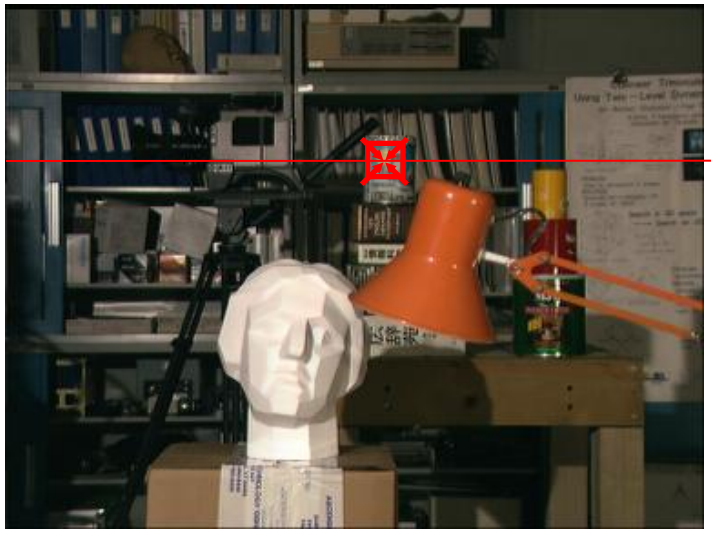
PatchMatch Stereo – Stereo Matching with Slanted Support Windows

Michael Bleyer (TU Vienna)
Christoph Rhemann (TU Vienna)
Carsten Rother (Microsoft Research Cambridge)

BMVC 2011

Local Stereo Algorithms



Left image

Right image

Local Stereo Algorithms

Implicit smoothness assumption:

- All pixels within support window have the **same integer-valued** disparity

Problems:

- Sub-pixel disparities
- Slanted surfaces
- Disparity discontinuities

Local Stereo Algorithms

Implicit smoothness assumption:

- All pixels within support window have the **same integer-valued** disparity

Problems:

- Sub-pixel disparities
- Slanted surfaces
- Disparity discontinuities ← We use adaptive Support Weights [Yoon, CVPR05]

Local Stereo Algorithms

Implicit smoothness assumption:

- All pixels within support window have the **same integer-valued** disparity

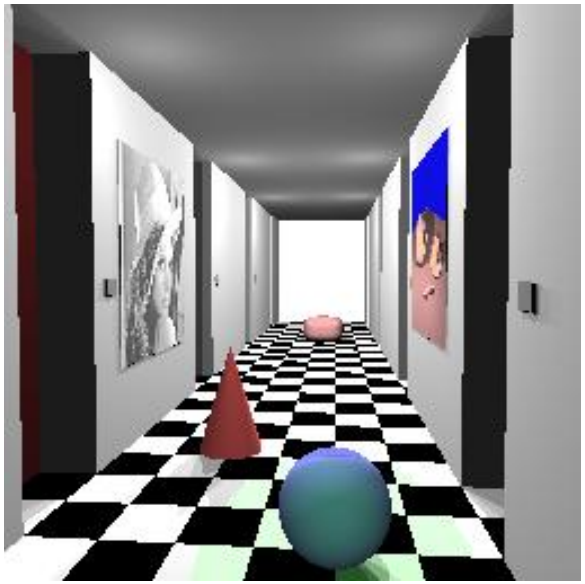
Problems:

- Sub-pixel disparities
- Slanted surfaces
- Disparity discontinuities

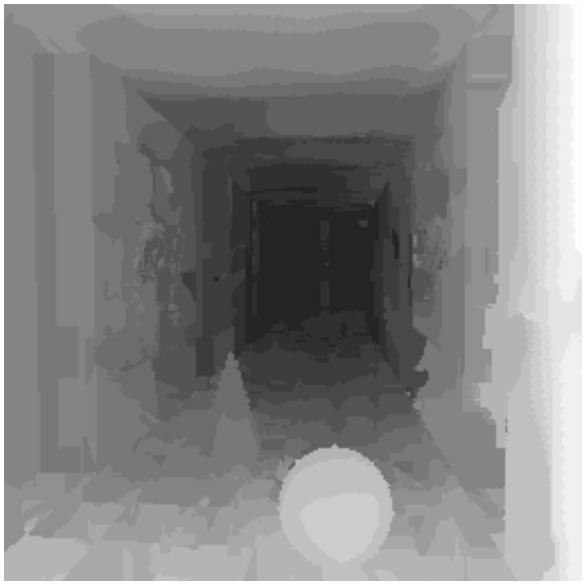
Focus of our work

We use adaptive Support Weights [Yoon, CVPR05]

Fronto-Par. Windows Matched at Integer Disparities



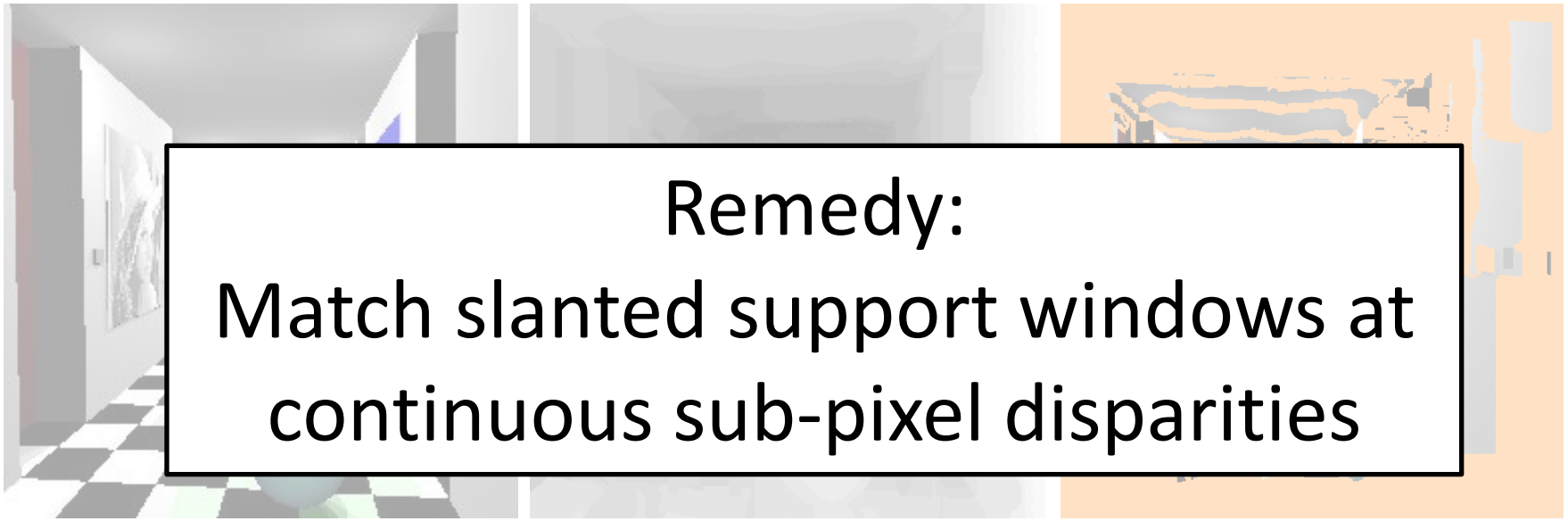
Left image



Disparity map

3D reconstruction

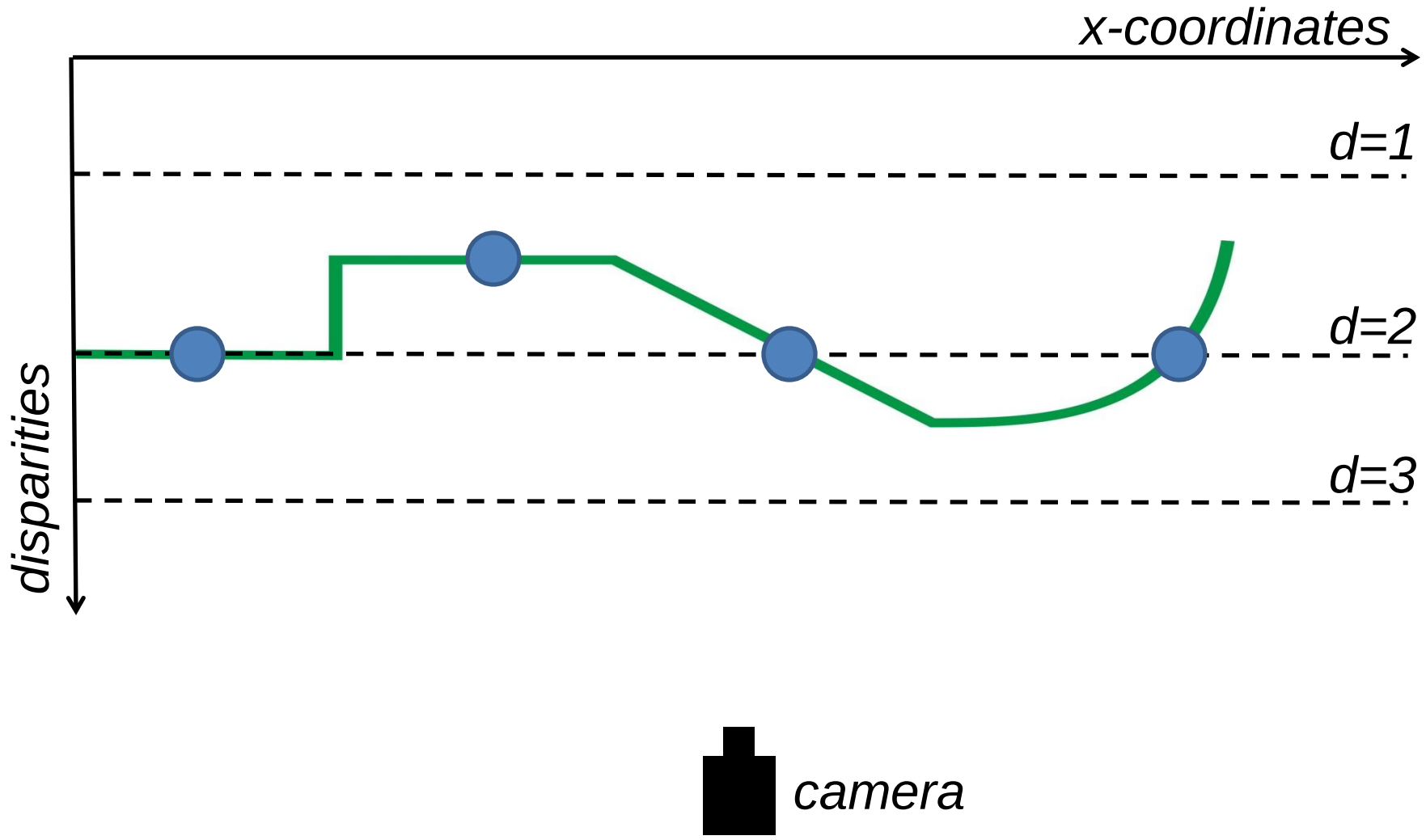
Fronto-Par. Windows Matched at Integer Disparities



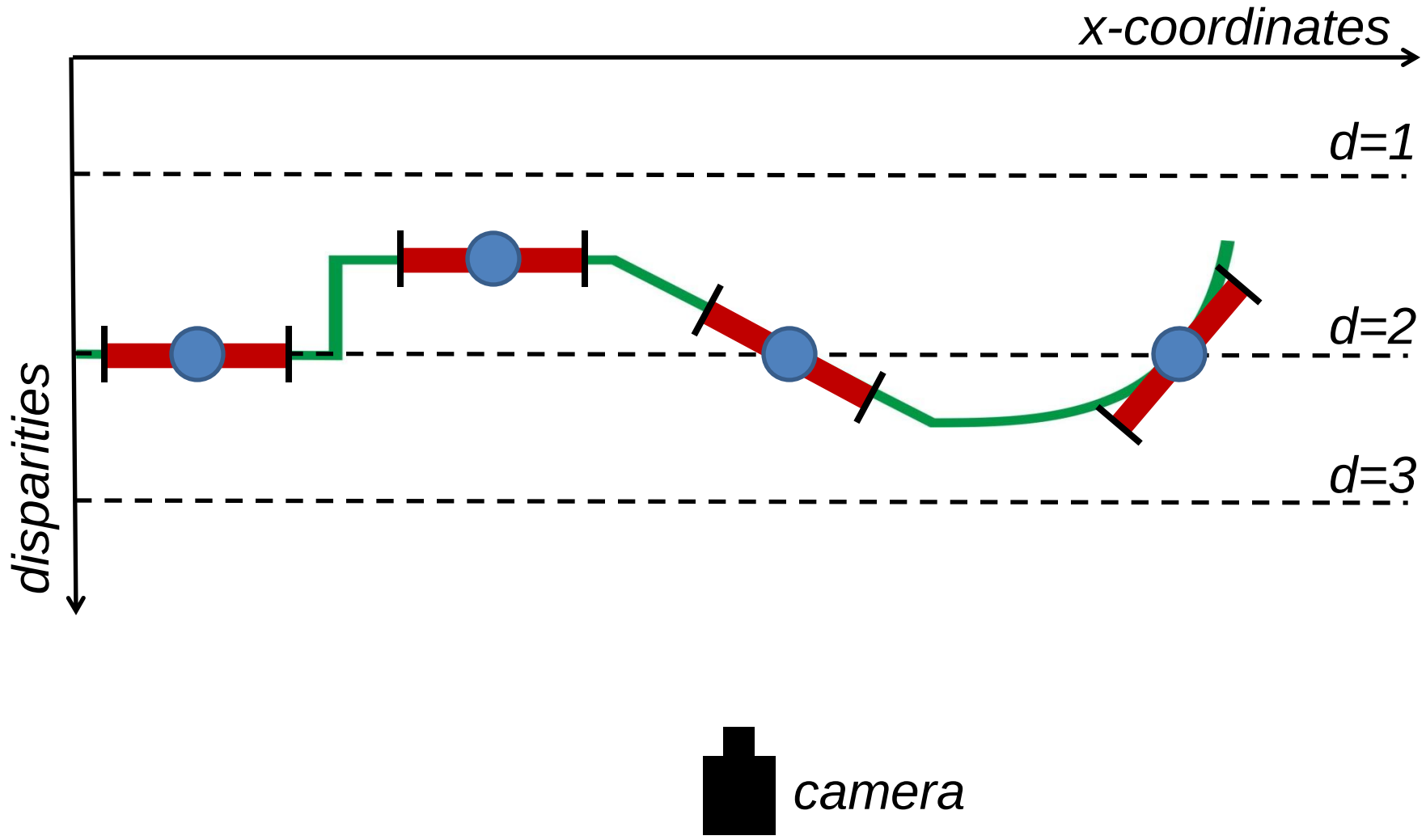
Remedy:
Match slanted support windows at continuous sub-pixel disparities

Left image | Disparity map | 3D reconstruction

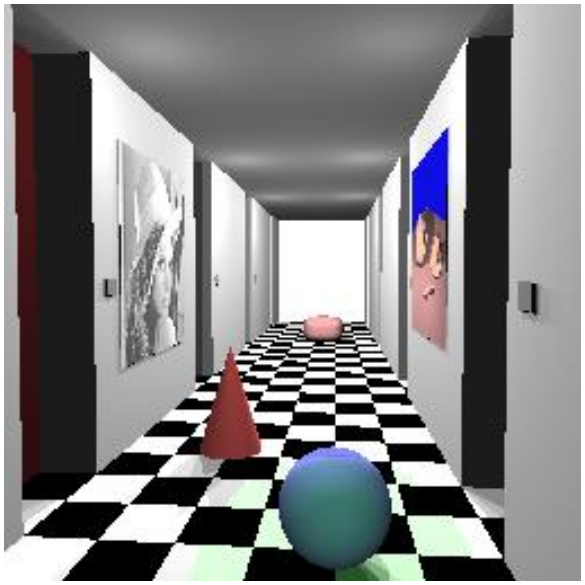
Slanted Support Windows



Slanted Support Windows

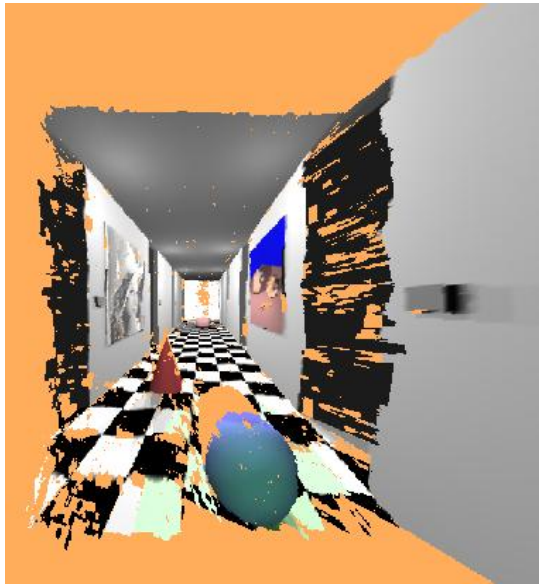


Our Slanted Windows



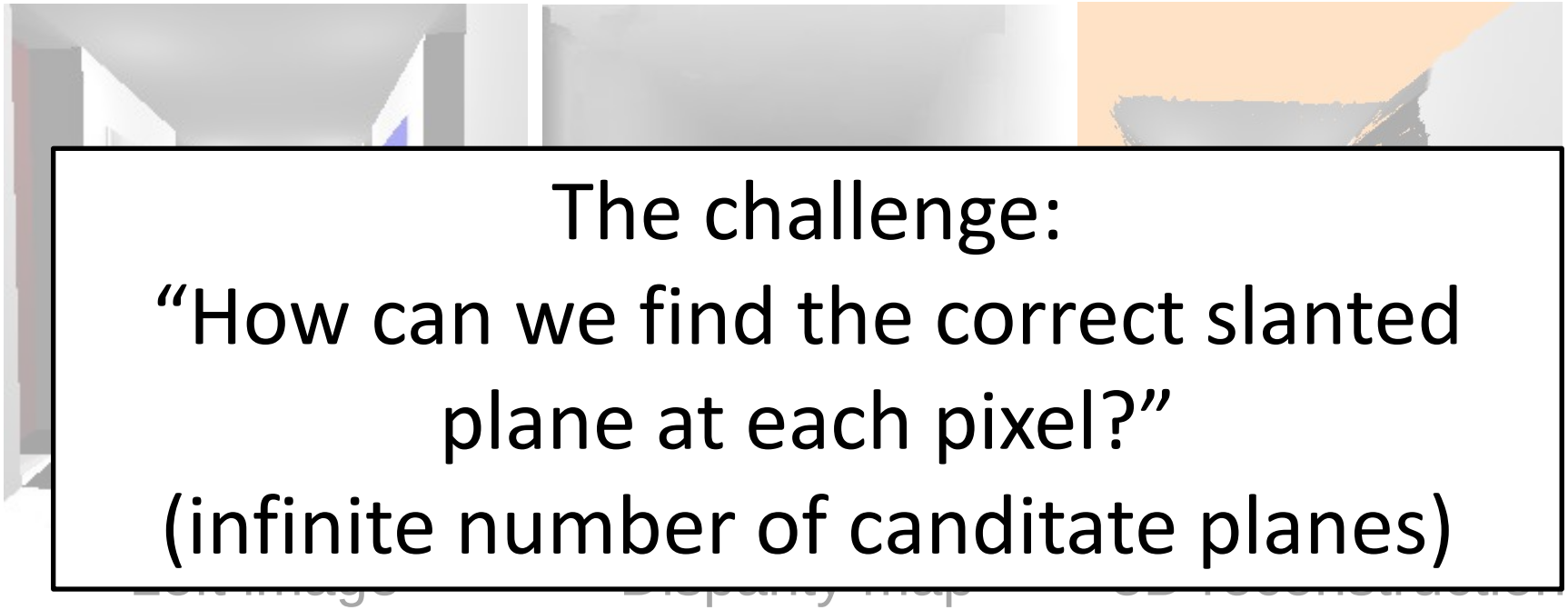
Left image

Disparity map



3D reconstruction

Our Slanted Windows



The challenge:
"How can we find the correct slanted plane at each pixel?"
(infinite number of canditate planes)

Previous Work

- Sub-pixel precision and slanted windows via extending label space:
  - Include fronto-parallel planes at sub-pixel disparities
  - Include slanted planes (plane sweeping)
- Problems
  - Each additional label adds extra computational complexity
  - Algorithm fails if correct plane not part of label space

Previous Work

- S
  - ex
  - P
    - Algorithm fails if correct plane not part of label space

Our algorithm:

- Continuous optimization over the set of all planes
- No quantization needed
- Based on PatchMatch [Barnes et al., SIGGRAPH09]

Our Algorithm

Basic Idea

- Relatively large regions of pixels can be modeled by approximately the same plane.
- Initialize each pixel with a random plane
- At least one pixel of a region should carry a plane close to correct one:
  - Many guesses (one per pixel)
- A single good guess is enough - it will be propagated to neighboring pixels

Left image – Sawtooth (Middlebury)

Ground truth disparities

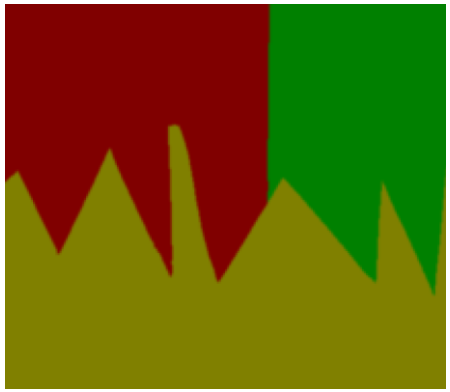
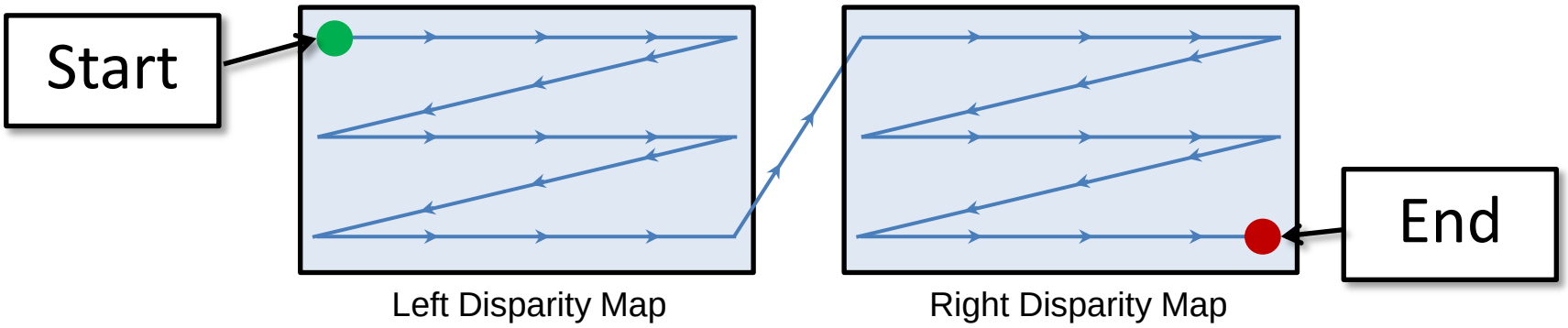


Image consists of 3 planes - ~80.000 guesses for yellow plane

Algorithm

- After random initialization process pixels in the following order:
  - Even iterations:



  - Odd iterations:
    - Reverse order
- We run 3 iterations

Algorithm

- After random initialization process pixels in the following order:

Run the following pipeline for each pixel *p*:

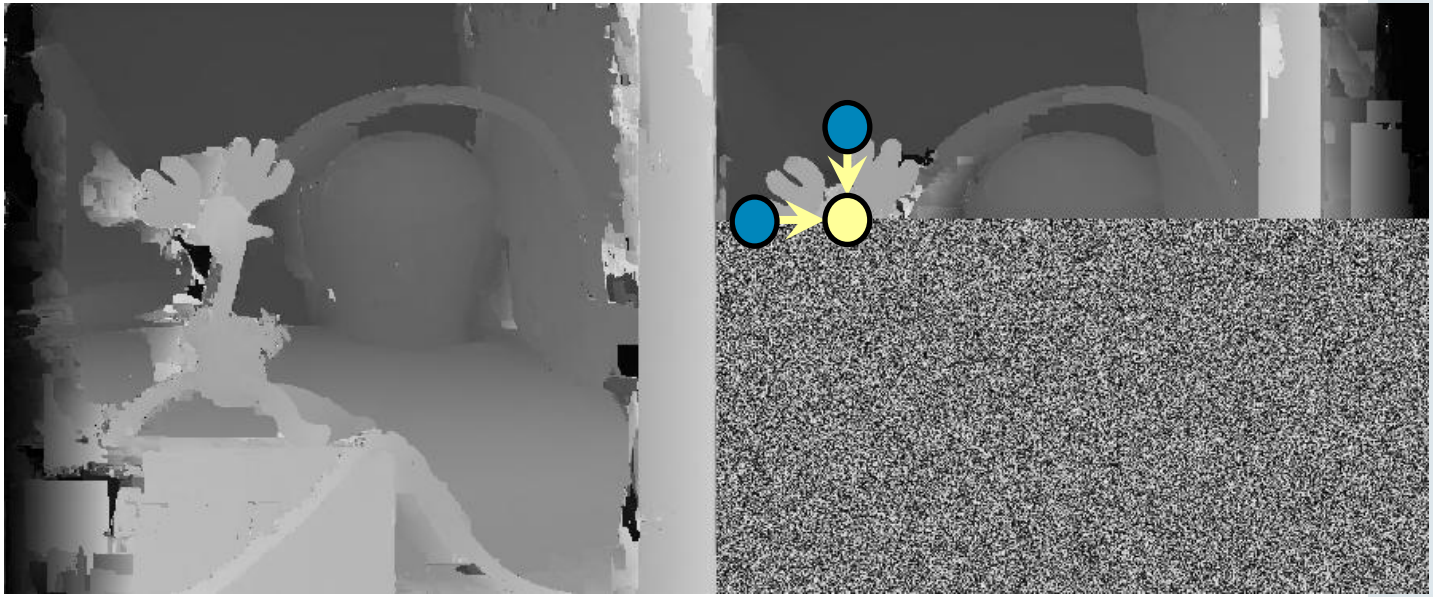
1. Spatial propagation
2. View propagation
3. Temporal propagation
4. Plane refinement

  - Odd iterations:
    - Reverse order
- We run 3 iterations

Spatial Propagation

- Look at the current pixel *p*'s spatial neighbors.
- Check if assigning *p* to a spatial neighbor's plane leads to lower aggregated costs.

Left image – Reindeer (Middlebury)

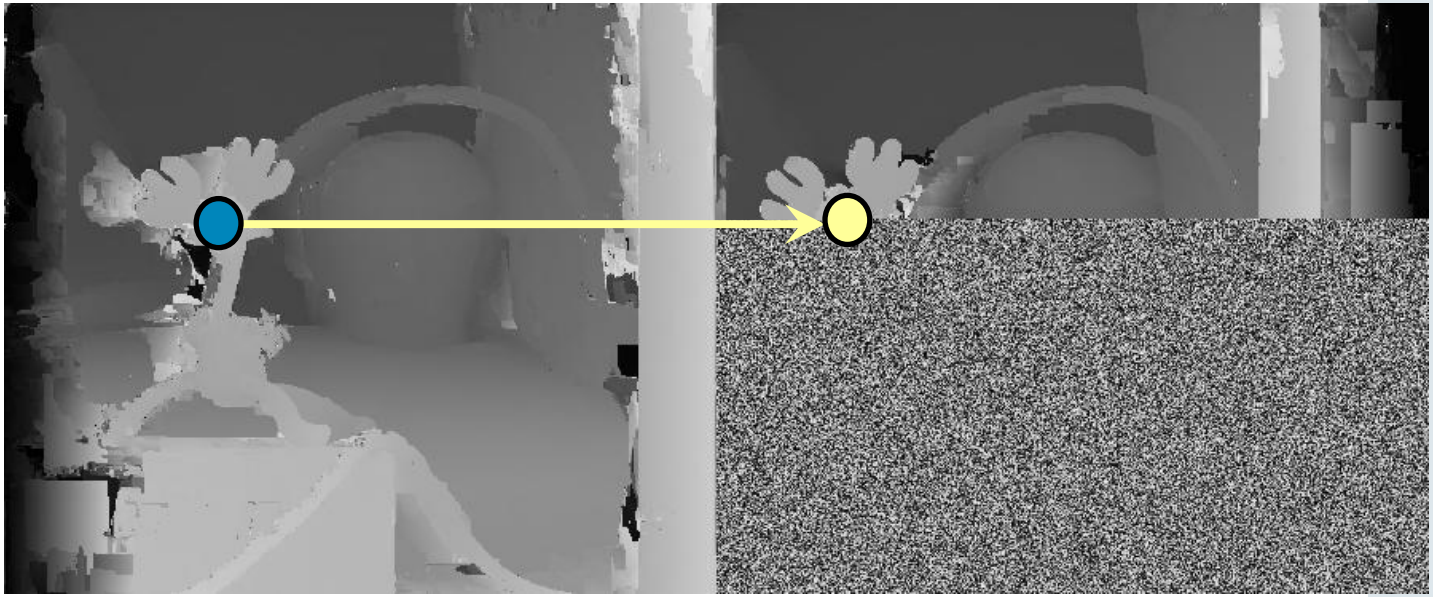


Left and right disparity maps (intermediate step of iteration 1)

View Propagation

- Find all pixels that have *p* as their matching point according to their current disparity.
- Check if plane of a matching point improves costs.

Left image – Reindeer (Middlebury)



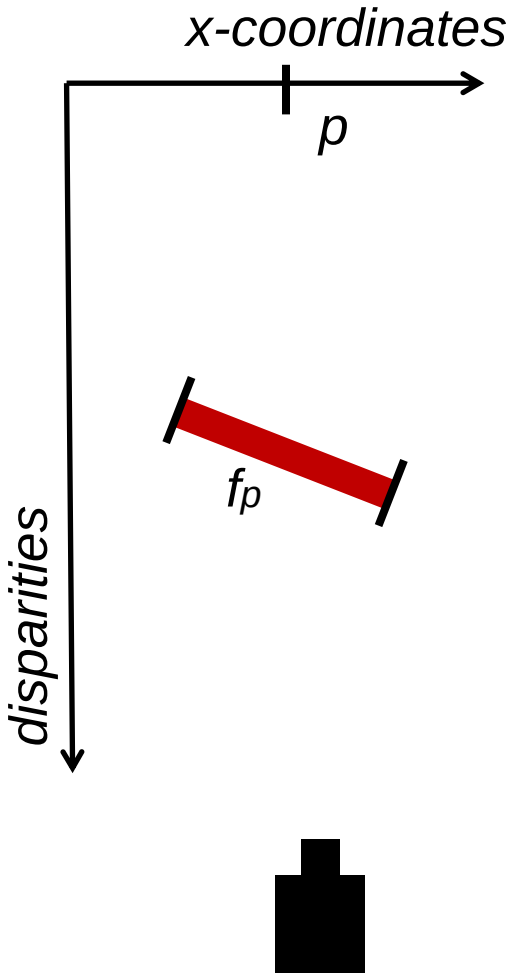
Left and right disparity maps (intermediate step of iteration 1)

Temporal Propagation

- Applicable if operating on video sequences
- Find planes at *p's* coordinates in the previous and consecutive frames.
- Check if planes improve current costs.

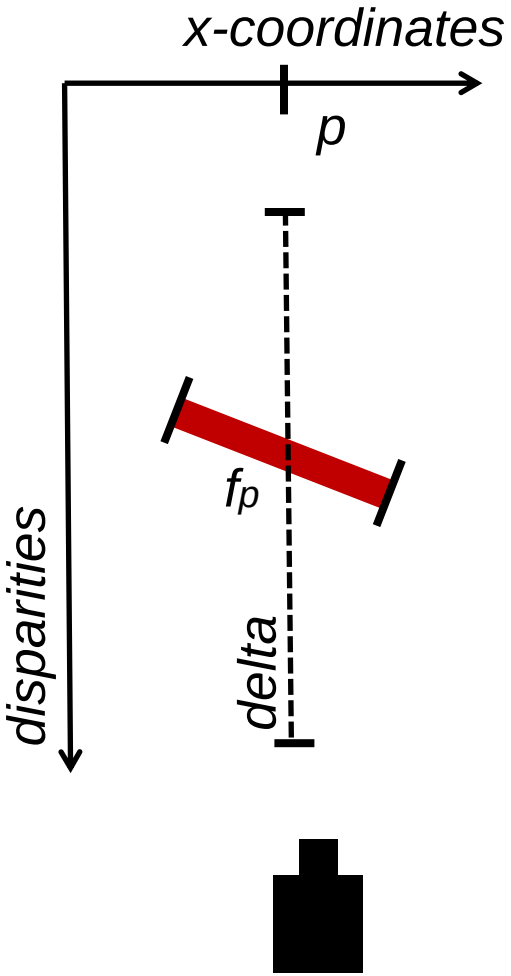
Plane Refinement

- Take *p*'s current plane f_p.



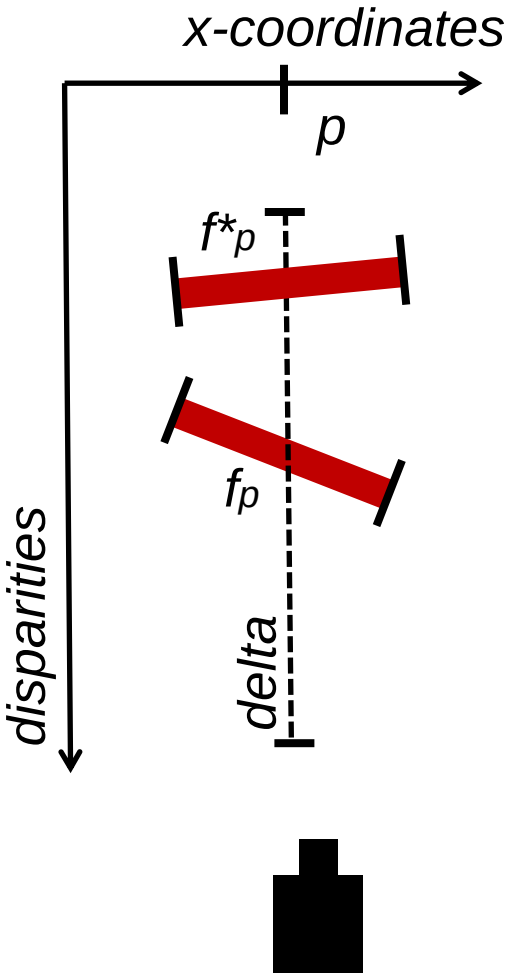
Plane Refinement

- Take *p*‘s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.



Plane Refinement

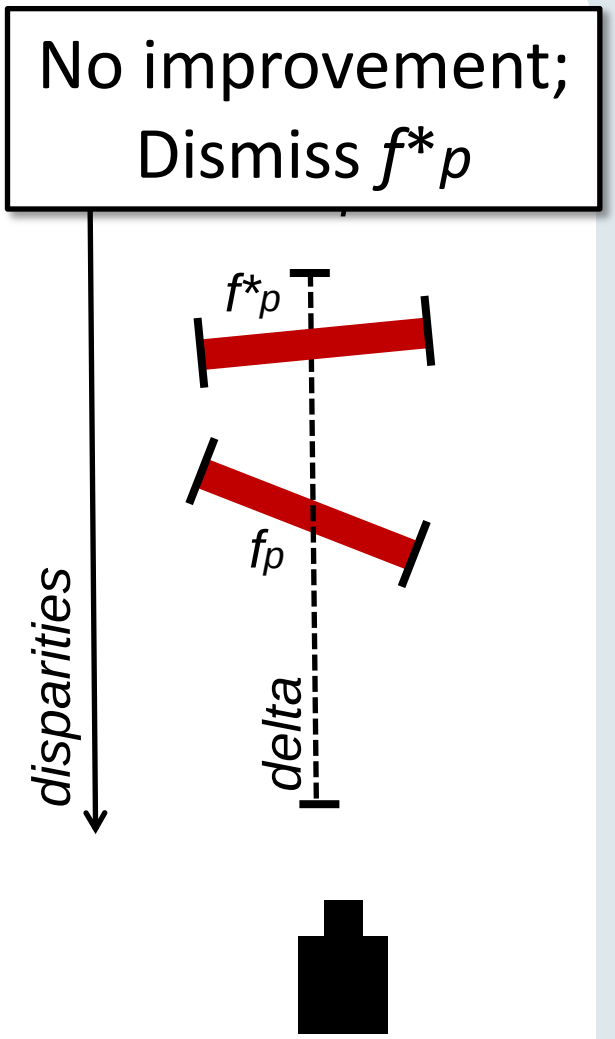
- Take *p*‘s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*



Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

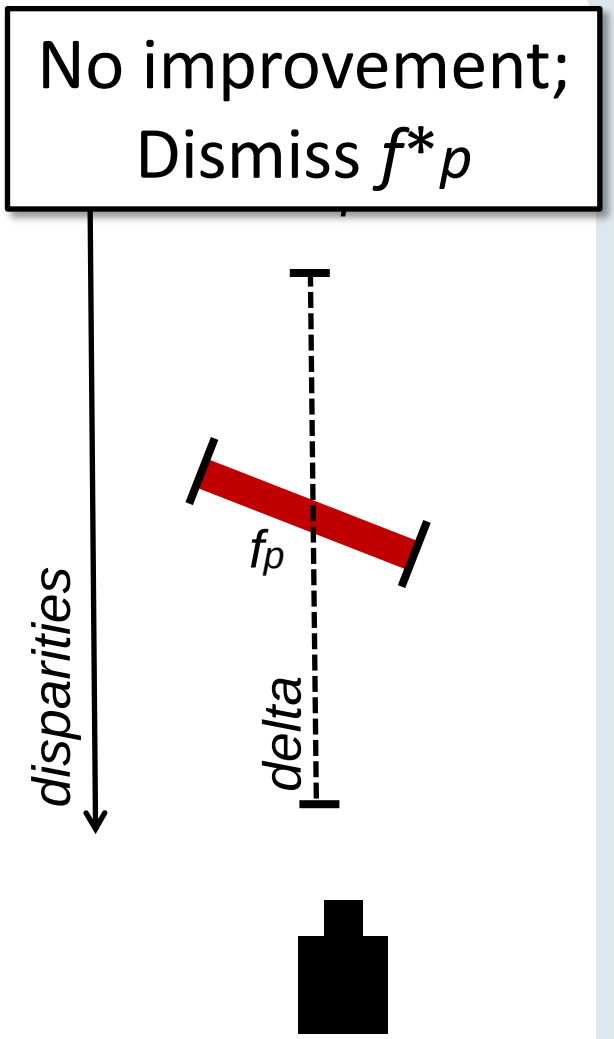
    $f_p := f^*_p$



Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

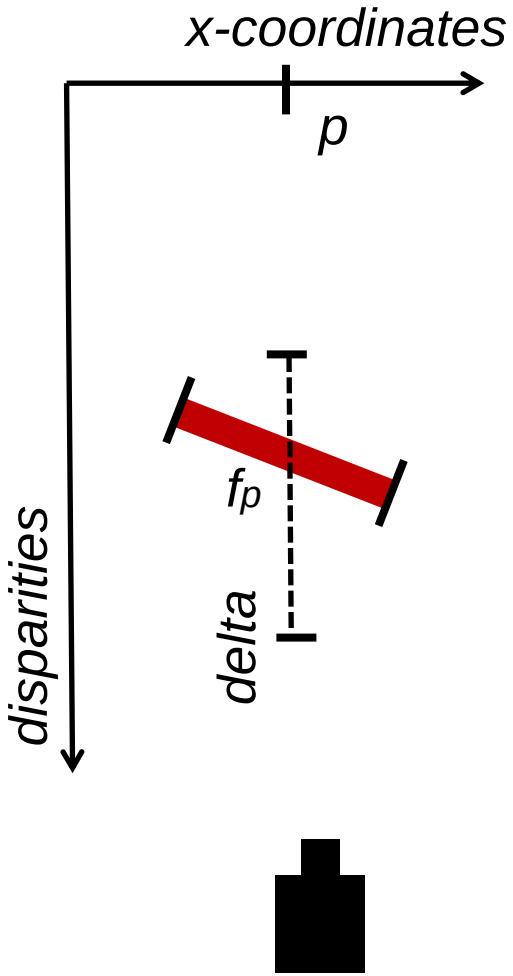
    $f_p := f^*_p$



Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

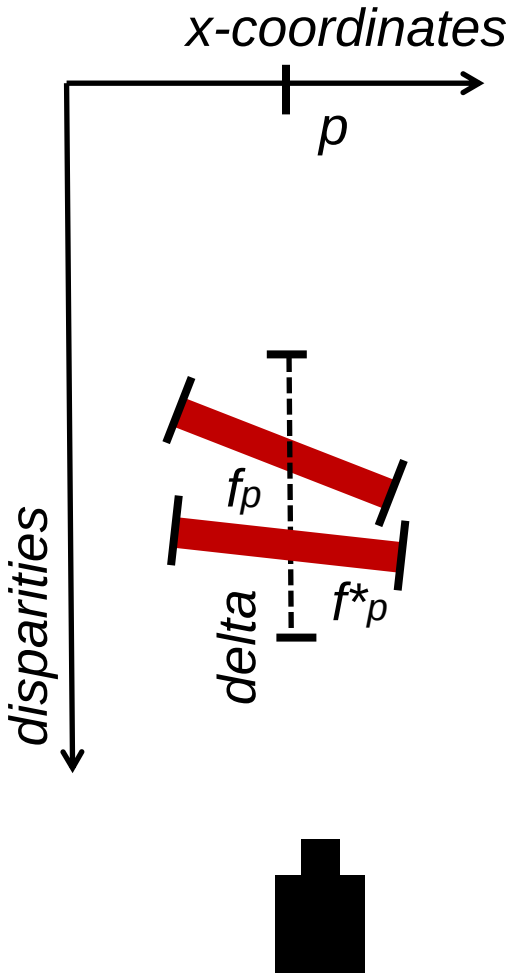
    $f_p := f^*_p$
  - *delta* := *delta* / 2 (exponential reduction)
  - Break if *delta* < *eps*



Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

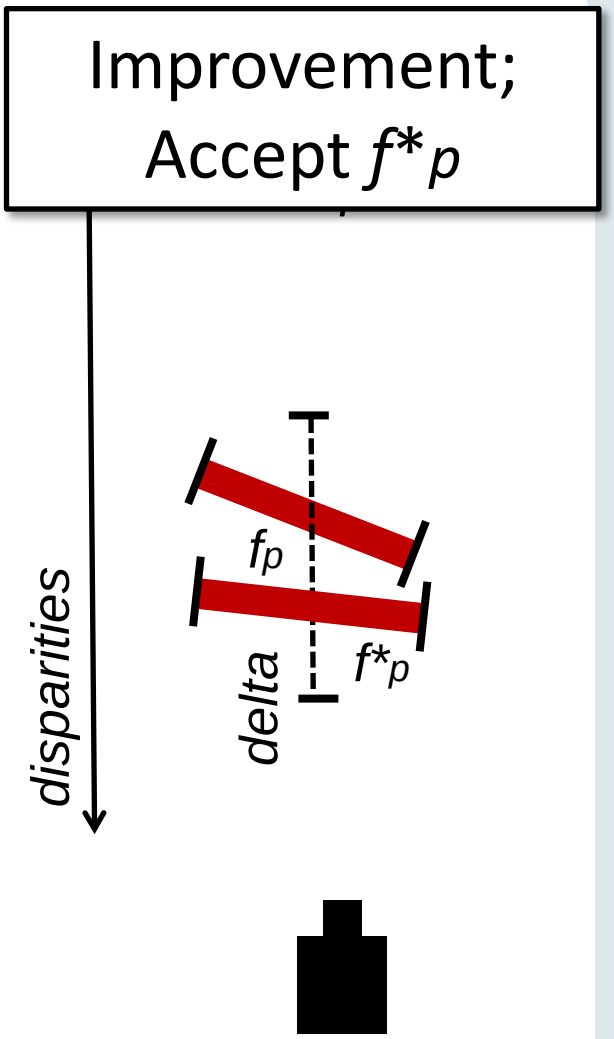
    $f_p := f^*_p$
  - *delta* := *delta* / 2 (exponential reduction)
  - Break if *delta* < *eps*



Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

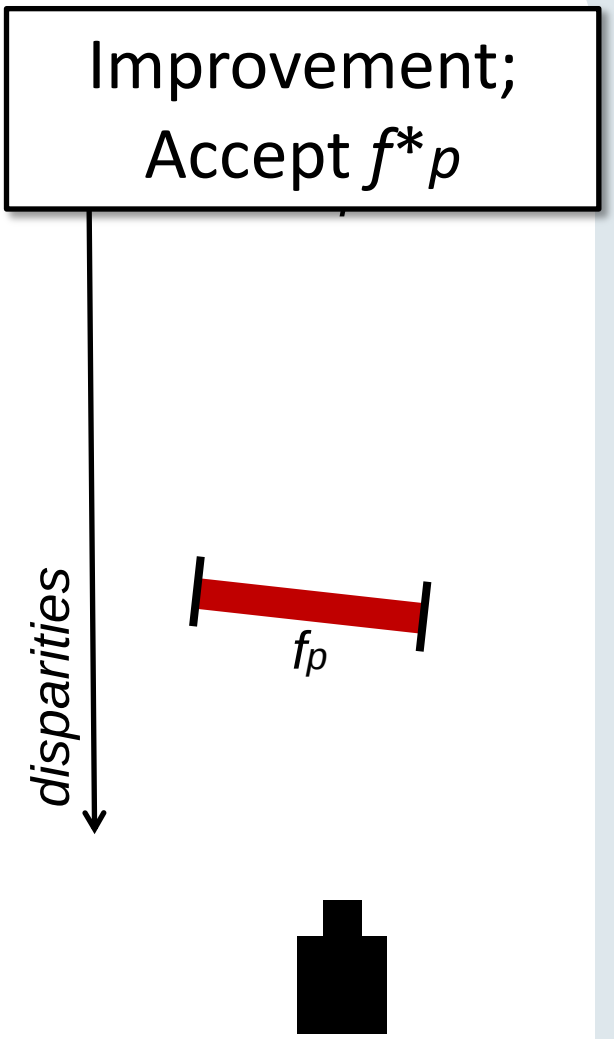
    $f_p := f^*_p$
  - *delta* := *delta* / 2 (exponential reduction)
  - Break if *delta* < *eps*

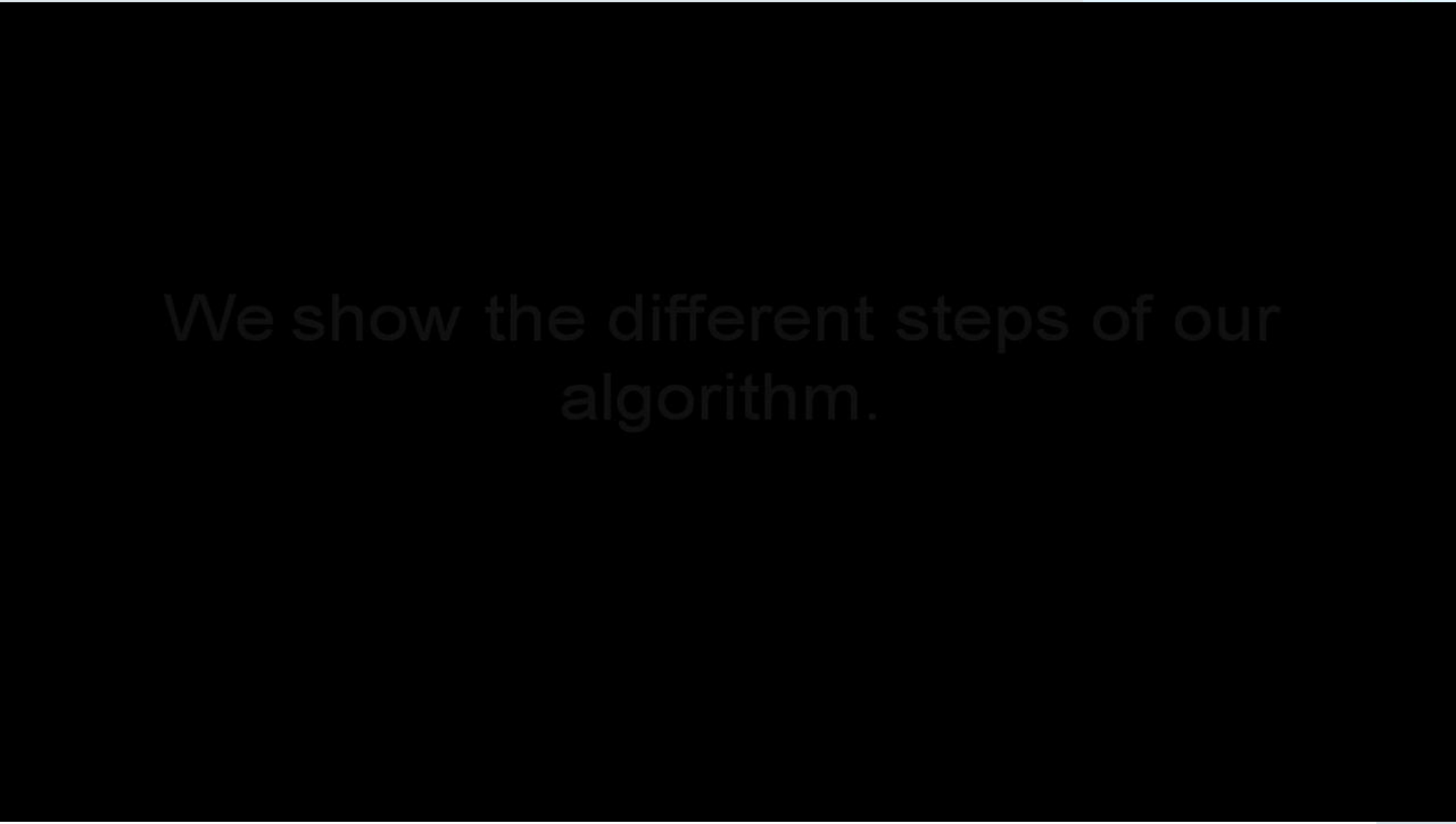


Plane Refinement

- Take *p*'s current plane f_p.
- Distance *delta* defines the maximum allowed change of f_p.
- Loop
  - Compute a random plane f^*_p within *delta*
  - If f^*_p improves costs

    $f_p := f^*_p$
  - *delta* := *delta* / 2 (exponential reduction)
  - Break if *delta* < *eps*




We show the different steps of our algorithm.

Global Methods

- Our slanted windows can be used as a data term for global methods (see paper).

Results

Evaluation

- We implement two competitors in our PatchMatch framework.
  - Competitor 1:
    - **Fronto-parallel** windows matched at **integer** disparities
  - Competitor 2:
    - **Fronto-parallel** windows matched at **sub-pixel** disparities
- Testbed:
  - Middlebury

Error threshold 1 (Middlebury default)

Algorithm	thresh.	Rank	Tsukuba			Venus			Teddy			Cones		
			nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	1.0	11	2.09_{57}	2.33_{43}	9.31_{54}	0.21_{20}	0.39_{15}	2.62_{26}	$\mathbf{2.99}_{1}$	8.16_{8}	9.62_{2}	2.47_{4}	7.80_{8}	7.11_{6}
Fronto-Par. Sub-Pixel	1.0	22	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integer	1.0	32	2.23_{58}	2.44_{44}	9.18_{53}	0.25_{25}	0.41_{16}	2.21_{16}	6.73_{34}	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}

Ranking in the Middlebury online table

Error threshold 1 (Middlebury default)

- Fronto-parallel windows at integer disparities (Comp. 1)
- Rank **32** of **110** Middlebury submissions

Algorithm	thresh.	Rank												
Slanted Support	1.0	11												
Fronto-Par. Sub-Pixel	1.0	2[illegible]	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integer	1.0	32	2.23_{58}	2.44_{44}	9.18_{53}	0.25_{25}	0.41_{16}	2.21_{16}	6.73_{34}	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}

Ranking in the Middlebury online table

Left image – Teddy set (Middlebury)

Disparity map

Error map (black = wrong)

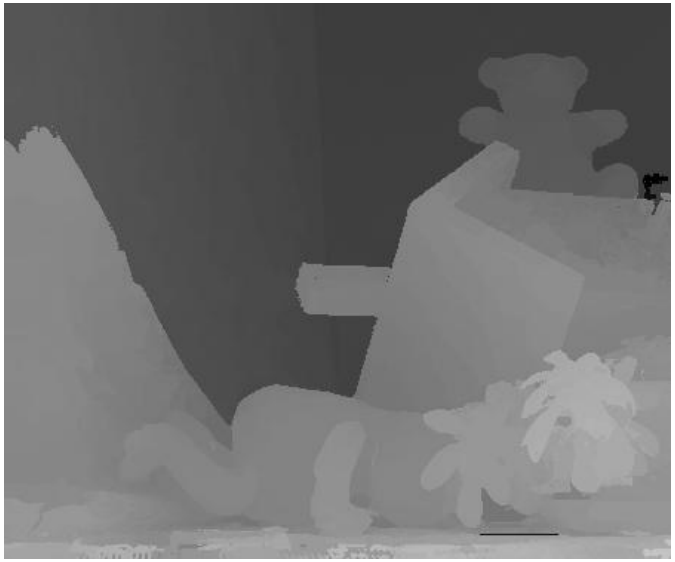
Error threshold 1 (Middlebury default)

- Fronto-parallel windows at sub-pixel disp. (Comp. 2)
- Rank **22** of **110** Middlebury submissions

Algorithm	thresh.	Rank												
Slanted Support	1.0	1	2.09_{57}	2.33_{43}	9.31_{54}	0.21_{20}	0.39_{15}	2.62_{26}	**2.99_{1}**	8.16_{8}	9.62_{2}	2.47_{4}	7.80_{8}	7.11_{6}
Fronto-Par. Sub-Pixel	1.0	22	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integer	1.0	32	2.23_{58}	2.44_{44}	9.18_{53}	0.25_{25}	0.41_{16}	2.21_{16}	6.73_{34}	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}

Ranking in the Middlebury online table

Left image – Teddy set (Middlebury)



Disparity map

Error map (black = wrong)

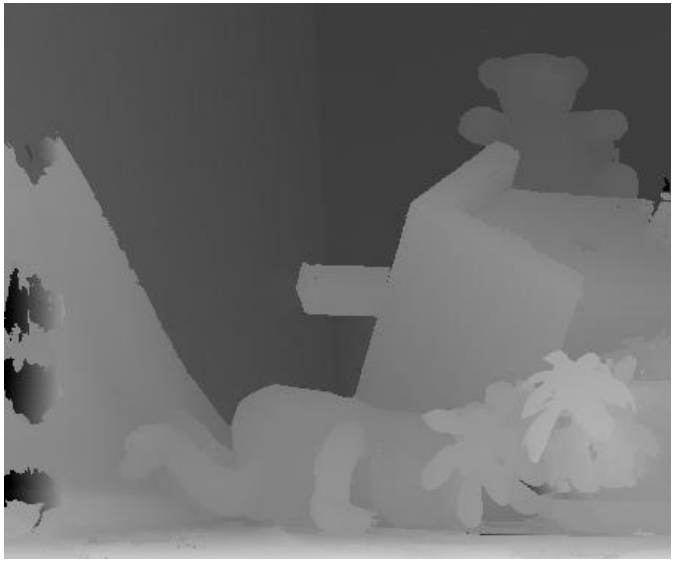
Error thresh

- Our slanted windows
- Rank **11** of **110** Middlebury submissions
- Best-performing local method

Algorithm	thresh.	Rank	nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	1.0	11	2.09_{57}	2.33_{43}	9.31_{54}	0.21_{20}	0.39_{15}	2.62_{26}	$\mathbf{2.99}_{1}$	8.16_{8}	9.62_{2}	2.47_{4}	7.80_{8}	7.11_{6}
Fronto-Par. Sub-Pixel	1.0	22	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integer	1.0	32	2.23_{58}	2.44_{44}	9.18_{53}	0.25_{25}	0.41_{16}	2.21_{16}	6.73_{34}	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}

Ranking in the Middlebury online table

Left image – Teddy set (Middlebury)



Disparity map

Error map (black = wrong)

Error thresh

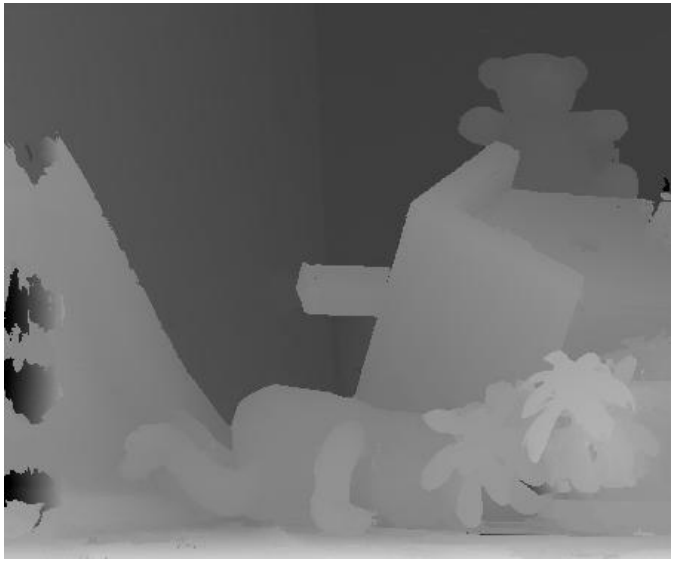
- Our slanted windows
- Rank **11** of **110** Middlebury submissions
- Best-performing local method

Algorithm	thresh.	Rank	nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	1.0	11	2.09_{57}	2.33_{43}	9.31_{54}	0.21_{20}	0.39_{15}	2.62_{26}	**2.99_{1}**	8.16_{8}	9.62_{2}	2.47_{4}	7.80_{8}	7.11_{6}
Fronto-Par. Sub-Pixel	1.0	22	2.13_{57}	2.32_{43}	9.92_{60}	0.25_{27}	0.42_{16}	2.76_{27}	5.52_{17}	10.5_{17}	14.2_{15}	3.23_{27}	8.33_{17}	8.20_{18}
Fronto-Par. Integ									34	12.2_{34}	16.9_{38}	3.71_{35}	8.90_{27}	9.31_{35}

e

- Rank #1 on Teddy in non-occluded regions

Left image – Teddy set (Middlebury)



Disparity map

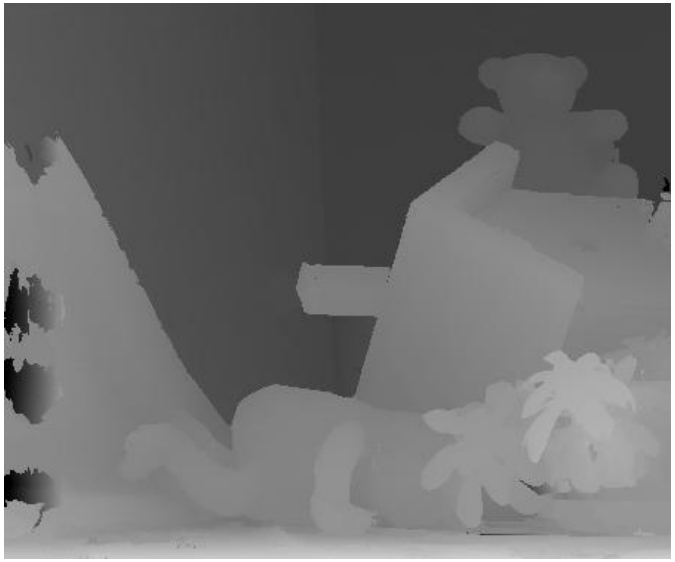
Error map (black = wrong)

Error threshold 0.5 (Sub-Pixel Evaluation)

Algorithm	thresh.	Rank	Tsukuba			Venus			Teddy			Cones		
			nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	0.5	2	15.0_{37}	15.4_{36}	20.3_{45}	1.00_{4}	1.34_{4}	7.75_{7}	$\mathbf{5.66}_{1}$	11.8_{2}	$\mathbf{16.5}_{1}$	$\mathbf{3.80}_{1}$	$\mathbf{10.2}_{1}$	$\mathbf{10.2}_{1}$
Fronto-Par. Sub-Pixel	0.5		14.1_{36}	14.4_{32}	19.2_{35}	1.73_{7}	2.15_{7}	7.84_{7}	9.63_{6}	16.2_{8}	22.6_{8}	7.08_{24}	12.6_{16}	13.2_{11}
Fronto-Par. Integer	0.5	41												

- Our slanted windows
- Rank **2** of **110** Middlebury submissions

Left image – Teddy set (Middlebury)



Disparity map

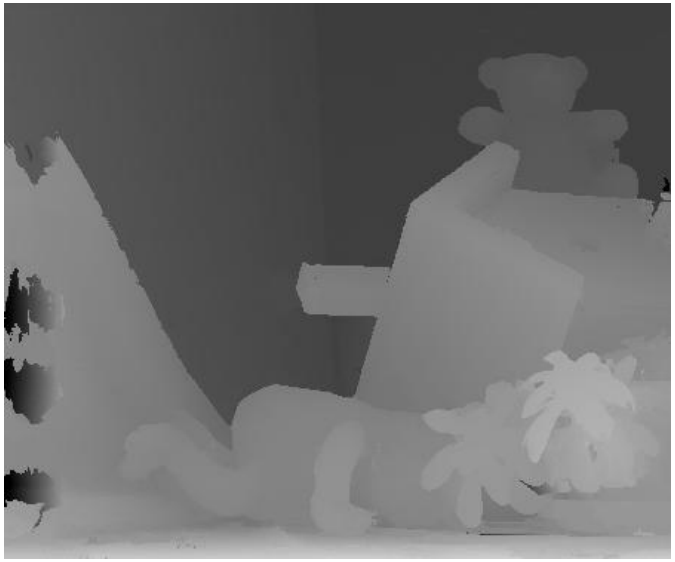
Error map (black = wrong)

Error threshold 0.5 (Sub-Pixel Evaluation)

Algorithm	thresh.	Rank	Tsukuba			Venus			Teddy			Cones		
			nocc	all	disc	nocc	all	disc	nocc	all	disc	nocc	all	disc
Slanted Support	0.5	2	15.0_{37}	15.4_{36}	20.3_{45}	1.00_{4}	1.34_{4}	7.75_{7}	**5.66_{1}**	11.8_{2}	**16.5_{1}**	**3.80_{1}**	**10.2_{1}**	**10.2_{1}**
Fronto-Par. Sub-Pixel	0.5	4	14.1_{36}	14.4_{32}	19.2_{35}	1.73_{7}	2.15_{7}	7.84_{7}	9.63_{6}	16.2_{8}	22.6_{8}	7.08_{24}	12.6_{16}	13.2_{11}
Fron											29.1_{37}	11.6_{53}	16.9_{42}	18.3_{41}

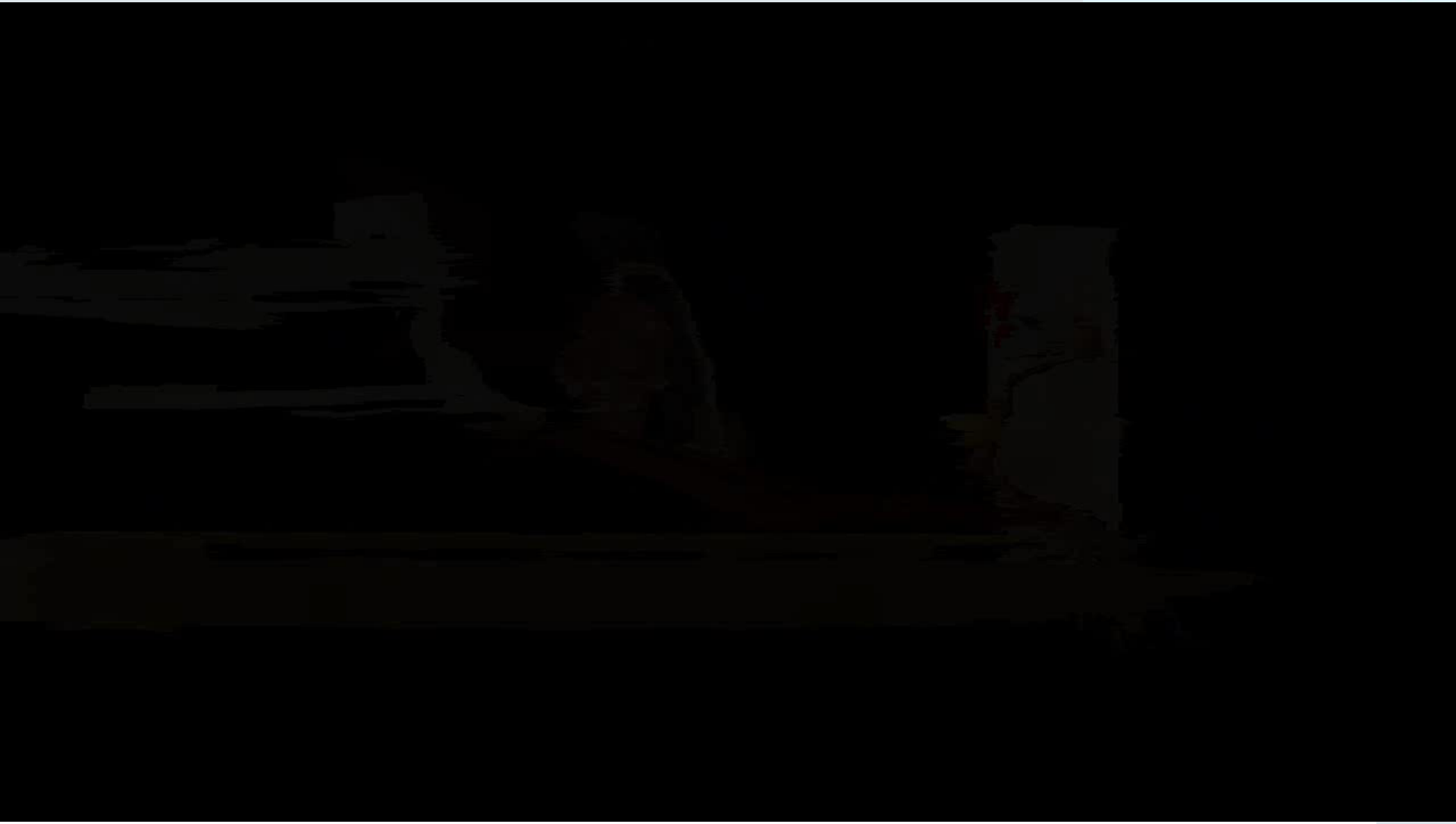
- First rank on almost all error measurements on the complex Teddy and Cones sets

Left image – Teddy set (Middlebury)



Disparity map

Error map (black = wrong)



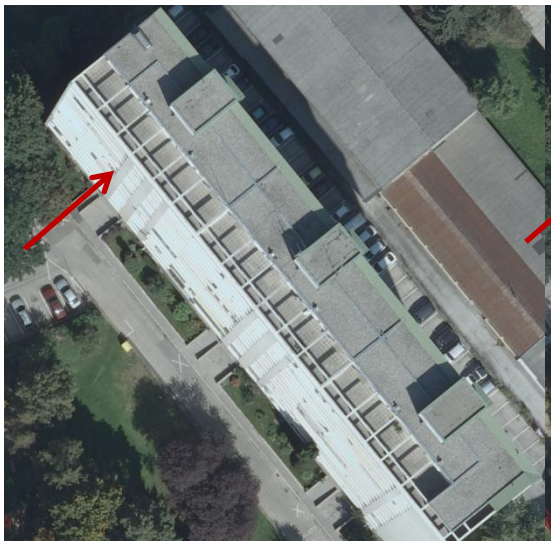
Computational Speed

- Approximately 1 minute per Middlebury pair (CPU)
- Runtime independent of the number of labels (disparities)
  - Well suited for optical flow computation (future work)
- Low memory requirements:
  - We only need to hold the current matching costs and a plane at each pixel in memory
  - => We can do high-resolution stereo

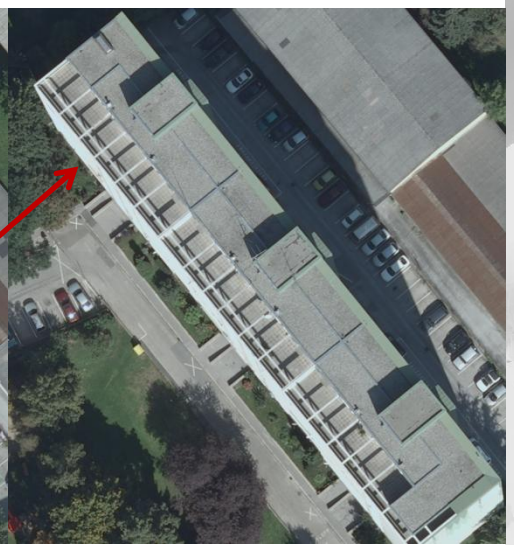
Image of Graz (2048x2048 pixels)

Image of Graz (2048x2048 pixels)

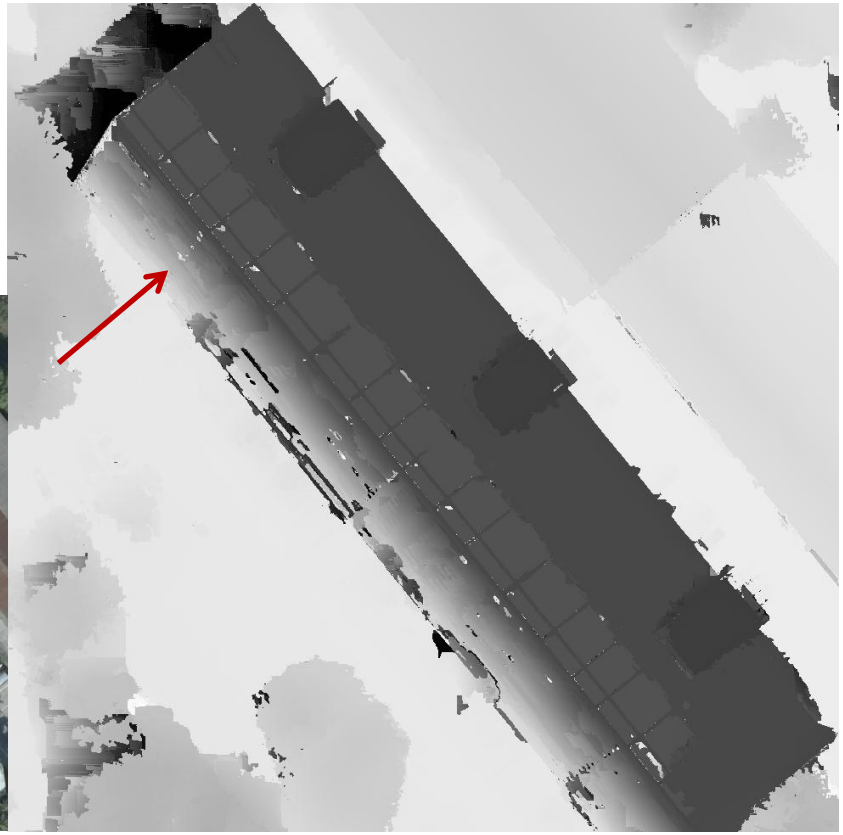
Image of Graz (2048x2048 pixels)



Crop of left image



Crop of right image



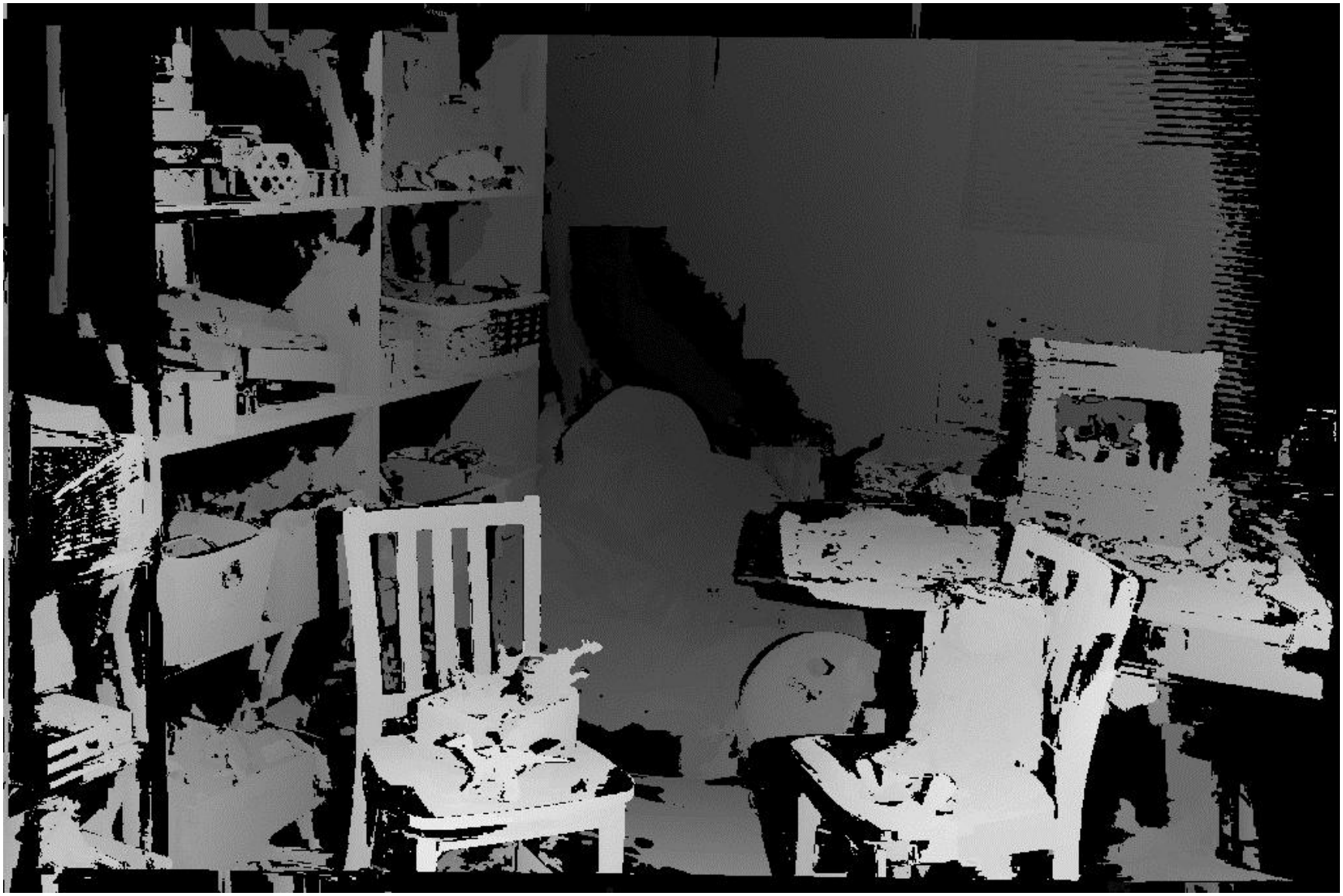
Left disparity map
(before left/right check)

Playroom Set (image courtesy Daniel Scharstein)

Playroom Set (image courtesy Daniel Scharstein)

Playroom Set (image courtesy Daniel Scharstein)

Playroom Set (image courtesy Daniel Scharstein)



Conclusions

- Local stereo algorithm uses slanted support windows
- An ideal algorithm to find the slanted windows is PatchMatch
- High quality results on complex images and videos at reasonable runtimes

Future Work

- Optical flow
- Improve computational speed (GPU implementation – real time)