GENERAL

- Problem set 3 due next Thursday.
- Recitation Tomorrow.
- Will probably announce an extra office hour for next week.
- Suggestions of papers for project posted.
  - You can still choose a different paper, or do a project on your own research.
- Project Proposals Due Sunday Oct 29th 11:59pm.
- PSET1 Grades Out. Should have received mail with password & folder for feedback.

RECTIFIED BINOCULAR STEREO

Epipoles at infinity
Epipolar lines all parallel to the X axis

L[x,y] matches to R[x',y]
Epipolar Lines are Horizontal

Why are we doing this? Two equations tell us 3D position of point
**RECTIFIED BINOCULAR STEREO**

**Left**

L[x,y] matches to R[x',y]

Visibility Constraint:  
\[ x' \leq x \]

(object in front of camera)

**Right**

Epipolar Lines are Horizontal

**Left**

L[x,y] matches to R[x-d[x,y],y]

\[ d[x,y] \geq 0 \] is called the "disparity map"

\[ d[x,y] \] is inversely proportional to depth

**Right**

Epipolar Lines are Horizontal

---

How do you find the correct match?

- Smooth regions are ambiguous. Too many pixels on the right will look like a pixel on the left.

- Non-lambertian regions are ambiguous. The correct pixel on the right will not look like the pixel on the left.
How do you find the correct match?

- Smooth regions are ambiguous. Too many pixels on the right will look like a pixel on the left.
- Non-lambertian regions are ambiguous. The correct pixel on the right will not look like the pixel on the left.
- Occlusions: pixel on the left is NOT visible in the image on the right and vice versa.

What is the right answer?

d(x,y) = Value that the pixel (x,y) in the left image has moved by, even if it is not visible.

In other words, (x, d(x,y)) should be the co-ordinate of the projection of that 3D surface point in the right image.

Useful because we want to eventually use it to estimate depth.

Consider a neighborhood

- Smooth regions are ambiguous. Too many pixels on the right will look like a pixel on the left.
- Non-lambertian regions are ambiguous. The correct pixel on the right will not look like the pixel on the left.
- Occlusions: pixel on the left is NOT visible in the image on the right and vice versa.

Last time: Matching with Census Transform

Just one of many 'robust features' / strategies for local matching.

Let us say we believe max value of disparity is D=1

So by considering all possible matches, we are building a WeIRD "cost volume"
C(x,y,d) measures the quality of the match between L(x,y) and R(x,y,d).

Let us say we believe max value of disparity is 6-1.
So by considering all possible matches, we are building a "cost volume".

\[ d[x, y] = \arg \min_d C(x, y, d) \]

But that still gives us pretty noisy disparity maps.

We've seen noise before. Smoothing helps.
We could just smooth the disparity map?

The errors in the disparity map can often be high magnitude.
In a smooth region or with repeated texture, there may be a second seemingly good match very far away, and that's what \( \arg \min \) chooses.

\[ d[x, y] = \arg \min_d C(x, y, d) \]

But that still gives us pretty noisy disparity maps.

We've seen noise before. Smoothing helps.
We could just smooth the disparity map?

We want to express the fact that we expect our disparity map to be smooth.
But do it before we compute the \( \arg \min \) below.

\[ d[x, y] = \arg \min_d C(x, y, d) \]

But that still gives us pretty noisy disparity maps.

We've seen noise before. Smoothing helps.
We could just smooth the disparity map?
Possible Solution: Smooth the cost volume!

Take each slice of the cost volume, and smooth it. Expresses the fact that if \((x, y)\) and \((x+3, y)\) match, then so should \((x+1, y)\) with \((x+1, y+1)\); \((x+1, y)\) with \((x+1, y-1)\)

Take \(\arg\min_{\delta} \sum_{x, y} C[x, y, \delta]

\[d(x, y) = \arg\min_{\delta} C[x, y, \delta] \]

But that still gives us pretty noisy disparity maps

We've seen noise before. Smoothing helps. We could just smooth the disparity map?

Here, "blurrier" means the location of disparity discontinuities, i.e. the contours, get spread out. It does not cause a more gradual change in the disparities themselves.
Going back, we want to express the fact that our disparity map is smooth. Cost volume filtering is an ad-hoc way of doing that. Still making independent decisions at each pixel. Averaging each disparity level promotes disparity maps where values are “equal” not close. If $C(x, y, d)$ is a good match, then $C[x + 1, y, d ± 1]$ gets no benefit from filtering. Not good for slanted surfaces. Could be fixed by smoothing

$$\min_{\delta \in \{-1, 0, 1\}} C[x, y, d + \delta]$$

But generally, would prefer expressing this as optimizing a well-defined cost.

$d = \arg \min_d \sum_n C[n, d[n]] + \lambda \sum_{(n, n') \in E} S(d[n], d[n'])$

- $n = [x, y]^T$ for pixel location.
- $C$ is cost-volume as before. Gives us “local evidence”
- $E$ is a set of all pairs of pixels that are “neighbors” / adjacent in some way.
- Can include all un-ordered pairs of pixels with $[(x, y), (x - 1, y)]$ and $[(x, y), (x, y - 1)]$ (four connected)
- Or diagonal neighbors as well.
- $S$ is a function that indicates a preference for $d[n]$ and $d[n']$ to be the same.
GLOBAL OPTIMIZATION

\[ d = \arg \min_d \sum_n C(n, d[n]) + \lambda \sum_{(n, n') \in \mathcal{E}} S(d[n], d[n']) \]

- \( S \) is a function that indicates a preference for \( d[n] \) and \( d[n'] \) to be the same.

- Choice 1:
  - 0 if \( d[n'] = d[n] \), 1 otherwise.

- Choice 2: \(|d[n'] - d[n]|\)

- Choice 3:
  - 0 if \( d[n'] = d[n] \)
  - \( T_1 \) if \(|d[n'] - d[n]| < \epsilon \)
  - \( T_2 \) otherwise.

Note that this is a discrete minimization. Each \( d[n] \in \{0, 1, \ldots, D - 1\} \).

How do we solve this?

One approach: Iterated Conditional Modes

- Begin with \( d_0 = \arg \min_d C[n, d[n]] \)
- At each iteration \( t \), compute \( d_{t+1} \) from \( d_t \), by solving for each pixel in \( d_{t+1} \) assuming neighbors have values from \( d_t \).

\[ d_{t+1}[n] = \arg \min_{d_t} C[n, d_t] + \lambda \sum_{(n, n') \in \mathcal{E}_n} S(d_t[n], d_t[n']) \]

So for each pixel,

- Take matching cost.
- Add smoothness cost from its neighbors, assuming values from previous iteration.
- Minimize.

Does it converge?

To a global optimum?