PMBP: PatchMatch Belief Propagation for Correspondence Field Estimation – Supplementary Material

Frederic Besse[1]
f.besse@cs.ucl.ac.uk
Carsten Rother[2]
carrot@microsoft.com
Andrew Fitzgibbon[2]
awf@microsoft.com
Jan Kautz[1]
j.kautz@cs.ucl.ac.uk

[1] University College London
London, UK
[2] Microsoft Research Cambridge
Cambridge, UK

1 Algorithm

1.1 Message Passing

As mentioned in the main paper, the message from node t to s is defined as

$$M_{t\to s}(\mathbf{u}_s) := \min_{\mathbf{u}_t} \ \psi_{st}(\mathbf{u}_s,\mathbf{u}_t)+B_t(\mathbf{u}_t)-M_{s\to t}(\mathbf{u}_t)$$

which can be rewritten as

$$M_{t\to s}(\mathbf{u}_s) := \min_{\mathbf{u}_t} \ \psi_{st}(\mathbf{u}_s,\mathbf{u}_t)+\hat{M}_{t\to s}(\mathbf{u}_t)$$

where

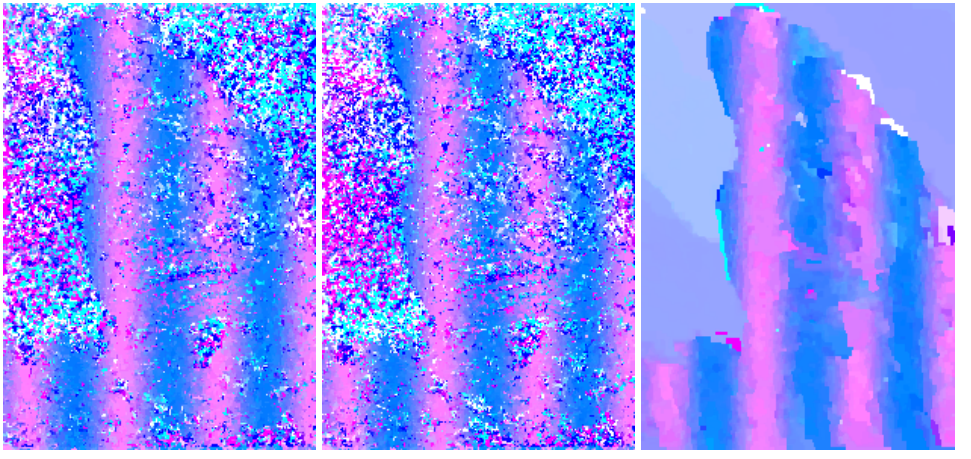
$$\hat{M}_{t\to s}(\mathbf{u}_t) := B_t(\mathbf{u}_t)-M_{s\to t}(\mathbf{u}_t)$$

is called the *pre-message* from t to s. Note that the pre-message is a function of $\mathbf{u}_t$. In our implementation, we *pull* messages from nodes, rather than *pushing* them, which means that the message $M_{t\to s}(\mathbf{u}_s)$ is pulled and calculated when the algorithm is updating node s, for all neighbours t. Consequently, since the pre-message $\hat{M}_{t\to s}(\mathbf{u}_t)$ is a function of $\mathbf{u}_t$ only, and thus independent from the states of s, we can cache all the pre-messages coming out of a certain node after having iterated over it. Since during one iteration, at each node, incoming messages need to be calculated every time a new particle is sampled, we gain in computational efficiency by using this caching mechanism.

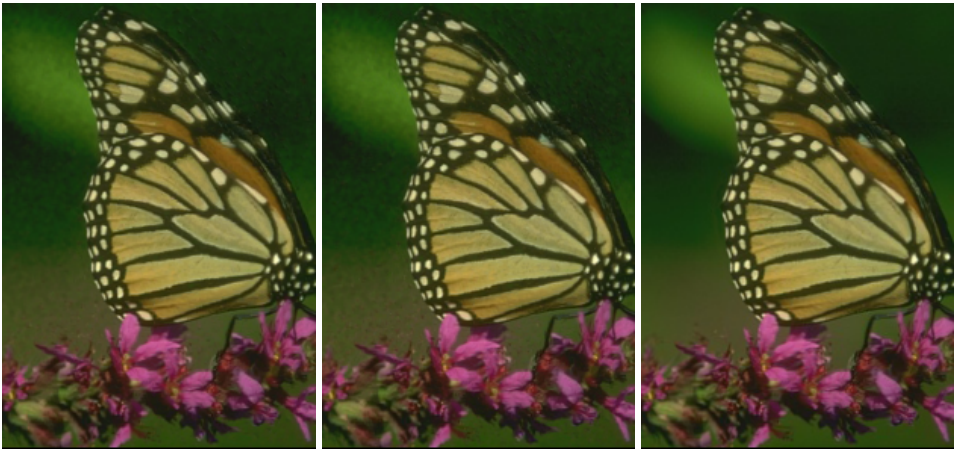
2 Experiments

2.1 Image Matching

In this section we present further image matching and reconstruction results. As mentioned previously, PatchMatch reaches its minimal reconstruction error at an early iteration. We



(a) PM_{best}: $Err = 2e+6$ (b) PM_{end}: $Err = 4e+6$ (c) PMBP: $Err = 4.3e+5$

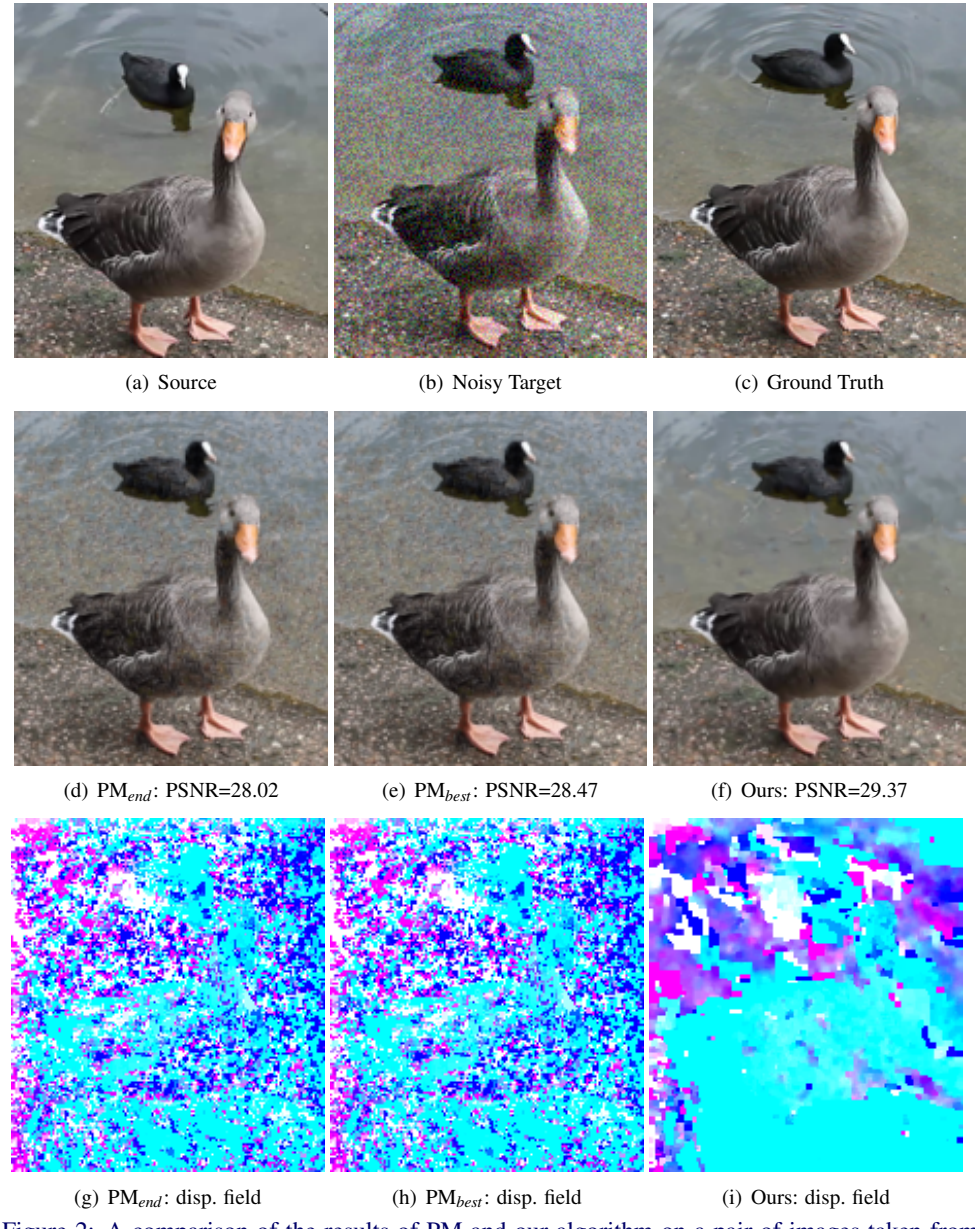


(d) PM_{best}: PSNR= 37.78 (e) PM_{end}: PSNR= 37.39 (f) PMBP: PSNR= 38.36

Figure 1: Displacement field using PatchMatch at the lowest error it reaches (a), at the final iteration (b), using our PMBP method (c). Reconstructed target image using PatchMatch at the lowest energy it reaches (d), at the final iteration (e), and using our PMBP method (f). We run the algorithms for 50 iterations, using 3x3 patches and we allow for subpixel translations only.

show the difference between the results obtained at this particular iteration compared to the last iteration in Figure 1, alongside our result for the butterfly case. As PatchMatch is optimizing the unary energy only, the results at the last iterations are naturally less smooth than at early iterations: it is essentially trying to match the noise. Another example is shown in Figure 2, where we reconstruct a noisy target image from a noise-free – but slightly different – source image. The same behaviour is observed.

We also use this application to show the effect of using more particles in PMBP. Results on cropped region of the Goose example can be found in Figure 3. We see that, at an early stage, fewer particles yield a lower energy than using more particles. After enough processing time more particles yield a solution with slightly lower energy. However, the differences in energy between the solutions after 500s is relatively small. Our results are

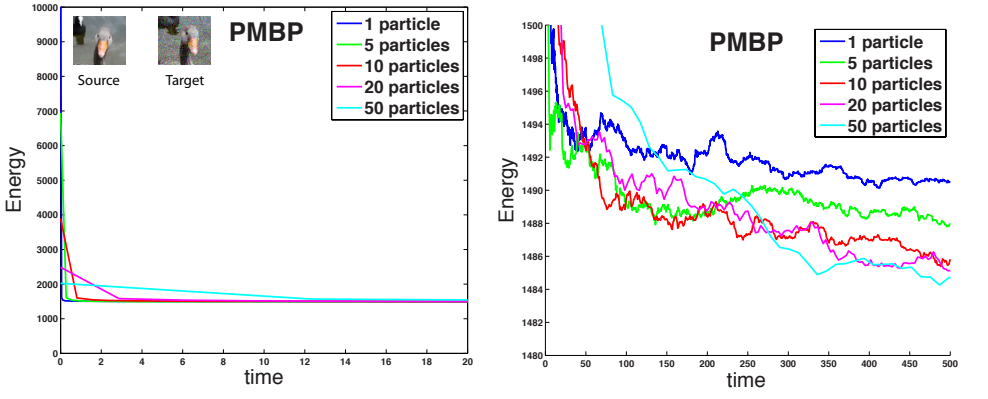


(a) Source (b) Noisy Target (c) Ground Truth

(d) PM$_{end}$: PSNR=28.02 (e) PM$_{best}$: PSNR=28.47 (f) Ours: PSNR=29.37

(g) PM$_{end}$: disp. field (h) PM$_{best}$: disp. field (i) Ours: disp. field

Figure 2: A comparison of the results of PM and our algorithm on a pair of images taken from a video, one of which having been degraded by adding noise. We run the algorithms for 500 iterations, using 3x3 patches and we allow for subpixel translations only. We can see that our algorithm, using a smooth Gaussian pairwise term, manages to reconstruct an image having a higher PSNR than the result output by PM. Furthermore, we once again see that the end iteration of PM yields worse results than its result at an early iteration (iteration 19), from which the PSNR starts decreasing.



(a) (b)

Figure 3: Effect of using more particles. (a) First 20 seconds. (b) Zoom on the energy range 1480-1500 for the whole time range (0-500s).

usually computed using five particles.

2.2 Stereo

In Figure 4, 5, 6 and 7 we show more detailed results of applying our method to the stereo problem. In particular, we show the influence of the weight of the pairwise term on the raw disparity map. As can be seen, using a pairwise weighting coefficient $b = 0$, which is equivalent to using PatchMatch (no smoothness), yields a higher overall error than when adding the pair-wise smoothness term. However, weighting the pairwise smoothness term too strongly starts to increase the error again, which is to be expected. For each case, we show the error curve with and without the post-processing step as proposed in the original PatchMatch Stereo work.

Finally, in Figure 8 we show a comparison between the results of PatchMatch Stereo versus the output of our algorithm taken with the smoothness weight that yields the lowest error.

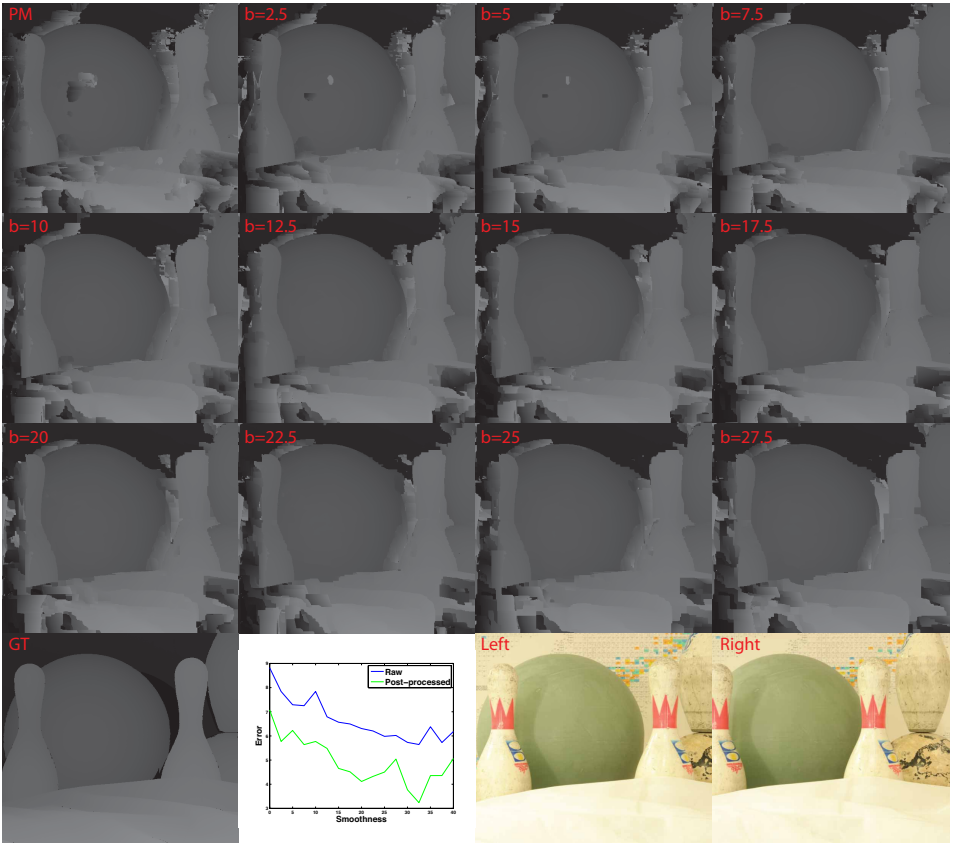


Figure 4: Raw results on the Bowling1 dataset. b is a pairwise weighting coefficient, controlling the amount of smoothness.

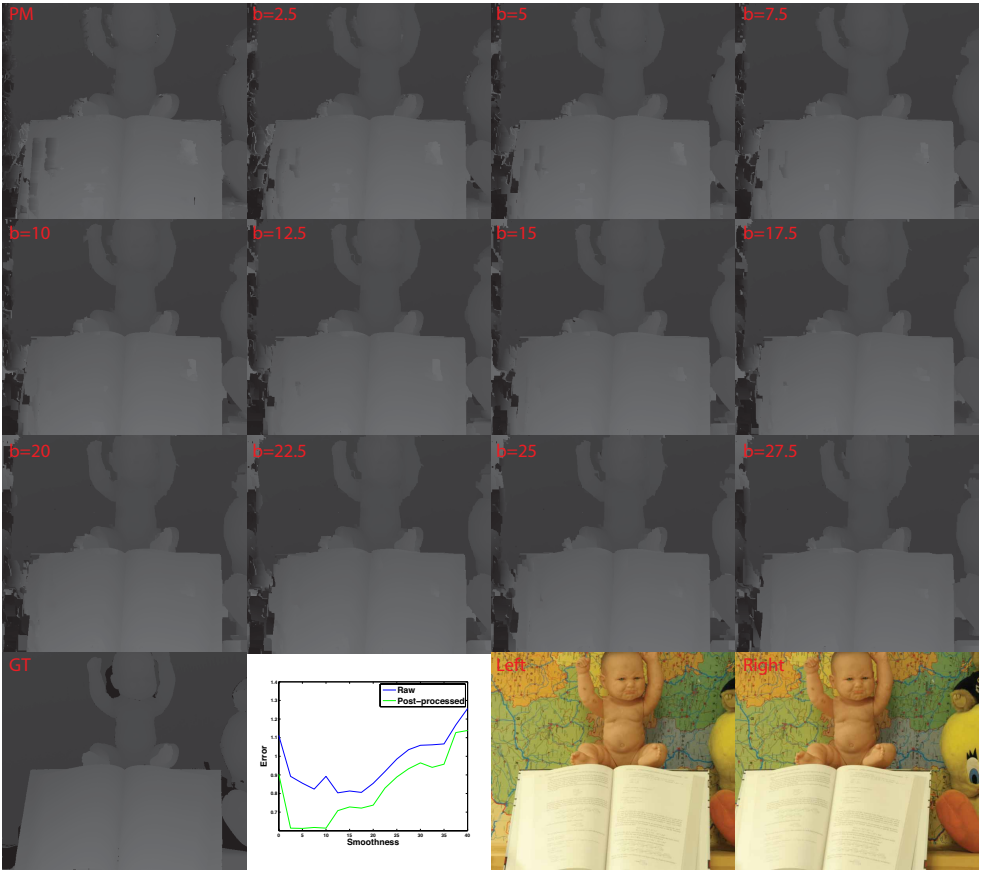


Figure 5: Raw results on the Baby2 dataset. b is a pairwise weighting coefficient, controlling the amount of smoothness.

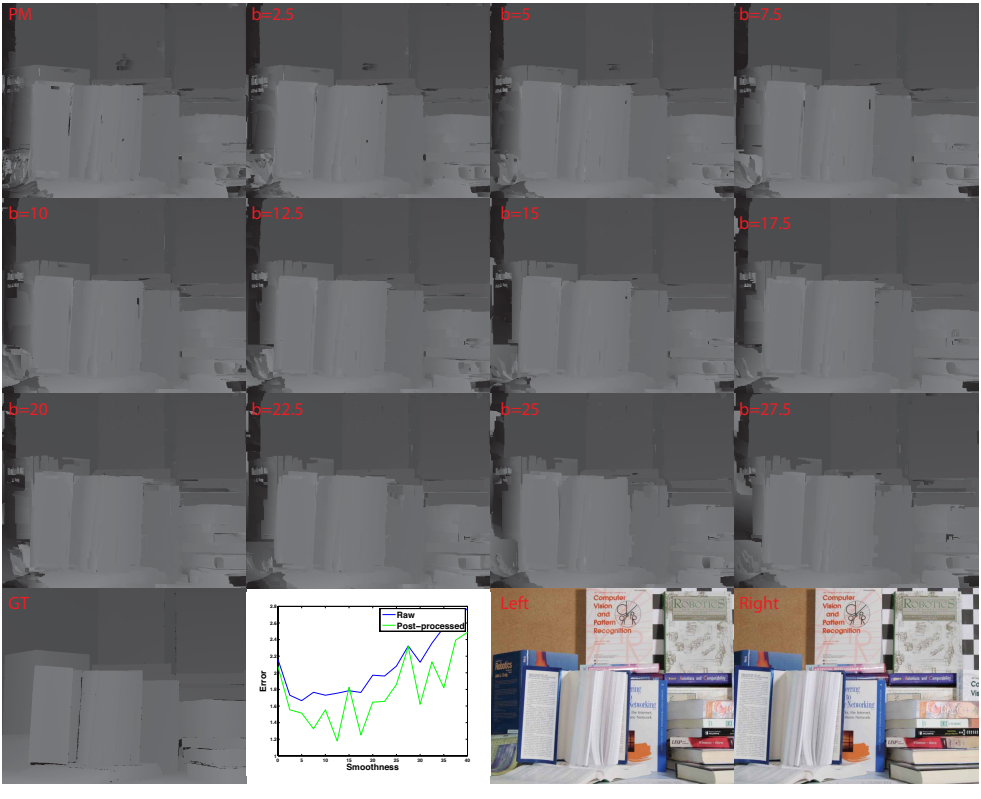


Figure 6: Raw results on the Books dataset. b is a pairwise weighting coefficient, controlling the amount of smoothness.

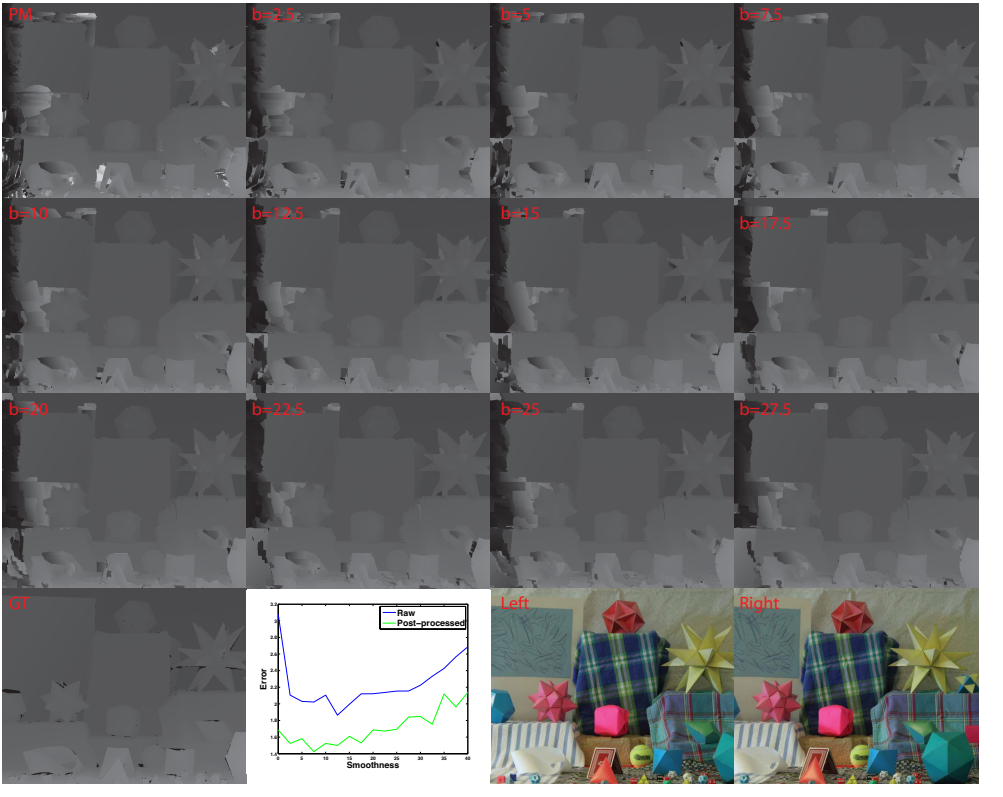


Figure 7: Raw results on the Moebius dataset. b is a pairwise weighting coefficient, controlling the amount of smoothness.

Figure 8: Top row: raw results of PatchMatch stereo. Bottom row: our best raw results. Note that these results are without post-process.