Midpoint Circle Drawing Algorithm

Note: *You must learn MPL before proceeding into this.*

This one is particularly easier than midpoint line drawing algorithm, in the sense that the formula for dinit is really simple, and you don't need to do zone conversion twice, but only once. However, updating the value of decision variable "d" can be a bit lengthier than MPL as the formula for those are bigger and complex (it is because you're following a curve now, not a straight line).

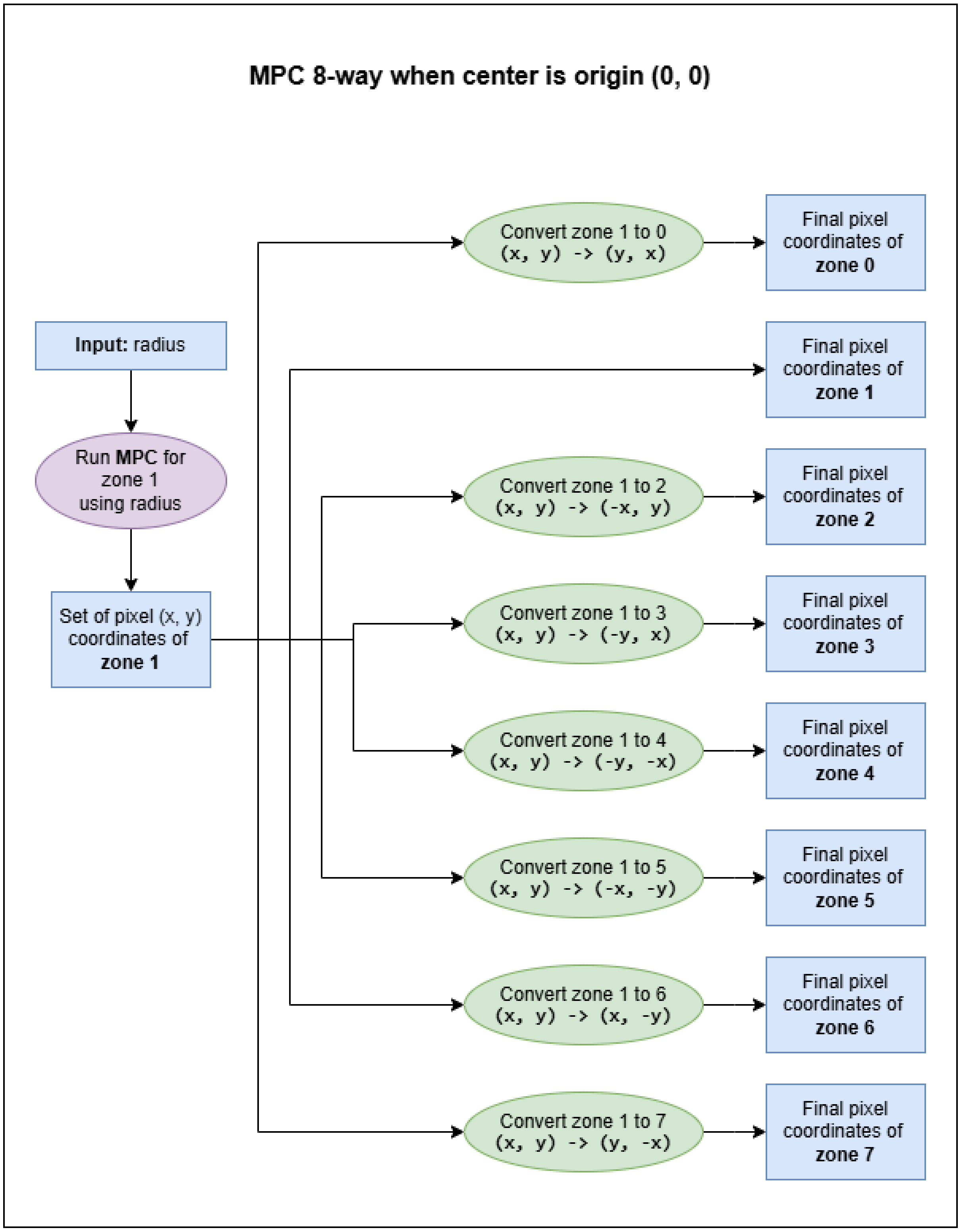
You'll be given these things: the radius of the circle, and the center. If the center is not given, you'll automatically assume that the center is at origin (0, 0). If the center is at origin, the workflow becomes shorter.

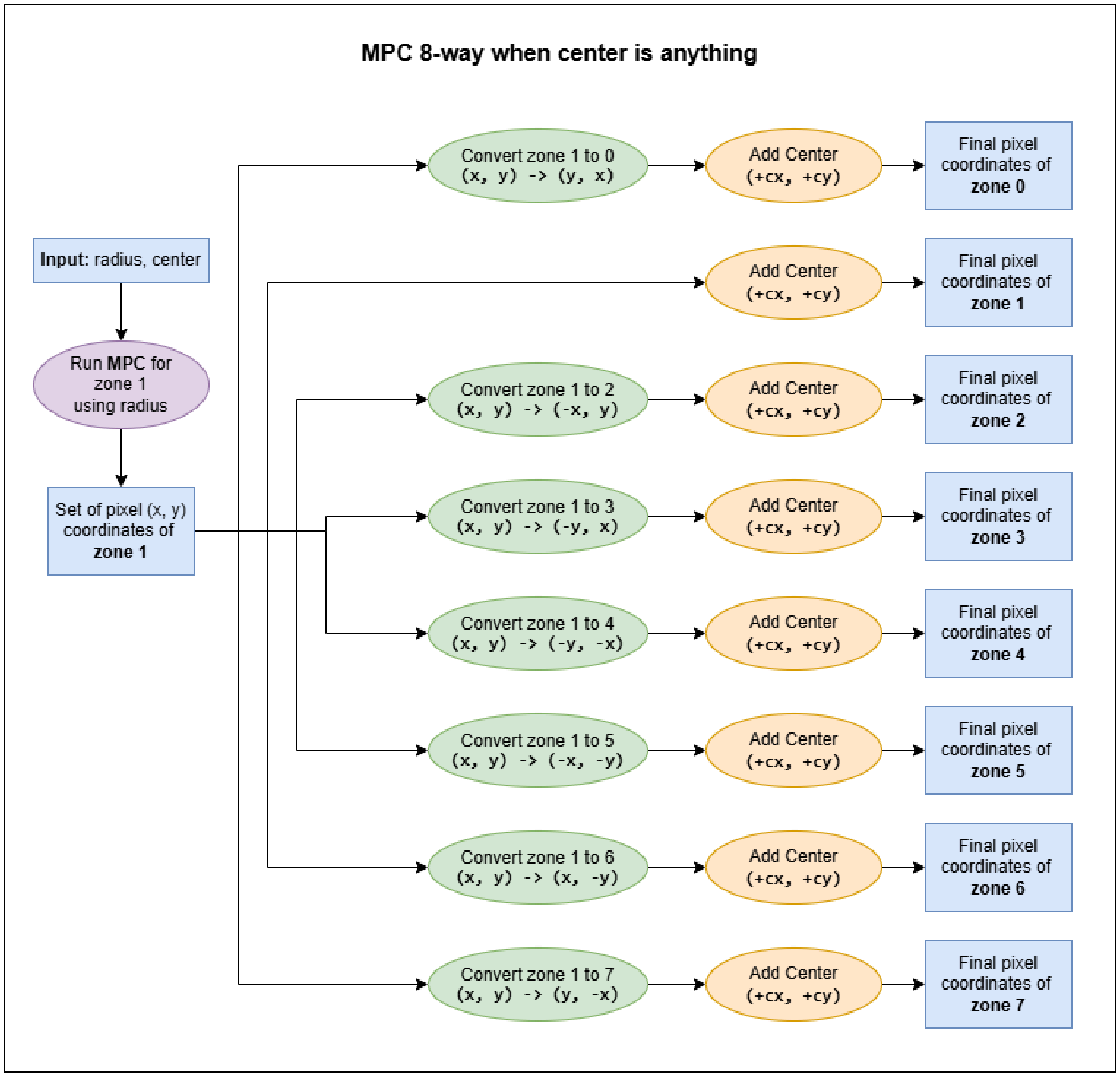
This algo (at least our our specific implementation) generates points in **zone 1**. It **starts at (0, radius) point**, and goes until the value of x becomes greater than y.

We apply **8-way symmetry** to quickly generate points for rest of the 7 zones. So we get 8 sets of points, for all 8 zones. All these points construct a full circle. Zone conversion happens here.

Lastly, **we add the center coordinate** to each and every points we've generated till now. This moves our circle to our desired center. This step **can be ignored** if the center is origin (0, 0).

Using the MPC algorithm, we can not only draw whole circles, but also half-circles, quarter-circles, arcs etc. by enabling only certain zones and ignoring the rest.


MPC 8-way when center is origin (0, 0)
Input: radius
Run MPC for zone 1 using radius
Set of pixel (x, y) coordinates of zone 1
Convert zone 1 to 0 (x, y) -> (y, x)
Convert zone 1 to 2 (x, y) -> (-x, y)
Convert zone 1 to 3 (x, y) -> (-y, x)
Convert zone 1 to 4 (x, y) -> (-y, -x)
Convert zone 1 to 5 (x, y) -> (-x, -y)
Convert zone 1 to 6 (x, y) -> (x, -y)
Convert zone 1 to 7 (x, y) -> (y, -x)
Final pixel coordinates of zone 0
Final pixel coordinates of zone 1
Final pixel coordinates of zone 2
Final pixel coordinates of zone 3
Final pixel coordinates of zone 4
Final pixel coordinates of zone 5
Final pixel coordinates of zone 6
Final pixel coordinates of zone 7


MPC 8-way when center is anything
Input: radius, center
Run MPC for zone 1 using radius
Set of pixel (x, y) coordinates of zone 1
Convert zone 1 to 0 (x, y) -> (y, x)
Convert zone 1 to 2 (x, y) -> (-x, y)
Convert zone 1 to 3 (x, y) -> (-y, x)
Convert zone 1 to 4 (x, y) -> (-y, -x)
Convert zone 1 to 5 (x, y) -> (-x, -y)
Convert zone 1 to 6 (x, y) -> (x, -y)
Convert zone 1 to 7 (x, y) -> (y, -x)
Add Center (+cx, +cy)
Final pixel coordinates of zone 0
Final pixel coordinates of zone 1
Final pixel coordinates of zone 2
Final pixel coordinates of zone 3
Final pixel coordinates of zone 4
Final pixel coordinates of zone 5
Final pixel coordinates of zone 6
Final pixel coordinates of zone 7

PseudoCode

```
Procedure CirclePoints(x, y, cx, cy)
    SetPixel( x + cx,  y + cy)  // zone 1 (our native zone!)
    SetPixel( y + cx,  x + cy)  // zone 0
    SetPixel( y + cx, -x + cy)  // zone 7
    SetPixel( x + cx, -y + cy)  // zone 6
    SetPixel(-x + cx, -y + cy)  // zone 5
    SetPixel(-y + cx, -x + cy)  // zone 4
    SetPixel(-y + cx,  x + cy)  // zone 3
    SetPixel(-x + cx,  y + cy)  // zone 2
End Procedure

Procedure MidpointCircle(radius, cx, cy)
    d ← 1 - radius // decision variable

    // start at (0, r)
    x ← 0
    y ← radius

    CirclePoints(x, y, cx, cy)  // plot the initial points

    While x < y Do
        If d < 0 Then
            // Move to east pixel
            d ← d + 2*x + 3   // update decision variable
            x ← x + 1
        Else
            // Move to south-east pixel
            d ← d + 2*x - 2*y + 5 // update decision variable
            x ← x + 1
            y ← y - 1
        End If
        CirclePoints(x, y, cx, cy)  // plot the new points
    End While
End Procedure
```

Important: *You must update the decision variable d before changing the value of x and y, since the update of d is dependent on the value of x and y (this is not the case in Midpoint Line Algo).*

Example 1: A circle with radius 6 and centre at (0, 0) a.k.a. origin.

`dinit = 1-r = 1-6 = -5`

`starting point = (0, r) = (0, 6)`

Now make the table. We stop when we reach x > y.

(x, y)	d	neighbor	d (new)
(0, 6)	-5 (dinit)	E	-2
(1, 6)	-2	E	3
(2, 6)	3	SE	0
(3, 5)	0	SE	1
(4, 4)	1	SE	---

This generates points or pixel coordinates for **zone 1**. But to make the rest of the circle, we need zone conversion via 8-way symmetry.

zone 1	zone 0	zone 2	zone 3	zone 4	zone 5	zone 6	zone 7
(0, 6)	(6, 0)	(0, 6)	(-6, 0)	(-6, 0)	(0, -6)	(0, -6)	(6, 0)
(1, 6)	(6, 1)	(-1, 6)	(-6, 1)	(-6, -1)	(-1, -6)	(1, -6)	(6, -1)
(2, 6)	(6, 2)	(-2, 6)	(-6, 2)	(-6, -2)	(-2, -6)	(2, -6)	(6, -2)
(3, 5)	(5, 3)	(-3, 5)	(-5, 3)	(-5, -3)	(-3, -5)	(3, -5)	(5, -3)
(4, 4)	(4, 4)	(-4, 4)	(-4, 4)	(-4, -4)	(-4, -4)	(4, -4)	(4, -4)

Notice that the zone 1 values are just copied in, it is because our algo worked in that zone and no conversion is necessary. The rest zones are converted via the "CirclePoints" function, which applies the 8-way symmetry.

Since our circle's center is (0, 0), we **don't need** to add the center coordinate value to all the values in the table above. So, this is our answer.

Example 2: A circle with radius 6 and centre at (4, 3).

`dinit = 1-r = 1-6 = -5`

`starting point = (0, r) = (0, 6)`

Now make the table. We stop when we reach x > y.

(x, y)	d	neighbor	d (new)
(0, 6)	-5 (dinit)	E	-2
(1, 6)	-2	E	3
(2, 6)	3	SE	0
(3, 5)	0	SE	1
(4, 4)	1	SE	---

This generates points or pixel coordinates for **zone 1**. But to make the rest of the circle, we need zone conversion via 8-way symmetry.

zone 1	zone 0	zone 2	zone 3	zone 4	zone 5	zone 6	zone 7
(0, 6)	(6, 0)	(0, 6)	(-6, 0)	(-6, 0)	(0, -6)	(0, -6)	(6, 0)
(1, 6)	(6, 1)	(-1, 6)	(-6, 1)	(-6, -1)	(-1, -6)	(1, -6)	(6, -1)
(2, 6)	(6, 2)	(-2, 6)	(-6, 2)	(-6, -2)	(-2, -6)	(2, -6)	(6, -2)
(3, 5)	(5, 3)	(-3, 5)	(-5, 3)	(-5, -3)	(-3, -5)	(3, -5)	(5, -3)
(4, 4)	(4, 4)	(-4, 4)	(-4, 4)	(-4, -4)	(-4, -4)	(4, -4)	(4, -4)

Notice that the zone 1 values are just copied in, it is because our algo worked in that zone and no conversion is necessary. The rest zones are converted via the "CirclePoints" function, which applies the 8-way symmetry.

Look how the above steps are same as the previous example (the radius was same BTW). But we are not done yet, since the center of this particular circle is not the origin.

The values we have in our hand now dictates a origin-centered circle. To make sure that its center is at (4, 3), we need to move all the generated points 4 units right and 3 units up, this moves the entire circle. Basically, **we need to add (4, 3)** to all the values in the above table.

Doing exactly that in the table below.

zone 1	zone 0	zone 2	zone 3	zone 4	zone 5	zone 6	zone 7
(4, 9)	(10, 3)	(4, 9)	(-2, 3)	(-2, 3)	(4, -3)	(4, -3)	(10, 3)
(5, 9)	(10, 4)	(3, 9)	(-2, 4)	(-2, 2)	(3, -3)	(5, -3)	(10, 2)
(6, 9)	(10, 5)	(2, 9)	(-2, 5)	(-2, 1)	(2, -3)	(6, -3)	(10, 1)
(7, 8)	(9, 6)	(1, 8)	(-1, 6)	(-1, 0)	(1, -2)	(7, -2)	(9, 0)
(8, 7)	(8, 7)	(0, 7)	(0, 7)	(0, -1)	(0, -1)	(8, -1)	(8, -1)

This is the answer.

Summary

For circle with radius r with **center (0, 0)**	For circle with radius r with **any center**
1. Run MPC algo to generate a set of points in zone 1. 2. Convert the zone 1 points to rest of the zones (0, 2, 3, 4, 5, 6, 7)	1. Run MPC algo to generate a set of points in zone 1. 2. Convert the zone 1 points to rest of the zones (0, 2, 3, 4, 5, 6, 7) 3. Add the center value (cx, cy) to all points obtained from **step 2**, and also all points obtained from **step 1**. Basically you need to do (+cx, +cy) to all 8 zone points.