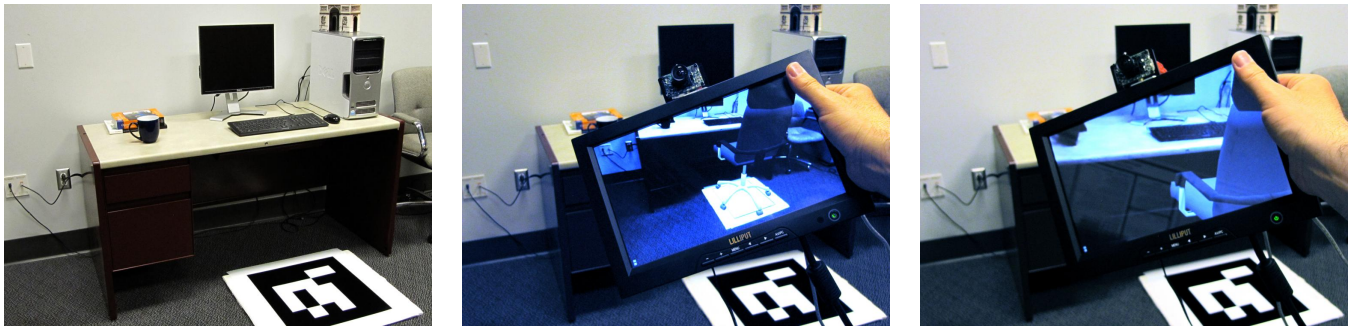
User-Perspective Augmented Reality Magic Lens From Gradients

Domagoj Baričević* Tobias Höllerer† Pradeep Sen‡ Matthew Turk§
University of California, Santa Barbara



(a) (b) (c)

Figure 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.*

Abstract

In this paper we present a new approach to creating a geometrically-correct user-perspective magic lens and a prototype device implementing the approach. Our prototype uses just standard color cameras, with no active depth sensing. We achieve this by pairing a recent gradient domain image-based rendering method with a novel semi-dense stereo matching algorithm inspired by PatchMatch. Our stereo algorithm is simple but fast and accurate within its search area. The resulting system is a real-time magic lens that displays the correct user perspective with a high-quality rendering, despite the lack of a dense disparity map.

CR Categories: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.7 [Image Processing and Computer Vision]: Scene Analysis—Stereo

Keywords: 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 Figure 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 (Figure 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 (Figure 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

*e-mail: domagoj@cs.ucsb.edu
†e-mail: holl@cs.ucsb.edu
‡e-mail: psen@ece.ucsb.edu
§e-mail: mturk@cs.ucsb.edu

VRST 2014, November 11 – 13, 2014, Edinburgh, Scotland, UK.

ACM 978-1-4503-3253-8/14/11 $15.00
http://dx.doi.org/10.1145/2671015.2671027

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 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 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) [Shum and Kang 2000]. 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 [Kopf et al. 2013] 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.

2 Related Work

The "magic lens" metaphor was first introduced by Bier et al. at Xerox PARC [Bier et al. 1993] 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. This window acts like an information filter that can reveal hidden objects, alter the visualization of data, or otherwise modify the view within the region that the window covers.

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 [Viega et al. 1996]. The Virtual Tricorder [Wloka and Greenfield 1995] 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. [Rekimoto and Nagao 1995] 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. The video from the camera is captured, augmented, and then displayed on the TV. This hand-held video-see-through approach soon became the norm for Augmented Reality interfaces [Zhou et al. 2008]. Optical see-through AR approaches (e.g. [Bimber et al. 2001; Olwal and Höllerer 2005; Waligora 2008] 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 [State et al. 2005; Canon 2014]. 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 [Mohring et al. 2004; Olsson and Salo 2011]. 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 [Baričević et al. 2012] or spatially constrained [Čopič Pucihar et al. 2013; Čopič Pucihar et al. 2014] 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 [Hill et al. 2011; Matsuda et al. 2013; Tomioka et al. 2013]; 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 [Baričević et al. 2012]. That prototype system was built using a Kinect depth sensor and a Wiimote. The Wiimote is used to track goggles worn by the user in order to obtain the 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. Firstly, the system does not gracefully handle dynamic scenes as the scene is rendered in two layers with different real time characteristics. One layer is rendered from the live Kinect stream and updates immediately, the other is rendered from a volumetric scene model that updates more slowly. Secondly, 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 [Scharstein and Szeliski 2002]. In recent years, a number of algorithms have been proposed that take advantage of GPU hardware to achieve real-time performance [Wang et al. 2006; Yu et al. 2010; Zhang et al. 2011; Kowalczuk et al. 2013]. 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

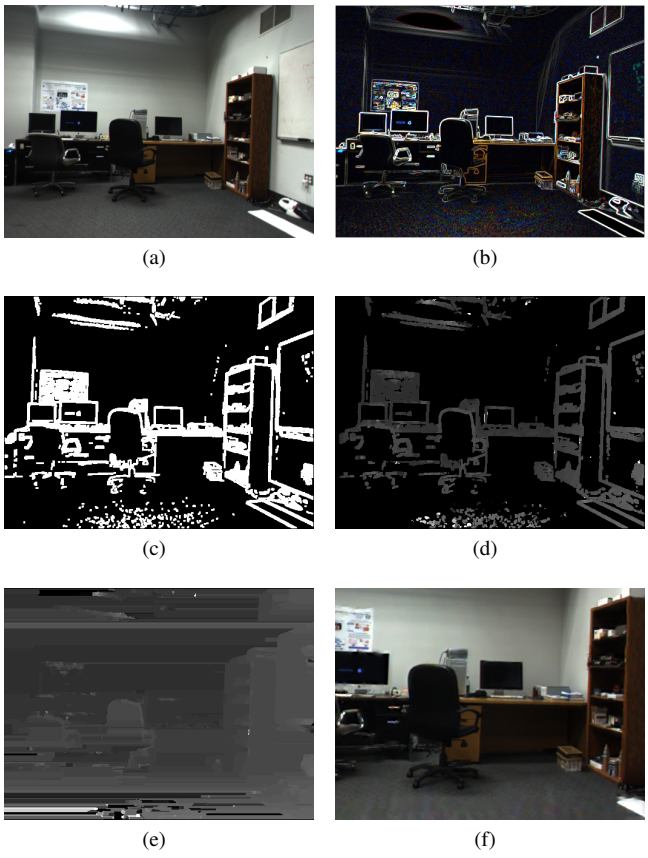


Figure 2: *The steps to rendering a novel view: (a) input image, (b) gradient magnitudes of input, (c) mask of strongest gradients, (d) disparity map for masked area, (e) filled-in disparity map, (f) final solution. (Note: (a) - (e) are for left camera, (f) is for final pose)*

[Barnes et al. 2009], an iterative probabilistic algorithm for finding dense image correspondences. PatchMatch is a general algorithm and has been applied to the field of stereo matching before. [Bleyer et al. 2011] proposed a stereo matching algorithm based on PatchMatch primarily designed to support matching slanted surfaces, although it also supports front facing planes. In [Pradeep et al. 2013] 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.

Imaged-based rendering techniques create novel views of a scene from existing images [Shum and Kang 2000]. These novel views can be rendered either purely from input image data [Levoy and Hanrahan 1996], or by using some form of geometry [Shade et al. 1998; Debevec et al. 1996]. Our approach is based on the gradient-domain image-based rendering work by Kopf et al. [2013]. 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 [Pérez et al. 2003] 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. [2013]. 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 [Barnes et al. 2009]. 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 Figure 2. The input to the pipeline is a stereo pair (Figure 2a shows left image) and a desired position for the novel view.

The first step (Figure 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 (Figure 2c).

Next, our stereo matching algorithm is run over the masked pixels. This results in a semi-dense disparity map (Figure 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 (Figure 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 (Figure 2f).

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 (or SIMD computing in general) 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 SIMD 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. 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.

4.2 Stereo matching

Now that we have a mask of the strong gradients in the image, we can run stereo matching on them. 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 two main steps: Random Search and Propagation. The full algorithm is run for a number of iterations, and in each iteration each step is iterated as well. Each iteration of each step is fully parallel at the individual pixel level. Only the steps themselves and their iterations are serialized.

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 is run, the disparities are initialized to zero, while the costs are initialized to the maximum possible value. In our implementation we use unsigned 8-bit values to store the disparities, with a disparity range of [0, 255]. The costs are stored as unsigned 16-bit values, giving a range of [0, 2^{16}-1]. The upper limit is above the maximum possible value that can be returned as a matching cost, so initializing the cost to 0xffff simplifies the search for the minimum cost disparity since there is no need to treat the first candidate disparity differently from the rest.

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.

The way the random disparity is generated requires some discussion. Regular PatchMatch [Barnes et al. 2009] 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. Firstly, the initialization and random search is a single unified step. Secondly, 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 index masks 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 index mask 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. For the matching cost we use the standard sum of absolute differences over a 7×7 support window. This process can be iterated, in our current implementation we run two iterations.

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 simpler approach to the propagation step. Instead of propagating serially through the entire image, we have each pixel in parallel check its neighborhood. Instead of computing another matching cost for each of its neighbor's disparities, the pixel uses the neighbor's cost as a proxy for what the cost would be for this pixel if it had the same disparity. The idea behind this is that if our disparity is the same as that of our neighbor, our matching costs will likely be very similar as well. We chose the neighbor with the lowest cost and take its disparity as a candidate solution, we only now compute a new matching cost. If this new cost is lower than our old cost we accept the new disparity, otherwise we keep the old one. This means that each iteration of the propagation step only does one matching cost computation. In our current implementation we run three iterations of the propagation step.

4.3 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 well for a standard plane sweep algorithm, in our case it could cause errors because our search is probabilistic and there is no guarantee that pixels that are unoccluded and belong to a correspondence pair will point to each other. It is possible for only one pixel of the pair to point to its match, while the other one points elsewhere. To help with this we run a step prior to the consistency check. For each pixel p in a disparity map we check if its match $p' = q$ points back. If it does not, we compare the matching costs of the two pixels. Since the matching cost is symmetric it should be the same (and minimal) for a correspondence pair. If q has a higher matching cost then p, we set its match q' to p and set the cost. This will always create a better solution. This process is run in both left-to-right and right-to-left directions. After this step, we run a traditional consistency check. Pixels that are not part of a correspondence pair are labeled 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 pixels 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. From each pixel (again in parallel) we search left and right for the first pixel that is valid. We look at the two disparity values and adopt the lower one (taking the lower value instead of interpolating helps prevent occlusion edges from bleeding into occluded areas). This is perhaps an overly simplistic approach, and it does result in considerable streaks in the disparity map. However, these streaks are mainly over low gradient strength areas and therefore do not cause many artifacts in the final re-rendered image.

4.4 Performance and accuracy

Through experimentation with the number of iterations of our stereo algorithm, we settled on two overall iterations, each doing two iterations of search and three iterations of propagation. This means that in total we only perform ten 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 [Scharstein and Szeliski 2003]. 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 only compute the cost for ten disparity levels. To achieve the equivalent precision a plane sweep algorithm would have to check all disparity levels and perform an order of magnitude more matching cost computations (256). Even if the plane sweep skipped over unmasked areas, it would not significantly reduce the runtime because of the GPU code path divergence problem mentioned above.

Table 1: *Per-frame timings and error rates for the Teddy and Cones datasets. The resolution of the input images and the disparity maps is 450x375. The error rate is the percentage of pixels within unoccluded masked areas with a disparity error greater than 1 pixel.*

	Teddy	Cones
Timings		
Computing mask	2.98 ms	3.18 ms
Stereo matching	12.59 ms	16.82 ms
Post-processing	1.92 ms	1.57 ms
Error rate	15.47%	7.52%

Prototype In our prototype system the stereo camera has a native resolution of 1024x768, but in order to improve performance we reduce this to 512x384 for the stereo matching algorithm. We do, however, use the full-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 implementation. It can take many iterations for the integration to converge to a solution, partly due

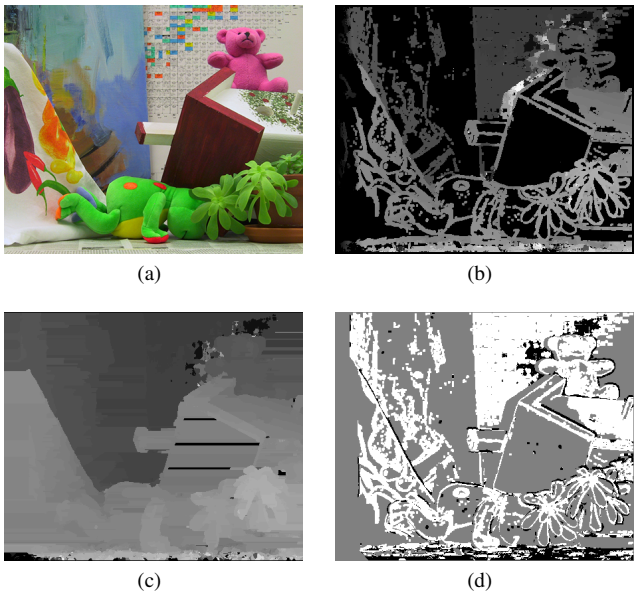


Figure 3: *Stereo matching for Teddy dataset. (a) Left input image. (b) Raw disparity. (c) Final (filled-in) disparity. (d) Disparity error: white - correct, black - error greater than 1 pixel, gray - not in mask or excluded because of occlusion.*

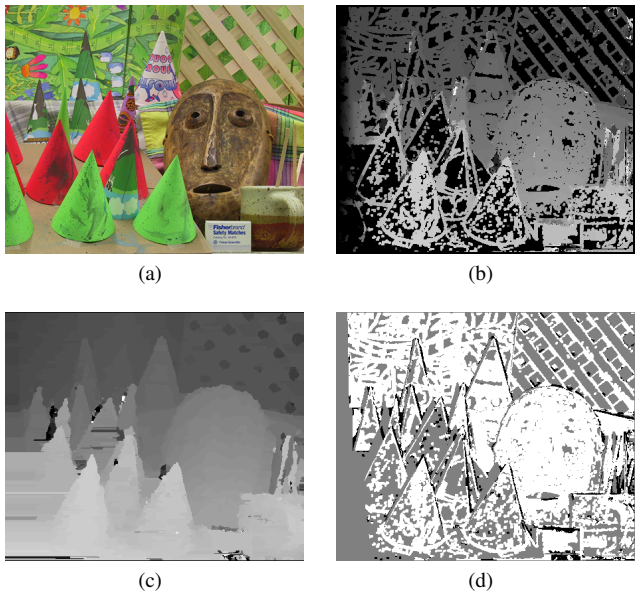


Figure 4: *Stereo matching for Cones dataset. (a) Left input image. (b) Raw disparity. (c) Final (filled-in) disparity. (d) Disparity error: white - correct, black - error greater than 1 pixel, gray - not in mask or excluded because of occlusion.*

to the unknown constant of integration. The method by Kopf et al. [2013] 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 [Kopf et al. 2013].

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 run-time 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 (640x480) 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.

The final augmented image is rendered at 800x600, 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 Figure 5, was built using common off-the-shelf parts. The central component of the magic lens is a

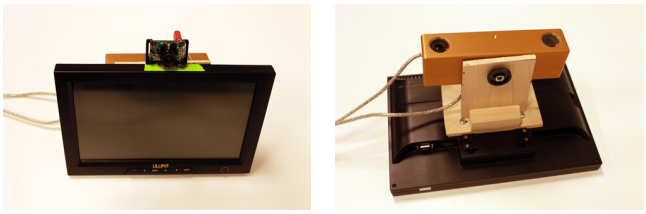


Figure 5: *Our user-perspective magic lens prototype hardware.*

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 Firefly MV 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 an NVIDIA Quadro 6000 GPU. This GPU provides most of the processing power and does most of the computational work. The workstation also has 32GB of RAM, and two dual-core AMD Opteron CPUs. The software stack of the system is built on Linux (Kubuntu 12.04), using CUDA 5.0 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. Firstly, using standard methods we acquired the intrinsic camera parameters for the Firefly, 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 common target. There are methods that can achieve this using mirrors, but we opted for a simpler approach. 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 did this by displaying a fiducial marker on the display, placing the rig in the calibration area, and using both the user-facing camera and an external camera. The setup consisted of having the user facing camera see some of the markers in the calibration area, with the external camera simultaneously seeing those same markers as well as the marker on the display. The external camera gives the relative pose of the display to the common markers, and the user-facing camera gives the common markers pose relative to itself. Combined, 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. Indeed, the recent Amazon Fire Phone features face tracking as a core element, implemented through the use of four user-facing cameras.

We implemented face tracking with FaceTracker [Saragih et al. 2009; Saragih and McDonald 2014], 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 and accompanying videos were taken by attaching a fiducial marker to the cameras, and using this alternate tracker.

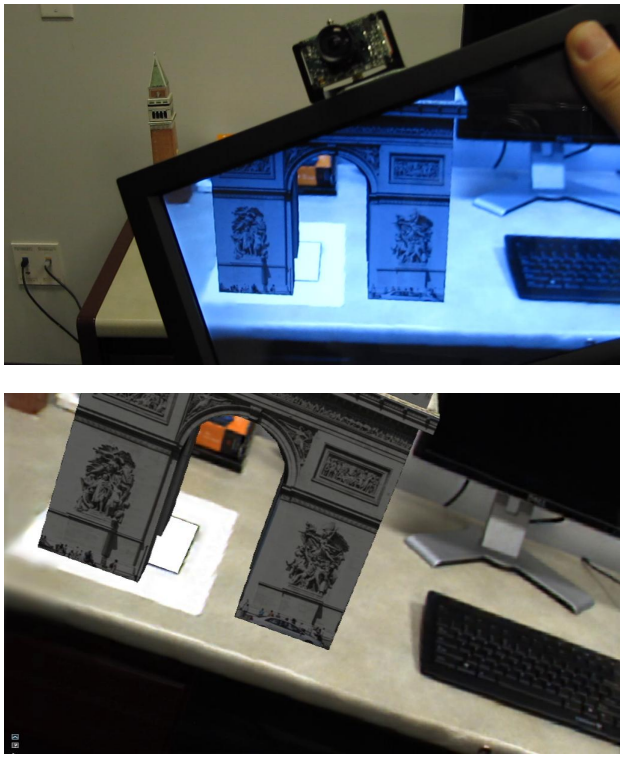


Figure 6: *Example of final result. Top: user's view showing correct user-perspective and good alignment of the view frustum. Bottom: corresponding screen capture showing good quality image with minimal rendering artifacts.*

7 Results

Some examples of the type of results we get can be seen in Figures 1a, 2, and 6.

Figure 6 shows a simple example of an AR scene, with both the user's view (top) and the corresponding screen capture from the magic lens display (bottom). The view frustum inside the magic lens is well aligned with the outside and the perspective of the scene matches that of the outside. The screen capture taken at the same moment shows that the image quality of the magic lens view is quite good with only minimal rendering artifacts.

Figure 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.

7.1 Evaluation and performance

Figures 7 and 8 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 Figure 8. 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 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 640x533) images for everything, versus using a half-resolution disparity map and a quarter-resolution data term is given in Figure 7. As can be seen, the reduced resolution does not have a significant effect on the quality of the final rendering.

Table 2: *Average per-frame timings for our prototype implementations. Average framerate is about 16 FPS.*

Timing	(ms)
Frame total	62.32
Prepare input pair	3.11
Stereo matching	7.92
Post-processing	3.73
Consistency check	0.18
Grow disparity	0.81
Fill disparity	2.74
Compute and update positions	7.34
Image-based rendering	36.33
Data term	13.66
Gradients	8.04
Merge left and right	1.08
Conjugate gradient solver	13.55
Other	3.89

The performance of our final system across the various steps in our pipeline can be seen in Table 2. The system has an overall average framerate of 16 FPS. The largest aggregate cost and about half the total cost is the image-based rendering. The stereo matching is very fast at less than 8ms. However post-processing adds another 3.7ms, most of which is spent on filling in the disparity. This is a very simple step, but it is not yet optimized and performs poorly if the masked regions are too sparse. Another unexpectedly high cost at over 7ms is the computing and updating of the 3D positions of the gradients. This is likely because this step makes OpenGL and CUDA synchronize which forces all GPU operations to complete, it also imposes a synchronization with the CPU.

7.2 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.

Figure 7: *Comparison between full resolution and reduced resolution. Left is data term, right is solution. Top is full resolution, bottom is reduced resolution.*

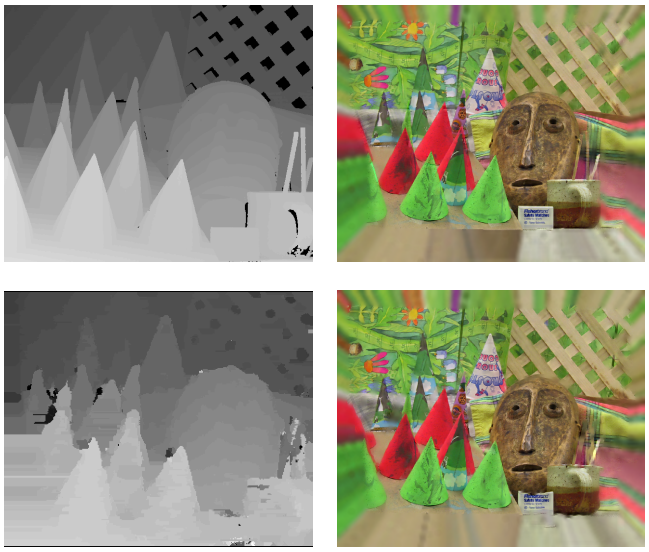


Figure 8: *Comparison between result using ground truth versus our stereo matching. Top is with ground truth, bottom is with our stereo algorithm.*

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.

Firstly, 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° view of the scene behind the display. In [Baričević et al. 2012] 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.

Secondly, our system has some noticeable latency. The latency is a compound of the latency from the face tracking, the latency from the pre-processing of the stereo video, and the latency from the frame rate. This latency causes a lag in correctly aligning the view frustum, most noticeable when the user makes fast motions or does long sweeps with the device.

Thirdly, 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.

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 [Hill et al. 2011; Tomioka et al. 2013] but they are generally as good as the geometrically-correct user-perspective magic lens in [Baričević et al. 2012].

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. These problems are common to local stereo algorithms, especially when using the simple sum of absolute differences as the matching cost. There have been many proposals for matching costs that can help address some of these issues. Our stereo matching algorithm is agnostic to the matching cost used, so exploring these alternative costs is of definite 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. 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.

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. Our stereo algorithm is fast and accurate in the areas of interest. The use of image-based rendering provides us with good imagery, even with limited scene model detail. Based on this approach we built a prototype device using common off-the-shelf hardware.

In addition to the various possible improvements to the system 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 [Baričević et al. 2012] or with approximations [Čopič Pucihar et al. 2013; Čopič Pucihar et al. 2014]. We hope to be able to do a fair comparison between device-perspective and user-perspective magic lenses with a full real system.

9 Acknowledgements

D. B. would like to thank UCSB for the Chancellor's Fellowship award, which provided funding. This work was partially supported by NSF grants IIS-1219261 and IIS-0747520, as well as ONR grant N00014-14-1-0133.

References

BARIČEVIĆ, D., LEE, C., TURK, M., HÖLLERER, T., AND BOWMAN, D. 2012. A hand-held AR magic lens with user-perspective rendering. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, 197–206.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Trans. Graph. 28*, 3.

BIER, E. A., STONE, M. C., PIER, K., BUXTON, W., AND DEROSE, T. D. 1993. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '93, 73–80.

BIMBER, O., FROHLICH, B., SCHMALSTEIG, D., AND ENCARNACAO, L. 2001. The Virtual Showcase. *Computer Graphics and Applications, IEEE 21*, 6 (Nov), 48–55.

BLEYER, M., RHEMANN, C., AND ROTHER, C. 2011. PatchMatch Stereo - Stereo Matching with Slanted Support Windows. In *Proceedings of the British Machine Vision Conference*, BMVA Press, 14.1–14.11.

CANON, 2014. Canon Mixed Reality. `http://usa.canon.com/cusa/office/standard_display/Mixed_Reality_Product`, accessed June 2014.

ČOPIČ PUCIHAR, K., COULTON, P., AND ALEXANDER, J. 2013. Evaluating Dual-view Perceptual Issues in Handheld Augmented Reality: Device vs. User Perspective Rendering. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ACM, New York, NY, USA, ICMI '13, 381–388.

ČOPIČ PUCIHAR, K., COULTON, P., AND ALEXANDER, J. 2014. The Use of Surrounding Visual Context in Handheld AR: Device vs. User Perspective Rendering. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, New York, NY, USA, CHI '14, 197–206.

DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-based Approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 11–20.

HILL, A., SCHIEFER, J., WILSON, J., DAVIDSON, B., GANDY, M., AND MACINTYRE, B. 2011. Virtual transparency: introducing parallax view into video see-through AR. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2011*, 239–240.

KOPF, J., LANGGUTH, F., SCHARSTEIN, D., SZELISKI, R., AND GOESELE, M. 2013. Image-based Rendering in the Gradient Domain. *ACM Trans. Graph. 32*, 6 (Nov.), 199:1–199:9.

KOWALCZUK, J., PSOTA, E., AND PEREZ, L. 2013. Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences. *Circuits and Systems for Video Technology, IEEE Transactions on 23*, 1 (Jan), 94–104.

LEVOY, M., AND HANRAHAN, P. 1996. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '96, 31–42.

MATSUDA, Y., SHIBATA, F., KIMURA, A., AND TAMURA, H. 2013. Poster: Creating a user-specific perspective view for mobile mixed reality systems on smartphones. In *3D User Interfaces (3DUI), 2013 IEEE Symposium on*, 157–158.

MOHRING, M., LESSIG, C., AND BIMBER, O. 2004. Video see-through AR on consumer cell-phones. In *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on*, 252–253.

OLSSON, T., AND SALO, M. 2011. Online user survey on current mobile augmented reality applications. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2011*, 75–84.

OLWAL, A., AND HÖLLERER, T. 2005. POLAR: Portable, Optical See-through, Low-cost Augmented Reality. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, NY, USA, VRST '05, 227–230.

PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson Image Editing. *ACM Trans. Graph. 22*, 3 (July), 313–318.

PRADEEP, V., RHEMANN, C., IZADI, S., ZACH, C., BLEYER, M., AND BATHICHE, S. 2013. MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, 83–88.

REKIMOTO, J., AND NAGAO, K. 1995. The World Through the Computer: Computer Augmented Interaction with Real World Environments. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, ACM, New York, NY, USA, UIST '95, 29–36.

SARAGIH, J., AND MCDONALD, K., 2014. FaceTracker. `facetracker.net`. accessed 1 June 2014.

SARAGIH, J. M., LUCEY, S., AND COHN, J. 2009. Face Alignment through Subspace Constrained Mean-Shifts. In *International Conference of Computer Vision (ICCV)*.

SCHARSTEIN, D., AND SZELISKI, R. 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision 47*, 1-3, 7–42.

SCHARSTEIN, D., AND SZELISKI, R. 2003. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, I–195–I–202 vol.1.

SHADE, J., GORTLER, S., HE, L.-W., AND SZELISKI, R. 1998. Layered Depth Images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 231–242.

SHUM, H., AND KANG, S. B., 2000. Review of image-based rendering techniques.

STATE, A., KELLER, K. P., AND FUCHS, H. 2005. Simulation-Based Design and Rapid Prototyping of a Parallax-Free, Orthoscopic Video See-Through Head-Mounted Display. In *Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality*, IEEE Computer Society, Washington, DC, USA, ISMAR '05, 28–31.

TOMIOKA, M., IKEDA, S., AND SATO, K. 2013. Approximated user-perspective rendering in tablet-based augmented reality. In *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on*, 21–28.

VIEGA, J., CONWAY, M. J., WILLIAMS, G., AND PAUSCH, R. 1996. 3D magic lenses. In *Proceedings of the 9th annual ACM symposium on user interface software and technology*, ACM, New York, NY, USA, UIST '96, 51–58.

WALIGORA, M. 2008. *"Virtual Windows: Designing and Implementing a System for Ad-hoc, Positional Based Rendering"*. Master's thesis, University of New Mexico, Department of Computer Science.

WANG, L., LIAO, M., GONG, M., YANG, R., AND NISTER, D. 2006. High-Quality Real-Time Stereo Using Adaptive Cost Aggregation and Dynamic Programming. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, 798–805.

WLOKA, M. M., AND GREENFIELD, E. 1995. The Virtual Tricorder: A Uniform Interface for Virtual Reality. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, ACM, New York, NY, USA, UIST '95, 39–40.

YU, W., CHEN, T., FRANCHETTI, F., AND HOE, J. 2010. High Performance Stereo Vision Designed for Massively Data Parallel Platforms. *Circuits and Systems for Video Technology, IEEE Transactions on 20*, 11 (Nov), 1509–1519.

ZHANG, K., LU, J., YANG, Q., LAFRUIT, G., LAUWEREINS, R., AND VAN GOOL, L. 2011. Real-Time and Accurate Stereo: A Scalable Approach With Bitwise Fast Voting on CUDA. *Circuits and Systems for Video Technology, IEEE Transactions on 21*, 7 (July), 867–878.

ZHOU, F., DUH, H.-L., AND BILLINGHURST, M. 2008. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, 193–202.