# Fast Bilateral-Space Stereo for Synthetic Defocus
# Supplemental Material

Jonathan T. Barron barron@google.com
Andrew Adams abadams@google.com
YiChang Shih yichang@mit.edu
Carlos Hernández chernand@google.com

## 1. Bilateral Representations

In the main paper, we described our bilateral representations as factorizations of a Gaussian affinity matrix $A$:

$$A \approx S^{\mathrm{T}} \bar{B} S \tag{1}$$

We will now dig deeper into the details of this matrix factorization, and discuss the two specific bilateral representations we use: the simplified bilateral grid, and the permutohedral lattice [1].

Filtering with both the permutohedral lattice and the simplified bilateral grid works by "splatting" a value at each pixel onto a small number of vertices, performing a separable blur in the space of vertices, and "slicing" out values at each pixel to get a filtered set of values. The difference between the two representations is in the arrangement of the vertices (the permutohedral lattice is tetrahedral, while the simplified bilateral grid is rectangular), and the nature of the splat interpolation (the lattice uses barycentric interpolation, and the simplified bilateral grid uses nearest-neighbor assignment). Intuitively, the permutohedral lattice approximates the Gaussian in the affinity function as the convolution of a tent filter (barycentric interpolation), a $[1, 2, 1]$ blur kernel, and another tent filter, which is a good binomial approximation of a Gaussian function. The simplified bilateral grid approximates a Gaussian using a boxcar or "rect" filter (nearest-neighbor interpolation) and a narrow [1,2,1] blur kernel, which is a significantly less accurate but more efficient representation. In the factorization produced by the permutohedral lattice, assuming that our Gaussian affinity is in a $D$ dimensional space, the splat matrix $S$ has $D + 1$ non-zero elements per row, we have $D+1$ blur matrices, and we approximate $\bar{B}$ as an outer product of the blur matrices. In the simplified bilateral grid, we have 1 non-zero element per row of our splat matrix, we have $D$ blur matrices, and we approximate $\bar{B}$ as the sum of blur matrices.

$$\bar{B}_{lattice} = B_1 B_2 \dots B_{D+1} \tag{2}$$
$$\bar{B}_{grid} = B1 + B2 + \dots + B_D \tag{3}$$

Regardless of the technique or the dimensionality of the problem, each $B_d$ matrix has no more than three non-zero elements per row. By approximating the blur matrix of the grid as a sum, the effective standard deviation of the Gaussian affinity being approximated is more narrow. This also means that blurring can be made slightly more efficient by processing all blurs in parallel rather than in sequence.

Note that in most literature, the bilateral grid [5] is implemented with bilinear interpolation and is "dense" (i.e., vertices are created even if they are not filled), and the blur matrix is approximated as an outer product of each dimension's blur matrix. In contrast, our simplified grid is "sparse" — vertices with no pixels assigned to them are never created. This allows the grid to scale to higher dimensions than the dense bilateral grid, which is difficult to extend beyond 3 dimensions.

Throughout the paper we repeatedly assume that $A$ is symmetric, which requires that $\bar{B}$ be symmetric. By construction we know that each $B_d$ is symmetric, but because of missing vertices in the grid or lattice, the outer product of blur matrices may not be symmetric. This means that for our math to be completely correct, we must symmetrize the blur matrix used in for the permutohedral lattice:

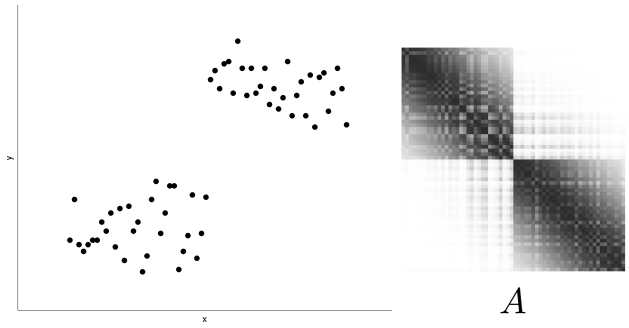
$$\bar{B}_{lattice\_sym} = B_1 B_2 \dots B_{D+1} + B_{D+1} B_D \dots B_1 \tag{4}$$

That being said, in practice it is possible to use the nonsymmetric $\bar{B}_{lattice}$ as though it is symmetric in our optimization without a noticeable loss in accuracy. Because $\bar{B}_{grid}$ is the sum of symmetric $B_d$ rather than the product matrices, it is guaranteed to be symmetric.

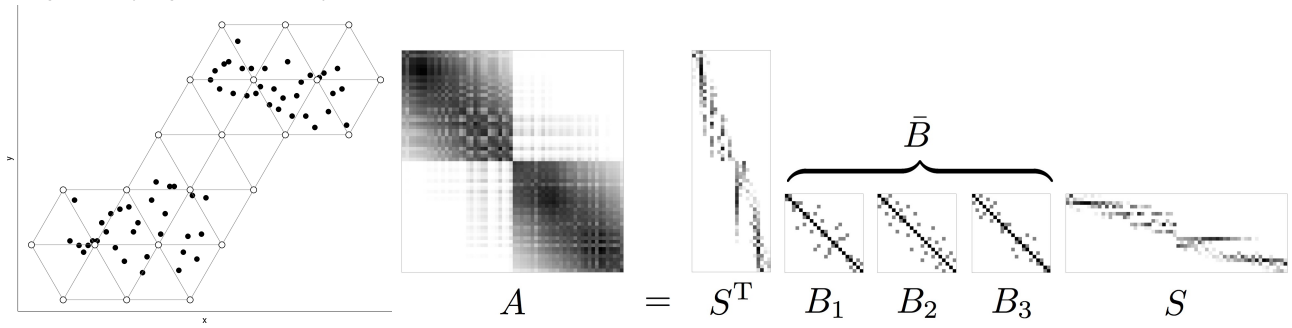
To clarify, we never actually construct $\bar{B}$, but we instead we evaluate matrix-vector products with $\bar{B}$ by repeatedly evaluating each constituent blur matrix:

$$\bar{B}_{lattice} x = (B_1(B_2(\dots(B_D x)\dots))) \tag{5}$$
$$\bar{B}_{grid} x = B_1 x + B_2 x + \dots + B_D x \tag{6}$$

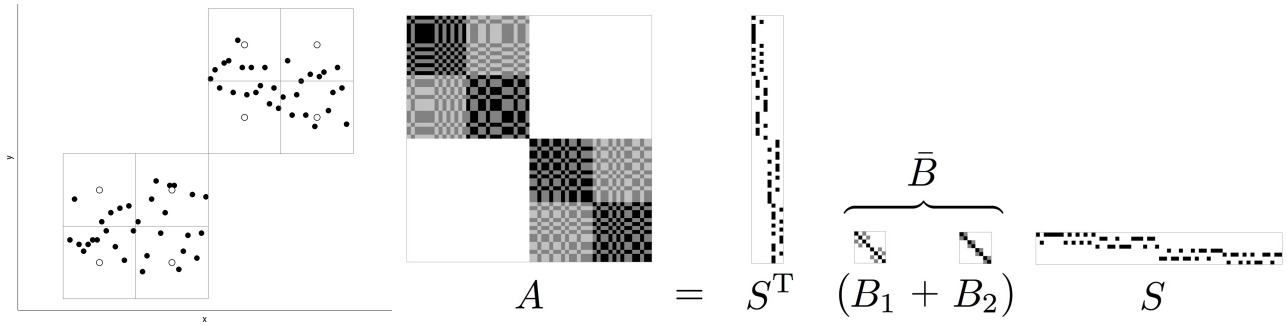
Note that, because our blur operation is a sum of blurs rather than a product, each blur can be efficiently computed in parallel. For a better understanding of these bilateral representations as matrix factorizations, see Figure 1.




(a) A toy 2D image of a step-edge on the left, and the true per-pixel affinity matrix that corresponds to that "image" on the right (dark = high affinity, light = low affinity)




(b) The toy 2D image of a step edge with a permutohedral lattice overlaid on top. The affinity matrix described by the lattice is shown in the middle, which we see closely resembles the true affinity matrix in Fig. 1a. We also show the matrix decomposition of $A$ according to the permutohedral lattice, which is a skinny and sparse "splat" matrix with 3 non-zero entries per row (where each row sums to 1), a sequence of 3 small and sparse "blur" matrices, and a skinny and sparse "slice" matrix which is the transpose of the "splat" matrix.




(c) The toy 2D image of a step edge with a simplified bilateral grid overlaid on top. In the middle we see the affinity matrix described by the simplified bilateral grid, which only coarsely resembles the true affinity matrix in Fig. 1a, due to the approximate nature of our "hard" nearest-neighbor assignment of pixels to grid vertices. We also show the matrix factorization of $A$ according to the simplified bilateral grid, which is significantly more compact than the permutohedral lattice representation: the splice matrix is binary and has only 1 non-zero entry per pixel (equivalent to a hard assignment of pixels to vertices), the number of vertices is reduced, and there are only 2 blur matrices.

Figure 1: The bilateral representations that we are concerned with (the permutohedral lattice and a simplified variant of the bilateral grid) can be viewed as special matrix decompositions of a Gaussian affinity matrix, as shown here. Here we consider a simple one-dimensional grayscale image with a step-edge. In Fig. 1a we see the true affinity matrix that corresponds to that grayscale image for some spatial and range standard deviation. In Fig. 1b and Fig. 1c we see that same step-edge overlaid with the grid or lattice that the bilateral representation tessellates our Euclidean space with, and the corresponding matrix factorization used by that bilateral representation.

## 2. Bilateral-Space Embedding

Let us derive the equivalence of the two versions of the smoothness terms in our per-pixel optimization problem. We start with the following smoothness cost:

$$\frac{1}{2}\sum_i\sum_j \hat{A}_{i,j}(p_i - p_j)^2 \tag{7}$$

This is just the sum of squared differences between each pixel's disparity, weighted by some weight $\hat{A}_{i,j}$. This can be expanded:

$$= \frac{1}{2}\sum_i\sum_j \hat{A}_{i,j}(p_i^2 - 2p_i p_+ p_j^2) \tag{8}$$

$$= -\sum_i\sum_j \hat{A}_{i,j}p_i p_j + \frac{1}{2}\sum_i\sum_j \hat{A}_{i,j}p_i^2 + \frac{1}{2}\sum_i\sum_j \hat{A}_{i,j}p_j^2 \tag{9}$$

Now, let us assume that the matrix $\hat{A}$ is symmetric and bistochastic (that its rows and columns all sum to 1). With this assumption, our smoothness term can be further simplified:

$$= -\sum_i\sum_j \hat{A}_{i,j}p_i p_j + \frac{1}{2}\sum_i p_i^2 + \frac{1}{2}\sum_j p_j^2 \tag{10}$$

$$= ||\mathbf{p}|| - \mathbf{p}^{\mathrm{T}}\hat{A}\mathbf{p} \tag{11}$$

$$= \mathbf{p}^{\mathrm{T}}(I - \hat{A})\mathbf{p} \tag{12}$$

Let's work through how A can be made bistochastic. Given that A is symmetric, we can find a diagonal matrix $N$ that bistochasticizes A with the simple iterative procedure of [8]:

$\mathbf{n} \leftarrow \vec{1}$
**while** not converged **do**
  $\mathbf{n} \leftarrow \sqrt{\mathbf{n}/(A\mathbf{n})}$
**end while**
$N \leftarrow \mathrm{diag}(\mathbf{n})$

Where division is element-wise. Several iterations are required for $\hat{A}$ to be bistochastic (usually 10-20 iterations for errors to be less than $10^{-6}$). With $N$, we can compute our bistochastic $\hat{A}$:

$$\hat{A} = NAN \tag{13}$$

This bistochasticization procedure is equivalent to the following:

$\hat{A} \leftarrow A$
**while** not converged **do**
  $\hat{A} \leftarrow D^{-1/2}\hat{A}D^{-1/2}$
**end while**

Of course, this algorithm would not work in practice as we never explicitly compute $A$.

A bistochasticized $A$ is useful for many reasons. First, some bilateral representations introduce an arbitrary scale factor, meaning that they actually approximate some affinity matrix which is only proportional to $A$. Bistochasticization normalizes out this scale factor. Second, a common result in image filtering and spectral clustering is that a bistochastic affinity matrix produces better results than a singularly-stochastic matrix [9]. Third, were we to simply row-normalize $A$ (i.e., treat $A$ as a simple normalized image filtering operation) then our resulting row-normalized matrix would not be symmetric. This would make optimization less efficient, as we repeatedly take advantage of the assumption that $\hat{A}^{\mathrm{T}} = \hat{A}$ throughout the design of our algorithm. This requirement for symmetry is also part of why we use our specific symmetry-preserving matrix normalization technique, instead of other techniques like Sinkhorn normalization.

Given the normalization matrix $N$ returned by our bistochasticization procedure, our variable substitution is straightforward:

$$\mathbf{p}^{\mathrm{T}}\left(I - \hat{A}\right)\mathbf{p} \tag{14}$$

$$= (S^{\mathrm{T}}\mathbf{v})^{\mathrm{T}}(I - NAN)(S^{\mathrm{T}}\mathbf{v}) \tag{15}$$

$$= \mathbf{v}^{\mathrm{T}}S(I - NAN)S^{\mathrm{T}}\mathbf{v} \tag{16}$$

$$= \mathbf{v}^{\mathrm{T}}(SS^{\mathrm{T}} - SNANS^{\mathrm{T}})\mathbf{v} \tag{17}$$

$$= \mathbf{v}^{\mathrm{T}}(SS^{\mathrm{T}} - SNS^{\mathrm{T}}\bar{B}SNS^{\mathrm{T}})\mathbf{v} \tag{18}$$

We can simplify things by introducing some new variables:

$$\mathbf{p}^{\mathrm{T}}\left(I - \hat{A}\right)\mathbf{p} = \mathbf{v}^{\mathrm{T}}(C'_s - C'_n\bar{B}C'_n)\mathbf{v} \tag{19}$$

$$C'_n = SNS^{\mathrm{T}}, \quad C'_s = SS^{\mathrm{T}} \tag{20}$$

Though our reformulated problem may seem fine at first glance, our intermediate $C'$ matrices hide a serious problem. Though these matrices are small (with as many rows and columns as there are vertices), as outer products of the splat matrix these matrices are not very sparse, and computing them requires a surprisingly expensive sparse-sparse matrix multiplication ($> 1$ second for a 5-megapixel image. We would like to replace these semi-sparse matrices with diagonal matrices, and we would like to estimate these diagonal matrices cheaply. This issue ties into bistochasticization, as these $C'$ matrices require the per-pixel bistochasticization matrix $N$, which itself required repeated expensive filtering in pixel-space.

Let us remind ourselves how we got here: we took our bilateral affinity $A$, bistochasticized it in pixel-space, and then constructed our $C'$ matrices by projecting those bistochasticization weights into vertex space. As a much faster alternative, we will take advantage of the structure of $A$ and bistochasticize $A$ in bilateral-space, while restricting ourselves to diagonal $C$ matrices. This will necessarily be an approximation, but we should expect this approximation

to be good because $A$ is well-approximated by our bilateral representations. An additional approximation is our requirement that the $C$ matrices be diagonal, which means that we will not model the "mixing" of the splat/slice matrix, and our affinity model will accordingly become slightly less "strong" (i.e., less inclined to smooth our disparity maps).

Let's investigate how we might bistochasticize our system in bilateral-space. Naively one might assume that we can simply bistochasticize $\bar{B}$, but that approach ignores the fact that each vertex has a different number of pixels "assigned" to it. Instead we must bistochasticize a version of $\bar{B}$ that is weighted by the number of pixels belonging to each vertex — the "mass" of each vertex. Formally, this mass is:

$$\mathbf{m} = S\mathbf{1} \tag{21}$$

What follows is a variant of the previous bistochasticization procedure which produces a "normalized" matrix $\bar{B}$ in which the rows and columns of that matrix sum to $\mathbf{m}$, while the previously described algorithm produced a normalized $A$ matrix whose rows and columns sum to $\mathbf{1}$:

$\mathbf{n} \leftarrow \vec{1}$
**while** not converged **do**
  $\mathbf{n} \leftarrow \sqrt{(\mathbf{n} \times \mathbf{m})/(\bar{B}\mathbf{n})}$
**end while**

Where $\times$ and $/$ are element-wise multiplication and division, respectively. Note that the $\mathbf{n}$ that we recover does not actually bistochasticize $\bar{B}$. Instead, it implicitly bistochasticizes $\hat{A}$ while staying entirely in bilateral-space. With the recovered $\mathbf{n}$, we can construct our new lightweight alternatives to the $C'$ matrices:

$$C'_n \approx C_n = \mathrm{diag}(\mathbf{n}), \quad C'_s \approx C_s = \mathrm{diag}(\mathbf{m}) \tag{22}$$

By construction, each $C$ matrix is diagonal and cheap to compute. As a sanity check for why this procedure makes sense, let us observe that:

$$(C'_n \bar{B} C'_n)\mathbf{1} \;=\; (C_n \bar{B} C_n)\mathbf{1} \;=\; \mathbf{m} \tag{23}$$

So it seems that $(C_n \bar{B} C_n)$ is a reasonable approximation to $(C'_n \bar{B} C'_n)$. In practice, we have noticed no significant difference in the quality of the output of our model whether we use $C$ or $C'$ during inference. For all results in our paper, we use these $C$ matrices instead of the originally defined $C'$ matrices.

## 3. Block-Matching Interval Cost Volume

As described in the paper, the "cost volume" in our optimization framework is implicitly defined as an upper and lower range of "valid" disparity values $[l_i, u_i]$ for each pixel $i$. Our bilateral-space embedding technique is agnostic to any specific design decisions made in computing these intervals, but of course, the choice of how these cost volumes are computed has a profound effect on the output disparity map.

Let us assume that we have two grayscale rectified images: $I_0$ and $I_1$. Given a pixel of interest at $(x, y)$ in $I_0$, because of rectification we know that the corresponding pixel (provided it is not occluded) lies somewhere in $I_1(x + [0 : D), y)$, where $D$ is the maximum possible disparity assumed to be present in the input image. To compute the lower bound for this pixel of interest, we must scan all values $x' \in [x, x + D)$, and record the $y$ coordinate of the first and last matching pixels in $I_1$. Of course, this requires a test for whether or not two pixels "match", which can be a difficult task given noise and the nature of image sampling. To deal with sampling, we use the technique of Birchfield-Tomasi [4] to compute upper and lower envelopes for $I_0$ and $I_1$, and measure differences between those envelopes instead of differences between the underlying images. This gives us a sampling-invariant similarity measure. To deal with noise, we artificially expand our bounds by some noise parameter $\epsilon$ ($\epsilon = 4$ in all of our experiments). This ad-hoc noise model means that we generally ignore the shot noise that is common in real images, though this is a very crude approach largely motivated by convenience. To formalize:

$$U_0 = \max2(\mathrm{blur2}(I_0)) + \epsilon \tag{24}$$

$$L_0 = \min2(\mathrm{blur2}(I_0)) - \epsilon \tag{25}$$

where $\max2(\cdot)$ is a $2 \times 2$ "max" filter, $\min2(\cdot)$ is a $2 \times 2$ "min" filter, and $\mathrm{blur2}(\cdot)$ is a $2 \times 2$ box filter. $U_1$ and $L_1$ are defined similarly. The pixel at $(x, y)$ in $I_0$ is said to match the pixel at $(x + d, y)$ in $I_1$ iff:

$$\begin{aligned} M(x, y, d) = (U_0(x, y) \geq L_1(x + d, y)) \\ \wedge\, (L_0(x, y) \leq U_1(x + d, y)) \end{aligned} \tag{26}$$

Where $M(x, y, d)$ is a binary volume of matches (a "match volume"), indicating whether or not the envelopes at $(x, y)$ in $I_0$ and $(x + d, y)$ in $I_1$ overlap. With $M(\cdot)$, we can define our upper and lower intervals as:

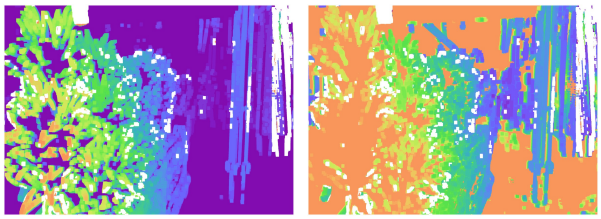
$$l_{x,y} = \arg\min_d M(x, y, d) \tag{27}$$

$$u_{x,y} = \arg\max_d M(x, y, d) \tag{28}$$

If no match is found we set $l_{x,y} = 0$ and $u_{x,y} = D - 1$, causing the data term corresponding to unmatched pixels to effectively be ignored during inference.

The upper and lower bounds as described here work reasonably well, but as is often the case in stereo matching, performance improves if patches are used instead of pixels. So before computing these intervals, we first filter each channel of our match volume $M$ with a $25 \times 25$ "and" filter

(a) A stereo pair



(b) Our recovered lower and upper disparity bounds

Figure 2: Given the stereo pair in Fig. 2a, for each patch in the right image we identify the set of corresponding patches along the epipolar scanline in the left image. The upper and lower bounds of each pixel's matching interval are shown in Fig. 2b. Flat, textureless regions have wide, uninformative ranges, while textured regions have tightly-localized bounds in disparity-space. Some patches (mostly near occlusion boundaries where the foreground and background both are textured) have no corresponding patch in the right image, shown in the visualization as white pixels.

in $x$ and $y$. This means that instead of $M(x, y, d)$ indicating a single pixel match, it indicates that *all* pixels in the $25 \times 25$ patch around $(x, y)$ in $I_0$ match all corresponding pixels in the patch around $(x + d, y)$ in $I_1$. Evaluating the $25 \times 25$ "and" filter is fast, as it is separable in $x$ and $y$ and can be computed recursively. Here is pseudocode for "and"-filtering our match volume in $x$:

$$\begin{aligned} M(x,y,d) \leftarrow & M(x,y,d) \\ & \wedge\ M(x-1,y,d) \wedge M(x-2,y,d) \\ & \wedge\ M(x+1,y,d) \wedge M(x+2,y,d) \end{aligned} \tag{29}$$

$$\begin{aligned} M(x,y,d) \leftarrow & M(x,y,d) \\ & \wedge\ M(x-5,y,d) \wedge M(x-10,y,d) \\ & \wedge\ M(x+5,y,d) \wedge M(x+10,y,d) \end{aligned} \tag{30}$$

Filtering in $y$ is done similarly.

This block-matching similarity is particularly brittle: a patch only matches another patch iff all constituent pixels of both patches match, according to our Birchfield-Tomasi pixel similarly metric. This may seem like a questionable design decision, but it produces desirable behavior at occlusion edges. At occlusions, it is very rare for the image content at both sides of the occluding edge to match (unless the background and the foreground look similar, in which case we can produce inaccurate depth without affecting defocus image quality). This means that our intervals tend to only provide information on the textured interiors of objects, and provide little information at object edges. Given that information is scarce at object edges, our output disparities near object edges are largely "inpainted" according to bilateral similarity. This gives us the nice edge-aware disparity maps that are characteristic of our technique.

This block-matching interval-finding procedure is fairly efficient even if implemented naively. Computing the Birchfield-Tomasi envelopes is simple, and performing a pixel match requires just two comparisons. Even naively computing all of $M$ is fairly efficient because, unlike standard cost-volumes whose entries are usually 8-bit or 32-bit values, our match-volume is composed of booleans and so can be stored compactly. This same property makes our $25 \times 25$ "and" filter especially fast, because if 64 values of $M$ are stored as a single 64-bit value they can be and-ed together with a single "and" operation. We implemented this block-matching procedure in Halide [10], which allows us to easily "schedule" our code such that $M$ is never explicitly in memory at once.

See Figure 2 for examples of our upper and lower bounds for a stereo pair.

## 4. Efficient Cost-Volume Splatting

In the paper, we claimed that it is possible to compute a weighted sum of hinge losses by first computing a weighted sum of delta functions, and then integrating that sum twice. More formally, assuming that we have a set of hinge losses with weights $w_i$ and hinge-locations $a_i$, we know that:

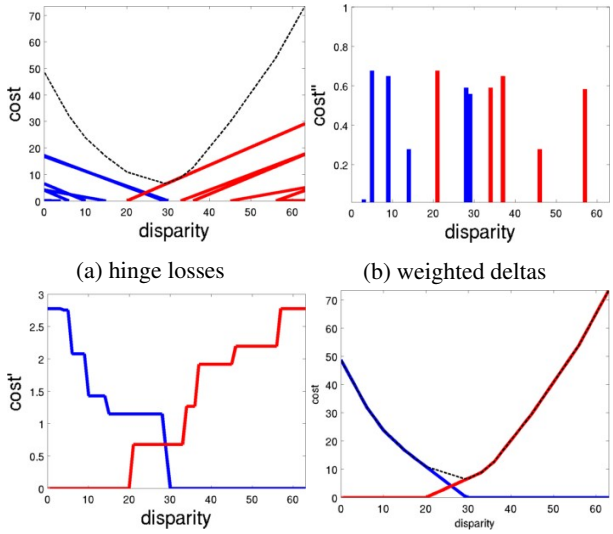
$$\sum_i w_i \max(0, x - a_i) = \iint \sum_i w_i \delta(x - a_i) \tag{31}$$

This equivalence is demonstrated in Figure 3.

## 5. Multiscale Optimization

As is often the case during optimization, convergence can be made faster by allowing optimization to proceed in a course-to-fine fashion, or to happen simultaneously at multiple scales. We will borrow the multiscale optimization technique from [3], but adapt it from Gaussian pyramids of images to pyramids of bilateral representations. We will construct a hierarchy of bilateral representations and solve our optimization problem in that hierarchical space, thereby allowing optimization to proceed in a multiscale fashion.

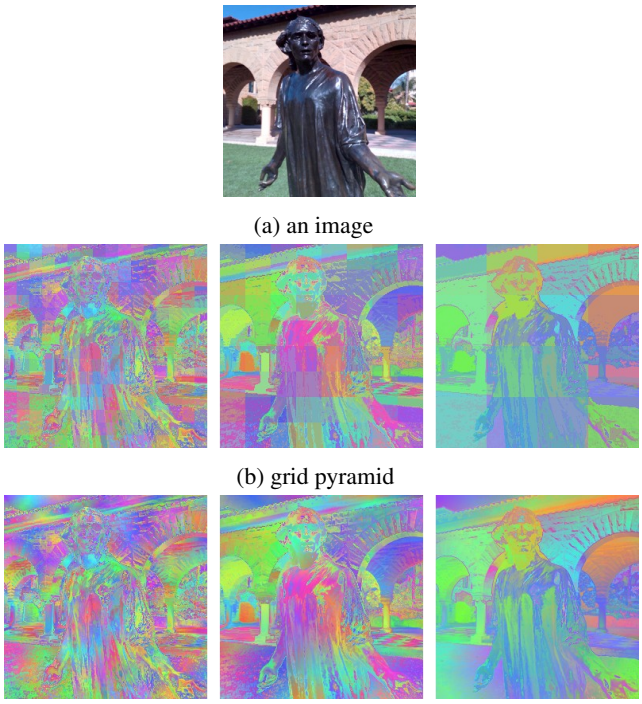
When we construct a bilateral representation (a grid or lattice), we produce a splat matrix $S$ and a matrix of vertex coordinates $V$. In our case, $V$ is an $m$ by 5 matrix, where $m$ is the number of vertices and 5 is the dimensionality of our bilateral-space (where the dimensions correspond to each vertex's RGB color value and XY pixel position). With this




(a) hinge losses (b) weighted deltas

(c) weighted Heavisides (d) total loss

Figure 3: Efficiently "splatting" our data term loss from pixel-space to bilateral-space requires that we exploit some properties of hinge-loss. In Fig. 3a we have a set of losses we wish to splat, shown in terms of their constituent upper-bound (red) and lower-bound (blue) losses as well as the total splatted loss (black), which can inefficiently be computed as the sum of the constituent losses. In Fig. 3b we have a splat (a weighted histogram) of the unsigned second derivatives of our constituent hinge-losses, which are delta functions. By integrating these second derivatives (where the lower-bound costs are integrated right-to-left and the upper-bound costs are integrated left-to-right) we get the summed Heaviside functions shown in Fig. 3c, and by integrating these again we get the total upper-bound and lower-bound losses and their sum in Fig. 3d, which is the same as the total loss shown in Fig. 3a. This accelerated approach of splatting hinge losses by splatting and integrating their second derivatives is orders of magnitude faster than the naive approach when the number of losses being splatted is large.



(a) an image

(b) grid pyramid

(c) lattice pyramid

Figure 4: When optimizing, we construct a pyramid of bilateral representations for an image and solve our optimization problem in that overcomplete pyramid space. Shown here is a small input image (Fig. 4a) and 3-level pyramids of simplified bilateral grids (Fig. 4b, left is fine-scale, right is coarse-scale) and of permutohedral lattices (Fig. 4c) where each grid/lattice is visualized by "slicing" a set of random colors — each color corresponds to a variable in our optimization. At higher scales of the pyramid, vertices parametrize much larger areas (in a bilateral sense) of the input image, which allows optimization to proceed more quickly.

matrix of vertices we can construct a more coarse bilateral representation on top of these "base" vertices, and then repeat that procedure until convergence to form a pyramid:

**for** $k = [1 : K]$ **do**
  $V \leftarrow V/2$
  $(S_k, V) \leftarrow \text{bilateral\_representation}(V)$
**end for**

Where $K$ is the number of scales, which we set to be as large as possible until we are left with just a single vertex at the top of the pyramid. We repeatedly divide all the vertex coordinates by 2, and then construct a new bilateral representation from those reduced coordinates, effectively downsampling the vertices at each level of the pyramid by a factor of 2. This is analogous to constructing a Gaussian pyramid of an image as was done in [3], with the only difference here that we are using bilateral representations to construct a "bilateral pyramid". Grid and lattice-based pyramids can be seen in Figure 4.

With our $K$ splat matrices we can project a vector of vertex depths $\mathbf{v}$ onto a pyramid of overlapping vertices:

$$P(\mathbf{v}) = [S_K \dots S_2 S_1 \mathbf{v}, \dots, S_2 S_1 \mathbf{v}, S_1 \mathbf{v}, \mathbf{v}] \quad (32)$$

We can transpose this pyramid operation as well:

$$P^{\mathrm{T}}(\mathbf{w}) = [S_1^{\mathrm{T}} S_2^{\mathrm{T}} \dots S_K^{\mathrm{T}} \mathbf{w}, \dots, S_1^{\mathrm{T}} S_2^{\mathrm{T}} \mathbf{w}, S_1^{\mathrm{T}} \mathbf{w}, \mathbf{w}] \quad (33)$$

Of course, $P^{\mathrm{T}}(\mathbf{w})$ and $P(\mathbf{v})$ can be computed efficiently from the bottom up by reusing information. As was the case

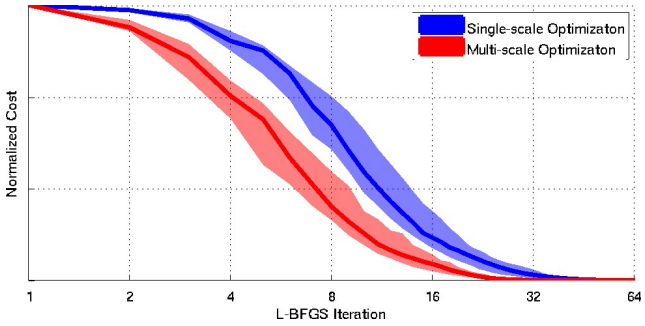



Figure 5: A summary of optimization traces for 20 4-megapixel image pairs, normalized to the same range. The line is the median normalized loss on each iteration, and the shaded areas show the $[25, 75]$ percentile range. Using our multiscale optimization scheme, we almost halve the number of iterations of L-BFGS required for convergence.

in [3], we found it beneficial to normalize each variable in our pyramid representation by the square root of its "mass":

$$Q = \text{diag}\left(P(S\mathbf{1})\right)^{-1/2} \tag{34}$$

We can now define a new optimization problem (that is, a new loss function and its gradient) in our pyramid-space in terms of the our previously defined bilateral-space loss function and gradient:

$$\text{pyr\_loss}(\mathbf{w}) = \text{loss}(QP^{\mathrm{T}}(\mathbf{w})) \tag{35}$$

$$\nabla\,\text{pyr\_loss}(\mathbf{w}) = QP(\nabla\text{loss}(QP^{\mathrm{T}}(\mathbf{w}))) \tag{36}$$

Our final optimization procedure is to solve for a set of pyramid-space depths:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \text{pyr\_loss}(\mathbf{w}) \tag{37}$$

and then from that produce a per-pixel disparity map:

$$\mathbf{p}^* = S^{\mathrm{T}}QP^{\mathrm{T}}(\mathbf{w}^*) \tag{38}$$

Because $QP(\cdot)$ and $QP^{\mathrm{T}}(\cdot)$ are both linear operations, this multiscale optimization scheme can be thought of as a preconditioner.

See Figure 5 for a demonstration of the improved convergence resulting from using this multiscale optimization scheme compared to single-scale optimization. Using this multiscale preconditioner increases the time taken to evaluate our loss function by about $10\%$, but causes convergence to occur roughly twice as fast. In all of our results we terminate optimization after $25$ iterations of L-BFGS, which Figure 5 suggests is sufficient.

## 6. Rendering

In the paper we use an algorithm for rendering synthetically-defocused shallow-depth-of-field images to produce our final output, and as an intermediate step in benchmarking our own technique and the baseline stereo techniques we compare against. Our rendering technique is simple, as this is not the focus of our contribution. Still, a somewhat accurate rendering algorithm is necessary for a fair evaluation.

The rendering technique we use is straightforward: given an image $I$, a disparity-map $D$, a target disparity that we want to be in focus $t$, a desired magnitude of the resulting shallow-depth-of-field effect $m$, and a function $\text{disc\_blur}(I, r)$ which applies a disc blur of radius $r$ to image $I$, our algorithm is as follows:

$I_n \leftarrow \vec{0}$
$I_d \leftarrow \vec{0}$
**for** $d = [\min(D) : 1/m : \max(D)]$ **do**
  $A \leftarrow |D - d| \leq 1/m$
  $B \leftarrow I \times A$
  $A_b \leftarrow \text{disc\_blur}(A, m|d - t|)$
  $B_b \leftarrow \text{disc\_blur}(B, m|d - t|)$
  $I_n \leftarrow I_n \times (1 - A_b) + B_b$
  $I_d \leftarrow I_d \times (1 - A_b) + A_b$
**end for**
$I \leftarrow I_n / I_d$

Where $\times$ and $/$ are element-wise multiplication and division respectively. This algorithm sweeps from the "back" of the disparity image to the "front", and composites a series of blurred images together with alpha matting such that the blur applied to each layer is proportional to the difference between the disparity at that layer and the target disparity $t$. This technique is simple, reasonably fast, and appears to work well in practice. It would be much cheaper to use Gaussian blurs instead of disc blurs, but a Gaussian synthetic blur lacks the distinctive "bokeh" of a disc blur.

When rendering synthetically defocused images, and when computing our ground-truth shallow-depth-of-field images by compositing images from our light field, we take care to first linearize the input image(s), do the compositing or blurring in that linear space, and un-linearize the output averaged/blurred image. This linearization is both more physically accurate, and produces much better looking images by preserving bright highlights.

## 7. Profiling

Because speed is an important aspect of our algorithm, let us analyze the relative cost of each stage of our algorithm. We ran the "grid" instance of our algorithm on a large set of 4-megapixel stereo pairs and gathered statistics on the time taken on each stage. Here are the means and standard deviations of the runtimes of these stages:

| Algorithm Component | Time (ms) |
|---|---|
| Block-Interval Matching | 179±7 |
| Bilateral Grid Construction | 27±7 |
| Bilateral Pyramid Construction | 7±3 |
| Interval Splatting | 83±49 |
| L-BFGS Optimization | 480±247 |
| Solution Slicing | 7±1 |
| Domain Transform Post-Processing | 64±9 |
| Total | 847±294 |

We see that time is mostly dominated by the time spent doing L-BFGS optimization in bilateral-space, with our block-matching / interval-finding procedure taking the next most significant amount of time. Our optimization step has an unusually high variance in its runtime, as the number of vertices in our optimization problem is dependent on image content.

## 8. Evaluation & Error Metrics

The most intuitive way to evaluate the performance of a stereo algorithm for defocus is to visually inspect the renderings produced using that algorithm. With that goal in mind, we present a series of visualizations from our own stereo dataset of 4-megapixel images with a 10mm baseline between the cameras, shown in Figures 7-12. Close inspection of these results shows that our algorithm consistently produces better looking defocused images than the baseline stereo algorithms we evaluate against. This is reinforced by our user study, as detailed in the main paper.

Though a qualitative user study is compelling, we would also like a quantitative evaluation of the quality of our defocused renderings. Traditional stereo error metrics measure the per-pixel difference between an estimated disparity map and some ground-truth disparity map. This is a reasonable approach for evaluating a stereo algorithm in general terms, but it has serious issues in the context of our specific defocus task. First, it requires that we have excellent ground-truth disparity maps, which are very difficult to obtain. Even the ground-truth depthmaps in state-of-the-art stereo datasets [13] often have missing values near occlusion edges, which are where noticeable defocus rendering artifacts tend to occur. Second, evaluating errors in disparity as a proxy for errors in defocus quality presupposes the assumption that good defocused images can be produced by rendering images according to disparity maps, which is precisely one of the claims that we wish to validate. Third, some mistakes that would be heavily penalized by traditional stereo error metrics produce no noticeable change in defocus quality, while other mistakes that would incur very little error by traditional error metrics may produce egregious artifacts in a defocused rendering. To this end, we introduce a benchmark in which we attempt to directly measure the quality of the rendered defocused images produced by a stereo algorithm, rather than measuring only disparity map quality.

For our benchmark we will recover a depth map from a stereo pair, render a set of images (a focal stack) using an image from that stereo pair and the recovered depth, and compare those renderings to "true" shallow-depth-of-field images generated from a light field. Such an end-to-end defocus evaluation does not require ground-truth depth maps, but instead requires rectified stereo pairs alongside ground-truth focal stacks. To accomplish this we use the Stanford Light Field Archive [2], which contains 12 calibrated light fields of real-world scenes and objects. From one such light field we can extract a rectified stereo pair and, by averaging the subset of the light field where the camera position lies within a disc, we can generate "ground-truth" shallow-depth-of-field focal stacks. These images are not perfect, as the light field may be angularly undersampled (only 7 of the 12 light fields in the archive are sampled densely enough for our purposes) but this dataset is otherwise well suited to our task. Unlike a rendering produced by an image and a depth map, an average image from a light field correctly models occlusions and specularities just as a camera lens does, and it is easy to guarantee that each focal stack is perfectly aligned to an image from each stereo pair. The sizes of the images in this dataset vary, but are on average $\sim$1.1 megapixels. For each of our 7 well-sampled light fields we generated a dense "ground truth" focal stack, and we selected 4 evenly-spaced disparities at which renderings from each stereo algorithm's disparity map were produced, giving us 28 distinct test-set scenes in total.

When scrutinizing the errors in defocused renderings some clear trends emerge. Some depth mistakes are completely unobservable in the defocused rendering. For example, often the depth of a clear blue sky or of a flat black background is completely mis-estimated by a stereo algorithm, but this mistake is subtle or invisible in the defocused image because a blurred flat patch looks identical to a non-blurred flat patch. Sometimes we see errors in which there is a foreground object occluding a background object of the same color, for which nearly all stereo algorithms produce a poorly localized depth discontinuity. Again, this is usually unobservable in the output rendering, as depth discontinuities are only visible in a defocused image if they coincide with image discontinuities.

Interestingly, the requirements of the defocus task are fundamentally forgiving to the shortcomings of stereo algorithms. Some of the most common error cases found in the disparity maps produced by stereo algorithms are invisible in the defocused images. The places where errors matter most are depth discontinuities which co-occur with image discontinuities (a black object occluding a white object, for example), which are exactly where a stereo algorithm has the most actionable information for estimating depth (provided the edge is not perfectly horizontal, and therefore

aligned with the baseline of the stereo rig). This insight informs the design of our bilateral-space stereo algorithm, and it also informs our choice of error metric.

The most offensive errors in defocused renderings tend to occur at highly textured occlusion boundaries in which the depth map does not tightly follow the input image. This causes textured regions to be split into in-focus and out-of-focus regions, and produces sharp edges which are neither present in the input image nor in the "ground-truth" defocused image. The most egregious errors tend to be image patches which, rather than failing to resemble "the" ground-truth image, fail to resemble *any image in the ground truth "focal stack" of the scene*. With this in mind, we will use a set of error metrics in which, for every part of the image, we compare the rendered image to *all* possible defocused images which may have come from the ground-truth light field. This makes us insensitive to subtle errors which humans tend not to notice (i.e., the interior of an object being slightly blurrier than it should be due to a mis-estimated disparity of the entire interior) but highlights noticeable errors (a sharp, jagged edge where there should be a smooth edge or no edge at all). For a visualization of the motivation for our error metric, see Figure 13. This approach of measuring error with respect to a ground-truth focal stack instead of a ground-truth image has the additional benefit that no ground-truth disparity is required for evaluation — though it does require a well-sampled light field.

We will build our focal-stack error metrics on top of four common image error metrics. Given two images, we will measure their per-pixel difference with the absolute deviation of individual pixel values ("pixel"), the gradient magnitude of each pixel ("grad"), the average absolute deviation of the $8 \times 8$ patch centered around each pixel ("patch"), and DSSIM [14]. Assuming an output rendering $R$ and a ground-truth focal stack $S$ where $S_k$ is the $k$'th image in the focal stack, we will take the per-pixel minimum of each error metric across the entire focal stack, producing one final error image for each metric. Though we take the minimum independently for each pixel, because three of our four error metrics are computed over a region, the selection of which layer to draw the minimum value from tends to be spatially smooth. To formalize our per-pixel error images:

$$\begin{aligned}
\text{min_pixel}(x,y) &= \min_k |R(x,y) - \nabla S_k(x,y)| \\
\text{min_patch}(x,y) &= \min_k \text{boxfilt}(|R(x,y) - S_k(x,y)|, 8) \\
\text{min_grad}(x,y) &= \min_k |\|\nabla R\|(x,y) - \|\nabla S_k\|(x,y)| \\
\text{min_dssim}(x,y) &= \min_k \text{DSSIM}(R, S_k)(x,y)
\end{aligned} \tag{39}$$

For color images, the error metrics are assumed to be summed before the min operator, except for DSSIM which we compute only with luma. The image gradient magnitude $\|\nabla \cdot\|$ is computed with the technique of [3]. We use standard SSIM parameters: $k = [0.01, 0.03]$, $\sigma = 1.5$.

Now that we have these per-pixel error images, we must reduce each metric's image down to a single number. We found that taking the average error across all pixels did not reflect the perceptual quality of an image; most viewers (especially trained photographers) tend to dislike a rendering with a single severe error much more than a rendering with several inconspicuous errors. We therefore report the maximum error (the $\infty$-norm) at any pixel in each error image, and because the maximum is a sensitive error metric we also report the $4$-norm. The $4$-norm and the $\infty$-norm both penalize a small number of severe errors much more heavily than a large number of small errors. To formalize:

$$\text{pixel}^4 = \left( \sum_{x,y} \text{min\_pixel}(x,y)^4 \right)^{1/4} \tag{40}$$

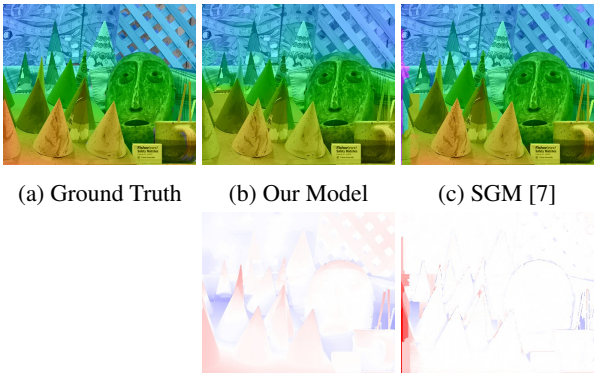
$$\text{pixel}^\infty = \max_{x,y} \text{min\_pixel}(x,y) \tag{41}$$

$\text{patch}^4$, $\text{patch}^\infty$, $\text{grad}^4$, $\text{grad}^\infty$, $\text{dssim}^4$, and $\text{dssim}^\infty$ are defined similarly. Given these error metrics for each image, we compute the geometric mean of each error metric to produce one average error metric. When reporting performance across the entire dataset, we report the geometric mean of each error metric, including the average, across each image age. The geometric mean is used instead of the arithmetic mean because it is invariant to the varying scale of each error metric, invariant to being dominated by a particularly challenging image, and associative.

## 9. Middlebury Evaluation

Though it is not our goal to produce a general-purpose stereo algorithm, we would be remiss to not include an evaluation against the Middlebury stereo dataset. Our technique performs poorly on the dataset, as the Middlebury error metrics (the percentage of pixels which are more than $\epsilon$ disparity values away from the ground-truth disparity, where $\epsilon$ is usually 1 or 2) heavily penalizes the kind of errors our algorithm readily make, such as systematically under- or over-estimating the disparity of flat textureless regions. The Middlebury error metric is also fairly insensitive to the sorts of errors that we care about most: failing to follow image edges at occlusion boundaries, where errors in disparity can result in dramatic rendering errors. Our performance on V2 of the Middlebury stereo dataset [12] is shown in Table 1. Included for comparison in that table is the performance of one of our baseline algorithms, taken from the Middlebury evaluation page. We visualize our performance relative to that baseline more closely in Figure 6.

We include this comparison despite the fact that it is unfavorable towards our algorithm, because it highlights an important aspect of our contribution. We have previously shown that our technique does better at the "defocus" task



(a) Ground Truth (b) Our Model (c) SGM [7]

(d) Our Model (e) SGM [7]

Figure 6: Our model compared with a baseline stereo algorithm on an image from the Middlebury dataset. On the top we show disparity rendered on top of a grayscale version of the input image, and on the bottom we show the signed error of each disparity map from the ground-truth. Our technique tends to produce biased disparities on the interiors of objects, though we closely follow object edges. Other algorithms tend to be accurate on the interiors of objects but inaccurate near edges, as this sort of behavior is favored by the Middlebury error metrics.

than the baseline algorithms that are conventionally thought of as the state-of-the-art. Similarly, we have shown that our algorithm performs worse than these other algorithms at the more conventional stereo task. This suggests that the goals of these two tasks are different — perhaps even contradictory. This finding reinforces the approach taken in this paper for the defocus task: that the target use-case of a stereo algorithm should be the primary driving force behind the choice of evaluation benchmark. It also suggests that the most useful stereo algorithm for a specific task may not be what is conventionally regarded as the state-of-the-art, depending on the nature of that task.

Error Threshold = 1

| Method | Tsukuba | | | Venus | | | Teddy | | | Cones | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc |
| SGM [7] | 3.26 | 3.96 | 12.8 | 1.00 | 1.57 | 11.3 | 6.02 | 12.2 | 16.3 | 3.06 | 9.75 | 8.90 |
| Ours (grid) | 19.4 | 20.3 | 62.4 | 21.9 | 23.0 | 50.0 | 26.8 | 33.0 | 53.5 | 26.9 | 32.0 | 49.6 |

Error Threshold = 2

| Method | Tsukuba | | | Venus | | | Teddy | | | Cones | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc | nonocc | all | disc |
| SGM [7] | 2.01 | 2.52 | 9.14 | 0.36 | 0.68 | 4.19 | 3.46 | 6.55 | 9.26 | 2.38 | 8.15 | 7.00 |
| Ours (grid) | 6.36 | 6.76 | 127.9 | 6.59 | 7.34 | 29.4 | 12.6 | 19.1 | 30.3 | 14.8 | 19.5 | 32.8 |

Table 1: The performance of our algorithm on the V2 Middlebury stereo dataset [12], shown with one of our baseline algorithms for comparison. Our algorithm does poorly at this benchmark, as it favors very different kinds of errors than our defocus benchmark.

## References


[1] A. Adams, J. Baek, and M. A. Davis. Fast high-dimensional filtering using the permutohedral lattice. *Eurographics*, 2010.

[2] A. Adams, V. Vaish, , B. Wilburn, N. Joshi, M. Levoy, and Others. The stanford light field archive. `http://lightfield.stanford.edu/index.html`.

[3] J. T. Barron and J. Malik. Shape, illumination, and reflectance from shading. Technical Report UCB/EECS-2013-117, EECS, UC Berkeley, May 2013.

[4] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *IJCV*, 1999.

[5] J. Chen, S. Paris, and F. Durand. Real-time edge-aware image processing with the bilateral grid. *SIGGRAPH*, 2007.

[6] A. Geiger, M. Roser, and R. Urtasun. Efficient large-scale stereo matching. *ACCV*, 2010.

[7] H. Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. *CVPR*, 2005.

[8] P. Knight, D. Ruiz, and B. Uar. A Symmetry Preserving Algorithm for Matrix Scaling. *SIAM. J. Matrix Anal. & Appl*, 2014.

[9] P. Milanfar. Symmetrizing smoothing filters. *SIAM J. Imaging Sci.*, 2013.

[10] J. Ragan-Kelley, A. Adams, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *SIGGRAPH*, 2012.

[11] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. *CVPR*, 2011.

[12] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. *CVPR*, 2003.

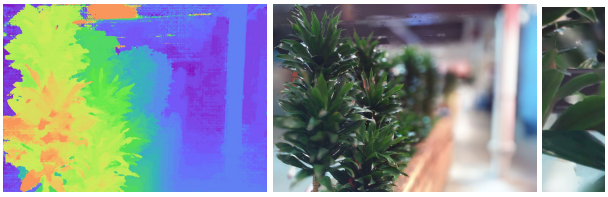
[13] S. N. Sinha, D. Scharstein, and R. Szeliski. Efficient high-resolution stereo matching using local plane sweeps. *CVPR*, 2014.

[14] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *TIP*, 2004.

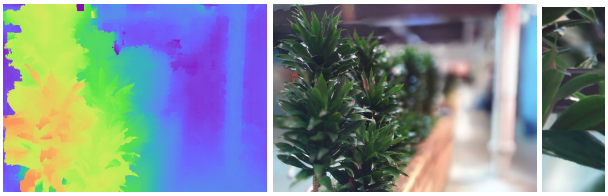
[15] K. Yamaguchi, D. A. McAllester, and R. Urtasun. Efficient joint segmentation, occlusion labeling, stereo and flow estimation. *ECCV*, 2014.

(a) A stereo pair with cropped subregions of the right image

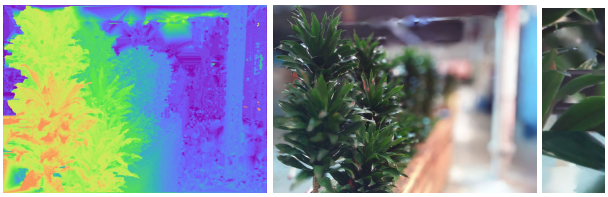
(b) Ours (grid) + DT



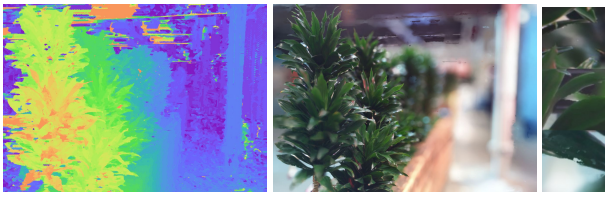
(c) SGM [7]



(d) SPS-StFl [15]



(e) LIBELAS [6]

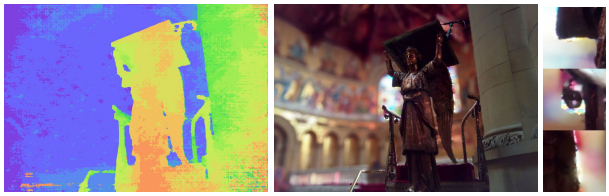


(f) CostFilter [11]

Figure 7: Given a 4-megapixel stereo pair, we evaluate a set of stereo algorithms on that pair, produce disparity maps, and with those render synthetically defocused images. Our technique tends to produce more natural and less objectionable renderings than others. The cropped tiles shown here are the same images that were used in our user study.

(a) A stereo pair with cropped subregions of the right image

(b) Ours (grid) + DT



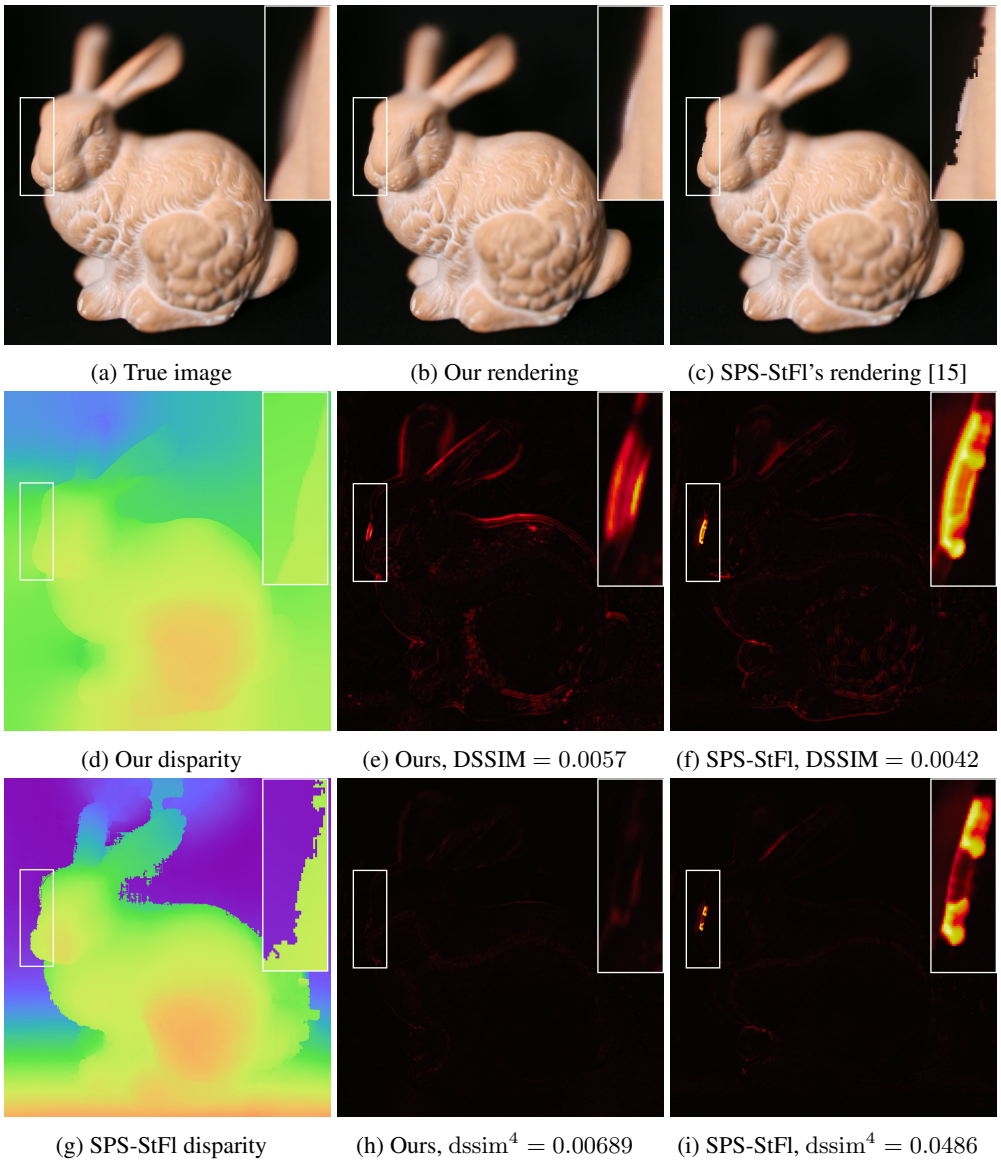
(c) SGM [7]

(d) SPS-StFl [15]

(e) LIBELAS [6]

(f) CostFilter [11]

Figure 8: More results in the same format as Figure 7.



(a) True image (b) Our rendering (c) SPS-StFl's rendering [15]

(d) Our disparity (e) Ours, DSSIM $= 0.0057$ (f) SPS-StFl, DSSIM $= 0.0042$

(g) SPS-StFl disparity (h) Ours, $\mathrm{dssim}^4 = 0.00689$ (i) SPS-StFl, $\mathrm{dssim}^4 = 0.0486$

Figure 13: A demonstration of the shortcomings of conventional error metrics for our task. In Fig. 13a we have a ground-truth shallow-depth-of-field image from our light field dataset. By looking at the recovered disparity maps in Fig. 13d and Fig. 13g we see that our algorithm's disparity, though very smooth and well-aligned to image edges, is incorrect at the ground-plane and the background. But these mistakes occur at flat image regions, and are therefore not visible in our output rendering in Fig. 13b, demonstrating our claim that not all disparity errors produce rendering errors. The baseline depth is often more accurate than ours but makes crucial errors at occlusion edges, thereby causing the conspicuous rendering artifacts in Fig. 13c. As can be seen in the zoomed region, both renderings are different from the ground-truth image, but our error is inconspicuous while the baseline error is very noticeable. A traditional error metric such as DSSIM does not reflect this, and instead reports large errors throughout both renderings, most of which are inoffensive or difficult to see. Though DSSIM is large at the conspicuous artifact in the baseline rendering, the mean DSSIM reported here is swamped by the error signal at the other parts of the image, which (erroneously, in our judgement) reports that the baseline rendering has less error than our algorithm's rendering. This is why we introduce our own error metrics (such as $\mathrm{dssim}^4$) which work by taking the per-pixel minimum error with respect to a ground-truth focal stack instead of a single ground-truth image, and then computing a $p$-norm over the entire error image where $p$ is large, which penalizes egregious errors more than small errors. As can be seen in Fig. 13h and Fig. 13h, this new error metric penalizes noticeable rendering artifacts more than inconspicuous ones. This error may seem like a corner-case, but in real-world scenes with heavy occlusion, errors like this are common.

(a) A stereo pair with cropped subregions of the right image

(b) Ours (grid) + DT

(c) SGM [7]

(d) SPS-StFl [15]

(e) LIBELAS [6]

(f) CostFilter [11]

Figure 9: More results in the same format as Figure 7.

(a) A stereo pair with cropped subregions of the right image

(b) Ours (grid) + DT

(c) SGM [7]

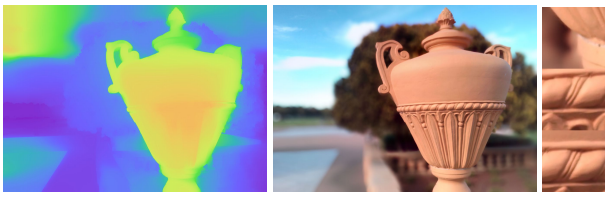
(d) SPS-StFl [15]

(e) LIBELAS [6]

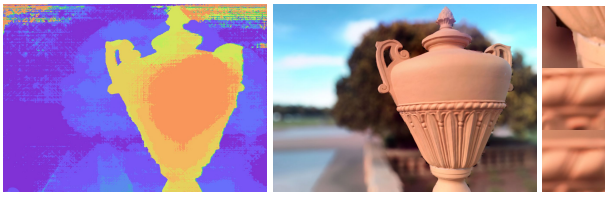
(f) CostFilter [11]

Figure 10: More results in the same format as Figure 7.

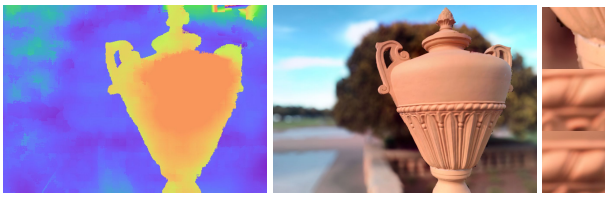
(a) A stereo pair with cropped subregions of the right image



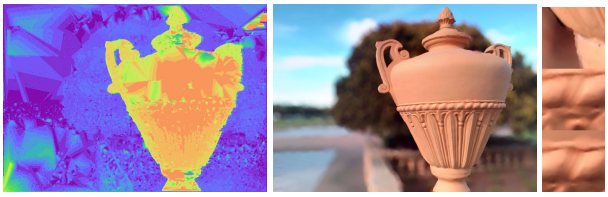
(b) Ours (grid) + DT



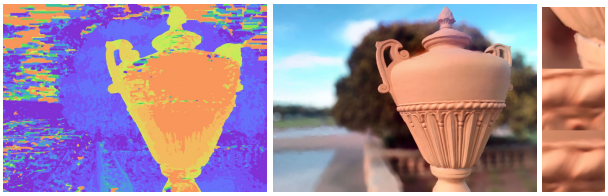
(c) SGM [7]



(d) SPS-StFl [15]



(e) LIBELAS [6]



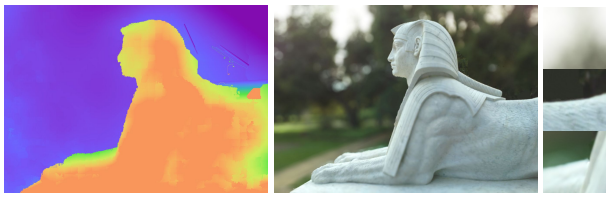
(f) CostFilter [11]

Figure 11: More results in the same format as Figure 7.

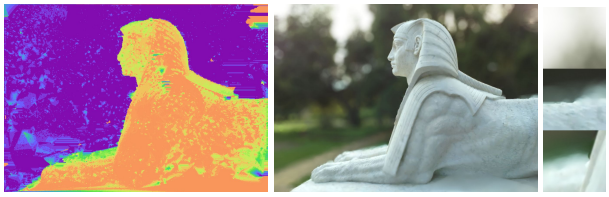
(a) A stereo pair with cropped subregions of the right image

(b) Ours (grid) + DT

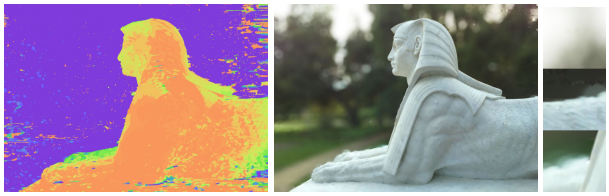
(c) SGM [7]



(d) SPS-StFl [15]



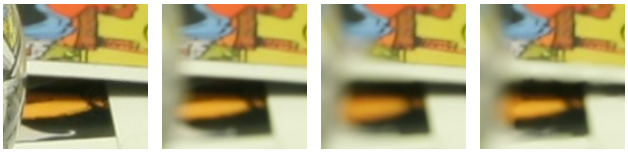
(e) LIBELAS [6]



(f) CostFilter [11]

Figure 12: More results in the same format as Figure 7.

(a) light field (b) true defocus (c) stereo pair

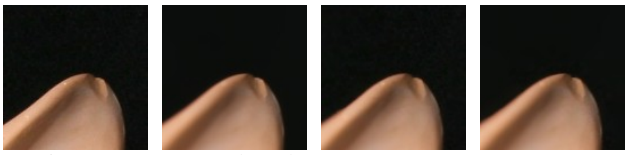


input ground truth Ours LIBELAS

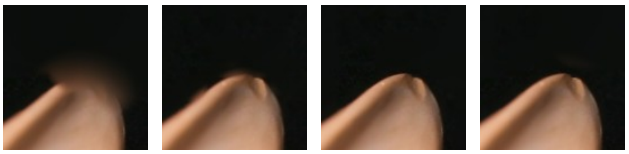
SGM LPS SPS-StFl SAD-lattice

Figure 14: In Fig. 14a we show the constituent images of one scene from the light field dataset. The images that we average together to produce our "ground truth" shallow-depth-of-field image (Fig. 14b) are circled in magenta, and the two images we took to form our stereo pair (Fig. 14c, visualized as a red-blue anaglyph) are circled in green. The following figures show a cropped region of one input image, the ground-truth refocused rendering, and the output defocused renderings produced with different stereo algorithms. These images were automatically cropped to the location where the renderings disagreed the most (the window with the maximum variance across the gradients-magnitudes of the renderings). Our output tends to look natural and similar to the ground-truth, while others tend to have false edges and artifacts.

(a) light field (b) true defocus (c) stereo pair

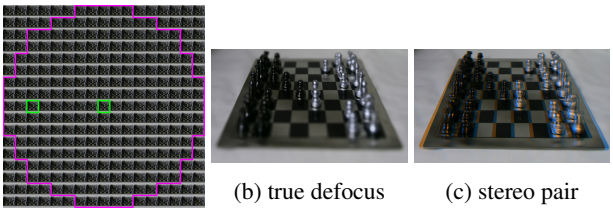


input ground truth Ours LIBELAS



SGM LPS SPS-StFl SAD-lattice

Figure 15: More results from our light field dataset, in the same format as Figure 14



(a) light field (b) true defocus (c) stereo pair

input ground truth Ours LIBELAS

SGM LPS SPS-StFl SAD-lattice

Figure 16: More results from our light field dataset, in the same format as Figure 14