**Watershed Segmentation  
ECE 847 Assignment #4**

**Purpose:** In this assignment you will implement the simplified Vincent-Soille marker-based watershed segmentation algorithm.

**Before you start:** Watch the Lecture 4 series.

**Background info:** As mentioned in the lecture, segmentation is a classic problem in machine and computer vision. A special case of segmentation is when the goal is to isolate homogeneous-colored foreground objects from the background. One of the more popular algorithms for this special version of the problem is the watershed algorithm, more specifically the Vincent-Soille watershed algorithm. Most descriptions of Vincent-Soille rely on dams to prevent water from flowing into adjacent regions, but as we saw in the lecture these dams are unnecessary. Therefore, we recommend the simplified (or dam-less) Vincent-Soille algorithm. In addition, as explained in the lecture, markers are necessary to avoid oversegmentation that occurs due to noise in the image; as a result the marker-based version of the algorithm is the focus of this assignment.

**Instructions:**

1. In this assignment you will write an application (that is, you will modify the code in *homework*) to perform simplified Vincent-Soille marker-based watershed segmentation of an image. The code should perform the following steps in order when it is run:
   1. Reads 2 command-line parameters, which we will call *filename* and *threshold*.
   2. The very first thing the program should do is print a suggested threshold value for both images that we will be using (holes.pgm and cells\_small.pgm). Most likely the grader will first run your code with 0 or 1 parameter, view the suggested threshold values, then run your program with 2 parameters, using the suggested value appropriate for the image being used. Therefore these suggested values should be printed every time, to make it easy for the grader.
   3. If fewer than 2 parameters are provided, then the program should exit. Otherwise, it should proceed.
   4. Detect markers, as defined by the threshold. Note that how you interpret the threshold value is completely up to you. All that matters is that the suggested values yield good results for the two images provided. Although you are free to detect markers any way you want, the recommended steps are the ones given in the lecture: threshold, then chamfer, then (non-marker-based) watershed, then edge detection, then logical “or”. For edge detection, simply find pixels whose value is different from one or more neighbors – do not do anything fancy like Canny.
   5. Perform the marker-based watershed algorithm. The basic algorithm involves two preliminary steps: (1) Compute the magnitude of the image gradient, quantized; (2) Construct a data structure allowing fast access to all the pixels with a certain value. Then the algorithm iterates through the image gradient values, applying three steps in sequence:
      1. Grow existing catchment basins by one pixel
      2. Apply breadth-first search to flood the pixels, assigning pixels to the nearest existing catchment basin, if there is one. Be sure to use the FIFO queue of *std::queue* or *std::deque* (unless you want to write your own) to ensure breadth-first search.
      3. Create new catchment basins for pixels that are not adjacent to an existing catchment basin. Use floodfill to create these new basins.
2. Your output should look like this:
   1. One figure window should show the original image. Additional figures should show the thresholded image, chamfer distance, non-marker-based watershed output, edges, and markers; as well as the gradient magnitude image and final marker-based watershed output, as displayed in Slide 99 of the lecture. Be sure to set the title of each figure to an appropriate human-readable string that indicates what is being displayed.
3. The grader will test your code on the images holes.pgm and cells\_small.pgm, as well as other similar images.
4. For this assignment, you may ***not*** use any Blepo functionality in Watershed.cpp. If you would like to use other functionality in Blepo, you may do so, but you must notify the grader in the body of your submission email, and you will be deducted points for doing so. The number of points deducted are as follows: threshold (5), chamfer (10), edge detection (5), gradient magnitude (10), and floodfill (10). For any other functions you wish to use, please contact the grader and/or instructor. Hopefully you will be able to use a fair amount of functionality from previous assignments.
5. Submit your code to the grader as described in the first assignment.