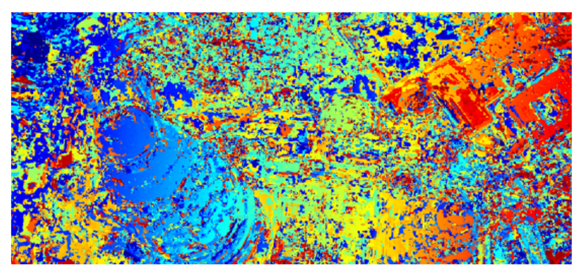
Computing Nearest-Neighbor Fields via Propagation-Assisted KD-Trees

Kaiming He and Jian Sun
Microsoft Research Asia

Introduction

- Computing Nearest-Neighbor Fields (NNF) is to densely match patches between two images.

Source Image A Target Image B

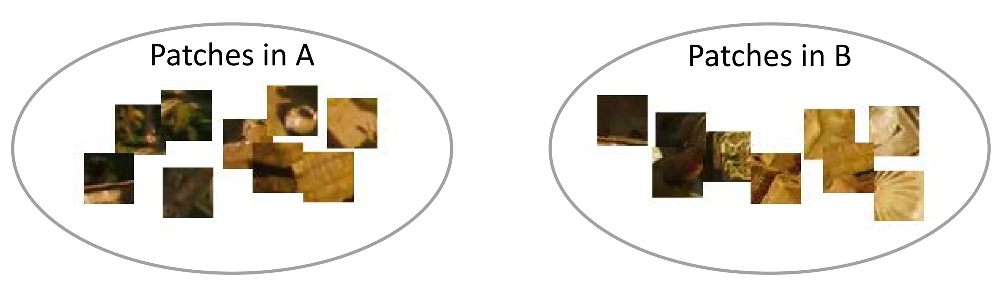


Nearest-Neighbor Field

- The applications of NNF: image inpainting/retargeting [1], texture synthesis, super-resolution, denoising, etc.
- Contributions: a fast and accurate method for computing NNF.

Overview

- *Our perspective*: ANN search for *multiple* but *dependent* queries.



Query Set Candidate Set

- *Methodology comparisons*:

Method	Strategy in Query Set	Strategy in Candidate Set
Traditional LSH *(Locality Sensitive Hashing)*	*None*	*Data-independent*
Traditional kd-tree	*None*	*Data-adaptive*
PatchMatch [1]	*Propagation*	*Random Sampling*
CSH [2] *(Coherency Sensitive Hashing)*	*Propagation*	*Data-independent (LSH)*
Ours	*Propagation*	*Data-adaptive (kd-tree)*

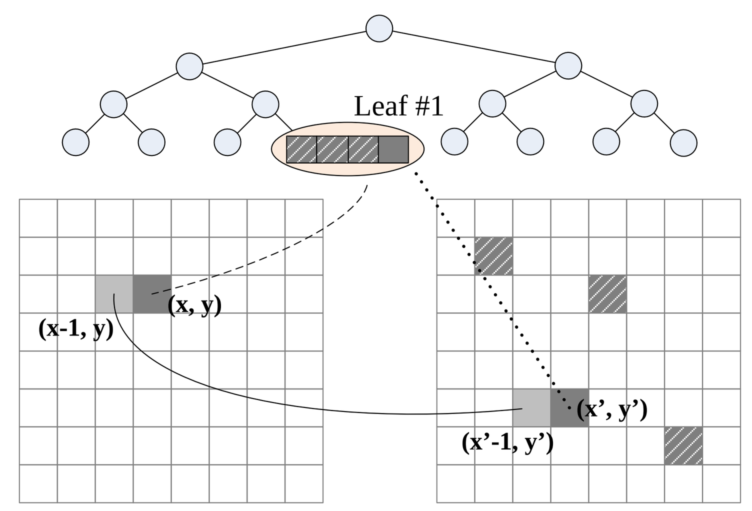
- *Our advantage*: better utilizes both the dependency of the patches in A and the dependency of the patches in B.

Algorithm

- **Basic Algorithm**

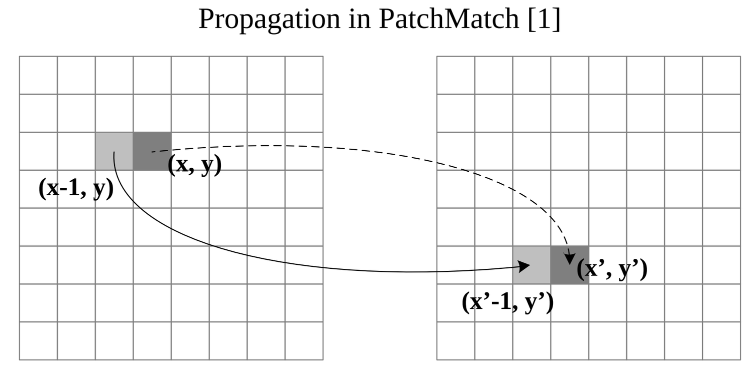
1. Build a kd-tree using all the patches in the Candidate Set. Each leaf contains no more than m candidates.
2. Scan the image A in raster order. For each patch $p_A(x, y)$ in A, do *propagation-assisted kd-tree search as follows*:
   i. Descend the tree to a leaf (Leaf #0);
   ii. Propagate a leaf from left/upper, using the already matched result of $p_A(x\text{-}1, y)$ and $p_A(x, y\text{-}1)$;
   iii. Find the NN of $p_A(x, y)$ in all the above leaves.

Propagation-Assisted KD-tree

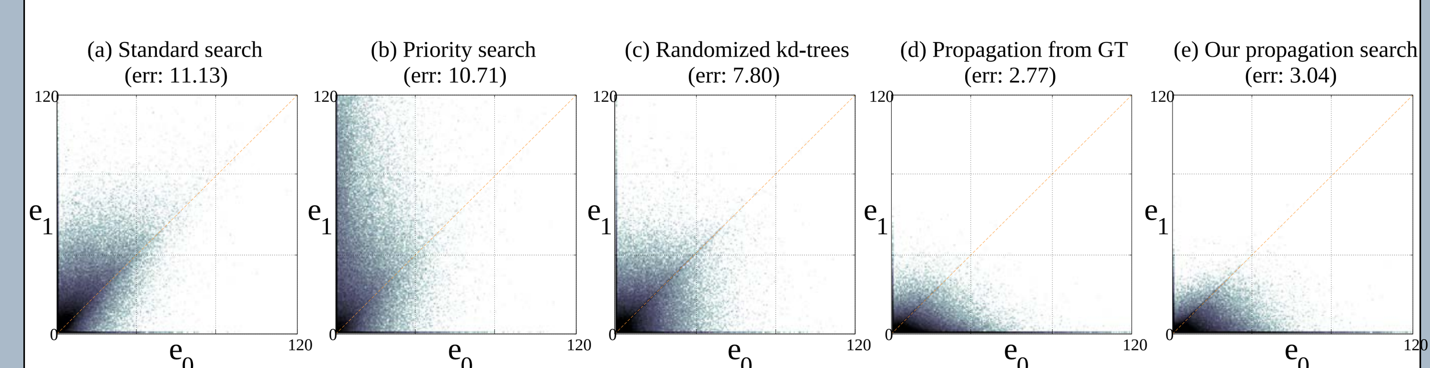


- Method Comparison: PatchMatch [1]

Propagation in PatchMatch [1]



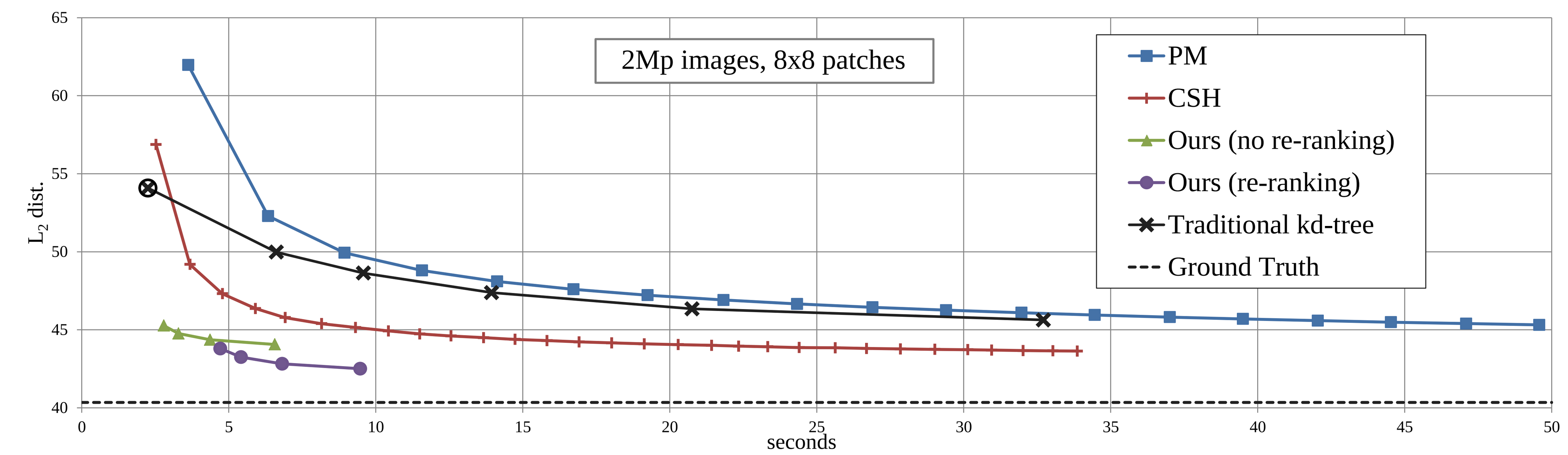
- Method Comparisons: kd-tree



e_0: error of Leaf #0; e_1: error of Leaf #1.
Leaf #1 is given by backtracking (standard/priority search), another random tree (randomized kd-tree), or propagation (ours).

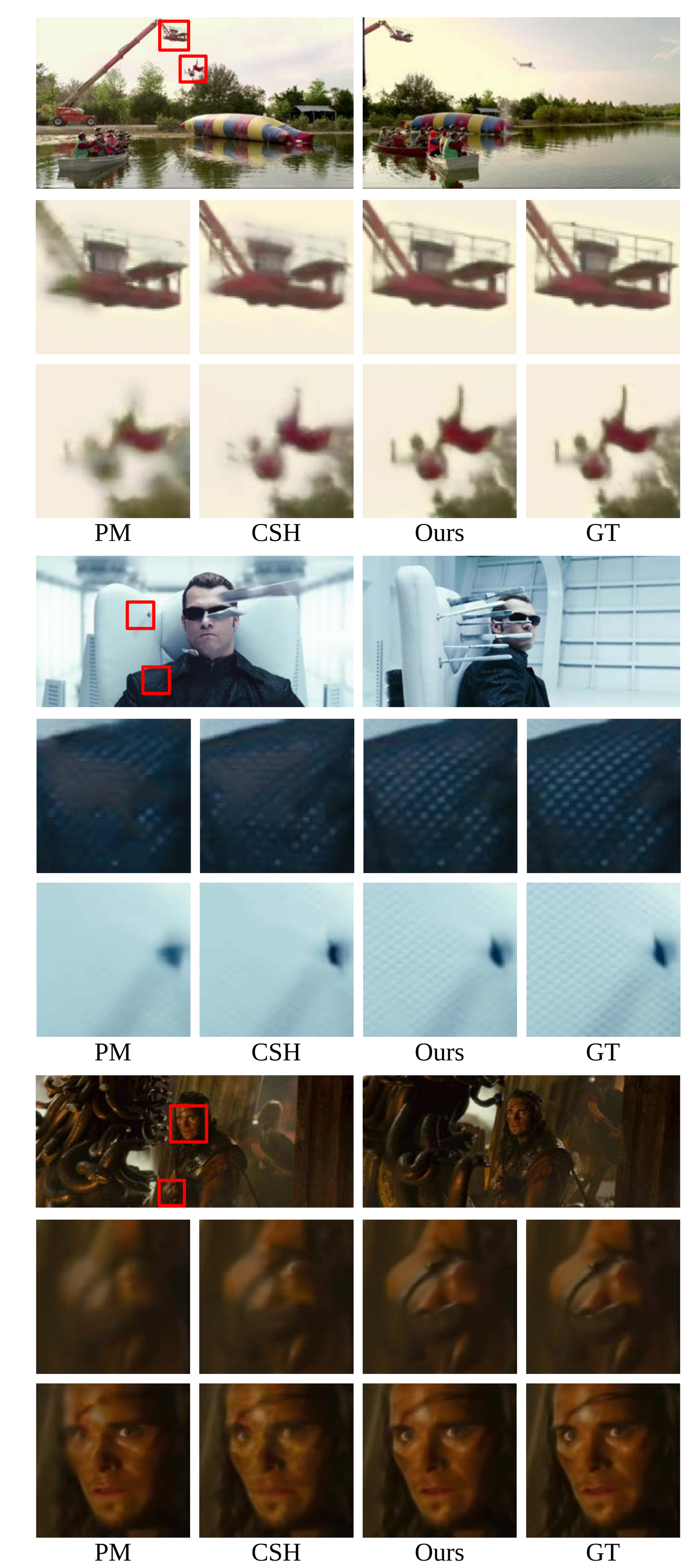
Results

- **Performance in VidPairs Benchmark [2]** (Accuracy vs. Time, 133 image pairs)



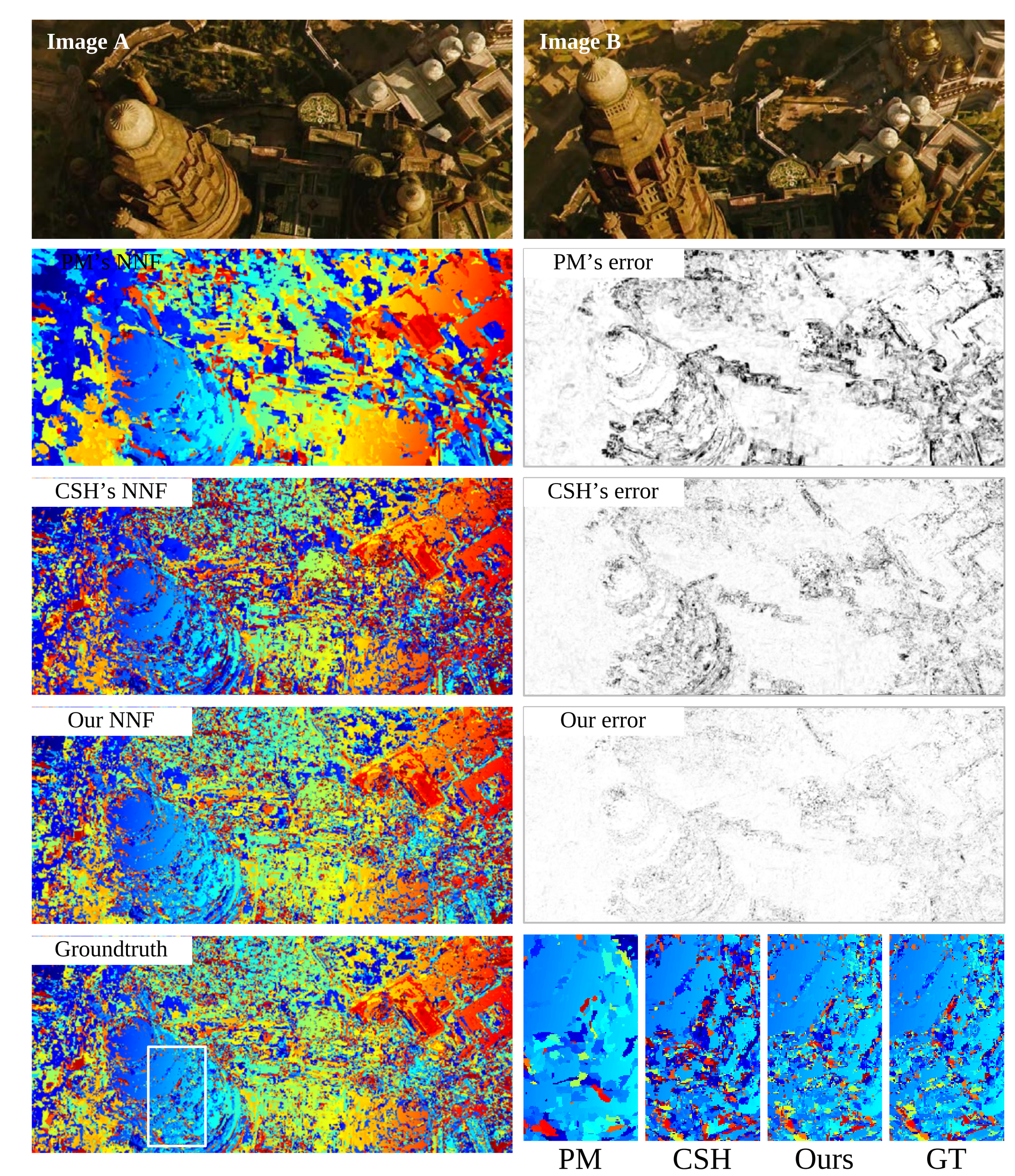
Observations:
1. 10-20x faster vs. PatchMatch [1], 2-5x faster vs. CSH [2], at the same accuracy.
2. 70% smaller error vs. PatchMatch, 50% smaller error vs. CSH, at the same running time.
3. Traditional kd-trees can be comparable with PatchMatch.

Reconstructed Images



- **Visual Comparisons**

NNFs and error maps



References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. *Patchmatch: a randomized correspondence algorithm for structural image editing*. In SIGGRAPH, 2009
[2] S. Korman and S. Avidan. *Coherency sensitive hashing*. In ICCV, 2011.