Video: MPG Stereo Correspondence Algorithm

[00:01] [slide 1] This video explores an algorithm for solving the stereo correspondence problem that was proposed many years ago by David Marr, Tommy Poggio, and Eric Grimson. We'll refer to it as the MPG algorithm. You're learning about this algorithm because it's a compelling example of the design of a computational model that incorporates many aspects of human stereo vision, and it also embodies a multi-resolution approach that's common in applications of computer vision in areas like stereo and motion.

[00:35] [slide 2] These are the properties of human stereo vision that were enumerated in the last video, and we'll start our introduction to the algorithm by considering the bold-faced items here - the image features used for matching, which are computed at multiple scales, and how the range of disparities for stereo matching varies with scale. We'll then add vergence eye movements to the matching process.

[01:03] [slide 3] The features that are matched between the left and right images are the zero-crossings in the result of convolving the images with Laplacian-of-Gaussian operators of different size. The results here were obtained from a pair of stereo images of a simple scene with three wooden blocks. Zero-crossings arise from significant intensity changes that occur around edges in a scene, like object boundaries and surface markings. They're likely to appear in both the left and right images, and their locations can be specified very precisely, at a resolution that's even higher than the pixels in an image. The goal of stereo matching here is to determine which contours in the left image correspond to which contours in the right, so that we can measure their disparity in position and use that information to recover the depth of surfaces in the scene.

[02:05] Let's first consider the coarse-scale zero-crossings at the top - they're obtained with a large operator size. In the matching process, we'll take each zero-crossing segment from the left image, like the one circled in red here, jump over to the same location in the right image, and search in the horizontal direction for a possible matching zero-crossing. It's not shown here, but the zero-crossings also have a sign - the convolution values change from negative to positive, or positive to negative, as they cross zero, depending on the sign of contrast of the intensity change in the image. We can set a rule that we only match positive zero-crossings in the left image to positive zero-crossings in the right, and the same with negative zero-crossings. We can also say that we'll only match contour segments that have a similar orientation in the two images. But still, when we go searching around for a match in the right image, there may be multiple zero-crossings of the same sign and orientation that are possible matches, so we'll limit the range of our search in the left and right directions to reduce the number of possible matches we need to consider. At the coarse scale, there's a lot of smoothing of the image, so there are fewer zero-crossings to match, which is a good thing, but the contours don't follow the actual edges in the scene as precisely, so we only compute rough disparities.

[03:42] Now consider the zero-crossings derived from the small operator size such as the one at the bottom. Here there's many more zero-crossing contours, so when we take a contour from the left image and search for a possible match in the right, we'll go to the same location, but only search over a narrow range of disparities. At the small scale the positions of the contours follow the actual edges in the scene more precisely, so when we do locate a matching zero-crossing, we can specify its disparity more precisely. So to summarize so far, at the coarse scale we'll be able to compute rough disparities over a larger range, and at the fine scale, we can compute more accurate disparities, but only over a smaller range. How can we get the best of both worlds? Through communication across scales. The basic idea is that we'll use the stereo disparities computed at the coarse scale to guide where we search for matches at the smaller scales. For example, if it was determined at the coarse scale, that the object boundary circled in red is shifted far to the left in the right image, then at the fine scale, we'll search around an area in the right image that's also shifted far to the left. That's a taste of how multiple scales will be used. The next thing we'll do is bring in the idea of moving the eyes around the scene to further expand the range of surface depths that can be handled.

[05:28] [slide 4] I'm going to walk through a sequence of slides that illustrate how the algorithm builds up a representation of the depths of surfaces in a complex scene that has a large range of depth. The algorithm does this by moving the eyes around to focus on different depths. Imagine a 3D scene with a stack of surfaces, shown in the upper left corner. The red arrow shows the direction the viewer is looking. The bird's eye view is shown in the upper right corner. Imagine that the observer is looking at a random-dot stereogram constructed from this scene that's displayed on a computer screen, just like the one you're looking at on this slide. In the bird's eye view, the computer screen is shown at the bottom of the stack, and the surfaces, after the observer can fuse together the stereogram and perceive the surfaces, the observer will see them as coming out of the screen, toward them, as portrayed in the diagram.

[06:47] [slide 5] Here I added colored lines to the stereogram, indicating where the border of each surface is located. The disparity map at the top shows the amount of shift in position, the stereo disparity, for each surface in the scene. For example, the center surface has a disparity of +30, which means that the region outlined in blue in the center of the stereogram, is shifted to the right by 30 pixels in the right image here - you can kind-of see that if you look back and forth between the two images.

[07:27] Initially, we'll say that the eyes are focused on the computer screen, at the point shown with the red dot in the bird's eye view, where the two lines of sight meet. The pink and blue bars around the red dot in this bird's eye view, they are meant to convey that there's only a limited range of depth around the fixation distance where the left and right images can be fused together, and reveal a clear surface at a particular depth. The pink region indicates the disparity range that can be handled at the fine scale, and the blue region depicts a larger range of depths that could be considered at the coarse scale. As the eyes shift their focus to a different depth, this range of fusible depths in pink and blue, will move with the eyes. The final piece of the picture is the evolving depth map in the lower right corner. At the outset, the eyes are

focused at the depth of the computer screen that contains the outer surface, so the outer surface has zero disparity, like the fixation point, and the eyes can fuse together the dots in the left and right images at the finer scale and see a clear surface at the depth of the screen. I depict this in the depth map with the same solid dark gray that I displayed in the disparity map in the outer region. At this point, the observer has no idea where those surfaces are in depth in the middle here, so I portrayed that just with random dots.

[09:24] [slide 6] So what might happen next? Imagine that the viewer now moves their eyes to focus on a point in space on the side of the stereogram, that's slightly in front of the screen, shown by this new position of the red dot and new lines of sight from the eyes. The range of depths where the visual system can fuse the images together has also moved forward, and the first step of the stack of surfaces is now within a range that can be fused at the coarse scale covered by the blue band, but it's not within range of fusion for the fine scale shown in pink. Remember that at the coarse scale, we only get a rough sense of where surfaces are in depth, and I portrayed this in the emerging depth map as a blurry area around that first step. Also remember that stereo disparities depend on where the eyes are focused, so in the disparity map at the top, the disparities of all the surfaces have now changed a bit. I also changed the locations of the colored outlines of the surfaces on the stereogram to reflect these new disparities. In the evolving depth map, there's still a central area of random dots here, because these central surfaces are still out of range of fusion. So again, the observer has no idea where they are in depth. Given this rough idea of where this first inner surface is in depth, the next action by the stereo system might be to move the eyes a little closer to the viewer so that the depth of this fuzzy surface becomes more clear.

[11:28] [slide 7] Here, in the next step, the eyes are focused on a location right on that surface, so its depth is now within range that can be fused at the fine scale in pink. In the evolving depth map, I now show that inner surface as a uniform gray color to indicate this. Given the movement of the eyes in depth, the disparities of all the surfaces again shifted slightly, as conveyed in the disparity map at the top, and the colored borders of the surfaces on the stereogram have also been shifted a little.

[12:12] [slide 8] Now suppose another eye movement is made to a location on the other side of the image that's even closer to the observer. The effect of this new eye movement is shown in this next slide. Let's say it's a bit further forward than the next inner surface here, so the new surface is within a range of disparity that can be fused at the coarse scale, it overlaps the blue region there, but it's outside the range that can be fused at the fine scale. So I again depict this in the evolving depth map with a blurry region. The disparities changed again for this new position of the eyes, and that's shown both in the disparity map at the top and reflected in the positions of the colored borders of the surfaces on the stereogram.

[13:08] [slide 9] Given this rough idea of where this particular surface is, that's shown fuzzy here, maybe the next step would be to move the eyes further away a little bit, onto that surface, so that's now a clear disparity. Here's the next surface, and that surface is now within range for

matching at the fine scale, and that's depicted by this light gray area here. Still the center is out of range of fusion, so the observer has no idea where that is in depth, so that's still shown as random dots. And the disparities have been adjusted in the disparity map and the positions of the surface borders have also been changed. The viewer at this point might figure that maybe that stuff in the center is even closer to them, they're seeing a pattern here. [slide 10] So let's say the viewer moves their eyes once more, to a depth that now puts that surface in range that can also be fused by the fine scale here, and so they now have a clear impression of where all the surfaces are. We've assumed all along here that even though at each moment, you can only fuse a certain range of the stereogram, once you have fused an area, you're putting that information into some kind of a memory, that's what this depth map is, and that persists as you then fuse more and more of the pattern. So that's the general idea of how we can build up a depth map for a scene with a wide range of depths, using a matching process that at each moment, can only match left and right features over a narrow range of depths around the fixation distance.

[15:30] [slide 11] How can these ideas be built into an algorithm that we can implement in a computer vision system, that matches features in real stereo images. We hinted at this earlier. The MPG stereo algorithm matches zero-crossing contours at multiple scales. At the coarse scale, the search for a match extends over a larger neighborhood, and the disparities computed at that scale are used to guide the matching of zero-crossings at the fine scale, where it only searches over a small neighborhood.

[16:05] [slide 12] First, let's be more explicit about the features that are matched, using these stereo images. This is a classic stereo image pair that's been used as a benchmark for testing many stereo algorithms, because the researchers also provided a true depth map of the scene, which allows the results of stereo processing to be evaluated in a rigorous way. I'll just use this scene to illustrate the image features for matching in more detail.

[16:40] [slide 13] This slide shows the zero-crossings derived from two operator sizes, and they look different from what you saw before. We said earlier that we only match zero-crossings of the same sign. The white contours here correspond to intensity changes that go from dark to light as you move from left to right across the image, and the black contours are zero-crossings where the intensity changes from light to dark across an edge. The other thing that's different is that there's something missing. The horizontal parts of the contours are missing. That's because we can't directly match locations along horizontal contours if we're just searching in the horizontal direction. If you think about the extended edges at the top of each one of these pictures here, if you have a particular location of the left image, searching in a small region, it's impossible to say where the matching feature is on the right - all of these points along that edge look exactly the same. So the stereo matching process only tries to match features like the zero-crossings I showed here, that have a vertical extent.

[18:03] The general matching strategy will be to start with the coarse scale, and for each zero-crossing on the left, search within a limited neighborhood of that location on the right, to

find a matching zero-crossing that has a similar contrast, and we can look for a similar orientation as well. Suppose that for this white contour circled in red, we find a matching contour on the right, and imagine that it's shifted 20 pixels to the left relative to the location of this contour in the left image. When we're matching features at the fine scale, let's suppose we want to find a match for a zero-crossing contour in the same region of the left image here. If we searched around the same location in the right image, the correct match may be way out of range, given the small search neighborhood used at the small scale. To guide the search at the fine scale, the algorithm looks in the same region at the coarse scale, to find the nearest contour that was matched at that scale. Suppose this is the nearest contour, and we recorded a disparity of -20 pixels for that location, a shift of 20 pixels to the left. Then when searching for a match at the fine scale, we'll start the search at a location that's 20 pixels to the left, and look in a neighborhood around this shifted location. In this case, this new search area actually includes the correct matching contour. So overall, when matching contours at the fine scale, we use disparities computed at the coarse scale to determine where to start the search. And as zero-crossing contours are matched up between the left and right images, the algorithm records the computed disparity for each zero-crossing contour. [slide 14] Here are some results of processing a pair of aerial stereo images with this algorithm. The result displays the zero-crossings that were matched, with the final disparity coded in color. Closer surfaces are displayed at the yellow end of the color spectrum and more distant surfaces, like the ground, shown in blues.

[20:40] [slides 15-16] To help you get a more concrete feel for the matching process, I'm going to present a simplified version of the algorithm that appeared in a textbook on AI by Patrick Winston. You'll be able to hand simulate the steps of this algorithm. I'll go through one example here, and there's another example that you'll complete with your group on the current assignment. The algorithm is described in words here, in two parts, but rather than reading all these words with no pictures, I'm going to jump to a visual example and describe the steps of the algorithm through this example. When you're doing your own hand simulation, you'll probably want to refer to the textual description as you go, as well.

[21:33] [slide 17] Our hand simulation uses a different visual representation of the edges we're trying to match between the left and right images, and as usual, we're doing things in one dimension here. The numbers on the horizontal axes indicate image position - we're looking at a really itty bitty image snippet here. The blue and red bars mark the locations of zero-crossing segments in the left and right images, and in this particular visualization, they're both overlaid on the same axis. I'm just going to refer to these segments as edges. The blue bars are edges from the left image and the red bars are edges from the right image, and these edges appear at different positions in the left and right images because of the different stereo perspectives. In the matching process, for each blue edge in the left image, we're trying to figure out which red edge in the right image is the correct match. The edges in the top row represent coarse scale zero-crossings that were derived from a large operator size. The middle and bottom rows are two copies of the same thing - these are the fine-scale zero-crossings derived from a small

operator size. Along the left side, w refers to the size of the Laplacian-of-Gaussian operator, as it was described in our introduction to edge detection. The parameter m refers to the search range we're going to use at each scale. In the text outline of the algorithm, this quantity is referred to as the matching tolerance, and you can see that it's half the operator size in this case.

[23:34] To match the edges at the coarse scale, the first step is, for each blue edge in the left image, we find the nearest red edge in the right image. There are just two edges in the left image, and these green arrows show the nearest edge in the right image. The next step is, for each red edge in the right image, find the nearest blue edge in the left image. These two dashed arrows show the nearest neighbors in the left image for each of the two red edges in the right. The next step of the algorithm says, for each pair of left and right edges that are closest neighbors of one another, determine whether their disparity in position is within the matching tolerance m. Both pairs here are mutually closest edges and the disparities are both within the matching tolerance of 4 for the coarse scale. So we accept these two matches and assign an initial disparity for those two regions of the image. Disparity refers to the shift in position as we go from the left image to the right image, so it's +4 for the pair on the left and -2 for the pair on the right.

[25:12] We have two copies of the fine-scale edges so we can compare what happens when we use the coarse-scale disparities as a guide, to what would happen if we don't. Let's jump to the bottom and see what happens if we apply the same strategy that we just used at the top, ignoring the information from the coarse scale. Again, the first step is to find the nearest edge in the right image for each edge in the left image. The solid green arrows here complete that step. Then we do the second step, we find the nearest blue edge in the left image for each red edge in the right image. The dashed arrows here show the results of this step. Finally, as before, we look for pairs of red and blue edges that are closest neighbors of one another, and there are two, the pair at locations 5 and 6, and the pair at locations 15 and 16. The disparity in position for those pairs is within the matching tolerance of 2 for the small scale, so we accept these two matches and again assign a disparity to tham, so that disparity is -1 for the left pair and +1 for the right pair. But… these are not actually correct matches for this scene. We know from the matches at the coarse scale, that stuff on the left side of the image really has a disparity around +4, and stuff on the right side really has a disparity around -2. So can we improve on the matching of fine-scale features by taking advantage of what we know at the coarse scale?

[27:18] Now we're using Part 2 of the text description of the algorithm, which is a bit more involved. The first step is to consider each blue edge in the left image, and for each of those edges, we find the nearest edge at the coarse scale that has a disparity assigned to it from the previous matching. For the two left edges on the left half of the image, we see that there is a matching edge from the coarse scale that had a disparity of +4. For the two fine-scale edges on the right side of the image, their nearest match at the coarse scale has a disparity of -2. The next step is to offset the left edges at the fine scale by that disparity that was found at the coarse scale. So we're going to shift the first two left edges to the right by 4 pixels and shift the

last two edges to the left by 2 pixels. I drew dashed blue bars at the new, shifted positions. Now we proceed as we did before. We find pairs of edges from the left and right images that are mutually closest neighbors of one another, with a shift in position that's within the matching tolerance of 2 for the smaller scale. But this time, we're going to match the offset versions of the blue bars from the left image. So we're matching up the dashed blue cars and the dashed red bars. This results in 4 pairs of matching features, shown with the green arrows as before.

[29:25] Let's scrutinize what we did so far. Consider the left edge at location 2. We effectively used the disparity at the coarse scale to shift the region where we started searching for a match in the right image there, so we shifted it over to position 6 and searched around there, over a small neighborhood, and we found a match to the red edge at position 5. So in the end we'll say that the left edge at position 2 matches the right edge at position 5, as shown with the solid purple arrow. The next edge was originally at location 6 here, and at the coarse scale, it said the disparities in that region are +4, so we shifted our region where we're going to start our search, over to location 10, and we found a nearby right edge at location 9. So in the end we can say that this bar at position 6 in the left image really matches with this red edge here in the right image, at location 9. Same thing for the other examples. The left edge at location 15, we used the disparity at the coarse scale to start our search at location 13 instead, and we found a right edge at position 12, nearby, so we're going to say that this left edge at 15 really matches the right edge at position 12. And similar for the last example, we're going to end up saying that this left bar at location 18 really matches a right edge at location 16.

[31:45] [slide 19] If we take away all the clutter, we end up with final matches at the fine scale between these pairs of edges in the left and right images, with the final disparities shown for each one. They tell us, for example, that the disparity on the left side of the image is really +3, not the rough disparity of +4 that we found at the coarse scale. So in more concrete terms, this is an example of how you can use information across scales to solve the stereo correspondence problem.

[32:22] To summarize, we explored the Marr-Poggio-Grimson stereo algorithm. It's a multi-resolution, feature-based stereo matching algorithm that embodies a role for vergence eye movements and multi-scale processing that we know are important aspects of human stereo vision. It's been implemented and tested in a computer vision system, and we also described a simplified, but very explicit, version of the algorithm that you can hand simulate, which embodies the interactions across scale that are a key element of the original algorithm.