Interactive Media Systems Group
Software Technology & Interactive Systems
Vienna University of Technology

Simple but Effective Tree Structures for Dynamic Programming-Based Stereo Matching

Michael Bleyer and Margrit Gelautz

Vienna University of Technology

Dense Stereo Matching

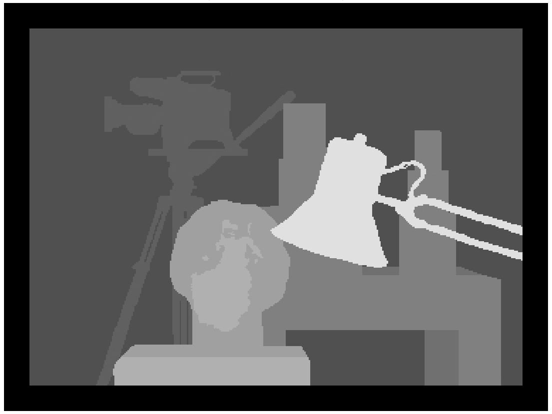
(Left Image)

(Right Image)

Dense Stereo Matching

(Left Image)

(Right Image)



(Disparity Map)

Structure

- Introduction
- Previous work
- The Simple Tree Method
  - Energy function
  - Energy optimization
- Results
- Conclusions

What stereo method to choose for a practical application?

- Local methods
  - Computationally efficient
  - Results often too poor
- Global methods
  - Good-quality results
  - Usually too slow
- Goal
  - Develop a stereo algorithm that delivers maximum accuracy at minimum computation time

Global Stereo Methods

- Find a disparity map *D* that minimizes

$$E(D) = E_{data}(D) + E_{smooth}(D).$$

Global Stereo Methods

- Find a disparity map *D* that minimizes

$$E(D) = E_{data}(D) + E_{smooth}(D).$$

Photo consistency assumption

Global Stereo Methods

- Find a disparity map *D* that minimizes

$$E(D) = E_{data}(D) + E_{smooth}(D).$$

Photo consistency assumption (→ E_{data})

Smoothness assumption (→ E_{smooth})

Global Stereo Methods

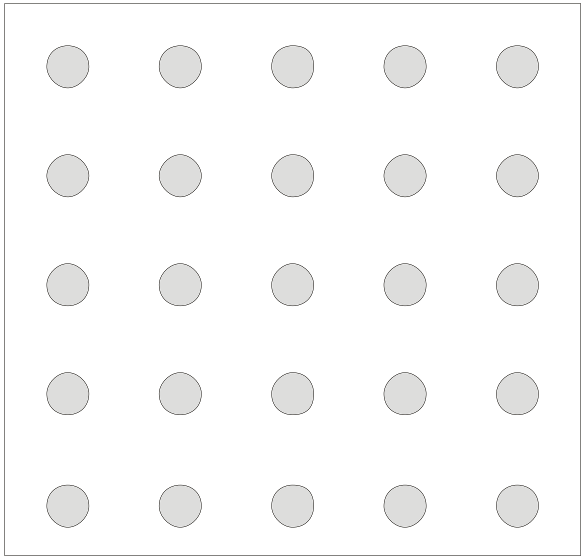
- Find a disparity map *D* that minimizes

$$E(D) = E_{data}(D) + E_{smooth}(D).$$

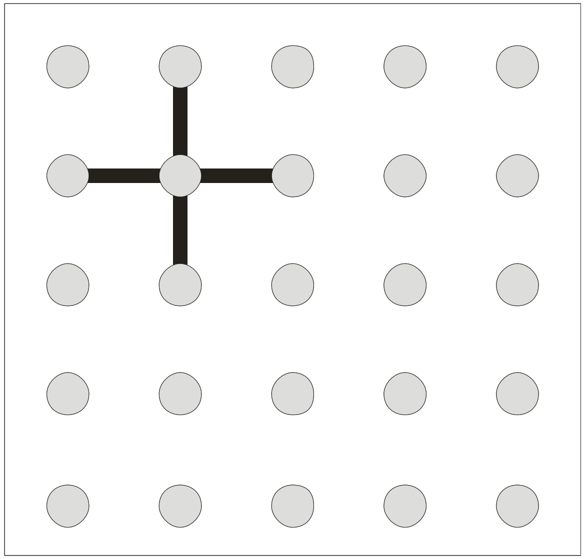
Photo consistency assumption (→ E_{data}) | Smoothness assumption (→ E_{smooth})

- Definition of smoothness neighbourhood defines complexity of optimization problem

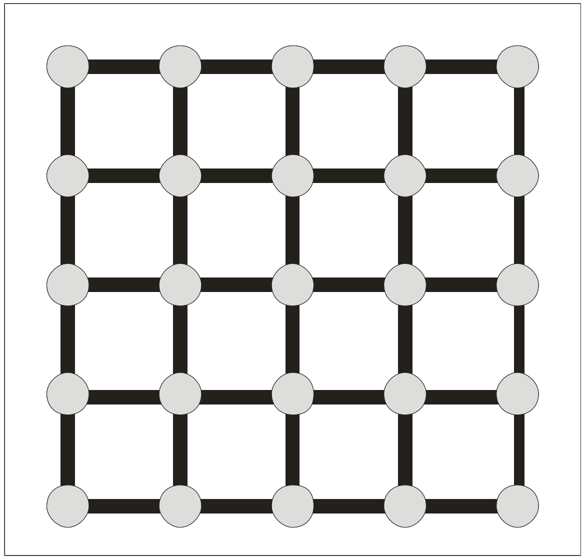
Optimization on 4-Connected Grid



Optimization on 4-Connected Grid



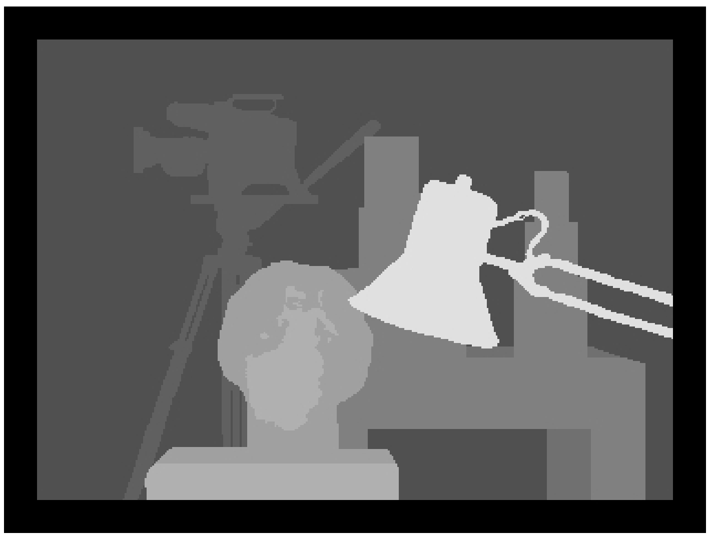
Optimization on 4-Connected Grid



(4-Connected Grid)

- Optimization NP-complete (discontinuity preserving smoothness functions)
- Approximation via Graph-Cuts or Belief Propagation
- Good results, but computationally expansive

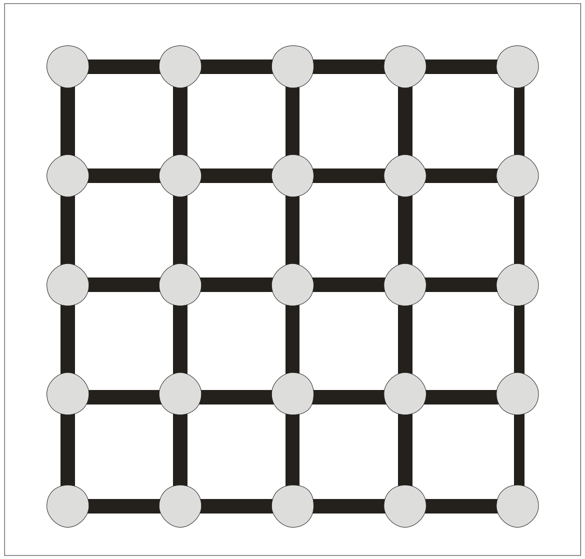
Disparity Map computed via Graph-Cuts
(taken from the Middlebury website)



(Ground Truth)

(Graph-Cuts)

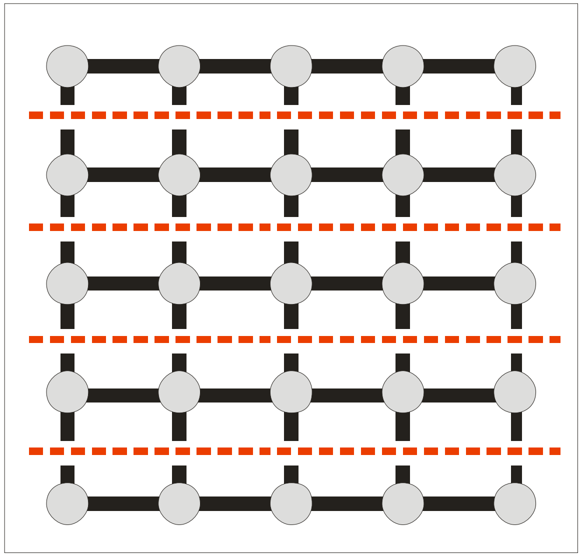
Dynamic Programming (DP)



(4-Connected Grid)

- Discard vertical smoothness edges
- Exact optimization via DP
- Computationally fast, but scanline streaking

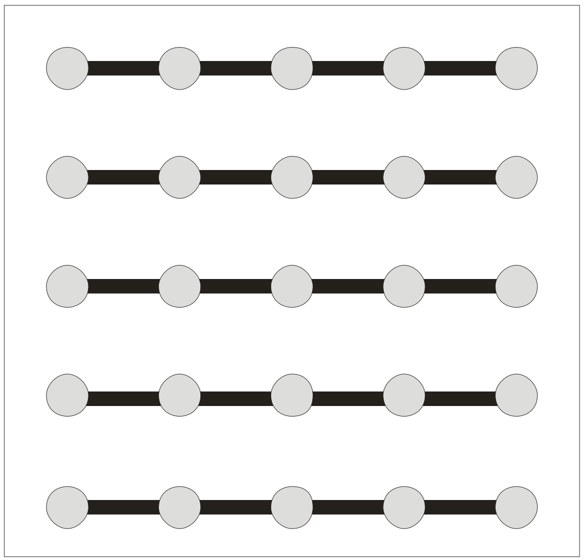
Dynamic Programming (DP)



(4-Connected Grid)

- Discard vertical smoothness edges
- Exact optimization via DP
- Computationally fast, but scanline streaking

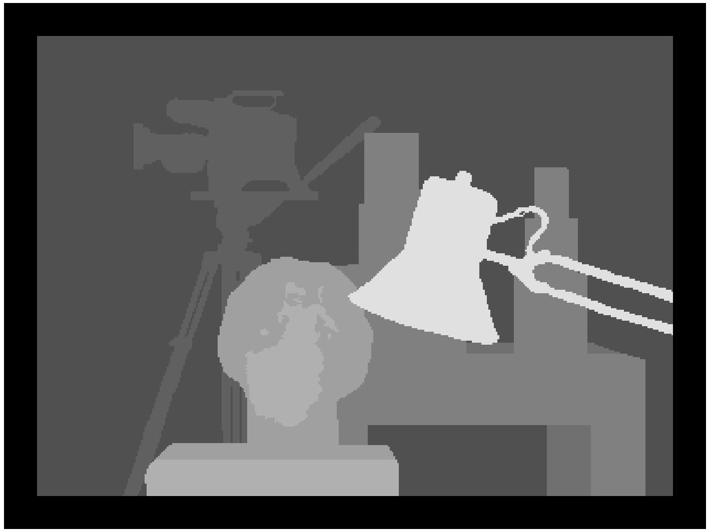
Dynamic Programming (DP)



(DP Neighbourhood Structure)

- Discard vertical smoothness edges
- Exact optimization via DP
- Computationally fast, but scanline streaking

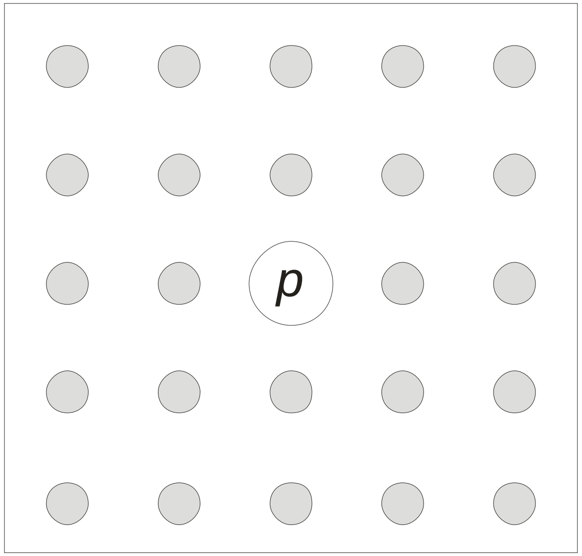
Disparity Map computed using DP
(taken from the Middlebury website)



(Ground Truth)

(Scanline Optimization)

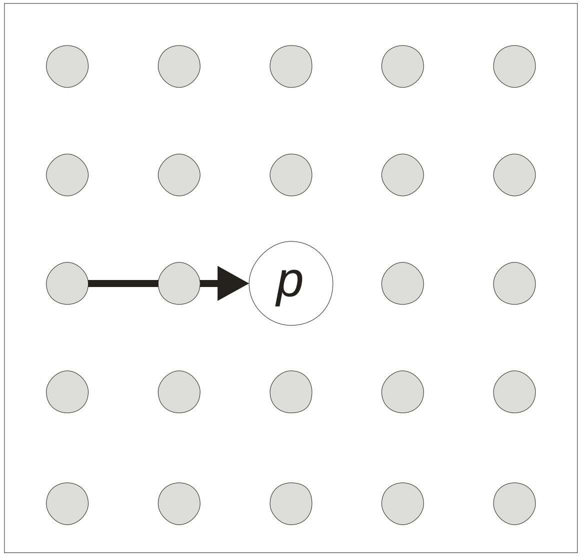
SemiGlobal Matching [Hirschmüller05]



(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

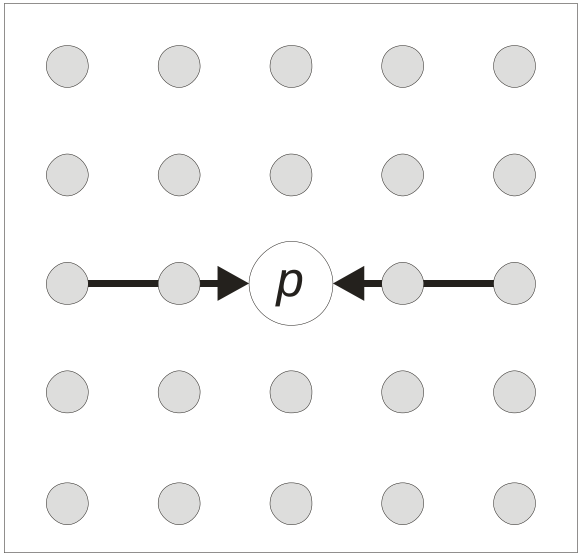
SemiGlobal Matching [Hirschmüller05]



(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

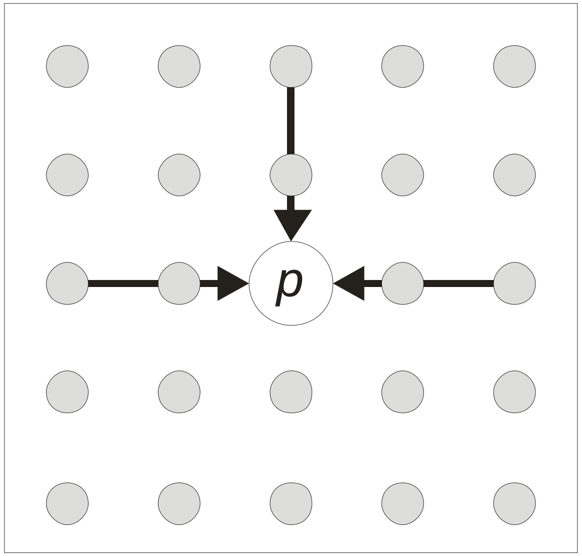
SemiGlobal Matching [Hirschmüller05]



(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

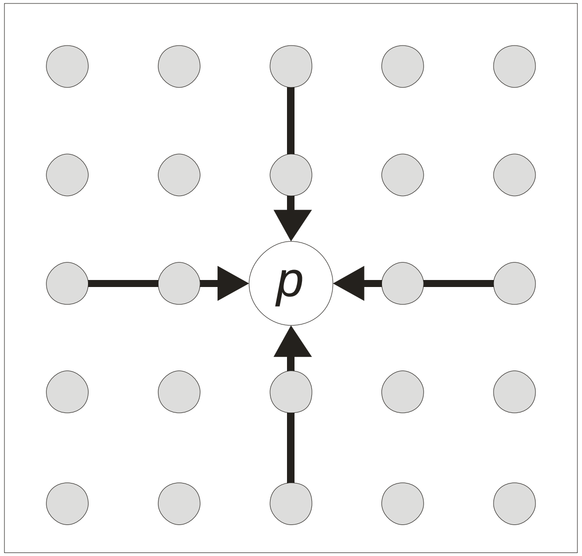
SemiGlobal Matching [Hirschmüller05]



(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

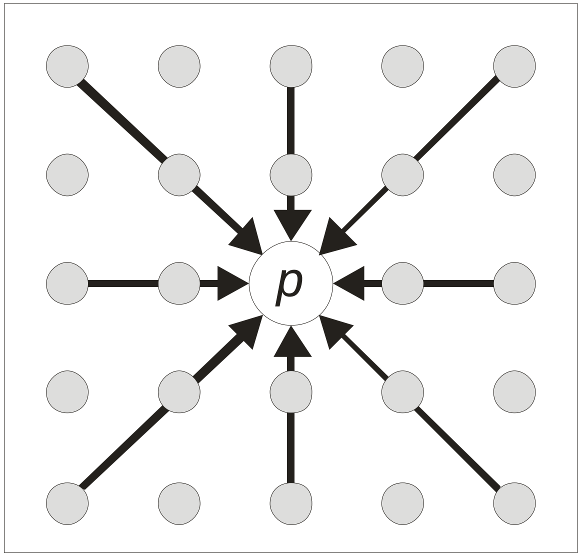
SemiGlobal Matching [Hirschmüller05]



(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

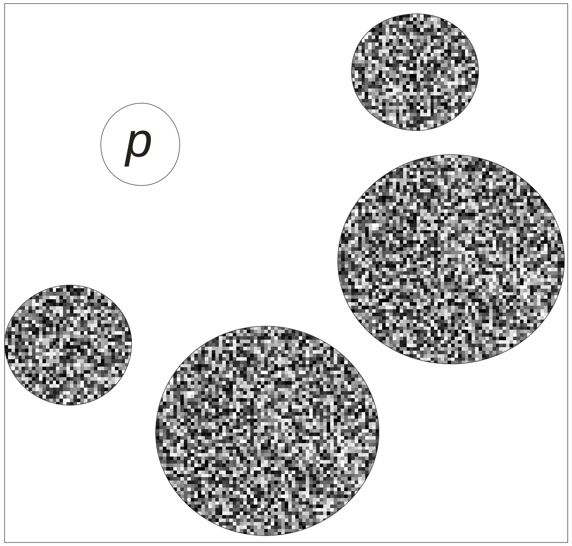
SemiGlobal Matching [Hirschmüller05]



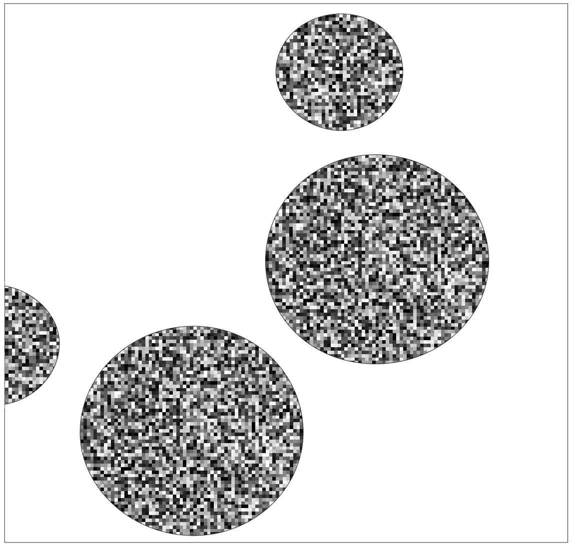
(4-Connected Grid)

- Individual disparity computation at each pixel
- Aggregate DP costs computed from paths in various directions
- Computationally fast, almost no streaks, but poor performance in regions of low texture

SemiGlobal Matching in Untextured Regions

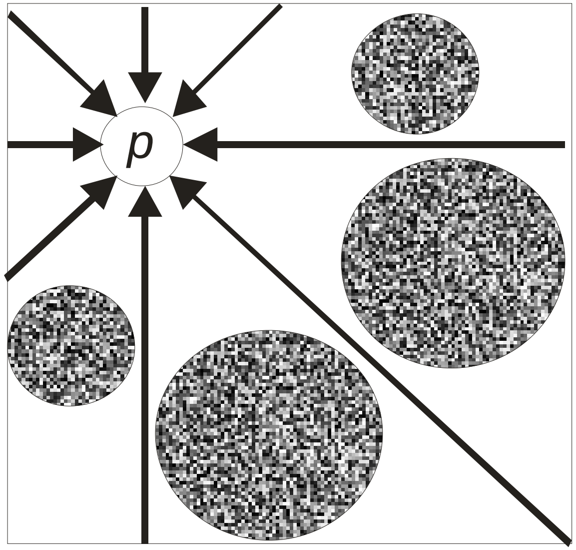


(Left Image)

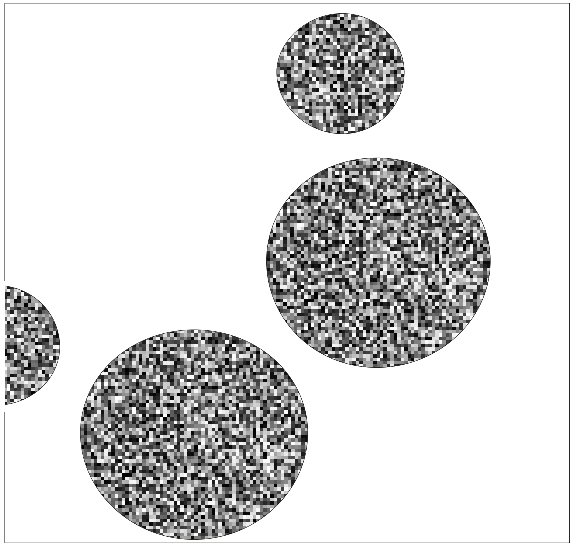


(Right Image)

SemiGlobal Matching in Untextured Regions

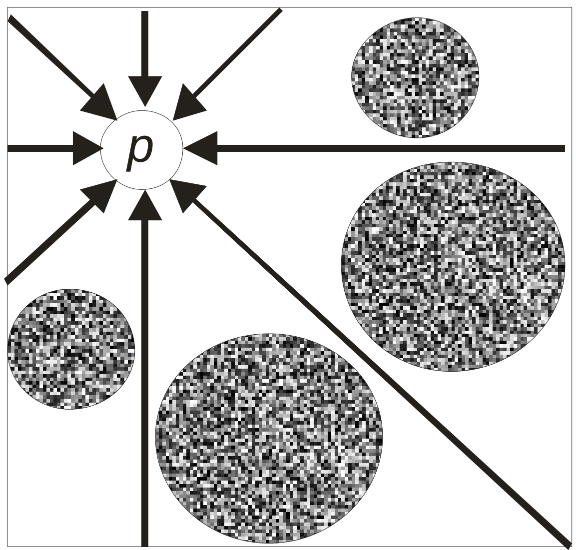


(Left Image)



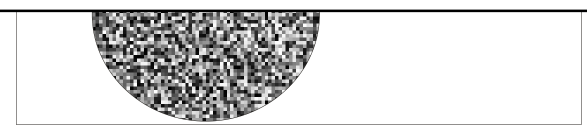
(Right Image)

SemiGlobal Matching in Untextured Regions



(Left Image)

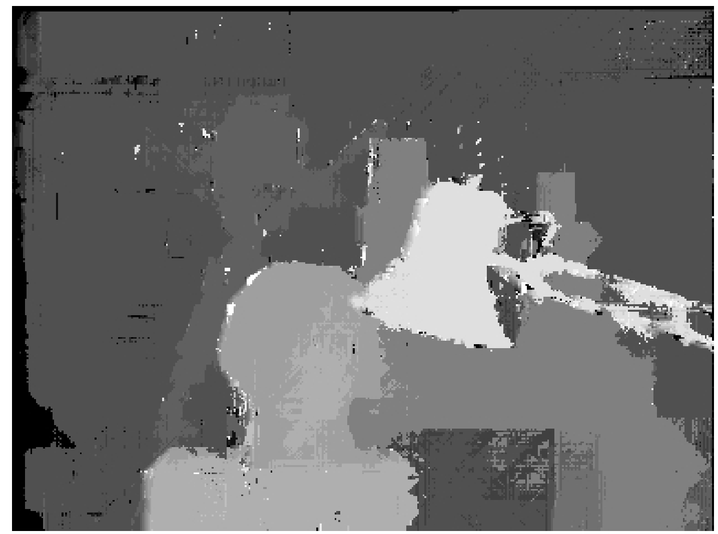
- None of the DP paths captures texture at the correct disparity
- Disparity selection guided by noise



(Right Image)

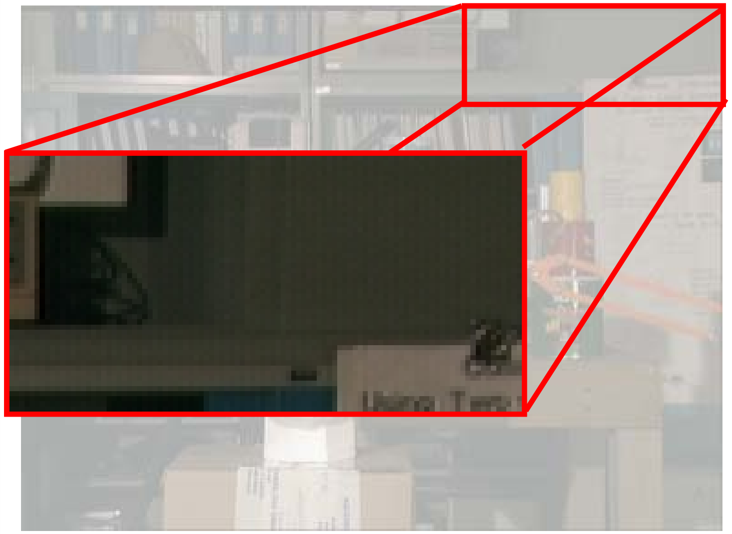
Reimplementation of SemiGlobal Matching

(Left Image)

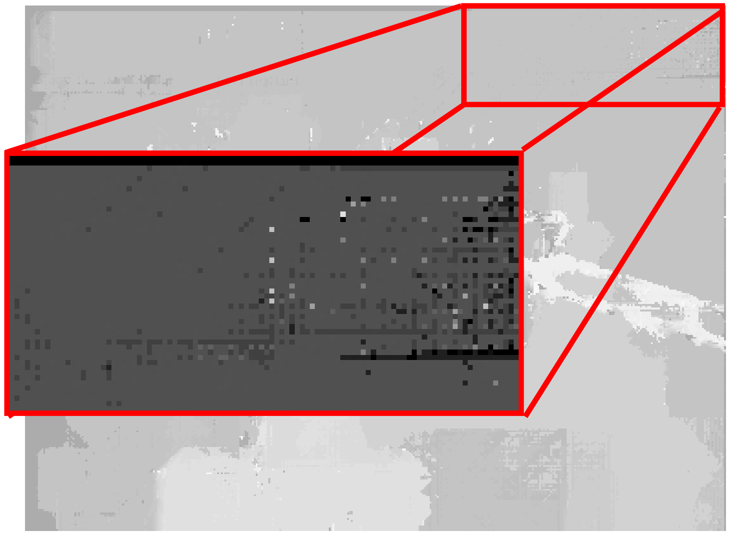


(Disparity Map)

Reimplementation of SemiGlobal Matching



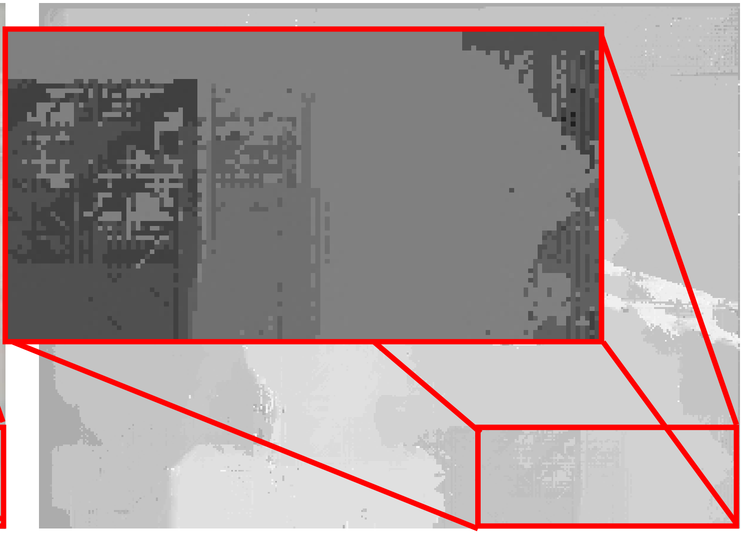
(Left Image)



(Disparity Map)

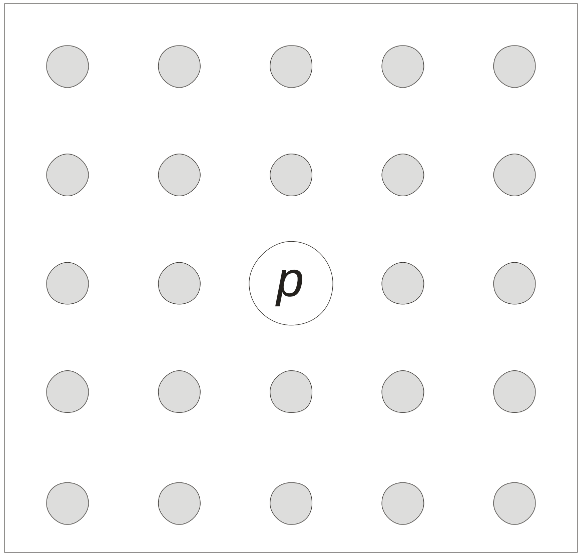
Reimplementation of SemiGlobal Matching

(Left Image)



(Disparity Map)

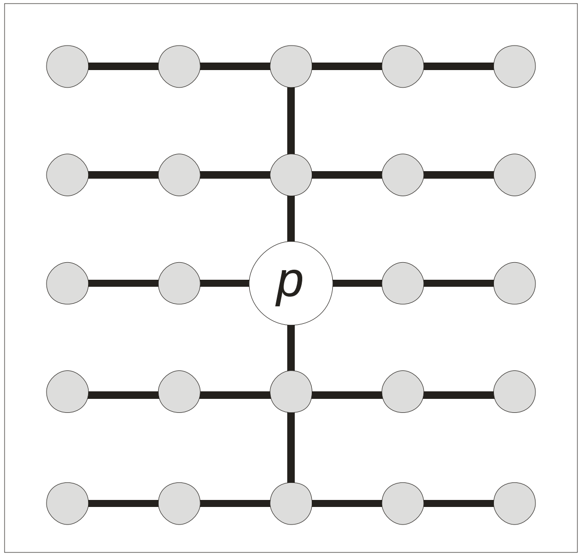
Our Approach (Simple Tree Method)



(Simple Tree Structure)

- Perform a separate disparity computation for each pixel
- Root a tree on the pixel
- DP also works on trees
- Compute exact energy minimum on the tree
- Assign *p* to the disparity that lies on the energy minimum

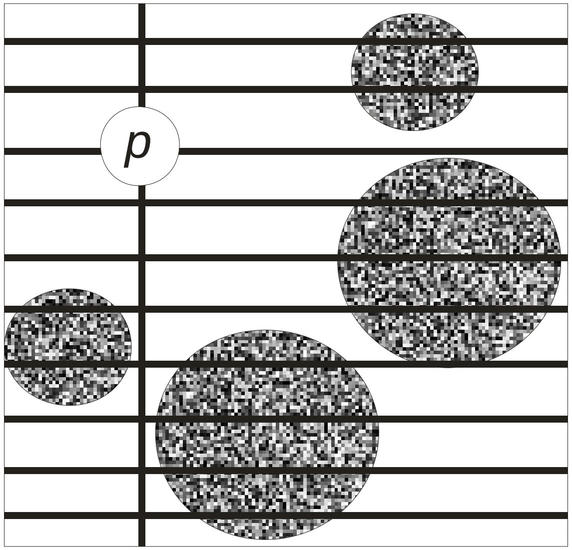
Our Approach (Simple Tree Method)



(Simple Tree Structure)

- Perform a separate disparity computation for each pixel
- Root a tree on the pixel
- DP also works on trees
- Compute exact energy minimum on the tree
- Assign *p* to the disparity that lies on the energy minimum

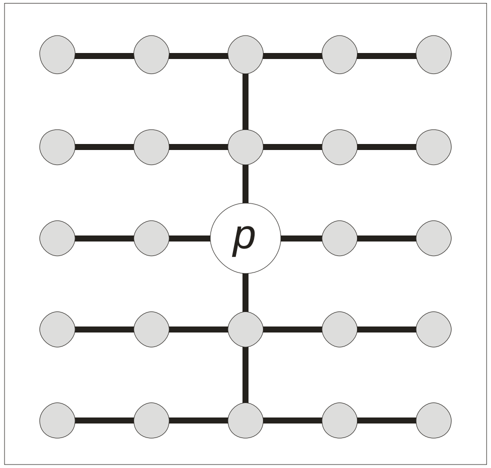
Advantages of Simple Trees



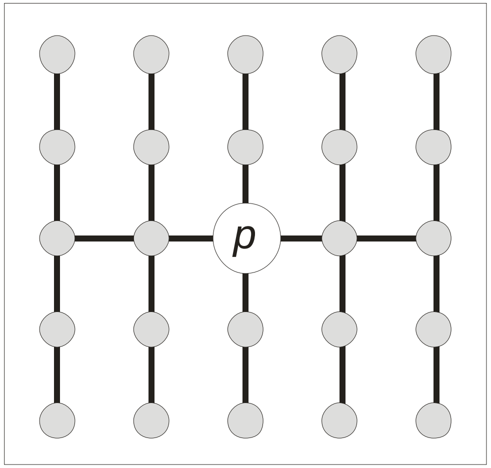
(Simple Tree on the Previous Example)

- Tree structure spans all pixels (does not miss image features)
- Vertical and horizontal smoothness edges (against scanline streaks)
- We include all smoothness edges by using two different tree structures

Two Simple Tree Structures



Horizontal Tree



Vertical Tree

- Allow for incremental computation of optima
- Only 4 DP passes needed

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

BT-measurement on RGB values

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

BT-measurement on RGB values

Modified Potts model

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

BT-measurement on RGB values

Modified Potts model

$$s(d_p, d_q) = \begin{cases} 0 & : \quad d_p = d_q \\ P_1 & : \quad |d_p - d_q| = 1 \\ P_2 & : \quad \text{otherwise.} \end{cases}$$

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

BT-measurement on RGB values

Modified Potts model

$$s(d_p, d_q) = \begin{cases} 0 & : \quad d_p = d_q \\ P_1 & : \quad |d_p - d_q| = 1 \\ P_2 & : \quad \text{otherwise.} \end{cases}$$

Weighted by intensity gradient

Energy Function

$$E(D) = \sum_{p \in I} m(p, d_p) + \sum_{(p,q) \in \mathcal{N}} s(d_p, d_q).$$

BT-measurement on RGB values

Modified Potts model

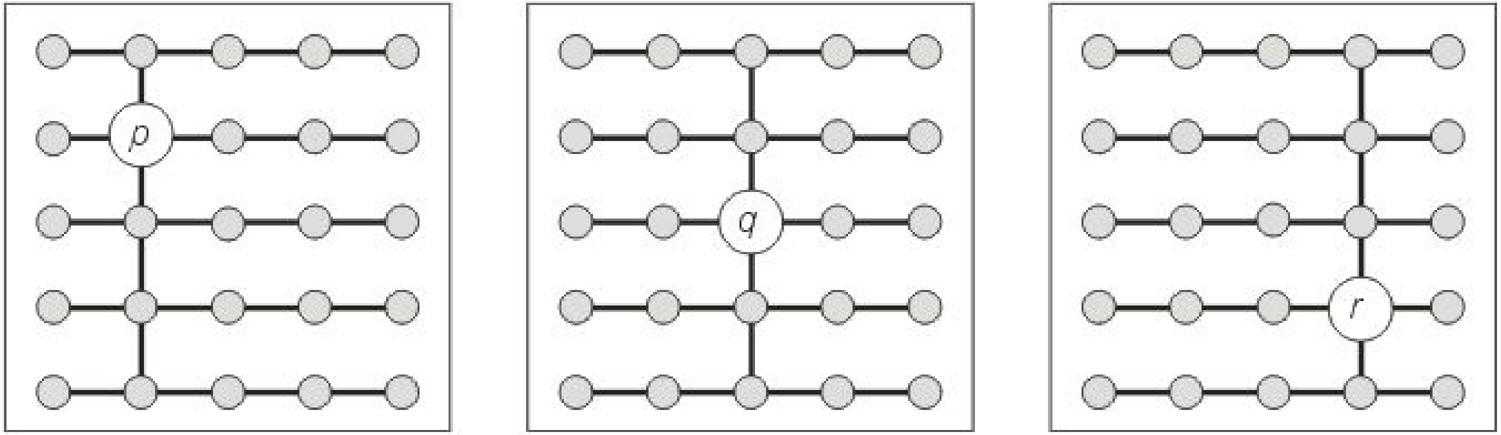
$$s(d_p, d_q) = \begin{cases} 0 & : \quad d_p = d_q \\ P_1 & : \quad |d_p - d_q| = 1 \\ P_2 & : \quad \text{otherwise.} \end{cases}$$

Weighted by intensity gradient

$$P_2 = \begin{cases} P_3 \cdot P_2' & : \quad |I_p - I_q| < T \\ P_2' & : \quad \text{otherwise} \end{cases}$$

Energy Optimization on Simple Trees

- Extremely large amount of different trees
- Tree DP on every tree is extremely slow

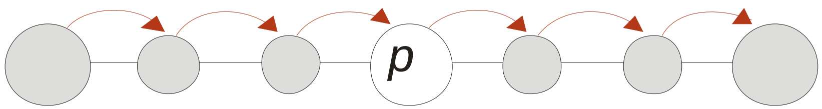


Incremental Computation of Horizontal Trees

- Optimize horizontal scanlines only
- Compute DP path costs for reaching each pixel *p* at each disparity *d* from left and right-most pixels of the scanline

Incremental Computation of Horizontal Trees

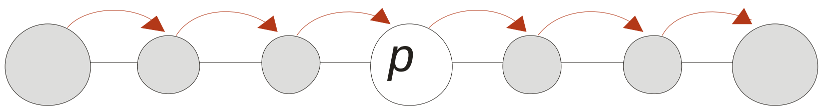
- Optimize horizontal scanlines only
- Compute DP path costs for reaching each pixel *p* at each disparity *d* from left and right-most pixels of the scanline



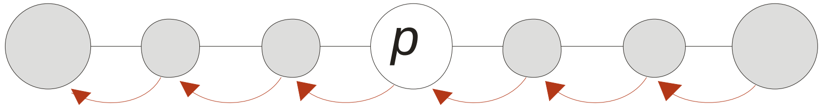
(Forward pass)

Incremental Computation of Horizontal Trees

- Optimize horizontal scanlines only
- Compute DP path costs for reaching each pixel *p* at each disparity *d* from left and right-most pixels of the scanline



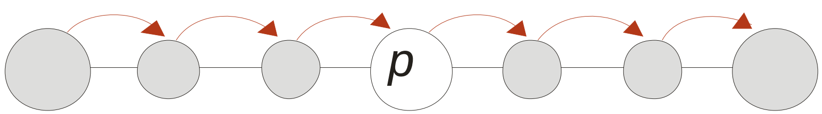
(Forward pass)



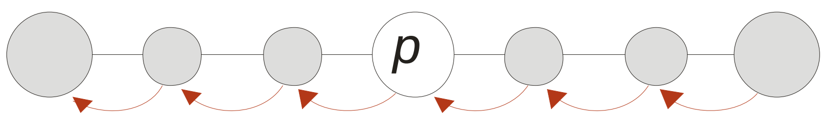
(Backward pass)

Incremental Computation of Horizontal Trees

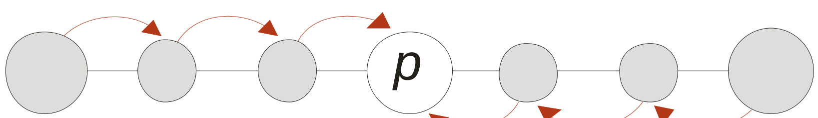
- Optimize horizontal scanlines only
- Compute DP path costs for reaching each pixel *p* at each disparity *d* from left and right-most pixels of the scanline



(Forward pass)



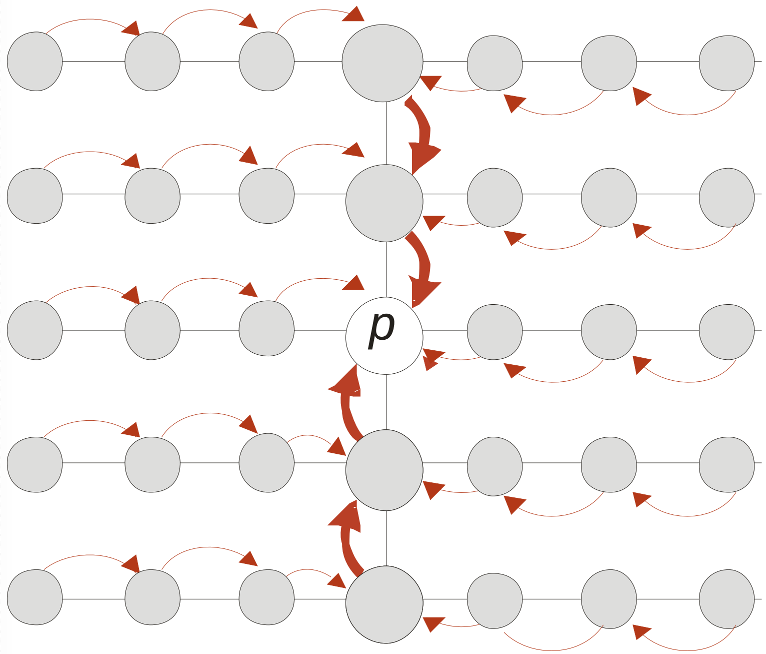
(Backward pass)



(Combining forward and backward passes)

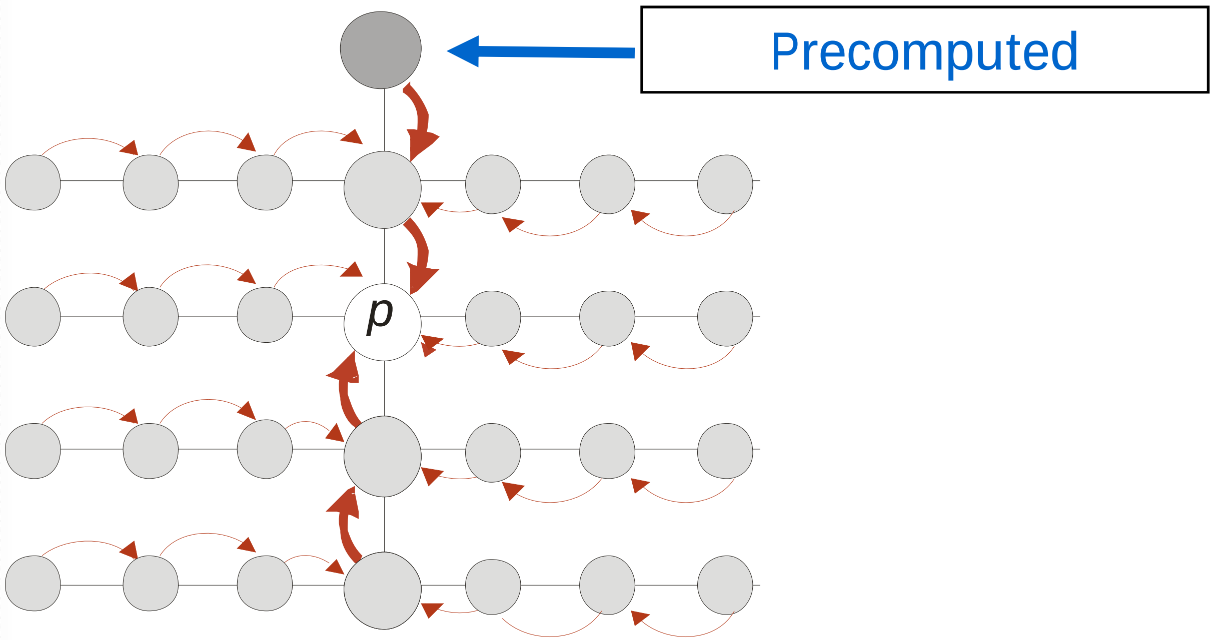
Incremental Computation of Horizontal Trees

- Compute path costs for reaching pixel *p* at disparity *d* from all leaf nodes



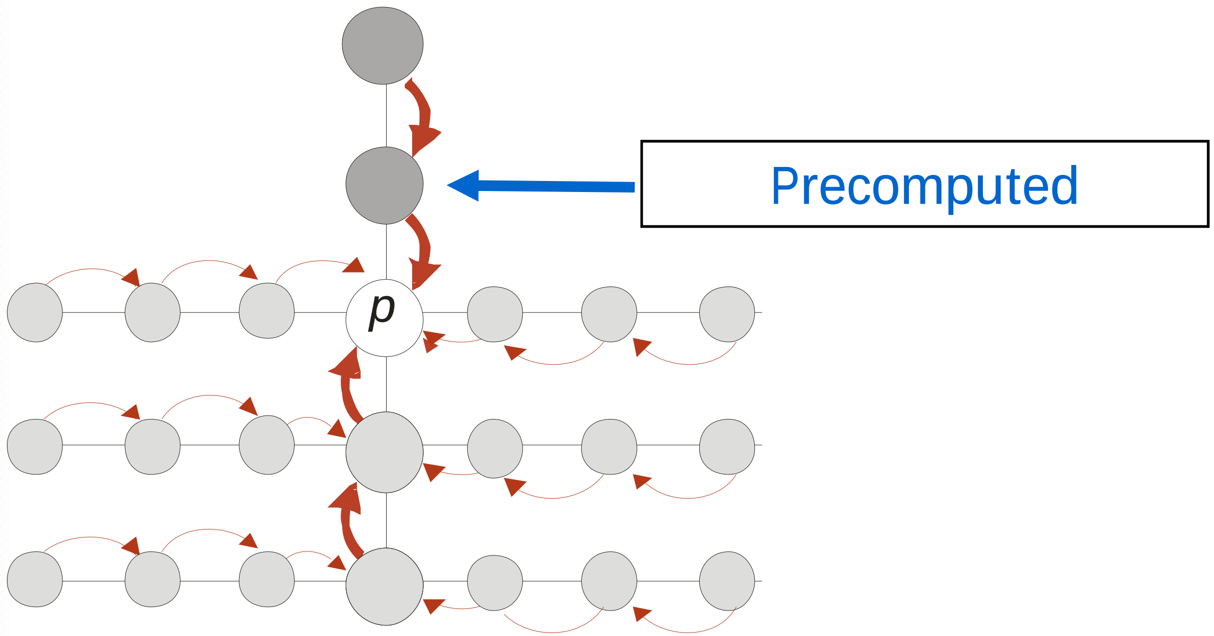
Incremental Computation of Horizontal Trees

- Compute path costs for reaching pixel *p* at disparity *d* from all leaf nodes



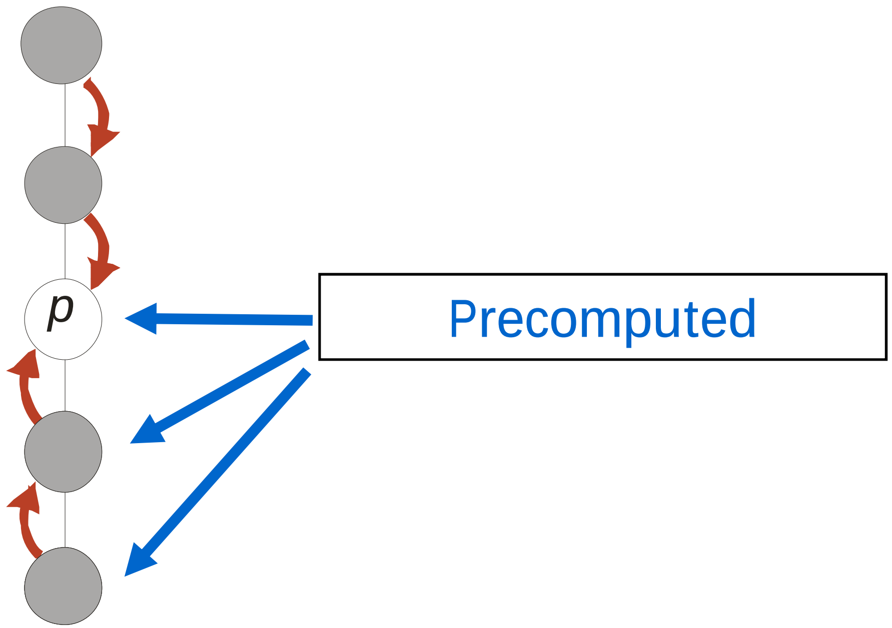
Incremental Computation of Horizontal Trees

- Compute path costs for reaching pixel *p* at disparity *d* from all leaf nodes



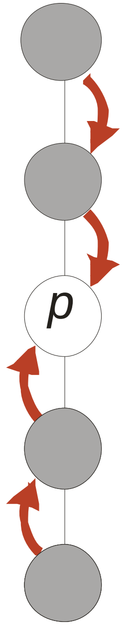
Incremental Computation of Horizontal Trees

- Compute path costs for reaching pixel *p* at disparity *d* from all leaf nodes



Incremental Computation of Horizontal Trees

- Compute path costs for reaching pixel *p* at disparity *d* from all leaf nodes

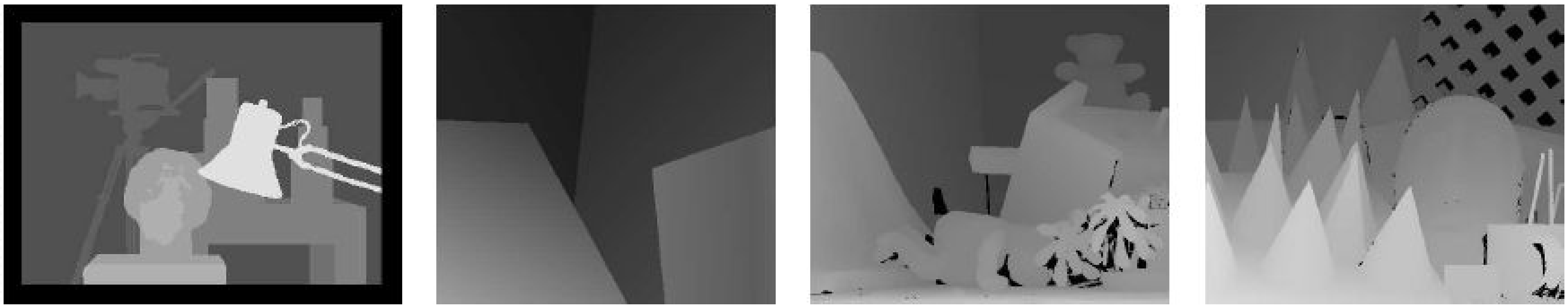


- Forward Path
- Backward Path
- Combination gives energy minima of all Horizontal Trees on this scanline

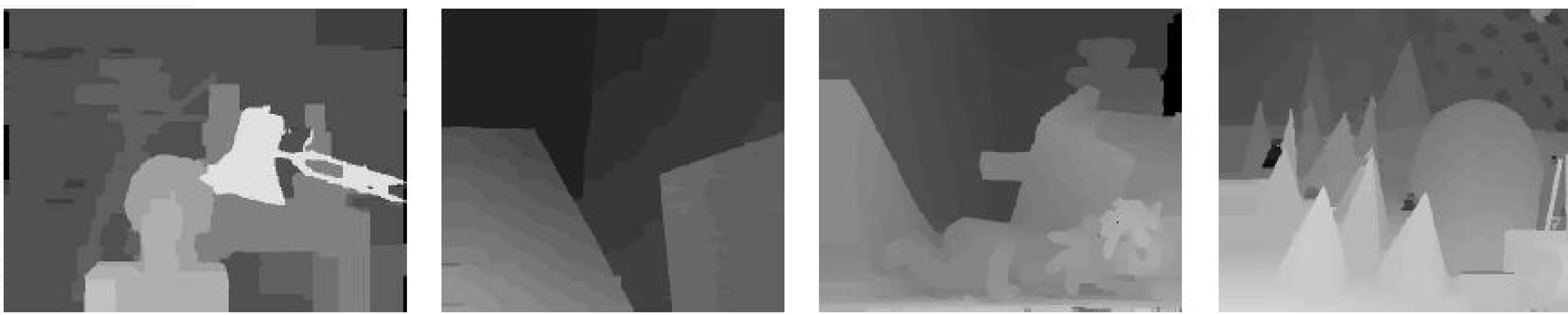
What I do not discuss in this talk (but in the paper)

- How are Horizontal and Vertical Trees combined in the algorithm?
- How are occlusions handeled?

Results on the Middlebury Evaluation Set



(Ground truth disparities)

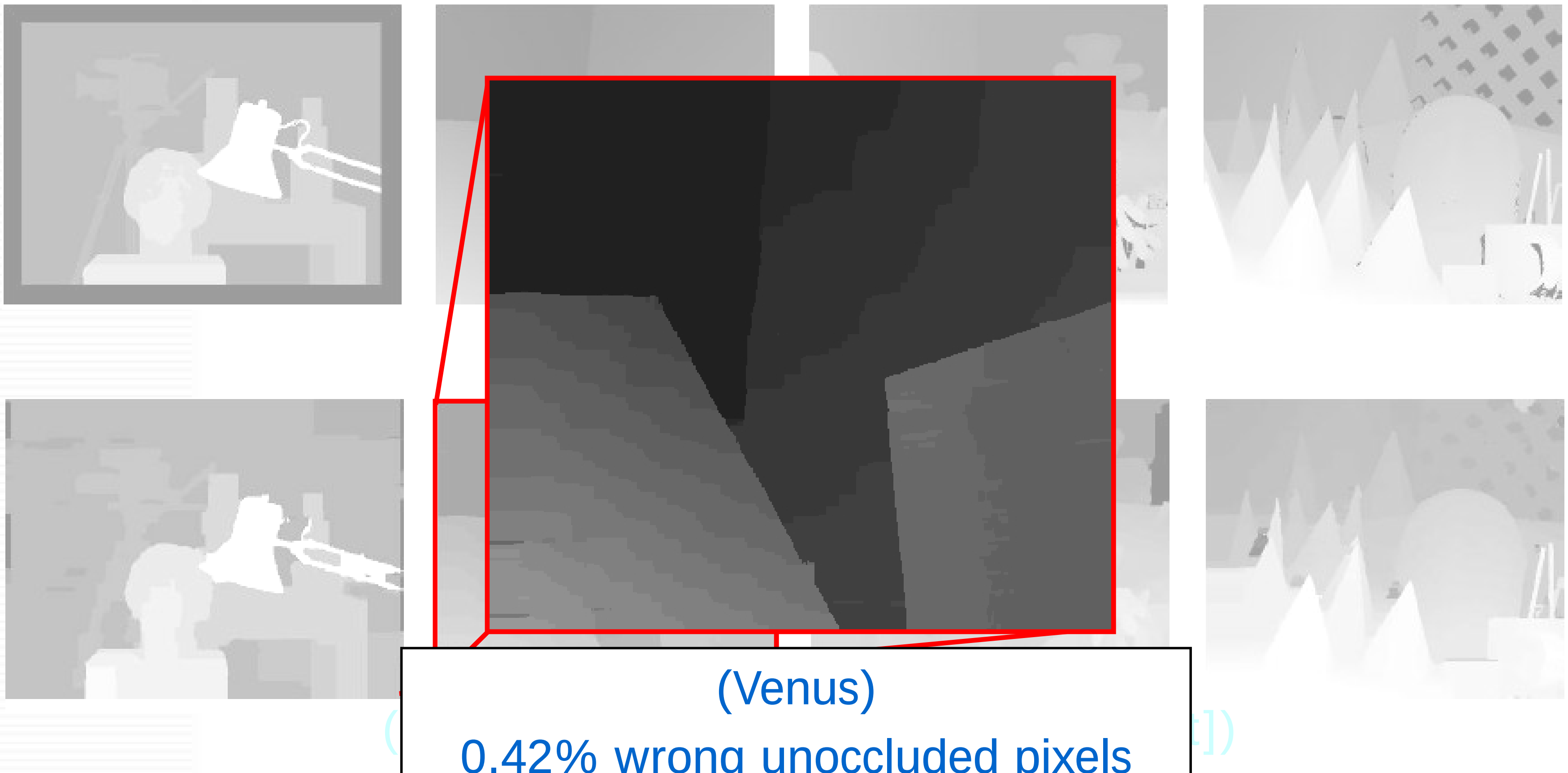


(Our results [Parameters kept constant])

Results on the Middlebury Evaluation Set

(Tsukuba)
1.86% wrong unoccluded pixels

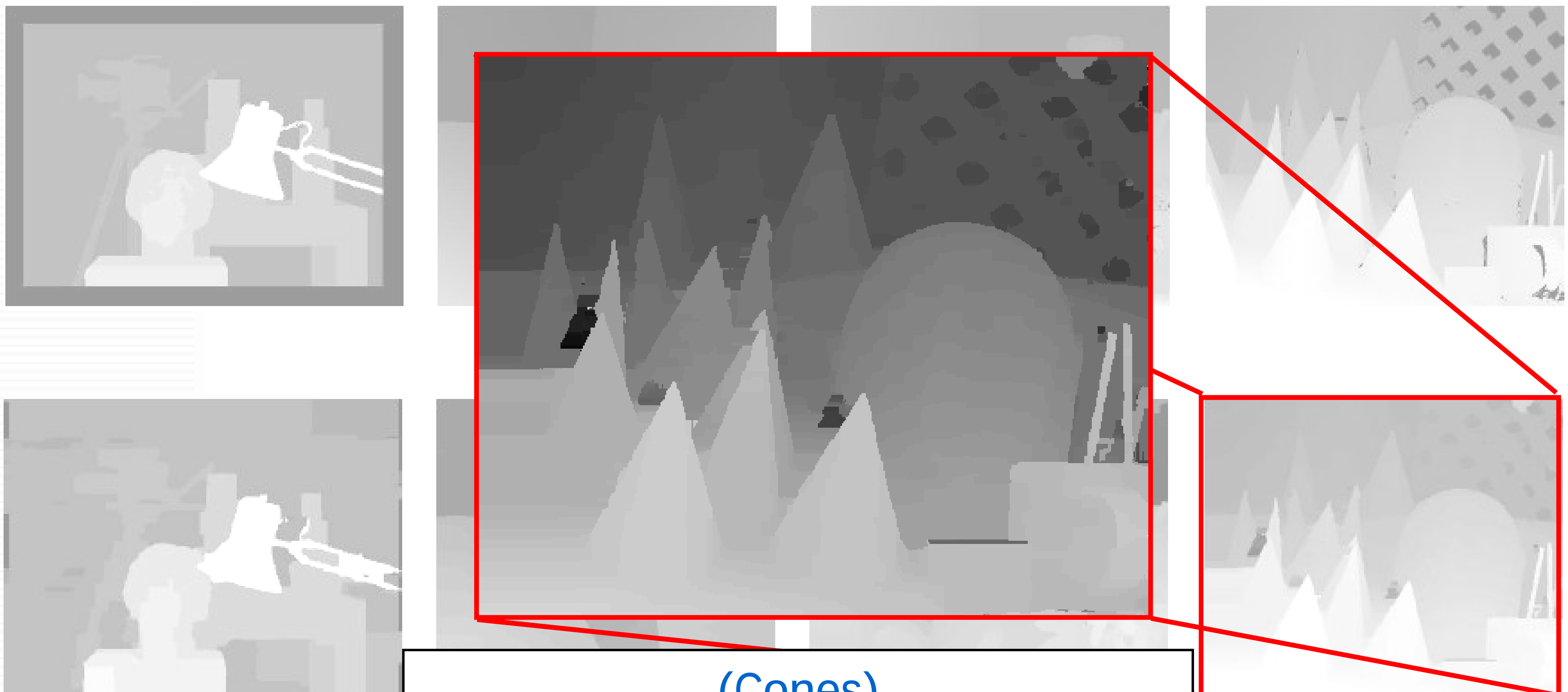
Results on the Middlebury Evaluation Set



Results on the Middlebury Evaluation Set

(Teddy)
7.31% wrong unoccluded pixels

Results on the Middlebury Evaluation Set



(Cones)
4.00% wrong unoccluded pixels

Middlebury Ranking

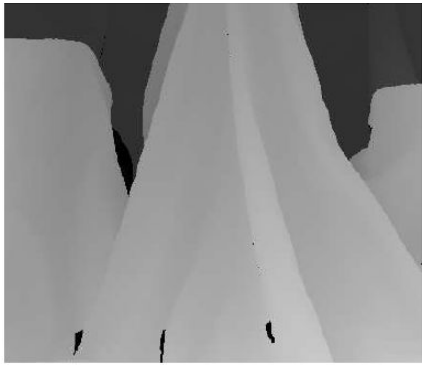
Algorithm	Rank	Tsukuba		Venus		Teddy		Cones	
		nocc	all	nocc	all	nocc	all	nocc	all
C-SemiGlob (Hirschmüller, 2006)	5	2.61	3.29	0.25	0.57	5.14	11.8	2.77	8.35
RegionTreeDP (Lei et al., 2006)	6	1.39	1.64	0.22	0.57	7.42	11.9	6.31	11.9
SimpleTree	**8**	**1.86**	**2.56**	**0.42**	**0.76**	**7.31**	**12.7**	**4.00**	**9.74**
SegTreeDP (Deng and Lin, 2006)	10	2.21	2.76	0.46	0.60	9.58	15.2	3.23	7.86
SemiGlob (Hirschmüller, 2005)	12	3.26	3.96	1.00	1.57	6.02	12.2	3.06	9.75
RealTimeGPU (Wang et al., 2006)	19	2.05	4.22	1.92	2.98	7.23	14.4	6.41	13.7
ReliabilityDP (Gong and Yang, 2005)	21	1.36	3.39	2.35	3.48	9.82	16.9	12.9	19.9
TreeDP (Veksler, 2005)	22	1.99	2.84	1.41	2.10	15.9	23.9	10.0	18.3
DP (Scharstein and Szeliski, 2002)	24	4.12	5.04	10.1	11.0	14.0	21.6	10.5	19.1
SO (Scharstein and Szeliski, 2002)	27	5.08	7.22	9.44	10.9	19.9	28.2	13.0	22.8

- Rank 8 of ~30 algorithms in online table
- Computationally more efficient than better-ranked methods
- Best-performing non-segmentation-based algorithm

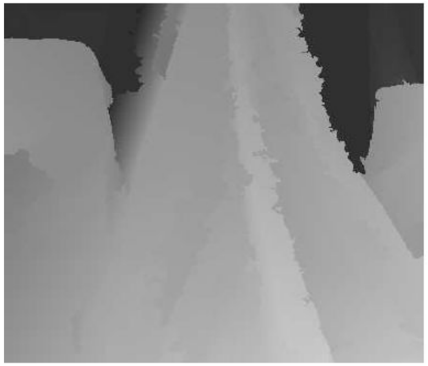
Why we did not use Colour Segmentation

- Segmentation is computationally expensive
- Works fine for Middlebury set, but not in general

(Left image)



(Ground truth)



(SegGlobVis [Bleyer04])

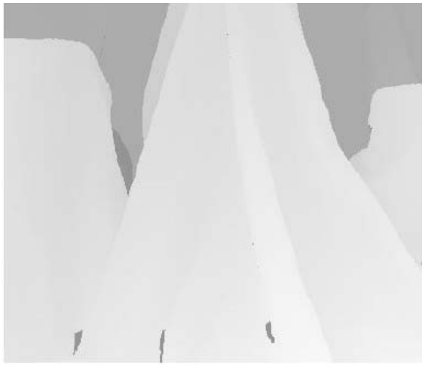
(SimpleTree)

Why we did not use Colour Segmentation

- Segmentation is computationa... expensive
- Works fine fo... Middlebury s... but not in ge...

(Left image)

...e)



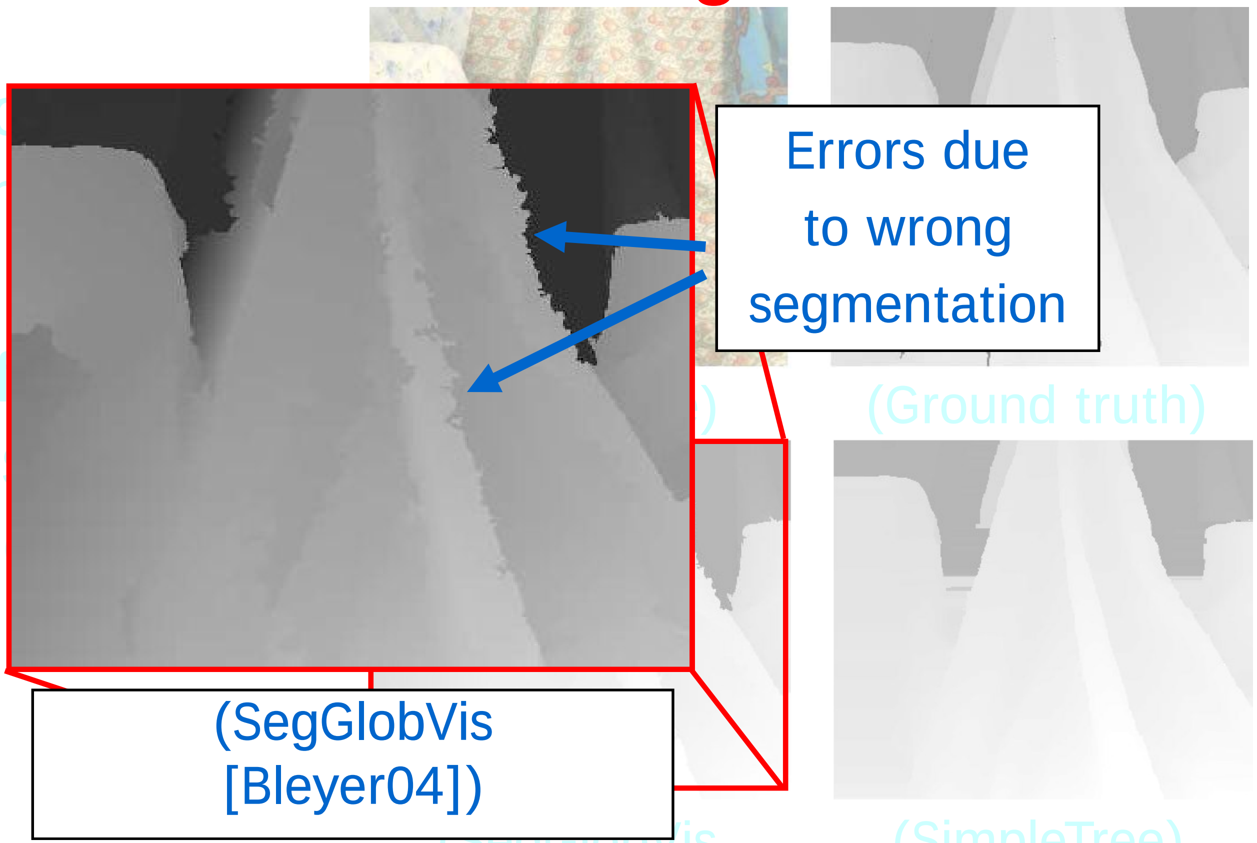
(Ground truth)

(SegGlobVis [Bleyer04])

(SimpleTree)

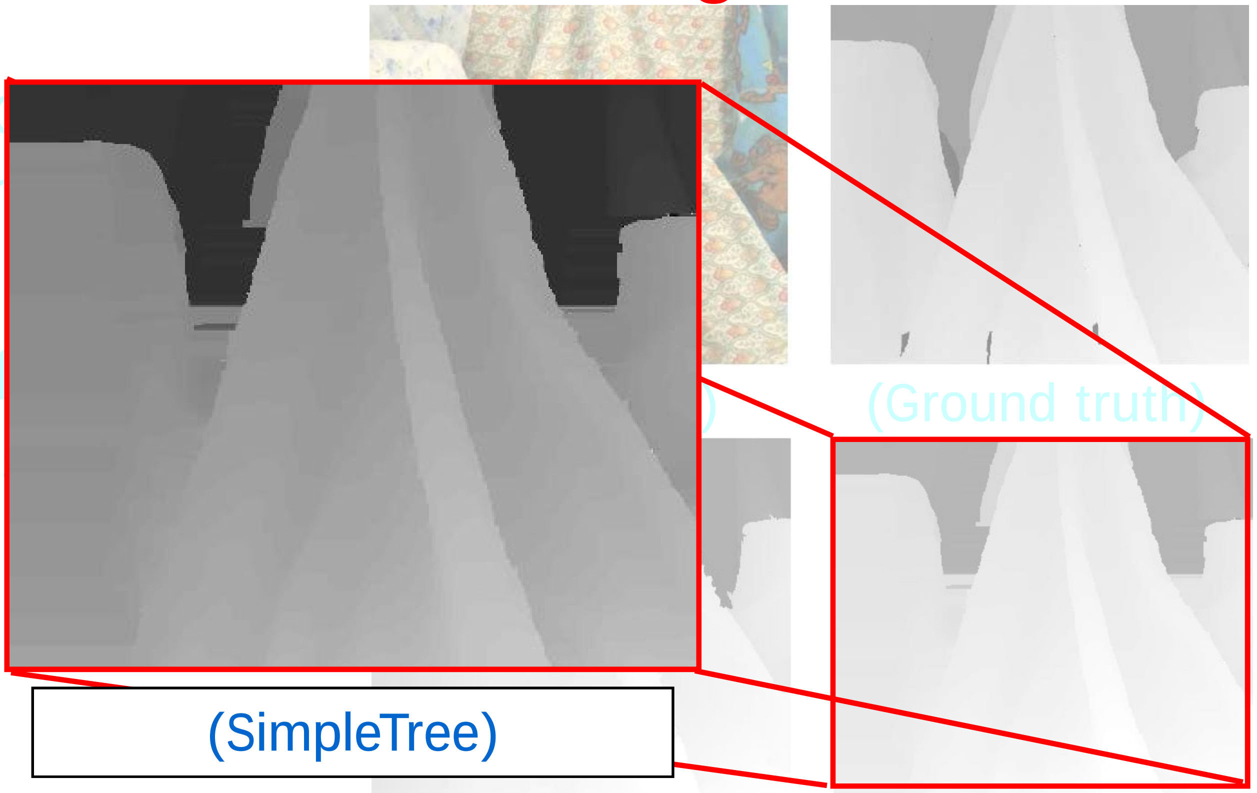
Why we did not use Colour Segmentation

- Segmentatio
  computation
  expensive
- Works fine f
  Middlebury s
  but not in g



Why we did not use Colour Segmentation

- Segmentati[...] computatio[...] expensive
- Works fine [...] Middlebury [...] but not in g[...]

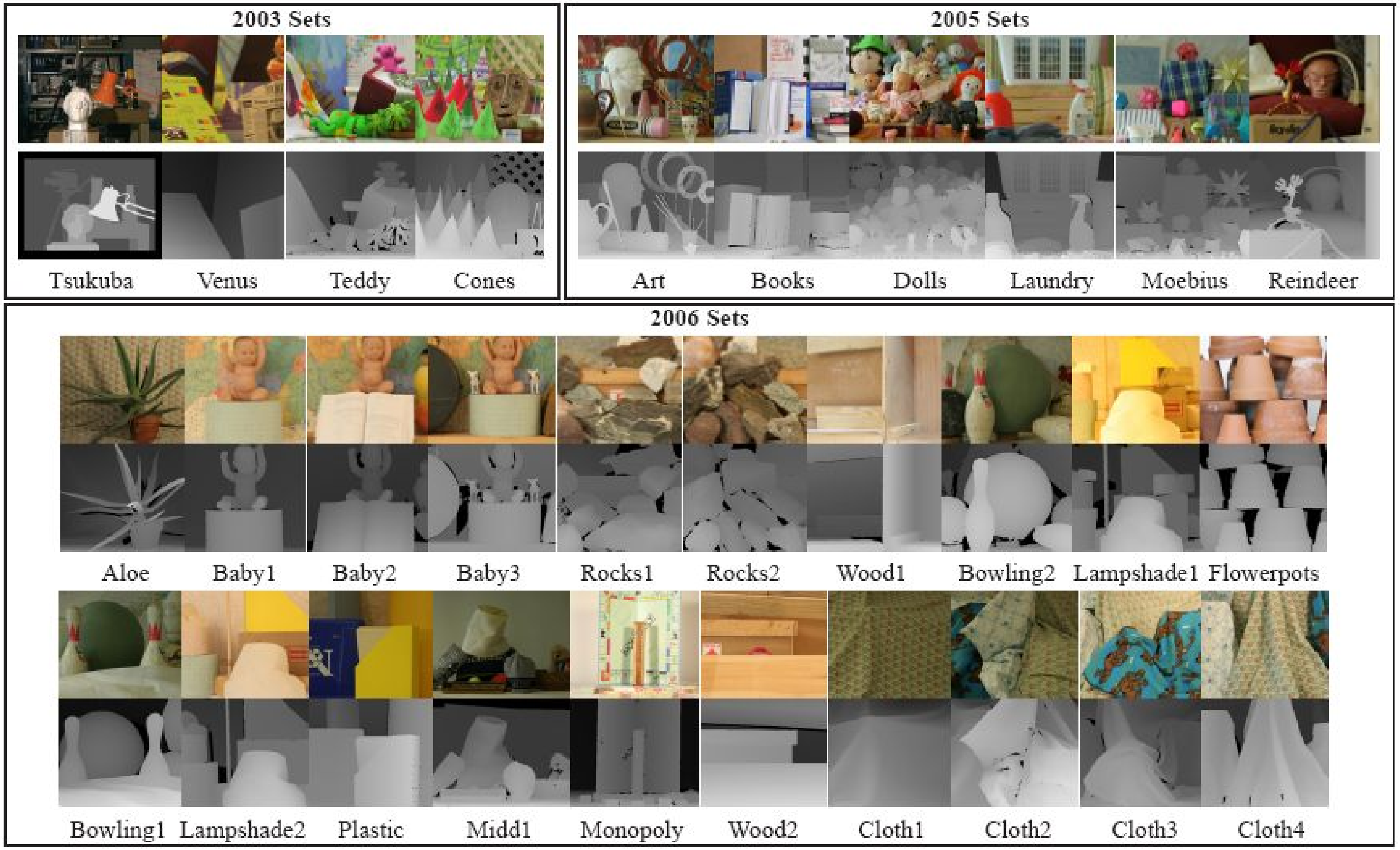


Computational Performance [given in sec]

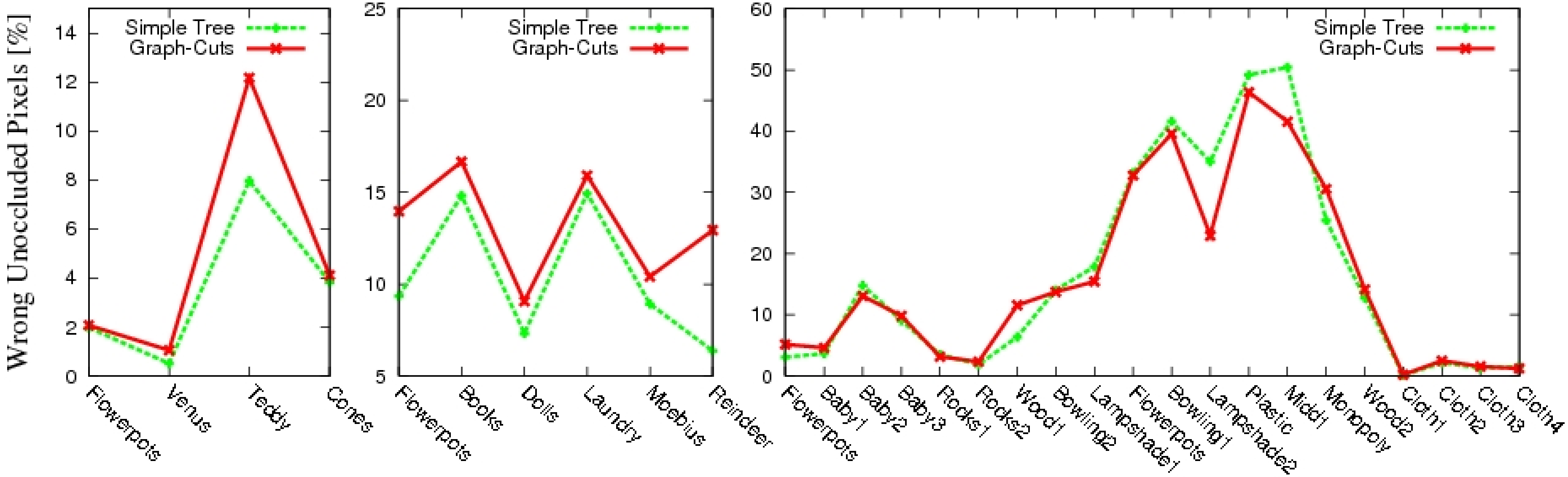
Data Set	Size	Range	1 CPU	4 CPUs
Tsukuba	384×288	0–15	0.45	0.18
Venus	434×383	0–19	0.70	0.23
Teddy	450×375	0–59	1.54	0.57
Cones	450×375	0–59	1.54	0.57

- Potential for real-time performance

The new Middlebury Data Sets



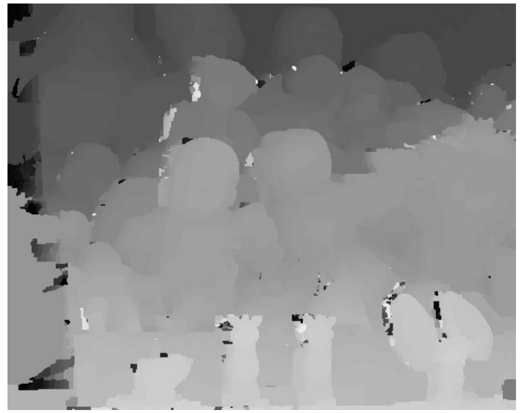
Comparison against Graph-Cuts



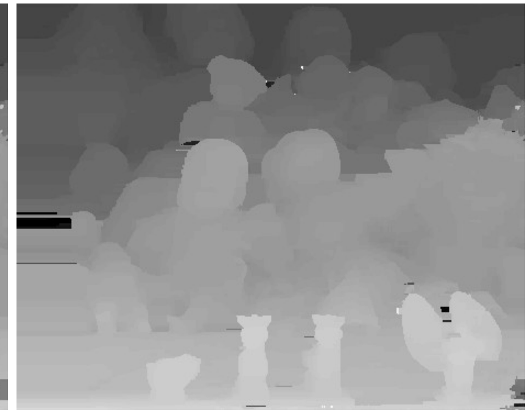
- Simple Tree outperforms Graph-Cuts on 20 of 30 stereo pairs
- Simple Tree is significantly faster.

The Dolls Test Set

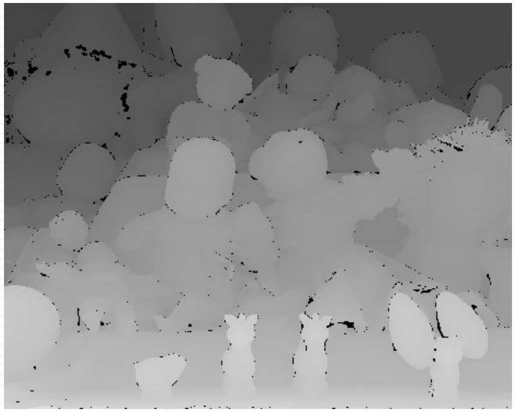
(Left image)



(Disparity Graph-Cuts)

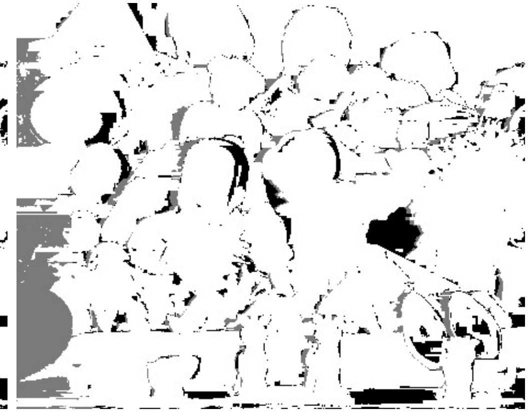


(Disparity Simple Tree)



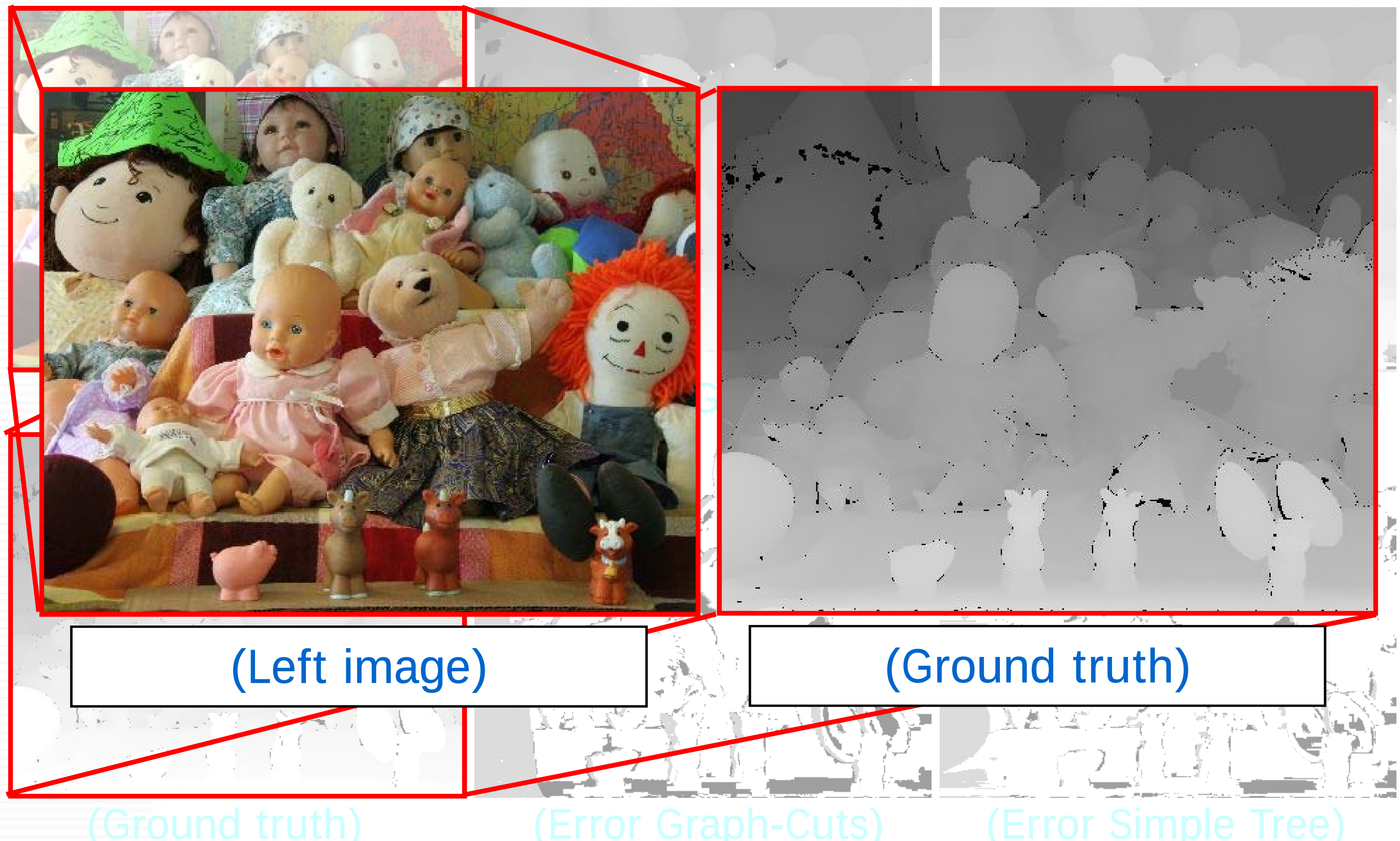
(Ground truth)

(Error Graph-Cuts)

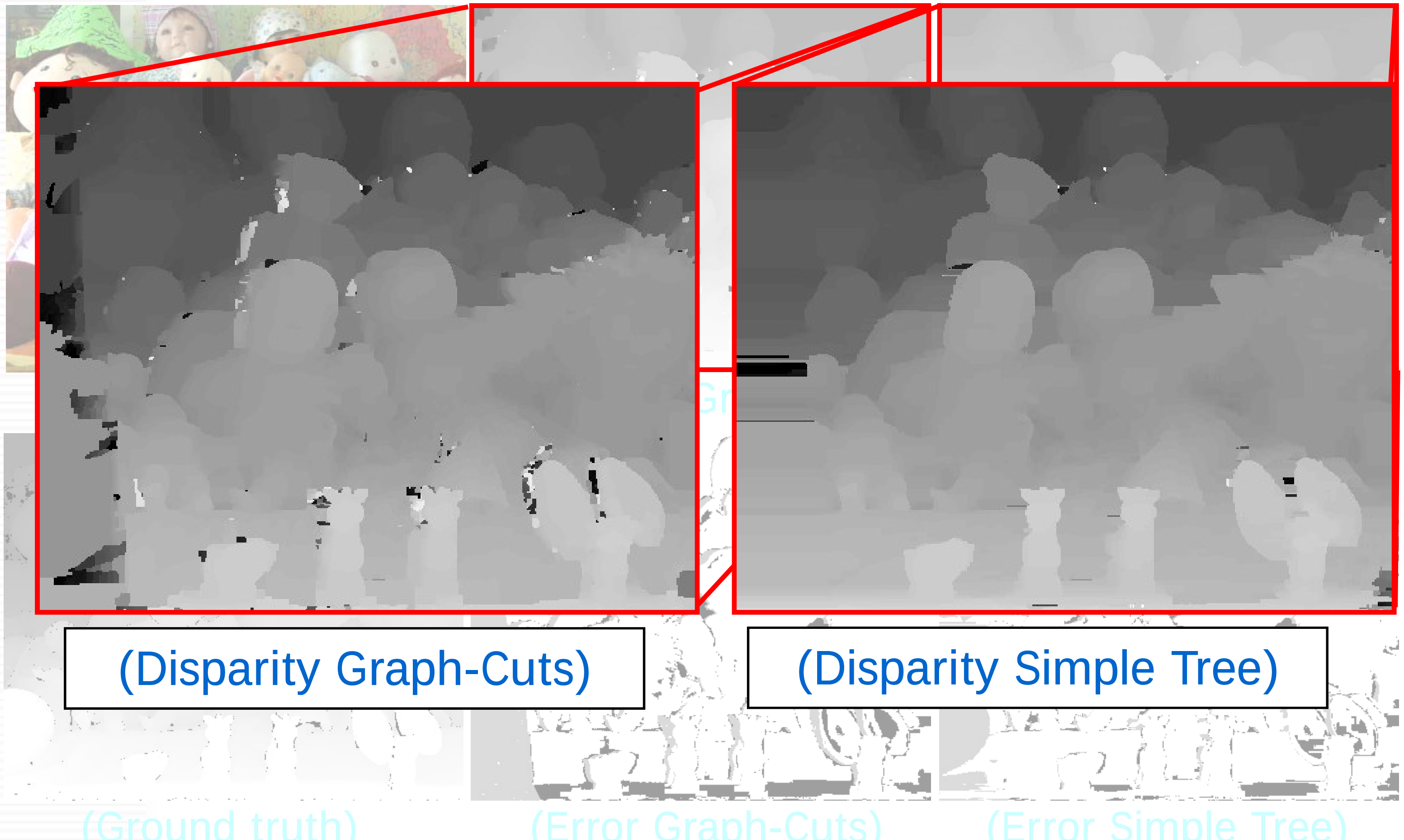


(Error Simple Tree)

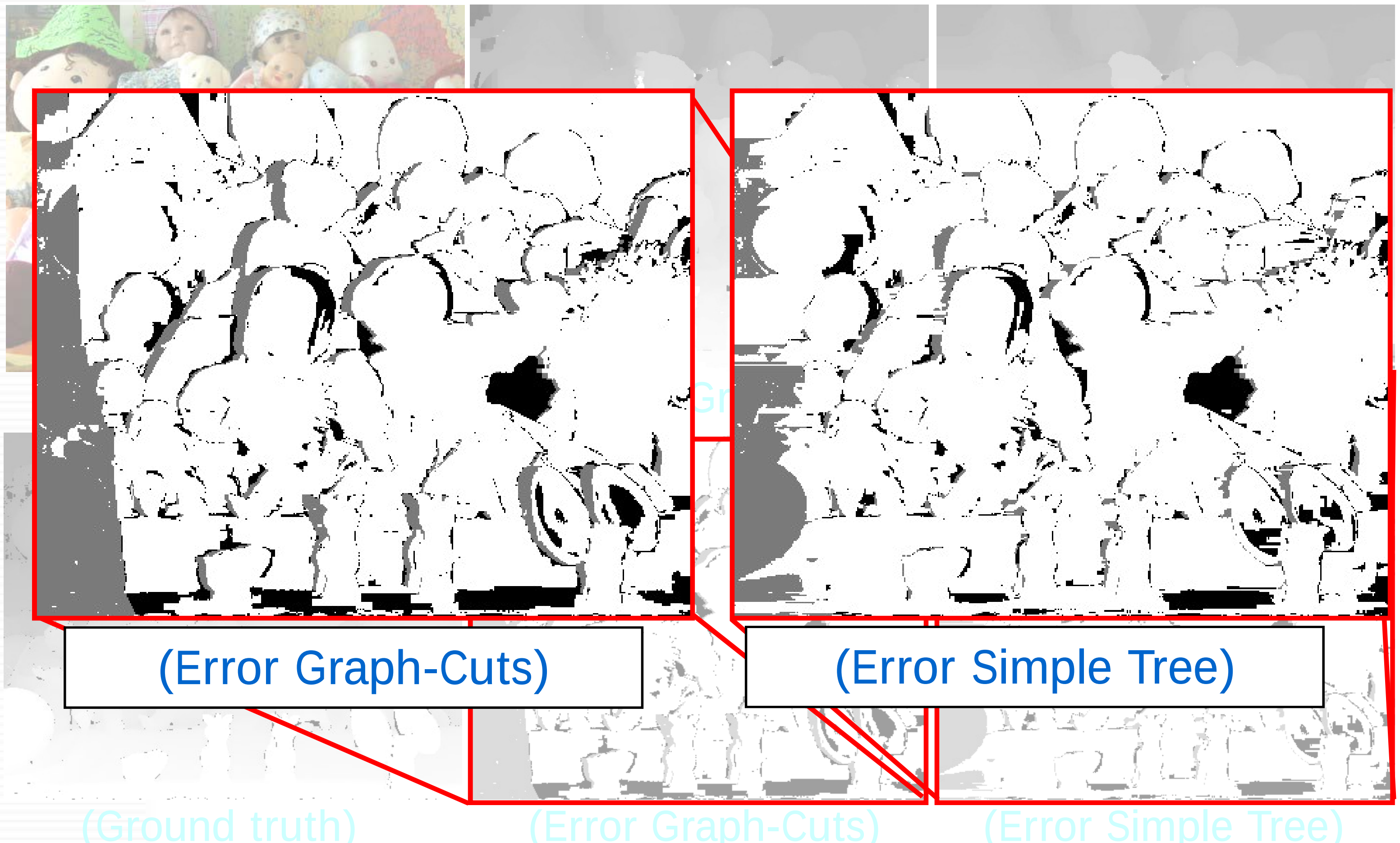
The Dolls Test Set



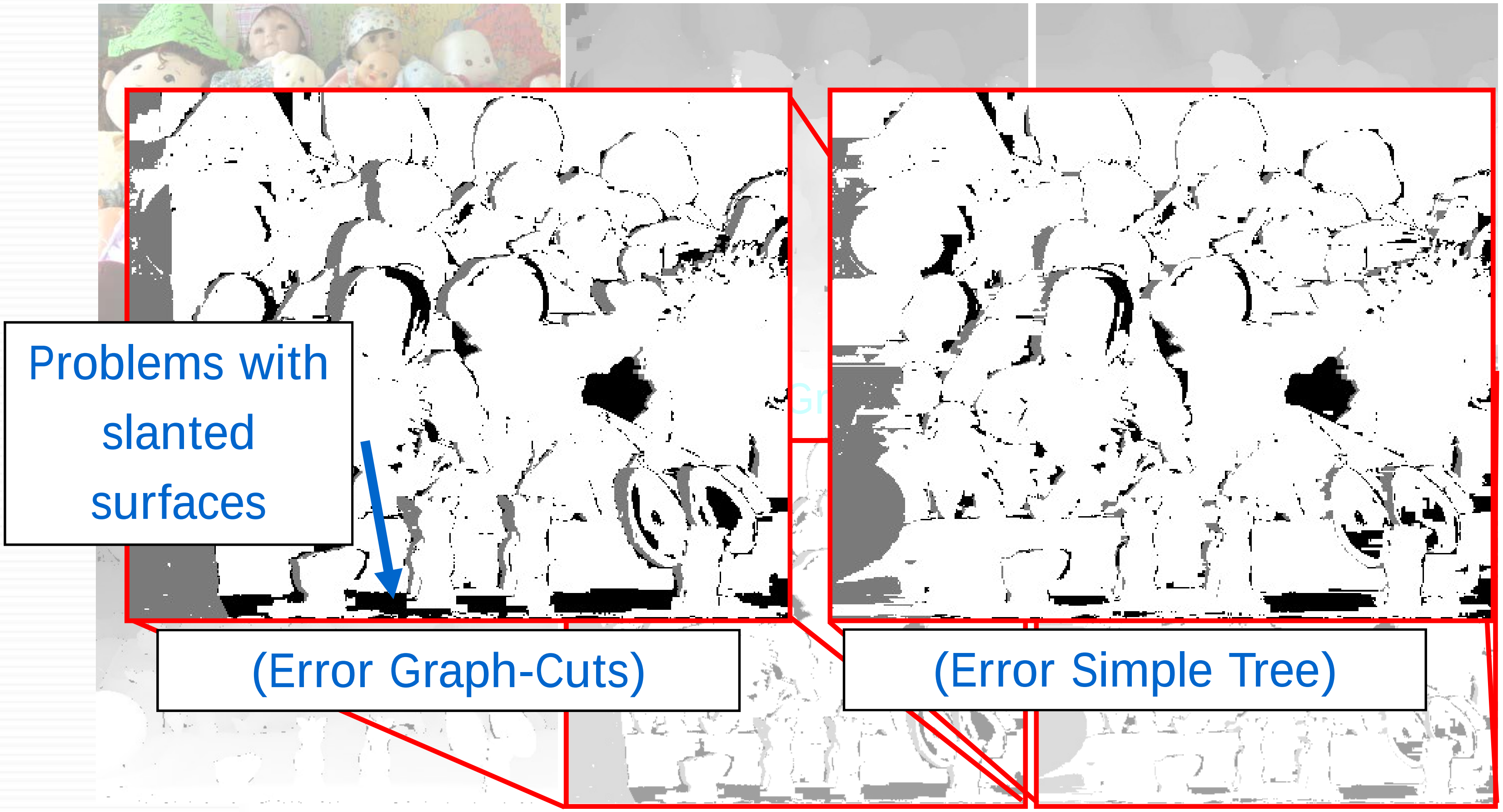
The Dolls Test Set



The Dolls Test Set



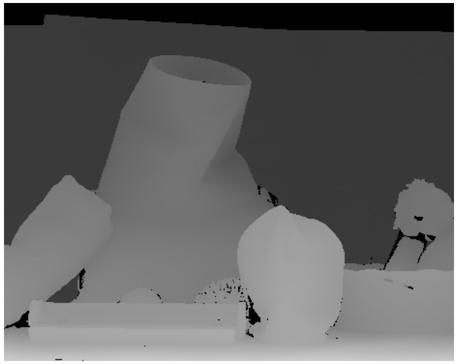
The Dolls Test Set



(Ground truth) (Error Graph-Cuts) (Error Simple Tree)

The Midd1 Test Set

(Left Image)



(Ground Truth)

(Graph-Cuts)

(Simple Tree)

- Graph-Cuts perform better on images with extremely low texture

Conclusions

- Compute disparity map by solving an optimization problem for each pixel
- Approximation of 4-connected grid via a tree
- Horizontal and Vertical Trees allow for fast computation
- Results almost free of scanline streaks
- Best-performing method in the Middlebury ranking that does not use colour segmentation
- Can represent a fast alternative to Graph-Cuts when speed matters

The End

Thank You