

The Fast Bilateral Solver

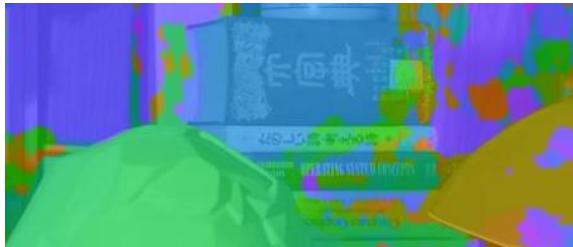
Jonathan T. Barron
barron@google.com

Ben Poole
poole@cs.stanford.edu

Abstract. We present the bilateral solver, a novel algorithm for edge-aware smoothing that combines the flexibility and speed of simple filtering approaches with the accuracy of domain-specific optimization algorithms. Our technique is capable of matching or improving upon state-of-the-art results on several different computer vision tasks (stereo, depth superresolution, colorization, and semantic segmentation) while being 10-1000 \times faster than baseline techniques with comparable accuracy, and producing lower-error output than techniques with comparable runtimes. The bilateral solver is fast, robust, straightforward to generalize to new domains, and simple to integrate into deep learning pipelines.

1 Introduction

Images of the natural world exhibit a useful prior – many scene properties (depth, color, object category, etc.) are correlated within smooth regions of an image, while differing across discontinuities in the image. Edge-aware smoothing techniques exploit this relationship to propagate signals of interest within, but not



(a) Input (MAE = 6.00, RMSE = 38.8)



(b) Output (MAE = 3.02, RMSE = 17.9)



(c) Input Confidence



(d) Input Reference

Fig. 1: The bilateral solver can be used to improve depth maps. A depth map (a) from a state-of-the-art stereo method [43] is processed with our robust bilateral solver using a reference RGB image (d). Our output (b) is smooth with respect to the reference image, resulting in a 50% reduction in error.

across edges present in an image. Traditional approaches to edge-aware smoothing apply an image-dependent filter to a signal of interest. Examples of this include joint bilateral filtering [37,40] and upsampling [20], adaptive manifolds [12], the domain transform [11], the guided filter [17,16], MST-based filtering [41], and weighted median filtering [30,44]. These techniques are flexible and computationally efficient, but often insufficient for solving more challenging computer vision tasks. Difficult tasks often necessitate complex iterative inference or optimization procedures that encourage smoothness while maintaining fidelity with respect to some observation. Optimization algorithms of this nature have been used in global stereo [34], depth superresolution [10,19,24,26,29,32], colorization [25], and semantic segmentation [6,22,28,45]. These approaches are tailored to their specific task, and are generally computationally expensive. In this work we present an optimization algorithm that is $10\text{-}1000\times$ faster than existing domain-specific approaches with comparable accuracy, and produces higher-quality output than lightweight filtering techniques with comparable runtimes.

Our algorithm is based on the work of Barron *et al.*[2], who presented the idea of using fast bilateral filtering techniques to solve optimization problems in “bilateral-space”. This allows for some optimization problems with bilateral affinity terms to be solved quickly, and also guarantees that the solutions to those problems are “bilateral-smooth” — smooth within objects, but not smooth across edges. In this paper we present a new form of bilateral-space optimization which we call *the bilateral solver*, which efficiently solves a regularized least-squares optimization problem to produce an output that is bilateral-smooth and close to the input. This approach has a number of benefits:

General The bilateral solver is a single intuitive abstraction that can be applied to many different problems, while matching or beating the specialized state-of-the-art algorithms for each of these problems. It can be generalized to a variety of loss functions using standard techniques from M-estimation [14].

Differentiable Unlike other approaches for edge-aware smoothness which require a complicated and expensive “unrolling” to perform backpropagation [45], the backward pass through our solver is as simple and fast as the forward pass, allowing it to be easily incorporated into deep learning architectures.

Fast The bilateral solver is expressible as a linear least-squares optimization problem, unlike the non-linear optimization problem used in [2]. This enables a number of optimization improvements including a hierarchical preconditioner and initialization technique that hasten convergence, as well as efficient methods for solving multiple problems at once.

2 Problem Formulation

We begin by presenting the objective and optimization techniques that make up our bilateral solver. Let us assume that we have some per-pixel input quantities \mathbf{t} (the “target” value, see Figure 1a) and some per-pixel confidence of those quantities \mathbf{c} (Figure 1c), both represented as vectorized images. Let us also assume that we have some “reference” image (Figure 1d), which is a normal RGB

image. Our goal is to recover an “output” vector \mathbf{x} (Figure 1b), which will resemble the input target where the confidence is large while being smooth and tightly aligned to edges in the reference image. We will accomplish this by constructing an optimization problem consisting of an image-dependent smoothness term that encourages \mathbf{x} to be bilateral-smooth, and a data-fidelity term that minimizes the squared residual between \mathbf{x} and the target \mathbf{t} weighted by our confidence \mathbf{c} :

$$\underset{\mathbf{x}}{\text{minimize}} \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} (x_i - x_j)^2 + \sum_i c_i (x_i - t_i)^2 \quad (1)$$

The smoothness term in this optimization problem is built around an affinity matrix \hat{W} , which is a bistochastized version of a bilateral affinity matrix W . Each element of the bilateral affinity matrix $W_{i,j}$ reflects the affinity between pixels i and j in the reference image in the YUV colorspace:

$$W_{i,j} = \exp \left(-\frac{\| [p_i^x, p_i^y] - [p_j^x, p_j^y] \|^2}{2\sigma_{xy}^2} - \frac{(p_i^l - p_j^l)^2}{2\sigma_l^2} - \frac{\| [p_i^u, p_i^v] - [p_j^u, p_j^v] \|^2}{2\sigma_{uv}^2} \right) \quad (2)$$

Where p_i is a pixel in our reference image with a spatial position (p_i^x, p_i^y) and color (p_i^l, p_i^u, p_i^v) ¹. The σ_{xy} , σ_l , and σ_{uv} parameters control the extent of the spatial, luma, and chroma support of the filter, respectively.

This W matrix is commonly used in the bilateral filter [40], an edge-preserving filter that blurs within regions but not across edges by locally adapting the filter to the image content. There are techniques for speeding up bilateral filtering [1,5] which treat the filter as a “splat/blur/slice” procedure: pixel values are “splatted” onto a small set of vertices in a grid [2,5] or lattice [1] (a soft histogramming operation), then those vertex values are blurred, and then the filtered pixel values are produced via a “slice” (an interpolation) of the blurred vertex values. These splat/blur/slice filtering approaches all correspond to a compact and efficient factorization of W :

$$W = S^T \bar{B} S \quad (3)$$

Barron *et al.*[2] built on this idea to allow for optimization problems to be “splatted” and solved in bilateral-space. They use a “simplified” bilateral grid and a technique for producing bistochastization matrices $D_{\mathbf{n}}$, $D_{\mathbf{m}}$ that together give the the following equivalences:

$$\hat{W} = S^T D_{\mathbf{m}}^{-1} D_{\mathbf{n}} \bar{B} D_{\mathbf{n}} D_{\mathbf{m}}^{-1} S \quad SS^T = D_{\mathbf{m}} \quad (4)$$

They also perform a variable substitution, which reformulates a high-dimensional pixel-space optimization problem in terms of the lower-dimensional bilateral-space vertices:

$$\mathbf{x} = S^T \mathbf{y} \quad (5)$$

Where \mathbf{y} is a small vector of values for each bilateral-space vertex, while \mathbf{x} is a large vector of values for each pixel. With these tools we can not only reformulate

¹ To reduce confusion between the Y’s in “YUV” and “XY” we refer to luma as “l”

our pixel-space loss function in Eq 1 in bilateral-space, but we can rewrite that bilateral-space loss function in a quadratic form:

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{y}^T A \mathbf{y} - \mathbf{b}^T \mathbf{y} + c \\ A = \lambda(D_{\mathbf{m}} - D_{\mathbf{n}} \bar{B} D_{\mathbf{n}}) + \text{diag}(S\mathbf{c}) \quad & \mathbf{b} = S(\mathbf{c} \circ \mathbf{t}) \quad c = \frac{1}{2}(\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \end{aligned} \quad (6)$$

where \circ is the Hadamard product. A derivation of this reformulation can be found in the supplement. While the optimization problem in Equation 1 is intractably expensive to solve naively, in this bilateral-space formulation optimization can be performed quickly. Minimizing that quadratic form is equivalent to solving a sparse linear system:

$$A\mathbf{y} = \mathbf{b} \quad (7)$$

We can produce a pixel-space solution $\hat{\mathbf{x}}$ by simply slicing the solution to that linear system:

$$\hat{\mathbf{x}} = S^T(A^{-1}\mathbf{b}) \quad (8)$$

With this we can describe our algorithm, which we will refer to as the “bilateral solver.” The input to the solver is a reference RGB image, a target image that contains noisy observed quantities which we wish to improve, and a confidence image. We construct a simplified bilateral grid from the reference image, which is bistoachastized as in [2] (see the supplement for details), and with that we construct the A matrix and \mathbf{b} vector described in Equation 6 which are used to solve the linear system in Equation 8 to produce an output image. If we have multiple target images (with the same reference and confidence images) then we can construct a larger linear system in which \mathbf{b} has many columns, and solve for each channel simultaneously using the same A matrix. In this many-target case, if \mathbf{b} is low rank then that property can be exploited to accelerate optimization, as we show in the supplement.

Our pixel-space loss (Eq 1) resembles that of weighted least squares filtering [8,9,31], with one critical difference being our use of bilateral-space optimization which allows for efficient optimization even when using a large spatial support in the bilateral affinity, thereby improving the quality of our output and the speed of our algorithm. Our algorithm is similar to the optimization problem that underlies the stereo technique of [2], but with several advantages: Our approach reduces to a simple least-squares problem, which allows us to optimize using standard techniques (we use the preconditioned conjugate gradient algorithm of [36], see the supplement for details). This simple least-squares formulation also allows us to efficiently backpropagate through the solver (Section 3), allowing it to be integrated into deep learning pipelines. This formulation also improves the rate of convergence during optimization, provides guarantees on correctness, allows us to use advanced techniques for preconditioning and initialization (Section 4), and enables robust and multivariate generalizations of our solver (see the supplement).

3 Backpropagation

Integrating any operation into a deep learning framework requires that it is possible to backpropagate through that operation. Backpropagating through global operators such as our bilateral solver is generally understood to be difficult, and is an active research area [18]. Unlike most global smoothing operators, our model is easy to backpropagate through by construction. Note that we do not mean backpropagating *through* a multiplication of a matrix inverse A^{-1} , which would simply be another multiplication by A^{-1} . Instead, we will backpropagate *onto* the A matrix used in the least-squares solve that underpins the bilateral solver, thereby allowing us to backpropagate through the bilateral solver itself.

Consider the general problem of solving a linear system:

$$A\mathbf{y} = \mathbf{b} \quad (9)$$

Where A is an invertible square matrix, and \mathbf{y} and \mathbf{b} are vectors. We can solve for $\hat{\mathbf{y}}$ as a simple least squares problem:

$$\hat{\mathbf{y}} = A^{-1}\mathbf{b} \quad (10)$$

Let us assume that A is symmetric in addition to being positive definite, which is true in our case. Now let us compute some loss with respect to our estimated vector $g(\hat{\mathbf{y}})$, whose gradient will be $\partial g / \partial \hat{\mathbf{y}}$. We would like to backpropagate that quantity onto A and \mathbf{b} :

$$\frac{\partial g}{\partial \mathbf{b}} = A^{-1} \frac{\partial g}{\partial \hat{\mathbf{y}}} \quad \frac{\partial g}{\partial A} = \left(-A^{-1} \frac{\partial g}{\partial \hat{\mathbf{y}}} \right) \hat{\mathbf{y}}^T = - \frac{\partial g}{\partial \mathbf{b}} \hat{\mathbf{y}}^T \quad (11)$$

This can be derived using the implicit function theorem. We see that backpropagating a gradient through a linear system only requires a single least-squares solve. The gradient of the loss with respect to the diagonal of A can be computed more efficiently:

$$\frac{\partial g}{\partial_{\text{diag}(A)}} = - \frac{\partial g}{\partial \mathbf{b}} \circ \hat{\mathbf{y}} \quad (12)$$

We will use these observations to backpropagate through the bilateral solver. The bilateral solver takes some input target \mathbf{t} and some input confidence \mathbf{c} , and then constructs a linear system that gives us a bilateral-space solution $\hat{\mathbf{y}}$, from which we can “slice” out a pixel-space solution $\hat{\mathbf{x}}$.

$$\hat{\mathbf{y}} = A^{-1}\mathbf{b} \quad \hat{\mathbf{x}} = S^T \hat{\mathbf{y}} \quad (13)$$

Note that A and \mathbf{b} are both functions of \mathbf{t} and \mathbf{c} , though they are not written as such. Let us assume that we have computed some loss $f(\hat{\mathbf{x}})$ and its gradient $\partial f / \partial \hat{\mathbf{x}}$. Remember that the A matrix and \mathbf{b} vector in our linear system are functions of some input signal \mathbf{t} and some input confidence \mathbf{c} . Using (11) we can compute the

gradient of the loss with respect to the parameters of the linear system within the bilateral solver:

$$\frac{\partial f}{\partial \mathbf{b}} = A^{-1} \left(S \frac{\partial f}{\partial \hat{\mathbf{x}}} \right) \quad \frac{\partial f}{\partial_{\text{diag}(A)}} = -\frac{\partial f}{\partial \mathbf{b}} \circ \hat{\mathbf{y}} \quad (14)$$

We need only compute the gradient of the loss with respect to the diagonal of A as opposed to the entirety of A , because the off-diagonal elements of A do not depend on the input signal or confidence. We can now backpropagate the gradient of the loss $f(\hat{\mathbf{x}})$ onto the inputs of the bilateral solver:

$$\frac{\partial f}{\partial \mathbf{t}} = \mathbf{c} \circ \left(S^T \frac{\partial f}{\partial \mathbf{b}} \right) \quad \frac{\partial f}{\partial \mathbf{c}} = \left(S^T \frac{\partial f}{\partial_{\text{diag}(A)}} \right) + \left(S^T \frac{\partial f}{\partial \mathbf{b}} \right) \circ \mathbf{t} \quad (15)$$

To review, the bilateral solver can be viewed as a function which takes in a reference image, some input signal and a per-pixel confidence in that input signal, and produces some smoothed output:

$$\text{output} \leftarrow \text{solver}_{\text{reference}}(\text{target}, \text{confidence}) \quad (16)$$

And we have shown how to backpropagate through the solver:

$$(\nabla \text{target}, \nabla \text{confidence}) \leftarrow \text{backprop}_{\text{reference}}(\nabla \text{output}) \quad (17)$$

Because the computational cost of the backwards pass is dominated by the least squares solve necessary to compute $\partial f / \partial \mathbf{b}$, computing the backward pass through the solver is no more costly than computing the forward pass. Contrast this with past approaches for using iterative optimization algorithms in deep learning architectures, which create a sequence of layers, one for each iteration in optimization [45]. The backward pass in these networks is a fixed function of the forward pass and so cannot adapt like the bilateral solver to the structure of the error gradient at the output. Furthermore, in these “unrolled” architectures, the output at each iteration (layer) must be stored during training, causing the memory requirement to grow linearly with the number of iterations. In the bilateral solver, the memory requirements are small and independent of the number of iterations, as we only need to store the bilateral-space output of the solver $\hat{\mathbf{y}}$ during training. These properties make the bilateral solver an attractive option for deep learning architectures where speed and memory usage are important.

4 Preconditioning & Initialization

Optimization of the quadratic objective of the bilateral solver can be sped up with improved initialization and preconditioning. In the previous work of [2], the non-linear optimization used a hierarchical technique which lifted optimization into a pyramid space, using a bilateral variant of the image pyramid optimization approach of [3]. This approach cannot be used by our solver, as most linear

solvers require a preconditioner where the input is of the same dimensionality as the output. Regardless, the approach of [2] is also suboptimal for our use case, as the simple linear structure of our system allows us to construct more accurate and effective preconditioning and initialization techniques.

To best explain our preconditioning and initialization techniques we must first present baselines techniques for both. We can extract the diagonal of our A matrix to construct a Jacobi preconditioner:

$$\text{diag}(A) = \lambda (\text{diag}(D_{\mathbf{m}}) - \text{diag}(D_{\mathbf{n}}) \bar{B}_{\text{diag}} \text{diag}(D_{\mathbf{n}})) + S\mathbf{c}$$

This is straightforward to compute, as $D_{\mathbf{m}}$ and $D_{\mathbf{n}}$ are diagonal matrices and \bar{B} has a constant value along the diagonal denoted here as \bar{B}_{diag} . The Jacobi preconditioner is simply the inverse of the diagonal of A :

$$M_{\text{jacobi}}^{-1}(\mathbf{y}) = \text{diag}(A)^{-1}\mathbf{y} \quad (18)$$

We can also initialize the state vector \mathbf{y} in our optimization to the value which minimizes the data term in our loss, which has a closed form:

$$\mathbf{y}_{\text{flat}} = S(\mathbf{c} \circ \mathbf{t}) / S(\mathbf{c}) \quad (19)$$

This preconditioner and initialization technique perform well, as can be seen in Figure 2. But we can improve upon these baseline techniques by constructing hierarchical generalizations of each.

Hierarchical preconditioners have been studied extensively for image interpolation and optimization tasks. Unfortunately, techniques based on image pyramids [38] are not applicable to our task as our optimization occurs in a sparse 5-dimensional bilateral-space. More sophisticated image-dependent or graph based techniques [21,23,39] are effective preconditioners, but in our experiments the cost of constructing the preconditioner greatly outweighs the savings provided by the improved conditioning. We will present a novel preconditioner which is similar in spirit to hierarchical basis functions [38] or push-pull interpolation [13], but adapted to our task using the bilateral pyramid techniques presented in [2]. Because of its bilateral nature, our preconditioner is inherently locally adapted and so resembles image-adapted preconditioners [23,39].

We will use the multiscale representation of bilateral-space presented in [2] to implement our hierarchical preconditioner. This gives us $P(\mathbf{y})$ and $P^T(\mathbf{z})$, which construct a pyramid-space vector \mathbf{z} from a bilateral-space vector \mathbf{y} , and collapse \mathbf{z} down to \mathbf{y} respectively (see the supplement for details). To evaluate our preconditioner, we lift our bilateral-space vector into pyramid-space, apply an element-wise scaling of each pyramid coefficient, and then project back onto bilateral-space:

$$M_{\text{hier}}^{-1}(\mathbf{y}) = P^T \left(\frac{\mathbf{z}_{\text{weight}} \circ P(\mathbf{1}) \circ P(\mathbf{y})}{P(\text{diag}(A))} \right) \quad (20)$$

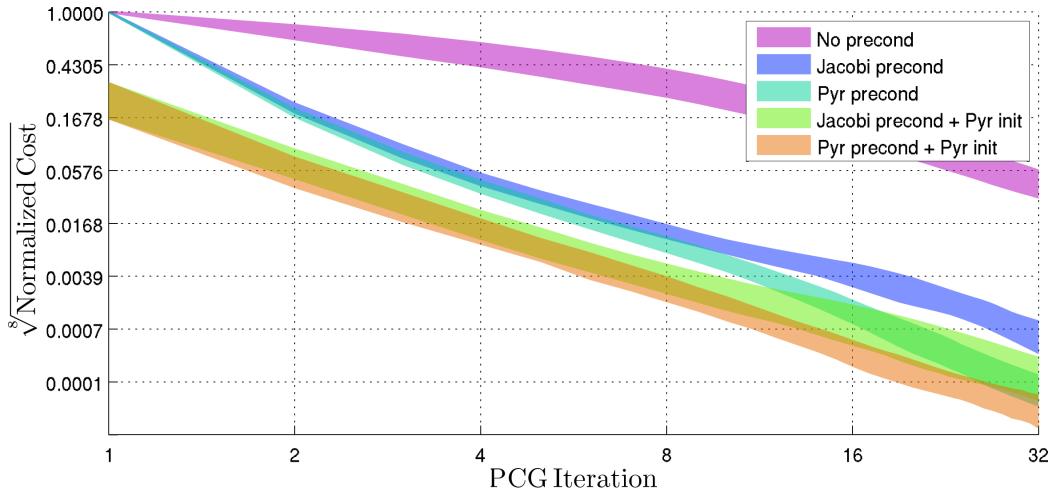


Fig. 2: Our loss during PCG for 20 4-megapixel images, with the loss for each image normalized to [0, 1] and with the 25th-75th percentiles plotted. We see that preconditioning is critical, and that our hierarchical (“Pyr”) preconditioning and initialization techniques significantly improve performance over the naive Jacobi preconditioner and “flat” initialization. Note the non-linear y -axis and logarithmic x -axis.

where the division is element-wise. $M_{hier}^{-1}(\cdot)$ includes an ad-hoc element-wise scaling:

$$\mathbf{z}_{weight} = \begin{cases} 1 & \text{if } k = 0 \\ \alpha^{-(\beta+k)} & \text{otherwise} \end{cases} \quad (21)$$

The pyramid-space scaling we use in Equation 20 is proportional to: 1) the number of bilateral-space vertices assigned to each pyramid-space coefficient (computed by lifting a vector of ones), 2) the inverse of the diagonal of the A matrix, computed by lifting and inverting the diagonal of the A matrix, and 3) an exponential weighting of each pyramid-space coefficient according to its level in the pyramid. This per-level scaling \mathbf{z}_{weight} is computed as a function of the level k of each coefficient, which allows us to prescribe the influence that each scale of the pyramid should have in the preconditioner. Note that as the coarser levels are weighed less (ie, as α or β increases) our preconditioner degenerates naturally to the Jacobi preconditioner. In all experiments we use ($\alpha = 2$, $\beta = 5$) for the preconditioner.

This same bilateral pyramid approach can be used to effectively initialize the state before optimization. Rather than simply taking the input target and using it as our initial state as was done in Equation 19, we perform a push-pull filter of that initial state with the pyramid according to the input confidence:

$$\mathbf{y}_{hier} = P^T \left(\frac{\mathbf{z}_{weight} \circ P(S(\mathbf{c} \circ \mathbf{t}))}{P(\mathbf{1})} \right) / P^T \left(\frac{\mathbf{z}_{weight} \circ P(S(\mathbf{c}))}{P(\mathbf{1})} \right) \quad (22)$$

Table 1: Our approach’s runtime has a lower mean and variance than that of [2]. Runtimes are from the same workstation, averaged over the 20 4-megapixel images used in [2] for profiling.

Algorithm Component	Time (ms)	
	Barron <i>et al.</i> [2]	This Work
Problem Construction	190 \pm 167	35 \pm 7
Optimization	460 \pm 207	152 \pm 36
Total	650 \pm 266	187 \pm 37

Like our hierarchical preconditioner, this initialization degrades naturally to our non-hierarchical initialization in Eq. 19 as α and β increase. In all experiments we use ($\alpha = 4$, $\beta = 0$) for initialization.

See Figure 2 for a visualization of how our hierarchical preconditioning and initialization improve convergence during optimization, compared to the “flat” baseline algorithms. See Table 1 for a comparison of our runtime compared to [2], where we observe a substantial speedup with respect to the solver of [2]. Though the techniques presented here for efficient optimization and initialization are framed in terms of the forward pass through the solver, they all apply directly to the backward pass through the solver described in Section 3, and produce equivalent improvements in speed.

5 Applications

We evaluate our solver on a variety of applications: stereo, depth superresolution, image colorization, and semantic segmentation. Each of these tasks has been the focus of significant research, with specialized techniques having been developed for each problem. For some of these applications (semantic segmentation and stereo) our solver serves as a building block in a larger algorithm, while for others (colorization and depth superresolution) our solver is a complete algorithm. We will demonstrate that our bilateral solver produces results that are comparable to or better than the state-of-the-art for each problem, while being either 1-3 orders of magnitude faster. For those techniques with comparable runtimes, we will demonstrate that the bilateral solver produces higher quality output. Unless otherwise noted, all runtimes were benchmarked on a 2012 HP Z420 workstation (Intel Xeon CPU E5-1650, 3.20GHz, 32 GB RAM), and our algorithm is implemented in standard, single-threaded C++. As was done in [2], the output of our bilateral solver is post-processed by the domain transform [11] to smooth out the blocky artifacts introduced by the simplified bilateral grid, and the domain transform is included in all runtimes. For all results of each application we use the same implementation of the same algorithm with different parameters, which are noted in each sub-section. Parameters are: the spatial bandwidths of the bilateral grid (σ_{xy} , σ_l , σ_{uv}), the smoothness multiplier (λ), the spatial and

range bandwidths of the domain transform (σ'_{xy} , σ'_{rgb}). Unless otherwise stated, the bilateral solver is run for 25 iterations of PCG.

5.1 Stereo

We first demonstrate the utility of the bilateral solver as a post-processing procedure for stereo algorithms. Because depth maps produced by stereo algorithms tend to have heavy-tailed noise distributions, we use a variant of our technique called the robust bilateral solver (RBS) with the Geman-McClure loss (described in the supplement). We applied the RBS to the output of the top-performing MC-CNN [43] algorithm on the Middlebury Stereo Benchmark V3 [43]. For comparison, we also evaluated against four other techniques which can or have been used to post-process the output of stereo algorithms. In Table 2 we see that the RBS cuts test- and training-set absolute and RMS errors in half while having little negative effect on the “bad 1%” error metric (the percent of pixels which whose disparities are wrong by more than 1). This improvement is smaller when we only consider non-occluded (NoOcc) as most state-of-the-art stereo algorithms already perform well in the absence of occlusions. The improvement provided by the RBS is more dramatic when the depth maps are visualized, as can be seen in Figure 1 and in the supplement. At submission time our technique achieved a lower test-set MAE and RMSE on the Middlebury benchmark than any published technique².

See the supplement for a discussion of how our baseline comparison results were produced, an evaluation of our RBS and our baseline techniques on three additional contemporary stereo algorithms, the parameters settings used in this experiment, how we compute the initial confidence c for the RBS, and many visualizations.

² <http://vision.middlebury.edu/stereo/eval3/>

Table 2: Our robust bilateral solver significantly improves depth map quality the state-of-the-art MC-CNN[43] stereo algorithm on the Middlebury dataset V3 [33].

Method	All			NoOcc		
	bad 1%	MAE	RMSE	bad 1%	MAE	RMSE
Test Set						
MC – CNN[43]	28.1	17.9	55.0	18.0	3.82	21.3
MC – CNN[43]+RBS	28.2	8.19	29.9	18.9	2.67	15.0
Training Set						
MC-CNN[43]	20.07	5.93	18.36	10.42	1.94	9.07
MC-CNN[43] + TF[41]	29.15	5.67	16.18	20.15	2.17	7.71
MC-CNN[43] + FGF[16]	32.29	5.91	16.32	23.62	2.42	7.98
MC-CNN[43] + WMF[30]	33.37	5.30	15.62	26.29	2.32	8.22
MC-CNN[43] + DT[11]	25.17	5.69	16.53	15.53	2.01	7.72
MC-CNN[43] + RBS (Ours)	19.49	2.81	8.44	11.33	1.40	5.23

5.2 Depth Superresolution

With the advent of consumer depth sensors, techniques have been proposed for upsampling noisy depth maps produced by these sensors using a high-resolution RGB reference image [10,19,24,26,29,32,4,27,31]. Other techniques have been developed for post-processing depth maps in other contexts [41,30,35], and many general edge-aware upsampling or filtering techniques can be used for this task[20,11,17,16,44]. We present an extensive evaluation of the bilateral solver against these approaches for the depth superresolution task. Given a noisy input depth map and an RGB reference image, we resize the depth map to be the size of the reference image with bicubic interpolation and then apply the bilateral solver or one of our baseline techniques. The hyperparameters used by the solver for all experiments are: $\sigma_{xy} = 8$, $\sigma_l = 4$, $\sigma_{uv} = 3$, $\sigma'_{xy} = \sigma'_{rgb} = 16$, $\lambda = 4^{f-1/2}$ (where f is the upsampling factor) and 15 iterations of PCG. Our confidence \mathbf{c} is a Gaussian bump ($\sigma = f/4$) modeling the support of each low-resolution pixel in the upsampled image. To evaluate our model, we use the depth superresolution benchmark of [10] which is based on the Middlebury stereo dataset [34]. Our performance can be see in Table 3 and Figure 3, with more detailed results in the supplement. The bilateral solver produces the third-lowest error rate for this task, though the two better-performing tasks [24,26] use large amounts of external training data and so have an advantage over our technique, which uses no learning for this experiment. Our approach is $600\times$, $1200\times$, and $3000\times$ faster than the three most accurate techniques. The techniques with speeds comparable to or better than the bilateral solver [11,41,16,31] produce error rates that are 25-40% greater than our approach. The bilateral solver represents a effective combination of speed and accuracy, while requiring no training or learning. See the supplement for a more detailed table, a discussion of baselines and runtimes, and many visualizations.

Table 3: Performance on the depth superresolution task of [10]. Runtimes in gray were not computed on our reference hardware, and algorithms which use external training data are indicated with a dagger.

Method	Err	Time (sec)				
Nearest Neighbor	7.26	0.003				
Bicubic	5.91	0.007				
†Kiechle <i>et al.</i> [19]	5.86	450				
Bilinear	5.16	0.004				
Liu <i>et al.</i> [27]	5.10	16.60				
Shen <i>et al.</i> [35]	4.24	31.48				
Diebel & Thrun [7]	3.98	—				
Chan <i>et al.</i> [4]	3.83	3.02				
GuidedFilter[17,10]	3.76	23.89				
Min <i>et al.</i> [31]	3.74	0.383				
†Lu & Forsyth[29]	3.69	20				
Park <i>et al.</i> [32]	3.61	24.05				
:	:	:				
Domain Transform [11]			3.56	0.021		
Ma <i>et al.</i> [30]			3.49	18		
GuidedFilter(Matlab)[17]			3.47	0.434		
Zhang <i>et al.</i> [44]			3.45	1.346		
FastGuidedFilter[16]			3.41	0.225		
Yang 2015 [41]			3.41	0.304		
Yang <i>et al.</i> 2007 [42]			3.25	—		
Farbman <i>et al.</i> [9]			3.19	6.11		
JBU [1,20]			3.14	1.98		
Ferstl <i>et al.</i> [10]			2.93	140		
†Li <i>et al.</i> [26]			2.56	700		
†Kwon <i>et al.</i> [24]			1.21	300		
BS (Ours)			2.70	0.234		

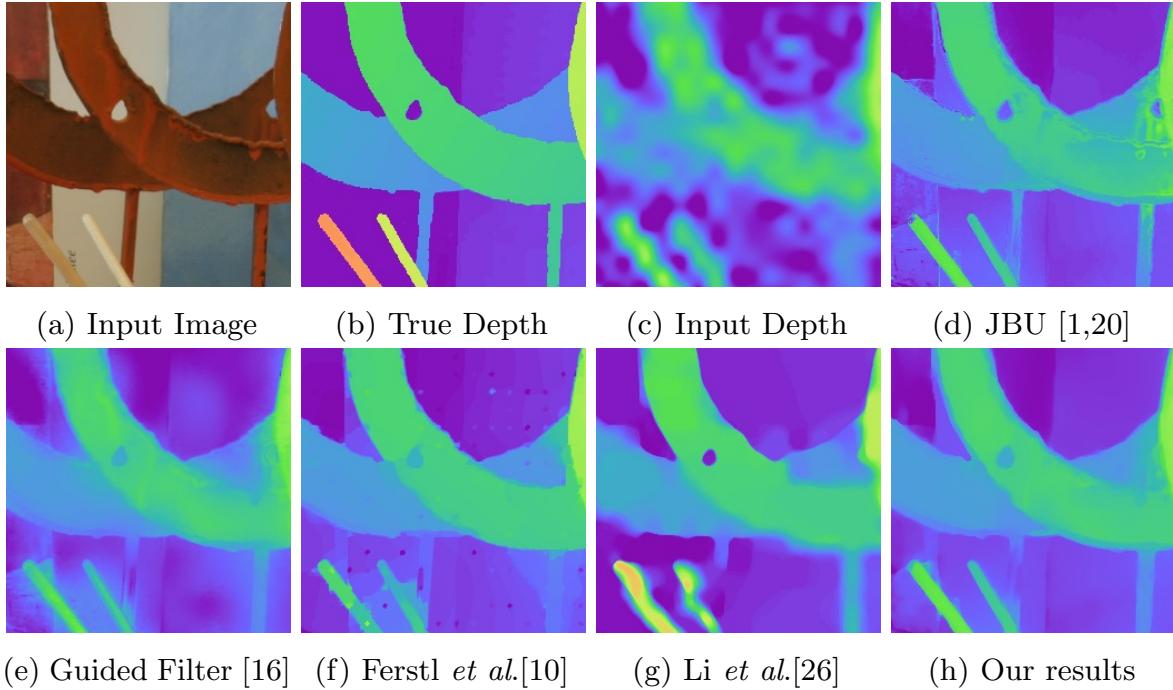


Fig. 3: Partial results for the depth superresolution task of [10], see the supplement for exhaustive visualizations.

5.3 Colorization

Colorization is the problem of introducing color to a grayscale image with a small amount of user input or outside information, for the purpose of improving black-and-white films or photographs. Levin *et al.*[25] presented an effective technique for this task by formulating and solving a specialized optimization problem. We can solve the same task using our bilateral solver: we use the grayscale image as the input reference image and the UV channels of the user-annotated scribbles as the input target images, with a confidence image that is 1 where the user has scribbled and 0 everywhere else. We then construct our final output by combining the grayscale image with our output UV images, and converting from YUV to



Fig. 4: Results for the user-assisted colorization task. Our bilateral solver produces comparable results to the technique of Levin *et al.*[25] while being $95\times$ faster.

RGB. Our results can be seen in Figure 4, where we see that our output is nearly indistinguishable from that of [25]. The important distinction here is speed, as the approach of [25] take 80.99 seconds per megapixel while our approach takes 0.854 seconds per megapixel — a $95\times$ speedup. For all results our parameters are $\sigma_{xy} = \sigma_l = \sigma_{uv} = 4$, $\lambda = 0.5$, $\sigma'_{xy} = 4$, $\sigma'_{rgb} = 8$. More results can be see in the supplement.

5.4 Semantic Segmentation

Semantic segmentation is the problem of assigning a category label to each pixel in an image. State-of-the-art approaches to semantic segmentation use large convolutional neural networks (CNNs) to map from pixels to labels [6,28]. The output of these CNNs is often smoothed across image boundaries, so recent approaches refine their output with a CRF ([6,45]). These CRF-based approaches improve per-pixel labeling accuracy, but this accuracy comes at a computational cost: inference in a fully connected CRF on a 500×500 image can take up to a second (see Table 4). To evaluate whether the bilateral solver could improve the efficiency of semantic segmentation pipelines, we use it instead of the CRF component in two state-of-the-art models: DeepLab-LargeFOV [6] and CRF-RNN [45]. The DeepLab model consists of a CNN trained on Pascal VOC12 and then augmented with a fixed dense CRF. The CRF-RNN model generalizes the CRF with a recurrent neural network, and trains this component jointly with the CNN on Pascal and MSCOCO.

As the bilateral solver operates on real-valued inputs, it is not immediately clear how to map it onto the discrete optimization problem of the dense CRF. For each class, we compute the 21-channel class probability image from the CNN

Table 4: Semantic segmentation results and runtimes on Pascal VOC 2012 validation set. The bilateral solver improves performance over the CNN output while being substantially faster than the CRF-based approaches. “Post” is the time spent post-processing the CNN output, which is the dense CRF for DeepLab, and the generalized CRF-RNN component for CRF-RNN. FCN is the convolutional neural network component of the CRF-RNN model. *DeepLab-LargeFOV model from [6] trained on Pascal VOC 2012 training data augmented with data from [15]. †CRF-RNN model from [45] trained with additional MSCOCO data and evaluated on reduced Pascal validation set of 346 images.

Method	IOU(%)	Time (ms)		
		CNN	Post	Total
DeepLab	62.25*	58	0	58
DeepLab + CRF	67.64*	58	918	976
DeepLab + BS(Ours)	66.00*	58	111	169
CNN	69.60†	715	0	715
CRF – RNN	72.96†	715	2214	2929
CNN + BS(Ours)	70.68†	715	217	913

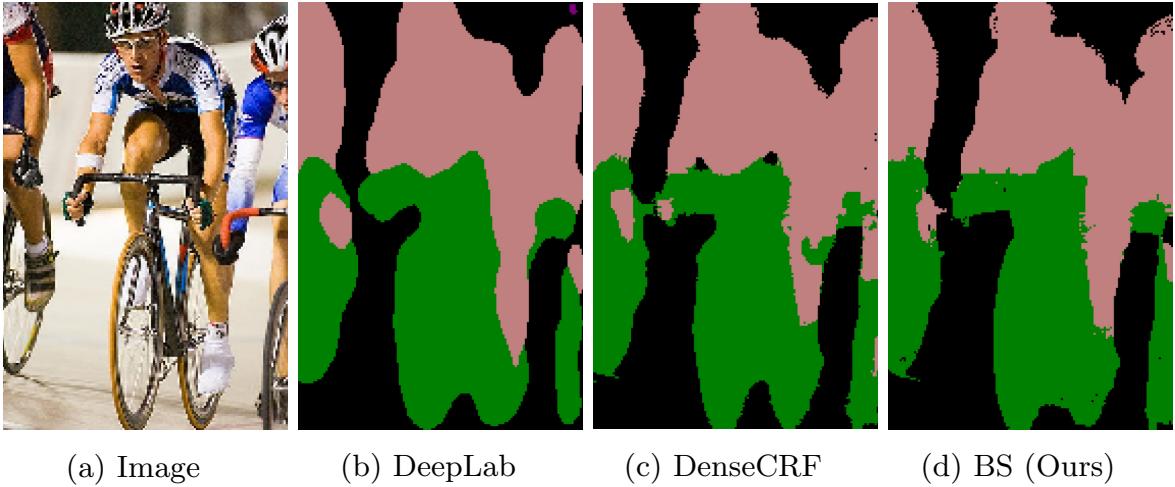


Fig. 5: Using the DeepLab CNN-based semantic segmentation algorithm [6] (5b) as input our bilateral solver can produce comparable edge-aware output (5d) to the DenseCRF [22] used in [6] (5c), while being 8 \times faster.

outputs of either the DeepLab or CRF-RNN model. As many class probability maps are zero across the entire image, the resulting \mathbf{b} matrix in the bilateral solver is often low-rank, allowing us to solve a reduced linear system to recover the smoothed class probability maps (see the supplement). This approach produces nearly identical output, with a 5 \times speedup on average.

We applied our bilateral solver to the class probability maps using uniform confidence. The resulting discrete segmentations are more accurate and qualitatively smoother than the CNN outputs, despite our per-channel smoothing providing no explicit smoothness guarantees on the argmax of the filtered per-class probabilities (Table. 4, Figure 5). The bilateral solver is 8 – 10 \times faster than the CRF and CRF-RNN approaches when applied to the same inputs (Table 4). Although the bilateral solver performs slightly worse than the CRF-based approaches, its speed suggests that it may be a useful tool in contexts such as robotics and autonomous driving, where low latency is necessary.

6 Conclusion

We have presented the bilateral solver, a flexible and fast technique for inducing edge-aware smoothness. We have demonstrated that the solver can produce or improve state-of-the-art results on a variety of different computer vision tasks, while being faster or more accurate than other approaches. Its speed and generality suggests that the bilateral solver is a useful tool in the construction of computer vision algorithms and deep learning pipelines.

References

1. Adams, A., Baek, J., Davis, M.A.: Fast high-dimensional filtering using the permutohedral lattice. *Eurographics* (2010)
2. Barron, J.T., Adams, A., Shih, Y., Hernández, C.: Fast bilateral-space stereo for synthetic defocus. *CVPR* (2015)
3. Barron, J.T., Malik, J.: Shape, illumination, and reflectance from shading. *TPAMI* (2015)
4. Chan, D., Buisman, H., Theobalt, C., Thrun, S.: A noise-aware filter for real-time depth upsampling. *ECCV Workshops* (2008)
5. Chen, J., Paris, S., Durand, F.: Real-time edge-aware image processing with the bilateral grid. *SIGGRAPH* (2007)
6. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected crfs. *ICLR* (2015)
7. Diebel, J., Thrun, S.: An application of markov random fields to range sensing. *NIPS* (2005)
8. Elad, M.: On the origin of the bilateral filter and ways to improve it. *Transactions on Image Processing* (2002)
9. Farbman, Z., Fattal, R., Lischinski, D., Szeliski, R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. *SIGGRAPH* (2008)
10. Ferstl, D., Reinbacher, C., Ranftl, R., Ruether, M., Bischof, H.: Image guided depth upsampling using anisotropic total generalized variation. *ICCV* (2013)
11. Gastal, E.S.L., Oliveira, M.M.: Domain transform for edge-aware image and video processing. *SIGGRAPH* (2011)
12. Gastal, E.S.L., Oliveira, M.M.: Adaptive manifolds for real-time high-dimensional filtering. *SIGGRAPH* (2012)
13. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. *SIGGRAPH* (1996)
14. Hampel, F.R., Ronchetti, E.M., Rousseeuw, P.J., Stahel, W.A.: Robust Statistics - The Approach Based on Influence Functions. Wiley (1986)
15. Hariharan, B., Arbelaez, P., Bourdev, L., Maji, S., Malik, J.: Semantic contours from inverse detectors. *ICCV* (2011)
16. He, K., Sun, J.: Fast guided filter. *CoRR* abs/1505.00996 (2015)
17. He, K., Sun, J., Tang, X.: Guided image filtering. *ECCV* (2010)
18. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix backpropagation for deep networks with structured layers. *ICCV* (2015)
19. Kiechle, M., Hawe, S., Kleinsteuber, M.: A joint intensity and depth co-sparse analysis model for depth map super-resolution. *ICCV* (2013)
20. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. *SIGGRAPH* (2007)
21. Koutis, I., Miller, G.L., Tolliver, D.: Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *CVIU* (2011)
22. Krähenbühl, P., Koltun, V.: Efficient inference in fully connected crfs with gaussian edge potentials. *NIPS* (2011)
23. Krishnan, D., Fattal, R., Szeliski, R.: Efficient preconditioning of laplacian matrices for computer graphics. *SIGGRAPH* (2013)
24. Kwon, H., Tai, Y.W., Lin, S.: Data-driven depth map refinement via multi-scale sparse representation. *CVPR* (2015)
25. Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. *SIGGRAPH* (2004)

26. Li, Y., Xue, T., Sun, L., Liu, J.: Joint example-based depth map super-resolution. ICME (2012)
27. Liu, M.Y., Tuzel, O., Taguchi, Y.: Joint geodesic upsampling of depth images. CVPR (2013)
28. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. CVPR (2015)
29. Lu, J., Forsyth, D.: Sparse depth super resolution. CVPR (2015)
30. Ma, Z., He, K., Wei, Y., Sun, J., Wu, E.: Constant time weighted median filtering for stereo matching and beyond. ICCV (2013)
31. Min, D., Choi, S., Lu, J., Ham, B., Sohn, K., Do, M.N.: Fast global image smoothing based on weighted least squares. Transactions on Image Processing (2014)
32. Park, J., Kim, H., Tai, Y.W., Brown, M.S., Kweon, I.: High quality depth map upsampling for 3d-tof cameras. ICCV (2011)
33. Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X., Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth. GCPR (2014)
34. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. IJCV (2002)
35. Shen, X., Zhou, C., Xu, L., Jia, J.: Mutual-structure for joint filtering. ICCV (2015)
36. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University (1994)
37. Smith, S.M., Brady, J.M.: Susan - a new approach to low level image processing. IJCV (1997)
38. Szeliski, R.: Fast surface interpolation using hierarchical basis functions. TPAMI (1990)
39. Szeliski, R.: Locally adapted hierarchical basis preconditioning. SIGGRAPH (2006)
40. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. ICCV (1998)
41. Yang, Q.: Stereo matching using tree filtering. PAMI (2015)
42. Yang, Q., Yang, R., Davis, J., Nistér, D.: Spatial-depth super resolution for range images. CVPR (2007)
43. Zbontar, J., LeCun, Y.: Computing the stereo matching cost with a convolutional neural network. CVPR (2015)
44. Zhang, Q., Xu, L., Jia, J.: 100+ times faster weighted median filter (wmf). CVPR (2014)
45. Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., Torr, P.: Conditional random fields as recurrent neural networks. ICCV (2015)

The Fast Bilateral Solver

Supplement

Jonathan T. Barron
barron@google.com

Ben Poole
poole@cs.stanford.edu

This supplement provides additional details on the bilateral solver and its extensions in Sections 1–4, and a plethora of experiments and comparisons to prior work in Section 5. In Section 1 we derive the bilateral-space quadratic objective at the core of the bilateral solver. Section 2 details the bilateral-space pyramid representation used for faster initialization and preconditioning. Section 3 extends the bilateral solver to use a robust error function to cope with outliers. Section 4 improves the performance of the bilateral solver when applied to multiple output channels. Finally, in Section 5 we provide additional evaluation of the bilateral solver and extensively compare it to numerous existing and novel baseline techniques.

1 Derivation

In the main paper we presented an optimization problem where we solve for a per-pixel quantity \mathbf{x} subject to a smoothness term that encourages \mathbf{x} to be smooth and a data term that encourages \mathbf{x} to resemble some observed input “target” quantities \mathbf{t} proportionally to a per-pixel “confidence” \mathbf{c} :

$$\underset{\mathbf{x}}{\text{minimize}} \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} (x_i - x_j)^2 + \sum_i c_i (x_i - t_i)^2 \quad (1)$$

We can use the procedure detailed in Barron *et al.*[2] to reformulate the smoothness term into matrix/vector notation. The data term can be similarly reformulated:

$$\sum_i c_i (x_i - t_i)^2 = (\mathbf{x} - \mathbf{t})^T \text{diag}(\mathbf{c})(\mathbf{x} - \mathbf{t}) \quad (2)$$

$$= \mathbf{x}^T \text{diag}(\mathbf{c}) \mathbf{x} - 2 \mathbf{x}^T \text{diag}(\mathbf{c}) \mathbf{t} + \mathbf{t}^T \text{diag}(\mathbf{c}) \mathbf{t} \quad (3)$$

$$= \mathbf{x}^T \text{diag}(\mathbf{c}) \mathbf{x} - 2 (\mathbf{c} \circ \mathbf{t})^T \mathbf{x} + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (4)$$

Combining this reformulated data term with the reformulated smoothness term from [2] we get:

$$\underset{\mathbf{x}}{\text{minimize}} \quad \mathbf{x}^T \left(\lambda \left(I - \hat{W} \right) + \text{diag}(\mathbf{c}) \right) \mathbf{x} - 2 (\mathbf{c} \circ \mathbf{t})^T \mathbf{x} + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (5)$$

Using the bistrostochasticization algorithm from [2], we can decompose \hat{W} :

$$\hat{W} = S^T D_{\mathbf{m}}^{-1} D_{\mathbf{n}} \bar{B} D_{\mathbf{n}} D_{\mathbf{m}}^{-1} S \quad (6)$$

Algorithm 1 Bilateral-space bistochastization, reproduced from [2]

Input: $W = S^T \bar{B} S$ // A splat-blur-splice decomposition of W **Output:** D_n, D_m // Matrices to bistochastize W

```

1:  $\mathbf{m} \leftarrow S\mathbf{1}$ 
2:  $\mathbf{n} \leftarrow \mathbf{1}$ 
3: while not converged do
4:    $\mathbf{n} \leftarrow \sqrt{(\mathbf{n} \circ \mathbf{m}) / (\bar{B}\mathbf{n})}$ 
5: end while
6:  $D_n \leftarrow \text{diag}(\mathbf{n})$ 
7:  $D_m \leftarrow \text{diag}(\mathbf{m})$ 

```

The bistochastization algorithm from [2] is reproduced in Algorithm 1.

Using the simplified bilateral grid of [2] gives us the following equivalence:

$$SS^T = D_m \quad (7)$$

We will use the same bilateral-space variable substitution as [2], by rewriting our optimization problem framed in terms of pixels \mathbf{x} as an optimization in terms of bilateral-space vertices \mathbf{y} :

$$\mathbf{x} = S^T \mathbf{y} \quad (8)$$

Let us perform this variable substitution and simplify the resulting expression using our known equivalences, first for the parts of Equation 5 which correspond to the smoothness term:

$$\mathbf{x}^T (I - \hat{W}) \mathbf{x} = \mathbf{x}^T (I - S^T D_m^{-1} D_n \bar{B} D_n D_m^{-1} S) \mathbf{x} \quad \text{by Eq 6}$$

$$= (S^T \mathbf{y})^T (I - S^T D_m^{-1} D_n \bar{B} D_n D_m^{-1} S) (S^T \mathbf{y}) \quad \text{by Eq 8}$$

$$= \mathbf{y}^T (SIS^T - SS^T D_m^{-1} D_n \bar{B} D_n D_m^{-1} SS^T) \mathbf{y} \quad (9)$$

$$= \mathbf{y}^T (D_m - D_m D_m^{-1} D_n \bar{B} D_n D_m^{-1} D_m) \mathbf{y} \quad \text{by Eq 7}$$

$$= \mathbf{y}^T (D_m - D_n \bar{B} D_n) \mathbf{y} \quad (10)$$

And now, we will perform the variable substitution on the parts of Equation 5 which correspond to the data term:

$$\mathbf{x}^T \text{diag}(\mathbf{c}) \mathbf{x} - 2(\mathbf{c} \circ \mathbf{t})^T \mathbf{x} + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (11)$$

$$= (S^T \mathbf{y})^T \text{diag}(\mathbf{c}) (S^T \mathbf{y}) - 2(\mathbf{c} \circ \mathbf{t})^T (S^T \mathbf{y}) + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad \text{by Eq 8}$$

$$= \mathbf{y}^T (S \text{diag}(\mathbf{c}) S^T) \mathbf{y} - 2(S(\mathbf{c} \circ \mathbf{t}))^T \mathbf{y} + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (12)$$

$$= \mathbf{y}^T \text{diag}(S\mathbf{c}) \mathbf{y} - 2(S(\mathbf{c} \circ \mathbf{t}))^T \mathbf{y} + (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (13)$$

Combining all of this we can rewrite Equation 5 in bilateral-space as follows:

$$\underset{\mathbf{y}}{\text{minimize}} \quad \frac{1}{2} \mathbf{y}^T (\lambda (D_m - D_n \bar{B} D_n) + \text{diag}(S\mathbf{c})) \mathbf{y} - (S(\mathbf{c} \circ \mathbf{t}))^T \mathbf{y} + \frac{1}{2} (\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \quad (14)$$

Unlike the non-linear optimization problem of [2], because we have a simple quadratic optimization problem we can rewrite it in standard form:

$$\begin{aligned} \underset{\mathbf{y}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{y}^T A\mathbf{y} - \mathbf{b}^T \mathbf{y} + c \\ A = \lambda(D_{\mathbf{m}} - D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}) + \text{diag}(S\mathbf{c}) \quad & \mathbf{b} = S(\mathbf{c} \circ \mathbf{t}) \quad c = \frac{1}{2}(\mathbf{c} \circ \mathbf{t})^T \mathbf{t} \end{aligned} \quad (15)$$

By taking the derivative of this loss function and setting it to zero we see that minimizing that quadratic form is equivalent to solving the sparse linear system

$$A\mathbf{y} = \mathbf{b} \quad (16)$$

We will solve this optimization problem using the preconditioned conjugate gradient (PCG) algorithm, using the initialization and preconditioning tricks described in the paper. We use the PCG implementation described in Section B3 of [28], which is reproduced here in Algorithm 2.

Algorithm 2 Preconditioned Conjugate Gradients, reproduced from [28]

Input:

$A(\cdot)$ // A function which implements $A\mathbf{x}$
 \mathbf{b} // The \mathbf{b} vector in the linear system
 \mathbf{x} // The initial value of the state \mathbf{x}
 $M^{-1}(\cdot)$ // A function which implements a preconditioner
 n // the number of iterations

Output:

\mathbf{x} // \mathbf{x} such that $A(\mathbf{x}) \approx \mathbf{b}$

```

1:  $i \leftarrow 0$ 
2:  $\mathbf{r} \leftarrow \mathbf{b} - A(\mathbf{x})$ 
3:  $\mathbf{d} \leftarrow M^{-1}(\mathbf{r})$ 
4:  $\lambda_{new} \leftarrow \mathbf{r}^T \mathbf{d}$ 
5: while  $i < n$  do
6:    $\mathbf{q} \leftarrow A(\mathbf{d})$ 
7:    $\alpha \leftarrow \frac{\lambda_{new}}{\mathbf{d}^T \mathbf{q}}$ 
8:    $\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{d}$ 
9:    $\mathbf{r} \leftarrow \mathbf{r} - \alpha \mathbf{q}$ 
10:   $\mathbf{s} \leftarrow M^{-1}(\mathbf{r})$ 
11:   $\lambda_{old} \leftarrow \lambda_{new}$ 
12:   $\lambda_{new} \leftarrow \mathbf{r}^T \mathbf{s}$ 
13:   $\beta \leftarrow \frac{\lambda_{new}}{\lambda_{old}}$ 
14:   $\mathbf{d} \leftarrow \mathbf{s} + \beta \mathbf{d}$ 
15:   $i \leftarrow i + 1$ 
16: end while

```

2 Bilateral-Space Pyramids

Applying the hierarchical initialization and preconditioning techniques in the main paper requires that we have a multiscale representation of our simplified bilateral grid. As was done in [2], will use the same bilateral grid which was applied to image pixels to instead construct a bilateral grid on top of the vertex coordinates V , where V is an m by 5 matrix produced when constructing a bilateral grid from the input image (m is the number of vertices in the simplified bilateral grid, and 5 is the dimensionality of our XYLUV bilateral-space). From V we can construct a more coarse simplified bilateral grid by dividing the elements of V by 2 (an arbitrary scale factor), and then repeat that procedure to form a pyramid:

```

for  $k = [0 : K - 1]$  do
     $V \leftarrow V/2$ 
     $(S_k, V) \leftarrow \text{simplified\_bilateral\_grid}(V)$ 
end for

```

Where K (the number of levels of the pyramid) is set such that the top of the pyramid contains just a single vertex. As was done in [2], we can use these K splat matrices to lift from some bilateral-space vector \mathbf{y} into a pyramid-space:

$$P(\mathbf{y}) = [S_{K-1} \dots S_1 S_0 \mathbf{y}, \dots, S_1 S_0 \mathbf{y}, S_0 \mathbf{y}, \mathbf{y}] \quad (17)$$

We can transpose this pyramid operation, collapsing back down to bilateral-space:

$$P^T(\mathbf{z}) = [S_0^T S_1^T \dots S_{K-1}^T \mathbf{z}, \dots, S_0^T S_1^T \mathbf{z}, S_0^T \mathbf{z}, \mathbf{z}] \quad (18)$$

We compute $P(\mathbf{y})$ and $P^T(\mathbf{z})$ efficiently from the bottom up and the top down, respectively, by reusing the information from the previous scale.

3 Robustness

Though the quadratic data term of Equation 1 enables our fast least-squares formulation, it has the natural consequence that the bilateral solver is sensitive to outliers in the input target (unless those outliers have been assigned a low confidence). For some applications where the input target has non-Gaussian noise we may wish to be robust to outliers. We therefore present a robustified variant of our bilateral solver, that we will call the “robust bilateral solver” (RBS). The RBS minimizes the following loss:

$$\underset{\mathbf{x}}{\text{minimize}} \frac{\lambda}{2} \sum_{i,j} \hat{W}_{i,j} (x_i - x_j)^2 + \sum_i \rho(x_i - t_i) \quad (19)$$

Where $\rho(\cdot)$ is some robust error function. Unless $\rho(\cdot)$ is defined as (weighted) squared-error like in Equation 1, the optimization problem in Equation 19 does not have a closed-form solution. However, Equation 19 can be solved using iteratively reweighted least squares (IRLS) [3] by repeatedly linearizing the loss

function around the current estimate of \mathbf{x} and then solving a least-squares problem corresponding to that linearization. The linearized version of Equation 19 in IRLS takes on exactly the same form as Equation 1, where the “confidence” \mathbf{c} is replaced by the “weight” generated during IRLS. Thus we can produce a robust bilateral solver by wrapping the standard bilateral solver in a loop, recomputing the weight at each iteration.

The RBS can be used with any standard robust loss function that would be used for M-estimation. We use the Geman-McClure loss function [10], a smooth approximation to the ℓ_0 -norm, whose estimator $\rho(\cdot)$ and corresponding weight in IRLS are:

$$\rho(e_i) = \frac{e_i^2}{\sigma_{gm}^2 + e_i^2} \quad w(e_i) = \frac{2\sigma_{gm}^2}{(\sigma_{gm}^2 + e_i^2)^2} \quad (20)$$

where σ_{gm} is a scale parameter. Pseudocode for the RBS with this Geman-McClure loss can be found in Algorithm 3.

Algorithm 3 The Geman-McClure Robust Bilateral Solver

Input:

`solve(·)` // The bilateral solver
`t` // The input target vector
`cinit` // The input confidence vector
 σ_{gm} // The scale parameter of our Geman-McClure function
 n // The number of IRLS iterations

Output:

$\hat{\mathbf{x}}$ // \mathbf{x} such that Equation 19 is minimized

```

1:  $\mathbf{c} \leftarrow \mathbf{c}_{init}$ 
2: while  $i < n$  do
3:    $\hat{\mathbf{x}} \leftarrow \text{solve}(\mathbf{t}, \mathbf{c})$ 
4:    $\mathbf{e} \leftarrow (\hat{\mathbf{x}} - \mathbf{t})$ 
5:    $\mathbf{c} \leftarrow \frac{2\sigma_{gm}^2}{(\sigma_{gm}^2 + (\mathbf{e} \odot \mathbf{e}))^2}$ 
6:    $i \leftarrow i + 1$ 
7: end while

```

Because our loss function is non-convex and therefore sensitive to initialization, our RBS interface takes as input some initial confidence \mathbf{c}_{init} that is used in the first least-squares solve, and then overwritten by the IRLS weights in subsequent iterations. Because the IRLS / robust M-estimation loop of the RBS is sensitive to initialization, we can improve performance by setting the initial weights used in the IRLS loop c_{init} to reflect some noise model computed from the input to the solver. In the case of our Middlebury stereo experiment, we construct this initial confidence according to the heuristic observation that accurate depth maps tend to have contiguous image regions with low-variance depths. To identify such regions we compute an edge-aware measure of depth variance on the input depth map Z using the recursive formulation of the domain transform [9], which is a

simple and fast edge-aware filter. To see how we do this, let us first review the definition of variance:

$$\text{Var}(x) = \text{E}[x^2] - (\text{E}[x])^2 \quad (21)$$

Where E is the expectation operator. Any (normalized) linear image filtering operation with non-negative weights can be thought of as computing some local expectation of the input image for every pixel. When coupled with an edge-aware image filtering operation such as the domain transform (DT), we can compute a local edge-aware variance V of our input depth map Z as follows:

$$V = \text{DT}(Z^2) - (\text{DT}(Z))^2 \quad (22)$$

We initialize our input confidence by exponentiating this scaled, negated variance:

$$\mathbf{c}_{init} = \exp\left(-\frac{V}{2\sigma_{dt}^2}\right) \quad (23)$$

Our parameters are tuned to maximize performance on the Middlebury training set: $\sigma_{xy} = \sigma_{rgb} = 32$, $\sigma_{dt} = 2$. We additionally modify this initial confidence using the observation that the depth map at one side of an image of each stereo pair generally has a poorly estimated depth, as the true match for those pixels are often not present in the other image of the stereo pair. We address this by setting the leftmost 80 columns of \mathbf{c}_{init} to 0, thereby causing them to be initially ignored. See Figure 1 for a visualization of the initial confidence estimated using this procedure on one of the test-set depth maps from the Middlebury stereo benchmark v3.

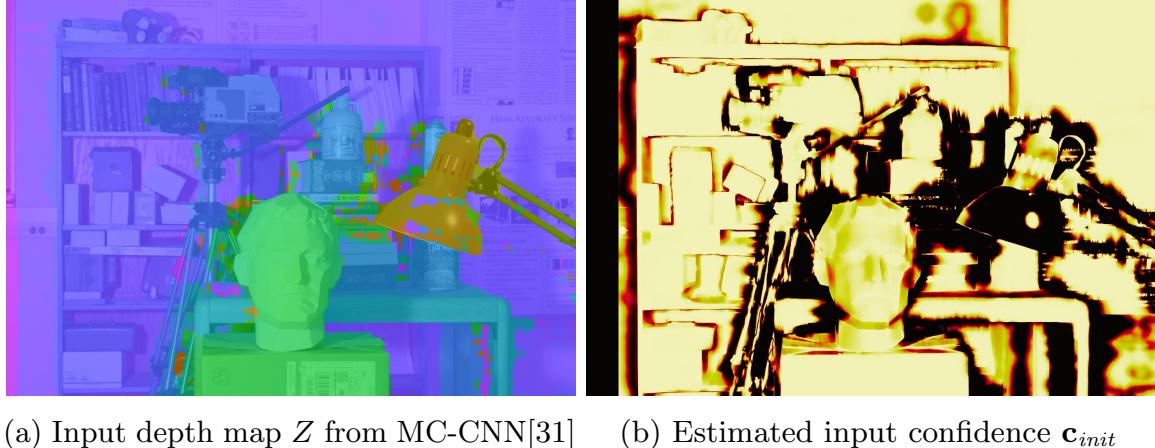


Fig. 1: When processing the depth maps produced by other stereo algorithms, we first use an edge-aware variance estimation technique to produce initial confidence measures used by our robust bilateral solver. This procedure downweights contiguous image regions with inconsistent depths.

4 Multiple Output Channels

The tasks which we apply our bilateral solver to consist of inputs with a single channel (depth, stereo, etc) or many channels (21, for our semantic segmentation benchmark). As mentioned previously, our model generalizes straightforwardly to problems with n -channel “target” inputs by simply decomposing into n independent optimization problems with the same A matrix and different \mathbf{b} vectors. By concatenating these \mathbf{y} and \mathbf{b} vectors into matrices, this can be rewritten as one large linear system:

$$AY = B \quad (24)$$

For applications such as semantic segmentation this B matrix is often very wide, but also very low-rank — many object categories may have extremely low probabilities in the input, and some object categories may be strongly correlated. In these cases we can produce a reduced linear system with fewer right-hand-sides by producing a low-rank approximation to B , solving the resulting linear system, and then expanding our low-rank solution back to input space. This can dramatically speed up convergence, often by an order of magnitude, as many semantic segmentation algorithms often only assign a non-trivial probability to a small fraction of object categories for any given scene. Our approach is similar to a simplified and approximate version of “block” conjugate gradient [23].

We use a rank-revealing QR factorization [5] to reduce our B matrix, which is often used for similar tasks due to its stability and speed.

$$BP = QR \quad (25)$$

Where P is a permutation matrix, Q is an orthonormal basis for the columns of B , and R is an upper triangular matrix. With this, let us construct a modified factorization of B :

$$B = Q'R' \quad (26)$$

Where we construct R' by taking $R P^T$ and then shuffling the rows of the resulting matrix such the rows of R' have non-increasing Euclidean norms. Q' is Q where the columns of Q have also been shuffled. Let us define a vector \mathbf{m} which is the “mass” (squared Euclidean norm) of each row of R' :

$$m_i = \sum_j R'^2_{i,j} \quad (27)$$

Let us define some tolerance ϵ , which is an upper bound on the residual tolerance fraction of B that we are willing to tolerate. We find the largest value of t such that:

$$\sum_{i=0}^{i < t} m_i \leq (1 - \epsilon) \sum_i m_i \quad (28)$$

With this we can drop the least important rows of R and columns of Q :

$$\tilde{Q} \approx Q'[:, 1:t] \quad \tilde{R} \approx (R' P^T)[1:t, :] \quad (29)$$

We can use \tilde{Q} to solve the reduced linear system, and then multiply the solution to that linear system by \tilde{R} to approximate the solution to the original system:

$$Y = (A^{-1}\tilde{Q})\tilde{R} \quad (30)$$

In our semantic segmentation experiment, For small values of $\epsilon = 0.01$ the output from this approximate is often indistinguishable from exact solution, despite being $\approx 5\times$ faster on average.

5 Results

Here we present many additional results for the four tasks explored in the main paper, in the form of additional figures and expanded tables of results, as well as details regarding the evaluation of baseline techniques.

5.1 Stereo

In the paper we demonstrated the value of our robust bilateral solver as a post-processing procedure for stereo algorithms, using the Middlebury Stereo Benchmark V3 [26]. See Figures 2-6 for examples of test set images from the Middlebury dataset in which we post-process the output of the state-of-the-art MC-CNN stereo technique[31], which we obtained using the publicly available source accompanying the technique. See Figures 7 and 8 for additional results on training set images from the Middlebury dataset, where we evaluate against a wider selection of stereo algorithms. See Figures 9 and 10 for results in which we have post-processed the training-set output of those these top-performing stereo algorithms with baseline edge-aware filtering or depth-map enhancement techniques. We present these results on the training set because the Middlebury benchmark makes the training-set output of other stereo algorithms freely available, while obtaining test-set results requires the code or cooperation of the authors of each technique, or re-implementing these techniques. The depth maps corresponding to each baseline stereo technique were therefore downloaded from the Middlebury website. The output corresponding to each post-processing technique we compare against were produced by us, while taking care to tune all parameters to maximize performance on the Middlebury training set (the geometric mean of all 6 error metrics) for the MC-CNN output (these parameters are then used for all other stereo techniques). For complete transparency, we will detail the procedure used to produce each baseline results:

TF This is the tree-filtering technique of Yang [29], which we ran ourselves using the publicly available code¹. The parameter settings used in these experiments are $\sigma_r = 0.25$ with nonlocal filtering, which experimentation showed to be the optimal parameter settings for this task.

¹ http://www.cs.cityu.edu.hk/~qiyang/publications/software/tree_filter.zip

FGF This is our own Matlab implementation of the (color) fast guided filter [11], which we found to be faster than the implementation built into Matlab 2015, and significantly faster than the released code² while producing identical results. The parameters were tuned for this task, with a box filter size of 8, $\epsilon = 0.01^2$, and a subsampling factor of 4.

WMF This is the weighted median filter approach of Ma *et al.*[21], which we ran using the publicly available code³. Because this technique was designed for a similar use case to its use here, we used the same parameter settings as were used in the paper: $r = \max(\text{width}, \text{height})$, $\epsilon = 0.01^2$, followed by a median filter.

DT This is the recursive formulation of the domain transform [9] with optimally-tuned parameters for this task ($\sigma_r = 64$, $\sigma_s = 32$).

For all Middlebury experiments the parameters of our RBS were: $\sigma_{xy} = \sigma_l = \sigma_{uv} = 4$, $\lambda = 0.25$, $\sigma'_{xy} = \sigma'_{rgb} = 4$, $\sigma_{gm} = 1$, (the scale of the Geman-McClure loss function) and we performed 32 iterations of IRLS.

On our test-set results, we present six error metrics for each image: bad-1% (the percent of pixels whose disparity is wrong by more than 1), MAE (the mean absolute error of the disparity map) and RMSE (the root mean squared error of each disparity map), for all pixels and for only non-occluded pixels. We generally see a reduction in the MAE and RMSE error metrics, usually by around 50%, and a relatively unchanged bad-1% metrics, which suggests that our solver has a substantial and positive impact on quality. This improvement is quite clear when visualizing the output depth maps, as shown in Figures 2–6, suggesting that the “all or nothing” bad-1% metric does not seem to correlate well with the visual quality of the depth map. Our error reduction is roughly consistent across all choices of stereo techniques used to produce the input depth maps to our algorithm (though for all post-processing technique the improvement is most significant for MC-CNN, as that is the environment in which parameters were tuned). The baseline depth post-processing techniques we evaluate against often do reduce some error measures, though all tend to increase the bad 1% error measure, and none produce as substantial a reduction of MAE and RMSE as our approach. The improvement of our approach over those baseline approaches is evident upon visual inspection, as can be seen in Figures 9 and 10. Our test-set errors were taken from the Middlebury website, while training-set errors were produced with our own evaluation code. We do not report runtime for this task, as the runtime of our technique and all baselines is dominated by the time taken by the MC-CNN technique common among all entries.

5.2 Stereo-based Defocus

Though the primary focus of this work is the bilateral solver and not the “defocus” task of Barron *et al.*[2], we would be remiss to omit a comparison of our technique

² [http://research.microsoft.com/en-us/um/people/kahe/eccv10/
fast-guided-filter-code-v1.rar](http://research.microsoft.com/en-us/um/people/kahe/eccv10/fast-guided-filter-code-v1.rar)

³ [http://research.microsoft.com/en-us/um/people/kahe/iccv13wmf/matlab_
wmf_release_v1.rar](http://research.microsoft.com/en-us/um/people/kahe/iccv13wmf/matlab_wmf_release_v1.rar)

with the optimization technique presented in [2], given the similarities between the two techniques. The stereo algorithm of [2] performs brittle block-matching on a rectified stereo pair to produce, for each pixel, an interval (lower and upper values $[l_i, u_i]$ parametrizing an interval) of likely depths for that pixel. This data term appears to have been chosen for its efficacy, and because its simple piecewise-linear form allowed this pixel-space loss to be easily “splatted” into bilateral space to construct a convex (though non-linear) optimization problem. This data term is not compatible with our bilateral solver, and so we must convert it to the expected input: a per-pixel “target” value and “confidence” measure.

Table 1: Our RBS improves depth map quality for a variety of state-of-the-art stereo algorithms on the Middlebury Stereo Dataset V3 [26]. Test-set numbers were taken from the Middlebury website, while training-set numbers were produced by our own evaluation code.

Method	All			NoOcc				
	bad	1%	MAE	RMSE	bad	1%	MAE	RMSE
Test Set								
MC – CNN[31]	28.1	17.9	55.0	18.0	3.82	21.3		
MC – CNN[31]+RBS	28.2	8.19	29.9	18.9	2.67	15.0		
Training Set								
MC-CNN[31]	20.07	5.93	18.36	10.42	1.94	9.07		
MC-CNN[31] + TF[29]	29.15	5.67	16.18	20.15	2.17	7.71		
MC-CNN[31] + FGF[11]	32.29	5.91	16.32	23.62	2.42	7.98		
MC-CNN[31] + WMF[21]	33.37	5.30	15.62	26.29	2.32	8.22		
MC-CNN[31] + DT[9]	25.17	5.69	16.53	15.53	2.01	7.72		
MC-CNN[31] + RBS (Ours)	19.49	2.81	8.44	11.33	1.40	5.23		
MeshStereo[32]	21.25	3.83	10.75	15.13	2.21	7.86		
MeshStereo[32] + TF[29]	27.37	3.81	9.91	21.70	2.26	7.03		
MeshStereo[32] + FGF[11]	29.28	3.96	10.03	23.44	2.38	7.12		
MeshStereo[32] + WMF[21]	32.16	3.87	10.10	27.18	2.39	7.30		
MeshStereo[32] + DT[9]	23.97	3.77	10.12	17.92	2.18	7.23		
MeshStereo[32] + RBS (Ours)	21.43	3.22	8.72	15.52	2.03	6.68		
TMAP[25]	23.08	3.98	11.55	16.29	2.24	7.61		
TMAP[25] + TF[29]	27.47	3.94	10.90	20.88	2.26	7.04		
TMAP[25] + FGF[11]	32.16	4.17	10.79	25.66	2.50	6.99		
TMAP[25] + WMF[21]	33.90	4.11	11.01	28.39	2.56	7.61		
TMAP[25] + DT[9]	24.93	3.86	10.92	17.99	2.15	7.02		
TMAP[25] + RBS (Ours)	22.79	3.31	9.44	16.09	2.06	6.72		
SGM[13]	24.37	3.85	10.68	17.91	2.44	8.04		
SGM[13] + TF[29]	31.47	3.82	9.55	25.36	2.51	6.95		
SGM[13] + FGF[11]	34.40	4.05	9.66	28.42	2.72	7.09		
SGM[13] + WMF[21]	35.47	3.97	9.99	30.48	2.76	7.72		
SGM[13] + DT[9]	28.78	3.85	9.90	22.24	2.49	7.28		
SGM[13] + RBS (Ours)	24.18	3.44	9.21	17.95	2.31	7.06		

We simply use the average of the upper and lower interval as the target and the exponentiated negative length of the interval as the confidence:

$$t_i = (l_i + u_i)/2 \quad c_i = \exp(l_i - u_i) \quad (31)$$

This simple reparametrization combined with our bilateral solver produces an effective stereo technique for this “defocus” task, while being roughly $3.5\times$ faster than the already-efficient solver of [2]. See Figures 11 and 12 for a visualization of our performance relative to [2], using some of the stereo pairs from [2]. Though this is only a modest improvement over [2], it is reassuring that our general-purpose bilateral solver not only applies to a host of problems in computer vision, but also performs at least as well as its much more specialized precursor technique. Note that [2] reported large errors on the Middlebury V2 dataset, while our technique can be used for both accurate depth maps and pleasing graphical effects.

5.3 Depth Superresolution

In Table 2 we present an expanded form of the depth superresolution results presented in the main paper, in which the task is broken down into its constituent images and scale factors, as was done in past work [8,16]. Our bilateral solver appears to be roughly the second or third most accurate technique, after [16] and sometimes [18]. But it should be noted that these top-performing techniques [14,16,18] are all dictionary-based techniques which are trained on a great deal of data, while our technique produces competitive results despite the simplicity of our model and our lack of training. What is perhaps the most important property of our model is its speed — our technique $600\text{-}3000\times$ faster than the three most accurate techniques, despite having comparable accuracy to all but [16]. Our performance is most competitive when the upsampling factor is large ($16\times$) which is when the task is inherently most difficult. Additionally, those techniques with speeds comparable to or better than the bilateral solver [9,29,11,22] produce error rates that are 25-40% greater than our approach. The only techniques with significantly faster runtimes than the bilateral solver are standard image interpolation techniques (bilinear, bicubic, etc) which produce low-quality output, and our implementation of the domain transform [9] which is a highly-optimized, vectorized, and multi-threaded implementation, unlike our own technique and all other techniques we compare against. More conventional implementations of the domain transform appear to have runtimes comparable to our own, or that of the guided filter[12]. See Figures 13-15 for example output for this task.

For transparency and completeness we evaluated against as many applicable baseline techniques as was possible. This is difficult, as though many papers present results for denoising or upsampling different versions of the Middlebury dataset, there is not one standard benchmark which is universally adopted. To this end, we build upon the benchmarked used in [8], but augment it heavily. We inherit some results from past work [8,16] which is necessary due to the lack of publicly available source code for many techniques. This means that

some runtimes are produced on other hardware platforms, and so these runtimes should be considered accordingly. Our own runtimes (indicated in Table 2 in white) were benchmarked on a 2012 HP Z420 workstation (Intel Xeon CPU E5-1650, 3.20GHz, 32 GB RAM), and our algorithm is implemented in standard, single-threaded C++. Runtimes highlighted with a color come from some other hardware platform other than our reference hardware. For clarity, we will now detail the experimental conditions of each baseline technique, both in terms of parameter settings and hardware environments. For all experiments in which we produced the model output, the parameters of each algorithm were tuned to minimize the average error on this task by starting with the baseline parameters in the publicly available code, and then performing coordinate descent on each parameter, halving and doubling each and taking the new parameter setting if the average error decreased. For all models in which the code was ran by us, the runtime reported is the median of the runtime over all images in the dataset, unless the model’s runtime is resolution-dependent, in which case we report the geometric mean of the runtimes.

Table 2: Performance on the depth superresolution task of [8]. We report root-means-squared error, as was done in [16], along with the geometric mean of those errors over the entire dataset. Times other than our own are indicated by colors: Green runtimes are from [8], blue runtimes are from [16], pink runtimes are from [20], and the teal runtime is from [21]. The beige runtimes were produced by us, but on a different, faster computer. Algorithms which use external training data are indicated with a dagger.

Method	Art				Books				Moebius				Avg.	Time (sec)
A) Nearest Neighbor	6.55	7.41	8.87	11.24	6.16	6.32	6.63	7.36	6.59	6.78	6.98	7.48	7.26	0.003
B) Bicubic	5.32	6.00	7.15	9.35	5.00	5.17	5.46	5.98	5.34	5.52	5.66	6.07	5.91	0.007
C)†Kiechle <i>et al.</i> [14]	2.82	5.10	6.83	10.80	3.83	5.10	6.12	8.43	4.50	5.73	6.64	8.96	5.86	450
D) Bilinear	4.57	5.53	6.99	9.45	3.94	4.31	4.71	5.38	4.19	4.55	4.83	5.37	5.16	0.004
E) Liu <i>et al.</i> [19]	4.10	5.43	7.69	11.36	3.08	3.87	4.82	6.46	3.18	4.04	5.11	6.62	5.10	16.60
F) Shen <i>et al.</i> [27]	3.49	4.62	6.13	8.68	2.86	3.48	4.43	5.57	2.29	3.07	4.22	5.43	4.24	31.48
G) Diebel & Thrun [6]	3.49	4.41	6.24	9.11	2.06	3.00	4.06	5.13	2.13	3.10	4.14	5.12	3.98	—
H) Chan <i>et al.</i> [4]	3.44	4.38	5.98	8.41	2.09	2.77	3.78	5.45	2.08	2.69	3.73	5.33	3.83	3.02
I) GuidedFilter[12,8]	3.55	4.31	5.59	8.22	2.37	2.73	3.42	4.52	2.48	2.82	3.54	4.53	3.76	23.89
J) Min <i>et al.</i> [22]	3.65	4.08	5.09	7.91	2.85	2.77	2.97	3.81	3.46	3.25	3.20	3.86	3.74	0.383
K)†Lu & Forsyth[20]	4.30	5.05	6.33	7.94	2.17	2.71	3.30	4.29	2.16	2.50	3.15	4.10	3.69	20
L) Park <i>et al.</i> [24]	3.76	4.48	5.80	8.75	1.95	2.60	3.30	4.86	1.96	2.49	3.21	4.48	3.61	24.05
M) Domain Transform [9]	3.95	4.76	6.14	8.49	1.80	2.40	3.23	4.44	1.83	2.40	3.35	4.64	3.56	0.021
N) Ma <i>et al.</i> [21]	3.27	3.99	5.08	7.39	2.39	2.70	3.09	3.77	2.55	2.84	3.23	3.81	3.49	18
O) GuidedFilter(Matlab)[12]	3.60	4.25	5.49	7.99	2.39	2.52	2.89	3.89	2.50	2.57	2.90	3.61	3.47	0.434
P) Zhang <i>et al.</i> [33]	4.15	4.22	5.03	7.86	1.96	2.24	3.13	4.80	1.80	2.19	3.22	4.90	3.45	1.346
Q) FastGuidedFilter[11]	3.40	4.16	5.46	7.97	2.08	2.51	3.04	3.95	2.13	2.55	3.08	3.79	3.41	0.225
R) Yang 2015 [29]	3.27	4.15	5.46	7.93	2.00	2.38	3.00	4.04	2.25	2.57	3.13	4.00	3.41	0.304
S) Yang <i>et al.</i> 2007 [30]	3.01	3.92	4.85	7.57	1.87	2.38	2.86	4.26	1.92	2.41	2.96	4.37	3.25	—
T) Farbman <i>et al.</i> [7]	3.14	4.00	5.30	7.70	1.76	2.26	2.90	3.88	1.79	2.29	2.98	3.93	3.19	6.11
U) JBU [1,15]	3.17	4.02	5.37	7.59	1.83	2.18	2.80	4.00	1.83	2.13	2.71	3.76	3.14	1.98
V) Ferstl <i>et al.</i> [8]	3.19	4.06	5.08	7.61	1.52	2.21	2.47	3.54	1.47	2.03	2.58	3.50	2.93	140
W)†Li <i>et al.</i> [18]	3.02	3.12	4.43	7.43	1.18	1.70	2.55	3.58	1.11	1.59	2.28	3.50	2.56	700
X)†Kwon <i>et al.</i> [16]	0.87	1.30	2.05	3.56	0.51	0.75	1.14	1.88	0.57	0.89	1.37	2.14	1.21	300
Y) BS(Ours)	2.93	3.79	4.95	7.13	1.39	1.84	2.38	3.29	1.38	1.80	2.38	3.23	2.70	0.234

- A** This is nearest-neighbor interpolation, and the reported runtime is that of the Matlab `imresize()` function on our workstation.
- B** This is bicubic interpolation, and the reported runtime is that of the Matlab `imresize()` function on our workstation.
- C** This is the technique of Kiechle *et al.*[14], with errors and runtimes taken directly from [16].
- D** This is bilinear interpolation, and the reported runtime is that of the Matlab `imresize()` function on our workstation.
- E** This is the Joint Geodesic Upsampling technique of [19], which we ran ourselves using the publicly available code⁴. This technique performs poorly for this task, possibly because it assumes noise-free input. We used the default parameters of the code, which we found optimal for this task: interval = 3, $\sigma = 0.5$, $\lambda_1 = 10$, $\lambda_2 = 1$.
- F** This is the Mutual-Structure technique of Shen *et al.*[27], which we ran ourselves using the publicly available code⁵. This technique performs poorly on this task, which appears to be due to the low-frequency nature of the depth-map noise inherent in this depth super-resolution task, unlike the high-frequency, per-pixel noise used in the experiments of [27]. The parameters were tuned for optimal performance on our task: $\epsilon_I = 10^{-5}$, $\epsilon_G = 2 \times 10^{-4}$, $\lambda_I = 1$, $\lambda_G = 4$, 20 iterations, window size of 2. A more aggressive smoothing effect could be achieved using larger windows sizes which were a function of the upsampling factor, but these oversmoothed depths had significantly higher errors than was produced using these settings.
- G** This is the technique presented in Diebel & Thrun [6], which was used as a baseline algorithm in [8]. We did not run this code ourselves, and produced these error rates using the precomputed output from [8], which is why we do not have runtimes.
- H** This is an implementation of Chan *et al.*[4], which was used as a baseline algorithm in [8]. We did not run this code ourselves, and produced these error rates using the precomputed output from [8], and reproduced the runtime quoted in [8].
- I** This is presumably an implementation of the guided filter [12], which was used as a baseline algorithm in [8]. We did not run this code ourselves, and produced these error rates using the precomputed output from [8], and we took the runtime quoted in [8]. Because the quoted runtime seems unusually slow for the guided filter, we ran two of our own evaluations of the guided filter on this task (models O and Q) which produced lower errors rates and significantly faster runtimes.
- J** This is the weighted least-squares approach of Min *et al.*[22], which we ran ourselves using publicly available code⁶. The parameters were tuned for optimal performance on our task: $\sigma = 0.00125 \times 2^f$ (where f is the upsampling factor), $\lambda = 30^2$, iteration = 3, attenuation = 4.

⁴ <http://www.merl.com/research/license/>

⁵ <http://www.cse.cuhk.edu.hk/leojia/projects/mutualstructure/index.html>

⁶ <https://github.com/soundsilence/ImageSmoothing>

- K** This is the technique presented in Lu & Forsyth 2015 [20]. The authors of [20] ran their own code on the task presented here on their own computer (with comparable specifications to our own) at our request, sent us the output, and reported their approximate runtime, which we report here.
- L** This is the technique presented in Park *et al.*[24], which was used as a baseline algorithm in [8]. We did not run this code ourselves, and produced these error rates using the precomputed output from [8], which is why we do not have runtimes.
- M** This is the recursive formulation of the domain transform [9] with optimally-tuned parameters ($\sigma_r = 64$, $\sigma_s = 8f$). We used our own highly optimized implementation of this code, implemented in Halide, with heavily parallelization and vectorization. This is unlike most other baselines and our own algorithm, which are single-threaded unoptimized code.
- N** This is the weighted median filter of [21], which we ran ourselves using the publicly available code⁷. We found the color version of the code (which we used for its improved accuracy) to perform slowly, so the runtime we cite is taken by extrapolating from the quoted runtime for the CPU implementation of [21], (60ms per megapixel-disparity) on a comparable computer to ours, suggesting that runtime on this task would be ~ 18 seconds per image (~ 200 disparities, ~ 1.5 megapixels per image). The parameters were tuned for optimal performance on our task: filter size = 2^f (where f is the upsampling factor) and $\epsilon = 0.02^2$.
- O** This is the implementation of the guided filter [12] which is built into the 2015 version of Matlab. The parameters were tuned for this task, “Neighborhood-Size” = 3×2^f , (where f is the upsampling factor) and “DegreeOfSmoothing” = 0.5.
- P** This is the weighted median filter of [33], which we ran ourselves using the publicly available code⁸. Because the code was only available as a compiled windows binary, we were forced to use a different computer for this evaluation, and so the runtime is for a significantly faster computer than was used for our other evaluation: A dual-processor Xeon CPU-E5-2690 v3 2.6GHz with 64 GB of ram. The runtime should therefore be considered a lower bound on the actual runtime for the reference hardware. We used the default parameter settings provided with the reference code for our experiments, which we found to perform optimally for our benchmark.
- Q** This is our own Matlab implementation of the (color) fast guided filter [11], which we found to be faster than the implementation built into Matlab 2015, and significantly faster than the released code⁹ while producing identical results. The parameters were tuned for this task, with a box filter size of 2^f (where f is the upsampling factor), $\epsilon = 0.02^2$, and a subsampling of 2.

⁷ http://research.microsoft.com/en-us/um/people/kahe/iccv13wmf/matlab_wmf_release_v1.rar

⁸ http://www.cse.cuhk.edu.hk/~leojia/projects/fastwmedian/download/JointWMF_mex.zip

⁹ <http://research.microsoft.com/en-us/um/people/kahe/eccv10/fast-guided-filter-code-v1.rar>

Our experimentation suggests that the subsampling could be made more aggressive with little drop in accuracy, but because the algorithm requires that the filter size must be divisible by the subsampling factor, the most aggressive subsampling we could use for all upsampling factors was 2. From this we can assume that a $4 \times$ speedup may be possible.

- R** This is the technique presented in Yang [29], which we ran ourselves using the publicly available code¹⁰. Because the code was only available as a compiled windows binary, like in Model P this evaluation was performed on a different, faster workstation, and so the runtime should be considered a lower bound on the actual runtime for the reference hardware. The parameter settings used in these experiments are $\sigma_r = 2^{(f-6)}$ (where f is the upsampling factor) and nonlocal filtering, which experimentation showed to be the optimal parameter settings for this task.
- S** This is the technique presented in Yang *et al.*[30], which was used as a baseline algorithm in [8]. We did not run this code ourselves, and produced these error rates using the precomputed output from [8], which is why we do not have runtimes.
- T** This is the WLS model of Farbman *et al.*[7], which we ran ourselves using the publicly available code¹¹, with the code was modified to use non-log color reference images. The parameter settings used in these experiments are $\lambda = 5000 \times 2^{(f-4)}$ (where f is the upsampling factor) and $\alpha = 2$, which experimentation showed to be the optimal parameter settings for this task.
- U** This model is standard joint bilateral upsampling [15] using a permutohedral lattice [1] with optimally-tuned parameters ($\sigma_{rgb} = 8$, $\sigma_x = \sigma_y = 8f$, where f is the upsampling factor). These errors and runtimes were produced by us using a C++ implementation of the permutohedral lattice¹².
- V** These errors and runtimes were taken from the website¹³ of [8]. Since the publication of this paper, it has been revealed that the error cited in the publication is MAE, not RMSE, so the errors in the paper are incorrect. For details, see the supplement¹⁴ of [16].
- W** This is the technique of Li *et al.*[18], with errors and runtimes taken directly from [16].
- X** This is the technique of Kwon *et al.*[16], with errors and runtimes taken directly from [16]. We attempted to obtain the output depth maps or source code from [16], but the authors were unable to comply with our request, and so we are unable to present the results of [16] in our figures or perform a more detailed analysis of how well the algorithm performs, nor can we reproduce

¹⁰ http://www.cs.cityu.edu.hk/~qiyang/publications/software/tree_filter.zip

¹¹ <http://www.cs.huji.ac.il/~danix/epd/wlsFilter.m>

¹² <https://code.google.com/archive/p/imagemstack/>

¹³ https://rvlab.icg.tugraz.at/project_page/project_tofusion/project_tofsuperresolution.html

¹⁴ <https://sites.google.com/site/datadrivendepthcvpr2015/>

these results. Details on the lack of availability of the source and output from this model can be found on the project website¹⁵.

5.4 Colorization

See Figure 16 for a comparison of our technique on the colorization task, compared against Levin *et al.*[17]. The images and code from [17] were taken from the paper’s website¹⁶, and were produced using the exact least-square solver presented in the paper for still image processing. Note that [17] also presents an approximate multigrid optimization which they use for colorizing videos, which is $6\times$ faster than their exact still image approach but presumably produces inexact results for the still image task. Several of the techniques we evaluate against for the depth super-resolution task can also be used for colorization, presumably with the same tradeoffs between accuracy and speed seen in that experiment. Since there is traditionally no ground-truth for this task, we do not present an exhaustive evaluation here.

5.5 Semantic Segmentation

See Figure 17 for additional examples of our bilateral solver and competing techniques applied to the semantic segmentation task. The bilateral solver produces more edge-aware results than CRF-based techniques on well-isolated objects.

¹⁵ <https://sites.google.com/site/datadrivendepthcvpr2015/>

¹⁶ www.cs.huji.ac.il/~yweiss/Colorization/

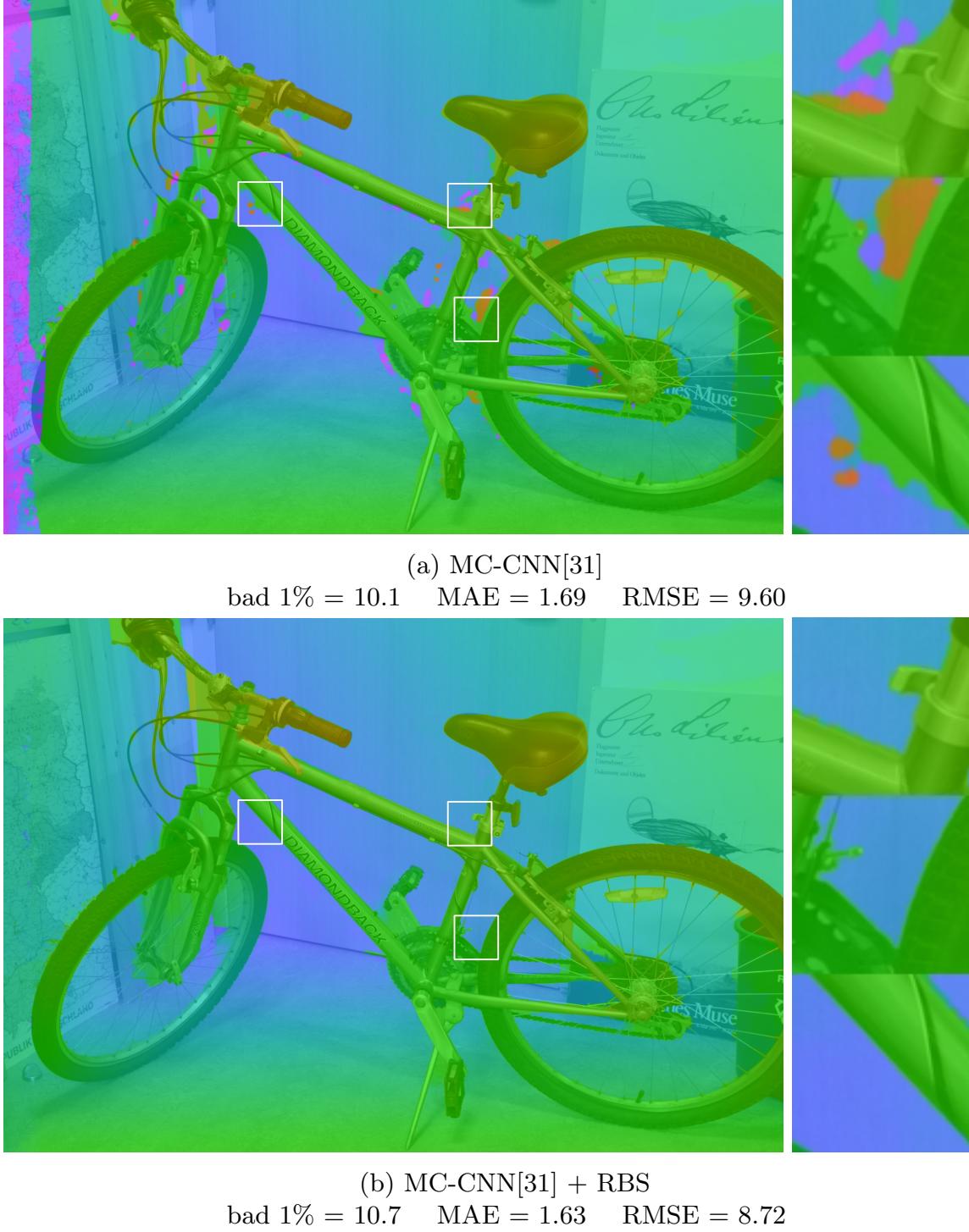


Fig. 2: Results on the test set of the Middlebury Stereo Dataset V3 [26] where our robust bilateral solver is used to improve the depth map predicted by the state-of-the-art MC-CNN technique[31]. On the top we have the depth map produced by [31] (with zoomed in regions) which is used as the target in our solver. On the bottom we have the output of our solver, where we see that quality is significantly improved. We report the error for each depth map in terms of the percent of pixels whose disparity is off by more than 1 (“bad 1%”), the mean-absolute-error of the disparity, and the root-mean-squared-error of the disparity.

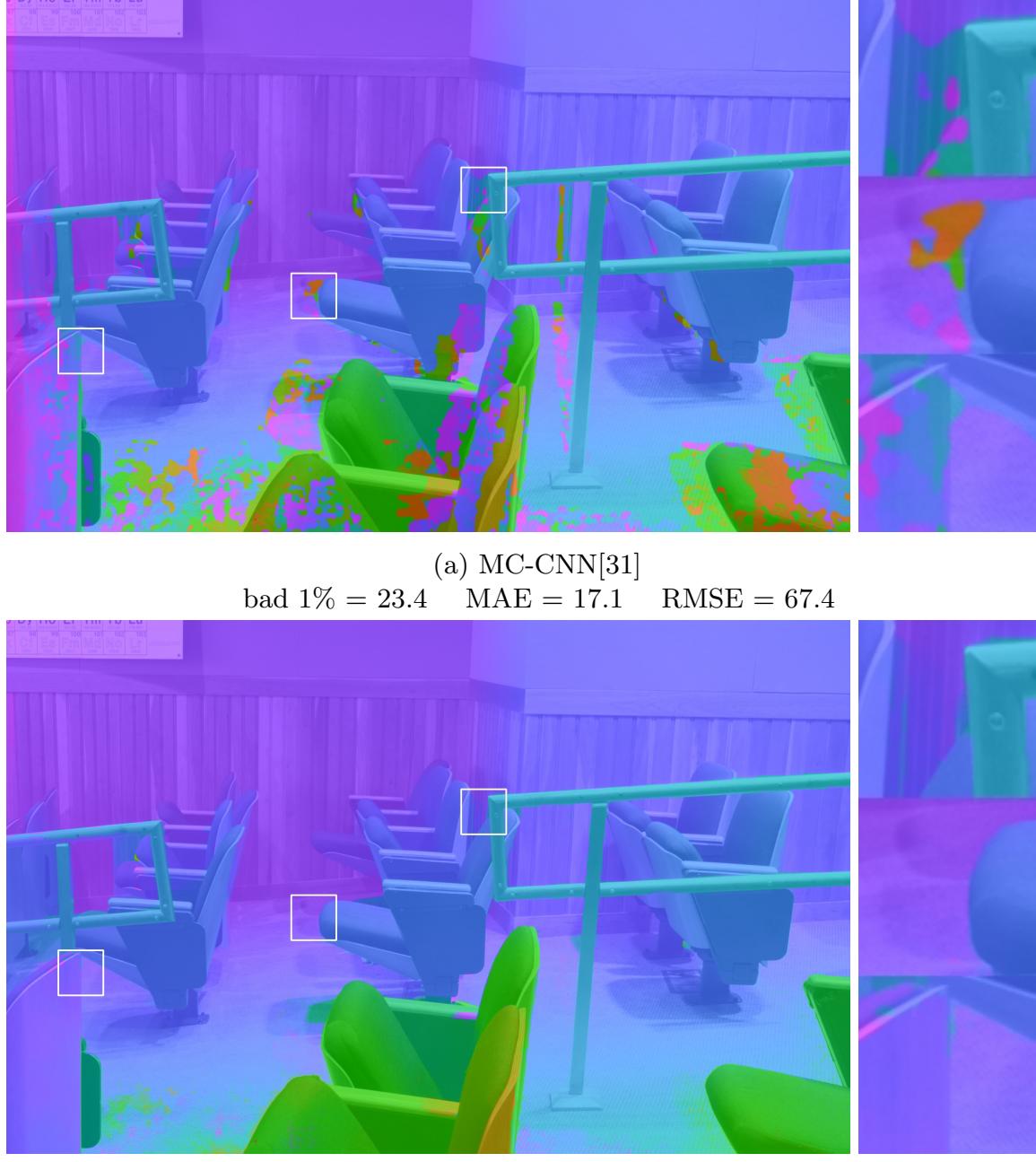


Fig. 3: More results on the test set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 2.

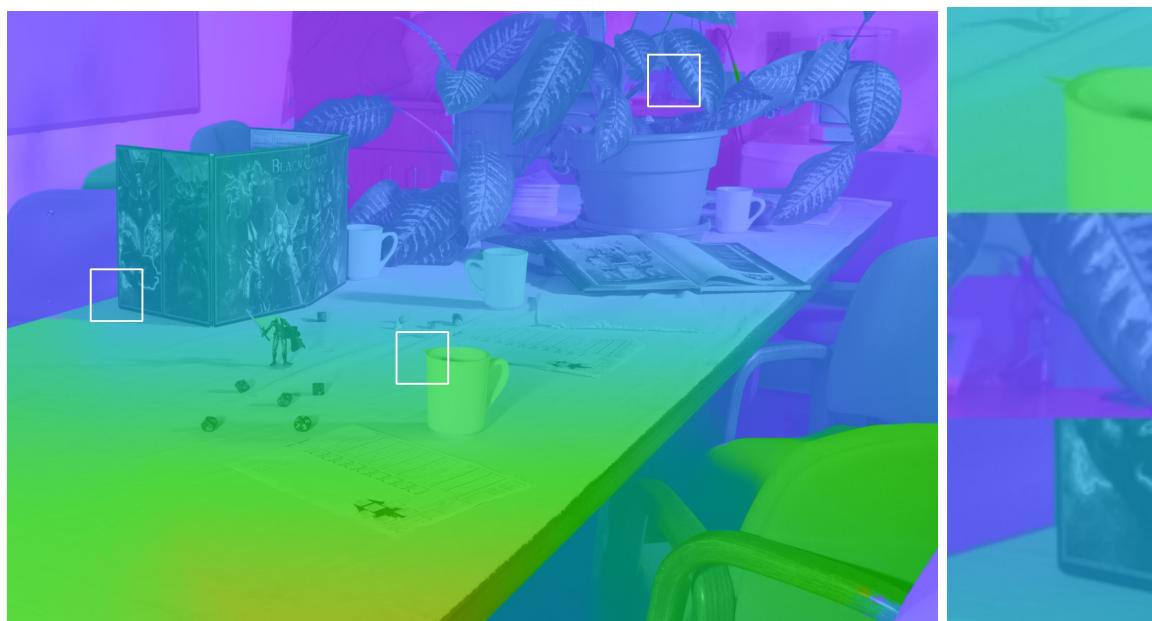
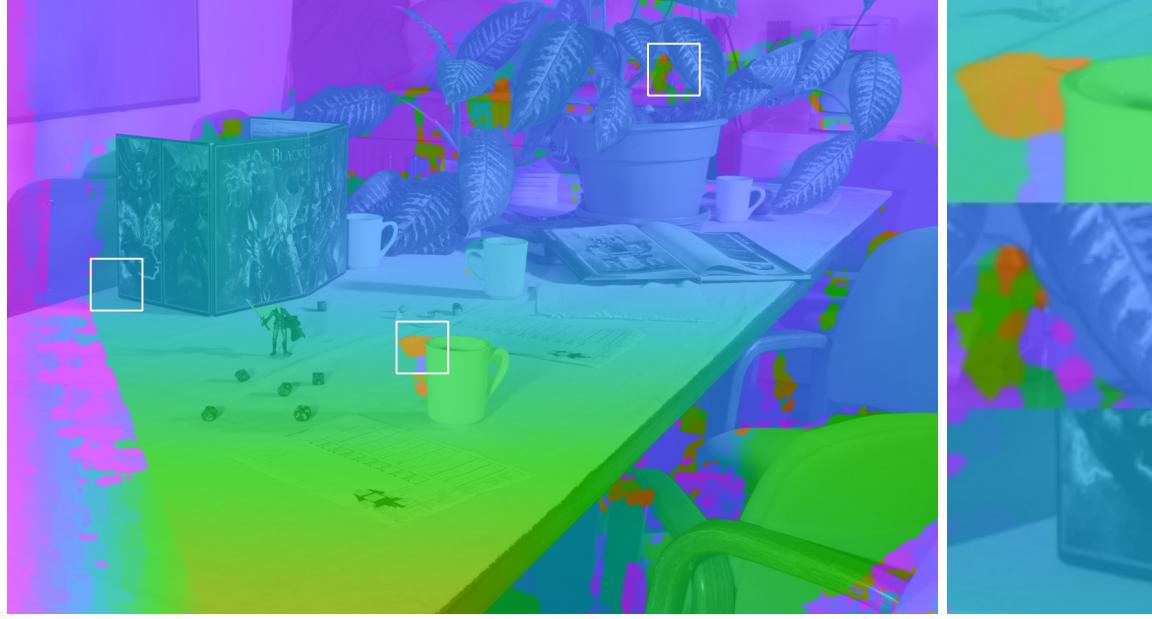


Fig. 4: More results on the test set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 2.

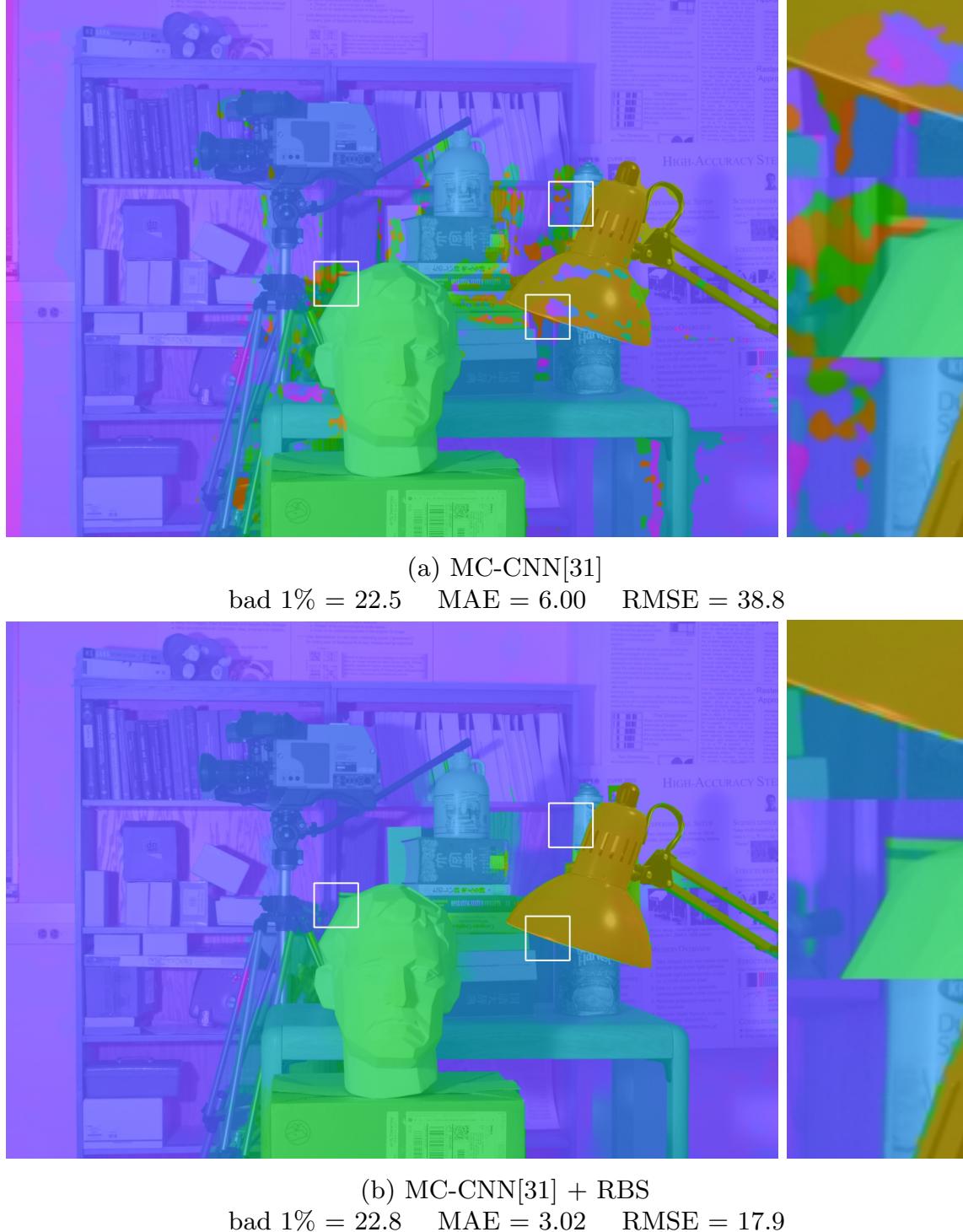


Fig. 5: More results on the test set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 2.

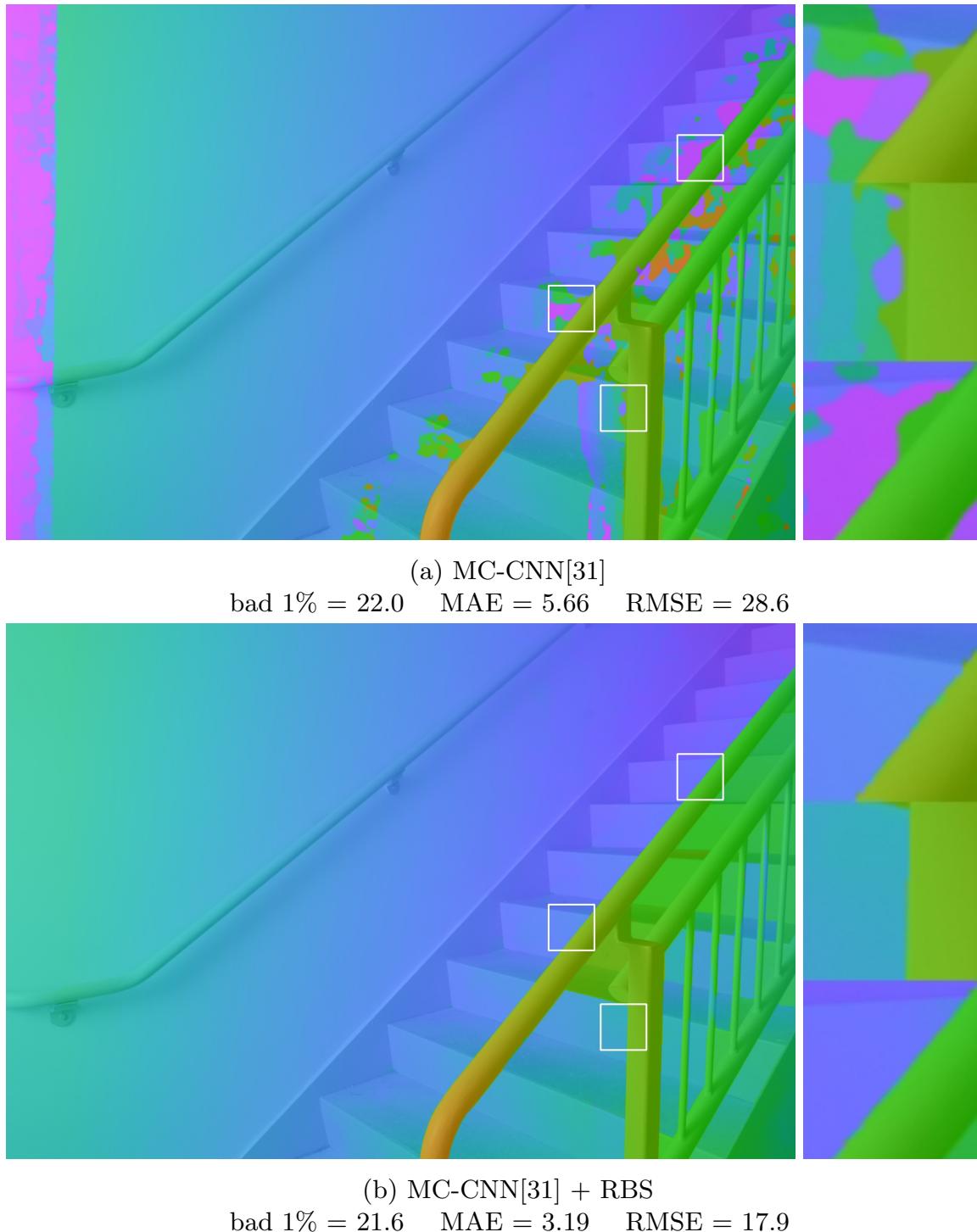


Fig. 6: More results on the test set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 2.

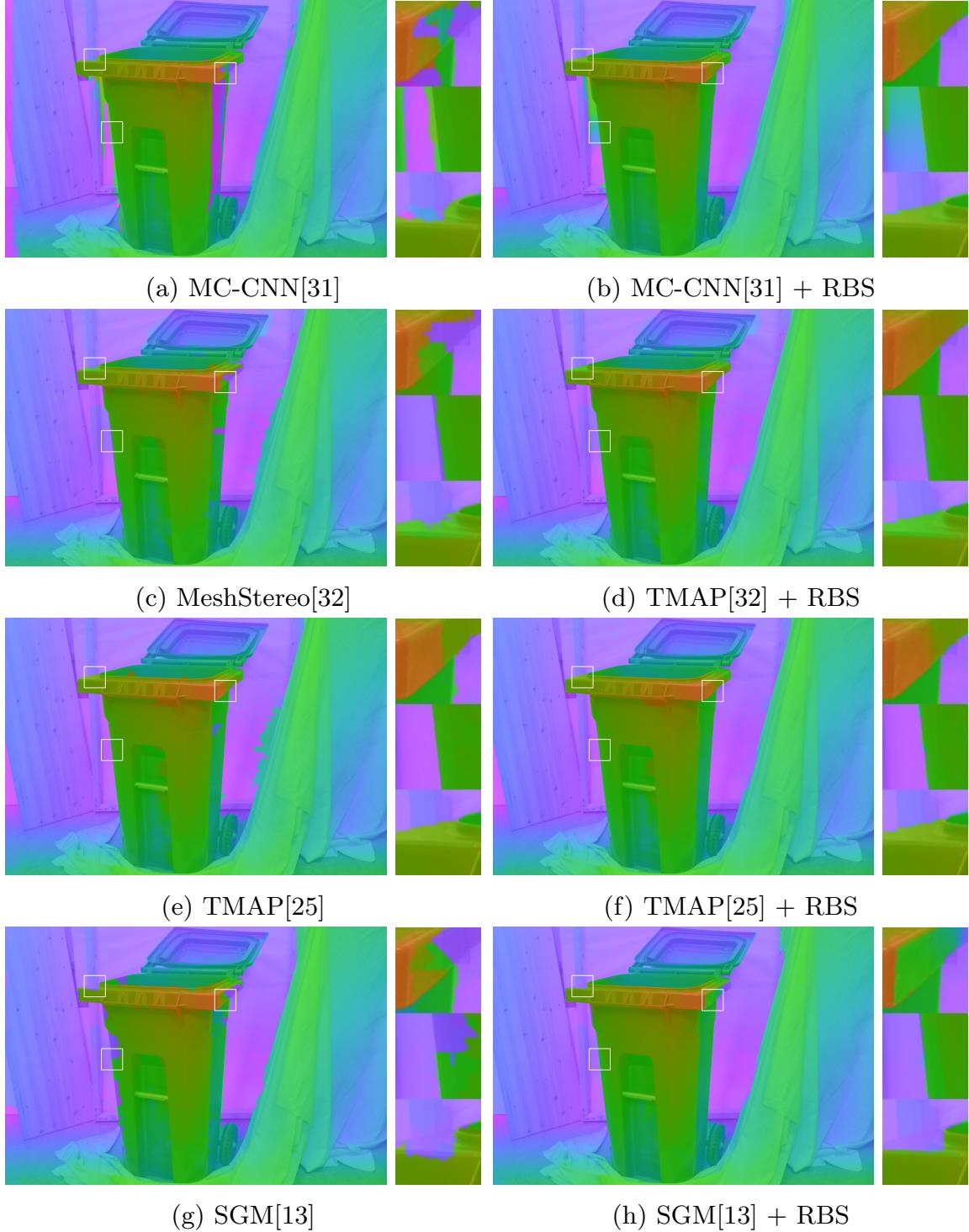


Fig. 7: Results on the training set of the Middlebury Stereo Dataset V3 [26] where our robust bilateral solver is used to improve the depth map predicted by four top-performing stereo algorithms. On the left we have the depth map produced by each stereo algorithm (with zoomed in regions) which is used as the target in our solver. On the right we have the output of our solver, where we see that quality is significantly improved.



Fig. 8: More results on the training set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 7.

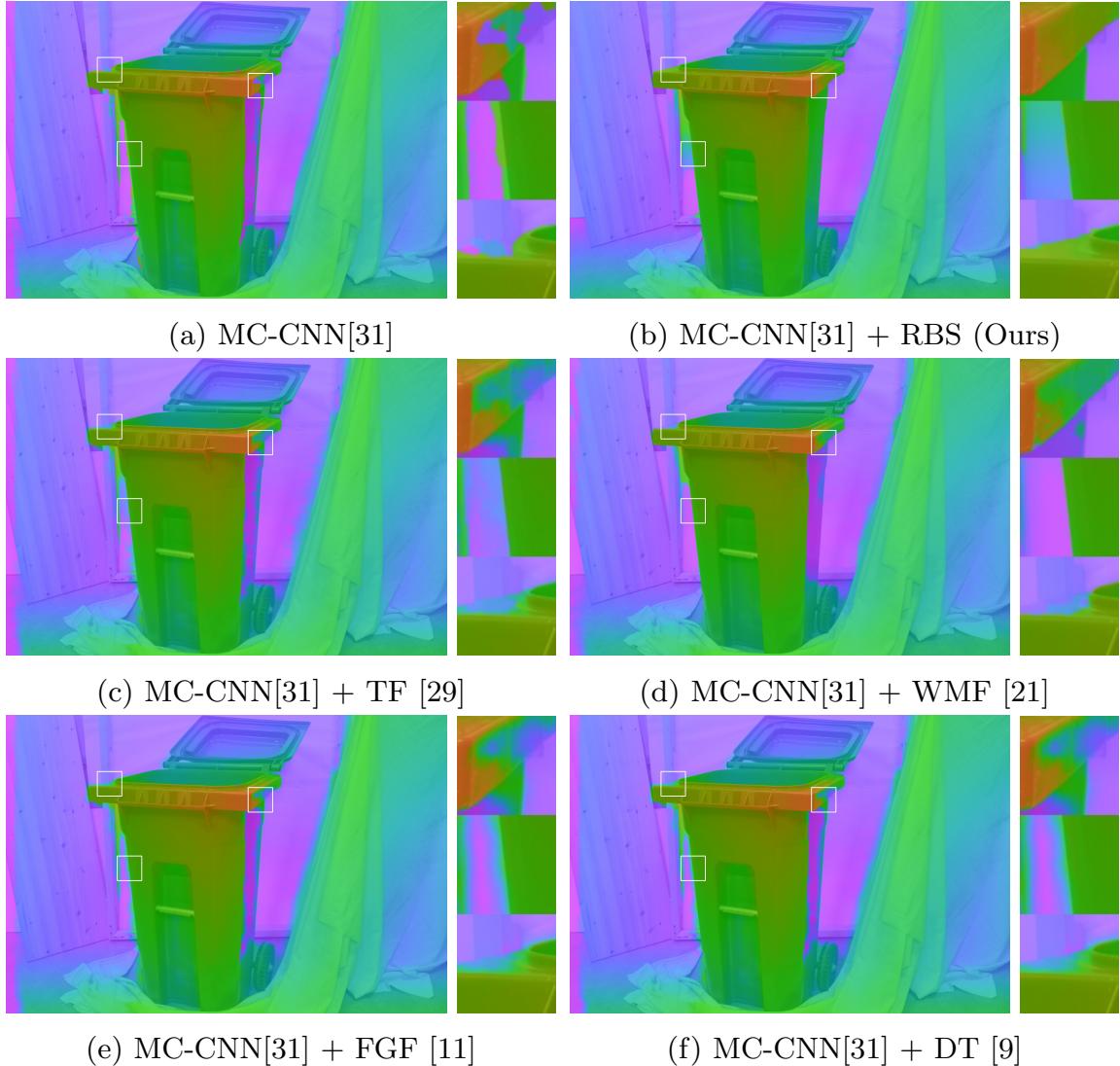


Fig. 9: Results on the training set of the Middlebury Stereo Dataset V3 [26], in which we compare our robust bilateral solver against several baseline techniques for post-processing depth maps. Our model’s output exhibits much higher quality than the input or any baseline, especially at the discontinuities shown in the cropped regions.



Fig. 10: More results on the training set of the Middlebury Stereo Dataset V3 [26] in the same format as Figure 9.

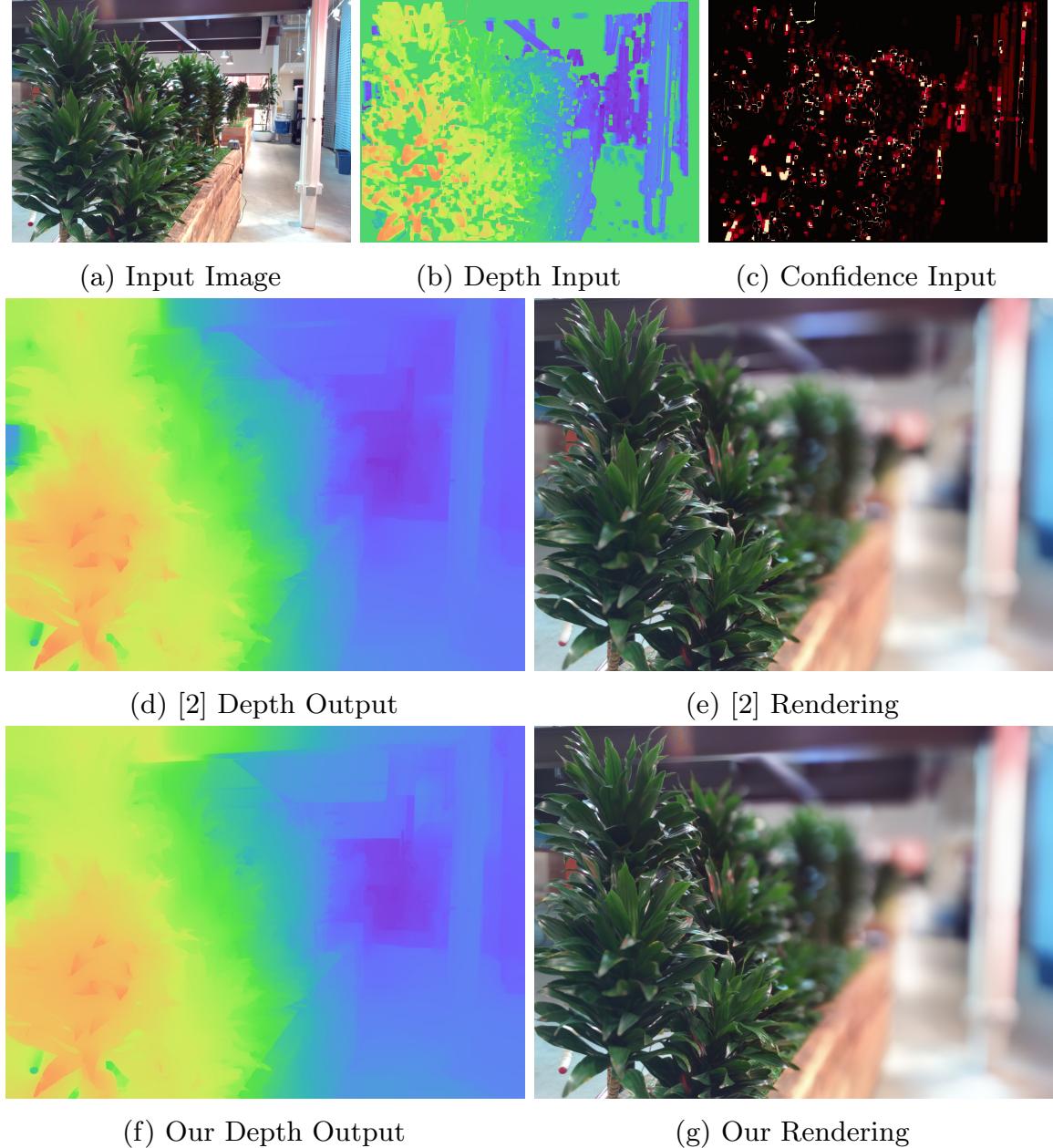


Fig. 11: Given the reparametrized input into the solver of [2] (b and c) our solver can produce depth maps and defocused renderings of a comparable quality to [2], while being $3.5\times$ faster.

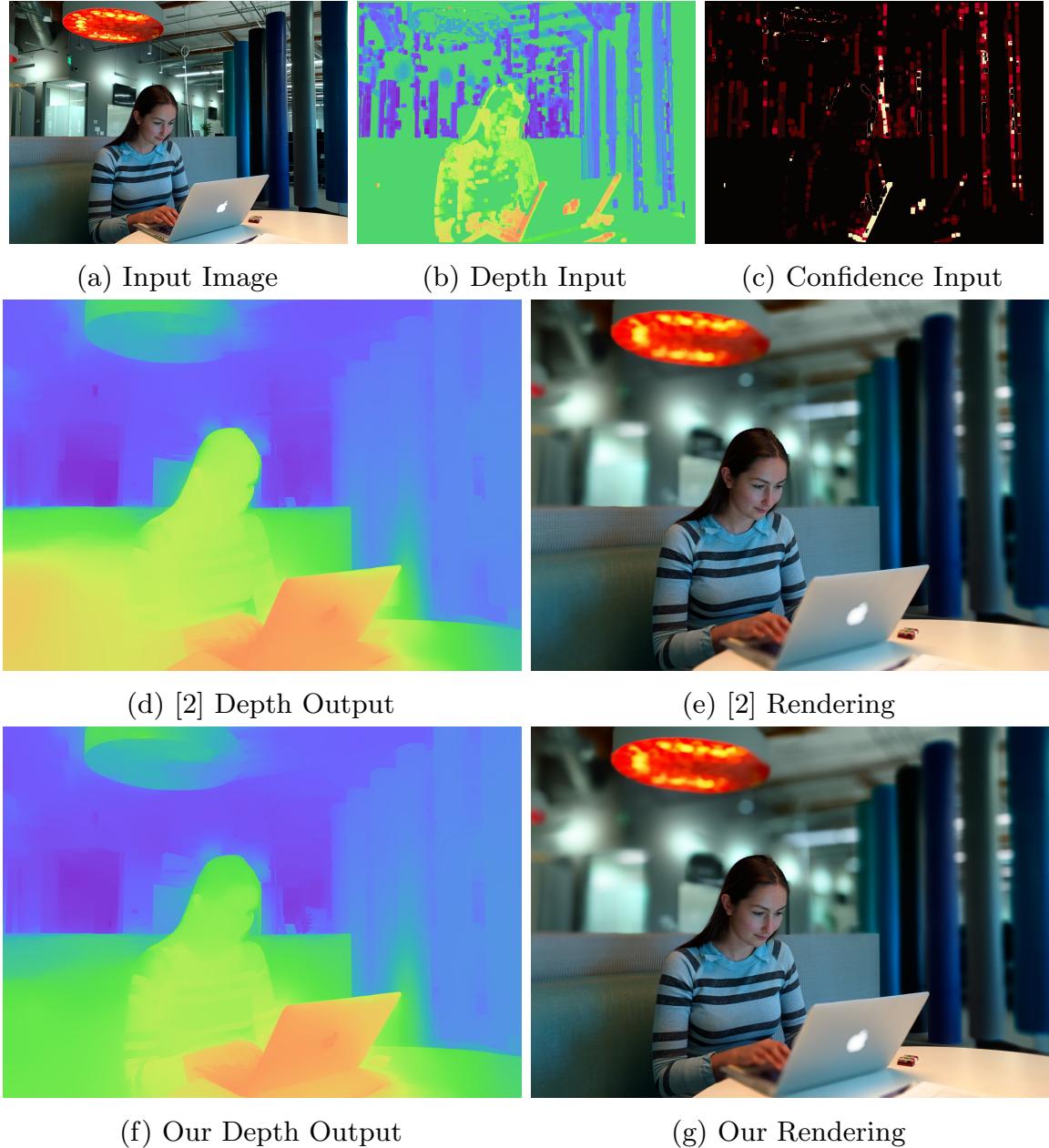


Fig. 12: Additional output for the stereo defocus task of [2], in the same format as Figure 11.

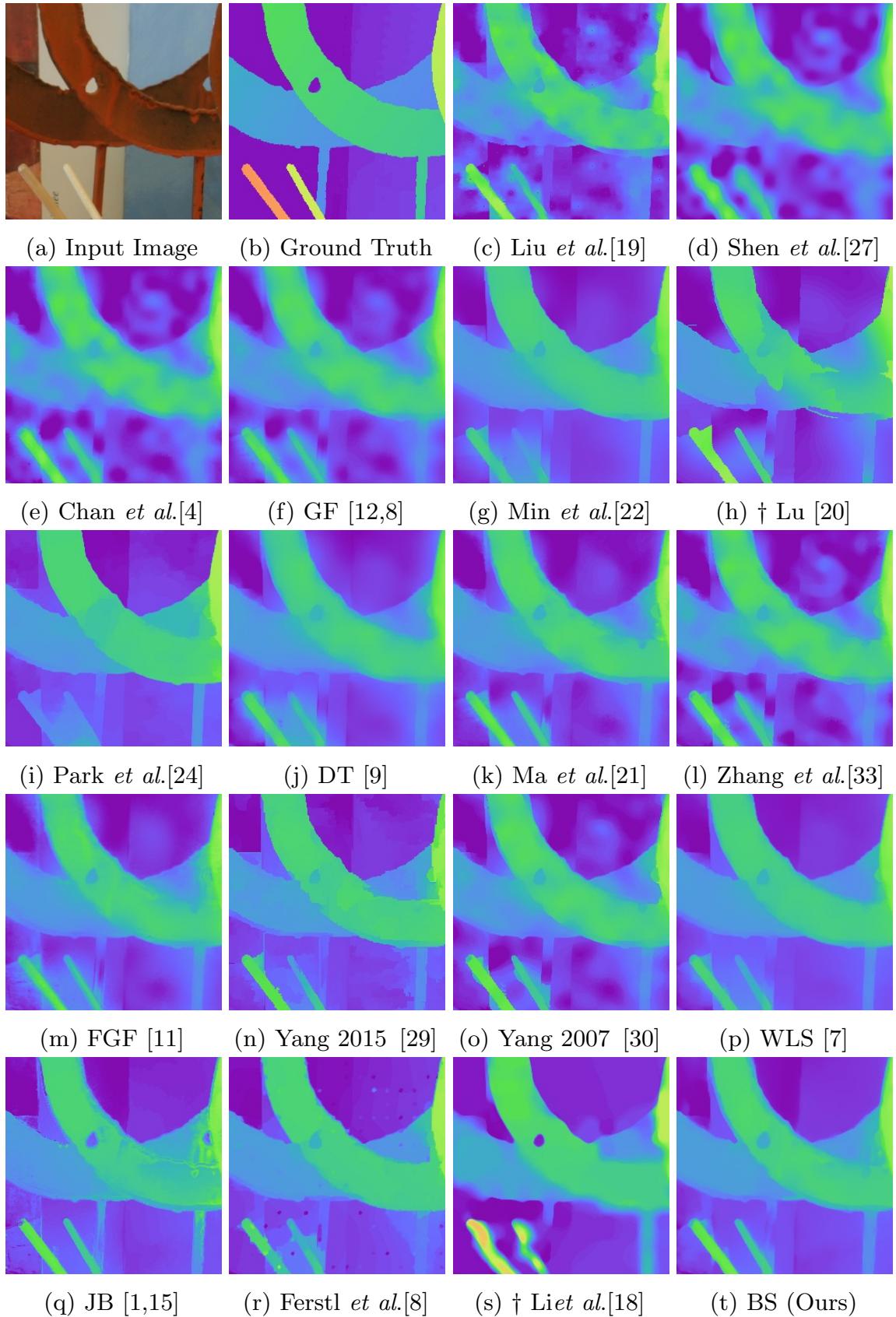


Fig. 13: Results for the depth superresolution task. Algorithms are sorted according to their average error on this benchmark, from upper left to lower right. Algorithms which use external training data are indicated with a dagger.

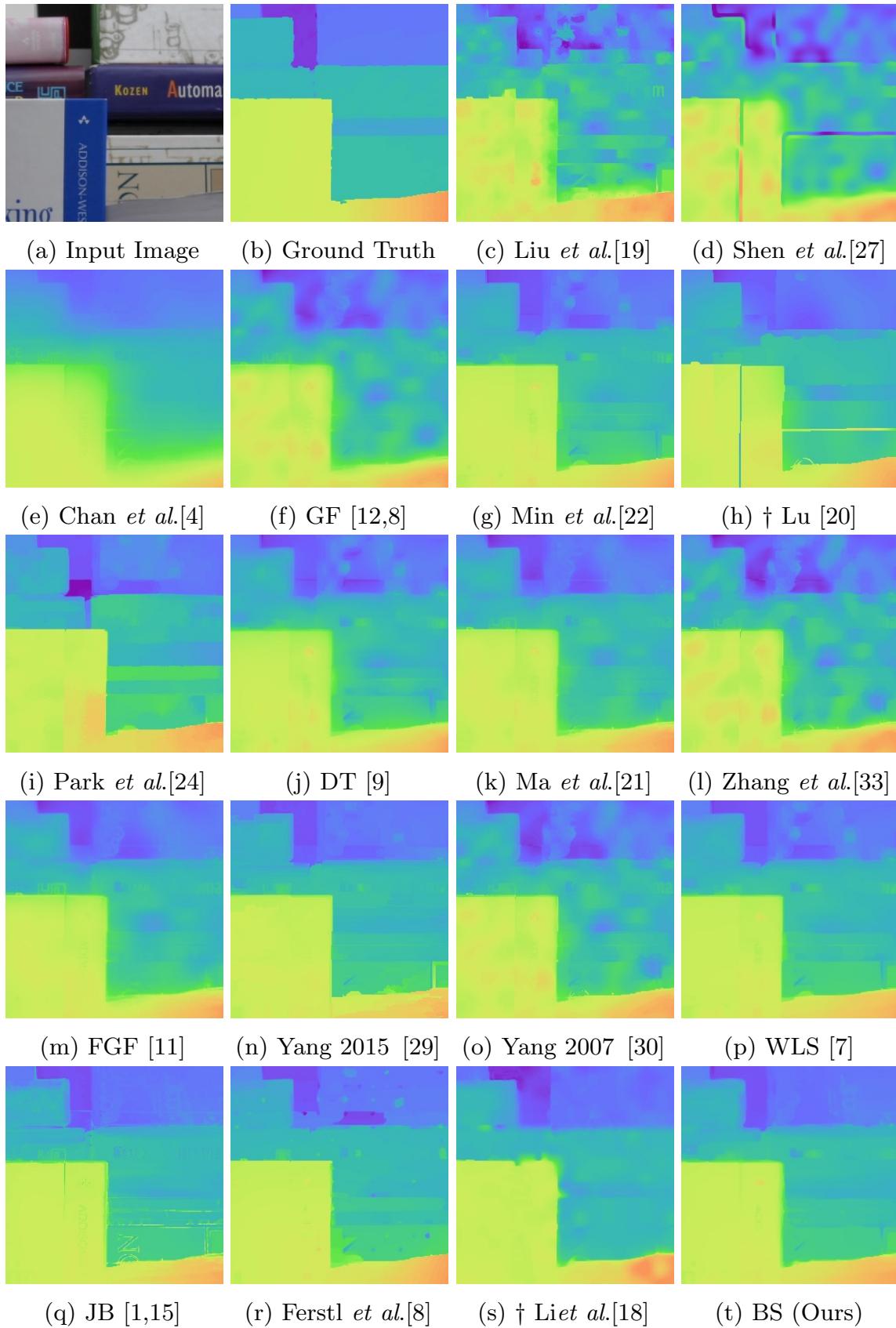


Fig. 14: More results for the depth superresolution task.

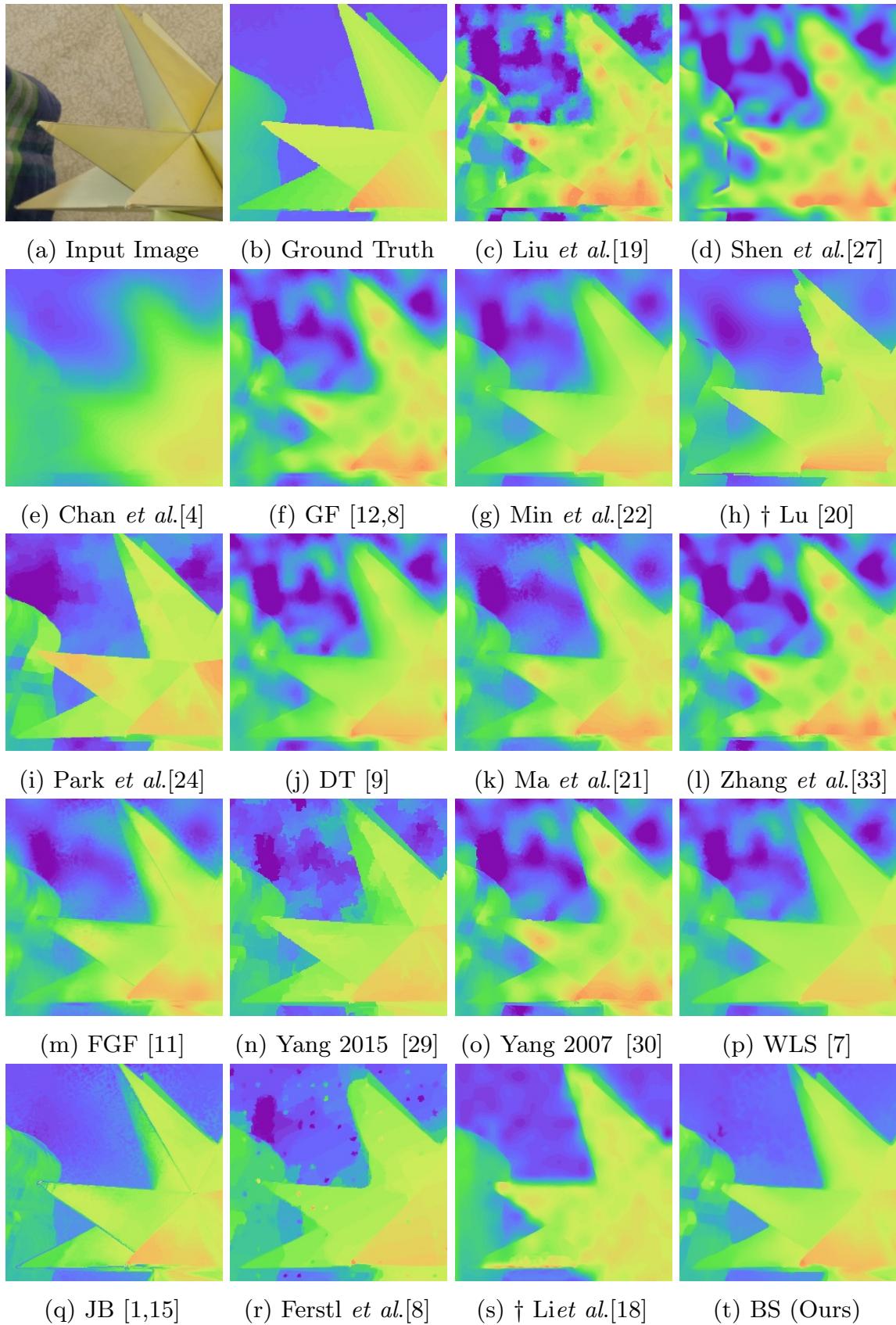


Fig. 15: More results for the depth superresolution task.



Fig. 16: Results for the colorization task, using the images and scribbles from [17]. Our algorithm's output is nearly indistinguishable from that of [17], while being $95\times$ faster.

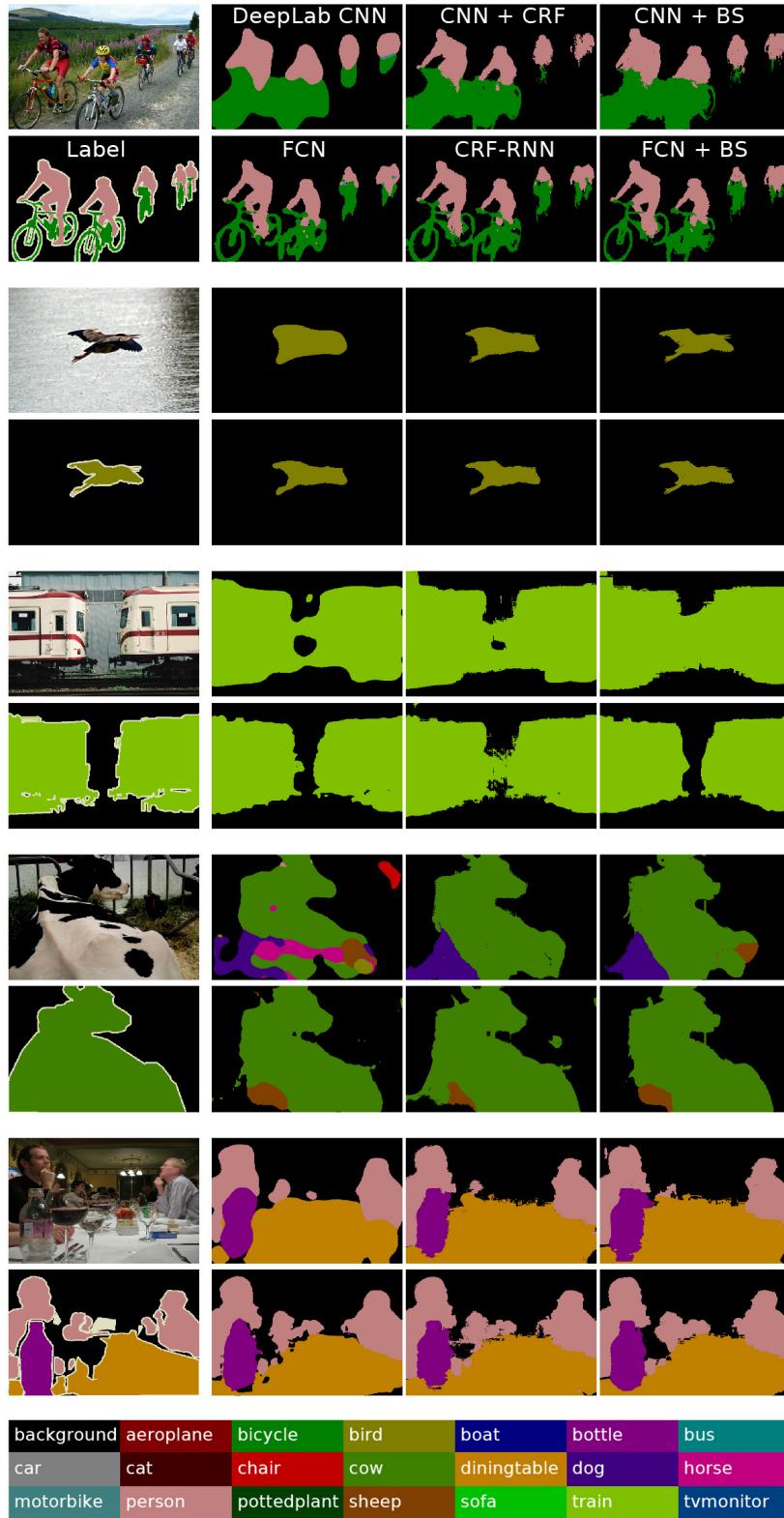


Fig. 17: Additional semantic segmentation results on Pascal VOC12 validation images. FCN refers to the fully convolutional network component of the end-to-end trained CRF-RNN model. While the CRF-augmented DeepLab model (top rows) and CRF-RNN model (bottom rows) perform best overall, the bilateral solver produces better results on isolated objects (second and third images) at a fraction of the cost.

References

1. Adams, A., Baek, J., Davis, M.A.: Fast high-dimensional filtering using the permutohedral lattice. *Eurographics* (2010)
2. Barron, J.T., Adams, A., Shih, Y., Hernández, C.: Fast bilateral-space stereo for synthetic defocus. *CVPR* (2015)
3. Beaton, A.E., Tukey, J.W.: The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data. *Technometrics* (1974)
4. Chan, D., Buisman, H., Theobalt, C., Thrun, S.: A noise-aware filter for real-time depth upsampling. *ECCV Workshops* (2008)
5. Chan, T.F.: Rank revealing qr factorizations. *Linear algebra and its applications* (1987)
6. Diebel, J., Thrun, S.: An application of markov random fields to range sensing. *NIPS* (2005)
7. Farbman, Z., Fattal, R., Lischinski, D., Szeliski, R.: Edge-preserving decompositions for multi-scale tone and detail manipulation. *SIGGRAPH* (2008)
8. Ferstl, D., Reinbacher, C., Ranftl, R., Ruether, M., Bischof, H.: Image guided depth upsampling using anisotropic total generalized variation. *ICCV* (2013)
9. Gastal, E.S.L., Oliveira, M.M.: Domain transform for edge-aware image and video processing. *SIGGRAPH* (2011)
10. Geman, S., McClure, D.E.: Statistical methods for tomographic image reconstruction. *Bull. Internat. Statist. Inst.* (1987)
11. He, K., Sun, J.: Fast guided filter. *CoRR abs/1505.00996* (2015)
12. He, K., Sun, J., Tang, X.: Guided image filtering. *ECCV* (2010)
13. Hirschmüller, H.: Accurate and efficient stereo processing by semi-global matching and mutual information. *CVPR* (2005)
14. Kiechle, M., Hawe, S., Kleinsteuber, M.: A joint intensity and depth co-sparse analysis model for depth map super-resolution. *ICCV* (2013)
15. Kopf, J., Cohen, M.F., Lischinski, D., Uyttendaele, M.: Joint bilateral upsampling. *SIGGRAPH* (2007)
16. Kwon, H., Tai, Y.W., Lin, S.: Data-driven depth map refinement via multi-scale sparse representation. *CVPR* (2015)
17. Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. *SIGGRAPH* (2004)
18. Li, Y., Xue, T., Sun, L., Liu, J.: Joint example-based depth map super-resolution. *ICME* (2012)
19. Liu, M.Y., Tuzel, O., Taguchi, Y.: Joint geodesic upsampling of depth images. *CVPR* (2013)
20. Lu, J., Forsyth, D.: Sparse depth super resolution. *CVPR* (2015)
21. Ma, Z., He, K., Wei, Y., Sun, J., Wu, E.: Constant time weighted median filtering for stereo matching and beyond. *ICCV* (2013)
22. Min, D., Choi, S., Lu, J., Ham, B., Sohn, K., Do, M.N.: Fast global image smoothing based on weighted least squares. *Transactions on Image Processing* (2014)
23. O’Leary, D.P.: The block conjugate gradient algorithm and related methods. *Linear Algebra and its applications* (1980)
24. Park, J., Kim, H., Tai, Y.W., Brown, M.S., Kweon, I.: High quality depth map upsampling for 3d-tof cameras. *ICCV* (2011)
25. Psota, E., Kowalcuk, J., Mittek, M., Pérez, L.: Map disparity estimation using hidden markov trees. *ICCV* (2015)

26. Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X., Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth. GCPR (2014)
27. Shen, X., Zhou, C., Xu, L., Jia, J.: Mutual-structure for joint filtering. ICCV (2015)
28. Shewchuk, J.R.: An introduction to the conjugate gradient method without the agonizing pain. Tech. rep., Carnegie Mellon University (1994)
29. Yang, Q.: Stereo matching using tree filtering. PAMI (2015)
30. Yang, Q., Yang, R., Davis, J., Nistér, D.: Spatial-depth super resolution for range images. CVPR (2007)
31. Zbontar, J., LeCun, Y.: Computing the stereo matching cost with a convolutional neural network. CVPR (2015)
32. Zhang, C., Li, Z., Chen, Y., Cai, R., Chao, H., , Rui, Y.: Meshstereo: A global stereo model with mesh alignment regularization for view interpolation. ICCV (2015)
33. Zhang, Q., Xu, L., Jia, J.: 100+ times faster weighted median filter (wmf). CVPR (2014)