CHAPTER 33

Stereo: Harder Topics

33.1 STEREO AS CONSTRAINED OPTIMIZATION

The key problem with the simple matching procedure is that matches using small windows tend to be noisy and matches using big windows tend to be oversmoothed. You should see this as a result of the fact that the matches are independent – each point finds its best matching point without regard to the others. When the window is small, there is nothing that communicates to a match that it is too different from neighboring matches. When the window is big, there is nothing that communicates to a match that it is too similar to neighboring matches. This can be fixed by phrasing stereo matching as an optimization problem. There are geometric constraints on matches that are useful, if not universally true.

33.1.1 Contraints on Matches

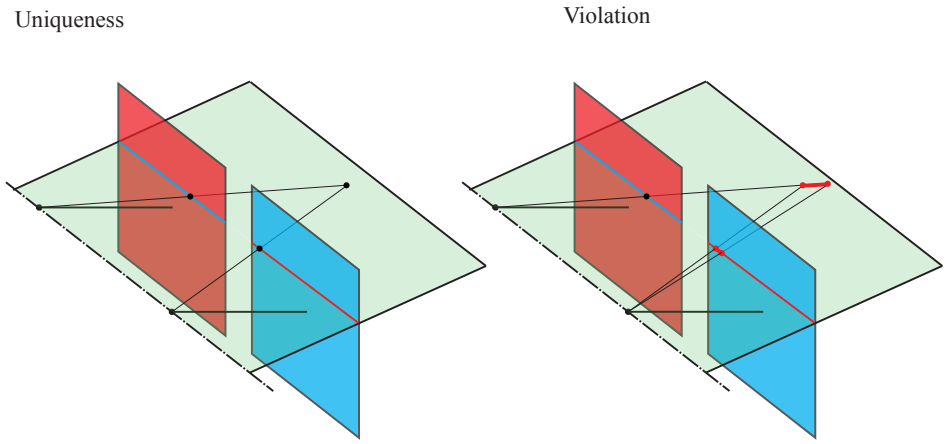


FIGURE 33.1: *The uniqueness constraint requires that matches be unique. Each point on one side should have at most one match on the other (as in the* **left** *drawing). It can be violated if you view a straight object in exactly the right viewing position (***right** *- the* **red** *interval), but this requires a special viewpoint. It is usual to ignore this effect and assume that the constraint applies.*

Definition: 33.1 *Uniqueness*

A *uniqueness constraint* requires that each point in one image have no more than one matching point in the other.

This constraint implies that something that is a point in one image should not be a line segment in the other. This is usually, but not always, true, as Figure 33.1 demonstrates. The constraint is only violated if one camera sees an object from a special direction. In jargon, the view is *not generic*. Notice that if one camera moves very slightly, the object will be a line segment in both views. In turn, the situations where the uniqueness constraint is violated can be (and are!) ignored.

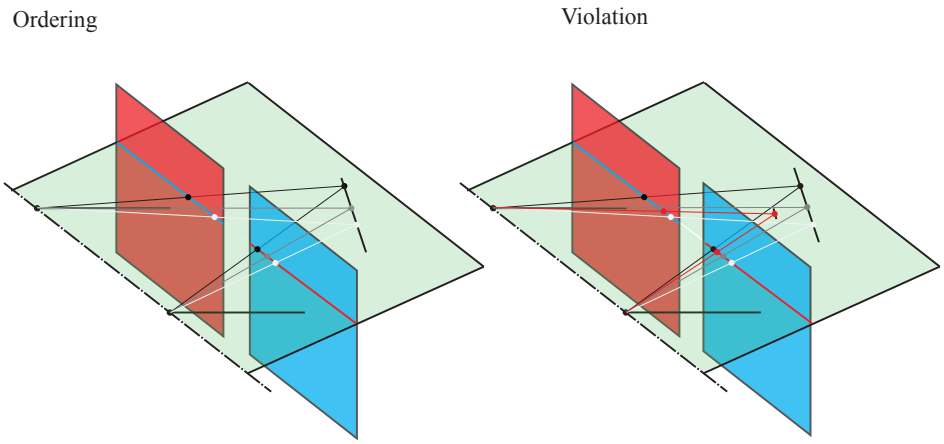


FIGURE 33.2: *The ordering constraint requires that matches appear in the same order on left and right, as on the* **left**, *where the points are matched in order* **light gray**, **mid gray**, **black** *on both sides. This constraint can be violated, as on the* **right** *side. Violation requires looking "into" a hole in a surface. Here the* **red** *point lies on a surface in front of the main piece, but the other points are still visible to both cameras. Points appear in a different order on the two sides. One can ignore this effect and assume that the constraint applies, but the consequences are more severe than for uniqueness.*

Definition: 33.2 *Ordering*

An *ordering constraint* requires that matched points appear in the same order in right and left image.

As Figure 33.2 demonstrates, this constraint does not always apply. The conditions where it is violated can be (and often are) ignored for the same reason

that possible violations of the uniqueness constraint are ignored. Applying the ordering constraint can result in poor disparities for thin objects or objects with large holes in them **exercises**.

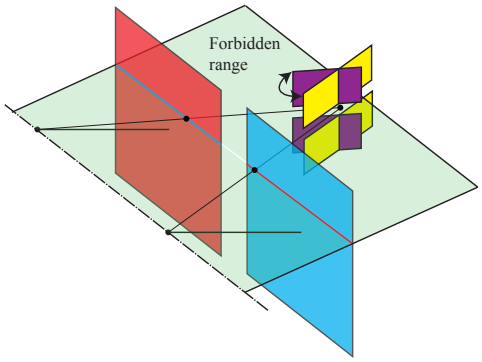


FIGURE 33.3: *There is some limit on the disparity gradient that depends on the particular point being viewed. This figure shows the cameras of Figure 32.1 viewing a point. This point lies on a surface, as points do. The surface on which the point lies cannot obstruct the view from either camera (or you wouldn't be able to see the point). This means that some orientations (*forbidden *in the figure) are not possible. You can compute surface orientation from the gradient of depth, and so from the gradient of disparity. This means that, near a matched point, some disparities cannot be possible. A detailed expression is tricky to use, and it is usual to require that disparity gradients be below some threshold.*

For a match to be valid, the camera must be able to see the point in both views. Points do not float in space, but lie on surfaces. If the surface is tilted too much toward one or another camera, the surface itself will make the point invisible in the other camera. Figure 33.3 illustrates this idea. This results in a constraint on the surface orientation. Surface orientation is given by the derivative of depth, so there is some constraint on the acceptable disparity gradient. The details depend on cameras, viewpoint and so on, and are usually ignored.

Definition: 33.3 *Disparity gradient*

The *disparity gradient constraint* means the gradient of disparity is limited. It is derived from visibility considerations. In detail, the constraint depends on a number of factors, but it typically appears as a fixed threshold on acceptable disparity gradient.

In the epipolar geometry of Figure 32.1, the disparity gradient constraint boils down to the following. Assume K is the threshold. If (x_1, y_1) matches (x'_1, y'_1), then $(x_1 + \Delta x, y_1)$ can match only points in the range from $x'_1 + (1 - K)\Delta x$ to $x'_1 + (1 + K)\Delta x$.

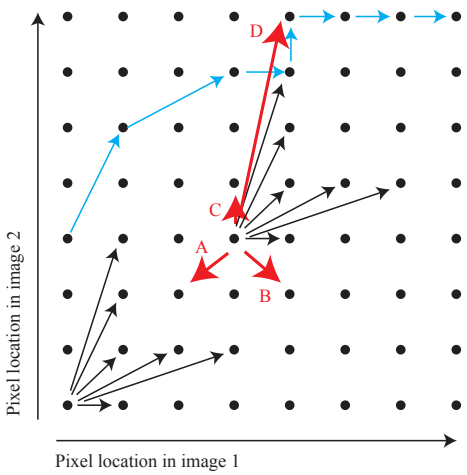


FIGURE 33.4: *Drawing the matching process for a single epipolar line as a graph exposes matching as a dynamic programming problem. The details appear in the text.*

33.1.2 Dynamic Programming

Assume that matches along an epipolar line interact, but matches do not interact across lines. Further, assume that the interactions are local – each match affects the neighbors on either side, but affects no other. You can then set up a straightforward constrained optimization problem to solve for matches. This problem is easily solved using dynamic programming along each epipolar line. Solutions tend to be good.

Choose an epipolar line to construct matches, say $y = y_0$. Assume you have N locations on the left epipolar line. At each location, you must determine the corresponding location on the right epipolar line, which also has N locations. Set up a grid of pairs of (image 1 location, image 2 location) for this pair of epipolar lines. Figure 33.4 shows this grid for two very small images. The element (i, j) of this grid represents a match between (i, y_0) in the left image and (j, y_0) in the right image.

Now insert directed edges in this grid. A directed edge between (i, j) and (k, l) represents a match between (i, y_0) and (j, y_0) followed by a match between (k, y_0) and (l, y_0). The constraints mean that not every directed edge can exist. Some of the many edges are shown in Figure 33.4, with edges that cannot occur in **red**. Matching proceeds along the left epipolar line, so directed edges can never go backward (which is why edge A is in red in that figure). The ordering constraint means that directed edges can never go down (which is why edge B is in red in that figure). The uniqueness constraint means that directed edges cannot be vertical (which is why edge C is red in that figure). The disparity gradient constraint means that other directed edges are impossible as well, because they imply too large a disparity (which is why edge D is in red in that figure). **exercises**

Now associate a cost with each node and a cost with each directed edge. Find



FIGURE 33.5: *Two frames (3 and 5) from a well known sequence of five frames, the Tsukuba dataset. On the* **right**, *the known ground truth disparity in the coordinate system of frame 3.* Image credit: *Images from the Middlebury dataset at* `https://vision.middlebury.edu/stereo/data/`, *originally described in Y. Nakamura, T. Matsuura, K. Satoh, andY. Ohta. Occlusion detectable stereo—occlusion patterns in camera matrix. In CVPR, pages 371–378, 1996.*

the directed path from the left of the grid to the right which has the lowest cost. The cost is chosen so this path represents the match. Each node represents a possible correspondence, so each node is associated with a cost derived from the photometric consistency constraint. Compute a representation $\mathbf{r}$ of the image for left and right images. The cost is obtained by comparing the representations. The representation could be pixel intensity, pixel color, or a vector of filter outputs. Write $C(i,j)$ for the cost that compares the representation of the left image at (i, y_0) to that of the right image at (j, y_0), so

$$C(i,j) = \|\mathbf{r}(i,y_0) - \mathbf{r}(j,y_0)\|^2.$$

Very often, $\mathbf{r}$ is just the image intensity (color is better) and

$$C(i,j) = \|\mathcal{I}(i,y_0) - \mathcal{I}(j,y_0)\|^2.$$

Alternatively, $\mathbf{r}(i,y_0)$ might be a window around (i,y_0) straightened into a vector.

Associated with each acceptable directed edge is a cost. For example, smoothness implies that large changes in disparity are deprecated, so you can associate a cost with a legal edge from (i,j) to (k,l) like

$$C(i,j;k,l) = \alpha\left[(j-i) - (k-l)\right]^2$$

A solution will be a directed path through the nodes representing pairs of points that correspond (**blue** in Figure 33.4). If i corresponds to j, the node (i,j) appears in a solution. Finding the best match is now the dynamic programming problem of finding the lowest cost path from start column to end column in the directed graph. **exercises**

The dynamic programming procedure has a problem. It obtains the best solution for each row independent of each other, which produces characteristic errors. Solutions have a characteristic streaky appearance, because the solution for row i ignores the solution for row $i-1$ (Figure 33.6; **exercises**). You could try and fix this by solving for row 1; then solving for row 2 conditioned on row 1 (you would



FIGURE 33.6: *Dynamic programming based matches produce characteristic streaking errors in the estimated disparity.* **Left**, *frame 3 of the Tsukuba sequence (Figure 33.5);* **center**, *the ground truth disparity; and* **right**, *an estimate from a dynamic programming based method.* Image credit: *disparity estimate is part of Figure 17 of A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms, D. Scharstein and R. Szeliski, 2001*



FIGURE 33.7: *CRF stereo solutions can be extremely strong.* **Left**, *frame 3 of the Tsukuba sequence (Figure 33.5);* **center**, *the ground truth disparity; and* **right**, *an estimate from a graph-cuts formulation for solving the CRF.* Image credit: *disparity estimate is part of Figure 17 of A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms, D. Scharstein and R. Szeliski, 2001*

add some terms to the cost function, but it isn't worth expanding them here); and so on. The difficulty with this approach is the disparity map you obtain depends on the row you start with. You could equally start at the bottom of the image and work your way up, and come up with a somewhat different disparity map. Further, an error early in the reconstruction can propagate **exercises**. Finally, dynamic programming requires you impose the ordering constraint **exercises**, meaning you should expect trouble with narrow things and objects with holes.

33.1.3 Stereo as a Conditional Random Field

An alternative is to set up a much harder optimization problem that takes into account pixels in rows above and below the current pixel. You want to know a disparity value for each pixel in the left image (say). Write $\delta_{i,j}$ for the disparity at i, j in the left image. A photometric consistency loss follows easily. Compute a representation $\mathbf{r}$ of the image for left and right images, as above.



FIGURE 33.8: *CRF stereo solutions can be extremely strong.* **Left**, *frame 3 of the Tsukuba sequence (Figure 33.5);* **center**, *the ground truth disparity; and* **right**, *an estimate from a graph-cuts formulation for solving the CRF.* Image credit: *disparity estimate is part of Figure 17 of A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms, D. Scharstein and R. Szeliski, 2001*

Write $C(\delta, i, j)$ for the cost of having disparity δ at location (i, j) in the left image. This disparity implies that (i, j) in the left image matches $(i + \delta, j)$ in the right image. Then use

$$C(i, j) = \| \mathbf{r}_l(i, j) - \mathbf{r}_r(i + \delta, j) \|^2$$

as a photometric loss. Very often, $\mathbf{r}$ is just the image intensity (color is better) and

$$C(\delta, i, j) = \| \mathcal{I}_l(i, j) - \mathcal{I}(i + \delta, j) \|^2.$$

The idea underlying dynamic programming stereo is that disparities along an epipolar line interact. Now you want disparities at different locations in the left image to interact. Do this by setting up a graph where each vertex corresponds to a pixel location in the left image, and each edge corresponds to a pair of pixels whose disparity you want to compare. Write $\mathcal{N}(i, j)$ for the set of pixels that are connected to the pixel at i, j by an edge – these are *neighbors* to the pixel at i, j. You compare the disparity at i, j to the disparities at each neighbor. $\mathcal{N}(i, j)$ could just be the next pixel on the row, which would get us the dynamic programming case. Alternatively, it could be the *four neighbors* – up, down, left, right. Or it could be the *eight neighbors* – the four neighbors, together with top left, top right, bottom left and bottom right.

Associated with each edge in the graph will be a cost function. This cost function penalizes pairs of disparities across the edge that are undesirable. It depends on the disparity at either end of the edge, and will be symmetric because the edge is undirected. For example, the cost function could be

$$B(u, v) = \begin{cases} \| u - v \|^2 & \text{if } \| u - v \| < k \\ L & \text{otherwise} \end{cases}$$

for L some large value, and u and v the disparities at either end of an edge. The form of B does not usually depend on the edge (but could in special cases).

Now quantize all disparities to a fixed set of V values, $\delta_1, \ldots, \delta_v$. You must choose the value δ of the disparity at each location. This is equivalent to choosing

a *one-hot vector* (a vector with all but one component zero, and the remaining component one) at each location. Place a vector $\mathbf{c}_{ij}$ at the node representing the i, j'th pixel, where

$$\mathbf{c}_{ij} = \begin{bmatrix} C(\delta_1, i, j) \\ \dots \\ C(\delta_v, i, j) \end{bmatrix}.$$

Associate with each edge a matrix

$$\mathcal{M}_{\mathrm{e}} = \begin{bmatrix} B(\delta_1, \delta_1) & \dots & B(\delta_1, \delta_V) \\ \dots & \dots & \dots \\ B(\delta_V, \delta_1) & \dots & B(\delta_V, \delta_V) \end{bmatrix}.$$

This usually does not depend on the edge (but could in special cases). Now at each location place a one-hot vector. Write $\mathbf{h}_{ij}$ for the one-hot vector at (i, j). The cost of your choice is

$$\mathcal{C}_{\mathrm{CRF}}(\mathbf{h}_{11}, \dots, \mathbf{h}_{MN}) = \sum_{i,j} \mathbf{h}_{ij}^T \mathbf{c}_{ij} + \sum_{ij} \left[\sum_{k,l \in \mathcal{N}(i,j)} \mathbf{h}_{ij}^T \mathcal{M}_{\mathrm{e}} \mathbf{h}_{kl} \right]$$

You must now obtain the $\mathbf{h}_{11}, \dots, \mathbf{h}_{MN}$ that minimize

$$\mathcal{C}_{\mathrm{CRF}}(\mathbf{h}_{11}, \dots, \mathbf{h}_{MN})$$

subject to two constraints. First, every entry in every $\mathbf{h}$ is either one or zero. Second $\mathbf{1}^T\mathbf{h} = 1$ for every $\mathbf{h}$. The solution to this optimization problem can be interpreted as the mode of a complicated probability distribution on the disparities. The particular class of distribution is known as a *conditional random field* or *CRF*.

This optimization problem is intractable. Remarkably, as long as the cost function has the property that it is cheaper for two disparities at either end of an edge to agree than to disagree, there are very effective approximate solution procedures. Any cost function that doesn't have this property will be penalizing smooth disparity fields, which isn't realistic.

One approximation procedure – *message passing* or *belief propagation* – involves adjusting a probability distribution on disparities at each location to be more consistent with their neighbors. An alternative approximation procedure repeatedly adjusts disparities by reducing the problem to a graph cut, then applying a fast solver to that. This is a *graph cuts* method. Details of either process are out of scope (or rather, there just isn't space), but extremely powerful stereo algorithms result (Figures 33.7 and 33.8).

33.2 USING STEREO GEOMETRY WITH ACTIVE SOURCES

The geometry of projectors is very like that of cameras. It is quite fair to abstract a projector as a camera where: (a) you get to choose what appears on the image plane; and (b) light travels backward from the image plane into the world. This means that you can build very strong stereo systems with one camera and one projector.

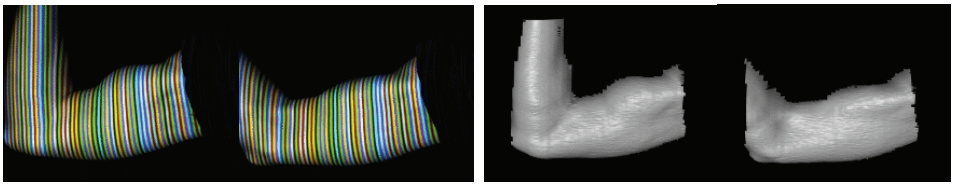


FIGURE 33.9: *On the* **left**, *two frames showing vertical colored stripes projected onto an arm, encoding disparity. On the* **right**, *corresponding frames showing the reconstruction.* Image credit: *Two frames from reconstructions described in Li Zhang, Brian Curless, and Steven M. Seitz. Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming. In Proceedings of the 1st International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT), Padova, Italy, June 19-21, 2002 and published at* `https://grail.cs.washington.edu/projects/moscan/`

33.2.1 Local Codes yield Disparity

Assume the first camera of Figure 32.1 is actually a projector. Arrange a slide so that each column of the slide has a different color. View a white surface with no other light, and think about the image that the second camera sees. Each point in the y_0 row in the image receives light from the y_0 row in the slide. The color of the light at the i'th pixel in the row reveals where that pixel is in the slide (each column has a different color) and so the disparity **exercises**. This means you can read the disparity and so the depth off the image. In principle (and depending on your matching procedure) you can do this extremely fast, and so measure depth accurately for moving objects (Figure 33.9)

This observation leads to some natural but elaborate constructions. It is quite inconvenient to distinguish between a large number of very similar colors, but the size of the disparity is limited. This means you could use a repeating pattern of fewer colors because you won't get mixed up (all but one will have too big a disparity). Rather than using vertical stripes of color, you could have a pattern of random dots. If you want your system to work without disturbing people, you could use a pattern of random dots and have the projector project in infrared (Figure 33.10).

33.2.2 Using Laser Stripes

An alternative version of this construction deals with one point on each epipolar line at a time. Imagine your projector shows a movie, where a vertical stripe is swept across the screen. If the camera was synchronized to the projector, the image of the stripe would yield disparity (Figure 33.11). A more practical version of this arrangement uses a laser projector that creates a plane of laser light. There are advantages to using a laser. It is very bright and is monochromatic, so there tend to be no problems with detecting the light. The light is strongly collimated, so that intensity on a plane of light goes down (roughly) as 1/distance, meaning that you can measure a significant range of depths.



FIGURE 33.10: *A pattern of random dots projected by the projector in a Kinect device. The projector slide is known, and the camera can estimate depth by matching as in a random dot stereogram. The dots are in infrared, so not visible to the user.* Image credit: *An image published at* `https://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/` *by*

In the simplest configuration, you rotate the laser projector about its focal point, sweeping a plane of light perpendicular to the epipolar lines (this is equivalent to the setup of Figure 33.11). If you are scanning a large object, you could put the projector and camera together into a head. Put that head on a rig that moves it automatically, and scan repeatedly. This gives you one point cloud per scane, and you must then register them to reconstruct the object (Section 36.3.

33.3 YOU SHOULD

33.3.1 be able to:

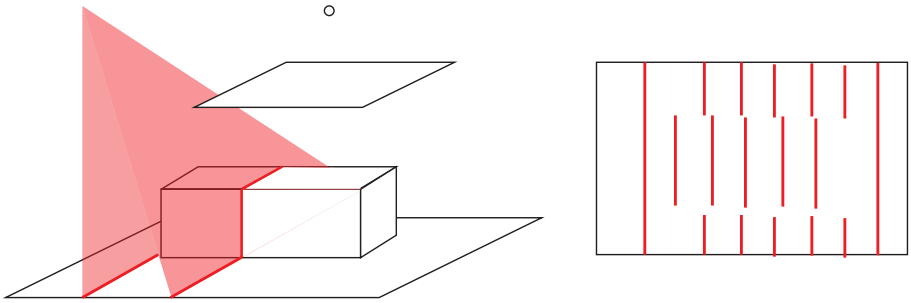


FIGURE 33.11: *The projector on the* **left** *casts planes of light into the scene. These are viewed by a camera. Their shape in the image (***right***) is a cue to the depth in the scene.*

EXERCISES

QUICK CHECKS

33.1. Section 33.1.1 has: "Applying the ordering constraint can result in poor disparities for thin objects or objects with large holes in them. " Explain.

33.2. Section 33.1.2 has: " Matching proceeds along the left epipolar line, so directed edges can never go backward (which is why edge A is in red in that figure)." Explain.

33.3. Section 33.1.2 has: "The ordering constraint means that directed edges can never go down (which is why edge B is in red in that figure)." Explain.

33.4. Section 33.1.2 has: "The uniqueness constraint means that directed edges cannot be vertical (which is why edge C is red in that figure)." Explain.

33.5. Section 33.1.2 has: "The disparity gradient constraint means that other directed edges are impossible as well, because they imply too large a disparity (which is why edge D is in red in that figure)." Explain.

33.6. Section 33.1.3 has: " Solutions have a characteristic streaky appearance, because the solution for row i ignores the solution for row $i-1$ (Figure 33.6)." Explain.

LONGER PROBLEMS

33.7. Section 33.1.2 has: "Finding the best match is now the dynamic programming problem of finding the lowest cost path from start column to end column in the directed graph." Explain (this is a longer problem, because the explanation takes some care).

33.8. Section 33.1.2 has: " You could try and fix this by solving for row 1; then solving for row 2 conditioned on row 1 (you would add some terms to the cost function, but it isn't worth expanding them here); and so on."

- **(a)** What terms can you add to the cost function so that you can use dynamic programming on each row?
- **(b)** If you do add these terms, an error early in the reconstruction can propagate. Why?
- **(c)** Section 33.1.2 has: "dynamic programming requires you impose the ordering constraint" Explain.
- **(d)** Why does this mean "you should expect trouble with narrow things and objects with holes."

PROGRAMMING EXERCISES

33.9. Write code to reproduce 33.6. Use your codebase to answer the following questions.

- **(a)** When you compute the photometric loss, do you get better results using color or intensity?
- **(b)** When you compute the photometric loss, do you get better results using RGB color or LAB color?
- **(c)** What happens if you change the size of the largest acceptable disparity gradient?
- **(d)** Section 33.1.2 has: "dynamic programming requires you impose the ordering constraint, meaning you should expect trouble with narrow things

and objects with holes." To what extent is this a problem? (Use your codebase and other stereo pairs).

(e) Section 33.1.2 has: "You could try and fix this by solving for row 1; then solving for row 2 conditioned on row 1 (you would add some terms to the cost function, but it isn't worth expanding them here); and so on. The difficulty with this approach is the disparity map you obtain depends on the row you start with. You could equally start at the bottom of the image and work your way up, and come up with a somewhat different disparity map. Further, an error early in the reconstruction can propagate". How significant are these effects? Can you make this strategy work in practice?