Problem Set 4 CMSC 426
Due: Thursday, April 17

1) Overview: You will implement a stereo algorithm that uses dynamic programming. This algorithm enforces the ordering constraint (or else you couldn't use dynamic programming), and matches individual pixels. Every pixel in one image can either match one pixel in another image, or be marked as occluded. The penalty for matching two pixels is the square of the difference in their intensity. The penalty for being occluded is a fixed value, *occlusion_penalty*. We will start by testing on random dot stereograms, where each pixel is zero or one, then run on real images. You should consult the paper: ``A Maximum Likelihood Stereo Algorithm'', by Cox, Hingorani, Rao, and Maggs, from the journal Computer Vision and Image Understanding, 63, 3, pp. 542-567. This is on reserve in the CS library, 3rd floor of A.V. Williams. Your assignment closely follows the method in that paper, and it contains significant hints. ***Please note that for consistency with that paper, all thresholds below are given assuming that image intensities fall in the range 0 to 1. To use these thresholds, you should scale the images to this range when you read them in. That is, do something like: RDSLa = imread(`RDSL.jpg'); RDSL = double(RDSLa)/255.*** **80 points total.**

a) Paper and pencil exercise: Using the cost function above, with *occlusion_penalty*= .01, compute all minimum cost matches for the 1D images, v1 = [1 0 1 1]; v2 = [1 1 0 1]. Write your answers as a disparity map for v1. That is, a map of [0 x -1 -1] means the disparity for the first point in v1 is 0 (v1(1) = 1 matches v2(1) = 1, the second point is occluded and unmatched, the third point has a disparity of -1, (v1(3) =1 matches v2(2) = 1), and the fourth point has a disparity of –1 (v1(4) =1 matches v2(3) = 0). Of course, this example is not necessarily a minimum cost matching. There may be one or more matches with the same minimum cost. (Note that for the simple camera geometry discussed in class, disparity must be positive for points in front of the camera. This is not true for all camera geometries. For this problem set, we will ignore this subtlety and compute everything allowing positive or negative disparity). **15 points**

b) Warm-up programming exercise: write a function to compute the cost of extending a matching by one value. That is, we are given two 1D images, v1 and v2. We want to compute the best cost of a matching between the vectors that includes the first m1 points in v1, and the first m2 points in v2. Call this cost c(m1,m2). We are given c(m1-1,m2), c(m1, m2-1), c(m1-1,m2-1). c(m1-1,m2), for example, gives the cost of the best matching that includes all the first m1-1 points in v1, and the first m2 points in v2. There are three ways we can extend these matches, by matching v1(m1) to v2(m2) or by allowing an occlusion in either image. Write a function c = extend_match(x,y,c1,c2,c3,oc).
c is c(m1,m2), the best resulting cost. x=v1(m1) and y=v2(m2). Note that you don't need to know what's in the rest of the images. c1 = c(m1-1,m2), c2 = c(m1,m2-2), c3=(m1-1,m2-1) and oc is the occlusion_cost. Test this with extend_match(1,1,3.7, 3.9, 3.5, .01), and extend_match(1,0,3.7, 3.9, 3.5, .01). Turn in code and result of these tests. **15 points**

c) Implement the dynamic programming stereo algorithm in 1D. You should write a function of the form d = stereo_1d(v1, v2,oc), where v1 and v2 are vectors to be matched, and d contains the disparity for every pixel in v1. Test this for, oc = .01, v1 = [1 0 1 1 0 1 1 0 0 1 1 0 1 1 1];
v2 = [1 0 1 0 1 0 1 1 0 0 1 1 1 1 1]; Turn in your code and the result of this test. **30 points**

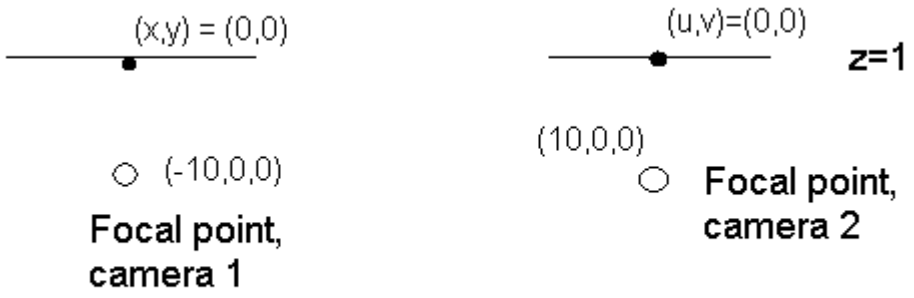
d) Extend this algorithm to 2D. We assume the images are rectified, so that the epipolar lines are horizontal lines. Lines in each image with the same y value match. You just need to run stereo_1d on each corresponding line in the two images. This will have the form: D = stereo_2d(V1, V2, oc). Test your algorithm by running on the images on the class web page. Use the images RDSL (image 2 on the web page) and RDSR (image 1), with oc= .01. Display the resulting disparity map as an image, with imagesc(D) or a similar function. To do this, you must be sure that occlusions in the disparity map are encoded as numbers that will display in a way that is distinct from any valid disparity. For example, if you expect disparities to range from –20 to 20, you might set occlusions to be -40. You may need to play with this a little, for example, multiplying D by some constant, so the individual disparities display clearly. Turn in your code and an image showing the disparity map. **10 points**

e) To speed up your code, modify it so that we never consider matching pixels with a disparity of more than 15, or less than -15. Now run this faster code on the images Tsukuba_left (T3bw, or image 5, on the web page), and Tsukuba_right (T4bw, or image 6), on the class web page. Turn in the new code, and an image showing the disparity map you get. **10 points**

f) **CHALLENGE PROBLEM:** One possible problem with this algorithm is that it treats all occlusions as equally bad. But usually regions, not pixels, of an image are occluded. This is a hard problem to deal with, but here's a simple strategy for handling it in images like random dot stereograms in which all surfaces are parallel to the image plane. Assign a cost to an occluded pixel of oc1. But then, if the next pixel is occluded, assign a smaller cost, oc2. So for example, let oc1 = .01, and oc2 = .001. The cost of one occluded pixel is .01. The cost of two consecutive occluded pixels is .011. Three consecutive occluded pixels cost .012. etc…. Modify your code from parts c and d to use dynamic programming to find the best matching with this cost function. To make things reasonably efficient, you may want to only impose this cost on strings of 15 occluded pixels. So, the cost of occluding 15 consecutive pixels is oc1 + 14*oc2, but the cost of occluding 16 consecutive pixels is 2*oc1 + 14*oc2. Run the old and new code on the images RDS2L (image 4 on the web page) and RDS2R (image 3 on the web page) from the web page. Turn in code and pictures of the disparities. **10 points**

g) For a small amount of additional extra credit, see if you can find values of oc1 and oc2 that will get this to work well on the Tsukuba images. By this I mean try to find an oc1 and oc2 where oc2 isn't trivial (oc1 or almost oc1). I wasn't able to find them myself, but I ran out of patience. **5 points.**

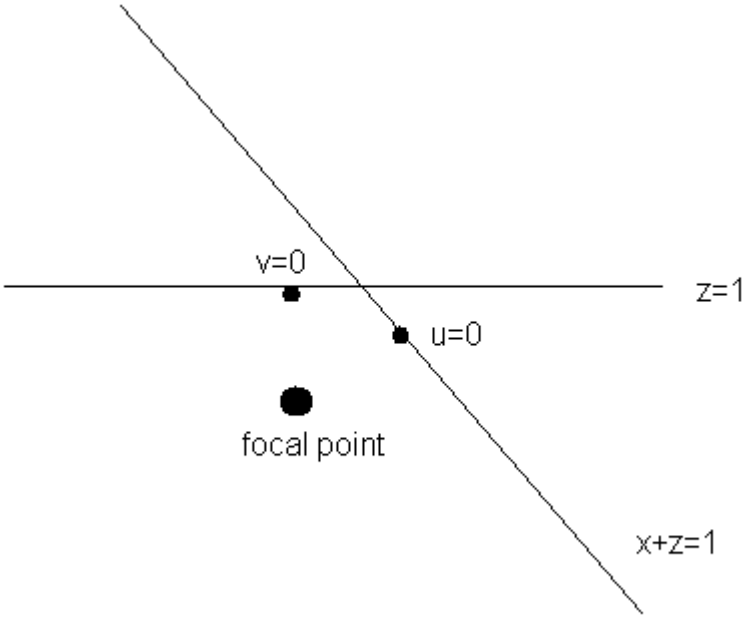
2) Pencil and paper. Consider two cameras whose image planes are the z=1 plane, and whose focal points are at (-10, 0, 0) and (10, 0, 0). We'll call a point in the first camera (x,y), and a point in the second camera (u,v). Points in each camera are relative to the camera center. So, for example if (x,y) = (0,0), this is really the point (-10,0,1) in world coordinates, while if (u,v) = (0,0) this is the point (10,0,1).



a) Suppose the point (x,y) = (1,0) is matched, with disparity of 2, to the point (u,v) = (-1,0) in the second camera. What is the 3D location of this point? **5 points**
b) Suppose the point (x,y) = (7,7) is matched with disparity of 4 to the point (u,v) = (3,7). What is the 3D location of this point? **5 points**
c) **CHALLENGE PROBLEM:** Consider points that lie on the line x+z=0, y = 0. Use the same stereo set up as before. Write an analytic expression giving the disparity of a point on this line after it projects onto the two images, as a function of its position in the left image. So your expression should only involve the variables u and d (for disparity). Your expression only needs to be valid for points on the line that are in front of the cameras, ie., with z > 1. **10 points.**

3) Pencil and Paper: In this problem we will consider a 2D world with x-z coordinates. That is, imagine the problem takes place in a horizontal slice through the world in which y is constant. Consider a camera with focal point at the origin, and an image plane (line, really since we're in a 2D world) along the line x+z = 1. We have an image, and we want to rectify it so that we get an equivalent image on the line z=1. We will parameterize the first image using the variable u, and the second image using v. In our original image, the camera center was at the point on the line x+z=1 nearest the focal point, ie., (½, ½). So we call this point u=0 in the original image. If we move a distance of 1 along the line x+z = 1, the u value increases by 1. So, for example, the point with world coordinates (x,z) = (0,1) has a u value in the image of sqrt(1/2). Suppose the first image appeared to

have intensities that increased linearly. That is, the first image had intensities described by the equation I(u) = u (don't worry about negative intensities).



a) In the rectified image, what is the intensity at v = 0 (that is, in the point in the image with world coordinates (x,z) = (0,1))? **5 points**

b) After rectifying the image, write an equation describing the image intensity as a function of v. **5 points**

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.