GENERAL

- No Class Tuesday (Fall Break)
- No office hours on Monday.
- There’s an incompatibility with the latest matplotlib in the support code in Problem 2 in PSET 3.
  - See Piazza post. You need to comment out lines saying `ax.hold()`
- Project Proposals Due next Thursday!

RECTIFIED STEREO MATCHING

Last Time

- 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.
  - Could be fixed by smoothing $\min_{\delta = (-1, 1)} C[x, y, d + \delta]$
- But generally, would prefer expressing this as optimizing a well-defined cost.

GLOBAL OPTIMIZATION

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

- $n = [x, y]^T$ for pixel location.
- $C$ is cost-volume as before. Gives us “local evidence”
- $\mathcal{E}$ 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**

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

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 \( d_t \).

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

- 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**

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

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 \( d_t \).

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

Does it converge?

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

**GLOBAL OPTIMIZATION**

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

- 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.
GLOBAL OPTIMIZATION

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:

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

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

GLOBAL OPTIMIZATION

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

Question: Are \( d[n] \) and \( d[n'] \) independent if:

- if \((n, n') \in E\) -- pixels are neighbors

- if \((n, n') \notin E\) -- pixels are not neighbors?

GLOBAL OPTIMIZATION

Question: Are \( d[n] \) and \( d[n'] \) independent if:

- \((n, n') \in E\) -- pixels are neighbors

- \((n, n') \notin E\) -- pixels are not neighbors?

GLOBAL OPTIMIZATION

Question: Are \( d[n] \) and \( d[n'] \) independent if:

- if \((n, n') \in E\) -- pixels are neighbors

- if \((n, n') \notin E\) -- pixels are not neighbors?

Unless \( n, n' \) are parts of disconnected components of graph.
Question: Are $d[n]$ and $d[n']$ independent if:

If $(n, n') \notin E$, "conditioned" on all the neighbors of $n$ being observed, $p(d[n], d[n'][|d[n']])$

YES. This is the Markov property. And these kinds of graphical models are called Markov random fields.

Graph structure encodes "conditional independence".

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

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.
You have costs stored for each individual allocation, as well as cost for edges

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

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

Say we only had two nodes:

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

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

This is the \( d_1 \) corresponding to the optimal path

\[ \sum_x C[x, d[x]] + \lambda \sum_x S(d[x], d[x + 1]) \]
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 \( d \) was in the inner minimization.

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

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.

Store best \( d' \) for each \( d \).

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

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

Forward

\[
\bar{C}[0, d] = C[0, d]
\]

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

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

Backward

\[
d[x_{end}] = \arg \min_d \bar{C}[x_{end}, d]
\]

\[
d[x] = z[x + 1, d[x + 1]]
\]
GLOBAL OPTIMIZATION

We could apply this on individual epipolar lines.

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 $\tilde{C}[x, d]$

- TRW-S
- Loopy Belief Propagation
- SGM

GLOBAL OPTIMIZATION

Semi-Global Matching

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

This is going left to right in the horizontal direction.

Idea: Compute different $\tilde{C}$ along different directions ...

and average.

GLOBAL OPTIMIZATION

Semi-Global Matching

$\tilde{C}_{lr}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{lr}[n-1, 0, d'] + \lambda S(d, d')$

$\tilde{C}_{rl}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{rl}[n+1, 0, d'] + \lambda S(d, d')$

$\tilde{C}_{du}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{du}[n, -1, 0, d'] + \lambda S(d, d')$

$\tilde{C}_{ud}[n, d] = C[n, d] + \min_{d'} \tilde{C}_{ud}[n, 1, 0, d'] + \lambda S(d, d')$

$d[n] = \arg\min_{d'} \tilde{C}_{lr}[n, d] + \tilde{C}_{rl}[n, d] + \tilde{C}_{ud}[n, d] + \tilde{C}_{du}[n, d]$
GLOBAL OPTIMIZATION

Semi-Global Matching

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

- Consider the case when \( S(d, d') \):
  - 0 if \( d = d' \)
  - \( P_1 \) if \( |d - d'| = 1 \)
  - \( P_2 \) 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?

Note: It doesn't matter if we add / subtract constants to all \( d \)’s:
- \( C[x, d] \) with \( C[x, d] + C_0[x] \)
- \( \tilde{C}[x, d] \) with \( \tilde{C}[x, d] + C_0[x] \)

Why not?
- Because the minimization will always be over \( d \). You are never comparing \( C[x_1, d_1] \) with \( C[x_2, d_2] \).
GLOBAL OPTIMIZATION

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

\[ S(d, d') = \begin{cases} 0 & \text{if } d = d' \\ P_1 & \text{if } |d - d'| = 1 \\ P_2 & \text{otherwise} \end{cases} \]

- Step 1 (Simplify): Replace \( \tilde{C}[x - 1, d'] \) with \( \tilde{C}[x - 1, d'] = \tilde{C}[x - 1, d'] - \min_{d'} \tilde{C}[x - 1, d''] \)
  
  What happens then?

What is the MAXIMUM value for \( \min_{d'} \tilde{C}[x - 1, d'] + S(d, d') \) for any \( d \)?

Can do this in parallel with matrix operations for all \( d \) and all lines.

GLOBAL OPTIMIZATION

SGM Algorithm Averages along four directions:

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

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

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

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

Wouldn't this be better?

But then...

GLOBAL OPTIMIZATION

Loopy Belief Propagation (one version)

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

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

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

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

Do this iteratively

More generally, at time step \(t\), pass a message from node \(n\) to \(n'\), based on all messages \(n\) has at that time, except for the message from \(n'\).

Read more:

GLOBAL OPTIMIZATION

- Other methods for discrete minimization—based on "Graph Cuts".
- SGM / Loopy BP: Generalize that there is an exact solution for a chain.
- Graph Cuts (with expansions / swaps): Generalize that there is an exact solution if only two values of \(d\).


http://vision.middlebury.edu/stereo/