**Stereo Matching  
ECE 847 Assignment #5**

**Purpose:** In this assignment you will implement efficient sliding window (block-based) matching of two rectified stereo images.

**Before you start:** Watch the Lecture 5 series.

**Background info:** As mentioned in the lecture, stereo matching is the process of determining corresponding pixels between the two images. Once correspondence has been determined, triangulation can be used to determine the depth of each matched pixel. The simplest stereo algorithm is to center a window at each pixel in one image, then slide that window across the other image, comparing the two windows by summing over the dissimilarities of the pixels in the windows, and storing the location that yields the minimum sum of the dissimilarities. To avoid the computational expense of the 5 nested for loops, the exact same result can be achieved by precomputing a 3D data structure of dissimilarities, followed by a series of separable convolutions with a kernel consisting of all 1s, which reduces the computation considerably.

**Instructions:**

1. In this assignment you will write an application (that is, you will modify the code in *homework*) to perform basic block-matching between a rectified stereo pair of images. The code should perform the following steps in order when it is run:
   1. Reads 3 command-line parameters, which we will call *left-filename*, *right-filename*, and *max-disparity*.
   2. Load the two (grayscale or color) images. If these images are color, then convert to grayscale, so that only grayscale is used for matching. This will make your code easier to debug and more computationally efficient, without sacrificing anything in the quality of the results. Nevertheless, you should read the color values so that you can output them in the MeshLab PLY file in step d) below.
   3. Perform block-based matching of the two images. For efficiency, your code should precompute the 3D array of dissimilarities, followed by a series of separable convolutions (one pair of convolutions per disparity). Use the SAD dissimilarity measure. Implement the left-to-right consistency check, retaining a value in the left disparity map only if the corresponding point in the right disparity map agrees in its disparity. The resulting disparity map should be valid only at the pixels that pass the consistency check; set other pixels to zero. *(Note: For simplicity, do not worry about pixels along the left border of the left image; it is ok the produce erroneous values there.)*
   4. Use triangulation to determine the depth of each matched pixel. Recall that the formula is depth = k / disparity, where k is the focal length times the baseline. Since we do not know the value of k, you will have to manually try a few values until you get a result that looks plausible. Output a PLY file that can be read by MeshLab (details below). For this assignment, you should output six columns (x y z r g b) for each matched pixel, ignoring the normal components.
      1. Use orthographic projection (rather than perspective) to get your x,y,z coordinates, because it is simpler to implement and will yield a more aesthetically pleasing point cloud, even though it is less accurate mathematically. In other words, if the coordinates of the matched pixel in the image are (x,y), and the depth is z (from the triangulation formula above), then the 3D coordinates are (x,y,z).
      2. Although there is no reason for your code to use the RGB values for matching, you should definitely use RGB color to make your PLY file more aesthetically pleasing.
2. Your output should look like this:
   1. Two figure windows should show the original images. An additional figure should show the left disparity map *without* the left-right consistency check, and a final figure should show the left disparity map *with* the left-right consistency check. Feel free to display any additional output you desire, but be sure to set the title of each figure to an appropriate human-readable string that indicates what is being displayed.
   2. A PLY file that is readable by MeshLab.
3. The grader will test your code on the images tsukuba\_left.ppm and tsukuba\_right.ppm, as well as other similar images. For fun, try your code on lamp\_left.pgm and lamp\_right.pgm, but do not expect it to perform well at all, since these images lack texture.
4. For this assignment, you may ***not*** use any Blepo functionality in RealTimeStereo.cpp. Otherwise, you may use any other functionality in Blepo that you find useful.
5. Submit your code to the grader as described in the first assignment.

**MeshLab instructions**

[MeshLab](http://meshlab.sourceforge.net/) is free and open-source software to enable easy viewing of 3D data. MeshLab reads PLY files, which are ASCII files with a simple format:  In the header you specify the number of vertices, along with the properties stored for each vertex (e.g., x y z nx ny nz r g b); then after the header there is one line per vertex.  An [example PLY file](http://www.ces.clemson.edu/~stb/ece847/fall2012/kermit.ply) created from a set of [Kermit](http://www.ces.clemson.edu/~stb/ece847/fall2012/kermit.jpg) images will help you to understand and replicate the format.  Notice that the example has 3 additional columns for the normal vectors that you will not use. (*Note: When you click the "Import Mesh" button (the one that looks like File.Open), MeshLab* ***merges*** *the file that you select with the current mesh -- it* ***does not replace*** *the current mesh with the new one. To overcome this problem, either be sure to click the "Reload" button (next to the "Import Mesh" button) or close MeshLab altogether and start it up again.*)