CS2302 - Data Structures

Fall 2016

Lab # 8

Dynamic Programming

Deadline: Thursday, November 30 (no penalty if it's submitted by report deadline)

In stereo computer vision we extract depth information using two cameras displaced horizontally from one another. The cameras are used to obtain two different views of a scene, in a manner similar to human binocular vision. By comparing these two images, the relative depth information can be obtained in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location.

A stereo pair. Notice that disparity is larger the closer an object is to the camera.

Disparity map computed from the stereo pair. Lower intensities indicate lower disparities and thus larger depths.

In this lab you will implement an algorithm to compute this disparity map using dynamic programming. Your method should receive a stereo pair consisting of a left image I_L and a right image I_R. These images are stored as *m-by-n-by-3* arrays, where m is the number of rows, n is the number of columns, and the last dimension encodes the red, green and blue color components of the corresponding pixel. The method should output a disparity map D of size *m-by-n* where $D[i, j]$ means that pixel $I_L[i, j]$ is matched to pixel

$I_R[i, j - D[i, j]]$. Notice that pixels in row i of I_L can only match to pixels in row i of I_R, which significantly simplifies the problem and allows us to find the disparity of each row independently.

The fist few lines of your method may be similar to these:

```
public static int[][] getImageDisparity(int [][][] IL,int [][][] IR, int occlusionCost){
      int [][] disparity = new int[IL.length][IL[0].length];
      for(int i=0;i<IL.length;i++)
        disparity[i] = getRowDisparity(IL[i],IR[i], int occlusionCost);
      ...
```

To compute the disparity of each row, you need to implement a dynamic programming algorithm that is very similar to the edit distance algorithm explained in class. Recall that in edit distance we try to match as many characters in string S1 to characters in string S2 as possible, assigning no cost for matches and a unit cost for insertions, deletions, and replacements. In stereo matching, we want to match as many pixels in the left image to pixels in the right image as possible, again assigning a cost to unmatched pixels (this is called the occlusion cost). Unlike character matches, pixel matches aren't perfect, so we will assign a cost to every match that is equal to the difference in color of the two pixels.

We will use a matrix d as we did in edit distance. Recall that the first row and column were filled with the costs of either deleting or inserting all the characters in the substring. In the image case, they should be filled with the costs of not matching any pixels. Thus d[0][i] = d[i][0]= i*oclussionCost. The rest of the matrix is filled as follows:

$$d[i+1, j+1] = min \begin{cases} d[i][j+1] + oclussionCost \\ d[i+1][j] + oclussionCost \\ d[i][j] + \sqrt{(\sum_{k=0}^{2}(L[r][i][k] - R[r][j][k])^2} \end{cases}$$

Once d is computed, you need to follow the path backward from the bottom-right to the top-left to determine the choices that were made and the disparities to store in the array (for a hint, see how this is done in the edit distance handout).

Experiment using stereo pairs from vision.middlebury.edu/stereo/data/scenes2014/. Use the "perf" versions of the images. Code to perform basic image processing is provided in the class webpage.

Given the short time available, we will not require a demo for this lab, thus it is very important that your report accurately describes your work.