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" $\mathbf{c}$:

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

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:

$$\begin{aligned} \sum_i c_i (x_i - t_i)^2 =& (\mathbf{x} - \mathbf{t})^{\mathrm{T}} \mathrm{diag}(\mathbf{c})(\mathbf{x} - \mathbf{t}) && (2) \\ =& \mathbf{x}^{\mathrm{T}} \mathrm{diag}(\mathbf{c})\mathbf{x} - 2\mathbf{x}^{\mathrm{T}} \mathrm{diag}(\mathbf{c})\mathbf{t} + \mathbf{t}^{\mathrm{T}} \mathrm{diag}(\mathbf{c})\mathbf{t} && (3) \\ =& \mathbf{x}^{\mathrm{T}} \mathrm{diag}(\mathbf{c})\mathbf{x} - 2\left(\mathbf{c} \circ \mathbf{t}\right)^{\mathrm{T}} \mathbf{x} + \left(\mathbf{c} \circ \mathbf{t}\right)^{\mathrm{T}} \mathbf{t} && (4) \end{aligned}$$

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

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

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

$$\hat{W} = S^{\mathrm{T}} D_{\mathbf{m}}^{-1} D_{\mathbf{n}} \bar{B} D_{\mathbf{n}} D_{\mathbf{m}}^{-1} S \tag{6}$$

Algorithm 1 Bilateral-space bistochastization, reproduced from [2]
Input:
$W = S^{\mathrm{T}}\bar{B}S$ // A splat-blur-splice decomposition of W
Output:
$D_{\mathbf{n}}, D_{\mathbf{m}}$ // Matrices to bistochastize W

$\mathbf{m} \leftarrow S\mathbf{1}$
$\mathbf{n} \leftarrow \mathbf{1}$
while not converged **do**
  $\mathbf{n} \leftarrow \sqrt{(\mathbf{n} \circ \mathbf{m})/(\bar{B}\mathbf{n})}$
end while
$D_{\mathbf{n}} \leftarrow \mathrm{diag}(\mathbf{n})$
$D_{\mathbf{m}} \leftarrow \mathrm{diag}(\mathbf{m})$

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

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

$$SS^{\mathrm{T}} = D_{\mathbf{m}} \tag{7}$$

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 $\mathbf{y}$:

$$\mathbf{x} = S^{\mathrm{T}}\mathbf{y} \tag{8}$$

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:

$$\begin{aligned}
\mathbf{x}^{\mathrm{T}}\left(I - \hat{W}\right)\mathbf{x} =& \mathbf{x}^{\mathrm{T}}\left(I - S^{\mathrm{T}}D_{\mathbf{m}}^{-1}D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}D_{\mathbf{m}}^{-1}S\right)\mathbf{x} && \text{by Eq 6}\\
=& (S^{\mathrm{T}}\mathbf{y})^{\mathrm{T}}\left(I - S^{\mathrm{T}}D_{\mathbf{m}}^{-1}D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}D_{\mathbf{m}}^{-1}S\right)(S^{\mathrm{T}}\mathbf{y}) && \text{by Eq 8}\\
=& \mathbf{y}^{\mathrm{T}}\left(SIS^{\mathrm{T}} - SS^{\mathrm{T}}D_{\mathbf{m}}^{-1}D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}D_{\mathbf{m}}^{-1}SS^{\mathrm{T}}\right)\mathbf{y} && (9)\\
=& \mathbf{y}^{\mathrm{T}}\left(D_{\mathbf{m}} - D_{\mathbf{m}}D_{\mathbf{m}}^{-1}D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}D_{\mathbf{m}}^{-1}D_{\mathbf{m}}\right)\mathbf{y} && \text{by Eq 7}\\
=& \mathbf{y}^{\mathrm{T}}\left(D_{\mathbf{m}} - D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}\right)\mathbf{y} && (10)
\end{aligned}$$

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

$$\begin{aligned}
& \mathbf{x}^{\mathrm{T}}\mathrm{diag}(\mathbf{c})\mathbf{x} - 2(\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{x} + (\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} && (11)\\
=& (S^{\mathrm{T}}\mathbf{y})^{\mathrm{T}}\mathrm{diag}(\mathbf{c})(S^{\mathrm{T}}\mathbf{y}) - 2(\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}(S^{\mathrm{T}}\mathbf{y}) + (\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} && \text{by Eq 8}\\
=& \mathbf{y}^{\mathrm{T}}(S\mathrm{diag}(\mathbf{c})S^{\mathrm{T}})\mathbf{y} - 2(S(\mathbf{c} \circ \mathbf{t}))^{\mathrm{T}}\mathbf{y} + (\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} && (12)\\
=& \mathbf{y}^{\mathrm{T}}\mathrm{diag}(S\mathbf{c})\mathbf{y} - 2(S(\mathbf{c} \circ \mathbf{t}))^{\mathrm{T}}\mathbf{y} + (\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} && (13)
\end{aligned}$$

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

$$\underset{\mathbf{y}}{\text{minimize}} \quad \frac{1}{2}\mathbf{y}^{\mathrm{T}}\left(\lambda\left(D_{\mathbf{m}} - D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}\right) + \mathrm{diag}(S\mathbf{c})\right)\mathbf{y} - (S(\mathbf{c} \circ \mathbf{t}))^{\mathrm{T}}\mathbf{y} + \frac{1}{2}(\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t} \tag{14}$$

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

$$\underset{\mathbf{y}}{\text{minimize}} \quad \frac{1}{2}\mathbf{y}^{\mathrm{T}} A\mathbf{y} - \mathbf{b}^{\mathrm{T}}\mathbf{y} + c \tag{15}$$

$$A = \lambda(D_{\mathbf{m}} - D_{\mathbf{n}}\bar{B}D_{\mathbf{n}}) + \mathrm{diag}(S\mathbf{c}) \qquad \mathbf{b} = S(\mathbf{c} \circ \mathbf{t}) \qquad c = \frac{1}{2}(\mathbf{c} \circ \mathbf{t})^{\mathrm{T}}\mathbf{t}$$

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

$$A\mathbf{y} = \mathbf{b} \tag{16}$$

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 $A(\mathbf{x}) \approx \mathbf{b}$

$i \leftarrow 0$
$\mathbf{r} \leftarrow \mathbf{b} - A(\mathbf{x})$
$\mathbf{d} \leftarrow M^{-1}(\mathbf{r})$
$\lambda_{new} \leftarrow \mathbf{r}^{\mathrm{T}}\mathbf{d}$
while $i < n$ **do**
    $\mathbf{q} \leftarrow A(\mathbf{d})$
    $\alpha \leftarrow \frac{\lambda_{new}}{\mathbf{d}^{\mathrm{T}}\mathbf{q}}$
    $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{d}$
    $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{q}$
    $\mathbf{s} \leftarrow M^{-1}(\mathbf{r})$
    $\lambda_{old} \leftarrow \lambda_{new}$
    $\lambda_{new} \leftarrow \mathbf{r}^{\mathrm{T}}\mathbf{s}$
    $\beta \leftarrow \frac{\lambda_{new}}{\lambda_{old}}$
    $\mathbf{d} \leftarrow \mathbf{s} + \beta\mathbf{d}$
    $i \leftarrow i + 1$
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:

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

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

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

We compute $P(\mathbf{y})$ and $P^{\mathrm{T}}(\mathbf{z})$ 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:

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

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:

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

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$(\cdot)$ // The bilateral solver
$\mathbf{t}$ // The input target vector
$\mathbf{c}_{init}$ // 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

$\mathbf{c} \leftarrow \mathbf{c}_{init}$
while $i < n$ **do**
  $\hat{\mathbf{x}} \leftarrow \text{solve}(\mathbf{t}, \mathbf{c})$
  $\mathbf{e} \leftarrow (\hat{\mathbf{x}} - \mathbf{t})$
  $\mathbf{c} \leftarrow \frac{2\sigma_{gm}^2}{\left(\sigma_{gm}^2 + (\mathbf{e} \circ \mathbf{e})\right)^2}$
  $i \leftarrow i + 1$
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:

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

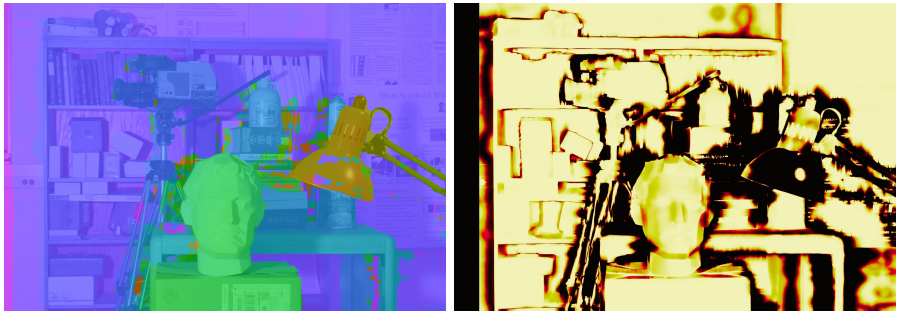
Where E is the expectation operator. Any (normalized) linear image filtering operation with non-negative weights can be thought of as computing some local expectation of the input image for every pixel. When coupled with an edge-aware image filtering operation such as the domain transform (DT), we can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

$$V = \mathrm{DT}(Z^2) - (\mathrm{DT}(Z))^2 \tag{22}$$

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

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

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.



(a) Input depth map Z from MC-CNN[31] (b) Estimated input confidence $\mathbf{c}_{init}$

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:

$$AY = B \tag{24}$$

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.

$$BP = QR \tag{25}$$

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:

$$B = Q'R' \tag{26}$$

Where we construct R' by taking RP^{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':

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

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:

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

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

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

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:

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

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 code[1]. 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 code[2] 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 code[3]. Because this technique was designed for a similar use case to its use here, we used the same parameter settings as were use in the paper: $r = \max(width, height)$, $\epsilon = 0.01^2$, 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`

[3] `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:

$$t_i = (l_i + u_i)/2 \qquad c_i = \exp(l_i - u_i) \tag{31}$$

This simple reparametrization combined with our bilateral solver produces an effective stereo technique for this "defocus" task, while being roughly 3.5× 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 600-3000× 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×) 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 *et al.*[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 *et al.* [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 *et al.* [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 *et al.*[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 *et al.*[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 *et al.*[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 *et al.* [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 *et al.* [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 *et al.* 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 *et al.* [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 *et al.*[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 *et al.*[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 *et al.*[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 code[4]. 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 code[5]. 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 code[6]. 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 code[7]. 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, "NeighborhoodSize" $= 3 \times 2^f$, (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 code[8]. 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 code[9] 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× speedup may be possible.

R This is the technique presented in Yang [29], which we ran ourselves using the publicly available code[10]. 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 code[11], 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 lattice[12].

V These errors and runtimes were taken from the website[13] 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 supplement[14] 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/imagestack/`

[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 website[15].

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 website[16], 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

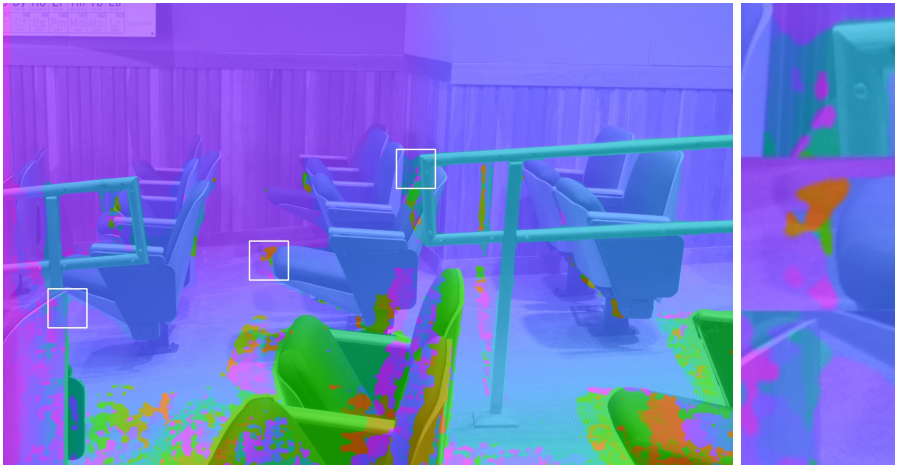
See Figure 17 for additional examples of our bilateral solver and competing techniques applied to the semantic segmentation task. The bilateral solver produces more edge-aware results than CRF-based techniques on well-isolated objects.

[15] `https://sites.google.com/site/datadrivendepthcvpr2015/`
[16] `www.cs.huji.ac.il/~yweiss/Colorization/`

(a) MC-CNN[31]
bad 1% = 10.1 MAE = 1.69 RMSE = 9.60

(b) MC-CNN[31] + RBS
bad 1% = 10.7 MAE = 1.63 RMSE = 8.72

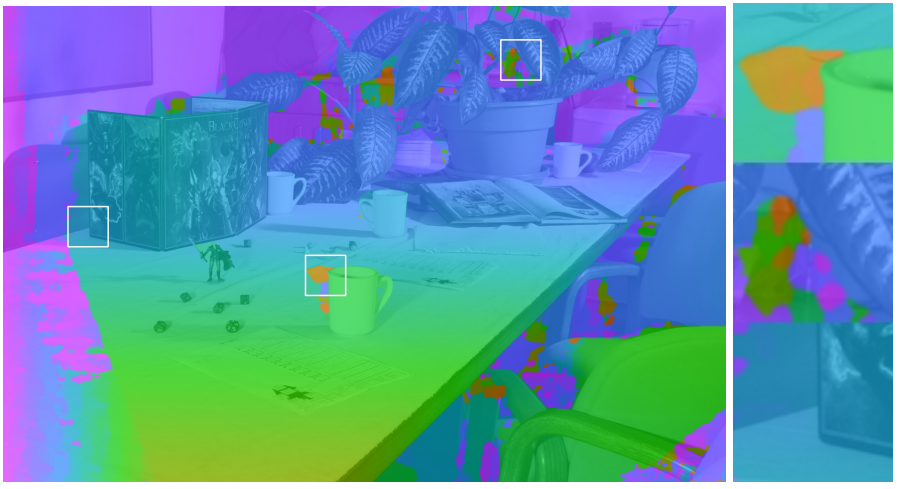
Fig. 2: Results on the test set of the Middlebury Stereo Dataset V3 [26] where our robust bilateral solver is used to improve the depth map predicted by the state-of-the-art MC-CNN technique[31]. On the top we have the depth map produced by [31] (with zoomed in regions) which is used as the target in 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.



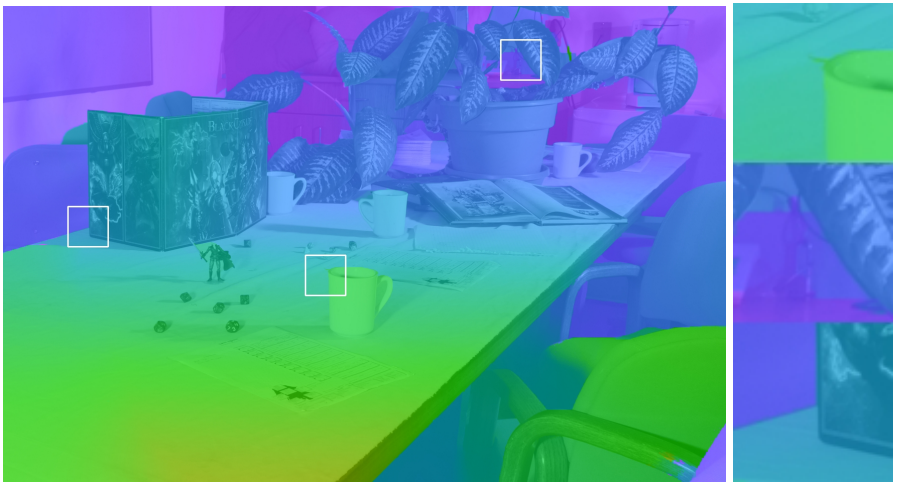
(a) MC-CNN[31]
bad 1% = 23.4 MAE = 17.1 RMSE = 67.4

(b) MC-CNN[31] + RBS
bad 1% = 24.4 MAE = 4.30 RMSE = 19.9

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

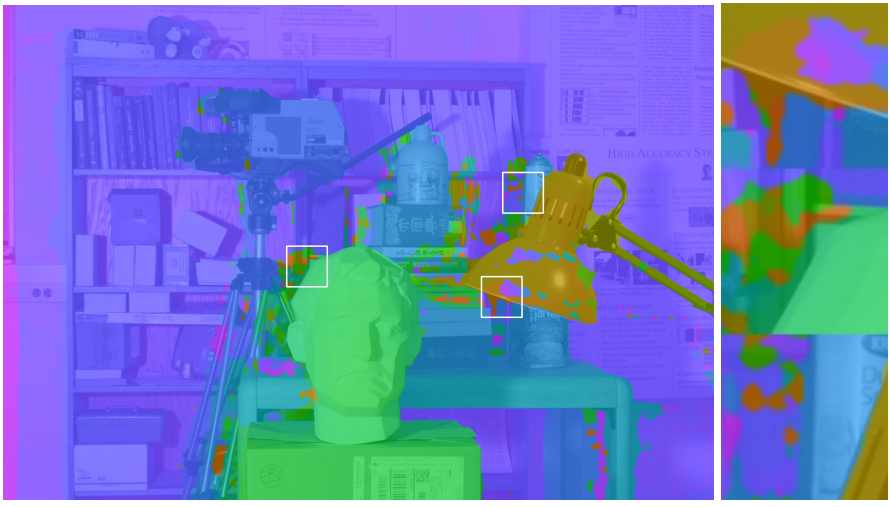


(a) MC-CNN[31]
bad 1% = 25.0 MAE = 2.47 RMSE = 23.2

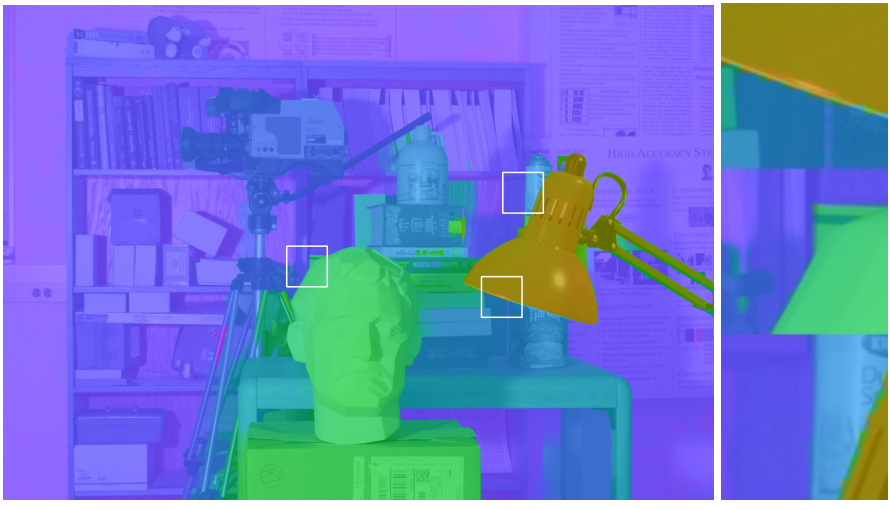


(b) MC-CNN[31] + RBS
bad 1% = 27.1 MAE = 2.81 RMSE = 24.2

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

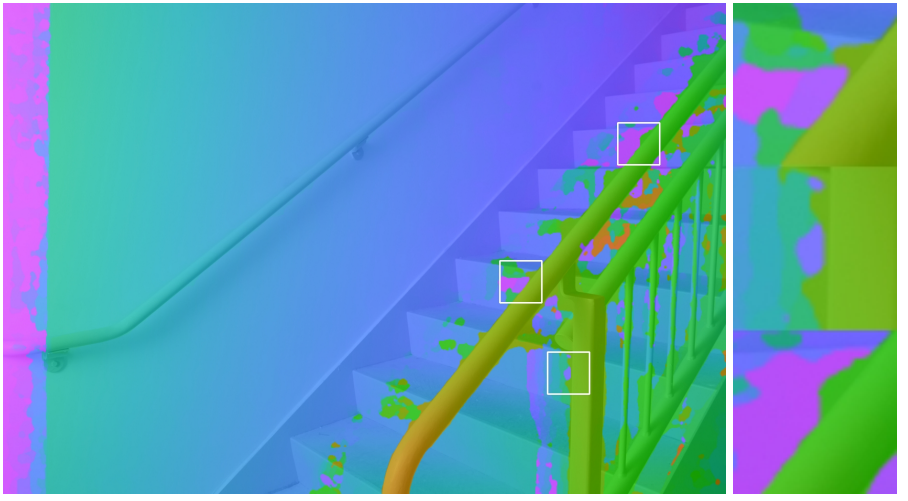


(a) MC-CNN[31]
bad 1% = 22.5 MAE = 6.00 RMSE = 38.8

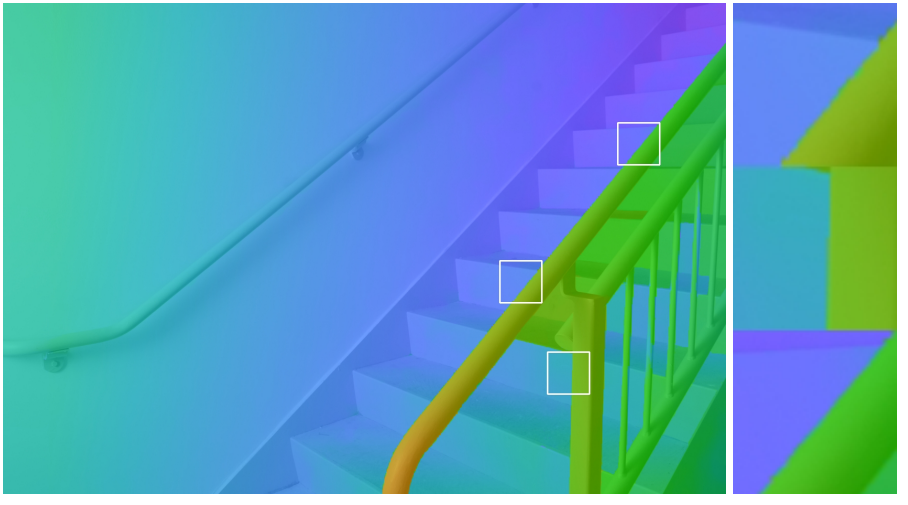


(b) MC-CNN[31] + RBS
bad 1% = 22.8 MAE = 3.02 RMSE = 17.9

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

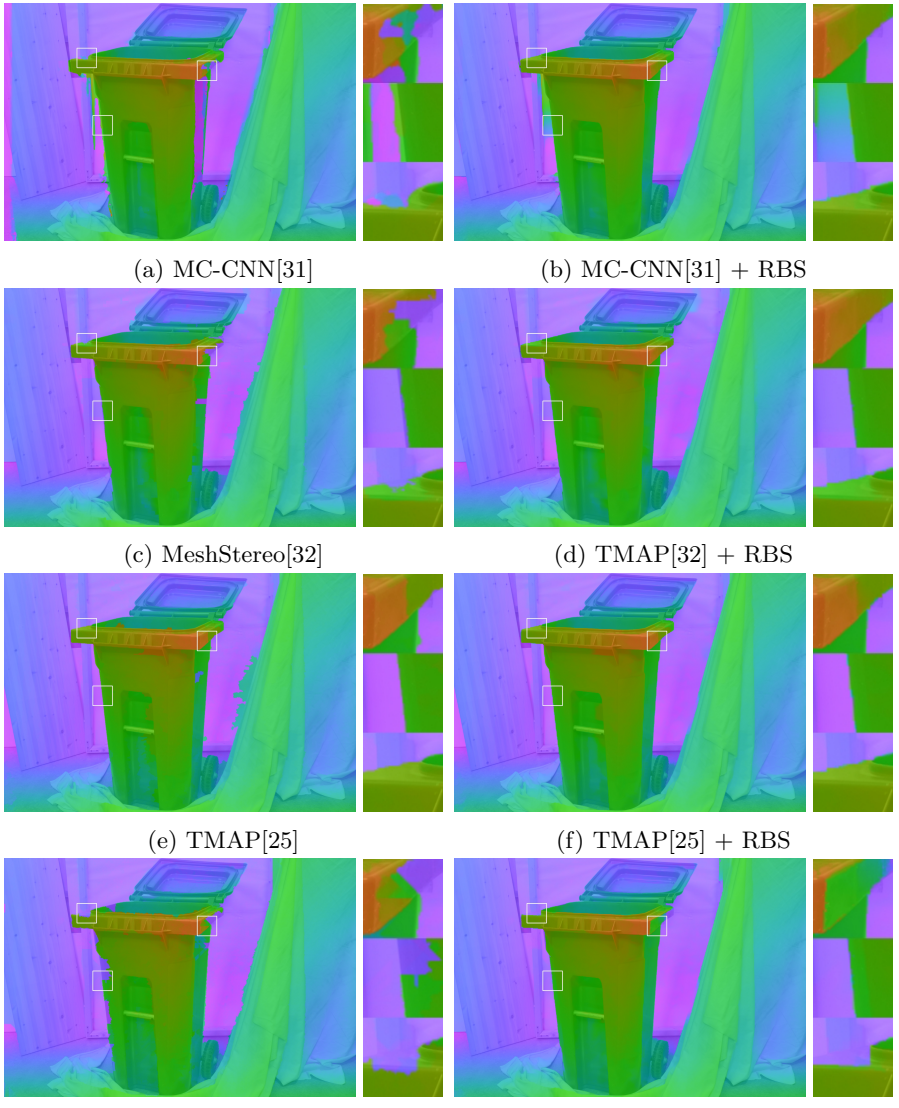


(a) MC-CNN[31]
bad 1% = 22.0 MAE = 5.66 RMSE = 28.6



(b) MC-CNN[31] + RBS
bad 1% = 21.6 MAE = 3.19 RMSE = 17.9

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



(a) MC-CNN[31] (b) MC-CNN[31] + RBS

(c) MeshStereo[32] (d) TMAP[32] + RBS

(e) TMAP[25] (f) TMAP[25] + RBS

(g) SGM[13] (h) SGM[13] + RBS

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.

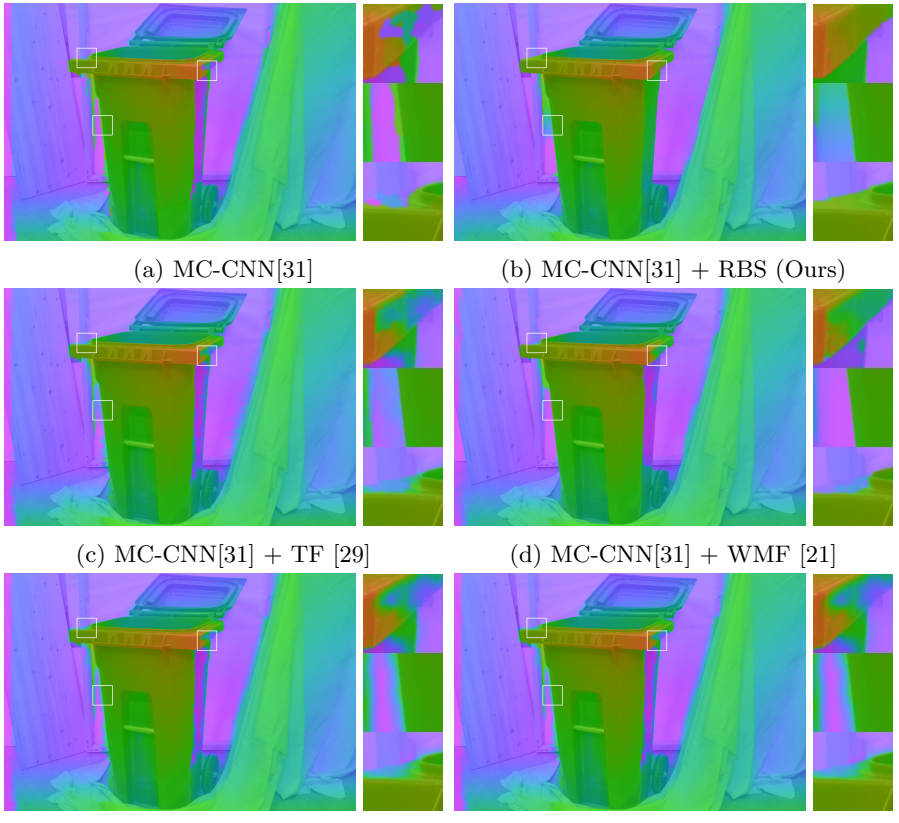
(a) MC-CNN[31] (b) MC-CNN[31] + RBS

(c) MeshStereo[32] (d) TMAP[32] + RBS

(e) TMAP[25] (f) TMAP[25] + RBS

(g) SGM[13] (h) SGM[13] + RBS

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

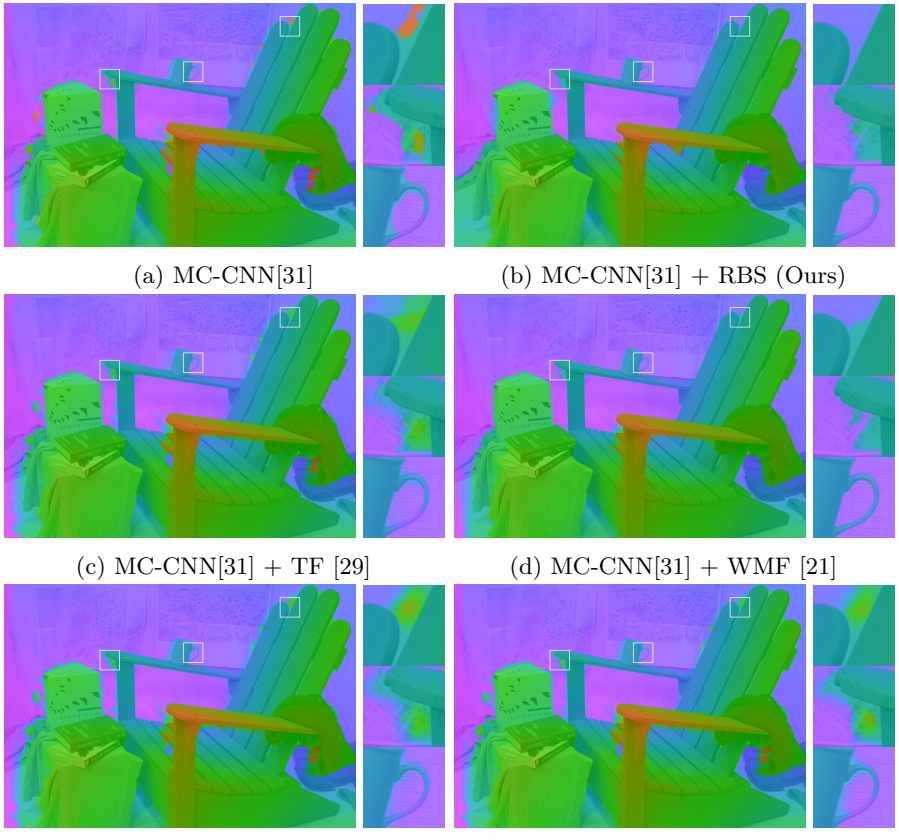


(a) MC-CNN[31] (b) MC-CNN[31] + RBS (Ours)

(c) MC-CNN[31] + TF [29] (d) MC-CNN[31] + WMF [21]

(e) MC-CNN[31] + FGF [11] (f) MC-CNN[31] + DT [9]

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.

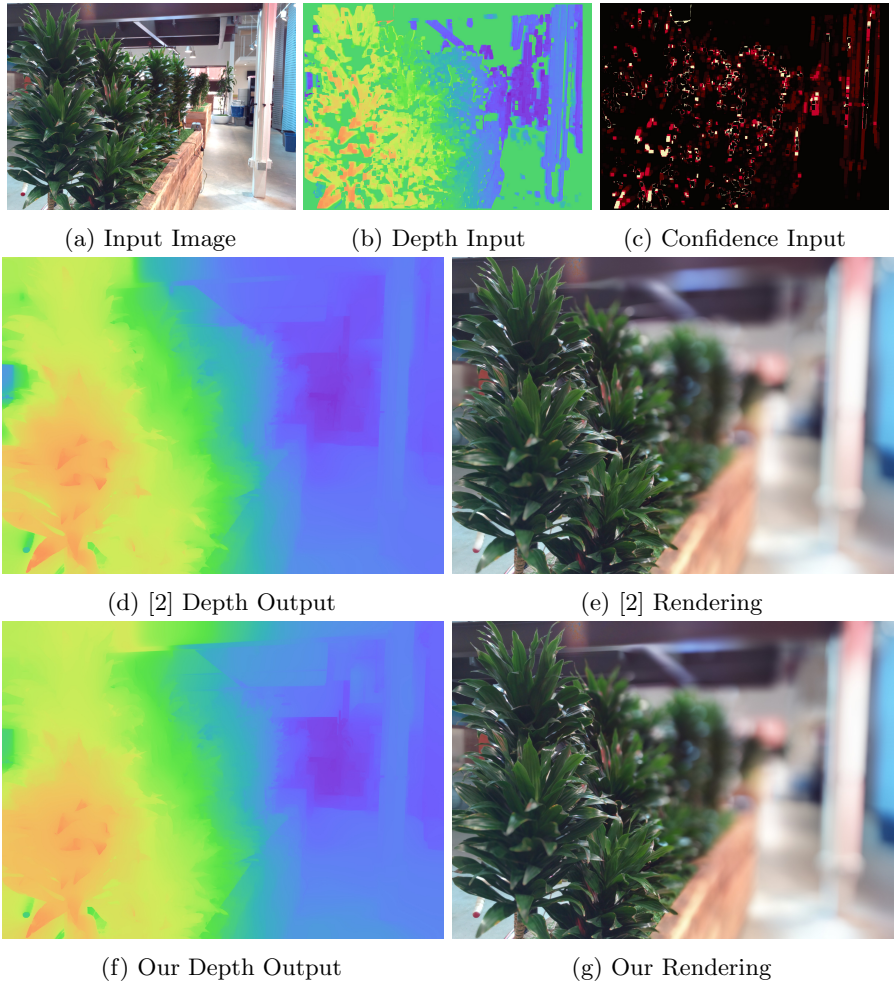


(a) MC-CNN[31] (b) MC-CNN[31] + RBS (Ours)

(c) MC-CNN[31] + TF [29] (d) MC-CNN[31] + WMF [21]

(e) MC-CNN[31] + FGF [11] (f) MC-CNN[31] + DT [9]

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

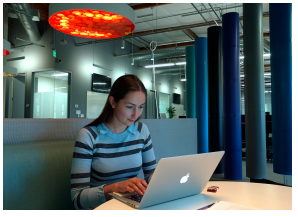


(a) Input Image (b) Depth Input (c) Confidence Input

(d) [2] Depth Output (e) [2] Rendering

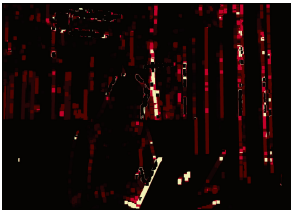
(f) Our Depth Output (g) Our Rendering

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× faster.

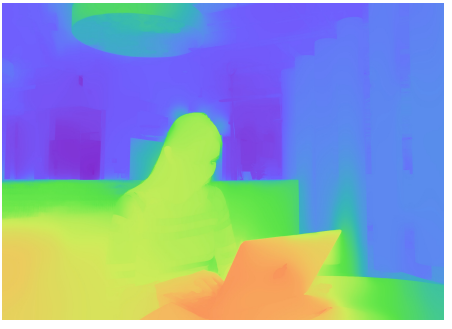


(a) Input Image

(b) Depth Input

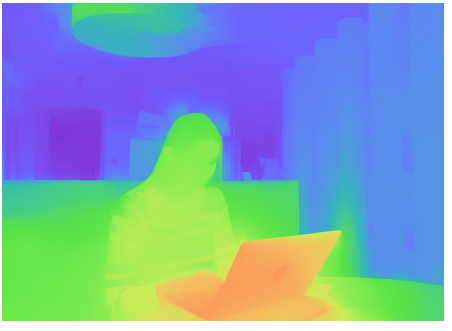


(c) Confidence Input

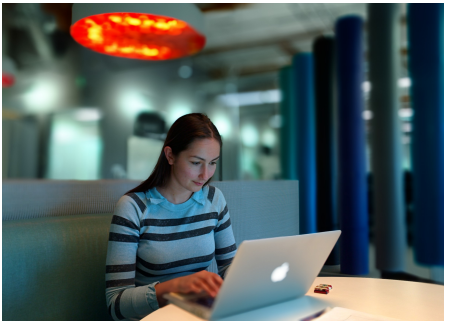


(d) [2] Depth Output

(e) [2] Rendering

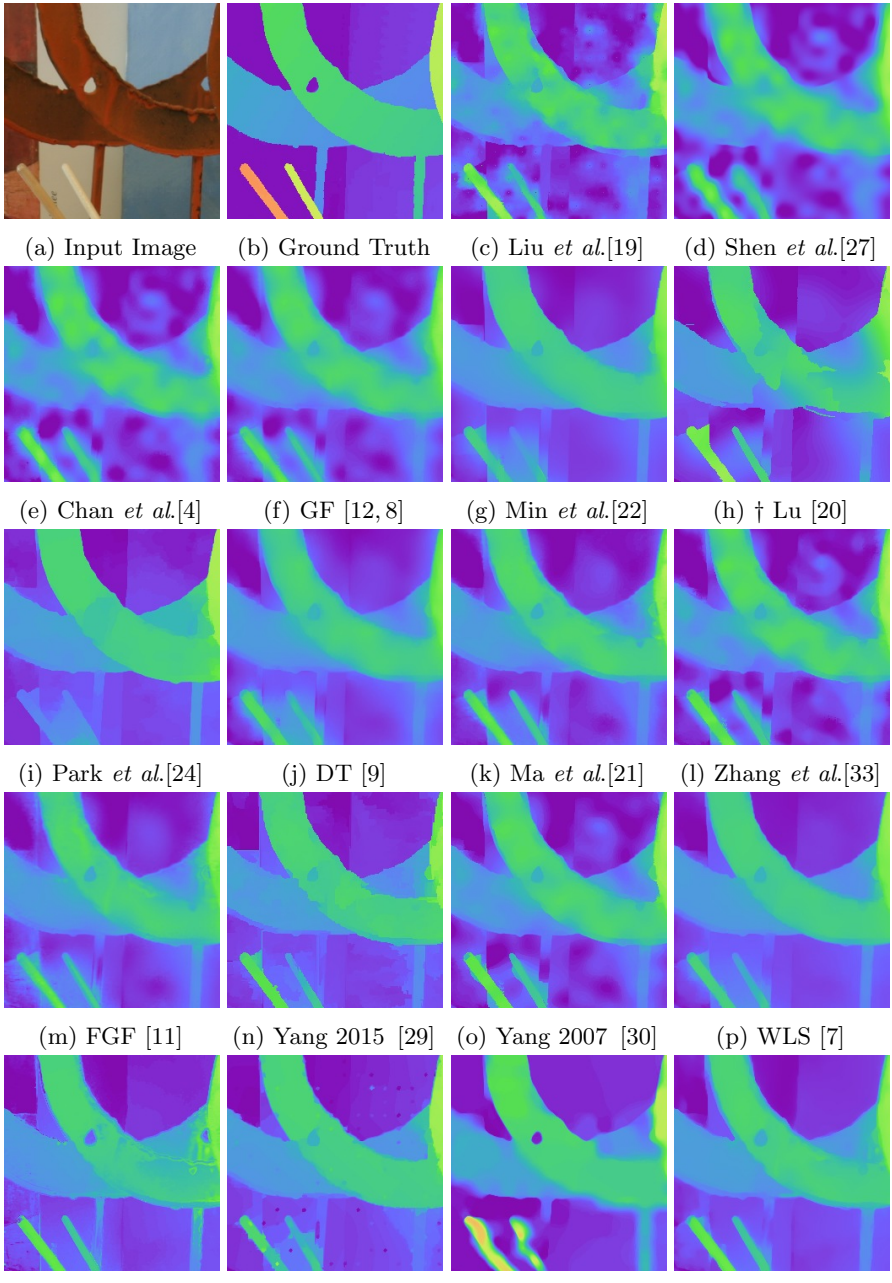


(f) Our Depth Output



(g) Our Rendering

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



(a) Input Image (b) Ground Truth (c) Liu *et al.*[19] (d) Shen *et al.*[27]

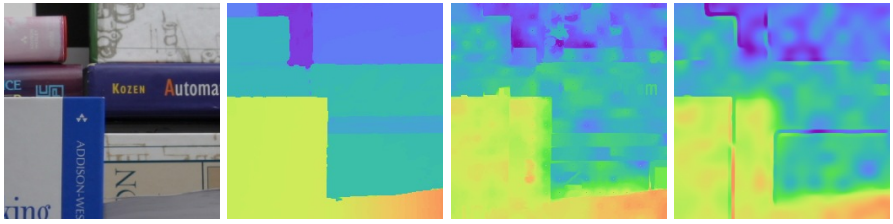
(e) Chan *et al.*[4] (f) GF [12, 8] (g) Min *et al.*[22] (h) † Lu [20]

(i) Park *et al.*[24] (j) DT [9] (k) Ma *et al.*[21] (l) Zhang *et al.*[33]

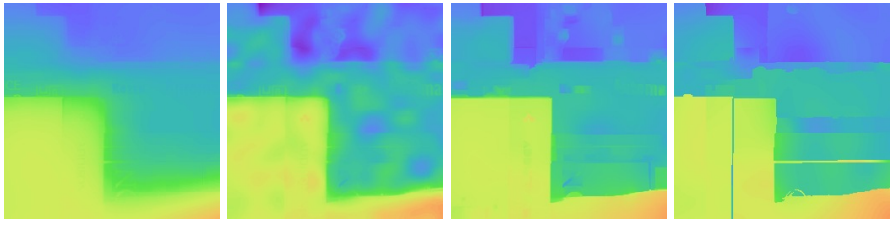
(m) FGF [11] (n) Yang 2015 [29] (o) Yang 2007 [30] (p) WLS [7]

(q) JB [1, 15] (r) Ferstl *et al.*[8] (s) † Li*et al.*[18] (t) BS (Ours)

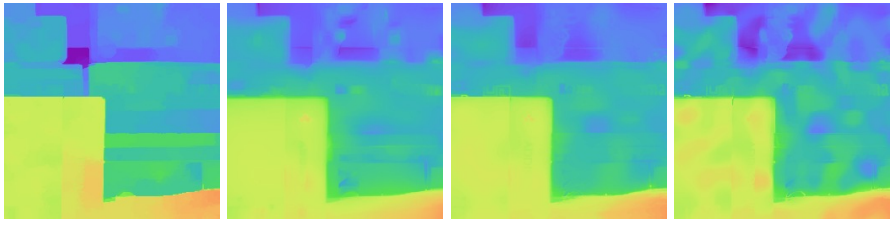
Fig. 13: Results for the depth superresolution task. Algorithms are sorted according to their average error on this benchmark, from upper left to lower right. Algorithms which use external training data are indicated with a dagger.



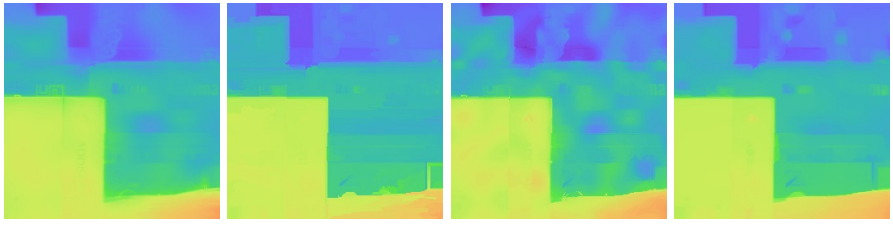
(a) Input Image (b) Ground Truth (c) Liu *et al.*[19] (d) Shen *et al.*[27]



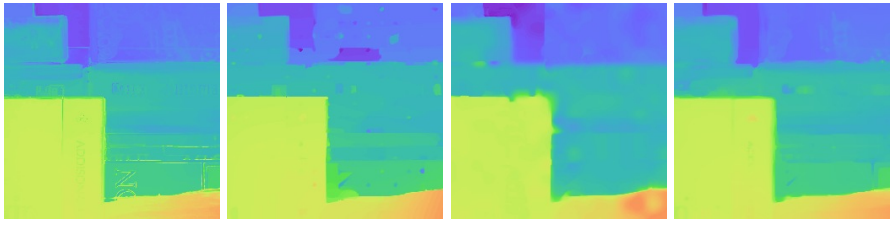
(e) Chan *et al.*[4] (f) GF [12, 8] (g) Min *et al.*[22] (h) † Lu [20]



(i) Park *et al.*[24] (j) DT [9] (k) Ma *et al.*[21] (l) Zhang *et al.*[33]

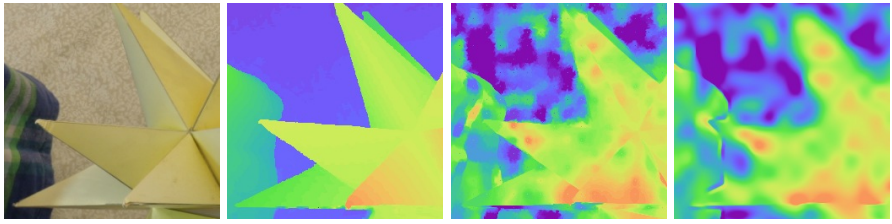


(m) FGF [11] (n) Yang 2015 [29] (o) Yang 2007 [30] (p) WLS [7]

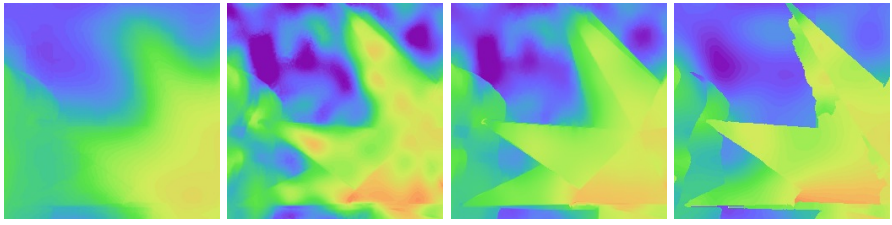


(q) JB [1, 15] (r) Ferstl *et al.*[8] (s) † Li*et al.*[18] (t) BS (Ours)

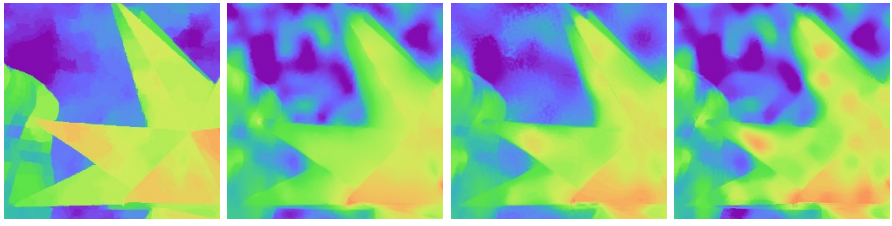
Fig. 14: More results for the depth superresolution task.



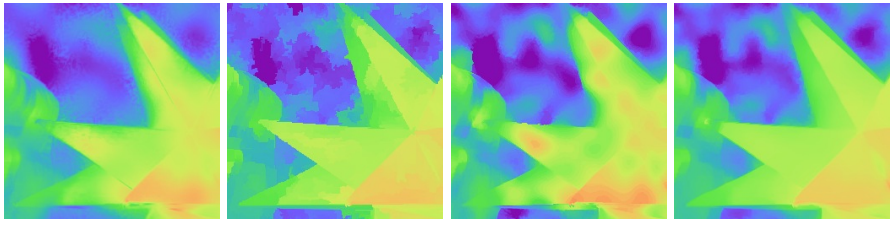
(a) Input Image (b) Ground Truth (c) Liu *et al.*[19] (d) Shen *et al.*[27]



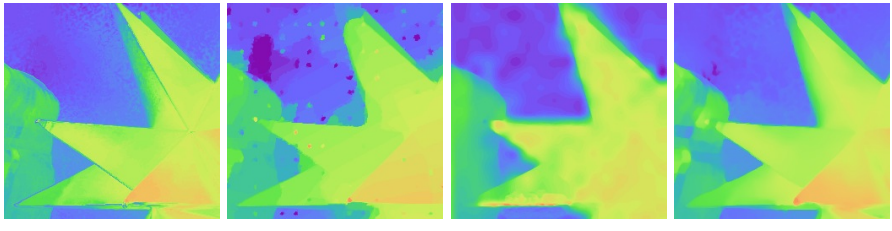
(e) Chan *et al.*[4] (f) GF [12, 8] (g) Min *et al.*[22] (h) † Lu [20]



(i) Park *et al.*[24] (j) DT [9] (k) Ma *et al.*[21] (l) Zhang *et al.*[33]



(m) FGF [11] (n) Yang 2015 [29] (o) Yang 2007 [30] (p) WLS [7]



(q) JB [1, 15] (r) Ferstl *et al.*[8] (s) † Li*et al.*[18] (t) BS (Ours)

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

(a) Input (b) Levin *et al.*[17] (c) Our results

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× faster.

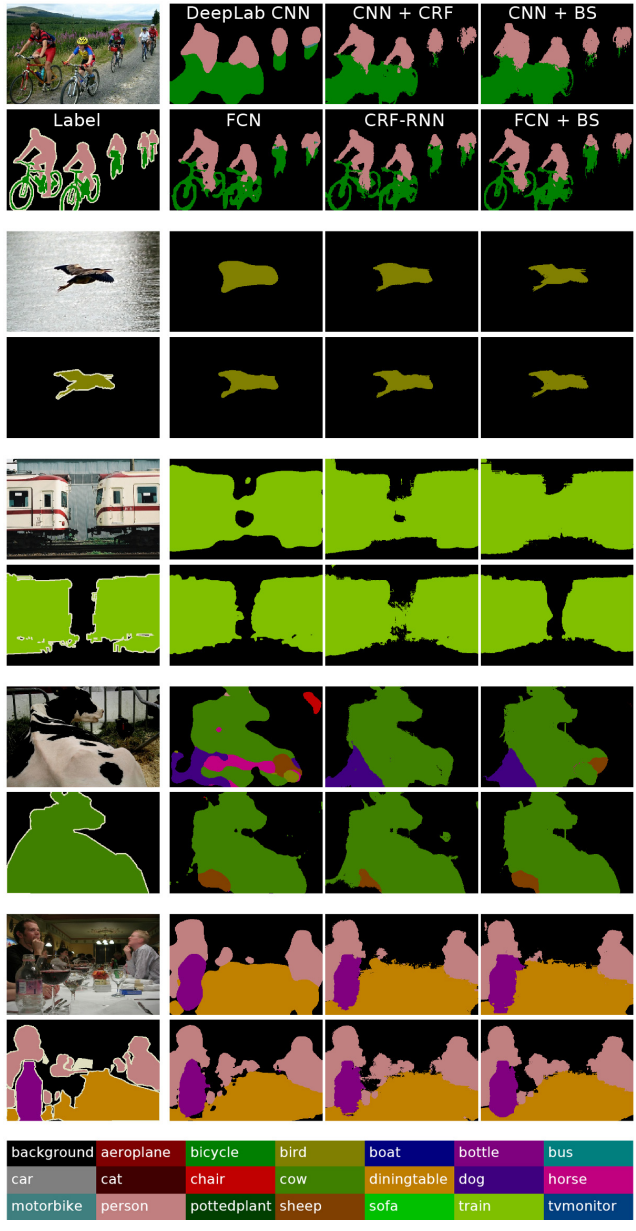


Fig. 17: Additional semantic segmentation results on Pascal VOC12 validation images. FCN refers to the fully convolutional network component of the end-to-end trained CRF-RNN model. While the CRF-augmented DeepLab model (top rows) and CRF-RNN model (bottom rows) perform best overall, the bilateral solver 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., Kowalczuk, 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)