EFFICIENT STEREO DEPTH ESTIMATION USING PATCHMATCH

Tyler Baumgartner

Stereo Image Depth Estimation

- Similar to how we estimate depth based on **disparities** between our **left** and **right** eyes.
- Pixels can be mapped between left and right images
- Open Questions:
  - *How do we properly match pixels?*
  - *What if a pixel in the left is obstructed in the right (or vice versa)?*
  - *How do we discriminate between identical pixels?*
  - *How do minor shifts in lighting alter output?*

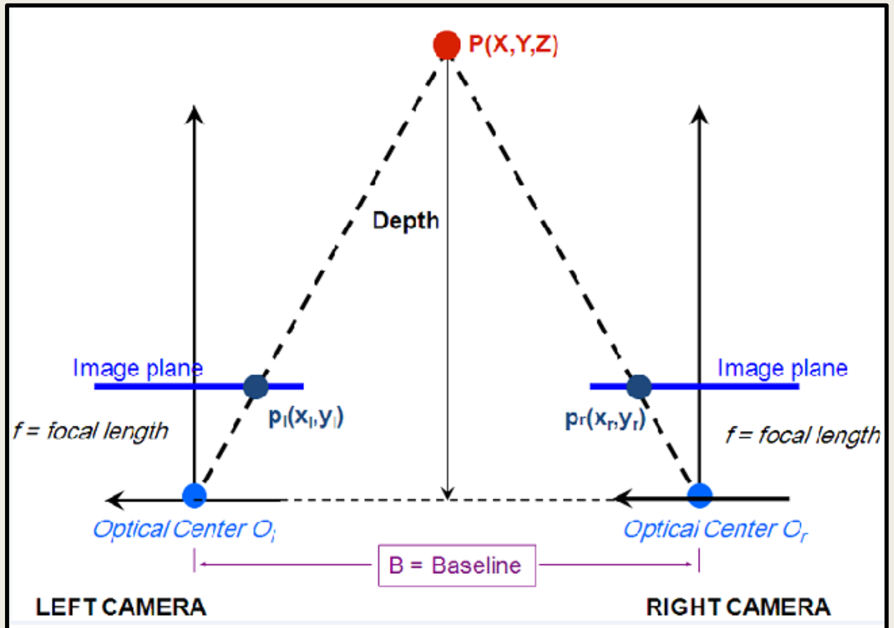
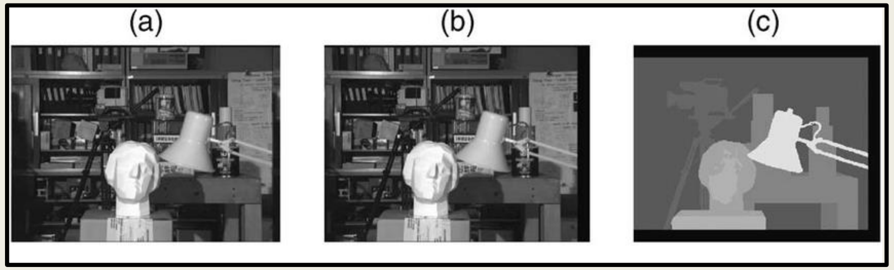
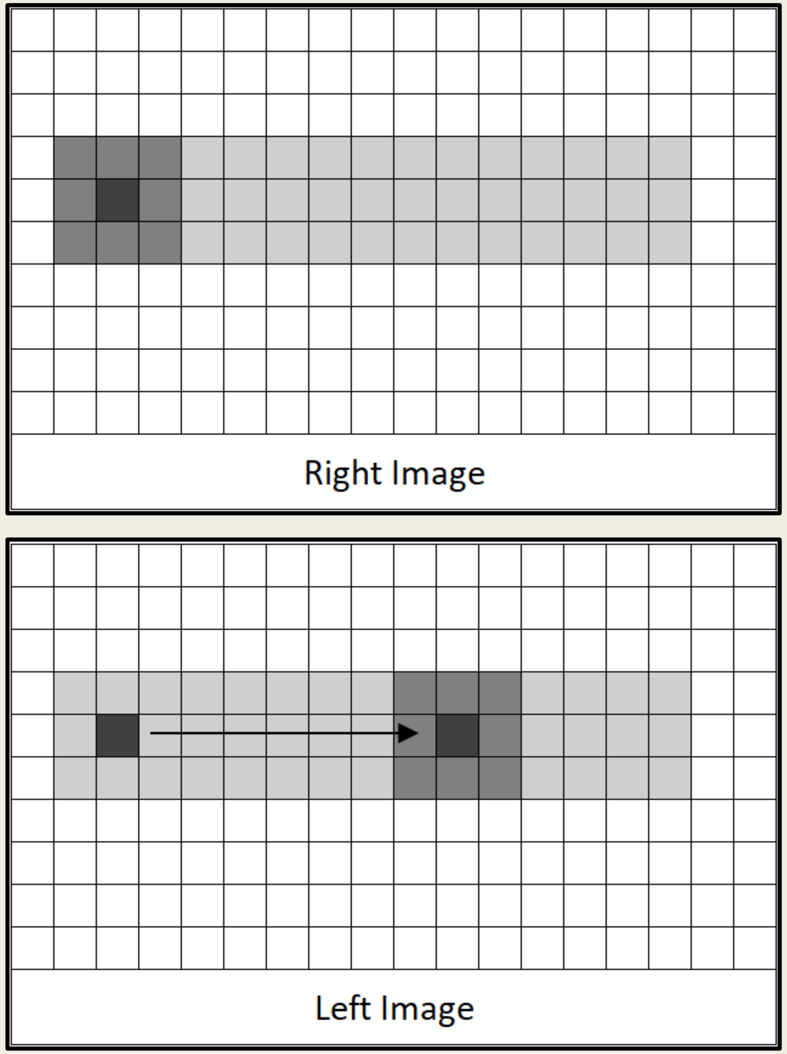


Fig. 2. Stereo vision basics

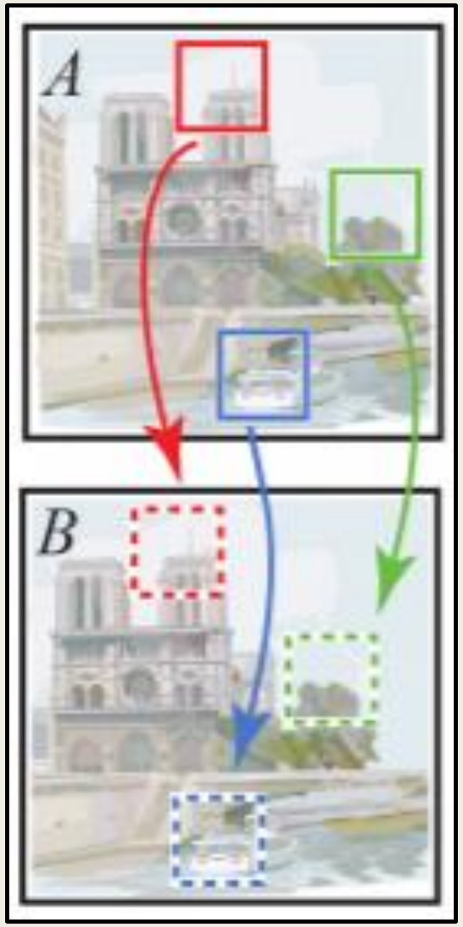


Basic Block Matching

- Advantages:
  - *Quick*
  - *Low memory usage*
- Disadvantages:
  - *No global scope*
  - *No refining process*
- Can be used to “prune” search space



PatchMatch



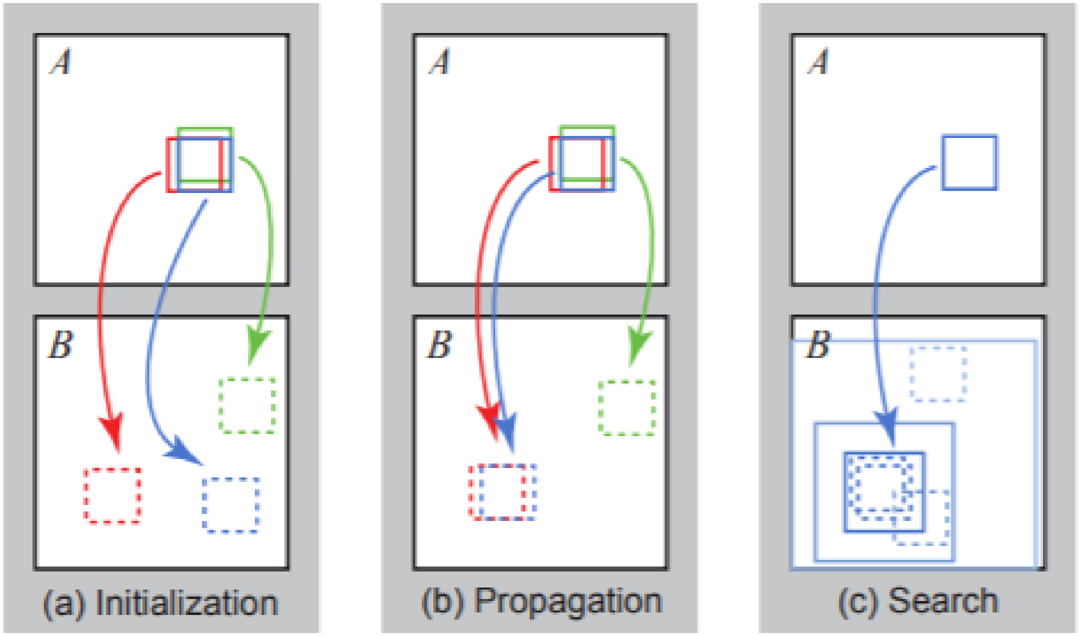


Figure 2: *Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods.*

DeepPruner – Merging the Two Ideas

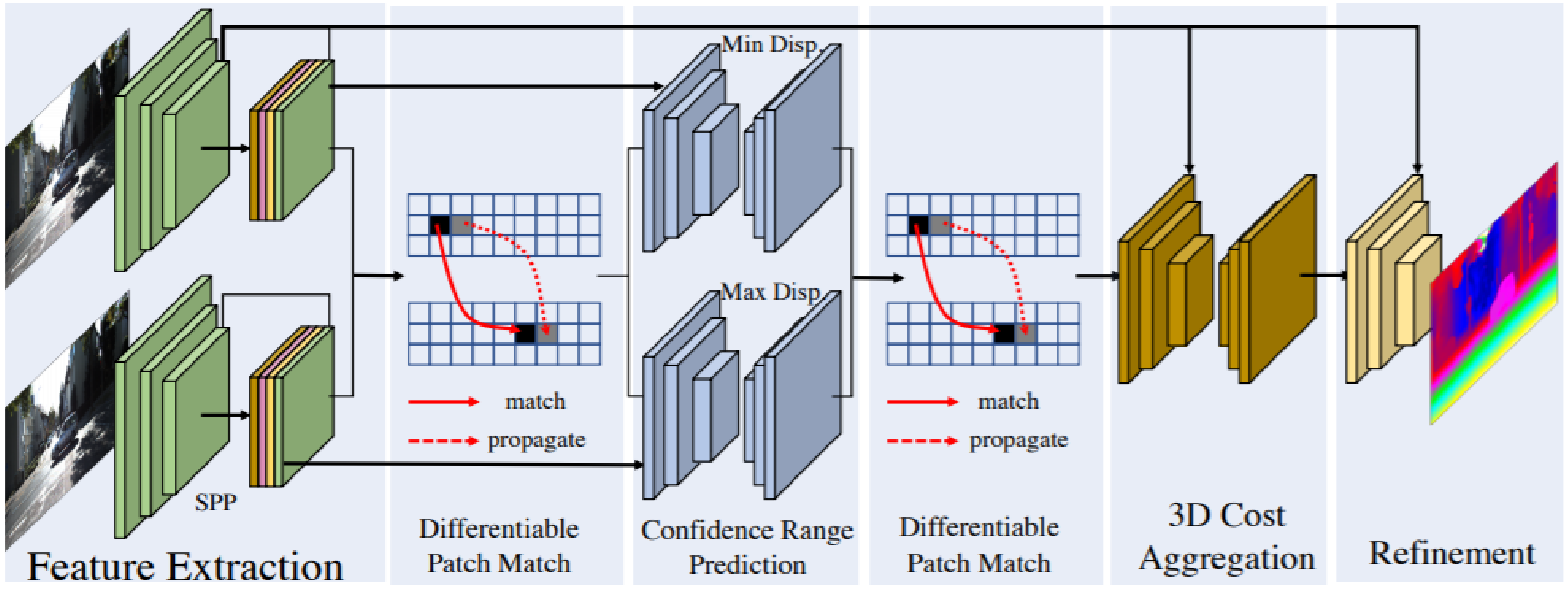


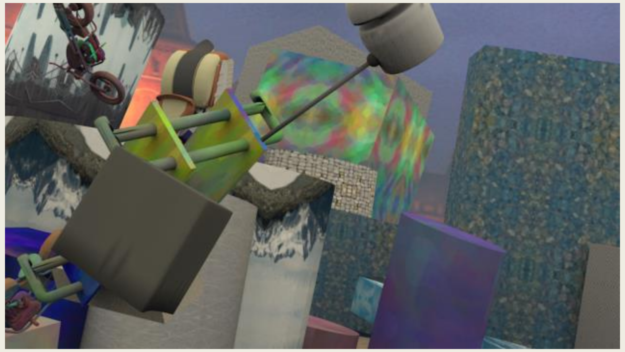
Figure 1: **Overview:** Given a pair of stereo images, we first extract deep multi-scale features. Then we exploit differentiable PatchMatch to estimate a small subset of disparities for each pixel and capitalize on confidence range predictor to further prune out the solution space. Unlike other approaches [8, 15] which operate on the entire disparity search range, we only aggregate the cost within the reduced search range. Finally, we leverage a light-weight network to refine the stereo output.

Implementation

- 3 Models
  - *Classic Stereo (Basic Block Matching)*
  - *My Implementation*
    - Non-Differentiable Patch Match (hard *arg min*)
    - Includes random search step
      - *Just one search at 50px radius (1/2 the total search radius)*
  - *DeepPruner's Differentiable PatchMatch Implementation*
    - Differentiable Patch Match (soft *arg max*)
    - Excludes random search step
- Design
  - *Python 3.8, NumPy 1.17.3, Matplotlib 3.0.2, PyTorch 1.3.0+cpu*
  - *Object-Oriented*
  - *Assumed disparity search radius of 100px*

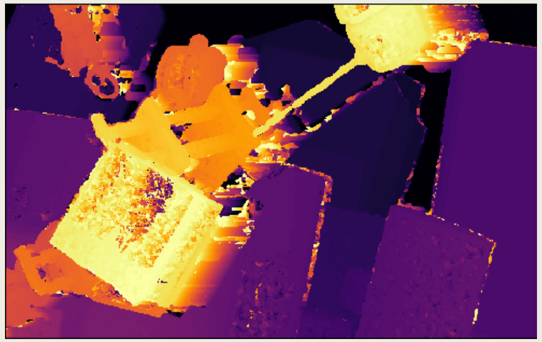
Experiments/Results

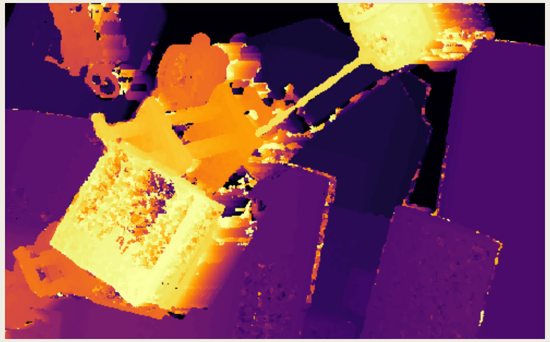

Left Image

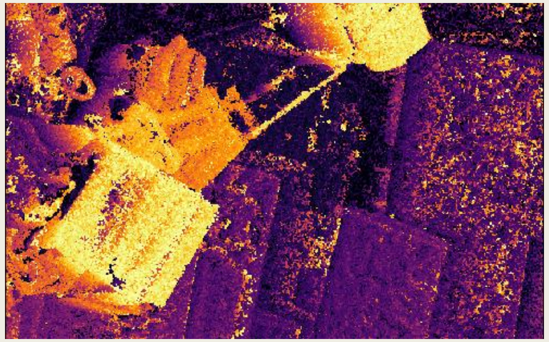

Right Image

		Driving			Flying Objects		
		0001.png	**0002.png**	**0003.png**	**1001.png**	**1002.png**	**1003.png**
PM	**1 iter**	27.922	45.469	46.391	30.281	51.391	60.672
	2 iter	46.719	100.109	86.219	54.328	86.500	99.672
	5 iter	110.563	190.625	181.688	178.438	201.203	226.875
DPM	**1 iter**	18.281	28.516	27.719	25.750	25.625	31.000
	2 iter	37.547	50.641	49.469	51.375	52.719	60.109
	5 iter	114.063	126.953	128.281	129.078	133.828	144.266
Classical		1044.828	1224.375	1320.578	840.594	931.609	547.375

Runtime Results (seconds)


Basic Block Matching


My Algorithm (5 Iterations)


DeepPruner Differentiable PatchMatch (5 Iterations)

Conclusion/Discussion

- Independent of a more sophisticated model, basic block matching does a poor job
- PatchMatch can significantly reduce disparity search field
- More research should be done into the random search phase