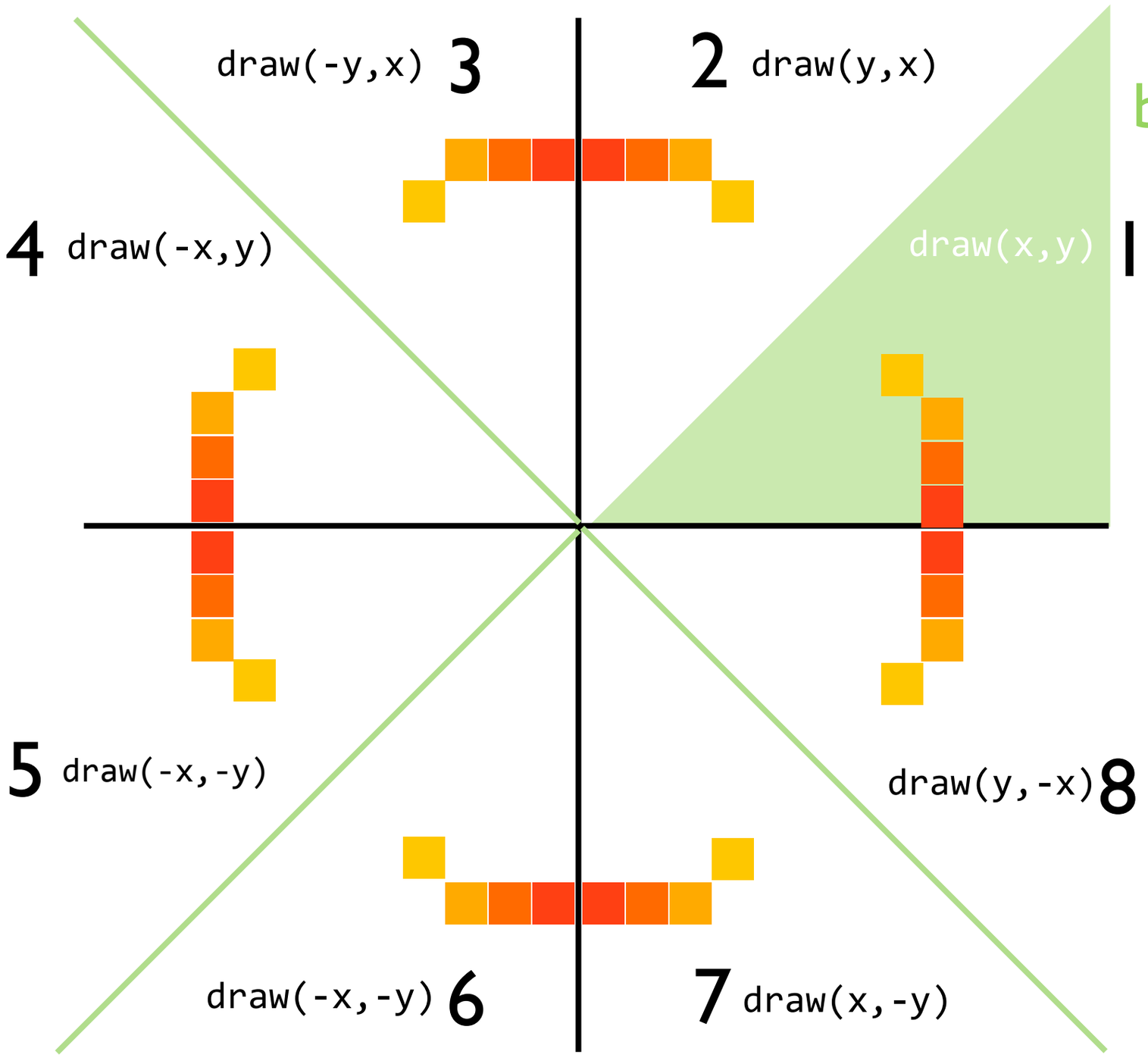
Fixing Circle Algorithm



Part traversed by the algorithm

Notice the use of symmetries

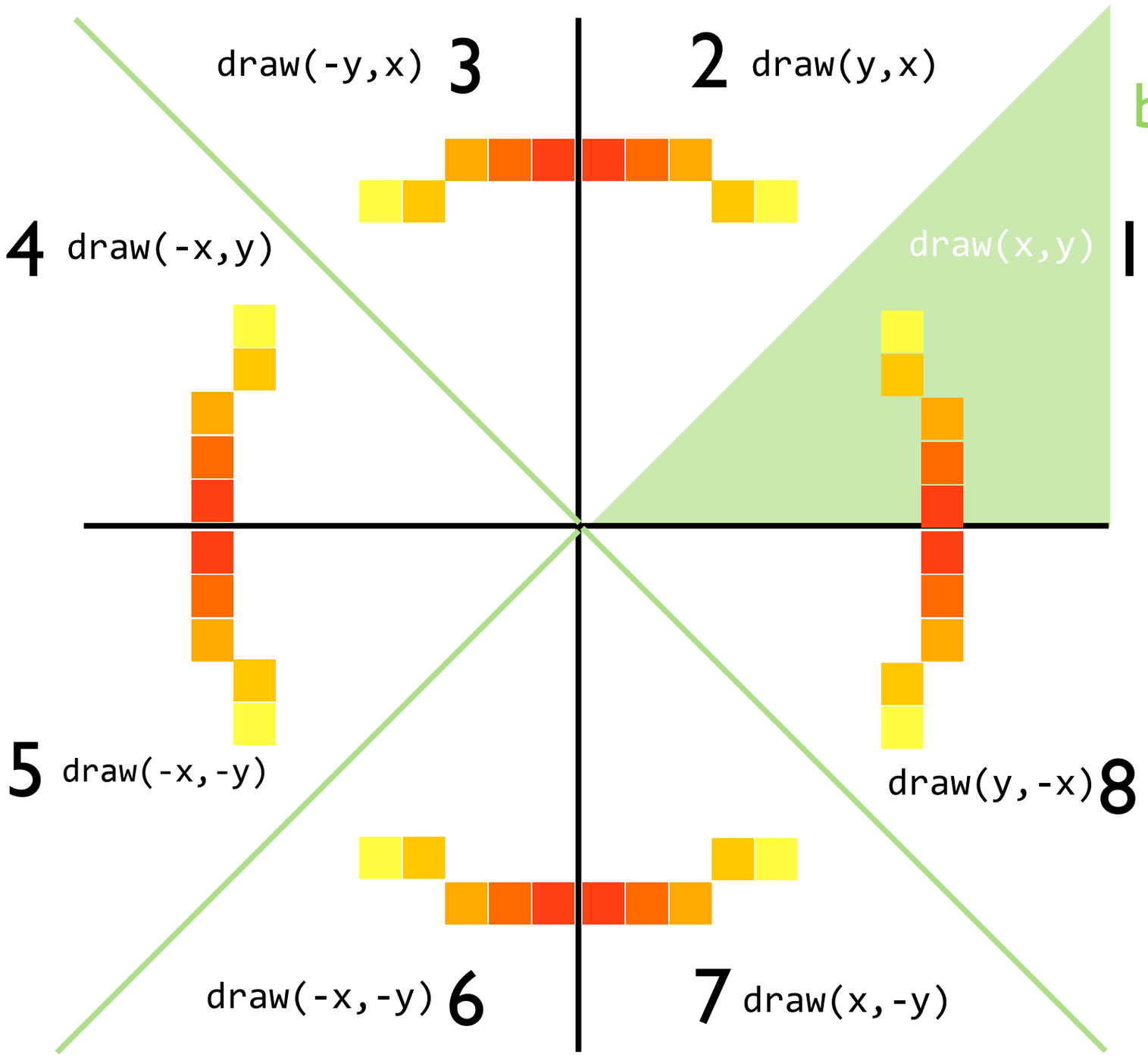
Fixing Circle Algorithm



Part traversed by the algorithm

Notice the use of symmetries

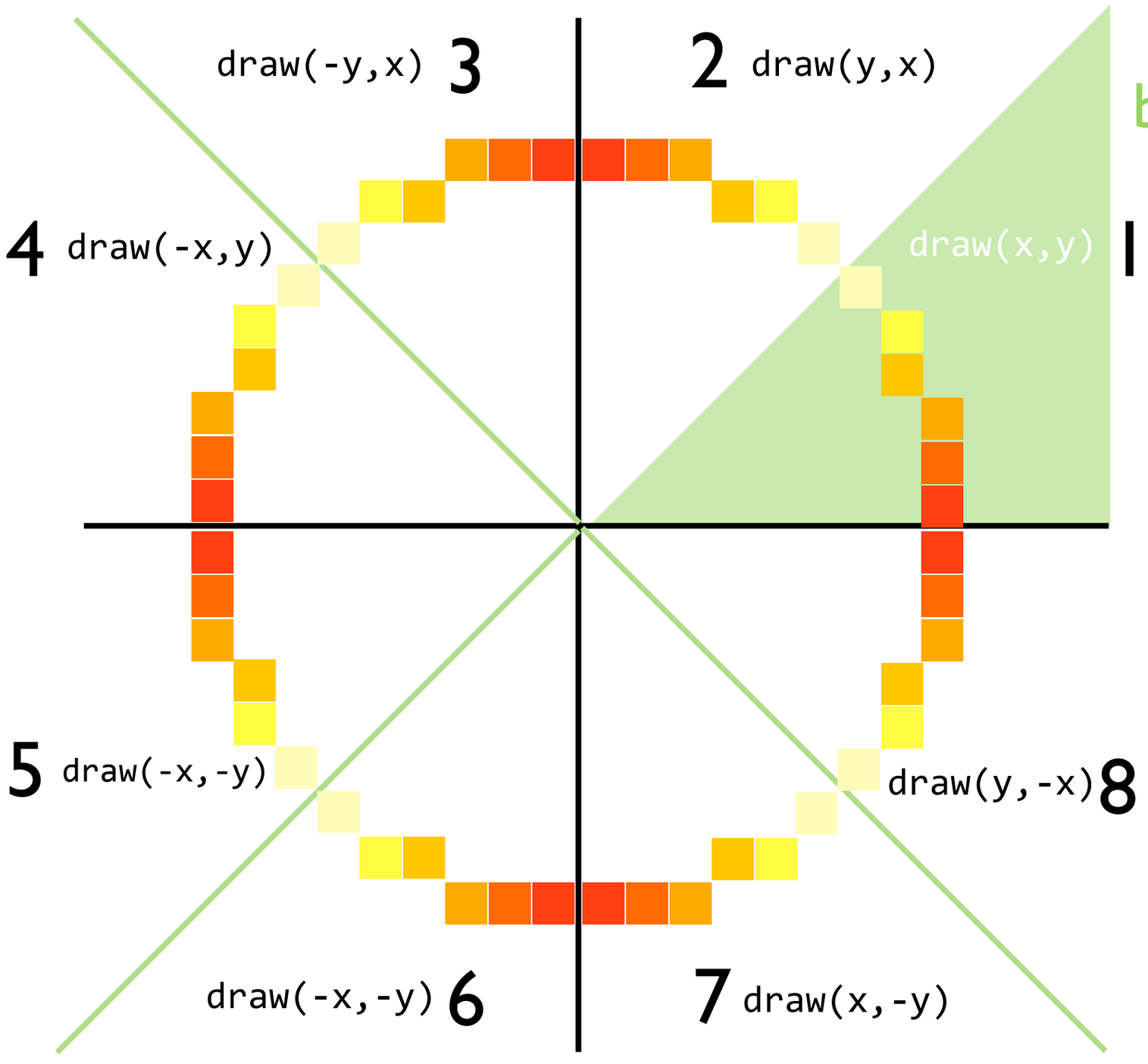
Fixing Circle Algorithm



Part traversed by the algorithm

Notice the use of symmetries

Fixing Circle Algorithm



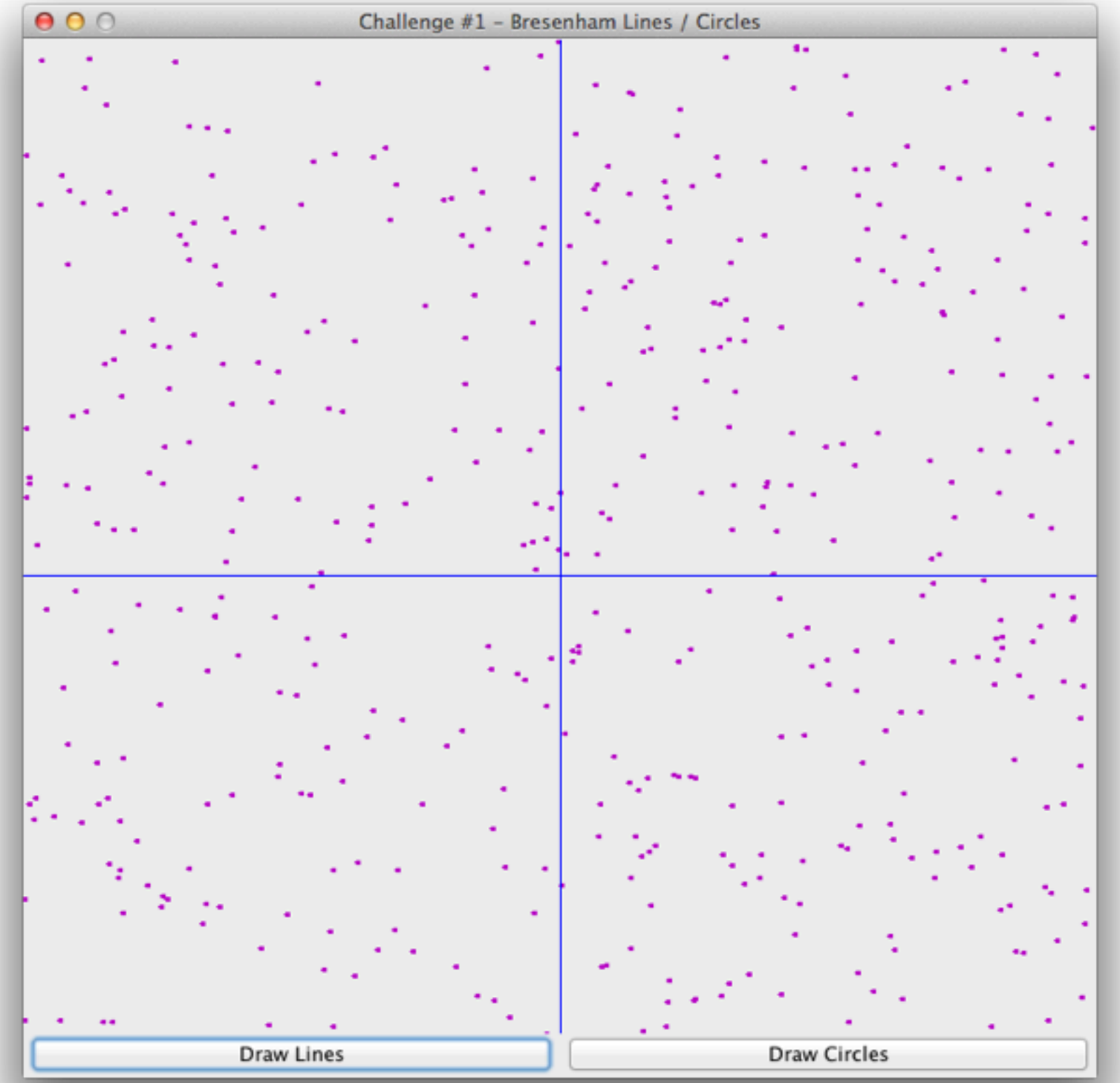
Part traversed by the algorithm

Notice the use of symmetries

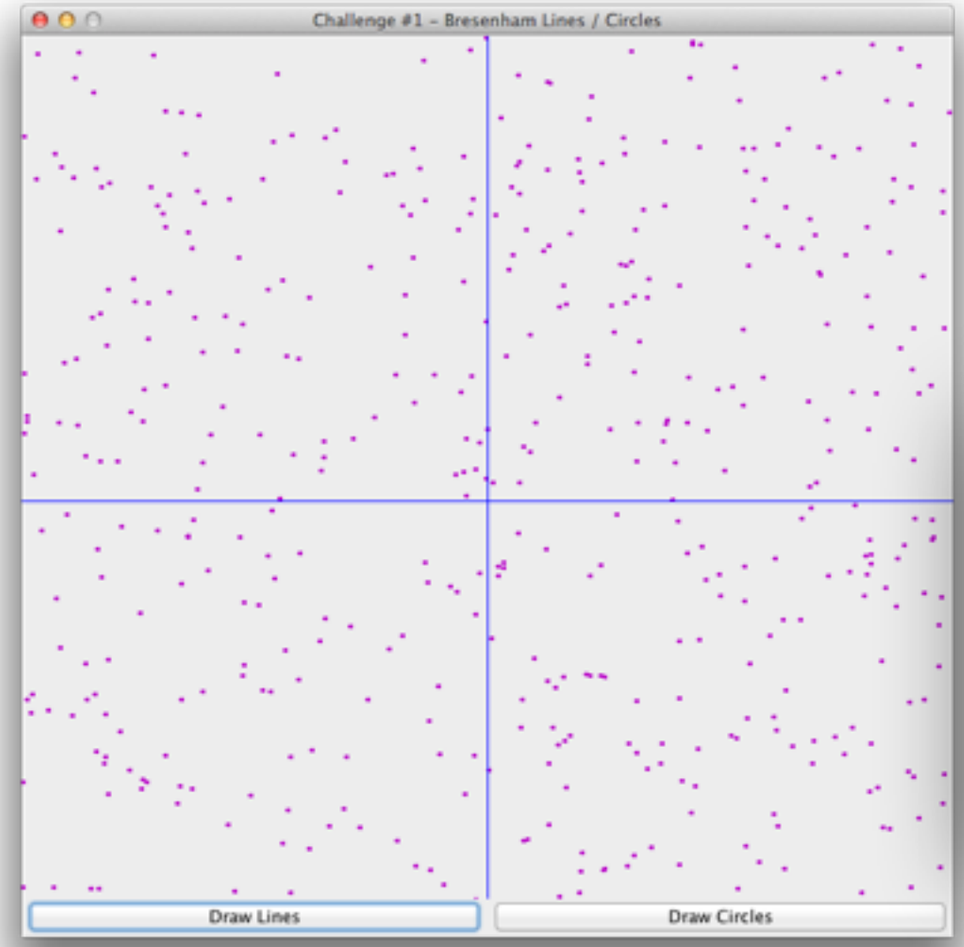
Advantages of Bresenham Variations

- Only uses addition/subtraction and comparisons, which are cheap operations while inside the loop.
- The control values which involve multiplication are computed only once, outside the loop, so their cost is almost meaningless.
- Control values use only integers, which are fast for operations and doesn't have the error induced by the use of floating point numbers.

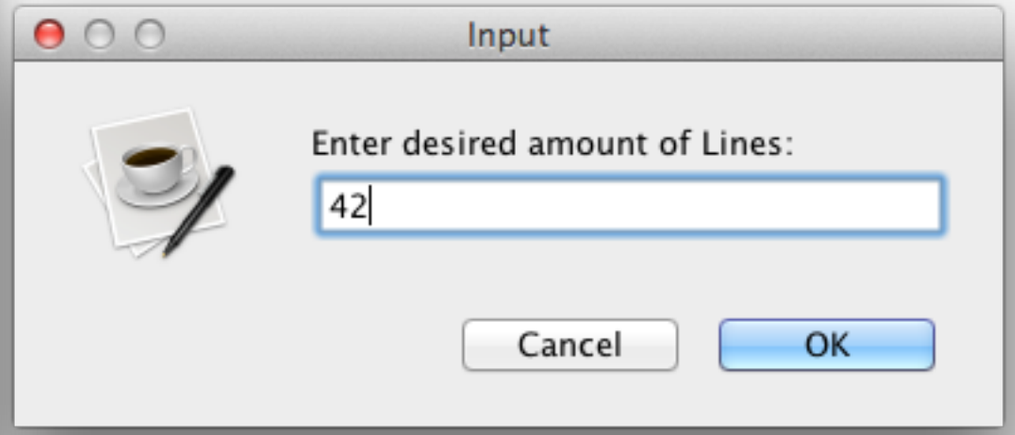
Use of the app

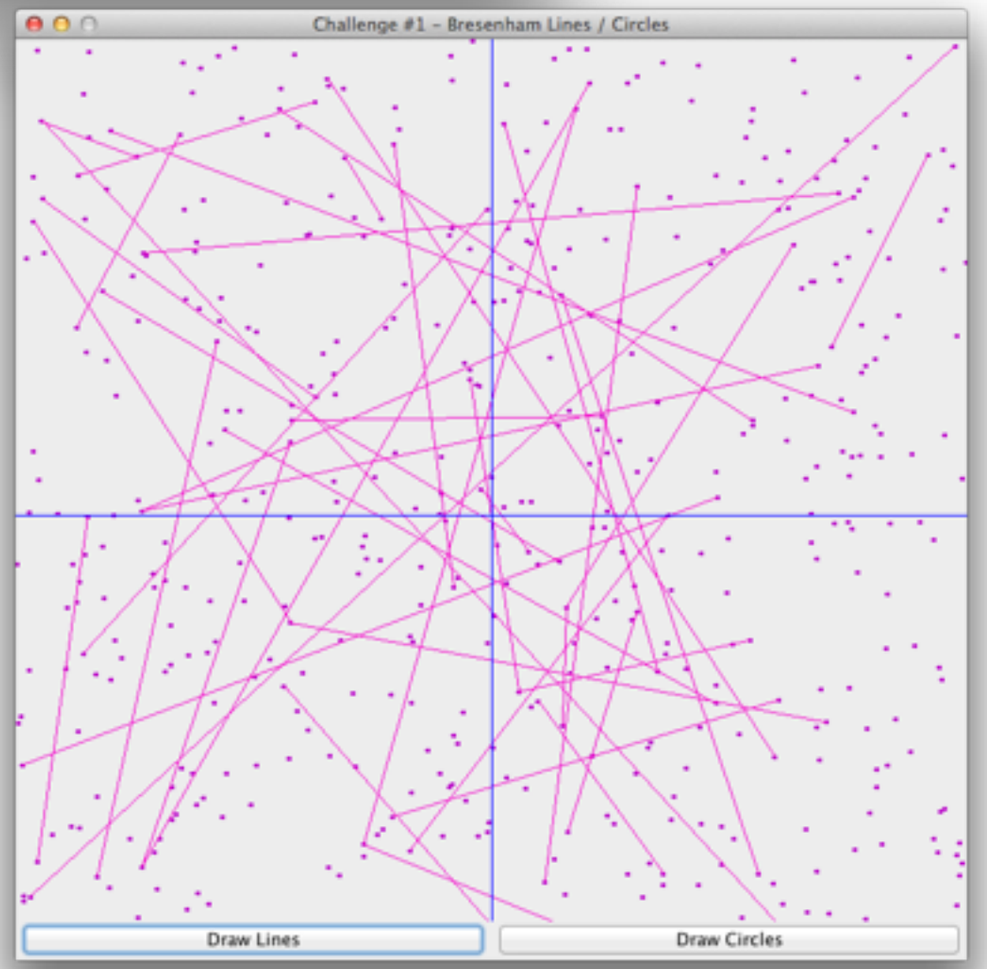


Draw Lines

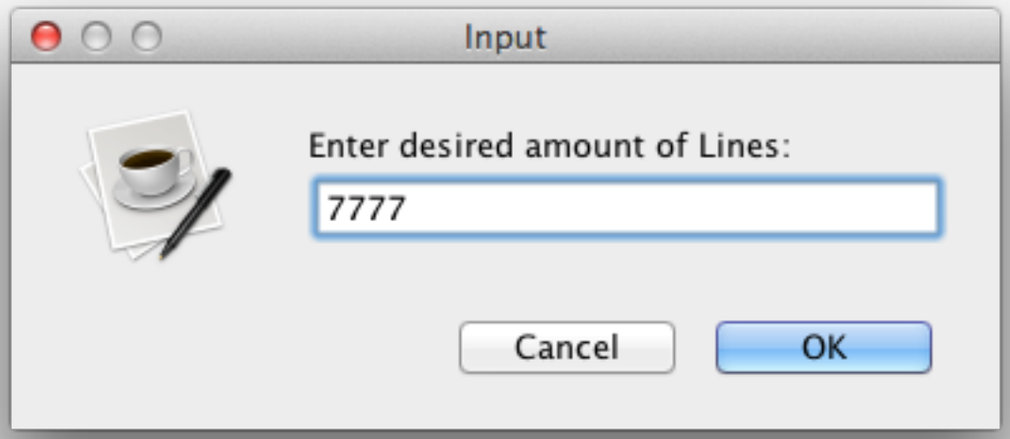


- By default opens in "Draw Lines" mode
- 500 points are generated randomly each time "Draw Lines" is pressed
- Each time the user clicks on the screen, a prompt will ask for a determinate amount of lines to be drawn, between the existing points.



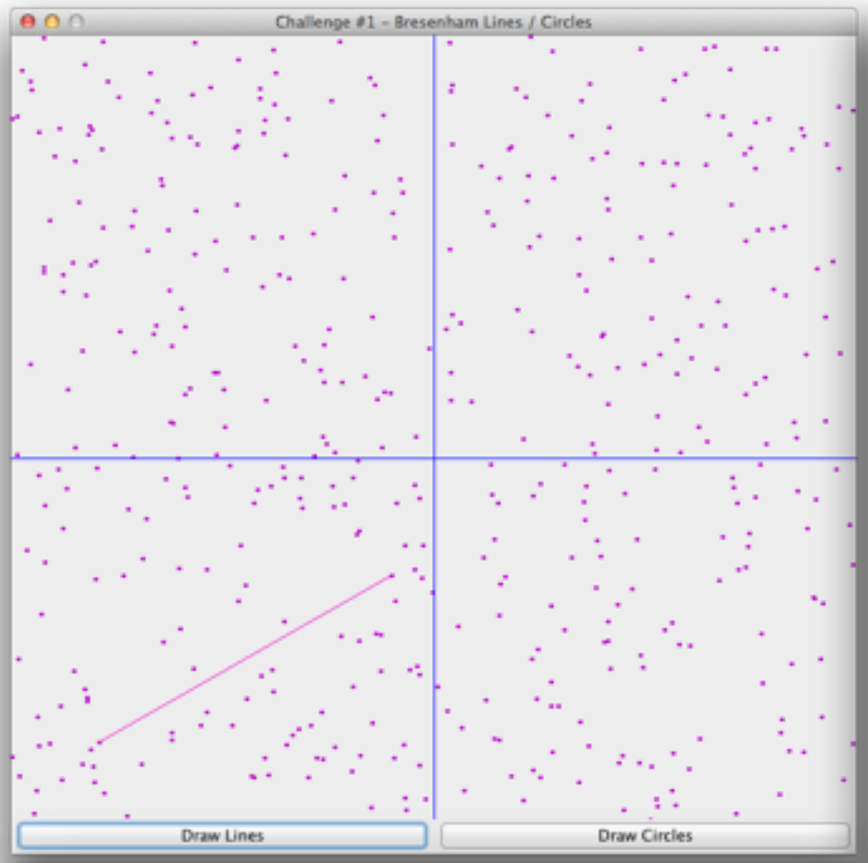


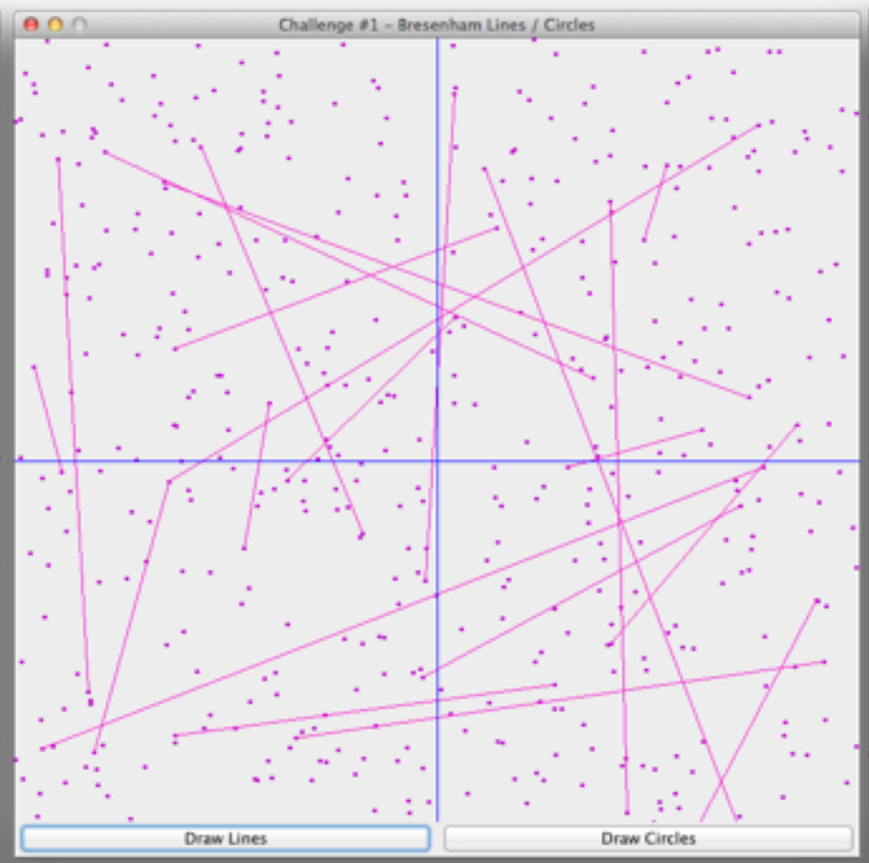
- Touching the screen again will ask for a new amount of lines, erasing all the previous ones, but maintaining the set of points.
- Entering an invalid value will result in 0 lines being drawn.
- Try drawing 7777 lines!

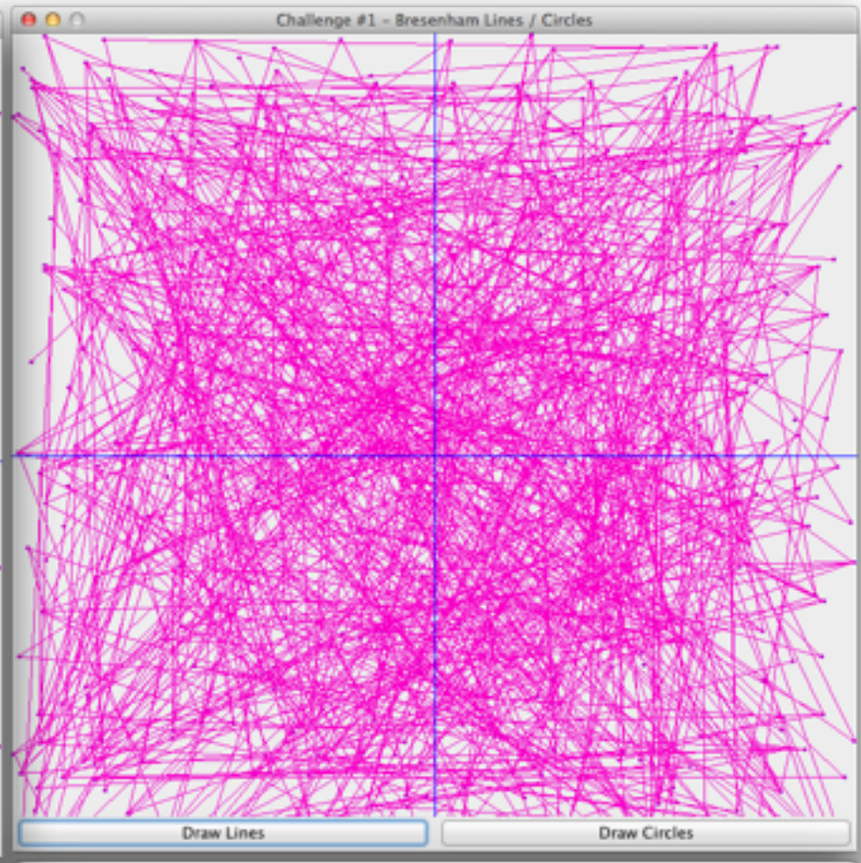


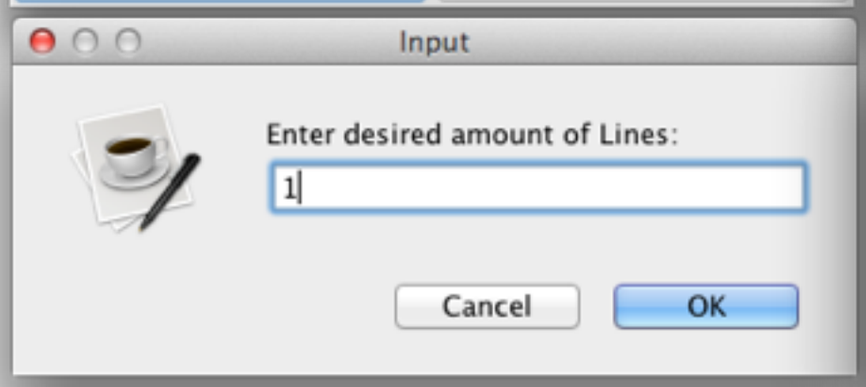
Trying different amounts of lines

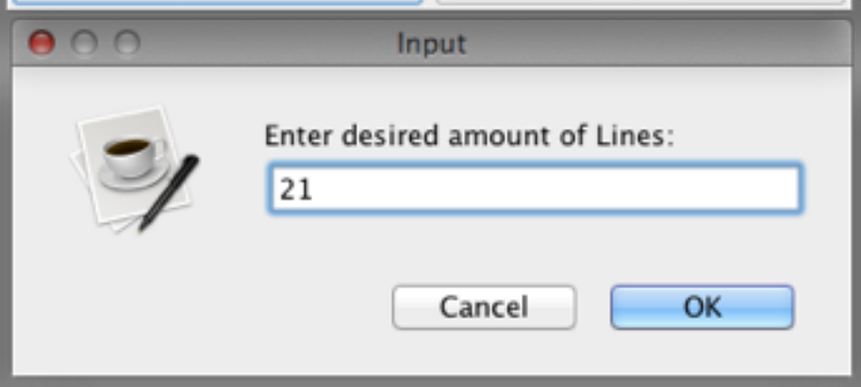
Even the two Axis are being drawn using Bresenham Algorithm!

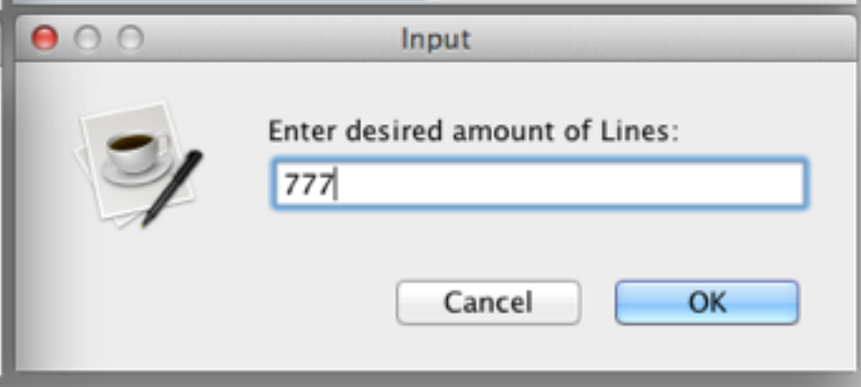


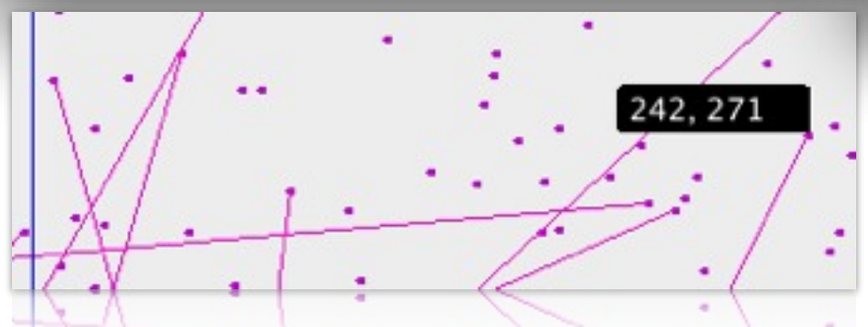








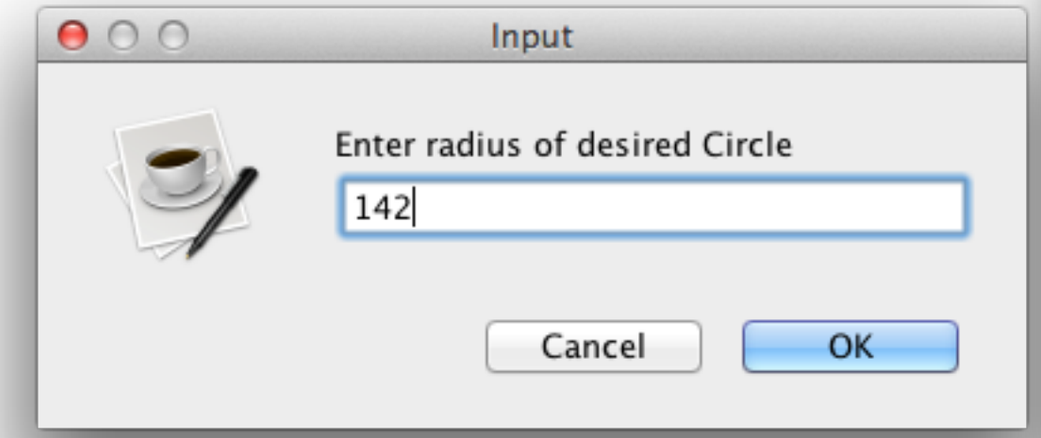


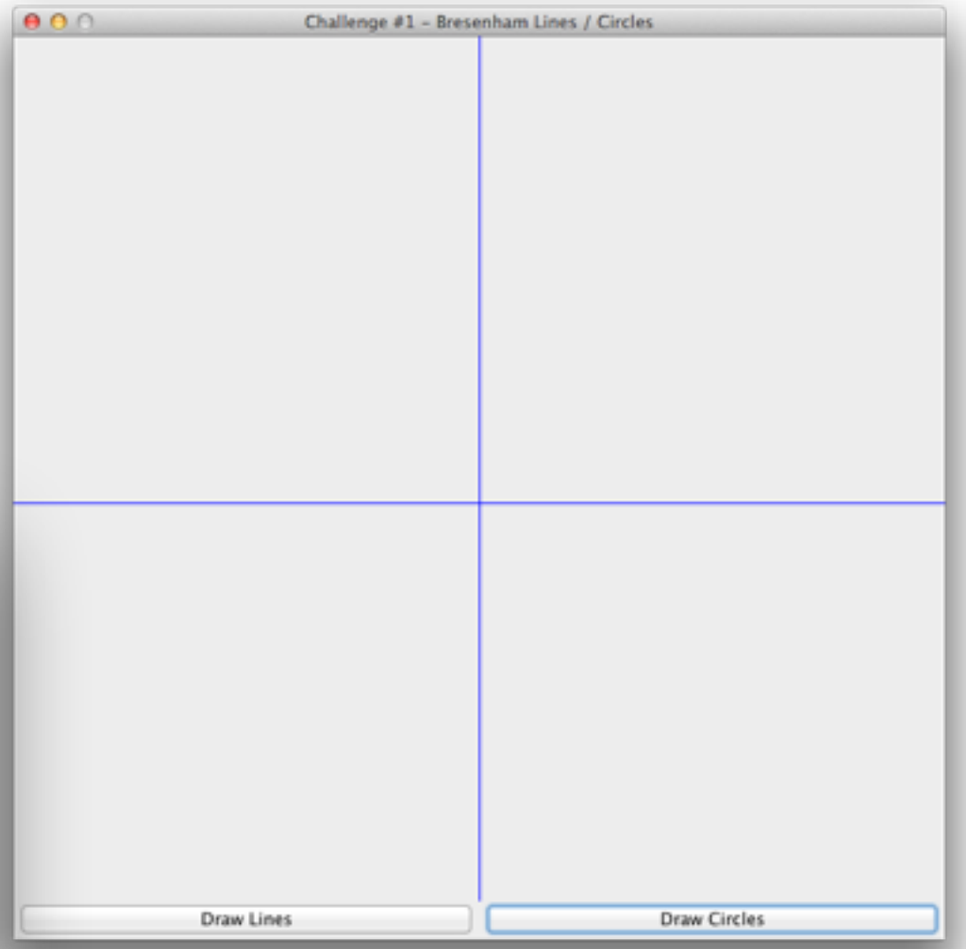


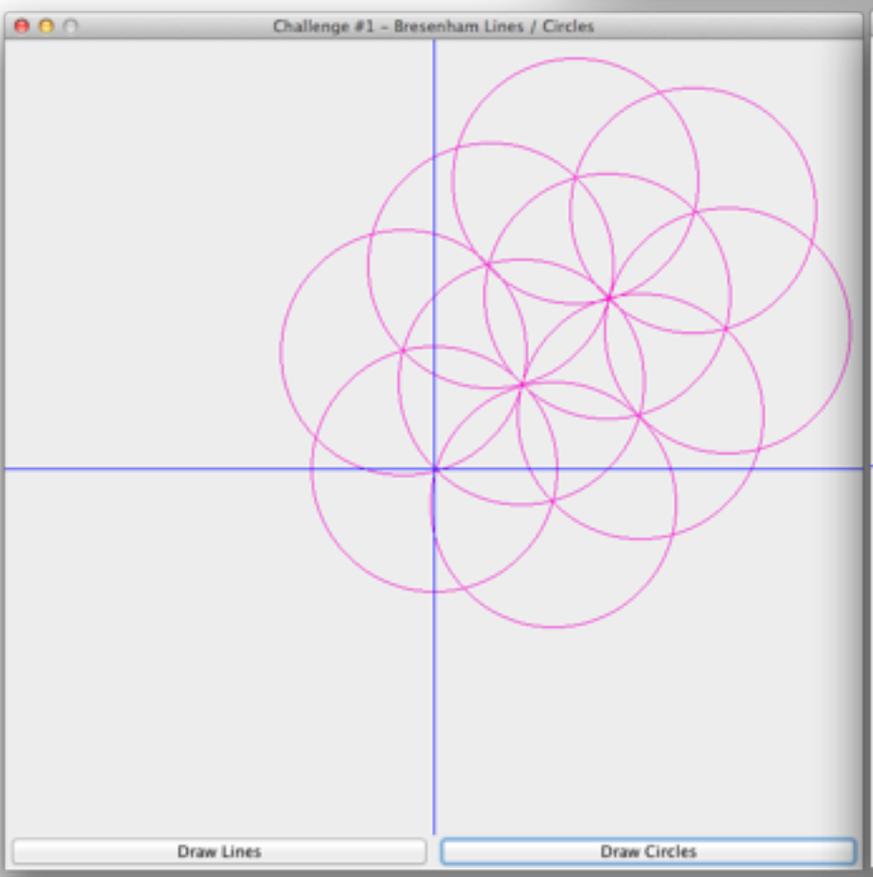
Also, a bubble containing the current pointer coordinate will follow the pointer. This will constantly refresh the screen, but the lines are not being constantly re-rendered. Once rendered, the current image is saved, so to avoid calling all the drawing methods until they are needed again.

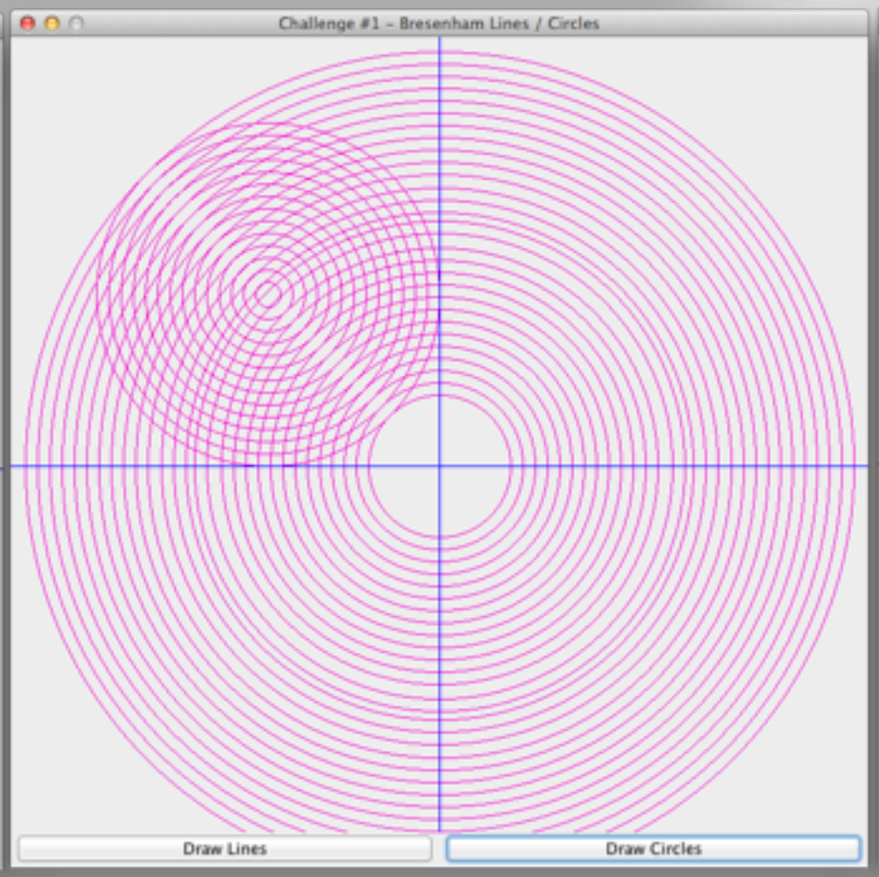
Draw Circles

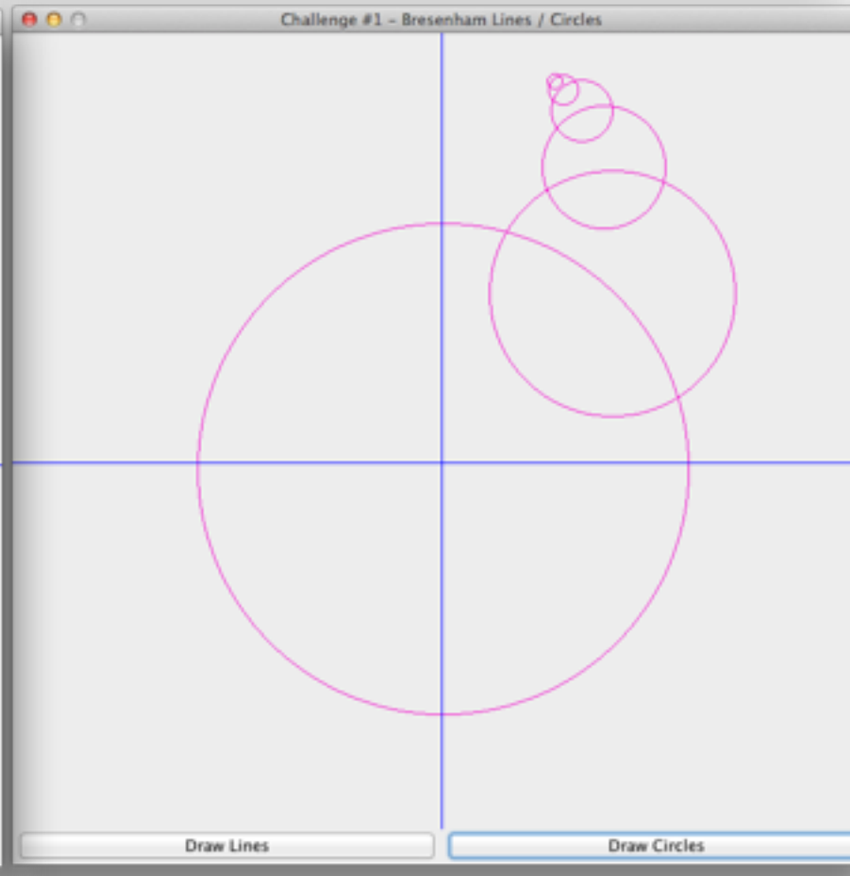
- Originally a clean canvas.
- Each time the user clicks on the screen, a prompt asks for the radius of a circle to be drawn.
- The program will draw a circle centered at the position of the click, with the given radius, using the Bresenham Circle Algorithm.
- Try different sizes for beautiful results!











Thanks for your time!

More challenges to follow!