Recipe

Recipe name: make_circle

Inputs:

- point0x, float: represents the x-coordinate of the first point
- point0y, float: represents the y-coordinate of the first point
- point1x, float: represents the x-coordinate of the second point
- point1y, float: represents the y-coordinate of the second point
- point2x, float: represents the x-coordinate of the third point
- point2y, float: represents the y-coordinate of the third point

Outputs:

- center_x, float: represents the x coordinate of the circle's center
- center_y, float: represents the y coordinate of the circle's center
- radius, float: represents the radius of the circle

Assumptions:

- The 3 given points are non-collinear
- The 3 given points are unique

Steps:

1. Find the slope of the line segment with endpoints on two chosen points
    a. slope_first <= slope(point0x, point0y, point1x, point1y)
        a. (point0x, point0y) and (point1x, point1y) each represent the x,y pair for the first and second points respectively, as stated in the Inputs
2. Find the midpoint of the same points chosen for the first slope
    a. midpoint_first_x, midpoint_first_y <= midpoint(point0x, point0y, point1x, point1y)
        a. midpoint() returns the x coordinate and the y coordinate of the midpoint
3. Find the line perpendicular to the first slope found that intersects the first midpoint found
    a. perp_first <= perp(slope_first)
    b. While perp only gives the perpendicular slope, given the slope and an endpoint (midpoint_first) the line can be found
4. Find the slope of the line segment with endpoints on two chosen points, with one point being different from the points chosen for steps 1-3
    a. slope_second <= slope(point0x, point0y, point2x, point2y)
5. Find the midpoint of the same points chosen for step 4
    a. midpoint_second_x, midpoint_second_y <= midpoint(point0x, point0y, point2x, point2y)
6. Find the line perpendicular to the second slope that intersects the second midpoint

    a. perp_second <= perp(slope_second)
7. Find the intersection between the two perpendicular lines to find the center of the circle
    a. center_x, center_y <= intersect(perp_first, midpoint_first_x, midpoint_first_y, perp_second, midpoint_second_x, midpoint_second_y)
8. Find the distance between the found center and one of the original points to find the radius of the circle
    a. radius <= distance(center_x, center_y, point0x, point0y)
9. Output the center point and the radius
    a. return center_x, center_y, radius

Discussion

I’m not sure what would be a better way for a computer to solve this given that I don’t know what resources a computer would have, but with certain assumptions I believe I know a more efficient method. Setting up a system of equations using the general form of a circle and inputting it into a matrix in order to use the Gauss-Jordan elimination method with reduced row echelon form should be faster in general, though I’m still uncertain as to the order of the operation.