RECTIFIED STEREO MATCHING

Summary

- We define a cost volume $C$ of size $W \times H \times D$
  - $C[x, y, d]$ measures dissimilarity between $(x, y)$ in left image and $(x - d, y)$ in right image
- Simplest Approach: $d[x, y] = \arg \min_d C[x, y, d]$
- Too noisy: want to express that disparity (and therefore depth) of nearby pixels is similar
- Ad-hoc Method: Cost Volume Filtering
- 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 $C[x + 1, y, d \pm 1]$ gets no benefit from filtering.
  - Not good for slanted surfaces.
- Want to do smoothing as optimization of a well defined regularization cost.

GLOBAL OPTIMIZATION

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

- $n = [x, y]^T$ 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

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

- \( S \) 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.

How do we solve this?

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

GLOBAL OPTIMIZATION

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 \( d_t \):

\[ d_{t+1}[n] = \arg \min_{d[n']} C[n, d[n']] + \lambda \sum_{(n, n') \in E_n} S(d[n], d[n']) \]

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

Does it converge?

No Guarantee: We are changing all pixel assignments simultaneously.

GLOBAL OPTIMIZATION

Per-pixel Iterated Conditional Modes (slow!)

- 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 one pixel in \( d_{t+1} \) assuming neighbors have values from \( d_t \):

\[ d_{t+1}[n+1] = \arg \min_{d[n]} C[n+1, d[n]] + \lambda \sum_{(n, n') \in E_{n+1}} S(d[n], d[n']) \]

Does it converge?

- Each iteration decreases the cost. So it will converge (but to a local optimum).

GLOBAL OPTIMIZATION

- These kind of cost functions / optimization problems are quite common in vision.
- The cost can be interpreted as a log probability distribution:

\[ p(d) \propto \prod_n \exp(-C[n, d[n]]) \prod_{(n, n') \in E} \exp(-\lambda S(d[n], d[n'])) \]

- Joint distribution over all the \( d[n] \) values.
**Joint distribution over all the values.**

**Graphical Model:** Probability Distribution Represented as a "Graph" $(V, E)$

\[
p(d) \propto \exp(-C[n, d[n]]) \prod_{(n, n') \in E} \exp(-\lambda S(d[n], d[n'])\}
\]

- Joint distribution over all the $d[n]$ values.
- **Graphical Model:** Probability Distribution Represented as a "Graph" $(V, E)$

\[
p(\{v \in V\}) = \prod_{v \in V} \Psi_v(v) \prod_{(v_1, v_2) \in E} \Phi_{v_1, v_2}(v_1, v_2)
\]

- Unary term for each node, pair-wise term for each edge.

(Directed Graphs represent Bayesian Networks)

**Question:** Are $d[n]$ and $d[n']$ independent if:

- If $(n, n') \in E$ – pixels are neighbors?

Reminder: Two variables are independent if we can express their joint distribution as a product of distributions on each variable.

**Question:** Are $d[n]$ and $d[n']$ independent if:

- If $(n, n') \in E$ – pixels are neighbors. **No**
- If $(n, n') \not\in E$ – pixels are not neighbors?
Question: Are \(d[n]\) and \(d[n']\) independent if:
- If \((n, n') \in E\) – pixels are neighbors. No
- If \((n, n') \notin E\) – pixels are not neighbors? NO. Unless \(n, n'\) are parts of disconnected components of graph.

Graph structure encodes conditional independence.

YES. This is the Markov property. And these kinds of graphical models are called Markov random fields.
Compute assignment with highest probability

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

Iterated Conditional Modes really slow.
No guaranteed solution for arbitrary graphs.
But could solve it if our graph were a chain (or more generally a tree).

Consider where we optimize each epipolar line separately.

\[ d = \arg \min_d \sum_x C[x, d[x]] + \sum_x S(d[x], d[x+1]) \]

- Consider where we optimize each epipolar line separately.
You have costs stored for each individual allocation

You have costs stored for each individual allocation, as well as cost for edges
\[ \sum_{x} C[x, d[x]] + \lambda \sum_{x} S(d[x], d[x + 1]) \]

Say we only had two nodes:

\[ \sum_{x} C[x, d[x]] + \lambda \sum_{x} S(d[x], d[x + 1]) \]

The total cost of those blocks and the edges was the least.
GLOBAL OPTIMIZATION

Say we only had two nodes:

\[ d_1, d_2 = \arg \min_{d_1, d_2} C[1, d_1] + C[2, d_2] + \lambda S(d_1, d_2) \]

Then:

\[ d_2 = \arg \min_{d_2} C[2, d_2] + \min_{d_1} (C[1, d_1] + \lambda S(d_1, d_2)) \]

\[ \sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1]) \]

GLOBAL OPTIMIZATION

Say we only had two nodes:

\[ d_1, d_2 = \arg \min_{d_1, d_2} C[1, d_1] + C[2, d_2] + \lambda S(d_1, d_2) \]

\[ d_2 = \arg \min_{d_2} C[2, d_2] + \min_{d_1} (C[1, d_1] + \lambda S(d_1, d_2)) \]

This is the \(d_2\) corresponding to the optimal path.

\[ \sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1]) \]
\[ d_1, d_2, d_3 = \arg \min \; C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3) \]

\[ d_3 = \arg \min \; C[3, d_2] + \min \; \frac{\lambda S(d_2, d_3) + C[2, d_2] + \min \; [\lambda S(d_1, d_2) + C[1, d_1]]}{d_1} \]

This is precisely what we computed for the 2 node case.

\[ \sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x+1]) \]

\[ d_1, d_2, d_3 = \arg \min \; C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3) \]

\[ d_3 = \arg \min \; C[3, d_2] + \min \; \frac{\lambda S(d_2, d_3) + C[2, d_2] + \min \; [\lambda S(d_1, d_2) + C[1, d_1]]}{d_1} \]

This is precisely what we computed for the 2 node case.

Also note that once you have this, you don’t care about what the value of \( d_i \) was in the inner minimization.

\[ \sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x+1]) \]
GLOBAL OPTIMIZATION

\[ d_1, d_2, d_3 = \arg \min C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3) \]

\[ d_3 = \arg \min_{d_3} C[3, d_3] + \min_{d_2} \left[ \lambda S(d_2, d_3) + C[2, d_2] + \min_{d_1} [\lambda S(d_1, d_2) + C[1, d_1]] \right] \]

\[ \overline{C}[2, \cdot] \]

\[ \overline{C}[3, \cdot] \]

\[ \overline{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \overline{C}[x, d'] \]

GLOBAL OPTIMIZATION

We go from left to right, and doing an arg min on the last \( C \) gives us the disparity of the last node.

GLOBAL OPTIMIZATION

\[ d_1, d_2, d_3 = \arg \min C[1, d_1] + C[2, d_2] + C[3, d_3] + \lambda S(d_1, d_2) + \lambda S(d_2, d_3) \]

\[ d_3 = \arg \min_{d_3} C[3, d_3] + \min_{d_2} \left[ \lambda S(d_2, d_3) + C[2, d_2] + \min_{d_1} [\lambda S(d_1, d_2) + C[1, d_1]] \right] \]

\[ \overline{C}[2, \cdot] \]

\[ \overline{C}[3, \cdot] \]

\[ \overline{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \overline{C}[x, d'] \]

GLOBAL OPTIMIZATION

We go from left to right, and doing an arg min on the last \( C \) gives us the disparity of the last node.

And then we backtrack to find the full chain.

\[ z[x + 1, d] = \arg \min_{d'} \lambda S(d, d') + \overline{C}[x, d'] \]

\[ \overline{C}[x + 1, d] = C[x + 1, d] + \min_{d'} \lambda S(d, d') + \overline{C}[x, d'] \]
We could apply this on individual epipolar lines.

Get “streaking” artifacts. Because we’re smoothing each line independently.

That’s why we want to use a full 2D grid.
But forward-backward only works on chains (or graphs without cycles).

One flavor of approximate algorithms apply the same idea of forming a $\hat{C}[x, d]$

- TRW-S
- Loopy Belief Propagation
- SGM
Semi-Global Matching

\[ C[x, d] = C[x, d] + \min_{d'} C[x - 1, d'] + \lambda S(d, d') \]

This is going left to right in the horizontal direction.

Idea: Compute different \( \tilde{C} \) along different directions … and average.

Semi-Global Matching

\[ \tilde{C}_v[n, d] = C[n, d] + \min_{d'} \tilde{C}_v[n - [1, 0]^T, d'] + \lambda S(d, d') \]

\[ \tilde{C}_h[n, d] = C[n, d] + \min_{d'} \tilde{C}_h[n + [1, 0]^T, d'] + \lambda S(d, d') \]

\[ \tilde{C}_{da}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{da}[n - [0, 1]^T, d'] + \lambda S(d, d') \]

\[ \tilde{C}_{aw}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{aw}[n + [0, 1]^T, d'] + \lambda S(d, d') \]

\[ d[n] = \arg \min_d \tilde{C}_v[n, d] + \tilde{C}_h[n, d] + \tilde{C}_{aw}[n, d] + \tilde{C}_{da}[n, d] \]

Consider the case when:

\[ 0 \text{ if } d = d' \]

\[ P_1 \text{ if } |d - d'| = 1 \]

\[ P_2 \text{ otherwise.} \]

Can we do this efficiently?

- Need to go through each line sequentially.
- But can go through all lines in parallel.
- But what about \( d \)? Do we need to do minimization for every \( d \) independently?