CSCI 4830 / 5722
Computer Vision

CSCI 4830 / 5722
Computer Vision

Dr. Ioana Fleming
Spring 2019
Lecture 24

Reminders

Submissions:

- Homework 4: Wed 3/20 at 11 pm

Readings:

- Szeliski:
  - chapter 11.5.1 (Dynamic Programming)
- P&F:
  - chapter 7.5 (Dynamic Programming)

Next subject: Segmentation

Dynamic Programming

- A technique for designing (optimizing) algorithms
- It can be applied to problems that can be decomposed in subproblems, but these subproblems overlap.
- Instead of solving the same subproblems repeatedly, applying dynamic programming techniques helps to solve each subproblem just once.

Dynamic Programming

- Like divide and conquer, DP solves problems by combining solutions to subproblems.
- Unlike divide and conquer, subproblems are not independent.
  - Subproblems may share subsubproblems,
  - However, solution to one subproblem may not affect the solutions to other subproblems of the same problem. (More on this later.)
- DP reduces computation by
  - Solving subproblems in a bottom-up fashion.
  - Storing solution to a subproblem the first time it is solved.
  - Looking up the solution when subproblem is encountered again.
- Key: determine structure of optimal solutions

Dynamic Programming Examples

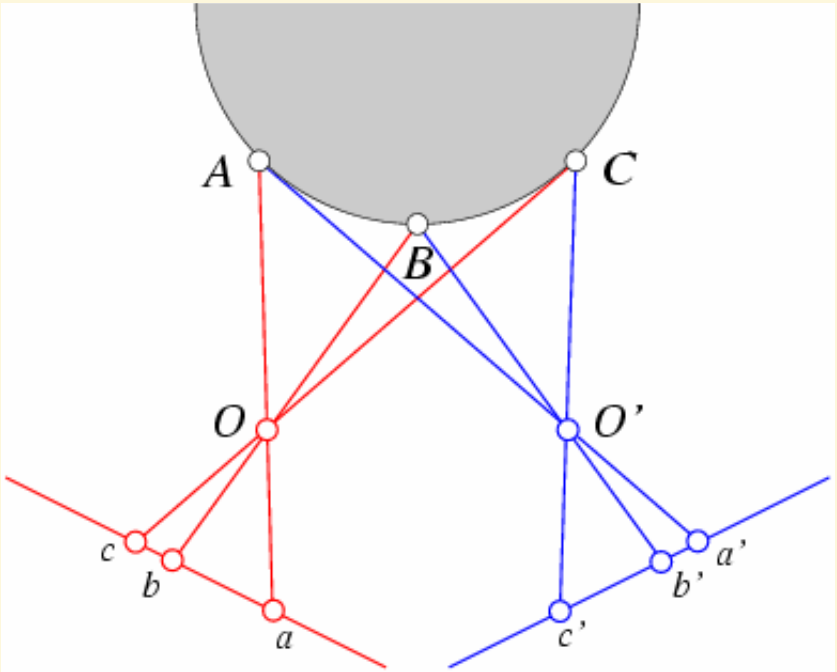
- String Alignment
- Binomial Coefficients
- The Integer Exact Knapsack problem
- Fibonacci numbers
- Longest Common Subsequence
- Commuters – intersection delays
- Stereo matching
- Other applications

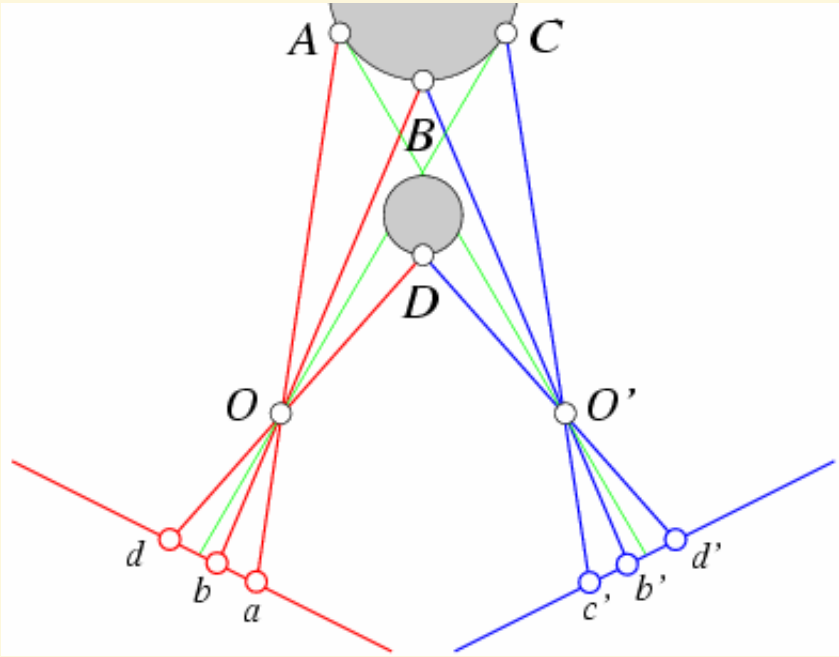
DP for stereo matching

Constraints:

- epipolar
- ordering
- uniqueness
- disparity smoothness

Ordering constraint in stereo

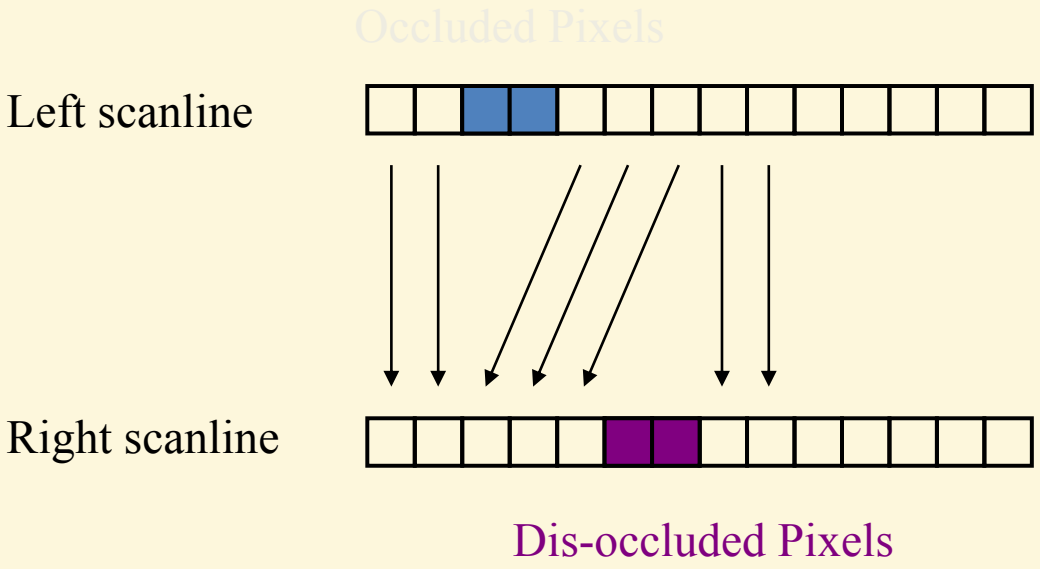




In general the points are in the same order on both epipolar lines.

But it is not always the case..

Search Over Correspondences



Three cases:

- Sequential – cost of match
- Occluded – cost of no match
- Disoccluded – cost of no match

Search Over Correspondences

$I_l(i)$ and $I_r(j)$, $1 \le i, j \le N$, where N is the number of pixels in each line.

Let d_{ij} be the cost associated with matching pixel $I_l(i)$ with pixel $I_r(j)$.

$$d_{ij} = (I_l(i) - I_r(j))^2$$

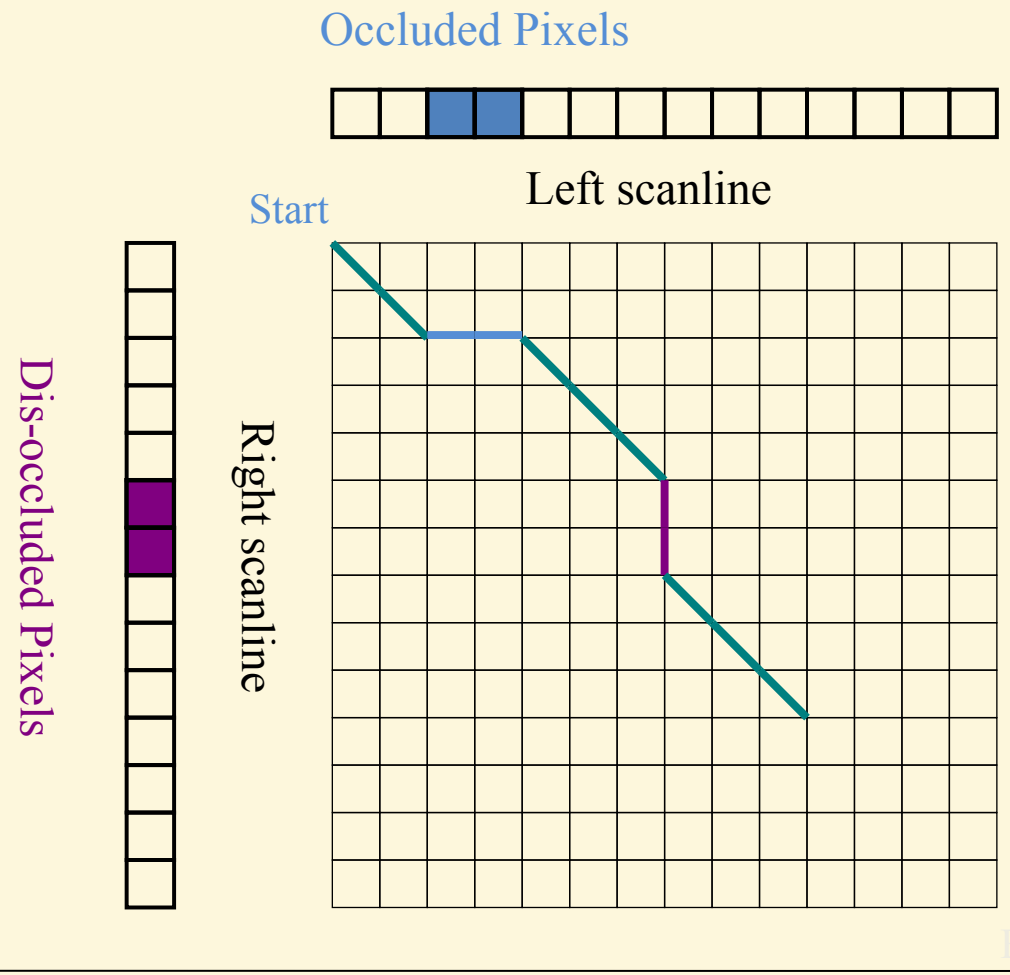
The cost of skipping a pixel (in either scanline) is given by a constant `occ`.

Optimal (minimal cost) alignment of two scalines (recursively):

1. `D(0,j) = j * occ`
2. `D(i,0) = i * occ`
2. $D(1, 1) = d_{11}$,
3. `D(i, j) =`

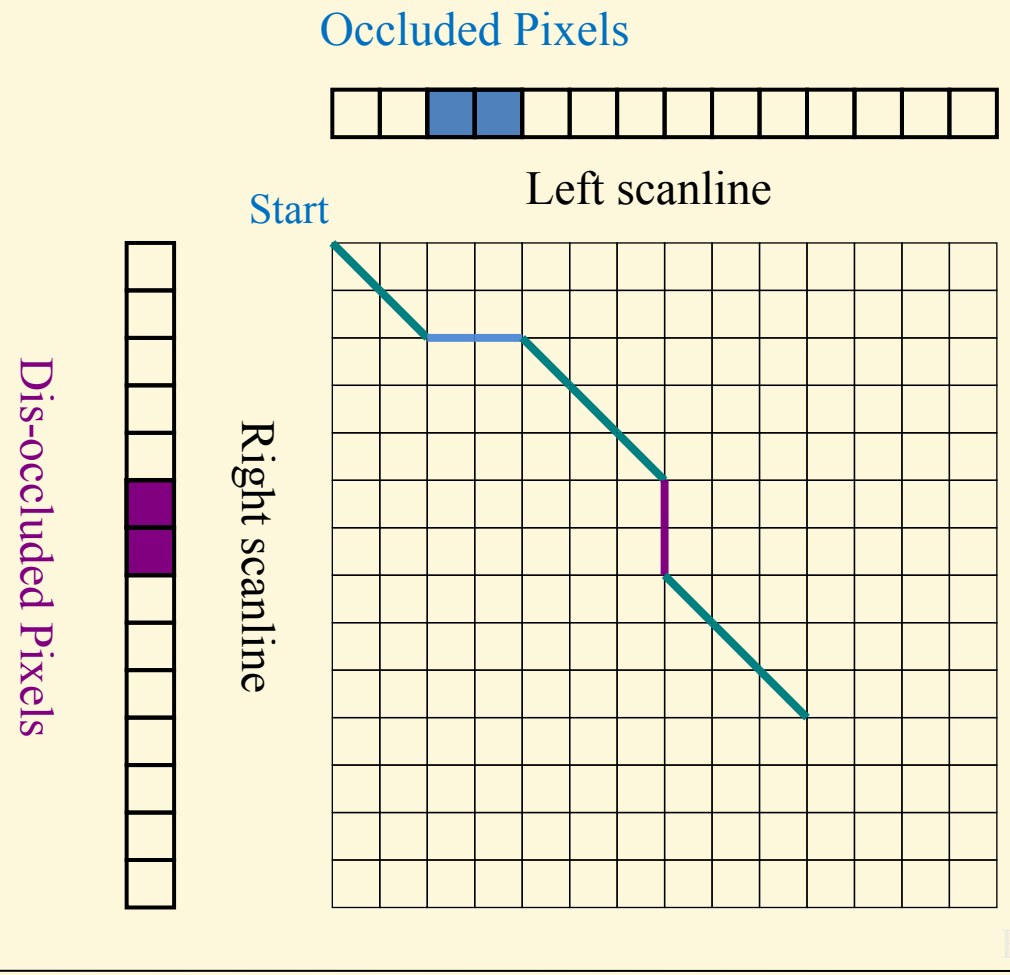
 $= \min(D(i-1,j-1) + d_{ij},\ D(i-1,j) + occ,\ D(i,j-1) + occ)$

Stereo Matching with DP



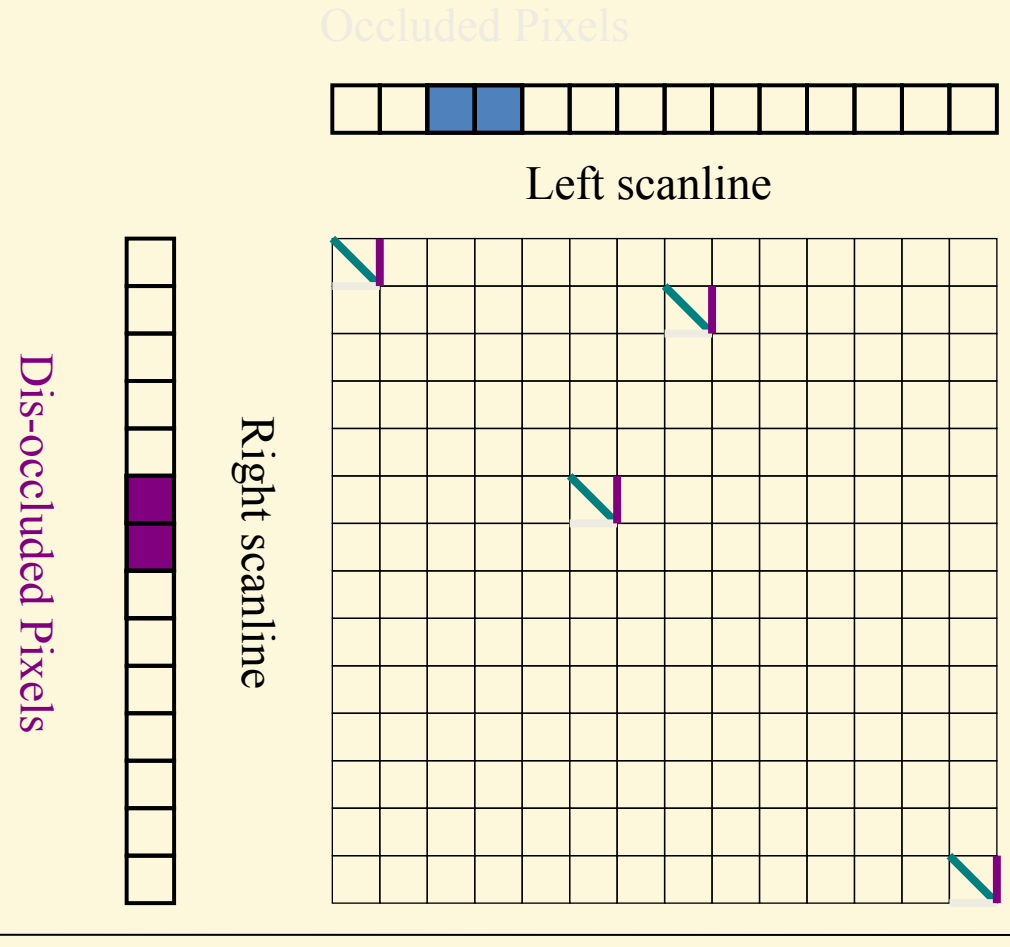
Dynamic programming yields the optimal path through grid. This is the best set of matches that satisfy the ordering constraint

Stereo Matching with DP



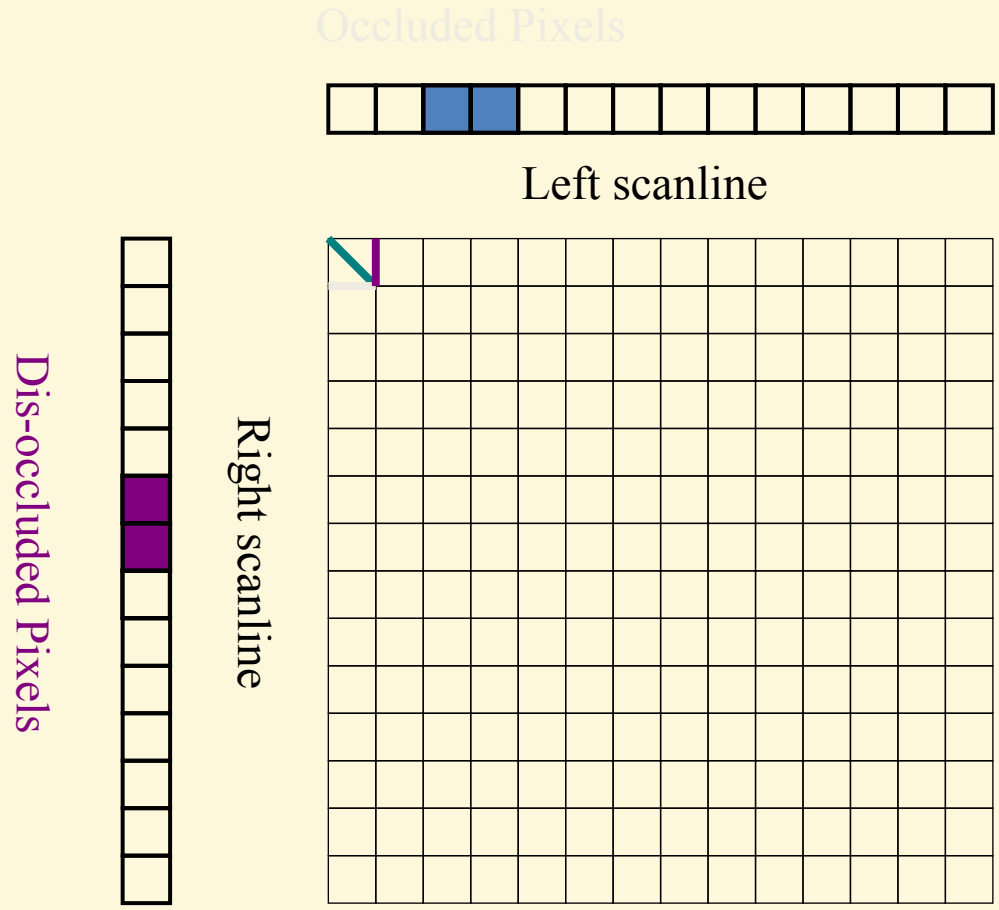
- Enforces ordering constraint.
- Given appropriate cost functions, solves for best path (matches, occlusions, disocclusions).

Stereo Matching with Dynamic Programming



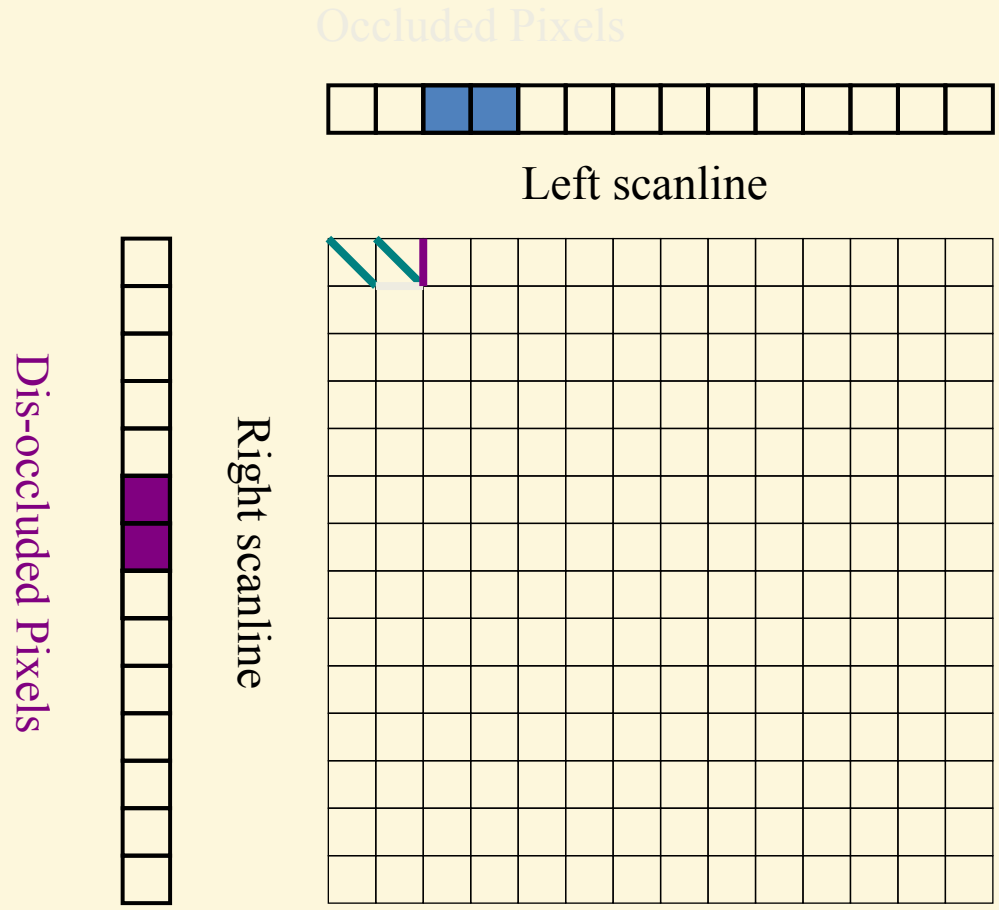
Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo Matching with Dynamic Programming



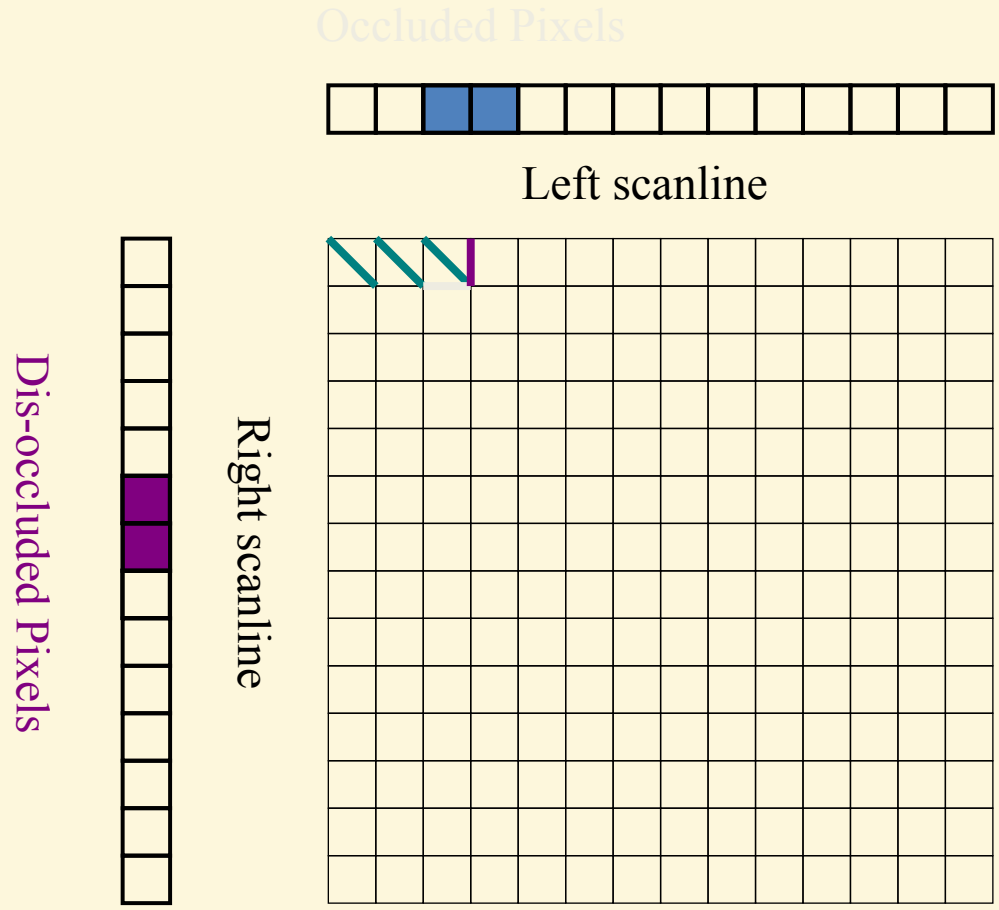
Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo Matching with Dynamic Programming



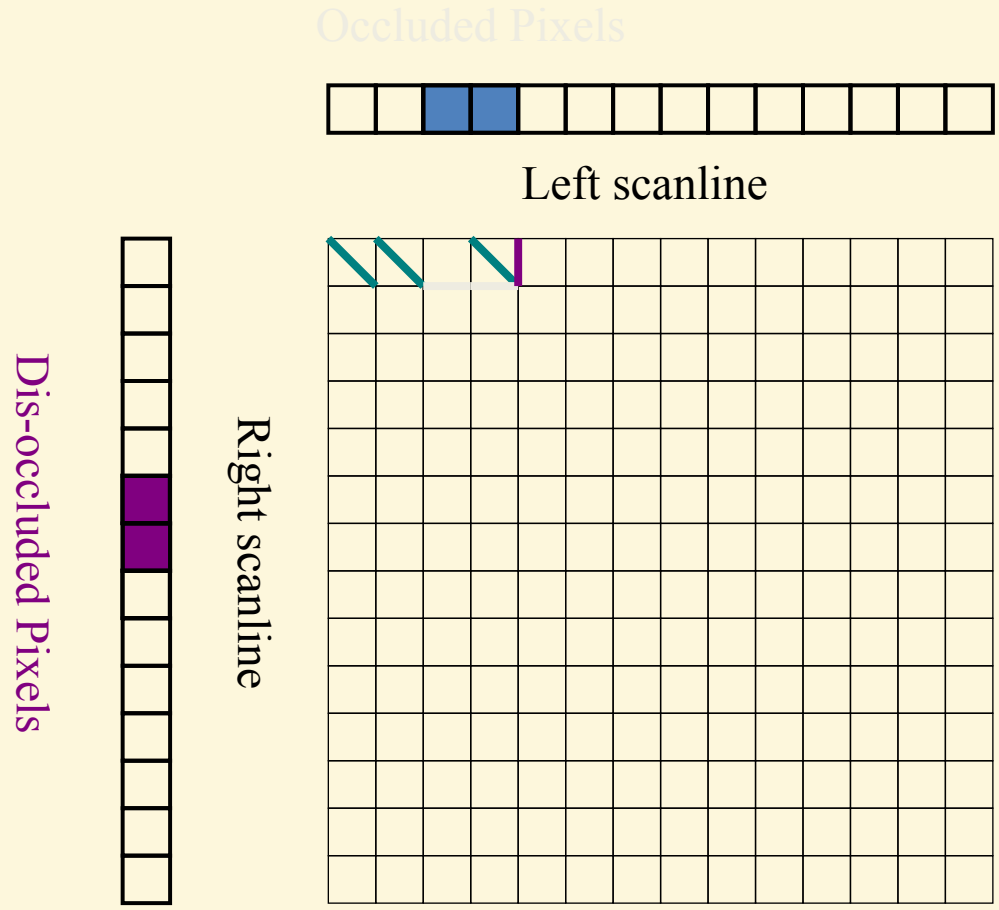
Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo Matching with Dynamic Programming



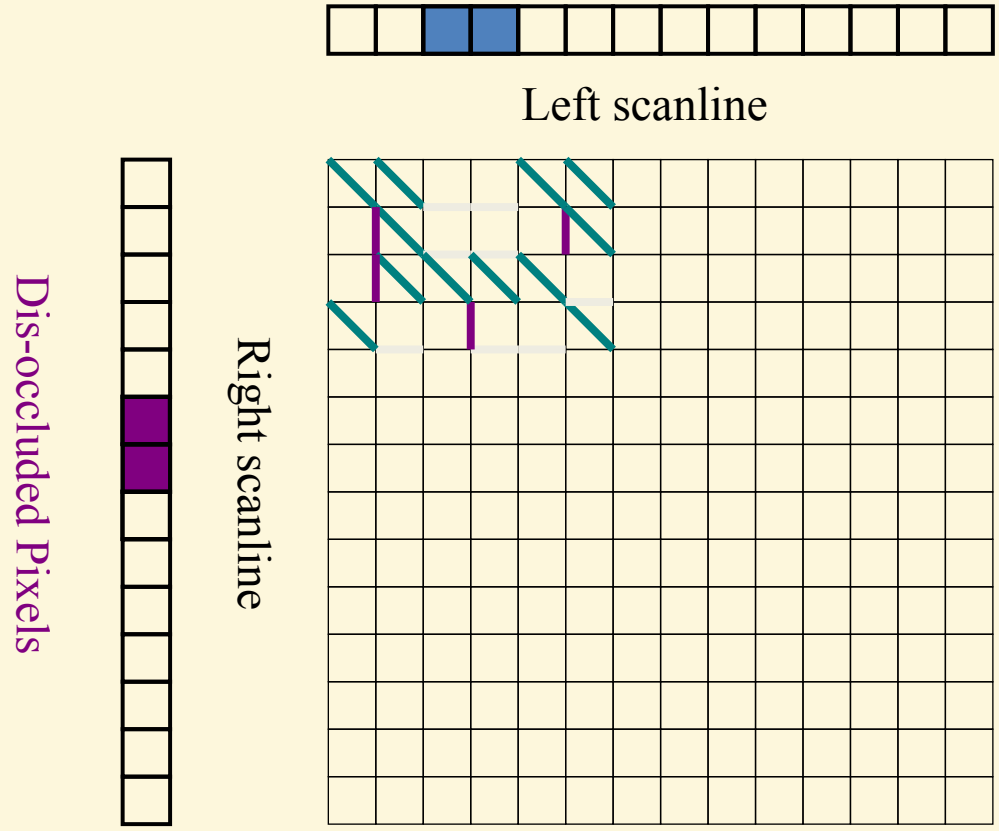
Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo Matching with Dynamic Programming



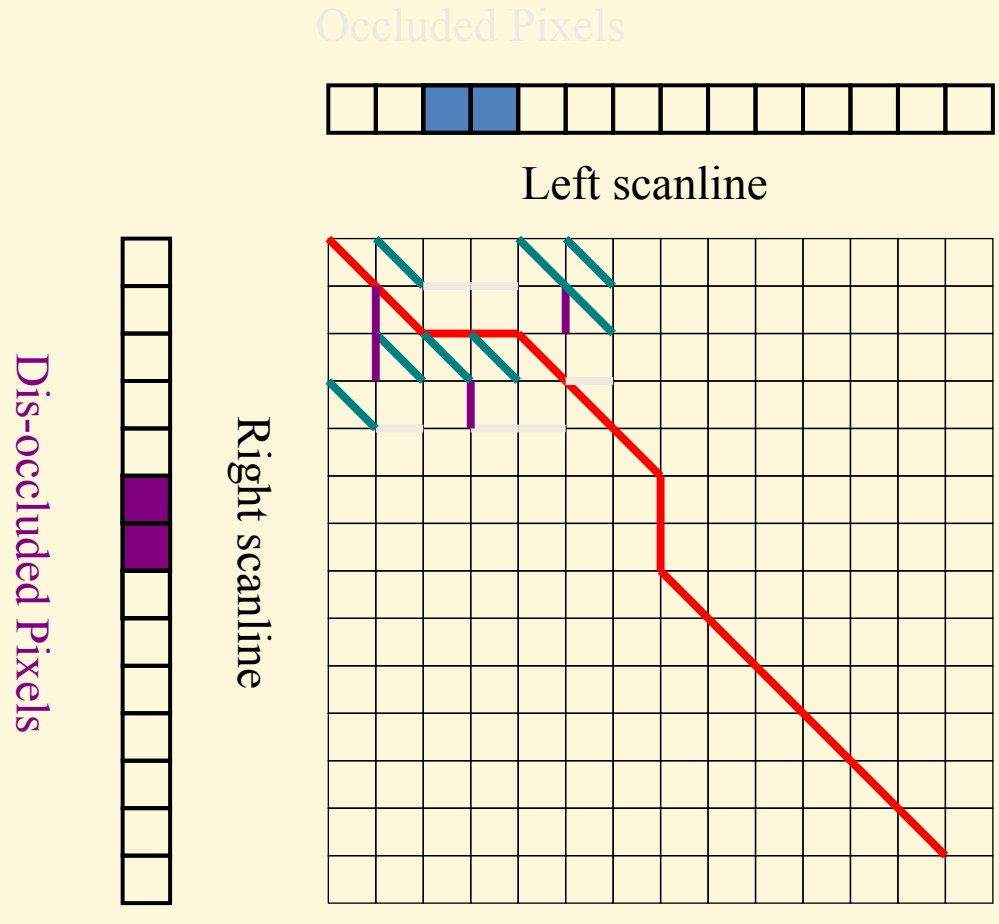
Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

Stereo Matching with Dynamic Programming



Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

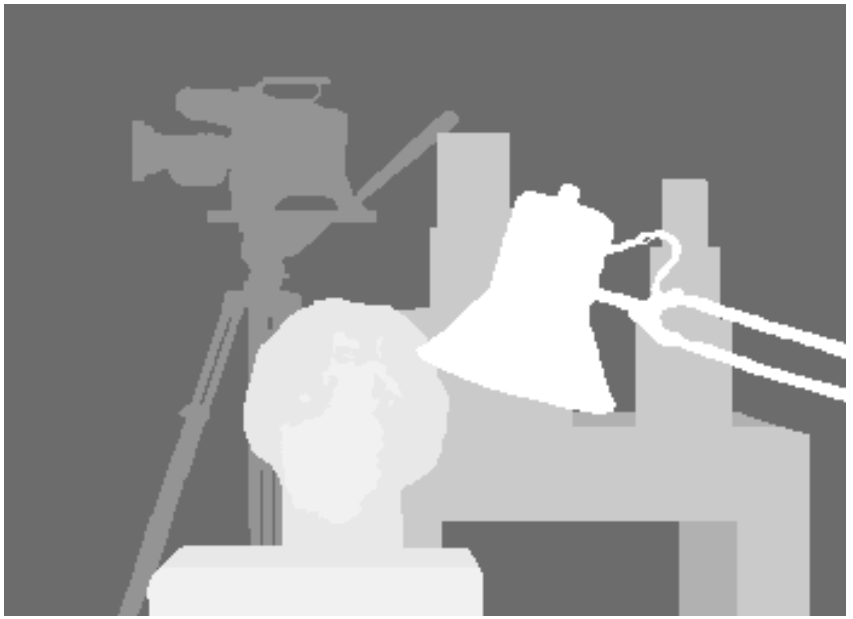
Stereo Matching with Dynamic Programming



Scan across grid computing optimal cost for each node given its upper left neighbors.
Backtrack from the terminal to get the optimal path.

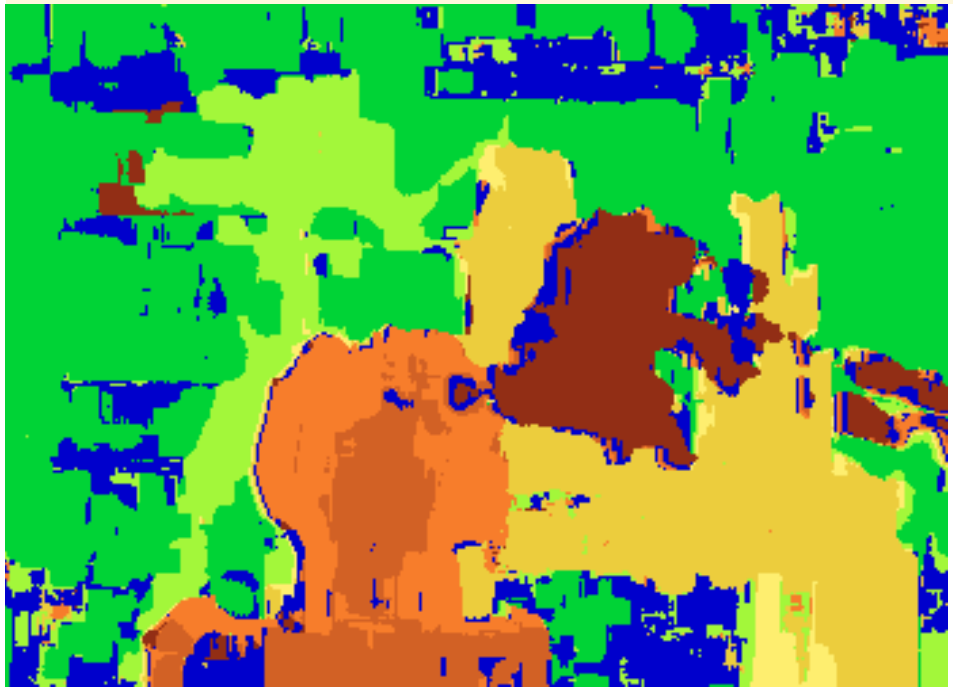
Data from University of Tsukuba

Scene



Ground truth

Results with window correlation



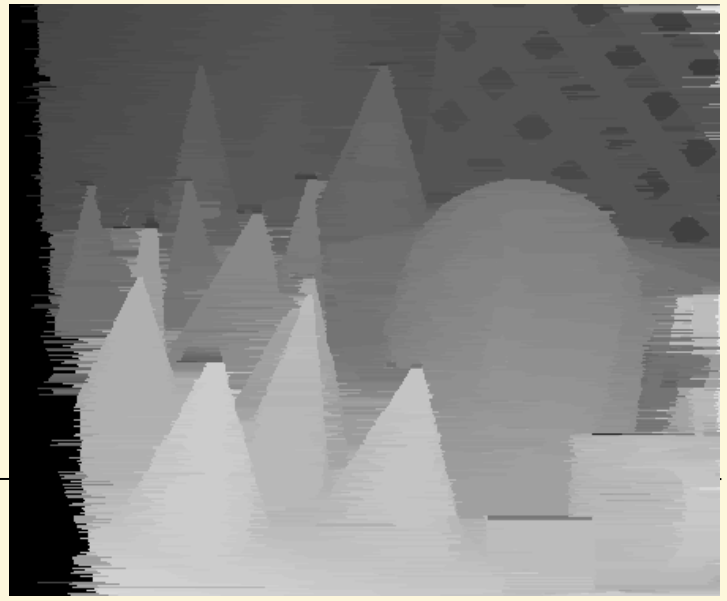
Window-based matching
(best window size)

Ground truth

Results with DP

- Much better results already!
- But no consistency between scanlines!

Dynamic Programming Results



Computing Correspondence

- Another approach is to match *edges* rather than windows of pixels:

- Which method is better?
  - Edges tend to fail in dense texture (outdoors)
  - Correlation tends to fail in smooth featureless areas