Hierarchical Upsampling for Fast Image-Based Depth Estimation

Blake Foster and Rui Wang

University of Massachusetts Amherst

Abstract

We propose a hierarchical upsampling method for dense image-based depth estimation. Given a set of high-resolution images, we first apply multiview stereopsis on downsampled images to obtain a sparse point cloud in a few minutes. Our goal is then to combine the original images with the sparse point cloud to recover a dense set of points. We select a set of reference views, and use joint bilateral upsampling to estimate per-pixel depths in each view. We then merge the results using estimated confidence values. We greatly improve the computation speed and robustness by upsampling hierarchically. Our GPU implementation can process high-resolution images at more than 15 frames per second. We can convert the sparse point cloud into a full scene reconstruction composed of millions of points in just a few seconds. Our results are robust, and can be used directly in many applications, including smooth surface reconstruction and depth-of-field simulation.

Categories and Subject Descriptors (according to ACM CCS): I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Depth cues I.4.3 [Image Processing and Computer Vision]: Enhancement—Geometric correction I.3.1 [Computer Graphics]: Hardware architecture—Parallel processing I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

While many stereo vision algorithms can quickly and robustly estimate sparse geometry from sets of photos, dense reconstruction remains a time-consuming and memory-intensive process. In this paper, we introduce an efficient algorithm for rapid 3D estimation from images. Given a set of high-resolution images, we first run multiview stereopsis [FP09] at a low resolution to obtain a sparse point cloud in a few minutes. This typically reconstructs only $3 \sim 5\%$ of the pixels in any input image. Our goal is then to combine the original high-resolution images with the sparse point cloud to estimate a dense reconstruction. Our main contribution is a hierarchical algorithm that exploits joint bilateral upsampling [KCLU07] to recover dense points.

Our algorithm first computes sparse depth maps for a small set of reference views by reprojecting the initial 3D points. We then use joint bilateral upsampling to estimate a dense depth map for each reference view. We account for visibility efficiently with a small modification to the standard bilateral filter weights. Because the initial depth maps are sparse, directly upsampling would be expensive due to the necessarily large search radius. We improve the speed and robustness of this step with a hierarchical approach. We first downsample each reference view to a lower resolution, decreasing the sparsity of the projected points. Then we hierarchically upsample the low-resolution depth maps to the resolution of the original images, filling in holes at each step. Finally, we merge the results from all the reference views, guided by estimated confidence values. This provides a dense reconstruction of the entire scene, composed of up to several million points.

Once we have the initial set of points, our GPU implementation can process more than 15 images per second. In a typical scene, we require no more than a few minutes to obtain the initial point cloud, and few seconds for upsampling. We show that our results are robust, even with only two input images. Finally, we apply our algorithm to manipulate the depth of field in focused images. Given only a small number of input images, we construct a depth map in a reference view, and then apply a depth-dependent blur to simulate a narrow depth of field.

2. Related Work

Image-Based Reconstruction. Image-based reconstruction algorithms have achieved remarkable accuracy in recent years. Seitz et al. [SCD*06b] provide a detailed comparison and evaluation of the state-of-the-art. Most techniques assume calibrated input images. Calibration is typically done with structure from motion (SfM) [HZ04].

Goesele et al. [GSC*07] obtain dense depth maps from community photo collections by intelligently picking images to match. Like our method, their system attempts to produce dense geometry from a sparse set of points. Patch-based multiview stereo (PMVS) [FP09] partitions images into grids and computes dense point clouds by enforcing local photometric consistency and global visibility constraints. The point clouds are often sufficiently dense for point-based rendering. The points are also suitable for smooth mesh computation via Poisson surface reconstruction [KBH06]. These methods are quite accurate, but producing dense point clouds can take hours. We gain a substantial speed advantage over these methods by running multiview stereo on downsampled images to compute sparse points, which we then upsample to produce dense geometry. Multiview stereo usually runs in just a few minutes on our downsampled images, and upsampling requires only a few seconds.

Yang and Pollefeys [YP03] use a GPU-based plane sweeping algorithm [Col96] to estimate dense depth maps from a set of stationary video cameras. Gallup et al. [GFM*07] use multiple plane orientations to improve the performance of this approach on non-frontoparallel surfaces. Merrell et al. [MAW*07] introduced a realtime system for merging rough depth maps obtained via plain-sweeping. Given sufficiently many views captured in video, their system produces excellent results.

Diebel and Thrun [DT05] upsample depth maps by modeling pixel colors and depths with a Markov random field. Their system requires dense geometry sampled at a low resolution.

Recently, Pan et al. [PRD09] presented a modeling system that reconstructs an object interactively as the user rotates it front of a Webcam. Their system works well for small objects with simple geometry, but is not suitable for larger scenes captured from discrete camera locations.

Bilateral Filter and Upsampling. The bilateral filter [TM98] is an efficient nonlinear filter that blurs images while preserving edges and features. The filter weighs each pixel's contribution to its neighbors by combining a spatial kernel and a range kernel. Durand and Dorsey [DD02] applied a fast approximation of the bilateral filter to HDR tone mapping. Their results were later improved in [Wei06, PD09]. Yoon and Kweon [YK06] use a metric similar to a bilateral filter to aggregate window-based support weights in stereo matching.

Joint bilateral upsampling [KCLU07] produces a high-resolution depth map from low resolution range data, while preserving sharp edges in a corresponding high-resolution image. The input must be a dense depth map, albeit at a lower resolution. Chan et al. [CBTT08] apply joint bilateral upsampling to depth maps obtained from active 3D range sensors. Their main focus is on denoising range data. Dolson et al. [DBPT10] use a high dimensional joint bilateral filter to upsample depth measurements obtained from a laser scanner in a dynamic scene. They also achieve real-time processing speed with their GPU implementation. While these two methods are similar to ours, they rely on a laser range device mounted on a video camera to obtain single-view depth maps. Their depth maps are relatively dense, and implicitly contain visibility information due to the colocation of the camera and the rangefinder. In contrast, our hierarchical upsampling method works with very sparse points obtained via stereo vision, and does not require visibility information.

Digital Refocusing. Depth of field manipulation is a popular artistic effect in photography. Digital refocusing techniques allow a photographer to modify the depth of field after an image has been captured. Bando and Nishita [BN07] introduced a single-image refocusing system. Their algorithm first deblurs the input image with a spatially-varying kernel, and then applies a depth-dependent blur to the sharp image. They implicitly compute depth values from the defocused input image, guided by some user interaction. Zhang and Cham [ZC09] take a similar approach, but they use a probabilistic framework to estimate the defocus map without user interaction. Rajagopalan and Mudenagudi [RM04] use depth from defocus in conjunction with stereo vision to estimate dense depthmaps and restore sharp images.

Our refocusing system differs from these methods in that we do not rely on depth from defocus or single view depth inference techniques. Instead, we use our algorithm to generate per-pixel depths from a small number of images, and use these depth values to compute a depth-dependent blur. While we could possibly use multiview stereo to obtain the per-pixel depths, our approach is faster and guarantees a depth at every pixel.

3. Algorithms

3.1. Multiview Stereopsis

Our algorithm operates on a set of photos taken from calibrated cameras. In the absence of calibration information, we use Bundler [SSS06], an open source tool for robust structure from motion. We compute our initial set of 3D points with patch-based multiview stereo (PMVS) [FP09]. We configure the PMVS software to downsample the images to around 300 pixels on the larger dimension. The initial reconstruction typically takes only 1 to 2 minutes for a dataset with 50 to 100 images. The resulting point cloud provides depth values for about 3 to 5% of the pixels in each of the original images image.

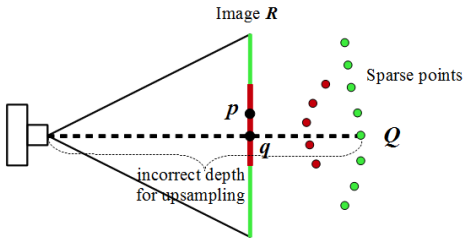

Figure 1: *An image showing a red object in front of a green object. We wish to recover the depth of pixel p, using color to guide the interpolation. A neighbor pixel q has a depth value obtained from an unoccluded background point Q. If the range kernel is evaluated with R(q), the depth at q will incorrectly have a high weight. The color of the 3D point Q provides a more reliable weight because the foreground and background often have very different colors.*

Note that although PMVS can produce dense reconstructions (e.g. by setting *level*=0 and *csize*=1), it usually takes hours to run, and requires a substantial amount of memory. Our goal is to provide a simple alternative for dense reconstruction that runs many times faster and requires far less memory. Our results are not as accurate as PMVS, but we have found that the quality is sufficient for many common applications including point-based rendering, surface reconstruction, and depth-of-field simulation.

3.2. Joint Bilateral Upsampling

Like [GSC*07], our algorithm attempts to compute a depth value for every pixel in a reference view R. We first obtain a sparse depth map D_s for R by reprojecting the 3D points. We then exploit the joint information in the corresponding high-resolution color image to upsample D_s to a dense depth map D_u. While this only covers pixels visible in R, we later combine the results from multiple views to obtain a full reconstruction.

Our goal is now to compute the depth of each pixel $p \in D_u$. Our method is based on joint bilateral upsampling [KCLU07]. The joint bilateral filter would compute the the depth of p as:

$$D_u(p) = \frac{1}{k_p} \sum_{q \in \Omega} D_s(q) f(\|p-q\|)\, g(\|R(p)-R(q)\|) \quad (1)$$

where $D_s(q)$ is the value of pixel q in the sparse depth map, f is the spatial kernel (typically Gaussian), g is the range kernel, Ω is the spatial support, and k_p is a normalization factor. Intuitively, this computes p's depth as a sum of its valid neighbor pixels $q \in \Omega$ (i.e. those with available depth values), weighted by both spatial proximity and color similarity.

Visibility. A naïve upsampling algorithm would use the image colors $R(p)$ and $R(q)$ in the range kernel to compute the weight for q. While this approach may work in some situations, it assumes that the depth map contains only foreground points, which is frequently not true. When a background point appears in the depth map, it can incorrectly contribute to its neighbor pixels, due to local color similarity in the image. Figure 1 illustrates this problem.

A simple solution would be to use the visibility information provided by PMVS; each 3D point is associated with a list of views in which it is identified. If we only projected points marked visible in the reference view, we would eliminate any occluded points from the depth map. However, we have found that in practice, the visibility estimates are far too conservative. Many truly visible points are discarded, and the resulting depth map is frequently too sparse to be useful.

We propose a different solution that filters out background points without relying on any visibility information. The PMVS software provides a color for each 3D point. We write these colors to the depth map, so that each valid pixel in D_s contains the color of the scene point that generated the depth value. We can then modify the range kernel to use the point colors in D_s, under the frequently valid assumption that foreground and background objects have different colors. Refer to Figure 1. We modify the upsampling equation to:

$$D_u(p) = \frac{1}{k_p} \sum_{q \in \Omega} D_s(q) f(\|p-q\|)\, g(\|R(p)-D_s^{rgb}(q)\|) \quad (2)$$

where $D_s^{rgb}(q)$ is the color of q stored in the depth map.

In practice, there are cases where the foreground and background objects have similar colors. To make our method robust in these situations, we make an additional modification to filter out occluded points that have colors similar to the foreground. We first obtain the median depth value $\text{med}(p)$ in the spatial neighborhood Ω around p. We then introduce a third weight to Eq. 2 computed as:

$$w_z = h(\|\text{med}(p) - D_s(q)\|) \quad (3)$$

We call this a *depth kernel*. The depth kernel gives higher weights to neighbor pixels that have depth values close to the median. This works well in practice, because the depth buffer resolves the visibility of at least some of the 3D points, which causes the median depth in a spatial neighborhood to favor foreground points.

3.3. Hierarchical Upsampling

Many real-world images contain large textureless areas, where multiview stereo algorithms typically find very few 3D points. While we could handle these areas by simply increasing the size of the spatial support Ω, our algorithm would quickly become inefficient. Moreover, if the background behind a sparse area is sufficiently dense, the background points will dominate the foreground points despite our modified weighting scheme. We solve these problems with a hierarchical approach.

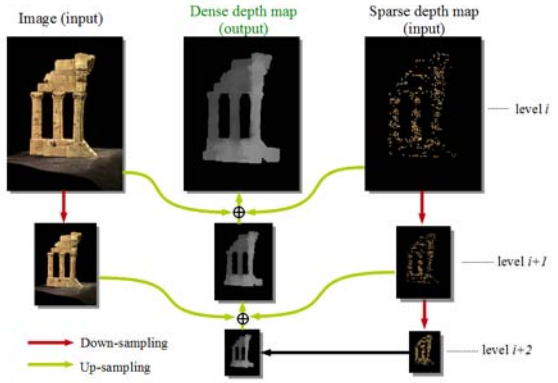

Figure 2: *Hierarchical upsampling using a Temple model from [SCD*06a]. At each level i, the reconstructed depth map at level D^{i+1} is upsampled under the guidance of image R^i to fill in the missing depth values in D^i.*

First, we generate image pyramids from both the reference view R and the initial sparse depth map D. We downsample R with a 2×2 box filter. For D, we use a min filter, so that the downsampled depth maps are equivalent to rasterizing the point cloud at lower resolutions.

If pyramid for R has k levels $R^1, \cdots, R^k$, the pyramid for D has $k+1$ levels $D^1, \cdots, D^{k+1}$. Our upsampling filter goes through k iterations, one for each level of the reference view pyramid. Starting from $i = k$, we use the depth values in D^{i+1} to fill in the missing depth values in D^i, guided by the pixel colors in R^i. Each level proceeds exactly as described in Section 3.2.

The i^{th} level of the upsampling filter generates dense depth values for D^i. Since the depth values are the estimated depths of the visible surface, we use R^i to initialize the colors of the newly interpolated points. The existing points in D^i remain unchanged. The new dense D^i is passed to the next level, until we reach the top of the pyramid, and thus obtain a full-resolution depth map. Figure 2 shows a schematic of the algorithm.

In most cases, we get good results by downsampling until the largest dimension is around 300 pixels. If the initial point cloud is particularly sparse, we can add an extra level. Since our method is fast, adjusting the number of downsampling levels is not time consuming.

3.4. Selecting and Fusing Multiple Views

Since upsampling is fast, we can interactively add more views until we have a fairly complete reconstruction. Our set of reference views is typically small (usually fewer than 10), so the upsampling cost for all of them is at most a few seconds. When processing large datasets, we can apply k-means clustering on the cameras to automatically select a set of reference views. We could also use the system presented in [FCSS10] to cluster the 6-DOF camera poses.

Confidence Values. We assign a confidence value to each reconstructed 3D point. In the single-level case, we define the confidence at pixel p to be $(\vec{n} \cdot \vec{v}) N_q / (2r+1)^2$, where N_q is the number of neighborhood pixels that have valid depth values, r is the radius of the neighborhood Ω, $\vec{n}$ is the normal at p estimated from the dense depth map, and $\vec{v}$ is the direction from the 3D location of p to the camera. The dot product penalizes points that are visible at large grazing angles; the fraction $N_q/(2r+1)^2$ penalizes points that are reconstructed from very sparse neighborhoods.

In the hierarchical case, we compute the confidence values at the bottom level as described above. Each higher level then uses the filter weights to blend the confidence values computed from the previous level. At the top level, we reject any points that have confidence values below a threshold. We set the threshold to 0.001 in all our experiments.

Fusing Depth Maps. Each reference view yields a dense point cloud representing the visible surface. We merge the depth maps from all the reference views to obtain a full point cloud. In all of our tests, we obtained satisfactory results by simply taking the union of these point clouds. Note that the input points are already globally optimized by multiview stereopis. Moreover, we usually pick reference views that have little overlap, so error due to mis-alignment is generally not noticeable. For more accuracy, we could use our confidence values in the realtime depthmap-fusion method presented in [MAW*07] without significantly decreasing the speed of our algorithm.

3.5. Implementation Details

GPU Implementation. For each reference view, we first use an OpenGL shader (with the depth test enabled) to create a depth map at the resolution of the original image. We write the color and depth values to an RGBA texture. We then build the image pyramids on GPU, as described in Section 3.3.

Next, we accelerate joint bilateral upsampling on the GPU with NVidia's CUDA. We create one thread per pixel, with 16×16 blocks. Each thread computes the depth of its corresponding pixel. To compute the depth kernel (Eq. 3), a thread must find the median of all the valid depths in its neighborhood. Due to memory constraints, we cannot find the median of every neighborhood simultaneously. However, for our purpose, an approximation of the median is good enough. To estimate the median, we first we create a 256-bin histogram of the valid depth values in each neighborhood. We then linearly scan the histogram to find the bin b that contains the median. Finally, we estimate the median as the average of the depth range of b. Once a thread has the median depth, it searches its neighborhood again, and sums

Figure 3: *Top: A point cloud computed with our algorithm, and two views of a smooth surface created using Poisson surface reconstruction with depth=8. Bottom: the same test performed on a dense point cloud from PMVS (level=1, csize=1).*

the contribution of each valid neighbor with the sampling weights (Eq. 2 and 3).

Outlier Rejection. While hierarchical upsampling is quite robust, some outliers may still exist very close to depth discontinuities. We use a simple outlier rejection step to filter them out. First, we search a small neighborhood around each pixel p in the dense depth map, and find the pixel q with the smallest 3D Euclidean distance from p. Because the reconstruction is dense, we assume that these distances conform to a Gaussian distribution (with positive x). Under this assumption, we compute the sample standard deviation, and reject any point that has fewer than n neighbors within k standard deviations. Typically, we use $k = 2$, and $n = 2 \sim 3$.

4. Results and Applications

We have evaluated our algorithm on several scenes, captured both indoors and outside. We performed all our tests on a PC with an Intel Core i7 920 CPU, 6 GB of RAM, and an NVIDIA GeForce GTX 280 graphics card. Figure 1 summarizes the results. We compare the running time of our algorithm to the time required to obtain a high-resolution reconstruction with the open source PMVS software. Our algorithm runs in considerably less time on all datasets. Moreover, our algorithm produces far more 3D points in most cases.

Temple. We use the temple dataset from [SCD*06a] to compare the quality of our dense point cloud with a dense reconstruction from PMVS. This dataset contains 312 images. It is a challenging case for our algorithm because there is little color variation. Nonetheless, we obtain a reasonable reconstruction, with nearly 400,000 points. We generated the initial sparse point cloud in under 4 minutes with PMVS, using *level*=1, *csize*=2. We ran our algorithm on 6 reference views. Upsampling took a less than a second. Figure 5 shows several views of the point cloud. We also used our results to generate a smooth mesh with Poisson surface reconstruction [KBH06] at *depth*=8. Figure 3 shows the reconstructed surface.

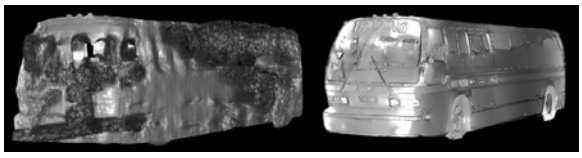

Figure 4: *Left: a point cloud created with naïve upsampling that ignores visibility of the 3D points. Right: our results.*

To compare with PMVS, we used their algorithm again to generate a dense point cloud using *level*=1, *csize*=1. The PMVS software took 22 minutes, and produced 188,000 points. Note that while their point cloud is more accurate and preserves more details, ours algorithm produced twice as many points in considerably less time. We also created a smooth surface from the PMVS points. The reconstructed surfaces have similar quality. The results are shown in Figure 3.

Bus. The bus scene (Figure 6) has simple geometry, but it is non-trivial due to the prominent specular highlights and large textureless areas. Despite these difficulties, our algorithm recovered an accurate and dense point cloud. To evaluate the effectiveness of our upsampling method, we also attempted to reconstruct the bus with a naïve filter that ignored the visibility and color of the 3D points (Figure 4). While our algorithm was able to separate foreground from background, the naïve interpolation blended foreground points on the bus with background points behind the bus, resulting in a noisy reconstruction with many holes.

Chapel. The chapel scene (Figure 7) shows a 360° reconstruction. We fused 17 reference views to obtain nearly 4 million points. We also use the chapel scene to demonstrate the effectiveness of our algorithm at reconstruction from very few images (Figure 9). We first used Bundler to compute a sparse point cloud from only two of the images. Because we used only two views, Bundler ran in 30 seconds. Although the Bundler points provided depths for only 0.1% of the pixels in each image, our algorithm computed an accurate surface with more than a million points.

Office. The office scene (Figure 8) has many depth-discontinuities, and multiple depth layers. Our algorithm performs well despite these difficulties. Note the density of the point cloud in the closeup image.

Depth-of-field Manipulation. We use our results to digitally modify the depth of field in photographs. We begin by reconstructing a sparse set of points from a small number of

Scene	#Photos	Res	Hierarchical Upsampling			PMVS low res.				PMVS high res.			
			#Views	#Points	Time	Levels	Cell Size	#Points	Time	Levels	Cell Size	#Points	Time
Temple	312	640 × 480	6	379k	0.55 s	1	2	12k	232 s	1	1	188k	22 m
Bus	62	1200 × 800	1	410k	0.37 s	2	3	14k	84 s	1	1	687k	37 m
Office	21	1600 × 1067	5	3,059k	5.61 s	2	2	44k	87 s	1	1	710k	26 m
Chapel1	132	1024 × 683	17	3,915k	3.58 s	2	2	60k	145 s	1	1	1,171k	32 m
Chapel2	2	1280 × 960	2	1,406k	0.99 s	–	–	–	–	–	–	–	–

Table 1: *A summary of our test datasets. For each dataset, we list the number of original photos and the image resolution. Next, we list the number of reference views used by our algorithm, the number of 3D points produced, and the running time for upsampling. The last 8 columns show the performance of PMVS. We summarize the low-resolution reconstructions that provided our initial points first, followed by high-resolution reconstructions for comparison. In each case, we list the number of subdivision levels, the cell size, the number of 3D points, and the running time. Because the last dataset only contains two views, did not run PMVS.*

images. We then use our algorithm to obtain per-pixel depths in the image to be refocused. Finally, we blur the image with a spatially-varying Gaussian filter, where the variance is a function of the depth value. We accelerate the blur on the GPU, which allows the user to change the depth of field interactively. We refocused several images taken from a tall building (Figure 10). In each example, we simulate a narrow depth of field to create a miniature model effect. Figures 11 and 12 show refocused images of an office scene and a toy house, respectively.

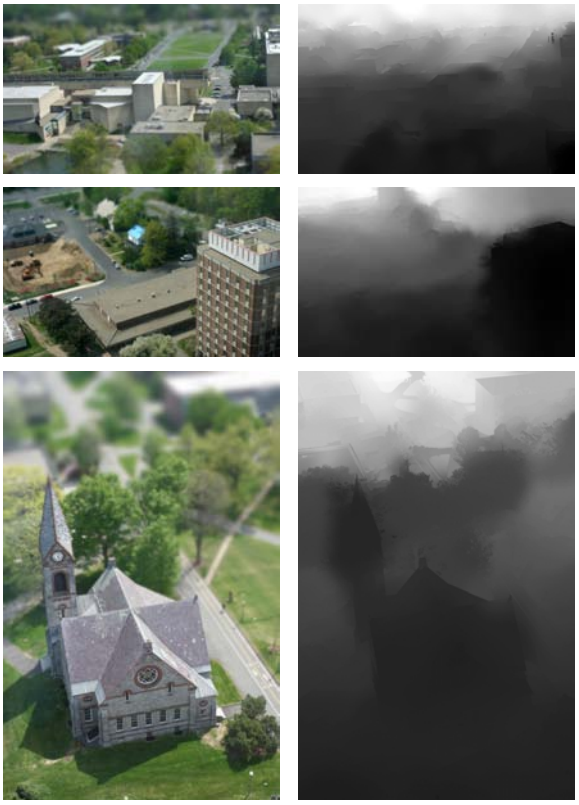

Figure 10: *We use our depth maps to digitally modify the depth-of-field in photographs, creating a miniature-faking effect. Zoom in to see the details in the images.*

Limitations and Future Work. At present, our algorithm requires a sparse set of depth values to seed the reconstruction. We could possibly use the massive parallelism of the GPU to find feature correspondences in areas where there are not enough initial depth values to estimate the depthmap accurately. Plane-sweeping has already been applied to real-time stereo [GFM*07]. While the results are usually noisy, we could keep only the most consistent points from a plane-sweeping algorithm and use our system to fill in the details. With such a system, we could use a calibrated stereo camera to recover depth from video without relying on multiple frames to eliminate noise.

Our algorithm performed well on all of our datasets with only a naïve fusion system. However, it is possible that our algorithm could benefit from a more sophisticated fusion system in some scenes. If we used our confidence values to merge multiple depthmaps, we could potentially improve our results. We could possibly merge the depthmaps from only the past few frames in a video in order to get more accurate results without requiring a large number of frames.

5. Acknowledgements

Omitted for review.

References

[BN07] BANDO Y., NISHITA T.: Towards digital refocusing from a single photograph. In *PG* (2007), pp. 363–372. 2

[CBTT08] CHAN D., BUISMAN H., THEOBALT C., THRUN S.: A noise-aware filter for real-time depth upsampling. In *ECCV Workshop on MMSFAA* (2008). 2

[Col96] COLLINS R.: A space-sweep approach to true multi-image matching. In *CVPR* (1996), pp. 358–3636. 2

[DBPT10] DOLSON J., BAEK J., PLAGEMANN C., THRUN S.: Upsampling range data in dynamic environments. In *CVPR* (2010). 2

[DD02] DURAND F., DORSEY J.: Fast bilateral filtering for the display of high-dynamic-range images. *ACM Trans. Graph. 21*, 3 (2002), 257–266. 2

[DT05] DIEBEL J., THRUN S.: An application of markov random fields to range sensing. In *NIPS* (2005), pp. 291–298. 2

[FCSS10] FURUKAWA Y., CURLESS B., SEITZ S., SZELISKI R.: Towards internet-scale multi-view stereo. In *CVPR* (2010). 4

Figure 5: *A full reconstruction of the temple model [SCD*06a]. We merged the depth maps from 6 reference views to produce nearly 400k points. Upsampling took less than a second. On the right, we show the initial sparse point cloud.*

Figure 6: *A reconstruction from a single reference view. This model has over 400k points, upsampled in under a second. While the bus has simple geometry, it is challenging because it is highly specular, and many surfaces have little texture. Again we show the initial point cloud on the right.*

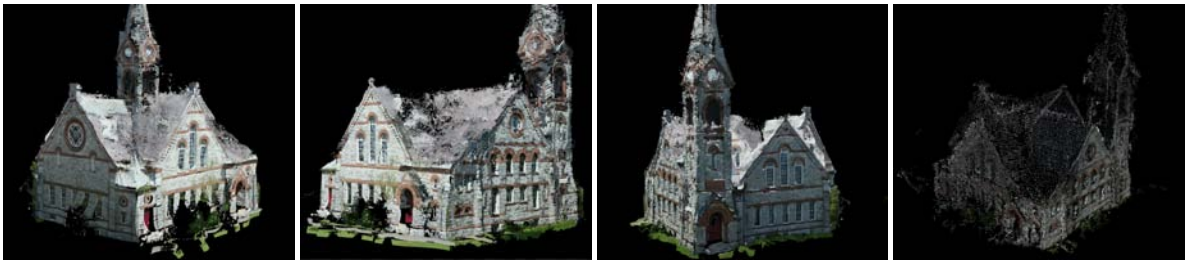

Figure 7: *A* 360° *reconstruction of the chapel scene from 17 reference views. This model contains nearly 4 million points, upsampled in under 4 seconds.*

Figure 8: *A reconstruction of an office from 5 reference views. This model contains over 3 million points, upsampled in under 6 seconds. The third image is a closeup of the bookshelf, which shows the density of our reconstruction.*

Figure 9: *A dense reconstruction obtained from only 2 views through our entire pipeline. Because there are only two views, we computed the initial point cloud with Bundler in about 30 seconds. Although Bundler provided depths for only about 0.1% of the pixels in each image, our algorithm produced an accurate model composed of more than a million points in under a second.*

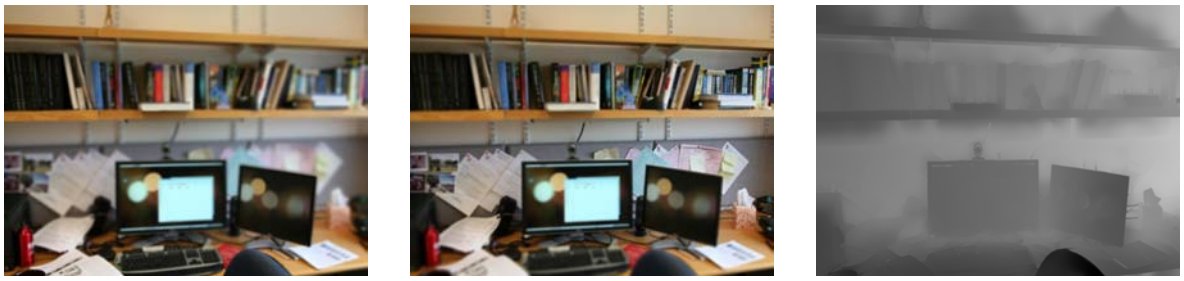

Figure 11: *Left: An image of an office, refocused on the chair. Middle: The same image, refocused on the wall behind the right monitor. Right: The depth map.*

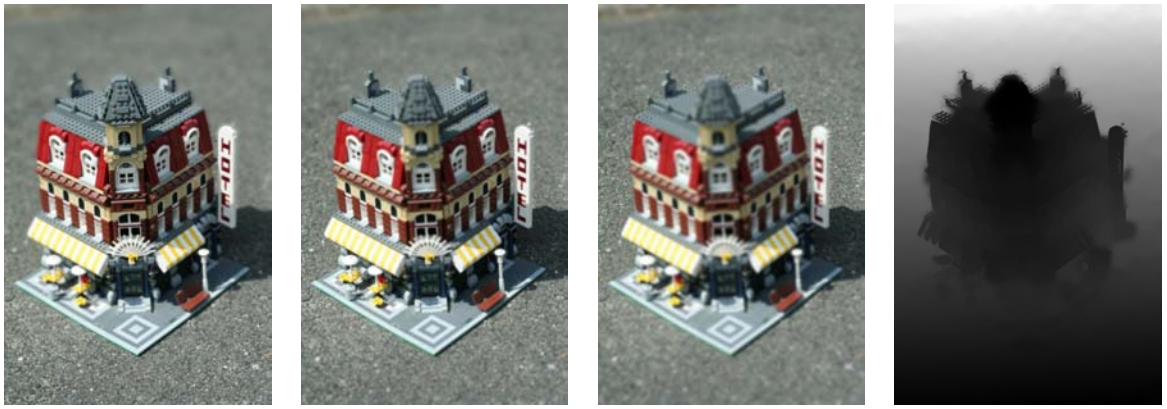

Figure 12: *A sequence of refocused images of a toy house. From left to right, the images are focused on the tower, the left corner of the patio, and the background. The far right image shows the depth map.*

[FP09] FURUKAWA Y., PONCE J.: Accurate, dense, and robust multi-view stereopsis. *IEEE PAMI 1*, 1 (2009), 1–8. 1, 2

[GFM*07] GALLUP D., FRAHM J., MORDOHAI P., YANG Q., POLLEFEYS M.: Real-time plane-sweeping stereo with multiple sweeping directions. In *CVPR* (2007), pp. 1–8. 2, 6

[GSC*07] GOESELE M., SNAVELY N., CURLESS B., HOPPE H., SEITZ S.: Multi-view stereo for community photo collections. In *ICCV* (2007), pp. 1–8. 2, 3

[HZ04] HARTLEY R., ZISSERMAN A.: *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. 2

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proc. of SGP* (2006), pp. 61–70. 2, 5

[KCLU07] KOPF J., COHEN M., LISCHINSKI D., UYTTENDAELE M.: Joint bilateral upsampling. In *SIGGRAPH* (2007), vol. 26, p. 96. 1, 2, 3

[MAW*07] MERRELL P., AKBARZADEH A., WANG L., MORDOHAI P., FRAHM J., YANG R., NISTÉR D., POLLEFEYS M.: Real-time visibility-based fusion of depth maps. In *ICCV* (2007), pp. 1–8. 2, 4

[PD09] PARIS S., DURAND F.: A fast approximation of the bilateral filter using a signal processing approach. *IJCV 81*, 1 (2009), 24–52. 2

[PRD09] PAN Q., REITMAYR G., DRUMMOND T.: ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In *BMVC* (2009), pp. 1–11. 2

[RM04] RAJAGOPALAN A., MUDENAGUDI U.: Depth estimation and image restoration using defocused stereo pairs. In *PAMI* (2004), vol. 26, pp. 1521–1525. 2

[SCD*06a] SEITZ S., CURLESS B., DIEBEL J., SCHARSTEIN D., SZELISKI R.: Multiview stereo evaluation, 2006. http://grail.cs.washington.edu/projects/mview/. 4, 5, 7

[SCD*06b] SEITZ S. M., CURLESS B., DIEBEL J., SCHARSTEIN D., SZELISKI R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR* (2006), pp. 519–528. 2

[SSS06] SNAVELY N., SEITZ S., SZELISKI R.: Photo tourism: exploring photo collections in 3d. *ACM Trans. Graph. 25*, 3 (2006), 835–846. 2

[TM98] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *ICCV* (1998), pp. 839–846. 2

[Wei06] WEISS B.: Fast median and bilateral filtering. *ACM Trans. Graph. 25*, 3 (2006), 519–526. 2

[YK06] YOON K., KWEON I.: Adaptive support-weight approach for correspondence search. *PAMI 28*, 4 (2006), 650–656. 2

[YP03] YANG R., POLLEFEYS M.: Multi-resolution real-time stereo on commodity graphics hardware. In *CVPR* (2003), vol. 1, pp. 211–218. 2

[ZC09] ZHANG W., CHAM W.: Single image focus editing. In *ICCV* (2009), pp. 1947–1954. 2