ME 290-R, Prof. Sara McMains
Homework # 3
Problem 1, one per pair, due at start of class, Monday Mar. 11; the rest is due Fri 5pm Mar. 22, and problem 2 is in pairs but problem 3 should be done individually (see policy on last homework).

1) Whose ITO surface generator code is faster, yours or your partners? Why? Does it depend on the input? Do both codes generate the same surface?

2) Find the Big-Oh running time of your original code. The input size has multiple dimensions, so you should use multiple variables. Call the Z-map surface input size $m \times n$, and the tool diameter/spacing ratio p, and assume $p > 1$. You can keep the highest order term for each variable, and for each combination of variables, so the answer might look like $O(n^2 + m^3 + mn + pm^5 + p\log(p))$. Compare to your partner. Does the Big-Oh time complexity explain the different running times, or is it a matter of the constants that Big-Oh drops, or something else Big-Oh ignores? Turn in an even faster version than either original code that still generates the correct result (email me the zip file) – it only needs to be faster in terms of wall clock time, but may or may not have better Big-Oh time. Bonus points if you are also faster than the others' revised code.

3) Write a function to generate the xy-parallel_oneway-finishing toolpath from the z-map of a design surface, extending the code you wrote to generate the ITO z-map for a ball end mill in the last homework assignment. Please make your function interface look like:
"function [X, Y, Z] = generate_toolpath(*dsurf_fname*, *spacing*, *diameter*, *omega*, *lamda_max*, *pi*),"
where *dsurf_fname* is the file name of the z-map matrix for the input design surface, *spacing* is the spacing between the z-map points in both the x and y directions, *diameter* is the diameter of the ball end mill, *omega* is the maximum cutting width (pick-feed distance in 3D), *lamda_max* is the maximum step length, *pi* is the machining tolerance, and $X/Y/Z$ are output matrices of the $x/y/z$ coordinates (respectively) of the CL-points your program generates for the toolpath.

Please base your implementation on the pseudo-code given in the C-space paper for TPG-algorithm (xy-parallel_oneway-finishing), but skip the pencil cutting in step (1), and fix the typo in step (3.6).

Hint 1: For step (3.1), most of the positions (x, y) will not be on your original grid! Use linear interpolation between given points to find the z-heights for new points. You will need to first interpolate between rows to add entire new rows of points between existing rows, and then interpolate within the new row.

Hint 2: For step (3.2), compute R_f for all the original grid points first, and then use linear interpolation (similar to Hint 1) to estimate the R_f for a point that is not on the original grid. To compute R_f for the original grid point, fit a circle through the point you are considering and its immediate left and right neighbors (handle the first and last points in the row as special cases – use its two neighbors on one side instead). Be careful when the three points are collinear (R_f will be infinity in this case). See next page for a sample derivation.

Hint 3: For step (3.4) and (3.5), you can compute the inclination angles in a way similar to how R_f is computed in Hint 2. First compute the inclination angles for all the original grid points by using circle fitting and taking the angle of the circle tangent at the grid point as its inclination angle, and then interpolate them to estimate the angle at a general position.

Hint 4: You need to perform linear interpolation and circle fitting multiple times in your program, so please implement them as two independent functions and test them separately on simple input before you use them in the rest of your code. What other subroutine functions can you identify so that you don't have long, difficult-to-debug functions?

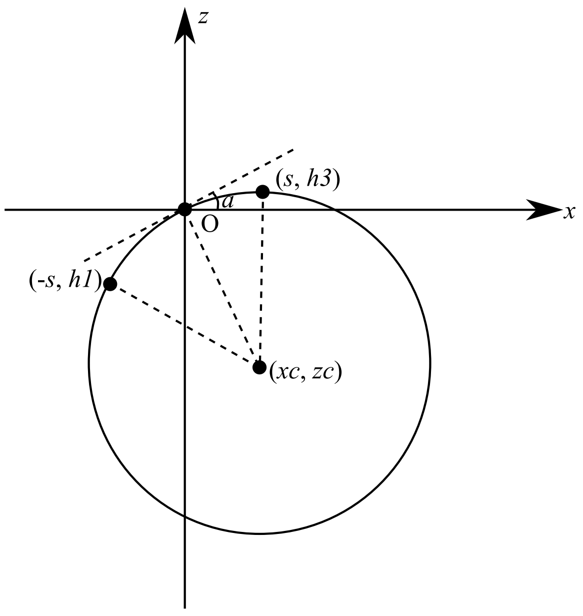

Figure 1: Fit a circle through three consecutive grid points

Circle Fitting

Suppose the z heights at the three consecutive grid points are z_1, z_2, and z_3 respectively, and the spacing between grid points is s. Move the origin to the second point (Figure 1), and define $h_1 = z_1 - z_2$ and $h_3 = z_3 - z_2$. The coordinates of the other two points become $(-s,\, h_1)$ and $(s,\, h_3)$. The center of the circle going through these three points must satisfy:

$$\begin{cases} x^2 + z^2 = (x+s)^2 + (z-h_1)^2 \\ x^2 + z^2 = (x-s)^2 + (z-h_3)^2. \end{cases} \tag{1}$$

The above two equations can be converted to a linear system as follows:

$$\begin{cases} 2xs - 2zh_1 + (s^2 + h_1^2) = 0 \\ -2xs - 2zh_3 + (s^2 + h_3^2) = 0. \end{cases} \tag{2}$$

Solve it to get the coordinates of the circle center $(x_c,\, z_c)$ as below (when $h_1 + h_3 \neq 0$):

$$\begin{cases} x_c = \frac{(s^2 - h_1 h_3)(h_1 - h_3)}{2s(h_1 + h_3)} \\ z_c = \frac{2s^2 + h_1^2 + h_3^2}{2(h_1 + h_3)}. \end{cases} \tag{3}$$

If $h_1 + h_3 = 0$, the three points are collinear. Handle it as a special case.

The radius of curvature r is:

$$r = \sqrt{x_c^2 + z_c^2}. \tag{4}$$

Left to reader: computing inclination angle a.