CPSC 535
Assignment 5a: Stereo Vision

The goal of this assignment is to write octave code to implement a simple algorithm to compute stereo disparity.

1 Stereo Disparities

Computation of stereo disparities requires matching of pixels in a left-eye image to pixels pixels in a right-eye image. If the cameras are aligned and their optical axes parallel, then a left-image pixel can only match a right-image pixel in the same row. If the optical axes converge then it becomes necessary to consider the where the epi-polar lines are in each image. To keep things simple (usually a good idea), we will assume that we have cameras that are aligned with parallel optical axes. Thus we only need to look for matches between images in the same row.

The approach we will use is to find, for each left-image pixel, the disparity that gives the best match to a right-image pixel. Looking at a single pixel will not work well since we have the potential to match spurious noise in the image. Instead, we match pixels by considering the quality of match in a region around a pixel. This effectively smoothes the matching process making it less susceptible to noise.

If one were to implement the algorithm by looping through the pixels and seeking a best match at each pixel, the result would be an inefficient algorithm because of the many redundant calculations. The algorithm in Figure 1 re-arranges the nesting of loops to remove the redundancies. $G(\sigma)$ is a Gaussian smoothing kernel. M is a measure of similarity between L and a shifted R. Small values of M indicate a better match. For each disparity M is computed, then, wherever M is smaller than M_{min}, there is a better match, so the algorithm records the disparity for that pixel and updates M_{min}. When the algorithm is complete, D will contain the disparities that produced the best matches, i.e., the lowest M.

Write an octave function that implements the algorithm in Figure 1. Name it **stereo** and call it using five parameters: L, R, σ, s_{min}, and s_{max}. The function should return a disparity map. To do the Gaussian smoothing you may use the Deriche filter provided. There are two sources of test images. The easiest is to use the function **layercake**, provided, to generate random-dot stereograms with different disparity layers. I recommend you start here because that will give the best results. Once you have your algorithm working, try it out with the other images I found while browsing the internet. The images contain left- and right-eye views in a single image file so you will have to separate the two images. These images are more complex than the layercake images so the results will not be perfect.

Write a wrapper script that creates or reads a stereo pair of images, calls **stereo**, and displays the result.

2 Options

If you have the time, you may want to experiment with the following ideas, but it is not required.

Input: L, R, the left- and right-eye images, S the set of possible disparities, σ, width of Gaussian smoothing kernel.

Output: D, the stereo disparity image.

for s in S do
  $M = G(\sigma) \otimes |L - \text{shift}(R, s)|$
  if *first iteration*
    $D = s$
    $M_{min} = M$
  else
    for all pixels i, j do

$$d_{ij} = \begin{cases} s & m_{ij} < m_{min_{ij}} \\ d_{ij} & \text{otherwise} \end{cases}$$

    $M_{min} = \min(M, M_{min})$

Figure 1: Algorithm for computing stereo disparities

When matching regions of the image, you normally try to avoid matching noise. Also, broad uniform regions are difficult to match because the match does not change with the disparity. This leads the the concept of *whitening*. The idea is to pre-process the images with a filter that will eliminate low-frequencies while still suppressing high-frequency noise. Recall that the LOG edge detection filter can do this. Try applying an LOG filter to the filter input to see if it improves the output.

Consider what happens when there are occluded surfaces in a scene (regions that cannot be seen because something is in the way). It is possible in stereo that some part of a scene is visible in one eye but cannot be seen by the other. The algorithm above finds a match, even though it should not. One way to deal with this is to match L to R, and then R to L. You can then check to see that the match from left to right is the same as the match from right to left. If they do not match then the algorithm has failed because of an occlusion or some other reason and the disparity computed for that point should be ignored. Try using the reverse match to verify that the forward match is correct and produce a binary image that has ones where the match is valid and zero elsewhere.

Hand In

1. Your code to compute the stereo grams.
2. Plots of depth and/or disparity (gsplot from octave works well).
3. A description and explanation of any salient observations you make while working on this assignment.

You will be graded on the quality of your code, your plots, and your written observations.