Fast Bilateral-Space Stereo for Synthetic Defocus

Jonathan T. Barron
barron@google.com

Andrew Adams
abadams@google.com

YiChang Shih
yichang@mit.edu

Carlos Hernández
chernand@google.com

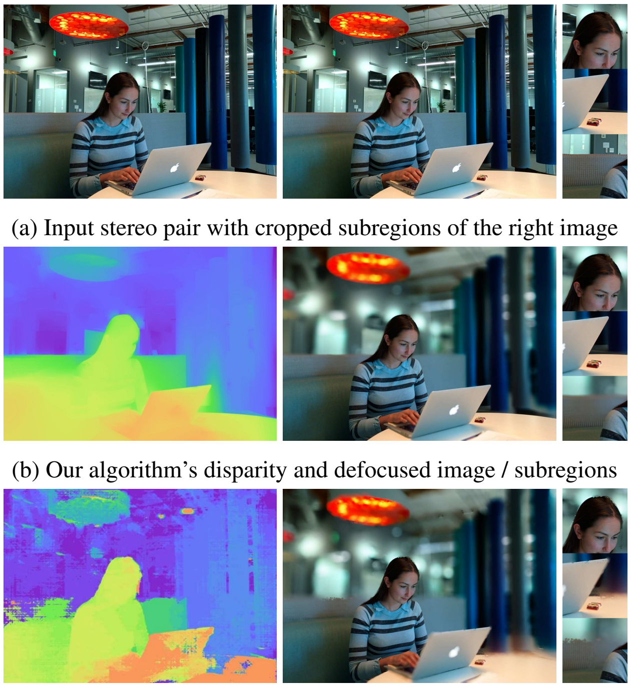
Abstract

Given a stereo pair it is possible to recover a depth map and use that depth to render a synthetically defocused image. Though stereo algorithms are well-studied, rarely are those algorithms considered solely in the context of producing these defocused renderings. In this paper we present a technique for efficiently producing disparity maps using a novel optimization framework in which inference is performed in "bilateral-space". Our approach produces higher-quality "defocus" results than other stereo algorithms while also being $10 - 100\times$ *faster than comparable techniques.*

1. Introduction

Expensive DSLR cameras are capable of producing shallow-depth-of-field images — objects at a certain distance from the camera are in focus, while nearer or farther objects are out of focus. This is due to the large apertures (and equally-large lenses) of DSLRs. Cellphone cameras, in contrast, have small apertures and therefore deep depths-of-field; in most non-macro cellphone camera images, the entire scene is in focus. Because consumers and photographers generally prefer shallow-depth-of-field images, companies such as HTC [15] have begun producing devices with two rear-facing cameras to enable this "defocus" effect. With two or more images of a subject, stereo techniques can be used to estimate a per-pixel disparity map, which can then be used to render synthetically defocused shallow-depth-of-field images.

Accomplishing this defocus effect requires a stereo algorithm with a number of distinct properties. Because consumer cameras have high resolutions (4-16 megapixel resolutions are common), the stereo algorithm must be tractable at high resolutions[1]. The algorithm must produce good depth maps, both in terms of accurate depth estimates and in terms of localization — edges in the depth map must closely track edges in the image to avoid rendering artifacts. And due to the time and power required to transfer high-resolution images between a mobile device and a server (and because users expect their cameras to be responsive, even without internet access) the stereo algorithm should be tractable even on a mobile platform. In this pa-



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

(b) Our algorithm's disparity and defocused image / subregions

(c) SGM's [14] disparity and defocused image / subregions

Figure 1: Given the 4-megapixel stereo pair in Fig. 1a, we produce the disparity map shown in Fig. 1b and then render a synthetically defocused image. Though our depths are sometimes incorrect, they are incorrect in ways that tend to not matter for the defocus task, and so our defocused images look natural even upon close inspection. Images produced using other stereo algorithms (such as Fig. 1c) tend to have artifacts that render then unacceptable to photographers. The reader is encouraged to zoom in and inspect the images closely.

[1] Of course, one can downsample a stereo pair, produce a low-resolution depth map, and then upsample that depth, but this discards depth information and (unless the upsampling is edge-aware) produces a blurry depth map. It is also possible to produce low-resolution defocused images, but consumers generally prefer high-resolution images.

978-1-4673-6964-0/15/$31.00 ©2015 IEEE

per we present an algorithm that can process a 4-megapixel stereo pair in less than one second and produces higher-quality depth maps for the defocus use-case than existing algorithms. A variant of our stereo algorithm is used as part of the "Lens Blur" feature in the Google Camera app [13].

Of course, there are other ways to produce depth maps besides stereo algorithms. An active depth sensor can be embedded in a mobile device [11], but this requires specialized hardware, consumes lots of power, typically produces low-resolution depth maps, and often only works indoors. Lytro [21] and Pelican [29] capture a light field using a microlens array or a camera array, respectively, and then resample that light field to produce defocused images. These approaches require specialized hardware, and the size of their synthetic apertures is limited by the size of their physical apertures — they cannot easily produce defocused images that are not a subset of the sampled light field. Researchers have investigated depth-recovery from semi-stationary videos [16, 34], but these approaches are difficult to use and fail on moving objects. Our approach requires just two conventional cameras, and our renderings are limited only by the quality of our stereo algorithm and the baseline of the cameras.

Stereo is a well-studied problem, comprehensively surveyed in [25]. Most past work has focused on small images (with some exceptions [26]), and many techniques become slow or intractable at higher resolutions. Our algorithm is tractable at resolutions as high as 64 megapixels, and is 10-100× faster than comparable state-of-the-art algorithms. Some stereo approaches use specialized hardware [12] or GPU-acceleration [19] to improve performance. Our approach's speed is due to algorithm design rather than implementation, making high speeds possible even with our straightforward C++ implementation.

Traditional stereo evaluation metrics favor algorithms that produce accurate metric depth estimates rather than well-localized depth edges, under the assumption that depth is a useful tool for robot navigation or a similar task. This means that algorithms that perform well on standard stereo benchmarks such as Middlebury [25, 26] or KITTI [9] often do not produce satisfactory results when used for defocus rendering applications, as shown in Figure 1. To this end, we present our own benchmark and dataset designed to measure the end-to-end defocus quality of a stereo algorithm. We find that, though our algorithm does not produce good results according to traditional stereo benchmarks, it outperforms the state-of-the-art at this defocus task.

Most stereo algorithms work by assigning a disparity label to each pixel in an image. The core idea of our technique is that we avoid per-pixel inference by leveraging techniques for fast bilateral filtering [1, 5] to "resample" a dense stereo problem from pixel-space into a much smaller "bilateral-space". Bilateral-space is a resampling of pixel-space (or any vector space in which bilateral affinities are computed) such that small, simple blurs between adjacent vertices in bilateral-space are equivalent to large, edge-aware blurs in pixel-space. Instead of performing inference with respect to all pixels in an image, we instead approximate a per-pixel optimization problem in this reduced bilateral-space. Our bilateral-space is defined such that it is cheap to compute, such that general optimization problems can be efficiently embedded into that space, and such that per-pixel depth-labelings we produce after inference have the edge-aware properties that make them useful for our defocus task. Because inference is done in this compact "bilateral-space" instead of pixel-space, our approach is fast and scalable despite solving a global optimization problem with non-local smoothness priors.

We are not the first to exploit edge-aware filtering in the context of stereo, or the similar problem of optical flow. Several optical flow techniques use bilateral filtering to produce edge-aware flow fields [27, 31]. Other techniques take this idea one step further, and use techniques for fast bilateral filtering to produce accelerated flow algorithms [18, 24]. Others have used different non-local edge-aware filters, such as the guided filter [23] and MST-based non-local filters [33]. Fast bilateral filtering has also been used successfully for semantic segmentation [17]. These approaches based on fast bilateral filtering generally perform inference with respect to pixels, and use fast bilateral filtering techniques to speed up a per-pixel filter used during inference. We take this idea a step further, and use the core ideas behind fast bilateral filtering techniques to *resample an optimization problem* into the space in which bilateral filtering is fast, and then solve the optimization problem in that reduced space. In contrast to previous techniques, our use of a bilateral filter allows our optimization problem to be solved *more* efficiently than most approaches based on simple local smoothness, and causes more aggressive bilateral filters (larger filter sizes) to make optimization faster. Though most fast stereo research tends to focus on "local" techniques, our technique is completely global (we solve an optimization problem with respect to every pixel) while being faster than these local techniques. Unlike most "global" stereo techniques, our optimization problem is convex, which make optimization easy and guarantees convergence.

2. Fast Bilateral Filtering

The bilateral filter [28] is an edge-preserving filter, which blurs along edges but not across edges by locally adapting the filter to the image content. A convenient way to think of the bilateral filter is as a normalized matrix-vector multiplication:

$$\mathbf{y} = (A\mathbf{x})/(A\mathbf{1}) \tag{1}$$

Where $A_{i,j}$ is the weight between pixels i and j, $\mathbf{x}$ is a (vectorized) input image, $\mathbf{y}$ is the filtered output image, $/$ is element-wise division, and $\mathbf{1}$ is a vector of all ones. Formally, A is defined as:

$$A_{i,j} = \exp\left(-\frac{\|[x_i,y_i]-[x_j,y_j]\|^2}{2\sigma_{xy}^2} - \frac{\|[r_i,g_i,b_i]-[r_j,g_j,b_j]\|^2}{2\sigma_{rgb}^2}\right) \quad (2)$$

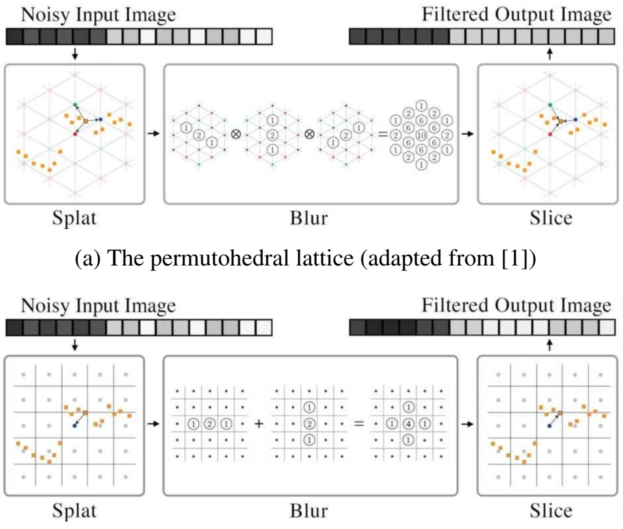
Where the σ_{xy} and σ_{rgb} parameters control the spatial and range bandwidths of the filter, respectively.

There have been many proposed techniques for speeding up bilateral filtering [1, 2, 5, 8]. Two of these techniques, the bilateral grid [5] and the permutohedral lattice [1] express bilateral filtering as a "splat/blur/slice" procedure: pixel values are "splatted" onto a small set of vertices in a grid or lattice (a soft histogramming operation), then those vertex values are blurred, and then the filtered values for each pixel are produced via a "slice" (an interpolation) of the blurred vertex values. Though neither paper presents itself in this fashion, it is convenient to describe these techniques as approximating A with a matrix factorization:

$$A \approx S^{\mathrm{T}} \bar{B} S \quad (3)$$

Where multiplication by S is the "splat", multiplication by $\bar{B}$ is the "blur", and multiplication by S^{T} is the "slice". Even though A may be large and dense, by construction S is a short, wide, and sparse matrix, and $\bar{B}$ is small and sparse. $\bar{B}$ is actually a product of several sparse matrices (though we will treat it as a single matrix for convenience). See the supplementary material for visualizations of these bilateral representations as matrix decompositions. While the naive filtering of Eq. 1 is often intractably slow, this factorization allows for fast filtering:

$$\mathbf{y} = \left(S^{\mathrm{T}}\left(\bar{B}\left(S\mathbf{x}\right)\right)\right) / \left(S^{\mathrm{T}}\left(\bar{B}\left(S\mathbf{1}\right)\right)\right) \quad (4)$$



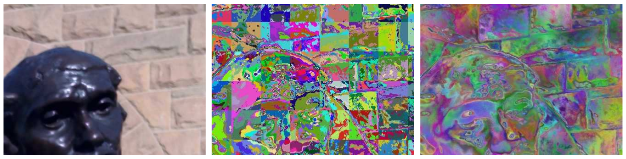
(a) The permutohedral lattice (adapted from [1])

(b) The simplified bilateral grid

Figure 2: The two bilateral representations we use in this paper, here shown filtering a toy one-dimensional grayscale image of a step-edge. This toy image corresponds to a 2D space visualized here (x = pixel location, y = pixel value) while in the paper we use RGB images, which corresponds to a 5D space (XYRGB). The lattice (Fig 2a) uses barycentric interpolation to map pixels to vertices and requires $d+1$ blurring operations, where d is the dimensionality of the space. Our simplified bilateral grid (Fig 2b) uses nearest-neighbor interpolation and requires d blurring operations which are summed rather than done in sequence. The grid is cheaper to construct and to use than the lattice, but the use of hard assignments means that the filtered output often has blocky piecewise-constant artifacts.

We will refer to splat/blur/slice techniques like the bilateral grid or the permutohedral lattice as "bilateral representations". In this work we will demonstrate how to take a certain class of global optimization problems phrased in terms of pixels and a bilateral affinity measure and embed that problem into a bilateral representation. We will focus on two bilateral representations: the permutohedral lattice, and a simplified variant of the bilateral grid, though other representations are possible. Both representations work by resampling a signal from pixel-space into "bilateral-space", where small blurs between adjacent vertices in bilateral-space are equivalent to large, non-local, edge-aware blurs in pixel-space. The permutohedral lattice accurately approximates A by lifting points into a higher dimensional space and using barycentric interpolation [1]. The simplified bilateral grid is based on the bilateral grid of [5], where interpolation has been replaced by "hard" assignment and where the "blur" is a sum of filters rather than a product. A similar data structure has been used for mean-shift image segmentation [6, 22]. Our grid is cheap to construct and to use but produces a worse approximation of A, while the permutohedral lattice is expensive but models A accurately. For a better intuition for these bilateral representations see Figures 2 and 3, or the supplemental material.

The efficiency of filtering with a bilateral representation is due to several factors: the sparsity of the splat/slice matrix, the sparsity of the blur matrices, and the fact that the number of vertices is generally much smaller than the number of pixels. For a 4-megapixel image with $\sigma_{rgb} = 8$ and $\sigma_{xy} = 32$ (the values we use in all of our experiments), the bilateral representation (which adapts its size according to image content) usually contains 40-200 thousand vertices — a 10-200× reduction in the number of variables. As these σ's grow larger, the number of vertices needed in the bilateral representation grows smaller. Our algorithm works by "splatting" a per-pixel global stereo algorithm into this compact bilateral-space, solving a convex optimization



(a) image (b) grid (c) lattice

Figure 3: In Fig 3b we visualize the simplified bilateral grid representation of the image in Fig 3a, where each vertex has been assigned a random color. Each color in this visualization corresponds to a variable in our optimization problem. In Fig 3c we have that same visualization for the permutohedral lattice, which "softly" models each pixel as a weighted combination of its surrounding vertices instead of using "hard" assignment.

problem in bilateral-space through repeated "blurring", and finally "slicing" out a per-pixel disparity map upon convergence. This is much more efficient than performing inference in the much-larger pixel-space, which would be much more memory-intensive in addition to requiring repeated splatting and slicing.

3. Problem Formulation

We will now construct a global stereo optimization problem, where we solve for a disparity p_i of every pixel in the image subject to some smoothness constraint:

$$\underset{\mathbf{p}}{\text{minimize}} \quad \frac{1}{2}\sum_i \sum_j \hat{A}_{i,j}\left(p_i - p_j\right)^2 + \lambda \sum_i f_i(p_i) \quad (5)$$

As is common, we have a smoothness term and a data term which are balanced with some multiplier λ. $\hat{A}_{i,j}$ is the affinity between pixels i and j, where a stronger affinity causes the disparities at pixels i and j to be more similar in a least-squares sense. $\hat{A}$ is a bistochastic version of A (all rows and columns sum to 1), which is easier to manipulate and has desirable properties when used as a filter [20]. Each $f_i(\cdot)$ is a convex cost function for pixel i which penalizes different values of p_i. In stereo nomenclature, the set of all $f_i(\cdot)$ collectively form the "cost volume" of the stereo pair. Given that $\hat{A}$ is bistochastic and symmetric, our problem can be rewritten as follows:

$$\underset{\mathbf{p}}{\text{minimize}} \quad \mathbf{p}^{\mathrm{T}}\left(I - \hat{A}\right)\mathbf{p} + \lambda \sum_i f_i(p_i) \quad (6)$$

See the supplemental material for a derivation of this equivalence. We will leverage our insight from Eq. 3 that A can be expressed as a matrix factorization by performing a variable substitution which reformulates our problem in terms of vertices instead of pixels:

$$\mathbf{p} = S^{\mathrm{T}}\mathbf{v} \quad (7)$$

Where $\mathbf{v}$ is a vector of disparity values for each bilateral-space vertex. This reparametrization not only reduces the dimensionality of the space, but also allows us to reduce the cost of evaluating the loss function by projecting our loss into bilateral-space. We then solve our optimization problem in bilateral-space to recover $\mathbf{v}^*$, and from that we produce a per-pixel disparity map $\mathbf{p}^* = S^{\mathrm{T}}\mathbf{v}^*$.

This reparametrization means that not all solutions are expressible: each pixel's disparity must be an interpolated function (a "slice") of the disparities of the bilateral-space vertices in that pixel's simplex, heavily constraining our output space. Our output disparity can only have sharp edges when there are corresponding edges in the input image, meaning that we *cannot* produce depth maps that are not smooth in a bilateral sense. Effectively, this reparametrization means that our output disparity map must resemble the output of a bilateral filter. This is consistent with our target application: depth discontinuities are only noticeable in defocused images where there are image edges, and a blurred flat region looks identical to a non-blurred flat region. This decision is also consistent with the inherent limits of stereo: depth discontinuities that are not visible in the input images are difficult to recover. This analysis breaks down slightly for the simplified bilateral grid which, due to the use of nearest-neighbor assignment instead of interpolation, can produce blocky artifacts in "sliced" disparity images — though as we will demonstrate, this can be ameliorated with simple post-processing.

Let us rewrite the smoothness term of the per-pixel optimization problem in Eq. 6 in bilateral-space:

$$\mathbf{p}^{\mathrm{T}}\left(I - \hat{A}\right)\mathbf{p} \approx \mathbf{v}^{\mathrm{T}}\left(C_s - C_n \bar{B} C_n\right)\mathbf{v} \quad (8)$$

Where C_s a diagonal matrix whose diagonal is equal to the row-sum of S ($C_s = \operatorname{diag}(S\mathbf{1})$), and C_n is a diagonal matrix which is constructed such that $C_n \bar{B} C_n \mathbf{1} = C_s \mathbf{1}$. See the supplemental material for the derivation of this equivalence, which also necessarily describes how we deal with the bistochasticization of $\hat{A}$.

Now let us rewrite the data term of our per-pixel optimization problem in bilateral-space. Remember that in pixel-space, our loss is a per-pixel function $f_i(p_i)$, which we will assume to be a lookup table that we will access with linear interpolation such that optimization is continuous with respect to $\mathbf{p}$. We will construct a set of bilateral-space lookup tables by "splatting" our per-pixel lookup tables, giving us a bilateral-space cost volume. Let us define $g_j(v_j)$ as a lookup table for vertex j, as follows:

$$\forall x \quad g_j(x) = \sum_{(w,i)\in S_j} w f_i(x) \quad (9)$$

Where S_j is the row of the splat matrix that corresponds to vertex j, which consists of a set of pixel indices i and weights w.

Combining our smoothness and data terms, we have our bilateral-space optimization problem:

$$\underset{\mathbf{v}}{\text{minimize}} \quad \mathbf{v}^{\mathrm{T}} \left(C_s - C_n \bar{B} C_n \right) \mathbf{v} + \lambda \sum_j g_j(v_j) \qquad (10)$$

This bilateral-space optimization problem is approximately equivalent to our pixel-space optimization problem of Eq. 5, where the approximation is due to some differences required for efficient bistochastization.

4. An Efficient Stereo Data Term

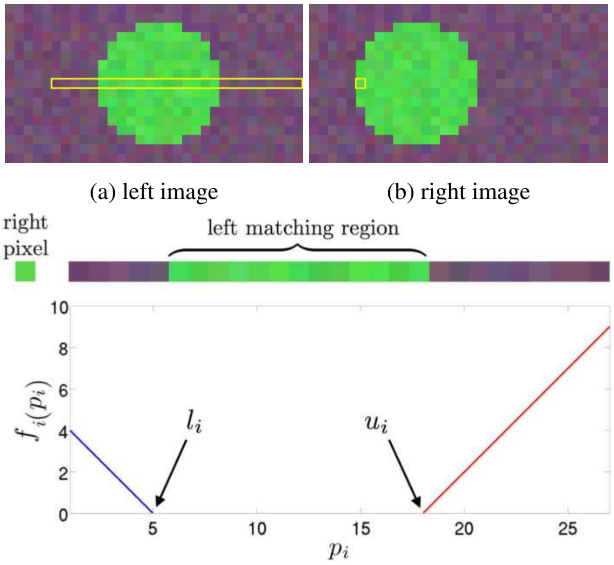
Our framework as we have described it is agnostic to the choice of the data term $f_i(p_i)$, as long as it is a convex lookup table. Naively, we could choose a standard cost such as sum-of-absolute-values and "splat" that pixel-space cost volume into bilateral-space as shown in Eq. 9. But this is inefficient: it requires computing an entire $N \times D$-sized cost volume (where N is the number of pixels and D is the number of disparities) which is both slow and memory-intensive, and it requires D "splat" operations. We will therefore use a particular data term that retains convexity and allows the bilateral-space cost volume to be quickly computed with $\mathcal{O}(N)$ complexity in time and space.

We will parametrize our per-pixel cost as follows:

$$f_i(p_i) = \max(0, p_i - u_i) + \max(0, l_i - p_i) \qquad (11)$$

This is the sum of two hinge-losses for each pixel, which penalizes values of p_i that lie outside of the range $[l_i, u_i]$. Similarly-convex costs have been used in stereo algorithms [35], though usually alongside progressive relaxation, which we do not attempt. An instance of this cost for a single pixel is visualized in Figure 4. Details of how these upper and lower bounds are computed can be found in the supplemental material, though our algorithm is agnostic to how these bounds are computed.

This loss is a coarse approximation to more descriptive losses such as sum-of-absolute-differences. For example, if we observe a match at two disparities, this loss assumes that all disparities between them are also viable matches. But this loss has many desirable properties: 1) it is compact, as we need only store two numbers per pixel, 2) it is convex, 3) it can be evaluated efficiently, and 4) it can be embedded into bilateral-space efficiently, as we will demonstrate. First, recall that the derivative of a hinge is a Heaviside function, that the derivative of a Heaviside function is a delta function, and that integration is a linear operator. With this knowledge, we see that it is possible to compute a weighted sum of hinge losses by computing a weighted sum of delta functions, and then integrating that sum twice. We can use this insight to construct each lookup table of our bilateral-space cost volume $g_j(\cdot)$ as follows:


(a) left image (b) right image

(c) the loss applied to the right pixel.

Figure 4: A toy stereo pair, for which we want to compute disparity for the right image. We have highlighted in yellow a pixel of interest in the right image and the corresponding scanline in the left image which may match that pixel (assuming rectification). We identify the range of the left scanline which may match the right pixel, as described in the supplemental material. The upper and lower bounds of that range $[l_i, u_i]$ completely define the two hinge-losses we impose on the right pixel.

$$g_j(v_j) = \sum_{y=0}^{v_j} \sum_{x=0}^{y} u''_j(x) + \sum_{y=v_j}^{D} \sum_{x=y}^{\infty} l''_j(x) \qquad (12)$$

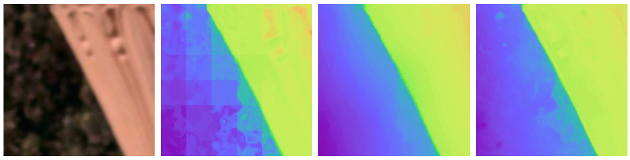
$$u''_j(x) = \sum_{(w,i) \in S_j} w \left[x = (u_i + 1) \right] \qquad (13)$$

$$l''_j(x) = \sum_{(w,i) \in S_j} w \left[x = (l_i - 1) \right] \qquad (14)$$

Where the square brackets are indicator functions. We construct each $g_j(\cdot)$ as the sum of two double-cumulative-sums of histograms of upper and lower bounds, producing a lookup table of losses. Computing this cost volume requires $\mathcal{O}(N)$ operations to splat the bounds and $\mathcal{O}(MD)$ operations to integrate the lookup tables (where M is the number of vertices), which in practice is 20-200× faster than the $\mathcal{O}(ND)$ operations required by the naive approach in Eq. 9.

5. Optimization

Now that we have projected our optimization problem into bilateral-space, we can solve it. Because our problem is



(a) image (b) grid (c) grid+DT (d) lattice

Figure 5: Our grid-based model introduces blocky artifacts in our output (Fig. 5b), which we remove by filtering with respect to the input image (Fig. 5a) using a domain transform [7], causing our output (Fig. 5c) to more-closely resemble that of our lattice-based model (Fig. 5d).

convex many optimization techniques are viable. We use a multiscale variant of L-BFGS. This requires that we be able to efficiently compute our loss function and its gradient. To review, our loss function is:

$$\mathrm{loss}(\mathbf{v}) = \mathbf{v}^{\mathrm{T}} \left(C_s - C_n \bar{B} C_n \right) \mathbf{v} + \quad (15)$$
$$\lambda \sum_j \left((\lceil v_j \rceil - v_j) g_j(\lfloor v_j \rfloor) + (v_j - \lfloor v_j \rfloor) g_j(\lceil v_j \rceil) \right)$$

This is simply Eq. 10 with linear interpolation written out in full. The gradient of this loss function is:

$$\nabla \mathrm{loss}(\mathbf{v}) = 2 \left(C_s - C_n \bar{B} C_n \right) \mathbf{v} + \quad (16)$$
$$\lambda \left[g_1(\lceil v_1 \rceil) - g_1\left(\lfloor v_1 \rfloor\right); \ldots ; g_M(\lceil v_M \rceil) - g_M\left(\lfloor v_M \rfloor\right) \right]$$

The loss and gradient are straightforward to compute simultaneously in large part because $C_n \bar{B} C_n$ is symmetric, which is due to the fact that $\hat{A}$ is bistochastic.

With our loss and gradient defined, we use L-BFGS (through the open-source optimization package Ceres [4]) to solve for a bilateral-space solution $\mathbf{v}^*$, from which we can "slice" a pixel-space solution $\mathbf{p}^*$:

$$\mathbf{v}^* = \arg\min_{\mathbf{v}} \mathrm{loss}(\mathbf{v}) \quad (17)$$
$$\mathbf{p}^* = S^{\mathrm{T}} \mathbf{v}^* \quad (18)$$

Optimization speed can be improved using multiscale techniques, as described in the supplemental material. In all experiments we terminate after 25 iterations of L-BFGS, which we found sufficient.

The nearest-neighbor assignment used in our simplified bilateral grid creates blocky artifacts in flat regions of the input image. To remove these artifacts, we post-process these disparity maps using the domain transform (DT) [7], a fast edge-aware filtering technique. See Figure 5 for a visualization of the output of our simplified grid model before and after the domain transform.

6. Results

Evaluating a stereo algorithm for the defocus task is challenging. Though there are many stereo datasets [9, 25, 26], these benchmarks primarily evaluate the metric accuracy of a disparity map (that is, how accurately objects are localized along the z-axis, rather than in x or y), under the assumption that stereo is most useful for knowing exactly how far away an object is, rather than knowing the exact shape of that object. We have found that algorithms that perform well according to this traditional metric often perform poorly in terms of defocus rendering quality, primarily because of mis-aligned depth edges. We therefore use a complete end-to-end evaluation of the quality of defocused renderings to evaluate each stereo algorithm, with both a quantitative benchmark and a user study.

In our experiments we evaluated against the top-performing stereo techniques with available code: LIBELAS [10], SGM (the OpenCV implementation implementation of [14], the top technique on V3 of the Middlebury stereo dataset [26] as of 9/2014), SPS-StFl [32] (the top technique on the KITTI stereo benchmark [9] as of 9/2014), LPS [26] (a top-performing technique on Middlebury V3 meant to perform well on high-resolution images), and CostFilter [23][2] (a technique which aims to be fast and edge-aware, like ours). We also benchmark against a baseline of our own creation: "SAD-lattice", a sum-of-absolute-values stereo algorithm where the cost volume is filtered with $\hat{A}$ before taking the $\arg\min$, which demonstrates that simply using a bilateral filter is not the same as working in bilateral-space. We tried to benchmark against [33], but the code ran out of memory on 1-megapixel images and crashed. For each baseline we include a variant in which disparity is post-processed with the domain transform. We did not modify the hyperparameters of any baseline, and we tuned our own models against the Middlebury training set [25]. This seems to be a fair evaluation, as we use the same training set as other techniques and use our own benchmark only as a test set.

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 a "true" shallow-depth-of-field image generated from a light field. We could instead render synthetically defocused images using the ground-truth disparity maps of a standard stereo dataset, but even "ground-truth" depth maps tend to have rough edges and missing pixels. Most importantly, such an approach would presuppose 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. This benchmark is constructed from the Stanford Light Field Archive [3], which allows us to produce rectified stereo pairs and ground-truth defocused images. See

[2] The authors of [23] only released a Matlab implementation which took 10-20 minutes per megapixel, so we omitted its runtime from our experiments.

Method	time (s)	pixel^4	pixel$^\infty$	patch^4	patch$^\infty$	grad^4	grad$^\infty$	dssim^4	dssim$^\infty$	avg.
Ours(grid)	0.30	1.177	0.443	1.315	0.311	0.421	0.234	0.756	0.245	0.500
Ours(grid)+DT	0.32	1.195	0.451	1.305	0.308	0.445	0.230	0.740	0.240	0.500
Ours(lattice)	4.32	1.243	0.448	1.315	0.305	0.440	0.211	0.737	0.252	0.499
LIBELAS[10]	1.91	1.297	0.471	1.302	0.319	0.488	0.269	0.788	0.278	0.540
SGM[14]	2.45	1.342	0.501	1.363	0.336	0.518	0.273	0.897	0.298	0.573
CostFilter[23]	-	1.429	0.523	1.414	0.352	0.584	0.313	0.872	0.299	0.604
SPS-StFl[32]	13.47	1.437	0.541	1.375	0.340	0.578	0.304	0.859	0.291	0.596
LPS[26]	-	1.214	0.454	1.303	0.312	0.478	0.262	0.821	0.287	0.534
SAD-lattice	8.66	1.331	0.440	1.454	0.324	0.465	0.269	0.891	0.289	0.554
LIBELAS+DT	1.94	1.236	0.463	1.333	0.321	0.457	0.263	0.737	0.244	0.519
SGM+DT	2.47	1.220	0.462	1.321	0.310	0.481	0.264	0.739	0.257	0.523
CostFilter+DT	-	1.273	0.488	1.360	0.322	0.487	0.273	0.768	0.256	0.539
SPS-StFl+DT	13.49	1.281	0.483	1.327	0.318	0.502	0.271	0.730	0.249	0.532
SAD-lattice+DT	8.71	1.231	0.451	1.377	0.327	0.484	0.261	0.776	0.251	0.529
LPS+DT	-	1.206	0.451	1.316	0.315	0.473	0.258	0.722	0.246	0.514

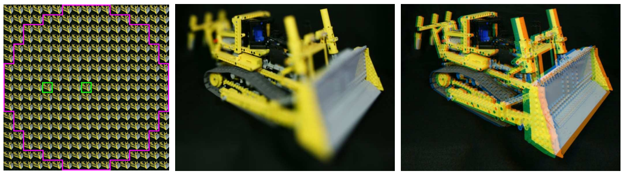
Table 1: Results on our light field benchmark. For each metric the best-performing technique is red and the second-best is yellow. Our approaches outperform the baselines, even those post-processed with the domain transform, and our grid approach is substantially faster.

Method	User						mean ± std.
	1	2	3	4	5	6	
Ours (grid) + DT	175	185	162	134	165	145	161.0 ± 18.8
LIBELAS[10]	9	5	10	8	16	8	9.3 ± 3.7
SGM[14]	19	9	18	27	15	21	18.2 ± 6.0
CostFilter[23]	15	12	13	24	23	11	16.3 ± 5.7
SPS − StFl[32]	22	31	29	35	19	54	31.7 ± 12.4
SAD − lattice	24	22	32	36	26	25	27.5 ± 5.4

Table 2: The results of our user study. We recorded the number of times that the output of each algorithm was preferred over the others, for 264 cropped image regions. Each user's preferences are shown in each column, with the average (mean and standard deviation) shown at the end. For each user the most-preferred technique is highlighted in red, and the second-most-preferred is in yellow.

Figure 6 for a visualization of a light field and the corresponding stereo pair and shallow-depth-of-field image that we extract from it. Some cropped defocused images rendered using depth maps from our technique and the baseline techniques can be seen in Figure 7. As shown in Table 1, our techniques outperform all baselines (even those whose output has been post-processed), and our grid-based technique is faster than the fastest baseline. The lattice-based model produces higher-quality results, though post-processing our grid-based model increases its quality to the level of the lattice-based model while being much faster. Details of this benchmark can be found in the supplemental material.

To perform our user study we acquired a large dataset of images taken from our own stereo camera, which consists of two 4-megapixel cellphone cameras spaced 10mm apart. Our stereo rig was calibrated using standard epipolar geometry techniques to produce rectified image pairs. For each pair we ran our own algorithm (the post-processed grid version) and the baseline algorithms. One stereo pair is shown



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

Figure 6: In Fig. 6a 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. 6b) are circled in magenta, and the two images we took to form our stereo pair (Fig. 6c, visualized as a red-blue anaglyph) are circled in green.

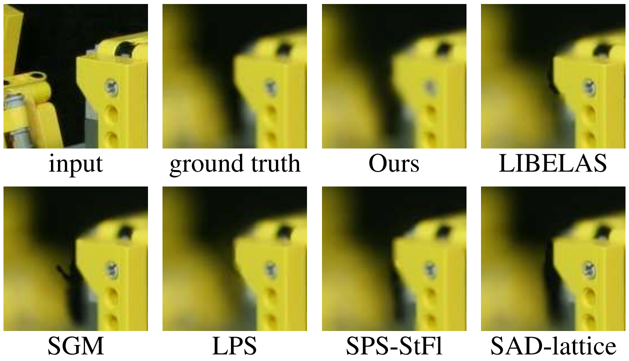


Figure 7: A set of cropped regions from: one image from our input stereo pair, a ground-truth shallow-depth-of-field image rendered from a light field, and the rendering produced using the disparity maps generated from our algorithm and the baseline stereo algorithms. Ours looks natural and similar to the ground-truth, while the others tend to have false edges and artifacts.

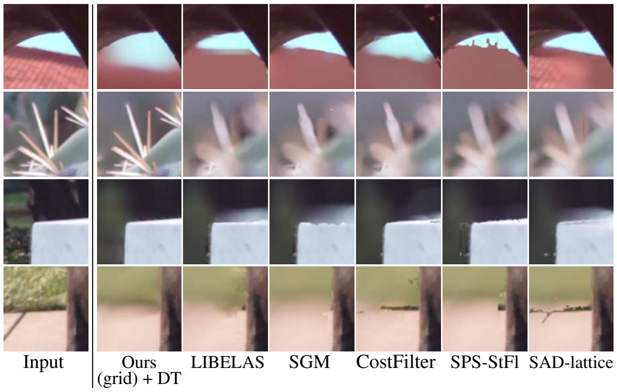


Figure 8: Some images from our user study. The user was presented with six images produced with six different stereo algorithms, and was asked to select the image that they preferred.

in Figure 1, and many more can be seen in the supplemental material.

For our user study, we had six people of varying levels of photographic expertise who were not involved in this project compare the output of our algorithm and the baseline algorithms. We repeatedly presented each user a cropped subset of a scene showing the renderings produced with each algorithm, and asked the user to select the rendering which they preferred. Examples of these trials can be seen in Figure 8. The users were each shown 264 cropped renderings in total: 4 cropped regions from 66 images, automatically cropped to the location where the rendering disagreed the most (the window with the maximum variance across the gradients-magnitudes of the renderings). The algorithms were presented in a random order on each presentation to avoid bias. The results can be seen in Table 2. The preferences of the users are surprisingly consistent, with the output of our algorithm being chosen by all participants roughly $5\times$ more often than the next most preferred algorithm (SPS-StFl). Even more surprisingly, the users strongly agreed with one another, with the median intra-user agreement on votes being 51%.

To evaluate the speed of our approach, we took 20 stereo pairs from our dataset and cropped or tiled them to produce an expanded set of images with resolutions from 0.25 to 64 megapixels (the maximum disparity remained unchanged). By running each algorithm on that set of images we can see how speed varies with respect to image resolution, as shown in Figure 9. Because code was not available for the LPS baseline [26] we used its performance on our light field dataset[3] to extrapolate runtimes at different scales, which roughly agree with [26]: LPS is slower than LIBELAS and faster than SGM. Our grid-based algorithm is roughly $10 - 20\times$ faster than LIBELAS, the next fastest algorithm, with our speed improvement increasing with the size of the image. All of our reported times exclude the disk I/O time require to read images and write depth maps.

7. Conclusions

We have presented a technique for computing a disparity map from a stereo pair, for the purpose of synthetically defocusing an image. Our technique produces better looking defocused images than comparable state-of-the-art techniques, while being 10-100× faster. Our algorithm is built on a novel technique for embedding optimization problems into the "splat-blur-slice" data structures that are traditionally used for fast bilateral filtering. We have presented a novel benchmark for evaluating stereo algorithms with respect to the defocus task as well as a user study, which both demonstrate that our technique produces higher-quality results than existing stereo algorithms.

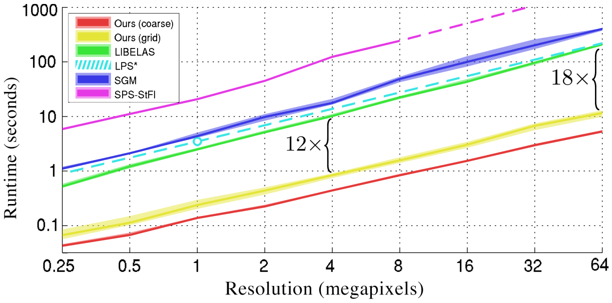


Figure 9: The runtime (median and 25-75 percentiles) for our algorithm and baselines, shown as a function of image resolution. "Ours (coarse)" is our grid model where we have doubled σ_{xy} and σ_{rgb} to show that runtime depends on the scale of the bilateral representation. The SPS-StFl baseline crashed at resolutions > 8 megapixels, so its times are extrapolated. Note that the LPS times are from a different computer, and are extrapolated from 1-megapixel runtimes.

Though our technique is fast and accurate, it comes with some caveats. Our emphasis on accurately localizing edges means that our algorithm is less accurate at estimating metrically correct depths — though this tradeoff seems worthwhile when targeting defocus applications. Our approach limits what disparity maps we can produce: if two pixels have similar color and positions, then the final disparity of those pixels must be similar. This is occasionally the source of errors when a foreground object resembles a nearby background object, though if the foreground and background are exactly the same color then these mistakes are generally not objectionable — a blurry flat patch looks identical to a un-blurred flat patch.

Our approach has several benefits besides speed and defocus quality. The memory required is low, as we do not construct a complete cost-volume and instead manipulate an implicit cost-volume. Because our algorithm is a global technique, information propagates across the entire image during inference, unlike most local stereo algorithms. Our loss is convex and therefore easy to optimize and without local minima, and our optimization approximately corrsponds to a straightforward per-pixel global stereo algorithm. Our technique has only three intuitive hyperparameters which can be tuned to trade quality for speed. Our output disparity tightly follows edges in the input image, making our depth maps well-suited to many graphics applications. Though we focused on two specific bilateral representations, our technique is agnostic to the specific "splat-blur-slice" approach being used and may generalize to future advances in fast bilateral filtering.

[3] The authors of [26] kindly ran their algorithm on our light field dataset to produce disparity maps and runtimes for each image

References

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

[2] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. Gaussian kd-trees for fast high-dimensional filtering. *SIGGRAPH*, 2009.

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

[4] S. Agarwal, K. Mierle, and Others. Ceres solver. `http://ceres-solver.org`.

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

[6] C. M. Christoudiadl, B. Georgescu, and P. Meer. Synergism in low level vision. *ICPR*, 2002.

[7] E. S. L. Gastal and M. M. Oliveira. Domain transform for edge-aware image and video processing. *SIGGRAPH*, 2011.

[8] E. S. L. Gastal and M. M. Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *SIGGRAPH*, 2012.

[9] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *CVPR*, 2012.

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

[11] Google. Project tango. `www.google.com/atap/projecttango/`.

[12] P. Grey. Stereo vision products. `www.ptgrey.com/products/stereo.asp`.

[13] C. Hernández. Lens blur in the new google camera app. `googleresearch.blogspot.com/2014/04/lens-blur-in-new-google-camera-app.html`.

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

[15] HTC. Htc one-m8. `www.htc.com/us/smartphones/htc-one-m8/camera`.

[16] N. Joshi and C. L. Zitnick. Micro-baseline stereo. Technical Report MSR-TR-2014-73, Microsoft Research, May 2014.

[17] P. Krähenbühl and V. Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. *NIPS*, 2011.

[18] P. Krähenbühl and V. Koltun. Efficient nonlocal regularization for optical flow. *ECCV*, 2012.

[19] S. Mattoccia, M. Viti, and F. Ries. Near real-time fast bilateral stereo on the gpu. *ECVW11*, 2011.

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

[21] R. Ng. *Digital Light Field Photography*. PhD thesis, Stanford University, 2006.

[22] S. Paris and F. Durand. A topological approach to hierarchical segmentation using mean shift. *CVPR*, 2007.

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

[24] C. Richardt, D. Orr, I. Davies, A. Criminisi, and N. A. Dodgson. Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid. *ECCV*, 2010.

[25] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

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

[27] M. W. Tao, J. Bai, P. Kohli, and S. Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. *Eurographics*, 2012.

[28] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. *ICCV*, 1998.

[29] K. Venkataraman, D. Lelescu, J. Duparré, A. McMahon, G. Molina, P. Chatterjee, R. Mullis, and S. Nayar. Picam: An ultra-thin high performance monolithic camera array. *SIGGRAPH Asia*, 2013.

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

[31] J. Xiao, H. Cheng, H. Sawhney, C. Rao, and M. Isnardi. Bilateral filtering-based optical flow estimation with occlusion detection. *ECCV*, 2006.

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

[33] Q. Yang. A non-local cost aggregation method for stereo matching. *CVPR*, 2012.

[34] F. Yu and D. Gallup. 3d reconstruction from accidental motion. *CVPR*, 2014.

[35] S. Zhu, L. Zhang, and H. Jin. A locally linear regression model for boundary preserving regularization in stereo matching. *ECCV*, 2012.