User-Perspective AR Magic Lens from Gradient-Based IBR and Semi-Dense Stereo

Domagoj Baričević, *Student Member, IEEE*, Tobias Höllerer, *Senior Member, IEEE*, Pradeep Sen, *Senior Member, IEEE*, and Matthew Turk, *Fellow, IEEE*

Abstract—We present a new approach to rendering a geometrically-correct user-perspective view for a magic lens interface, based on leveraging the gradients in the real world scene. Our approach couples a recent gradient-domain image-based rendering method with a novel semi-dense stereo matching algorithm. Our stereo algorithm borrows ideas from PatchMatch, and adapts them to semi-dense stereo. This approach is implemented in a prototype device build from off-the-shelf hardware, with no active depth sensing. Despite the limited depth data, we achieve high-quality rendering for the user-perspective magic lens.

Index Terms—Augmented reality, magic lens, user-perspective, image based rendering, gradient domain, semi-dense stereo

1 Introduction

The metaphor of the magic lens is used to describe a common interface paradigm in which a display region reveals additional hidden information about the objects the user is interacting with. This metaphor was originally introduced for traditional GUIs, but it has also been adopted as an intuitive interface for some VR applications. More prominently, today it is the *de facto* standard interface for Augmented Reality due to the wide adoption of hand-held computing devices such as smartphones and tablets. Indeed, it is these devices that are largely responsible for bringing AR into the mainstream consumer market.

However, while the concept of the magic lens is a natural fit for hand-held AR, the typical approach falls short of the full vision of the metaphor. At issue is the perspective of the augmented scene. While concept imagery for AR often presents the magic lens as an almost seamless transparent display, in reality nearly all current magic lens implementations rely on video-see-through methods where the device displays and augments video captured by the camera on the back of the device. As a result the AR scene is presented from the perspective of the device, instead of that of the user.

This device-perspective approach does not provide a fully intuitive and seamless experience for the user. There is a clear break between what is in the real world and what is mediated by the device. Furthermore, the sometimes dramatic change in perspective can have negative effects on usability and spatial understanding. As an example, consider an AR application for visualizing interior design and furniture arrangement (see Fig. 1); this is a popular use case for AR. The entire purpose of AR in this type of application is to give the user a better sense of what the space will look like with the new décor or furniture. However, device-perspective AR will distort the perspective and scale of the room (Fig. 1b) so the user will not get a true feel for the future remodeled room. On the other hand, a true magic lens would show the augmented scene at the same human scale as the real world. Ideally there would be no perspective difference between the scene inside and outside the magic lens (Fig. 1c). This type of interface is referred to as a *user-perspective magic lens*.

Scene reconstruction has been at the heart of the problem of creating a user-perspective magic lens. Since the scene is mediated by an opaque display, it has to be re-rendered from a scene model. Reconstruction is still a challenging research problem. While active depth sensors provide good results and have recently become commonplace, they have constraints such as range limits and inability to work outdoors in strong sunlight. Another approach to scene reconstruction is stereo vision, where the depth of the scene is reconstructed by matching two views of a stereo camera pair. The advantage of stereo reconstruction is that it can work with standard cameras, it does not need active illumination, and there are no major restrictions with regard to outdoor scenes.

Some stereo reconstruction algorithms can provide quite accurate depth maps, but this comes at a performance penalty. Fully accurate depth maps cannot yet be achieved at frame rate. Real-time stereo can produce depth maps that are sufficient for many applications, but they are not very good for the purpose of re-rendering a real world scene. Typically, real-time stereo approaches achieve speed by using a small depth range (limiting the number of different

- *D. Baričević, T. Höllerer, and M. Turk are with the Department of Computer Science, University of California, Santa Barbara, CA 93106. E-mail: {domagoj, holl, mturk}@cs.ucsb.edu.*
- *P. Sen is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106. E-mail: psen@ece.ucsb.edu.*

Manuscript received 16 Feb. 2015; revised 4 Apr. 2016; accepted 17 Apr. 2016. Date of publication 27 Apr. 2016; date of current version 26 May 2017. Recommended for acceptance by D. Schmalstieg. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TVCG.2016.2559483

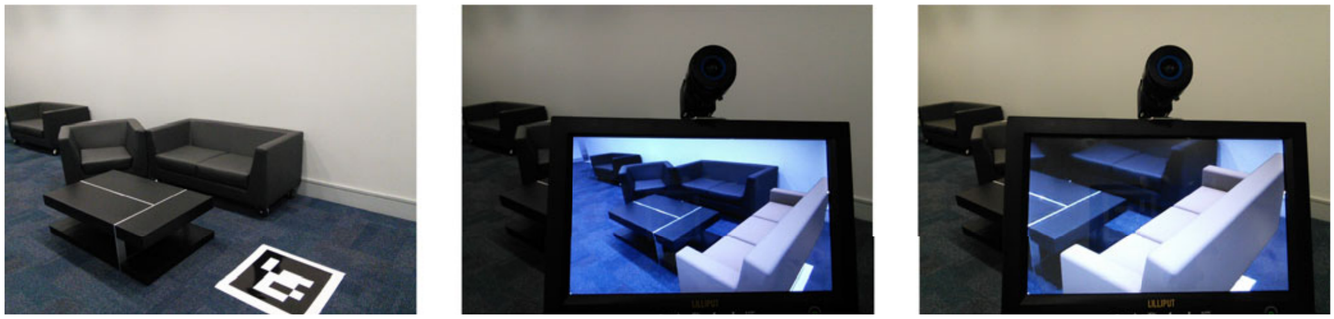

Fig. 1. An example Augmented Reality application showcasing the difference between user-perspective and device-perspective magic lens interfaces. (a) Real world environment only. (b) Augmented Reality scene with the conventional device-perspective magic lens. (c) AR scene rendered with our user-perspective magic lens prototype.

depth values), resulting in a scene model composed of distinct front facing planes. Re-rendering this model from a new point-of-view can result in a scene composed of obvious distinct layers.

In this paper we extend our previous work [1] and present a new approach to solving the problem of creating a user-perspective magic lens. We observe that accurate dense scene reconstruction is a requirement imposed by the traditional rendering methods, and not an inherent requirement of creating a user-perspective view. By taking a different approach to rendering, we lower the requirements for reconstruction while also achieving good results. We do this by using image-based rendering (IBR) [2]. IBR can produce high quality results with only limited scene models by leveraging existing imagery of the scene. This fits very well with the nature of our problem.

The key to our approach is the adoption of a recent gradient domain IBR algorithm [3] that is paired with a novel semi-dense stereo matching algorithm we developed. The IBR algorithm we use renders from the gradients in the image instead of the pixel color values. It achieves good results as long as the depth estimates of the strongest gradients are good, even if the depths of the weak gradients are incorrect. This fits well with the general behavior of stereo reconstruction, but we exploit it further by using a semi-dense stereo algorithm to compute depths only at the strongest gradients.

With this approach we have created a geometrically-correct user-perspective magic lens with better performance and visual quality than previous systems. Furthermore, we use only passive sensing, and support fully dynamic scenes with no prior modeling. Due to the use of face tracking, we do not require instrumenting the user. Although our prototype system is tethered to a workstation and powered by a GPU, we are confident that given the rate of advancement of mobile hardware this will be possible on a self-contained mobile platform in just a few years.

This paper extends our previous work [1] in several ways. The stereo pipeline has been substantially reworked, and the algorithm modified to make use of adaptive support weights. In order to improve overall rendering quality, we have added explicit support for temporal coherency. We also provide a more extensive evaluation of our results. We include a comparison of our stereo results to those of other algorithms, and we compare our user-perspective rendering to ground truth references.

2 Related Work

The "magic lens" metaphor was first introduced by Bier et al. [4] as a user interface paradigm developed for traditional desktop GUI environments. The basic idea is that of a movable window that alters the display of the on-screen objects underneath it, acting like an information filter that reveals hidden objects or information.

This concept of an information filtering widget was quickly adopted outside traditional desktops. Viega et al. developed 3D versions of the magic lens interface, both as flat windows and as volumetric regions [5]. The Virtual Tricorder [6] was a interaction device for an immersive VR environment that featured a mode in which a hand-held tool revealed altered views of the 3D world. In [7], Rekimoto and Nagao introduced hand-held Augmented Reality with the NaviCam system. The NaviCam was a video-see-through AR system consisting of a palmtop TV with a mounted camera and tethered to a workstation. This hand-held video-see-through approach soon became the norm for Augmented Reality interfaces [8]. Optical see-through AR approaches (e.g., [9], [10], [11]) can implement perspectively correct AR magic lenses without the need for scene reconstruction but have to cope with convergence mismatches of augmentations and real objects behind the display unless they use stereoscopic displays.

There have been efforts in the AR community to design and develop video see-through head-worn displays that maintain a seamless parallax-free view of the augmented world [12], [13]. This problem is slightly simpler than correct perspective representation of the augmented world on hand-held magic lenses since the relationship between the imaging device and the user's eyes is relatively fixed.

With the proliferation of smartphones and tablets AR has reached the mainstream consumer market; this has made hand-held video-see-through the most common type of AR and it is what is often assumed by the term "magic lens" when used in the context of AR [14], [15]. Since the display of the augmented environment from the perspective of the device's camera introduces a potentially unwanted shift of perspective, there is renewed interest in solutions for seamless user-perspective representation of the augmented world on such self-contained mobile AR platforms. User studies conducted using simulated [16] or spatially constrained [17], [18] systems have shown that user-perspective views have benefits over device-perspective views. Several systems have attempted to create a user-perspective view by warping the video of a video-see-through magic lens [19], [20], [21]; however these approaches can only approximate the true user-perspective view as they are unable to change the point of view and therefore do not achieve the geometrically correct view frustum.

The most directly relevant work to this paper is the geometrically-correct user-perspective hand-held augmented reality magic lens system in [16], which used a Kinect depth sensor and a Wiimote to track the user's head position. The approach relies on the fairly high quality depth information provided by the Kinect to obtain an accurate 3D model of the real world; the final scene is then rendered using conventional rendering methods (raycasting and scanline rendering). While the approach is fairly straightforward, it has certain constraints. First, the system does not gracefully handle dynamic scenes as the scene is rendered in two layers with different real time characteristics. Second, active depth sensors like the Kinect cannot operate well under strong sunlight (or any other strong light source that emits at their frequency).

Stereo reconstruction is one of the most well researched areas of computer vision. A full overview is well beyond the scope of this paper. For an excellent review of the field we refer the reader to [22]. In recent years, a number of algorithms have been proposed that take advantage of GPU hardware to achieve real-time performance [23], [24], [25], [26]. While these algorithms can produce fairly accurate dense disparity maps, the real-time speeds are achieved for relatively low resolutions and narrow disparity ranges. Our stereo algorithm is inspired by PatchMatch [27], an iterative probabilistic algorithm for finding dense image correspondences. Bleyer et al. [28] proposed a stereo matching algorithm based on PatchMatch primarily designed to support matching slanted surfaces, although it also supports front facing planes. In [29] this was adapted for real-time 3D shape reconstruction by using a faster matching cost and relying on a volumentric fusion process to compensate for the noisy per-frame depth maps.

Image-based rendering techniques create novel views of a scene from existing images [2]. These novel views can be rendered either purely from input image data [30], or by using some form of geometry [31], [32]. Our approach is based on the gradient-domain image-based rendering work by Kopf et al. [3]. Their method creates novel views by computing dense depth maps for the input images, reprojecting the gradients of the images to the novel view position, and finally using Poisson integration [33] to generate the novel view.

3 Overview

As mentioned above, our approach is based on the gradient domain image-based rendering algorithm by Kopf et al. [3]. For a detailed description of the algorithm we refer the reader to the original paper; here we will only give a brief high level overview in order to introduce the idea. We also give a more detailed explanation of how we adapted the method for our system in Section 5 below.

The main idea behind gradient domain methods is that an image can be reconstructed from its gradients by performing an integration. Therefore, if one needed to generate an image corresponding to a new viewpoint of a scene (as in a user-perspective magic lens), one could do so by integrating the gradient images for those viewpoints. These gradient images can be obtained by reprojecting the gradients computed for an existing view of a scene for which there is scene geometry information. Since strong gradients are generally sparse in a scene, and since stereo matching algorithms work best at strong gradients, this approach provides a way to create a high quality image even without a fully dense and accurate depth map as long as the strongest gradients are correctly reprojected. While there will be errors in the reprojected gradient image, they will be mostly confined to weak gradients that do not have a large effect on the integration of the final solution. In contrast, a standard reprojection method would result in a noisy solution with much more noticeable artifacts.

Using a rendering method that only requires good depth information at the gradients gives us the opportunity to optimize our stereo reconstruction. Instead of the standard approach of computing a dense depth map across the input image pair, we can compute semi-dense depth maps that only have information at the parts of the image that have strong gradients. The depth of the rest of the image can then be approximated by filling in depth values extrapolated from the computed parts of the depth map. As long as the depth information for the strongest gradients is correct, the final rendered solution for the novel view will not have significant artifacts.

In order to achieve this goal we have developed a novel semi-dense stereo matching algorithm inspired by PatchMatch [27]. The algorithm is simple and fast, but it computes accurate results over the areas of interest. A detailed description of the algorithm is given in Section 4 below.

3.1 Creating a Novel View

The basic steps to generating a novel view with our approach are shown in Fig. 2. The input to the pipeline is a stereo pair (Fig. 2a shows left image) and a desired position for the novel view.

The first step (Fig. 2b) is to filter the input image pair in order to produce a mask that marks the pixels that are at the strong gradients. We define the gradients as forward difference between neighbors. The overall strength of the gradient is computed by taking the maximum between the horizontal and vertical strengths, which are defined as the average of the per channel absolute differences.

We then apply a threshold to this gradient strength image to create a gradient mask. We use a global threshold for the entire image. The threshold can be either a set fixed value or the current average gradient magnitude. In practice, we find a fixed threshold between 5 and 10 to work well. We first clean the mask by removing pixels that have no neighbors above the threshold and then perform a dilation step (Fig. 2c).

Next, our stereo matching algorithm is run over the masked pixels. This results in a semi-dense disparity map (Fig. 2d) with good depth estimates for the masked areas with strong gradients, and no data for the rest of the image. We then perform a simple extrapolation method to fill-in the disparity map across the image (Fig. 2e). Then the 3D position of each pixel is computed from the disparity map. The renderer takes the 3D position information, as well as the desired novel view's camera parameters (position, view frustum, etc.) and generates the final image (Fig. 2f).

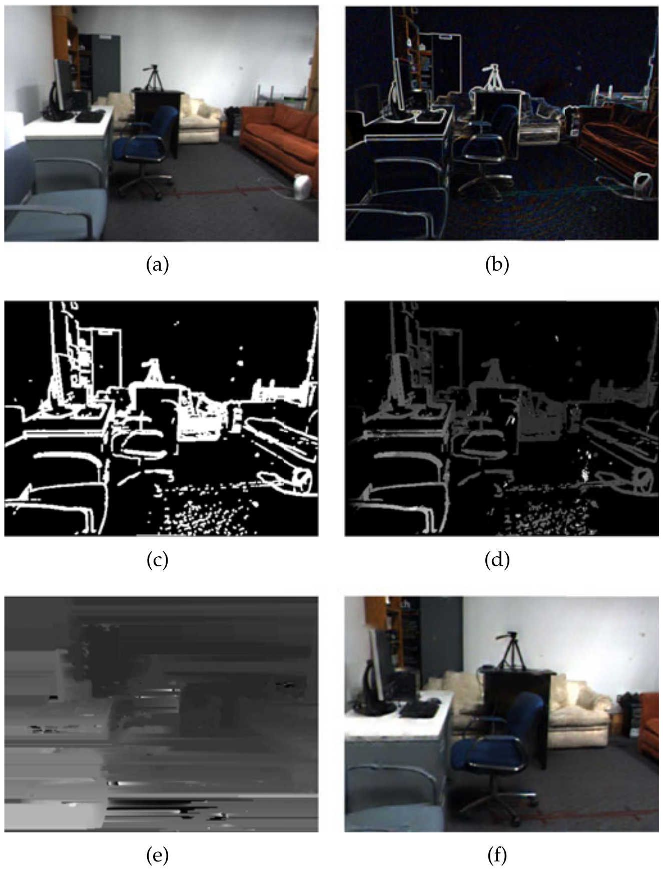

Fig. 2. The steps to rendering a novel view: (a) input image, (b) gradient magnitudes of input, (c) mask of strongest gradients, (d) disparity map for masked area, (e) filled-in disparity map, (f) final solution. (Note: (a)-(e) are for left camera, (f) is for final pose)

4 Stereo Reconstruction

One of the most important considerations in the development of our algorithm was the need to run as fast as possible. This led to a parallel GPU-based approach, which in turn set additional constraints. One of the principal tenets of GPU computing is to avoid code path divergence. That is, each thread in a concurrently running group of threads should execute the same steps in the same order at the same time, just using different data. This demand led to several design decisions regarding our algorithm.

4.1 Mask Indexing

The mask computed from the gradient magnitudes determines the pixels for which the stereo algorithm will compute disparities. However, since the algorithm is implemented on the GPU using CUDA, using this mask directly would be inefficient. A naïve approach would be to run a thread per pixel and simply exit the thread if the pixel is not in the mask. However, this is very inefficient, as these threads will not truly exit. The SIMT (Single Instruction Multiple Threads) nature of the GPU hardware requires all the threads that are concurrently running on a core to follow the same code path. If even one thread in that group is in the mask and needs to run the algorithm, then all the threads in the group might as well run since they would introduce (almost) no overhead. In order to get any performance gain, all the pixels in the image region covered by the group would have to be outside the mask. This is rare in natural images, as there are almost always a few strong gradient pixels in any part of the image. This means that the naïve approach to the gradient-guided semi-dense stereo algorithm degenerates to a dense algorithm.

In order to prevent this waste of computational power we re-group the gradient pixels so that they cluster together, by performing per-line stream compaction. We process the mask image to create an array of pixel indices. Each row of the mask is traversed in parallel, and when a pixel that is inside the mask is encountered, its index is saved in the output array at the same row and in the next available column. Pixels outside the mask are simply ignored. As a result the indices of the masked pixels are densely stored in the output array. The count of masked pixels in a row is saved in the first column of the output array. This process creates a mask whose blocks are mostly completely full or completely empty, with only a few that are partially full. This mask is much more suitable for parallel processing on GPU architectures.

This compact representation of the masked areas also serves a second purpose. Since it indexes the strong gradients in the image, it is used as a look-up table for potential matches during the Random Search step (see Section 4.3.2).

4.2 Matching Cost

A key aspect of stereo reconstruction algorithms is the matching cost. The matching cost is a measure of the error in matching a pixel from the left image to a pixel in the right image. Stereo algorithms work by minimizing this error in order to produce an accurate disparity map. Here we will give an overview of the matching cost we adopted. These details will be particularly relevant to the discussion of the Spatial Propagation step in Section 4.3.3.

Local stereo algorithms (such as ours) typically compute the matching cost for a pixel by considering the areas around the pixel and its match (the support window). This consists of two parts: per-pixel dissimilarity, and cost aggregation. The first computes a dissimilarity of the individual pixels in the support window, the second aggregates these into a final matching cost for the pixel of interest.

Given an image pair I_{left} and I_{right}, consider a pixel p in the left image I_{left} for which we want to compute the matching cost $M(p,d)$ given a disparity d. We define a support window W_p around this pixel, and each pixel $q \in W_p$ has a per-pixel dissimilarity $D(q,d)$. This dissimilarity is computed between q and its corresponding pixel in I_{right} denoted as q' where $q' = q - d$. The simplest form of this is the absolute color difference between the color intensities of q and q', but a common addition is to consider the difference in the gradients as well.

With the dissimilarities $D(q,d)$ computed for every $q \in W_p$, the next step is to aggregate them into a matching cost $M(p,d)$ for pixel p. A simple aggregation strategy is to compute a sum of the per-pixel dissimilarities:

$$M(p,d) = \sum_{q \in W_p} D(q,d). \quad (1)$$

If D is computed as an absolute color difference this becomes the common SAD (sum of absolute differences, also known as the L^1 norm) matching cost metric. This aggregation strategy is simple and computationally efficient, but leads to edge fattening because each pixel in the support window is given equal consideration. In [34] Yoon and Kweom introduced a simple strategy to overcome the edge fattening problem. The idea is to assign individual weights to the per-pixel dissimilarities during the aggregation:

$$M(p,d) = \sum_{q \in W_p} W(p,q) \cdot D(q,d). \tag{2}$$

This *adaptive support weight* aggregation strategy has led to local stereo matching approaching the quality of much slower global algorithms, with researchers proposing a number of competing methods to computing the support weights [35]. Among these, the best performing ones are the original method proposed by Yoon and Kweaom, and the cost volume filtering by Hosni et al. [36]. Although the method in [36] is faster, it is not well suited to our stereo matching approach. It assumes dense stereo, and computes the entire cost (per-pixel dissimilarity) volume for a limited disparity range. By contrast, our semi-dense approach is designed to avoid computation for known invalid disparities, and so enable a wider disparity range. Therefore, we adopt a modified version of the support weight proposed in [34] as it is better suited to our approach.

Yoon and Kweaom proposed a weight $W(p,q)$ based on the color and spatial distance between p and q:

$$W(p,q) = exp\left(-\left(\frac{|I(p) - I(q)|}{\gamma_c} + \frac{\|p-q\|}{\gamma_s}\right)\right). \tag{3}$$

Their matching cost aggregation used two such weights per pixel q, one for the left side and one for the right side of the matching pair ($W(p,q)$ and $W(p',q')$). However, Hosni et al. [35] show that the overhead of this symmetric approach can be avoided, and that a single weight can be used. Furthermore, [35] also shows that omitting the spatial component does not have a significant effect on the results. Hence, we adopt a single weight based only on the color distance, and our final matching cost is therefore:

$$M(p,d) = \sum_{q \in W_p} e^{-\left(\frac{|I(p)-I(q)|}{\gamma_c}\right)} \cdot D(q,d). \tag{4}$$

We compute this cost over a 7×7 patch centered around the pixel of interest. We set γ_c to $1/7$ of the maximum color distance. The matching cost is normalized by the sum of the weights.

4.3 Stereo Matching

We implemented a simple, fast, and accurate stereo matching algorithm inspired by PatchMatch. Our algorithm takes the basic ideas of random search and propagation from PatchMatch and applies it to the domain of semi-dense stereo matching at the gradients and in parallel. Although inspired by PatchMatch, the specific details are somewhat different due to the nature of the problem.

The algorithm consists of three main intra-frame steps: Random Search, Spatial Propagation, and Local Sweep. There is also an inter-frame step: Temporal Propagation. The Random Search and Spatial Propagation steps are iterated. Each step (and each iteration in a step) is fully parallel at the individual pixel level. Only the steps themselves and their iterations are serialized.

4.3.1 Data and Initialization

The algorithm takes as its input the stereo image pair and the arrays with the mask indices. It outputs the disparity values and matching costs for each camera of the stereo pair. Before the algorithm starts, the disparities are initialized to zero, while the costs are initialized to the maximum possible value. In our current implementation we use unsigned 8-bit values to store the disparities, with a disparity range of [0, 255]. Although the raw matching costs values are floating point, they are rescaled and stored as unsigned 16-bit integers. This gives a maximum range of [0, 2^{16}-1] which is more than sufficient for our needs. We reserve the top of that range so that the maximum possible value for a matching cost is always below the upper limit of the range. This then enables us to initialize the cost to 0xffff, which simplifies the search for the minimum cost disparity since there is no need to treat the first candidate disparity differently from the rest.

4.3.2 Random Search

The random search step consists of generating a random disparity value, computing the matching cost given that disparity, and keeping it if the cost is lower than the current cost. This can then be repeated a number of times before continuing to the propagation step.

Regular PatchMatch [27] initializes fully randomly from all possible correspondences, and the random search is done by randomly searching from all possible correspondences within a shrinking window centered on the current solution. Our approach is different. First, the initialization and random search is a single unified step. Second, the random disparity is not generated from the disparity range but from the valid indices for that epipolar line. We are matching only the strong gradients that are within our masks.

In general, if a part of the scene is labeled as a strong gradient in the left image it will also be labeled as a strong gradient in the right image (and vice-versa). This is not the case for parts that are occluded in one image of the pair, but those do not have a correct match anyway. If follows that a pixel within the gradient mask of one image will have its corresponding pixel within the gradient mask of the other image. Since the gradients are generally sparse, this significantly reduces the possible valid disparities. This reduction in search space means that each random guess has a higher probability of being correct, which improves convergence.

Therefore, when generating a random disparity we sample from the space of valid indices, not from the full disparity range. As mentioned above, the first column of each row in the mask index stores the number of valid pixels. This value is used as the range of a uniform random distribution. We generate a random integer from this distribution, this number gives us the column in the mask index row to sample. The index stored in that column gives us our random match candidate. We then compute the matching cost for this candidate correspondence, if the cost is lower than the current cost we save the disparity and the cost as the current best match. This process can be iterated, in our current implementation we run two iterations.

4.3.3 Spatial Propagation

The random search step will generate a very noisy disparity map where most of the disparities are wrong, but some are correct. The propagation step serves to propagate the good matches across the image. Here our algorithm also differs significantly from PatchMatch.

Taking the standard PatchMatch approach to propagation would present several problems for our application scenario. Firstly, the computation cost is too high. In the serial version the image is processed linearly from one corner to the next. At each pixel the disparities of the preceding horizontal and vertical neighbors are used as possible new disparities and new matching cost are computed. If the cost of a candidate disparity is lower than the current one, the new disparity is adopted. Computing the matching cost is expensive in general, and doing it serially is prohibitive. The performance would be far too slow for real-time use. Parallel versions of PatchMatch have been proposed, but they are still not well suited to our application. Although the computations are done in parallel, much more are needed per pixel. Even the parallel versions require too many expensive matching cost computations per frame.

Secondly, PatchMatch is meant for computing dense correspondences. We only compute disparities within the masked areas. This means there are large gaps in the image. Although it is possible in principle to propagate by skipping those gaps, this would violate the assumption of propagating between neighbors and it is unlikely that that kind of propagation would be useful. In the case of parallel implementations of PatchMatch, the propagation is limited in radius so it would not be able to skip gaps anyway.

We take a different approach to the propagation step. Instead of propagating serially through the entire image, we have each pixel in parallel check its neighborhood. However, we do not compute another matching cost for each of the neighbor's disparities as that would be prohibitively expensive (for a 7×7 patch it would require up to 48 matching cost computations). We observe that we are not really interested in the matching costs for all the possible candidate disparities. Instead, we simply want to find the best disparity among those proposed by the pixel's neighbors. It would be ideal if this disparity could be found without a lot of expensive computation.

Recall that the matching cost is aggregated over a support window, and that it is computed from two parts: the per-pixel dissimilarities, and the per-pixel support weights. For a pixel p and one of its neighbors $q \in W_p$ (with a current disparity $d = disp(q)$) there is an overlap between the support window W_p of p and the support window W_q of q. The per-pixel dissimilarities in this overlapped region will be the same for both support windows, therefore their contributions to the matching costs of p and q will be the same. This leads to the question of whether it is possible to compute an approximation of the matching cost $M(p,d)$ from the cost $M(q,d)$. One issue with this is that the summation in the matching cost computation is not a reversible operation, and so it would not be possible to entirely remove the influence from the pixels outside the overlapping region. However, the bigger influence on the matching cost are the support weights which are computed with respect to the center pixel of the support window. Although the per-pixel dissimilarities are shared between overlapping support windows, the support weights in general are not.

While this makes computing an actual approximation of $M(p,d)$ from any general $M(q,d)$ infeasible, it leads to a possible criterion for choosing among the candidate disparities. The support weights in the matching cost are based on the color difference between the center of the support window and the pixels in the window. Therefore, the pixels q in the neighborhood of p whose matching costs $M(q,d)$ are most likely to be similar to $M(p,d)$ will be those whose color $I(q)$ is similar to $I(p)$. Conversely, the neighborhood pixels whose colors are dissimilar to $I(p)$ will have matching costs that are not applicable to p. The criterion for choosing a candidate disparity should therefore bias toward pixels of similar color, and away from dissimilar ones. We accomplish this by applying an additional weight to the neighbors' matching costs, lowering the cost of similar pixels and penalizing dissimilar ones. This weighted matching cost $M'(q,d)$ is given as:

$$M'(q,d) = \left(1 - e^{\frac{\|I(p)-I(q)\|}{\gamma_c}}\right) \cdot M(q,d). \tag{5}$$

Note that the disparity d above is different for each $q \in W_p$, and that $M'(q,d)$ is in no way an approximation of $M(p,d)$. One potential addition to this would be a spatial weight to account for the greater support window overlap between nearby pixels. However, given our relatively small window size, this would lead to either biasing too much toward the immediate neighbors, or not having much effect at all except for the corners. Therefore, we omitted this from our current implementation.

From here, we choose the neighbor with the lowest $M'(q,d)$, and take its disparity $d = disp(q)$ as a candidate solution for p. We only now compute a new matching cost $M(p,d)$. If this new cost is lower than our old cost we accept the new disparity, otherwise we keep the old one. Of course, this selection criteria does not guarantee that the chosen disparity will truly be the best one among the neighbors. In practice however, we find that this method moves the disparity toward the correct solution, while only needing a single matching cost computation in an iteration of propagation (instead of 48). We found that spatial propagation tends to nearly converge after about four iterations.

4.3.4 Local Sweep

While spatial propagation quickly moves toward the correct disparity, it does not always converge at the correct solution. This can be simply because the exact disparity for a particular pixel does not appear as a candidate in the pixel's neighborhood, or because our approximate criteria does not select it. Also, sometimes propagation does not converge in the 4 iterations used.

However, often the solution provided by spatial propagation is near the correct one. We therefore perform a refinement by conducting a local sweep. Four more disparity levels are considered in the neighborhood of the current solution d. First, a matching cost is computed for disparities $d-1$ and $d+1$. Then, these two costs are compared and a sweep direction is chosen in the direction of the disparity with the lower cost. Finally, two more disparities are considered in that direction.

In other words, if $M(p, d-1) < M(p, d+1)$ the disparities $d-2$ and $d-3$ are considered, otherwise disparities $d+2$ and $d+3$ are tried. During this sweep, if the new matching cost is lower than the previous one, the new disparity and cost are accepted as the current disparity and cost.

At the end of the local sweep we have an initial raw semi-dense disparity map.

4.4 Post Processing

Most stereo algorithms have a post-processing step that follows the initial computation of the disparity map. The purpose of this step is to further refine the disparities. We take a fairly simple approach to post processing. The goal is to determine which of the computed disparities are likely correct. Then the rest of the image is filled in with extrapolated values.

The simplest way to determine good disparities is to run a consistency check. This involves comparing the left and right disparity map and only keeping the values for those pixels whose target points back at them, i.e., only keep correspondence pairs. This eliminates the parts of the image that are occluded in the other view, and therefore cannot have a good match. Although this works fine for most algorithms, in our case it could in principle cause errors because our search is probabilistic and it is technically possible for only one pixel of the pair to point to its match, while the other one points elsewhere. In practice, this happens extremely rarely and even then can be corrected by later refinement. Therefore, we simply run a consistency check and label pixels that are not part of a correspondence pair as invalid.

Since the invalid pixels are in the masked area, they are important so we do not want to naïvely fill them in the same way as the unmasked pixels. Instead we attempt to grow the valid disparity values into the invalid ones. This is a parallel process where each invalid pixel checks it direct neighbors and adopts the lowest valid disparity from the neighbors, this invalid pixel is now marked valid but its cost is set to maximum. Each iteration of this further grows the disparity, we settled on five iterations for our system. The operation contributes somewhat to the disparity edge fattening, but it improves the disparity map overall.

Finally, after the previous steps we can fill in the remainder of the disparity map. We assign new values to any pixels that are still left as invalid, or were not in the gradient mask. To extrapolate the disparity map we use a simple linear search across the epipolar lines. For each line (in parallel) we search first left and then right in order to record the closest valid pixels to the left and right for each pixel. We also note whether there is a masked but invalid pixel along the path from a pixel to its closest valid pixel. If there is not, we compute an interpolated disparity from the closest valid pixels. If there is a masked but invalid pixel in the path, we simply adopt the lower disparity value. This is perhaps an overly simplistic approach, and it can result in streaks in the disparity map. However, these are mainly over low gradient strength areas and therefore do not cause many artifacts in the final re-rendered image.

4.5 Temporal Propagation

In addition to the previously described intra-frame steps that compute the disparity map within a frame, we also use another step in order to propagate disparity information between successive frames. This main purpose of this step is to help with temporal coherency.

An important issue with the stereo algorithm as described above is that its probabilistic nature can cause some temporal incoherency. The base algorithm can cause the disparity map to experience changes between frames even if there was no change in the input stereo pair. This then results in flickering artifacts in the final rendering.

This is particularly noticeable if there are pixels for which the computed disparity is incorrect. While the rendering artifacts caused by disparity errors may not be very objectionable for a static image, they can become quite distracting if they flicker between frames as human vision is sensitive to motion.

In order to combat this, we propagate the disparity map from one frame to the next. This simultaneously achieves two objectives. First, it ensures that the disparity map is more stable between frames, reducing overall flicker. Second, it keeps already computed good solutions from being discarded, and gives more chances for a pixel to find its correct disparity. Thus, a pixel is not only considering disparities propagated from its neighbors, but also disparities that have been propagated across time from the pixel's previous incarnation.

In order to achieve this, we use feature based tracking to compute a pose change between successive frames. For efficiency reasons, we only track the left frame but propagate both sides based on the calibrated relative pose of the left and right camera. We use this computed pose to reproject a point cloud created from the previous frames disparity into the current frame, and render a new propagated disparity map computed for the new pose. We then compute the matching costs for this propagated disparity map given the new input stereo pair. This disparity map is then used as the initial solution for the base stereo algorithm described above, starting with the random search step.

This process greatly improves the quality of the disparity map by helping it to converge to a good solution. Even in the case of pixels that do not resolve to a good disparity, temporal propagation provides inter-frame stability. This then eliminates the flickering of rendering artifacts caused by disparity errors, making them much less noticable.

4.6 Performance and Accuracy

As described above, our algorithm performs very few matching cost computations: two for random search, four for spatial propagation, four for local sweep, and one for temporal propagation. This means that in total we only perform eleven matching cost computations per frame per masked pixel. Despite this we get accurate disparity results. Table 1 gives the timings and error rates for the Teddy and Cones pairs from the Middlebury dataset [37]. Figures 3 and 4 show the disparity maps and disparity errors.

Because of the probabilistic nature of the algorithm we have an effective disparity range of 256, even though we

TABLE 1
Per-Frame Timings and Error Rates for the Teddy and Cones Datasets

	Teddy	Cones
Timings		
Computing mask	1.99 ms	1.92 ms
Stereo matching	13.08 ms	17.04 ms
Post-processing	3.75 ms	3.20 ms
Error rate	13.95%	5.9%

Resolution is 450x375. Error rate is percentage of pixels within unoccluded masked areas with a disparity error greater than 1 pixel.

only compute the cost for 11 disparity levels. To achieve the equivalent precision a more traditional plane sweep algorithm would have to check all disparity levels and perform an order of magnitude more matching cost computations (256).

Table 2 shows a comparison of run times and error rates between our method and other stereo algorithms (both GPU and CPU-based). We compare to OpenCV's [38] implementations of CSBP (Constant Space Belief Propagation) and SGBM (Semi-Global Block Matching), the implementation of ELAS (Efficient Large-scale Stereo Matching) [39] publicly available from the authors, and published results for PatchMatch Stereo. Note that the error rates shown are for the semi-dense strong-gradient unoccluded areas of interest, although these algorithms can produce dense disparity maps. All methods except PatchMatch Stereo were run on the same system, and (where applicable) use the same number of disparity levels (256) and the same support window size (7×7). Other algorithm parameters are at their default values.

The results in Table 2 show that while our method does not achieve the accuracy of offline methods, we are competitive with other (near) real-time methods. Our error rates are slightly lower than the other fast algorithms, while our runtimes are much better. We are not only faster than state-of-the art CPU methods (SGBM and ELAS), but also faster than the GPU-based CSBP from OpenCV.

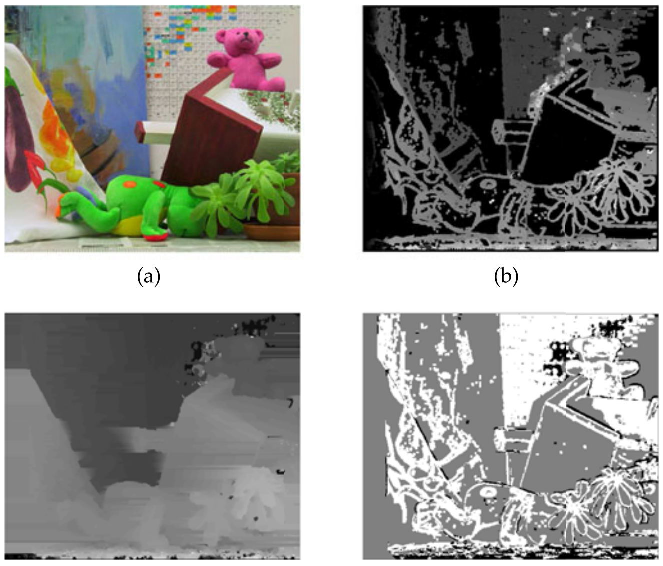
Fig. 3. Stereo matching for Teddy dataset. (a) Left input image. (b) Raw disparity. (c) Final (filled-in) disparity. (d) Disparity error: white—correct, black—error greater than 1 pixel, gray—not in mask or excluded because of occlusion.

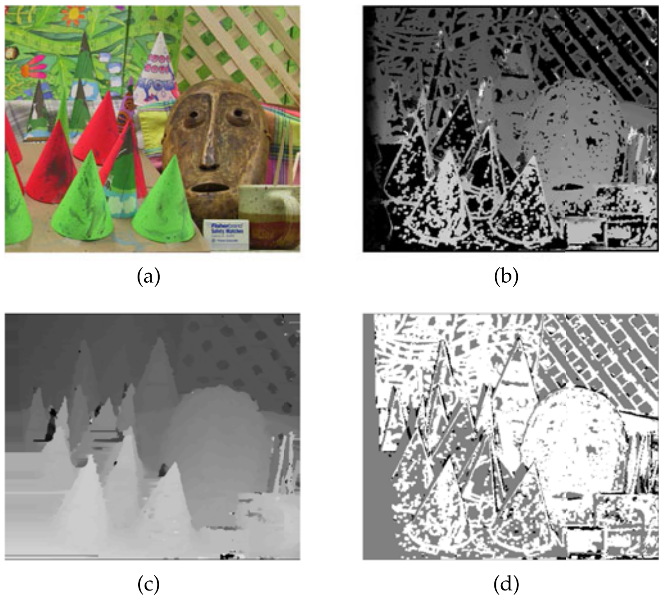
Fig. 4. Stereo matching for Cones dataset. (a) Left input image. (b) Raw disparity. (c) Final (filled-in) disparity. (d) Disparity error: white—correct, black—error greater than 1 pixel, gray—not in mask or excluded because of occlusion.

4.6.1 Prototype

In our prototype system the stereo camera has a native resolution of 1024×768, but in order to improve performance we reduce this to 512×384 for the stereo matching algorithm. We do, however, use a color image for the matching, instead of the common grayscale reduction. Although the stereo matching is at half resolution, this is upscaled back to full resolution before computing the gradient positions and calling the IBR algorithm.

5 Rendering

As mentioned above, the basic idea of the method is to create a novel view by integrating gradient images that are formed by reprojecting the gradients of the original view. Integrating a solution just from the gradients is a computationally expensive operation, even with a GPU-based parallel

TABLE 2
Comparison with other Algorithms on the Teddy and Cones Datasets

	Teddy		Cones	
	time	error	time	error
Our method	18.82 ms	13.95%	22.16 ms	5.90%
OpenCV CSBP	61.83 ms	14.62%	61.96 ms	9.03%
OpenCV SGBM	162.19 ms	16.19%	163.75 ms	7.99%
ELAS	58.98 ms	22.02%	61.13 ms	12.97%
PM Stereo	(~1 min)	(3.78%)	(~1 min)	(2.80%)

Resolution is 450x375. Error rate is percentage of pixels within unoccluded masked areas with a disparity error greater than 1 pixel.
PM Stereo values based on published results

implementation. It can take many iterations for the integration to converge to a solution, partly due to the unknown constant of integration. The method by Kopf et al. [3] uses an approximate solution (the data term) as an initial solution in order to significantly reduce the number of iterations.

The key to the approximation step is to consider that when a gradient changes position from the original view to the new view it should alter the color of the regions that it passes over. To clarify, consider a quadrilateral whose two opposing edges are the gradient's original position and the new position. This quad can be drawn over the original view, and the gradient value can be applied to the pixels that the quad covers. This may add or subtract to those pixels' value. If this process is done for all gradients, the resulting image will be very similar to what the correct image should be from the new view. For a more in-depth description, please see [3].

5.1 Performance Considerations

While developing our prototype system we aimed to strike a balance between real-time performance and good image quality. The various bottlenecks were identified through profiling, and adjustments were made to reduce the runtime while minimizing any loss of quality. Here we give some details about those considerations.

The IBR algorithm can be divided into three distinct steps that have different performance behaviors.

The first step is the rendering of the data term, which is surprisingly the most expensive. The performance hit here comes from the number and size of the quads. Each quad corresponds to a gradient, so there are twice as many quads as there are pixels (one horizontal and one vertical). Furthermore, the nature of the shifting gradients means that each quad will typically generate a large number of fragments. The cost of this step changes considerably based on the novel view position.

The second (also fastest) step is rendering the gradients images, i.e., simply reprojecting the lines of the gradients at their new positions.

Finally, the third step is the integration of the final solution from the gradients, initializing with and biasing toward the data term. This step is fairly expensive, but its runtime is mostly constant, depending mainly on the number of iterations.

The original work by Kopf et al. used a super-resolution framebuffer for rendering all the steps in the algorithm, i.e., the framebuffer size is several times larger than the input resolution. They also bias the final solution toward the approximate solution. We take a somewhat different approach. We observe that we can treat the approximate solution as simply the low frequency component of the final solution, while the reprojected gradients can provide the high frequency detail. We then use the approximate solution just as a initial solution, and do not bias towards it during the integration. This then allows us to use a much lower resolution image for our data term, since it only needs to capture low frequency information. By using a lower resolution data term we significantly improve performance. We set the data term resolution to a quarter of the regular framebuffer resolution. We also reduce the number of integration steps to five, and use a framebuffer size smaller than the original image. Although our framebuffer size (640×480) is smaller than the raw input resolution, it does not actually lower the quality of the final results. This is because the field of view of the user's frustum is usually narrower than that of the camera. As a result, the input image is effectively scaled up when shown on the magic lens and therefore still oversampled by the framebuffer.

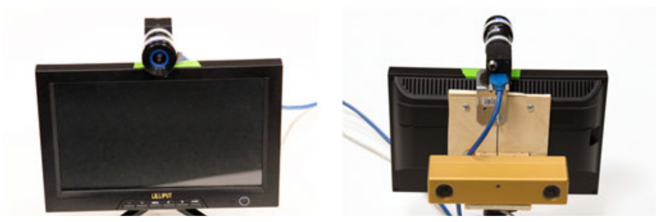
Fig. 5. Our user-perspective magic lens prototype hardware.

The final augmented image is rendered at 800×600, which is the resolution of our display. The various resolutions in our pipeline were empirically determined to give a good balance of performance and quality for our system.

6 Prototype

Our system consists of a hand-held magic lens rig tethered to a workstation. The rig, shown in Fig. 5, was built using common off-the-shelf parts. The central component of the magic lens is a Lilliput 10.1″ LCD display. We mounted a PointGrey Bumblebee2 stereo camera, used for the scene reconstruction, to the back of the display. A PointGrey Flea3 with a wide angle lens is mounted on the front and is used to track the user.

The magic lens is tethered to a workstation with two NVIDIA Quadro K5000 GPUs. These GPUs provide most of the processing power and do most of the computational work. The workstation also has 16 GB of RAM, and an Intel Core i7 CPU. The software stack of the system is built on Linux (Kubuntu 14.04), using CUDA 5.5 and OpenGL 4.3.

6.1 Calibration

In order to ensure a properly aligned view, the various components of the magic lens rig needed to be calibrated, both individually and to each other. First, using standard methods we acquired the intrinsic camera parameters for the Flea3, and the stereo camera parameters of the Bumblebee2. These parameters are loaded by our system in order to undistort the captured video. In the case of the stereo camera, the parameters are also used to rectify the input so that the epipolar lines are aligned with the horizontal axis.

Secondly, we needed to calibrate the positions and orientations of all the cameras and the display. We use the left camera of the stereo pair as our reference coordinate system. The right camera is already calibrated to the left from the stereo calibration. Calibrating the user-facing camera and the display required more effort.

Since the stereo camera and the user-facing camera are facing opposite directions, they cannot be easily calibrated by using a simple common target. There are several methods for getting the relative pose of cameras with non-overlapping views, our approach was to use a calibration environment with known geometry. We prepared a small workspace as a calibration area, covering it with coded fiducial markers. The area was arranged so that it had markers on opposing sides.

The area was captured with a calibrated camera and the relative transformations between the markers were computed, with one of them used as a reference. We then placed the magic lens rig inside the calibration area and captured simultaneous views from the front and back cameras. From the pose of each camera relative to the reference, we computed the transformation between the cameras.

With the user-facing camera calibrated to the stereo camera, the only remaining part was calibrating the display. The display was calibrated to the user-facing camera, which by extension calibrated it to the the stereo camera. We used the mirror-based method by Takahashi et al. [40]. A fiducial marker was shown on the display, while the user-facing camera observed the display through a mirror. Three of the marker's corners were defined as the reference points, and the marker was observed with three mirror poses. This gives the pose of the display relative to the user-facing camera, and therefore relative to the reference left camera as well.

6.2 Face Tracking

In order to create a user-perspective view, we need to be able to acquire the user's perspective. That is, we need to capture the position of the user's eyes relative to the display. We achieve this with face tracking, which requires a user-facing camera, available on most smartphones and tablets.

We implemented face tracking with FaceTracker [41], [42], a real-time deformable face tracking library. The library provides us with the user's head position, from which we compute an approximate 3D position for the user's "middle eye" (i.e., the point halfway between the left and right eye). Due to monocular scale ambiguity and the differences in the dimensions of human faces, this position is only approximate, but it is sufficient for our prototype. This size ambiguity can be resolved by introducing user profiles with the exact facial features of the user, and using face recognition to automatically load such a profile. We leave that for future work. Alternatively, using two or more camera's for the face tracking can also resolve the scale ambiguity.

The user tracking is implemented as a separate system running in its own process and communicating with the main software through a loopback network interface. This allows us to easily swap out the tracking system if needed. We used this feature to implement a marker based tracker for the purpose of capturing images and video of the magic lens. The images in this paper were taken by attaching a fiducial marker to the camera, and using this alternate tracker.

7 Results

Some examples of the type of results we get can be seen in Figs. 1c, 2, and Figs. 6 through 8. Fig. 2 shows the main steps of our approach for a somewhat cluttered live scene with various different features: dark areas, bright areas, textured surfaces, homogenous surfaces, specularities, and thin geometry. The stereo matching is only run on a small percentage of the image, and the filled-in disparity map is very coarse. However, the final rendering has relatively minor artifacts.

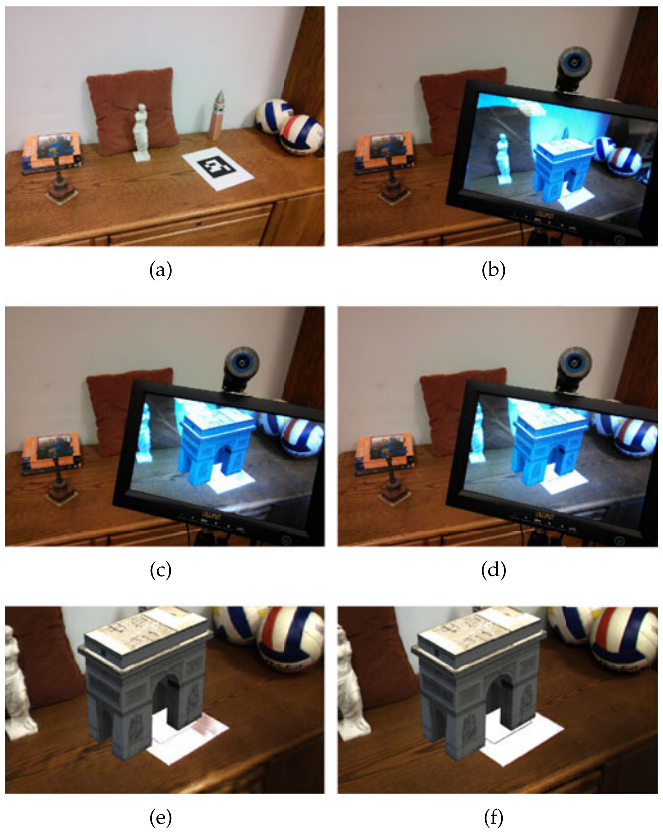

Fig. 6. Results for Arch1 scene. (a) The scene. (b) Device-perspective magic lens. (c) Our user-perspective magic lens. (d) Reference user-perspective magic lens. (e) Screengrab of our user-perspective magic lens. (f) Screengrab of reference user-perspective magic lens.

7.1 Evaluation

Figs. 6 and 7 provide a comparison of our method and system with both the traditional device-perspective magic lens approach and a ground truth reference user-perspective view. In each figure, the top row shows the unaugmented direct view of the scene (left), and the device-perspective view (right). Below is a comparison of user-perspective views, left is our method, right is the reference. The middle row show the user's view, the bottom row shows the corresponding screen capture from the magic lens display. The figures show that the view frustum for our magic lens is well aligned with the outside and the perspective of the scene matches that of the outside. Due to small inaccuracies in the prototype calibration and the viewer tracking, the frustum alignment isn't perfect, but this equally effects both our method and the reference.

The reference view was created using a reconstructed 3D model of the scene, captured in an offline pre-process. This model was acquired with a depth sensor and an implementation of the KinectFusion algorithm, which provides a state-of-the-art reconstruction of the scene. During the scene capture, the scene model is registered to the fiducial marker which allows us to later easily correctly position it in the AR view. To create the reference view, the scene model is loaded by our software and used to render the real world background (in place of the re-rendered background from our method). To get the best possible quality results, we additionally use projective texturing to texture the captured scene model with the live video from the camera. This

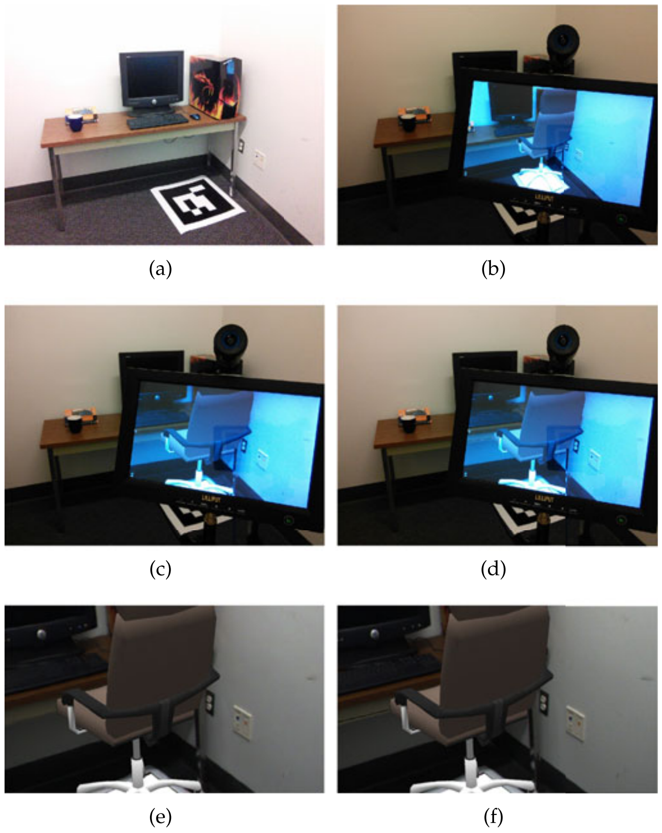

Fig. 7. Results for Office scene. (a) The scene. (b) Device-perspective magic lens. (c) Our user-perspective magic lens. (d) Reference user-perspective magic lens. (e) Screengrab of our user-perspective magic lens. (f) Screengrab of reference user-perspective magic lens.

approach provides a reference that is of near ground truth quality. This reference represents the best that can be achieved with the current state-of-the-art (however the approach only works for completely static scenes). As can be seen by the side-by-side images in Figs. 6 and 7, our method compares quite well with the ground truth reference. The image quality is good, with mostly minor rendering artifacts. Notably, we achieve these results in real time with no pre-processing of the scene.

Fig. 8 shows our magic lens working outside in the sun and with distant objects. This would not be possible for a depth-sensor based system, as they cannot work in strong sunlight and have range limits (hence, we cannot provide a reference comparison). Furthermore, the figure shows that our method also works for fully dynamic scenes with fast moving objects (cars). Dynamic scenes cannot be handled well by approaches based on scene reconstruction, such as the one used above for the reference.

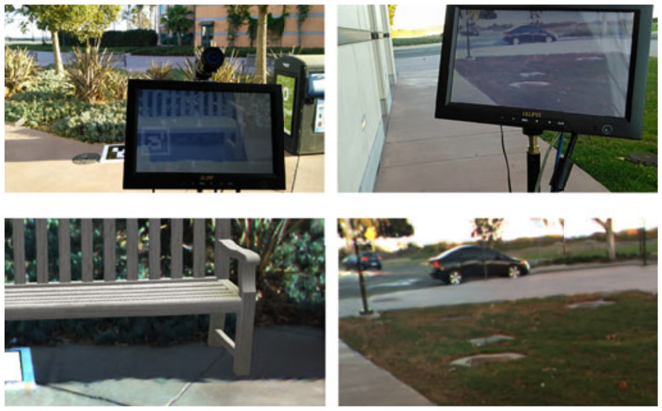

Fig. 8. Results for outdoor scenes showing examples of strong sunlight, dynamic scenes, and distant objects. Top: view through magic lens. Bottom: corresponding screen capture.

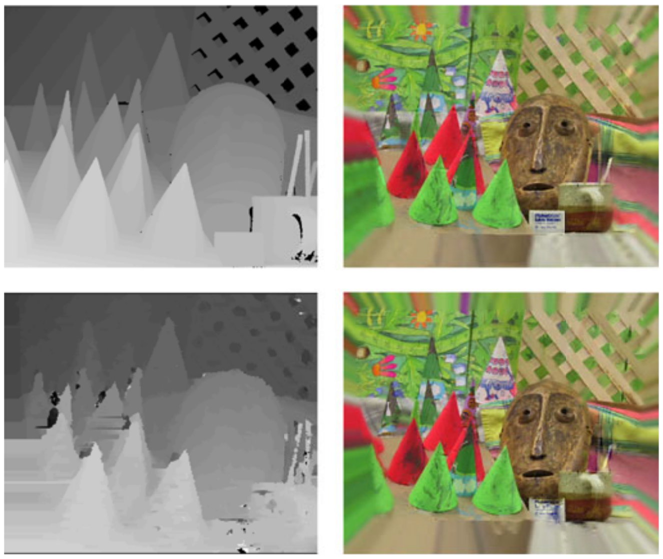

Fig. 9. Comparison between result using ground truth versus our stereo matching. Top is with ground truth, bottom is with our stereo algorithm.

7.1.1 Additional Evaluation

Figs. 9 and 10 illustrate the effects of the individual parts of our approach using the standard images from the Middlebury dataset.

The effect of using an approximate disparity map can be seen in Fig. 9. The top shows the results of our rendering when using ground truth data for the disparity map. The bottom show the results with a disparity map produced by our stereo algorithm. Despite the much coarser disparity

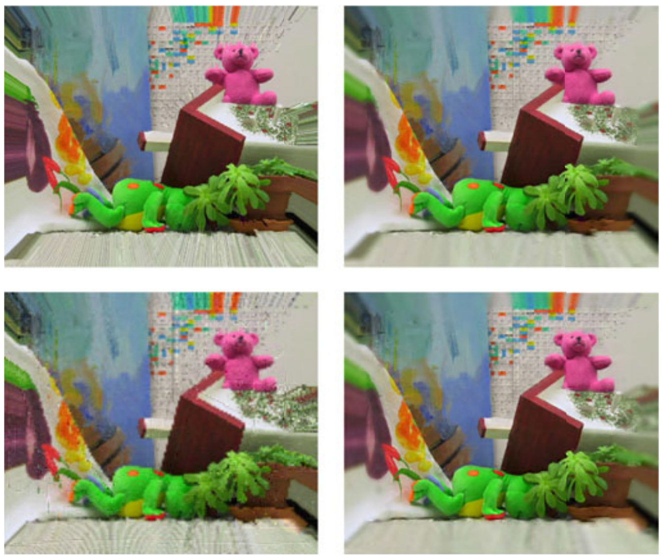

Fig. 10. Comparison between full resolution and reduced resolution. Left is data term, right is solution. Top is full resolution, bottom is reduced resolution.

map, the final results are fairly similar, with the most serious artifacts confined to the pencils in the mug.

A comparison of the rendering from using full resolution (in this case 640 × 533) images for everything, versus using a half-resolution disparity map and a quarter-resolution data term is given in Fig. 10. As can be seen, the reduced resolution does not have a significant effect on the quality of the final rendering.

7.2 System Performance and Optimizations

Compared to [1], the changes to our system and newly introduced features (in particular the adaptive support weights) add to the overall computational load. To maintain performance, some optimizations were made. Various bottlenecks were identified through profiling, and the GPU resources were better allocated. This reduced the computational cost of some steps and regained overall system performance. The average performance of the steps in the pipeline can be seen in Table 3 (note that these are wall-clock timings with instrumentation enabled). The largest aggregate cost and about half the total cost is the image-based rendering. The stereo matching is very fast at less than 7 ms. However post-processing adds another 2.23 ms, mostly for growing and filling in the disparity. Feature-based pose tracking, which is currently CPU powered, has a rather high cost of 16.32 ms. This overhead can be reduced with some optimization, and in particular by moving feature tracking to the GPUs. The pose tracking is a new step introduced to support the temporal coherency feature. If this feature is disabled, then the overall system works at about 20 FPS (which is the camera frame rate). The full featured system has an overall average framerate of about 16 FPS, on par with the previous system.

It is worth noting that the second GPU in our system is not yet fully utilized, and is mostly used for computing the disparity maps of the right image. Therefore, there is a lot of room for further performance gains. The first step toward that is to more evenly spread out the computational load across GPUs.

TABLE 3
Average Per-Frame Timings for our Prototype Implementations

Timing	(ms)
Frame total	63.88
Prepare input pair	3.69
Pose tracking	16.32
Stereo matching	6.60
Post-processing	2.23
Consistency check	0.14
Grow disparity	0.84
Fill disparity	1.25
Compute and update positions	2.42
Image-based rendering	32.38
Data term	9.86
Gradients	5.40
Merge left and right	0.91
Conjugate gradient solver	16.12
Other	0.24

Average framerate is about 16 FPS.

7.3 Discussion

Overall, our system provides quite satisfactory results but it does have some remaining challenges. From a user perspective, the challenges are issues with the view frustum, and issues with the image quality. From a technical standpoint these are caused by issues with face tracking, stereo reconstruction, rendering, and calibration.

7.3.1 View Frustum

The heart of the user-perspective magic lens problem is providing a correct view frustum for the user. While our system generally accomplishes this goal, it has some constraints.

Since it is a fully live system it can only show what the stereo cameras currently see. Although we use cameras with a fairly wide field of view, it is still possible for the user to orient the magic lens in such a way that the user's view frustum includes areas that the cameras do not see. This problem is somewhat mitigated by the fact that the best way to use a user-perspective magic lens is to hold it straight in order to get the widest view. This then keeps the desired view frustum within the region visible by the cameras. Nevertheless, this issue warrants some discussion. Currently our system simply fills in those areas using information from the known edges. A possible simple solution to this problem could be to use fisheye lenses or additional cameras in order to get a 180 degree view of the scene behind the display. In [16] the approach was to create a model of the environment and render from the model, this way the out-of-sight areas could still be rendered if they were once visible. This type of compromise approach where currently visible areas are rendered from live data, while out-of-sight areas are rendered from a model could also be a promising solution here. Since we use image-based rendering, the scene model can simply be a collection of keyframes with depth maps.

The view frustum can be slightly misaligned due to inaccuracies with the face tracking. We use a free off-the-shelf face tracker and only estimate an approximate position using a general model of the human face. Better results could be achieved by using a more robust face tracker and by using per-user face profiles.

7.3.2 Image Quality

The overall quality of our system is quite good. However, we do not yet achieve a level of quality that would be satisfactory for mainstream commercial applications. The visuals are not as clean as with systems that only approximate the user-perspective view through image warping [19], [21] but they are generally as good as the geometrically-correct user-perspective magic lens in [16].

The artifacts we get are primarily caused by errors in stereo reconstruction. In general, when the correct stereo correspondence has a higher matching cost than another incorrect correspondence, an error in the disparity map will occur. This can occur with highly specular surfaces or occlusion boundaries where the background is different between the stereo views. Another cause is when there is a low texture or periodic feature that is aligned with the epipolar lines of the stereo camera. While our use of adaptive support weights helps mitigate some problems due to occlusion boundaries, it does not entirely solve them. Issues due to specularities are common to

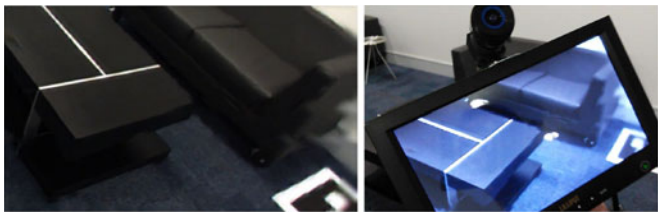

Fig. 11. Synchronized (a) screen capture and (b) and user view, showing artifacts that can occur due to failure cases. These include streaks/breaks caused by disparity errors, and blurriness at the edge of the observed scene. (Note: the bright spots on the screen are reflections of the room lights.)

stereo algorithms, as they implicitly assume that surfaces are diffuse. There have been many proposed methods for tackling the issue of specularities, and exploring those is of interest for future work.

In general, the artifacts are not very severe. They are mostly unnoticable in the weak gradient areas, and occur primarily when there is an error with the disparity of a strong gradient. Due to the nature of the gradient domain image-based rendering algorithm, any errors are usually blurred out which helps in making them less objectionable. Furthermore, the temporal propagation of depth values helps maintain temporal coherency in the rendering and so reduces the visibility of artifacts. In most cases the artifacts are either fuzzy waves along some straight edges, or occasional blurry streaks from some occlusion boundaries. In areas that are visible from the viewer's position but not seen from the cameras, the gap is filled by smooth streaks connecting the edges. Fig. 11 shows examples of these failure cases.

8 Conclusion and Future Work

We have presented a new approach to creating a geometrically-correct user-perspective magic lens, based on leveraging the gradients in the real world scene. The key to our approach is in the coupling of a recent image-based rendering algorithm with a novel semi-dense stereo matching algorithm. We have extended our previous work in this area, made improvements to our prototype device, and performed additional evaluations. These improvements include a reworked stereo algorithm, better temporal coherency, and better performance. We have compared our stereo results with those of the other algorithms, and have shown a comparison of our real-time user-perspective magic lens renderings to results from an offline reference.

In addition to making further improvements to the prototype, we would also like to evaluate the system with a formal user study. Previous user studies on user-perspective magic lenses have either been in simulation [16] or with approximations [17], [18]. We expect that with additional improvements and a better prototype we will be able to do a fair comparison between device-perspective and user-perspective magic lenses with a full real system.

Acknowledgments

This work was partially supported by NSF grant IIS-1219261 and ONR grant N00014-14-1-0133.

References

[1] D. Baričević, T. Höllerer, P. Sen, and M. Turk, "User-perspective augmented reality magic lens from gradients," in *Proc. 20th ACM Symp. Virtual Reality Softw. Tech.*, 2014, pp. 87–96.
[2] H. Shum and S. B. Kang, "Review of image-based rendering techniques," in *Proc. SPIE*, 2000, pp. 2–13.
[3] J. Kopf, F. Langguth, D. Scharstein, R. Szeliski, and M. Goesele, "Image-based rendering in the gradient domain," *ACM Trans. Graph.*, vol. 32, no. 6, Nov. 2013.
[4] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses: The see-through interface," in *Proc. 20th Annu. Conf. Comput. Graph. Interactive Tech.*, 1993, pp. 73–80.
[5] J. Viega, M. J. Conway, G. Williams, and R. Pausch, "3D magic lenses," in *Proc. 9th Annu. ACM Symp. User Interface Softw. Tech.*, 1996, pp. 51–58.
[6] M. M. Wloka and E. Greenfield, "The virtual tricorder: A uniform interface for virtual reality," in *Proc. 8th Annu. ACM Symp. User Interface and Softw. Tech.*, 1995, pp. 39–40.
[7] J. Rekimoto and K. Nagao, "The world through the computer: Computer augmented interaction with real world environments," in *Proc. 8th Annu. ACM Symp. User Interface and Softw. Tech.*, 1995, pp. 29–36.
[8] F. Zhou, H.-L. Duh, and M. Billinghurst, "Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR," in *Proc. 7th IEEE/ACM Int. Symp. Mixed Augmented Reality*, Sep. 2008, pp. 193–202.
[9] O. Bimber, B. Frohlich, D. Schmalsteig, and L. Encarnacao, "The virtual showcase," *IEEE Comput. Graph. Appl.*, vol. 21, no. 6, pp. 48–55, Nov. 2001.
[10] A. Olwal and T. Höllerer, "POLAR: Portable, optical see-through, low-cost augmented reality," in *Proc. ACM Symp. Virtual Reality Softw. Tech.*, 2005, pp. 227–230.
[11] M. Waligora, "Virtual windows: Designing and implementing a system for ad-hoc, positional based rendering," Master's thesis, Dt. Comput. Sci., Univ. New Mexico, Albuquerque, NM, USA, Apr. 2008.
[12] A. State, K. P. Keller, and H. Fuchs, "Simulation-based design and rapid prototyping of a parallax-free, orthoscopic video see-through head-mounted display," in *Proc. 4th IEEE/ACM Int. Symp. Mixed Augmented Reality*, 2005, pp. 28–31.
[13] Canon. (Mar. 2016). MREAL HMD [Online]. Available: http://www.usa.canon.com/internet/portal/us/home/products/details/mixed -reality/mreal-displays/head-mounted-display
[14] M. Mohring, C. Lessig, and O. Bimber, "Video see-through AR on consumer cell-phones," in *Proc. 3rd IEEE and ACM Int. Symp. Mixed Augmented Reality*, Nov. 2004, pp. 252–253.
[15] T. Olsson and M. Salo, "Online user survey on current mobile augmented reality applications," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2011, pp. 75–84.
[16] D. Baričević, C. Lee, M. Turk, T. Höllerer, and D. Bowman, "A hand-held AR magic lens with user-perspective rendering," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, Nov. 2012, pp. 197–206.
[17] K. Čopič Pucihar, P. Coulton, and J. Alexander, "Evaluating dual-view perceptual issues in handheld augmented reality: Device vs. user perspective rendering," in *Proc. 15th ACM Int. Conf. Multimodal Interaction*, 2013, pp. 381–388.
[18] K. Čopič Pucihar, "The use of surrounding visual context in handheld AR: Device vs. user perspective rendering," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 197–206.
[19] A. Hill, J. Schiefer, J. Wilson, B. Davidson, M. Gandy, and B. MacIntyre, "Virtual transparency: Introducing parallax view into video see-through AR," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2011, pp. 239–240.
[20] Y. Matsuda, F. Shibata, A. Kimura, and H. Tamura, "Poster: Creating a user-specific perspective view for mobile mixed reality systems on smartphones," in *Proc. IEEE Symp. 3D User Interfaces*, Mar. 2013, pp. 157–158.
[21] M. Tomioka, S. Ikeda, and K. Sato, "Approximated user-perspective rendering in tablet-based augmented reality," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2013, pp. 21–28.
[22] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vis.*, vol. 47, no. 1/3, pp. 7–42, 2002.
[23] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in *Proc. 3rd Int. Symp. 3D Data Process., Vis., Transmission*, Jun. 2006, pp. 798–805.

[24] W. Yu, T. Chen, F. Franchetti, and J. Hoe, "High performance stereo vision designed for massively data parallel platforms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 11, Nov. 2010, pp. 1509–1519.
[25] K. Zhang, J. Lu, Q. Yang, G. Lafruit, R. Lauwereins, and L. Van Gool, "Real-time and accurate stereo: A scalable approach with bitwise fast voting on CUDA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 7, Jul. 2011, pp. 867–878.
[26] J. Kowalczuk, E. Psota, and L. Perez, "Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, Jan. 2013, pp. 94–104.
[27] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A randomized correspondence algorithm for structural image editing," *ACM Trans. Graph.*, vol. 28, no. 3, 2009, Art. no. 24.
[28] M. Bleyer, C. Rhemann, and C. Rother, "PatchMatch stereo—Stereo matching with slanted support windows," in *Proc. Brit. Mach. Vis. Conf.* 2011, pp. 14.1–14.11.
[29] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera," in *Proc. IEEE Int. Symp. Mixed Augmented Reality*, Oct. 2013, pp. 83–88.
[30] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. 23rd Annu. Conf. Comput. Graph. Interactive Tech.*, 1996, pp. 31–42.
[31] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proc. 25th Annu. Conf. Comput. Graph. Interactive Tech.*, 1998, pp. 231–242.
[32] P. E. Debevec, C. J. Taylor, and J. Malik, "Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach," in *Proc. 23rd Annu. Conf. Comput. Graph. Interactive Tech.*, 1996, pp. 11–20.
[33] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 313–318, Jul. 2003.
[34] K.-J. Yoon and I.-S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognition*, 2005, pp. 924–931.
[35] A. Hosni, M. Bleyer, and M. Gelautz, "Secrets of adaptive support weight techniques for local stereo matching," *Comput. Vis. Image Understanding*, vol. 117, no. 6, 2013, pp. 620–632.
[36] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 2, pp. 504–511, Feb. 2013.
[37] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognition*, Jun. 2003, pp. 195–202.
[38] OpenCV (2015) [Online]. Available: opencv.org
[39] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proc. 10th Asian Conf. Comput. Vis.*, 2010, pp. 25–38.
[40] K. Takahashi, S. Nobuhara, and T. Matsuyama, "A new mirror-based extrinsic camera calibration using an orthogonality constraint," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognition*, 2012, pp. 1051–1058.
[41] J. M. Saragih, S. Lucey, and J. Cohn, "Face alignment through subspace constrained mean-shifts," in *Proc. Int. Conf. Comput. Vis.*, Sep. 2009, pp. 1034–1041.
[42] J. Saragih and K. McDonald. (Jun. 2014). FaceTracker [Online]. Available: facetracker.net

Domagoj Baričević received the dipl ing degrees in computer science and electrical engineering from the University of Split, Croatia. He is currently a graduate student in computer science at the University of California, Santa Barbara. His research interests include computer graphics, augmented/virtual reality, and interactive systems. He received the UCSB Chancellor's Fellowship and has interned at Citrix Online, and Qualcomm Research. He is a student member of the IEEE.

Tobias Höllerer received the graduate degree in informatics from the Technical University of Berlin, and the MS and PhD degrees in computer science from the Columbia University. He is currently a professor of computer science at the University of California, Santa Barbara, where he co-directs the UCSB Four Eyes Lab. He is (co-)author of more than 150 peer-reviewed journal and conference publications in the areas of augmented and virtual reality, information visualization, 3D displays and interaction, mobile and wearable computing, and social computing. He received the NSF CAREER Award, for his work on "Anywhere Augmentation". He is a senior member of the IEEE.

Pradeep Sen received the BS degree in computer and electrical engineering from the Purdue University, and the MS and PhD degrees in electrical engineering from the Stanford University. He is an associate professor in the UCSB MIRAGE Lab in the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His research interests include algorithms for image synthesis, computational image processing, and computational photography. He is a co-author of more than 30 technical publications, and has been awarded more than $2.2 million in research funding, including an NSF CAREER award in 2009. He is a senior member of the IEEE.

Matthew Turk received the PhD degree from the Massachusetts Institute of Technology in 1991 and has worked at the Martin Marietta Aerospace, LIFIA/ENSIMAG (Grenoble, France), Teleos Research, and Microsoft Research. He is currently a professor of computer science and former chair of Media Arts and Technology at UC Santa Barbara, where he co-directs the UCSB Four Eyes Lab. He is on the editorial board of the *ACM Transactions on Intelligent Interactive Systems* and the *Journal of Image and Vision Computing*. He is a fellow of the the IAPR and received the 2011-2012 Fulbright-Nokia Distinguished Chair in Information and Communications Technologies. He is a fellow of the IEEE.