Video: Stereo Correspondence Problem

[00:01] [slide 1] In the last video, I described the three main steps of the stereo process. The first is to extract features from the left and right images whose stereo disparity will be measured. The second is to determine which features in the left image match up with which features in the right, and to measure the disparity in position of corresponding features in the two images. The final step is to use the stereo disparity of the features to compute their depth in the scene. We said that by far, the most challenging step of this process is the matching step that we refer to as the stereo correspondence problem. In this video, you'll learn why this problem is hard, and explore a simple strategy for solving it.

[00:48] Why is the problem hard? In the stereo images you're looking at here, it's pretty easy for you to identify the same features in the left and right images, everything is pretty distinctive. [slide 2] But have a look at these stereo images - here, the visual textures are fairly homogeneous, and it's hard, for example, to take a little patch of leaves, like the one in the upper left corner there, that I took out of this left image at the top, and figure out where that same patch appears in the right image - it's like doing a hard jigsaw puzzle. Yet we have no problem fusing together stereo images like this and perceiving a vivid three-dimensional scene.

[01:34] [slide 3] The challenge of the stereo problem is emphasized even more in a type of stereo image called a random-dot stereogram, which was first conceived by Bela Julesz in the early 1970s. How is it constructed? The left image is just a pattern of randomly positioned black and white dots. The right image is sort-of a copy of the left, except that parts of the pattern are shifted to the left or right in the right image. For example, we might copy most of the left image as it is, over to the right, but take a square region in the middle, like the area outlined in red, and copy this region over to the right image at a different horizontal position, in this case a position that's further to the right. If we then view the stereogram in a way that forces our left eye to view the left image and our right eye to view the right image, our visual system is able to sense that there's a region of the left image that's shifted in the right, measure that shift, and perceive a square surface in three dimensions that's floating above the background. On the far right, I display the left and right images superimposed in the same region, as I did before. The region that's shifted between the two images appears to move back and forth in this demonstration, and the motion itself gives you a sense of a three-dimensional surface in front of a background, but the percept of a surface at a different depth would be more potent if you could view it stereoscopically.

[03:21] You can create random-dot stereograms portraying much more elaborate surfaces than our simple square here. These kinds of stereograms are used extensively to study human stereo vision. Why is that, and what do they tell us? First, they tell us that the stereo system can function independently of other visual cues that we use to infer three-dimensional structure. When we view the static left and right images stereoscopically, there is no motion, no shading, no surfaces occluding other surfaces, no objects that we can recognize in the image. There's only stereo disparity, so our stereo system is operating on its own here. Second, they tell us

that we can match very simple features between the left and right images - there's only dots here, there's no significant edges or objects, for example. And finally, they highlight the ambiguity of the matching process. In principle, a black or white dot in the left image could match with any dot in the right, yet somehow, we figure out the correct correspondence between dots in the left and dots in the right. How do we do that?

[04:40] [slide 4] We need to be able to identify constraints on the matching process that allow us to narrow down the possible ways that features could be matched up, and build these constraints into an algorithm to solve the correspondence problem. What kind of constraints could we consider here? Here are four that are commonly used in stereo correspondence algorithms - uniqueness, similarity, continuity, and a constraint that goes by the peculiar name of the epipolar constraint. What is the meaning of these constraints? Let's start with uniqueness - in general, each feature in the left image should match up with only one feature in the right image, and vice versa. In our random-dot stereogram, for example, we don't need to consider the possibility that a single dot in the left image could be matched with multiple dots in the right. Second is the similarity constraint - a particular feature in the scene, like the border of an object, is likely to appear roughly the same from the viewpoints of the left and right eyes. If our object border, for example, is oriented vertically in the left image and there's a high-contrast sharp change of intensity from green to red across the border, we should try to match it up with a high-contrast vertical edge in the right image that goes from green to red across the border.

[06:14] What about continuity? This constraint is used in many ways, but the essence of it is that features that are nearby in the image will tend to have similar disparities. Most of the image consists of smooth, continuous surfaces whose depth varies slowly from one location to the next. This is not true everywhere - you can have an object border, for example, with a sharp change in depth across the border, like the border of the square in the random-dot stereogram that we viewed earlier. But for most of the image, the stereo disparities at nearby locations are similar, as you'd expect from a smooth, continuous surface.

[06:59] Finally, what about the epipolar constraint? The simple version is that, our eyes are separated horizontally, so you might figure that the disparity in position of features in the left and right eye is mostly horizontal. This means that if we have a feature in the left image, like the left border of the sculpture in front here, and we're looking for a match in the right image, we only need to search along the same horizontal line, figuring that matching features will have similar vertical coordinates. This is true for the simple geometry that we assumed at the end of the last video, where the cameras were facing forward with their optical axes in parallel, but this isn't true in general.

[07:51] [slide 5] In this diagram, imagine there's a point P out in space that projects onto the left image at a particular location labeled p_L in the blue left view here. If all we know is where P appears in one image, it could actually be located anywhere in space along the line that connects p_L and P. The points labeled P_1, P_2, and P_3 are three examples of where this point could be located in space. For each possible location along this line, we can ask, where would it

appear in the right image? The right view in this figure shows you the possible places where that point could appear, and they all lie along this line shown in red, which is referred to as the epipolar line. And when we're looking for a feature in the right image that matches up with p_L in the left, we should really be searching along this red line, not along a horizontal line that's at the same vertical height. [slide 6] In computer stereo systems, the position and orientation of the stereo cameras is generally known, and with this knowledge, we can transform the stereo images in a way that corresponding features in the left and right images occur on the same horizontal lines. In this example here, if we really had the geometry indicated by the gray planes here, that's where the cameras are looking and oriented, the corresponding lines would actually be oblique like they're shown at the top here. But if we know the stereo geometry of the cameras, then we can transform those images so that they would mimic what we would see if we had a simpler geometry where in this case here, corresponding features are on the same horizontal lines. This process is referred to as stereo camera calibration, and it greatly simplifies the solution to the stereo correspondence problem. In your work on stereo, you'll just assume the simple viewing geometry that's portrayed in the yellow images here.

[10:30] [slide 7] So now, how can we solve the stereo correspondence problem? I'm going to demonstrate one general approach, and it should remind you of the fingerprint identification problem that you're working on in Assignment 1. The basic idea is to walk through the left image in a systematic way, row by row, considering small patches of the image, and for each patch, we'll jump to the same location in the right image and look for a similar looking patch, along the same horizontal line. For example, suppose we're considering this patch that's lower down in the image, on the wine bottle here that's centered at the location of this green dot. What we're going to do is go over to the right image and then start searching along the horizontal line - we're not going to search along the entire horizontal row of the image, but within a neighborhood, we'll look at patches centered on each location around that neighborhood, and try to determine which patch actually most closely resembles the particular patch that we're looking for on the left. How will we compare two patches on the left and right? We might measure the difference between the left patch and all the potential right patches it could match to, and select the one with the least difference. This should definitely remind you of the fingerprint problem - we're going to use that same measure of the sum of absolute differences between the pattern of intensities in the two patches. We could also measure instead, how similar the two patches are, and select the one that's the most similar. For this we'll use a measure of similarity referred to as normalized correlation, which I'll describe in a moment.

[12:37] [MATLAB demo] Right now, I'm going to go to MATLAB and just demonstrate the process a little more directly. So here, I'm imagining just a snippet of the process, where I've got my particular patch in the left image, centered on this green dot and I'm going to try to look for a match on the right. I'm going to consider first the same location, and notice that the green dot here isn't in the same place on the wine bottle, because the wine bottle is shifted in the right image relative to the left. Every time I push this next button here, I'm going to consider a patch

in the right image at a different location, and compute both how different it is from the left patch, and how similar it is, using normalized correlation, and I'm going to plot that value on the graphs at the bottom. So let me just start. I'm also, by the way, printing out what the disparity is between the position of the patch on the left and the position of the patch that I'm considering on the right. Right now this is a patch that's shifted way over to the left -45 pixels to the left, and I'm comparing those two patches and showing how they're different and how they're similar. Let me just continue, I'm moving the patch over, bit by bit, and plotting in each case, how different and similar they are. And I'll keep doing this until I get to a patch that's shifted much more to the right in the right image, such as that. So which of these patches was actually the best match to my patch on the left? None of them that I checked were actually a perfect match, but the best one is the patch that has the least difference or the largest similarity. The shift occurs in the same place in both, it's either the minimum of this plot here or the maximum of the similarity plot. I'm going to start the simulation again, and this time, stop when I get to that patch that's really the best match. Here we go - that's where we are, and the two patches do look fairly similar, and at this point, I'm at a disparity of -10, which means that the position of the patch in the right image is shifted by 10 pixels to the left, relative to that patch on the left. And that disparity of -10, that's the stereo disparity that I'll assign to this location from the left here, in a global disparity map, for that particular location. And later I can use that information to figure out what's the depth of that particular location in the scene.

[16:16] [slide 8] Let's go back to the slides. The question here is, how do we actually measure these two quantities, the sum of absolute differences and normalized correlation? [slide 9] In each case, you're given patches from the left and right images to compare, and I'll refer to those as p_{left} and p_{right} - here I just displayed tiny 4x4 patches. For the sum of absolute differences, we first compute the element-by-element difference between the intensities at corresponding locations in the two patches, and then take the absolute value of the differences, because we care about how large the difference is, not whether it's positive or negative. Finally, we add up all the differences over the patch, and that's a measure of how different the patches are. Optionally, we could divide by the number of pixels in the patch in order to compute the average difference between the intensities at corresponding locations.

[17:34] Normalized correlation is a measure of similarity between the patterns of intensity variation within the two patches - intuitively, we're asking how well correlated is the pattern of ups and downs of intensity within the patches? I'll first describe briefly, what's in the mathematical expression here, and then illustrate what it means with pictures. The two quantities p_{left} and p_{right} with the bars on top are the average, or mean intensity within each patch, so the first thing we do is subtract the average from the intensities within each patch. Then we multiply the two results together in an element-by-element way. The two quantities in the denominator denoted by sigma, are the standard deviations of the intensities within each patch - this is a measure of the spread of the intensities, and we're multiplying those two quantities together. Finally, this overall measure is added up over all the locations within the patch, just like in the sum of absolute differences. To understand the computation of normalized

correlation in more detail, we'll consider one-dimensional strips of intensity through two patches.

[19:04] [slide 10] Here, the blue curve is a strip through a patch in the left image, and it's the same in the left and right graphs that I've shown here. The red and green curves are intensity profiles for two different patches in the right image. What we're asking is, how well correlated is the intensity variation in the left patch, shown in blue, with the pattern of intensity variation in the right patch, shown in either red or green? Our first step is to calculate the mean intensity in each patch and subtract that from the intensities in the patch.

[19:50] [slide 11] In that case, it doesn't actually change the shape of those curves, it just centers them around 0 - the zero line here is shown in black. That's the first step. The next steps are to multiply these left and right profiles in an element-by-element way, and divide by the product of the standard deviations of the intensities in each patch. Remember that we're measuring how similar the two patches are, so we ultimately want to get a larger value for patterns that are more similar. And you can get a sense here that probably it's the example shown in red that's going to hopefully end up being more similar than the pattern that's shown in green. When we multiply the two curves element-by-element, if they're both positive, or they're both negative in the same place, we'll get a positive product, so it contributes in a positive way to the similarity. For the blue and red curves, they're positive and negative, for the most part, in the same places, so the element-by-element products shown here on the left, are all positive. But if one curve is positive in a place where the other is negative, the product is negative, as it is in many places when we multiply the blue and green curves. The final measure of correlation is the sum of these results so it's the sum of all the values in these two curves, and you can see it's a high positive value in the case of the red patch, and a small negative value for the case of the green example. So we would say that this is definitely more similar than the pattern shown in green - the numbers here fit our intuition about that.

[22:08] [slide 12] Now, let's summarize this overall strategy for solving the stereo correspondence problem, which is often described as a region-based stereo matching algorithm in this case. You'll be exploring, and enhancing, an implementation of this algorithm in the second assignment. Overall, we'll systematically step through all the patches in the left image and for each patch, we'll find the best match in the right and record the disparity in position between the left patch and its best match in the right. So we have a nested for loop here - for each row, we'll visit each column, and let p_{left} be a square patch centered on this row and column in the left image. As we check patches in the right image, we want to keep track of the best match score we've seen so far, and where we got that best score. In this version of the algorithm, we're going to use the sum of absolute differences as our match score, so a better score is a smaller value. So we can initialize the best score to a very large number, like infinity - we can actually assign values to infinity in MATLAB - and we'll initialize our best disparity. Then we consider every possible patch in the right image over some range of disparities, I'll say from $-d_{range}$ to $+d_{range}$, and we extract the right patch for each of those disparities. We compute the

match score between the left and right patches, and if the match score is better, if it's less than than the best that we've seen so far, then we'll assign our best score to the new score we just measured, and record the disparity at this place where we've got a better score. In the end, after we've looked at all the patches, we'll record that best disparity in a global disparity map, for this particular row and column. If we use normalized correlation instead, to measure how similar are the left and right patches, we just need to make a few minor adjustments, because better scores here are now larger values. We could initialize the best match score to negative infinity, measure normalized correlation, and as we go, in the case of normalized correlation if we find a higher value here instead of a lower value, then we'll record that and the disparity. The big question now is, how did we actually use the four constraints that we described earlier: uniqueness, similarity, continuity, and the epipolar constraint. I'm going to let you think about this yourself and express your thoughts in the assessment for this topic.

[25:33] [slide 13] Let's look at some results - here are some results of applying this strategy to the analysis of the stereo images of our eclectic scene of objects. On the bottom, the left picture shows the computed disparity map for the scene, where disparities corresponding to depths that are closer to the camera are shown as darker. This dark blob in the center here, this is the Arizona can that's out in front. There's a medium gray region here, for example, that corresponds to the clock, and there's a painting here at the way back, and the depths there are shown in this bright area, because they're the furthest away in the depth map. On the right, I just superimposed the zero-crossings that are obtained from convolving the images with the Laplacian of Gaussian operator, and it's mostly to give you a sense of where the edges of things actually are in the scene, so that you can compare the depths that were obtained to the actual positions of objects in the scene. For example, we can see the borders of the Arizona can and how that compares to that dark region out in front.

[27:00] [slide 14] I'd like to finish by just pointing out some of the unavoidable challenges for stereo matching. In the case of the lunchbox at the top, the bright highlights due to the reflection of a light source, they can appear in different places - they're on the far right end of the top of the lunchbox in one image, and on the left in the other image, so that definitely messes up the matching of patches in those regions. As the checkerboard suggests, there may not be one patch in the right image that's the best match - there might be lots of patches that all look the same. And the example at the bottom shows that if a surface is slanted in depth, the patches may be compressed in one image relative to the other - this elongated box here appears much shorter in the right eye, so the patches in that region won't line up so well in the left and right images. And finally, when you have objects in the scene that are hiding other objects from view, it might be that there are parts of the image that one eye sees that don't even appear at all in the other eye. That's the case in this little snippet of our images that we had before, where the left eye sees the Leonardo here, but the right eye doesn't see that at all. What's going on here is described in this diagram that's a bird's eye view of the eyes looking at a surface in the background with a surface floating out in front here that's hiding parts of it from view. So the left eye, for example, can see this patch here, but not that patch that's hidden from view, or the right

eye can’t see that particular patch. This also creates a challenge, in fact all of these factors create a challenge, both for computer vision systems and for biological vision systems.

In our next class, we’ll examine human stereo vision more closely, and explore a different strategy for stereo correspondence that captures some of the interesting aspects of human vision.