Only lines of gcode with an X or Y need to be modified, all others can be ignored

Final format for a line should be G(x) X(x) Y(x) A(x) B(x) then possibly also Z(x) C(x) F(x)

Starting Points : $X_0 = 0$, $Y_0 = 0$, $A_0 = 0$, $B_0 = 0$ (The subscripts are just for identification purposes and will not be included in the final gcode)

Every X_n , Y_n point will have a corresponding A_n , B_n point, this means that every X_n , Y_n to X_{n+1} , Y_{n+1} line segment will also have a corresponding A_n , B_n to A_{n+1} , B_{n+1} line segment.

Any code starting points are just ideas and can be disregarded if there is a better way.

Current max straight line distance allowed between any X_n , Y_n point and its corresponding A_n , B_n point is 100mm. Anything less than this will be considered “close” and anything greater than this will be called “far”. The same goes for the distance from a current A_n , B_n point to a future X_{n+m} , Y_{n+m} point. Greater than 100mm will be “far” and less than 100mm will be “close”.

There are four main situations of moves for X and Y which will need to be accounted for:

1. A_n , B_n to X_{n+1} , Y_{n+1} > 100mm & X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} > 100mm
2. A_n , B_n to X_{n+1} , Y_{n+1} > 100mm & X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} < 100mm
3. A_n , B_n to X_{n+1} , Y_{n+1} < 100mm & X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} < 100mm & X_m , Y_m to X_{m+1} , Y_{m+1} < 100mm
4. A_n , B_n to X_{n+1} , Y_{n+1} < 100mm & X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} < 100mm & X_m , Y_m to X_{m+1} , Y_{m+1} > 100mm

For #1: “Cutting the corner” - Bisect the angle between the lines from X_n , Y_n to X_{n+1} , Y_{n+1} and X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} . A_{n+1} , B_{n+1} will then be placed 100mm from X_{n+1} , Y_{n+1} along the bisecting line.

Code starting point:

- from sympy.geometry import Point, Circle, Triangle
- p1, p2, p3 = Point(X_n , Y_n), Point(X_{n+1} , Y_{n+1}), Point (X_{n+2} , Y_{n+2})
- t = Triangle(p1, p2, p3)
- r = 100
- c = Circle(p2, r)
- p4 = c.intersection(t.bisectors()[p2])

  p4 = A_{n+1} , B_{n+1}

For #2: Same as #1 but instead of A_{n+1} , B_{n+1} being 100mm from X_{n+1} , Y_{n+1} along the bisecting line it will need to be half the distance from X_{n+1} , Y_{n+1} to X_{n+2} , Y_{n+2} along the bisecting line.

Code starting point:

- from sympy.geometry import Point, Circle, Triangle, Line
- p1, p2, p3 = Point(X_n , Y_n), Point(X_{n+1} , Y_{n+1}), Point (X_{n+2} , Y_{n+2})
- t = Triangle(p1, p2, p3)
- l = Segment(p2, p3)
- r = (l.length)/2
- c = Circle(p2, r)
- p4 = c.intersection(t.bisectors()[p2])

  p4 = A_{n+1} , B_{n+1}

For #3: When X_{n+1} , Y_{n+1} is close to A_n , B_n then the distance from A_n , B_n to all the following X , Y points will need to be calculated until one is found to be more than 100mm away. Call this point X_{m+1} , Y_{m+1} with the last point still within 100mm now called X_m , Y_m . The total distance traveled by the X , Y system will need to calculated and added up between points X_n , Y_n and X_m , Y_m . Call this distance ΔXY for now. If the distance from X_m , Y_m to X_{m+1} , Y_{m+1} is also less than 100mm then the midpoint between X_m , Y_m and X_{m-1} , Y_{m-1} will need to be calculated. Call this point A_m , B_m . The distance, ΔAB , of the line from A_n , B_n to A_m , B_m will then need to be calculated. Next ΔXY will need to be divided by ΔAB to get the distance scaling factor. The length of the line segment from X_n , Y_n to X_{n+1} , Y_{n+1} will then be divided by this scaling factor to determine the length of line from A_n , B_n to A_{n+1} , B_{n+1} with A_{n+1} , B_{n+1} being placed that distance from A_n , B_n along the line to A_m , B_m which was determined earlier. At this point A_{n+1} , B_{n+1} and X_{n+1} , Y_{n+1} will now become the new A_n , B_n and X_n , Y_n and this whole process will be run again from the new starting points until reaching a point where X_{m+1} , Y_{m+1} is more than 100mm from X_m , Y_m .

Code starting point:

- Loop checking distance from A_n , B_n to X_{n+1} , Y_{n+1} is less than 100mm. This loop can also sum the X , Y distance traveled as it checks each point in order. Should probably also save the distance from X_n , Y_n to X_{n+1} , Y_{n+1} as its own variable since that number will be needed later.
- Once X_{m+1} , Y_{m+1} is found, A_m , B_m can be determined by finding the midpoint from X_m , Y_m and X_{m-1} , Y_{m-1} using:

    - from sympy.geometry import Point
    - p1, p2 = Point(X_{m-1} , Y_{m-1}), Point(X_m , Y_m)

- p1.midpoint(p2)

- Divide the total X , Y by the segment from A_n , B_n to A_m , B_m then use that product to divide the segment from X_n , Y_n to X_{n+1} , Y_{n+1}
- That length can then be used as the radius for a circle at point A_n , B_n and then intersected with the line from A_n , B_n to A_m , B_m to find A_{n+1} , B_{n+1}

For #4: Everything about this one will be the same as #3 except that A_m , B_m will be placed at the “cutting the corner” location between X_m , Y_m and X_{m+1} , Y_{m+1} . The loop will then continue until A_{n+1} , B_{n+1} coincides with point A_m , B_m . This should result in either a #1 or #2 type of movement case.

All new A_{n+1} , B_{n+1} points will need to checked to make sure they are still within 100mm of their corresponding X_{n+1} , Y_{n+1} point. If they are more than 100mm away then they will need to me moved along a straight line to 100mm away from X_{n+1} , Y_{n+1} . This would probably involve using the .intersection function of a 100mm radius circle centered at X_{n+1} , Y_{n+1} intersecting a line from X_{n+1} , Y_{n+1} to the current A_{n+1} , B_{n+1} to give the new A_{n+1} , B_{n+1} point.