Resolving Occlusion in Augmented Reality

Matthias M. Wloka* and Brian G. Anderson*
Science and Technology Center for Computer Graphics and Scientific Visualization,
Brown University Site

Abstract

Current state-of-the-art augmented reality systems simply overlay computer-generated visuals on the real-world imagery, for example via video or optical see-through displays. However, overlays are not effective when displaying data in three dimensions, since occlusion between the real and computer-generated objects is not addressed.

We present a video see-through augmented reality system capable of resolving occlusion between real and computer-generated objects. The heart of our system is a new algorithm that assigns depth values to each pixel in a pair of stereo video images in near-real-time. The algorithm belongs to the class of stereo matching algorithms and thus works in fully dynamic environments. We describe our system in general and the stereo matching algorithm in particular.

Keywords: real-time, stereo matching, occlusion, augmented reality, interaction, approximation, dynamic environments

1 Introduction

Augmented reality systems enhance the user's vision with computer-generated imagery. To make such systems and their applications effective, the synthetic or *virtual* imagery needs to blend convincingly with the real images. Towards this goal, researchers study such areas as minimizing object registration errors [2] and overall system lag [2] [17] so as to increase the "realness" of virtual objects.

Since occlusion provides a significant visual cue to the human perceptual system when displaying data in three dimensions, proper occlusion resolution between real and virtual objects is highly desirable in augmented reality systems. However, solving the occlusion problem for augmented reality is challenging: little is known about the real world we wish to augment. For example, in an optical see-through head-mounted display (HMD), no information at all is available about the surrounding real world. In a video see-through HMD, however, at least a pair of 2D intensity bitmaps is available in digital memory.

*Box 1910, Department of Computer Science, Brown University, Providence, RI 02912. Phone: (401) 863 7600, email: {mmw|bga}@cs.brown.edu.

Typical augmented reality scenarios further complicate the problem. Because they do not restrict the real environment to be static, precomputing depth maps to resolve occlusion is impossible. As a result, occlusion between virtual and real objects needs to be determined for every frame generated, i.e., at real-time rates.

We introduce here a video see-through system capable of resolving occlusion between real and virtual objects at close to real-time rates. We achieve these near-real-time rates by computing depth maps for the left and right views at half the original video image resolution (depth maps are thus 320 by 240 pixels). Few additional assumptions are made on the real and virtual environments; in particular, both can be fully dynamic.

The heart of our video see-through system is a new stereo matching algorithm that infers dense depth maps from a stereo pair of intensity bitmaps. This new algorithm trades accuracy for speed and thus outperforms known stereo matching algorithms except for those that run on custom-built hardware. Furthermore, our algorithm is robust: it does not require a fully calibrated pair of stereo cameras.

1.1 Overview

We briefly review related work in Section 2. In Section 3 we outline the architecture of our video see-through augmented reality system. The basic algorithm for stereo matching video images in close to real-time is explained in Section 4. Several extensions, described in Section 5, make the basic algorithm faster and more robust. Finally, in Section 6 we discuss drawbacks of the algorithm and propose possible future work.

2 Related Work

While several other augmented reality systems are described in the literature [3] [7] [9], none of these systems addresses the occlusion problem. We know of only one augmented reality system other than our own that attempts to correct this deficiency [10]. The envisioned application of the competing project [10], i.e. virtual teleconferencing, is more ambitious than our own, i.e. augmented interior design, but the basis of both systems is to compute dense depth maps for the surrounding real world. Preliminary results in [10] indicate process times of several minutes per depth map on an high-end workstation [1]. In contrast, we claim sub-second performance for depth maps of similar resolution, although our depth maps are not as accurate.

The work by Koch [12] also applies computer vision techniques to infer dense, accurate depth maps from image pairs, and uses this information to construct 3D graphical representations of the surveyed world. Unfortunately, his methods are far from real-time, restricted to static environments, and thus not suitable for augmented reality applications.

Like Koch, we use a computer vision technique known as stereo matching to infer depth from stereo image pairs. Stereo matching is a well-established research area in the computer vision literature [8] [5]. Nonetheless, real-time algorithms for stereo matching are only a recent development.

We believe that our near-real-time stereo matching algorithm is new. It is faster than other published near-real-time algorithms [13] [15], and is excelled only by algorithms running on custom-built hardware [15] [14] [11] (see Table 1). However, since our algorithm runs on general-purpose workstations, it is more affordable (no expensive, single-use, custom-built hardware is required) and more flexible (none of the parameters are hard-wired) than those.

Even though our algorithm is faster than some of those previously published efforts, our resulting depth maps are also less accurate.

Algorithm	Hardware	Resolution	Time
Ours	1-proc. SGI Onyx	320x240x30	620ms
Ours	2-proc. SGI Onyx	320x240x30	370ms
Ours	2-proc. SGI Onyx	160x120x15	100ms
Matthies [12]	68020 w/8 image proc. cards	64x 60x 6	1000ms
Ross [15]	Sun Sparc II	256x240x16	2460ms
Ross [15]	64 Cell iWarp	256x240x16	150ms
Ross [15]	64 Cell iWarp	512x480x88	2180ms
Nishihara [13]	custom-design	512x512x ?	33ms
Kanade [10]	custom-design	256x240x30	33ms

Table 1: Running times of our stereo matching algorithm compared with previous real-time or near-real-time stereo matching algorithms. Resolution is the resolution of the generated depth map in x, y, and depth, i.e., the range of possible disparity values for a matched point.

3 System Description

Figure 1 outlines the architecture of our augmented reality system. Two black and white video cameras are mounted on top of a Fakespace Boom. The cameras need to be aligned so that their epi-polar lines[1] roughly coincide with their horizontal scan-lines. Unfortunately, simply aligning the cameras' outer housings is insufficient due to large manufacturing tolerances in internal image sensor-array alignments (for example, our cameras had a pitch difference of several degrees). While we achieved alignment of ± 3 pixels manually by trial and error, less time-consuming options are available; for example, the images could be aligned in software [4] or by using calibrated off-the-shelf hardware [16].

The cameras continuously transmit gen-locked left/right video image pairs, such as shown in Figures 2 and 4, to the red and green inputs of a Sirius video card. The Sirius video card digitizes the analogue video signal and transfers the bitmaps into the main memory of an SGI Onyx.

We then apply the stereo matching algorithm described in Section 4 to the image pair. Figures 3 and 5 show the resulting depth maps. We copy the z-values for the left image into the z-buffer and transfer the left video image to the red frame-buffer. Since every pixel of the video image now has an associated z-value, we simply render all computer graphics objects for the left view — z-buffering resolves all occlusion relations.

The procedure is repeated for the right view: we clear the z-buffer, copy the generated z-values for the right view into it, transfer

[1]An epi-polar line is the intersection of the image plane with the plane defined by the projection centers of the two cameras and an arbitrary point in the 3D world space.

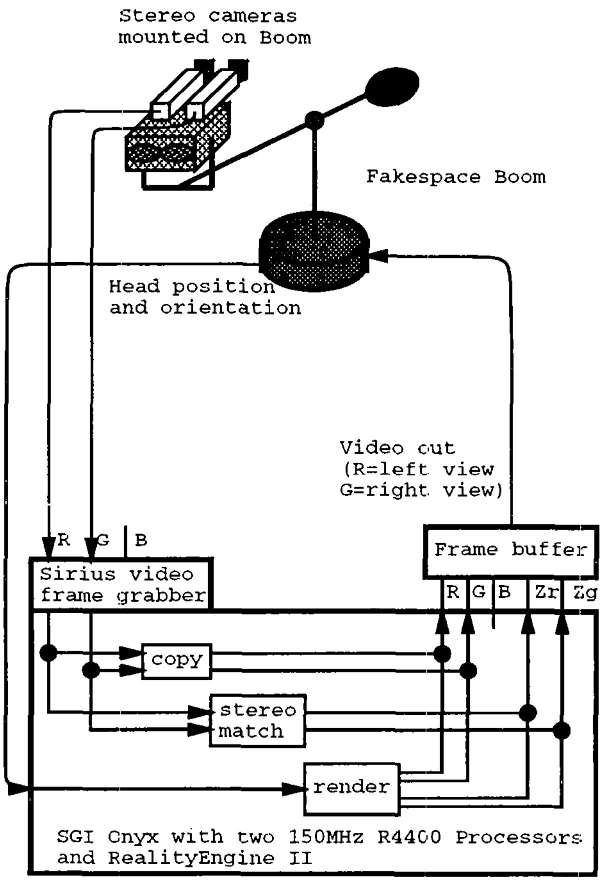
Stereo cameras mounted on Boom
Fakespace Boom
Head position and orientation
Video out (R=left view G=right view)
R
G
B
Frame buffer
Sirius video frame grabber
R
G
B
Zr
Zg
copy
stereo match
render
SGI Onyx with two 150MHz R4400 Processors and RealityEngine II

Figure 1: Schematic of our video see-through augmented reality system. The inputs and outputs R, G, and B correspond to the red, green, and blue channels, respectively. Zr and Zg are the depth- or z-values for the red and green channels, respectively. Since only one z-buffer is available in double-buffering mode, we first render the left view (red channel) completely and then the right view (green channel).

Operation	Resolution	Time
Stereo matching algorithm	320x240	370ms
z-value transfer per frame	320x240	20ms
RGB-value transfer per frame	640x240	20ms
Video capture	640x240	0ms
Rendering per frame	1280x1024	< 10ms
Total for stereo image pair	1280x1024	~470ms

Table 2: Results of timing tests for the various parts of our system, running on a 150MHz two-processor SGI Onyx with a RealityEngine II. Capturing images from video does not use the CPU, but does introduce an additional lag of at least 100ms. Rendering is highly scene-dependent — our extremely simple test scenes take less than 10ms to render.

the right video image into the green frame-buffer, and finally render all computer graphics objects for the right view.

The Fakespace Boom then displays the red/green augmented reality image pairs so generated as a stereo image pair. Figures 6 and 8 and Figures 7 and 9 show two examples. The Boom also allows us to couple the virtual camera pair position and orientation

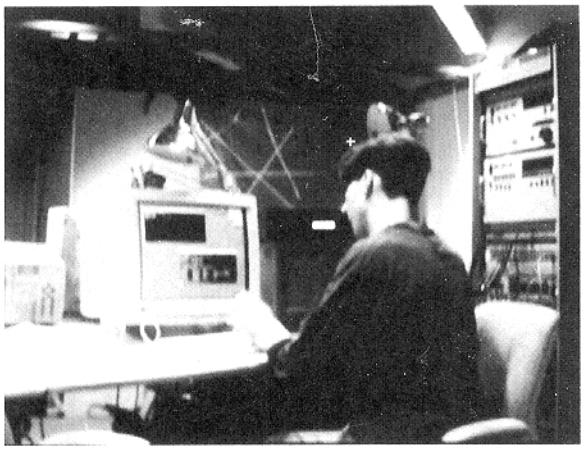

Figure 2: Left video camera image. Using this and the right video image, we infer depth for every pixel in the video image in near-real-time.

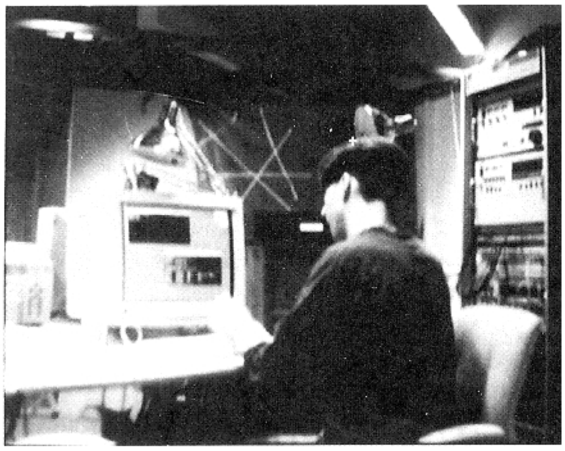

Figure 4: Right video camera image.

directly with those of the video camera pair. Therefore, the computer graphics objects are rendered with the same perspective as the video images.

While our system is video see-through, the same setup is used for optical see-through systems. Instead of a Fakespace Boom, the user wears a head-mounted optical see-through display and a pair of head-mounted video cameras. The video signal is processed as before, except that the video images are never transferred to the frame-buffer. Therefore, only the computer graphics objects — properly clipped by the generated z-values to occlude only more distant objects — are displayed on the optical see-through display.

The various parts of our system require varying amounts of compute time. Table 2 shows the results of our timing tests. While the stereo-frame rate of roughly two updates per second is still an order of magnitude too slow for practical augmented reality systems, our work may guide hardware architects to address the needs for faster and more affordable video processing hardware. Alternatively, resolution of the depth maps or video images may be reduced further.

4 Basic Algorithm

The new stereo matching algorithm we use to infer depth for the video images is central to our occlusion-resolving augmented reality system. Like all other stereo matching algorithms, it works by matching points in the left image to points in the right image and vice versa. Once the relative image positions of a pair of matched points are established, triangulation is used to infer the distance of the matched points to the cameras [8].

Our algorithm is area-based, i.e., it attempts to match image areas to one another. It works in five phases.

4.1 Phase One

In the first phase, we subsample the original video image. Currently, we operate at half the resolution of the video images. Higher (lower) resolution gives more (less) accurate results while slowing down

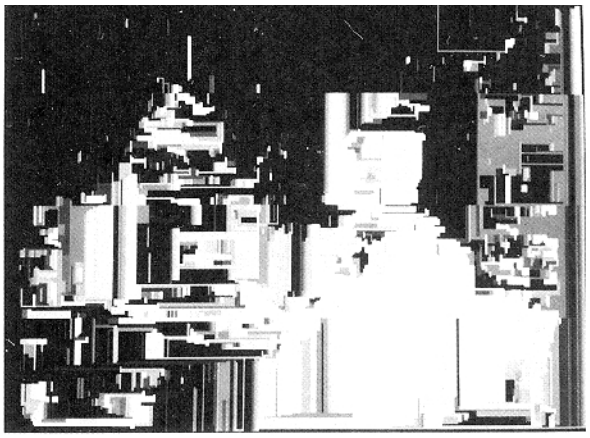

Figure 3: Our new stereo matching algorithm produces a half-resolution, approximate depth map for the left and right camera-view in near-real-time. The depth map for Figure 2 is shown here.

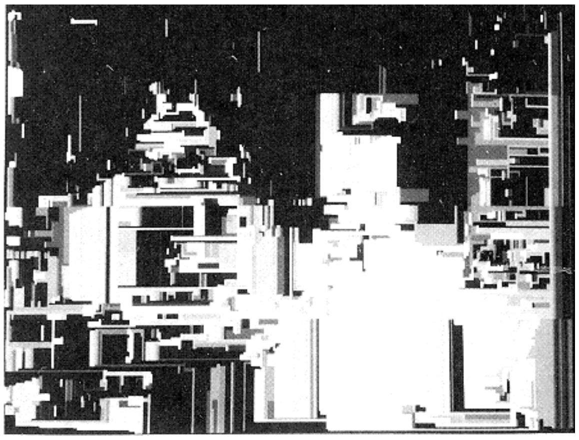

Figure 5: The computed depth map for Figure 4.

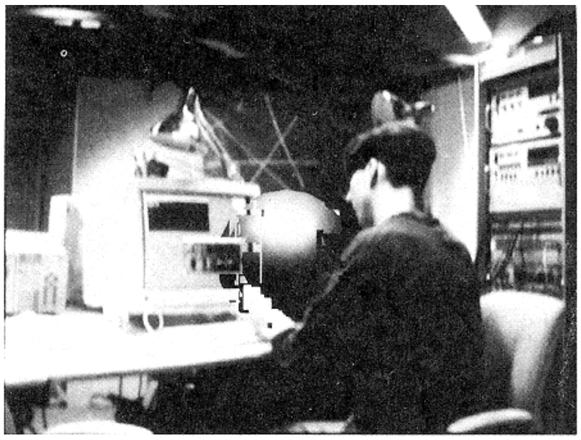

Figure 6: Real and virtual imagery are combined via the standard z-buffer algorithm. Here, a virtual sphere occludes and is occluded by real-world objects in the left camera view.

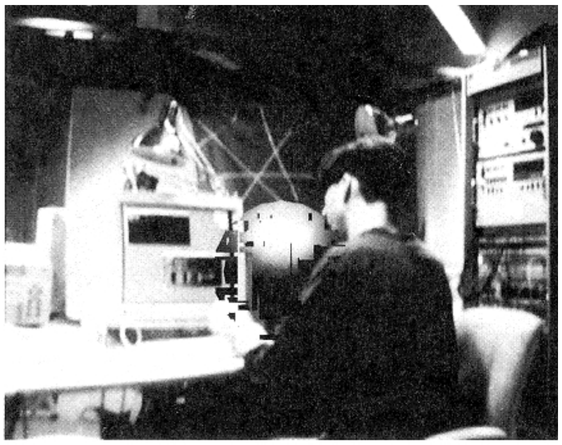

Figure 8: The right view of the scene in Figure 6.

(speeding up) the algorithm.

4.2 Phase Two

The second phase analyzes the vertical pixel spans of the subsampled video images for sudden changes in intensities; the vertical pixel span is split at the points at which such a change occurs. Figures 10 and 12 illustrate the result of this operation.

4.3 Phase Three

The third phase generates the individual areas or *blocks*. A block is part of a vertical pixel span whose length is delimited by the splits introduced in the second phase of the algorithm. Therefore, all the pixels belonging to a particular block vary little in intensity (otherwise the second phase would have generated a split). Accordingly, only a few parameters suffice to describe a block: its x and y position, its length, the average intensity of all its pixels, and the standard deviation of the intensities. To ensure that average intensity and standard deviation properly characterize a block, we impose a minimum length of 3 pixels.

4.4 Phase Four

The fourth phase of our algorithm matches blocks in the left image to blocks in the right image and vice versa. We compare every given block with all blocks in the other image that share the same horizontal scan-lines to find the best match. (This is less work than a full search because the range of possible disparity values restricts the number of blocks we must examine; see Figure 11 and also Section 5.2.) Two blocks match if the differences in their y-position, their length, their average intensity, and their standard deviation are below preset tolerances. The resulting depth estimates for the left and right block are entered into the depth map for the left and right image, respectively. The differences in the matching blocks' parameters are also recorded and used to weight the depth

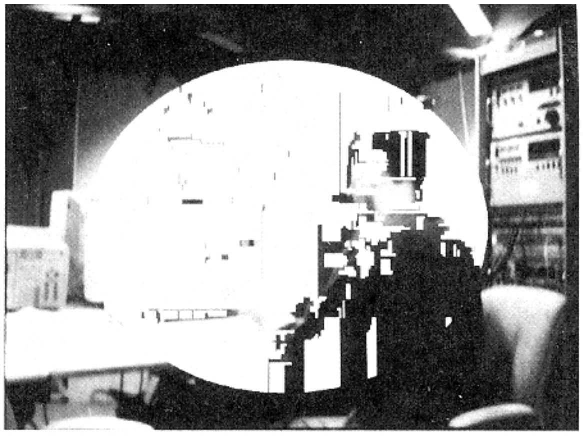

Figure 7: To visualize depth we let a virtual screen-aligned disk occlude and be occluded by real-world objects. Due to errors in the computed depth map for the video image, occlusion is not always resolved properly.

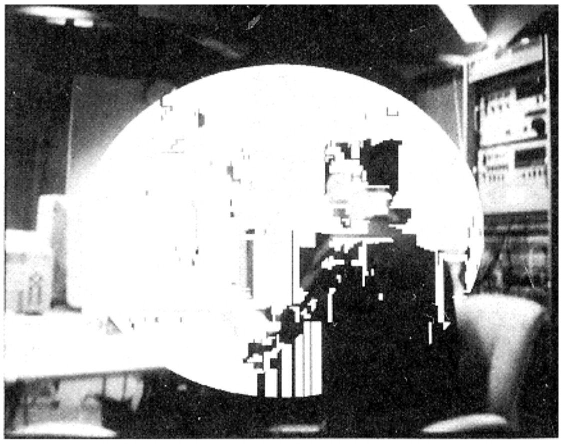

Figure 9: The virtual, screen-aligned disk in the right view.

estimate.

At the end of the fourth phase we have two depth maps, one for the left and one for the right image. Since not every block is guaranteed to match and since some blocks might match several times, each depth map has between zero and several depth entries for each pixel.

4.5 Phase Five

In the fifth and final phase, we average multiple depth entries for each pixel in each depth map (using the earlier recorded weights). Pixels with no depth entries are interpolated from the neighboring entries in the same horizontal scan-line of the depth map.

4.6 Critical Features

Several features of our algorithm are critical. First, blocks are part of vertical pixel scans. Compared to horizontal pixel scans, vertical pixel scans are less distorted by perspective foreshortening and occlusion differences in the left and right views. Therefore, matching is less error-prone.

Second, blocks are only one pixel wide. The same advantages as above apply.

Third, we search for matching blocks only along the same horizontal scan-line. If we assume that the camera's epi-polar lines roughly align with the horizontal scan-lines, then these blocks are the only possible matching candidates — even if we tilt the head (i.e., both cameras).

Fourth and last, the exceptional speed of our algorithm results from its ability to group pixels into blocks and then match those blocks by comparing only a few characterizing values (i.e., y-position, length, average intensity, and standard deviation). On the other hand, these few values do not always characterize a block distinctively enough; hence matching is subject to error, and thus our algorithm does not always estimate depth correctly.

5 Extensions

Several techniques exist to increase accuracy and speed of the above basic algorithm. We describe these techniques here.

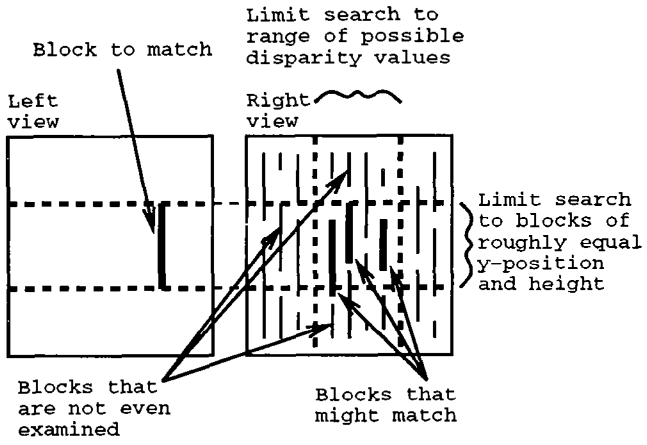
Block to match
Limit search to range of possible disparity values
Left view
Right view
Limit search to blocks of roughly equal y-position and height
Blocks that are not even examined
Blocks that might match

Figure 11: A given block matches only blocks that have roughly the same y-position, length, average intensity, and standard deviation. The block that matches most closely is selected for computation of the depth estimate.

5.1 Allowing for Inaccurate Alignment

Since our stereo camera pair is not fully calibrated — in particular, the epi-polar lines correspond only to a band of horizontal scan-lines — we adjust the matching algorithm to take inaccurate alignment into account. To match a block we therefore first find the scan-line that crosses its middle. We then consider all blocks that cross that scan-line (not necessarily in the middle) as possible candidates. A block matches the original block only if the difference in the vertical placement of their start- and end-points is within the alignment error, e.g., in our case within ±3 pixels.

5.2 Horizontal Depth Coherency

When matching a block in the left image to blocks in the right image, it is unnecessary to examine all the blocks on the right that share the same middle scan-line (as described in Section 5.1). The disparity range, i.e., the difference in x-position of the projection

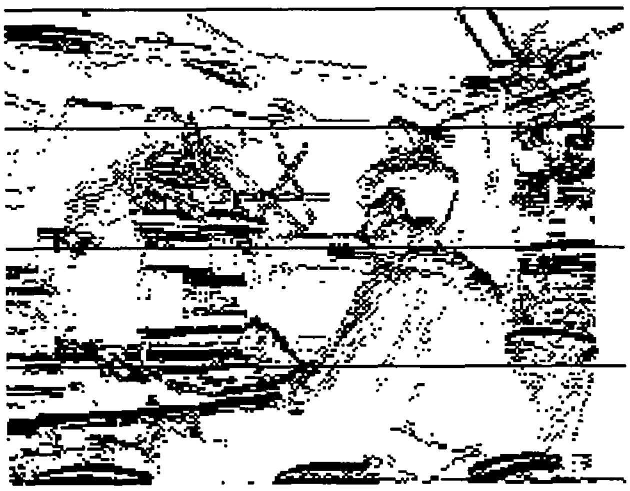

Figure 10: Vertical pixel scans are split into blocks whenever the intensity along the vertical scan changes rapidly. We visualize the process by drawing a black pixel at every split.

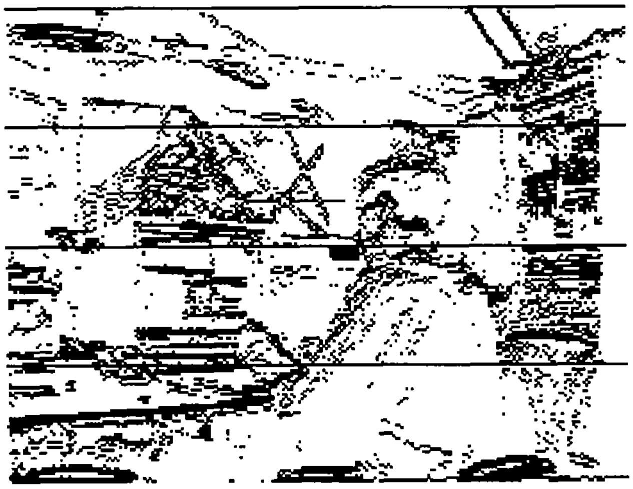

Figure 12: The blocks for the right view.

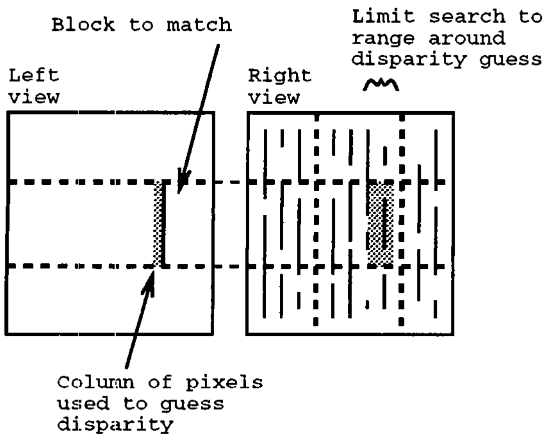
Block to match
Limit search to range around disparity guess
Left view
Right view
Column of pixels used to guess disparity

Figure 13: We use the previously computed depths of the pixels immediately to the left of the block we are currently trying to match to guess its disparity value.

of the same object in the two video images, is limited. The objects closest to the cameras have the largest disparity. (Similarly, objects at infinity have no disparity.) Thus, for example, the video images shown in Figures 2 and 4 have a disparity range of 0 to 30 pixels. As hinted in Section 4 and Figure 11, we examine blocks only within that disparity range.

Furthermore, the depth of horizontally adjacent pixels in a video image is likely to be coherent. This coherency lets us further limit the number of blocks we examine to determine matches. Therefore, instead of searching the whole disparity range for a match, we inspect the previously computed depth of the pixels immediately to the left of the block we are currently trying to match. The average depth of these pixels determines the likely disparity of a match for the current block. Thus, we only search a narrow band (e.g., 6 pixels wide) around that disparity value for the match. Figure 13 illustrates this process.

To ensure stability we use the suggested disparity only if at least as many pixels as the block is long contribute to the average depth, i.e., only if all those pixels to the left of the block have an associated depth value. Otherwise, we inspect the column of pixels immediately to the left of them. Again we compute the average depth and likely disparity (taking into account the estimates generated earlier). We then search for a match around that disparity value in a band twice the original width (e.g., 12 pixels wide). We double the width of the search band since the pixels that generate this new depth estimate are further to the left and thus the depth coherency is weaker. Figure 14 shows this process.

We continue to inspect the depths of the pixels further to the left and accordingly widen the search band to three times the original width, four times the original width, etc., in case too few pixels contribute to the disparity estimate. The process terminates either when enough pixels contribute to the depth estimate or when the search band is as wide as the total disparity range of the images.

The algorithm described above exploits depth coherency in only one direction — left to right — since only pixels to the left influence the depth of pixels to the right. To avoid this bias we use left-right coherency only when matching blocks in the right image to blocks in the left image. When matching blocks in the left image to blocks in the right image we employ right-left coherency. Differences in the resulting depth estimates are averaged out in the final phase of the basic algorithm (see Section 4).

Taking advantage of horizontal depth coherency as described above makes the algorithm at least twice as fast, i.e., total running time is halved.

5.3 Disregarding Image Borders

The camera parameters of the calibrated stereo camera pair are ideally identical except for the x-coordinates with respect to the camera's image plane coordinate system: they differ by the interocular distance. Because of this shift in viewpoint, the left-most vertical pixel scans of the left camera record objects that the right camera does not see. Thus, it is futile to match or even process these pixels. Similarly, the rightmost vertical pixel scans of the right camera cannot be matched.

Depending on the interocular distance and the range of camera-object distances, it is thus possible to cull the number of pixels to process by up to 5%. Accordingly, the algorithm becomes faster by a factor of up to 0.95 and accuracy of matching improves since we eliminate candidates for erroneous matches.

5.4 Parallelizing the Algorithm

Our stereo matching algorithm is parallelizable so that it takes near-optimal advantage of the two processors in an SGI Onyx. Since the data for the left and right image in the first, second, third, and fifth phase of the algorithm (see Section 4) do not interact, i.e., left-image data never influence right-image data or vice versa, we assign one processor each to process the left and right image.

The fourth phase, i.e., the matching phase, is different: each match generates data that are written into the depth maps of both images. Therefore, care has to be taken to avoid having the processors simultaneously write to the same memory location. Furthermore, we must not impede processing speed, for example, by imposing mutex locks or similar delay schemes.

Dividing the stereo image pair into disparate zones solves these problems. We create these zones by modifying the second phase of the algorithm to generate additional splits in the vertical pixel spans. These additional splits are easily identified in Figure 10 and 12; they are the three horizontal lines crossing the width of the left and right image.

While a single split (instead of three) seems to suffice to avoid simultaneous memory access by the two processors, a small amount of overlap does occur at the borders. Therefore, we separate the images into four zones each to ensure disparity. We assign the first processor to match the top zones of the left and right image,

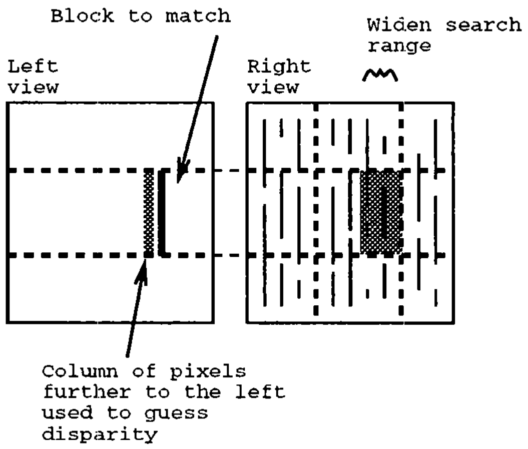
Block to match
Widen search range
Left view
Right view
Column of pixels further to the left used to guess disparity

Figure 14: If not enough pixels immediately to the left of the block have depth values, we examine pixels further to the left. Accordingly, we widen the search band around the resulting disparity estimate to take into account the reduced coherency relation.

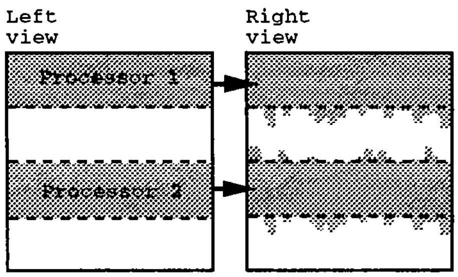

Figure 15: The first stage of matching blocks with two processors. Each processor matches left blocks to blocks in the right image and writes the resulting depth values into the left and right depth map. The processors work on disjunct zones of the image to avoid overwriting the same memory locations simultaneously.

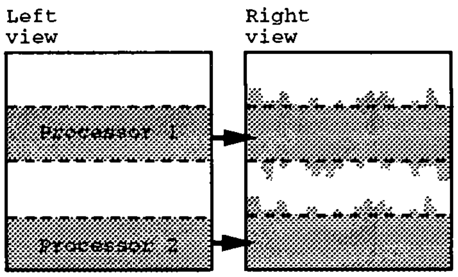

Figure 17: The third stage of matching blocks with two processors. The processors work on the image zones not previously processed. Each processor matches left blocks to blocks in the right image and writes the resulting depth values into the left and right depth map.

while the second processor matches the lower-middle zones of the left and right image, as illustrated in Figures 15 and 16. When both processors finish, we reassign the first processor to process the upper-middle zones, while the second processor computes the bottom zones, as shown in Figures 17 and 18.

The same idea of dividing images into disparate horizontal zones is also applicable to the other phases of the algorithm for the case of more than two processors. However, the more processors we use, the lower the height of each zone for the matching phase of the algorithm, and thus the shorter the blocks. Since our algorithm seems to work best if the maximum block length is between 16 and 128, the current resolution of 320 by 240 thus limits us to at most eight processors, unless resolution in the y-dimension is increased.

Using the above parallelization scheme we achieve an additional speed-up factor of 0.6. We do not reach the optimum because workloads for the processors may vary and thus the processors may have to wait for one another at the end of each phase of the algorithm before computations can proceed.

5.5 Sacrificing Software Structuring and Readability

Our current implementation of the matching algorithm is optimized for speed. In particular, we use pointers instead of arrays, we abstain from using subroutines (to ensure maximum optimization by the compiler), we do not loop unnecessarily over the video image or z-value data, and we try to move data as little as possible. Therefore, the first three phases of the algorithm described in Section 4 are actually implemented as a single pass.

The result of hand-optimizing the code is a performance gain of a factor of roughly 0.5. Currently, we believe that the data-transfer rates, i.e. bus bandwidth (and not CPU power), limit further speed-ups.

6 Conclusion

6.1 Discussion

Our solution is imperfect. As Figures 3 and 5 show, the generated depth maps sometimes include gross errors. Worse, these errors may not persist over several frame-pairs, so that objects may blink off and on as the algorithm classifies them as occluded and not occluded. Such blinking turns out to be more distracting than a consistent misqualification.

In particular, our algorithm (as most other area-based stereo matching algorithms) has difficulties in computing the depth for rectangular image areas that are evenly lit, non-textured, and horizontal. Figures 3 and 5 exhibit this failure in the areas of the left desk surface and the computer screen of Figures 2 and 4.

Furthermore, even though our algorithm is several orders of magnitude faster than other software algorithms, its current running time of over 300ms per stereo image pair still prohibits its use in practical augmented reality systems. To be useful, lag must be reduced by another order of magnitude.

On the other hand, despite the inaccuracies and despite the inadequate speed, our system already demonstrates the positive impact of occlusion resolution for augmented reality applications. In addition, our algorithm lets us experiment with various quality/speed tradeoffs. For example, at the expense of depth map resolution, we can improve computation speed. Particularly in highly dynamic

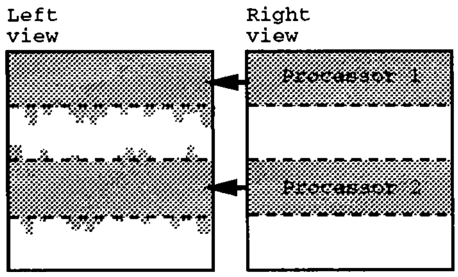

Figure 16: The second stage of matching blocks with two processors. Each processor matches right blocks to blocks in the left image and writes the resulting depth values into the left and right depth map.

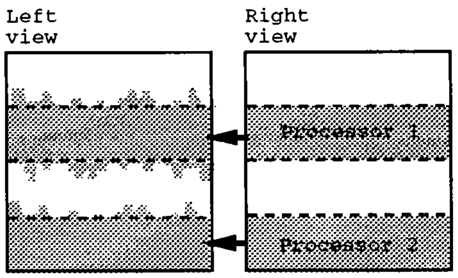

Figure 18: The fourth and final stage of matching blocks with two processors. Each processor matches right blocks to blocks in the left image and writes the resulting depth values into the left and right depth map.

environments, it seems unnecessary to compute depth maps with the same resolution that objects are rendered.

6.2 Implications

Our work is evidence for the continuing interaction between the research areas of computer vision and computer graphics [6]. As such it points towards the need of graphics workstations to better attend to computer vision requirements, for example, to allow high-speed data transfers between the CPU and main memory or between main memory and the frame-buffer (see Table 2).

6.3 Future Work

Future work might improve the accuracy and speed of our stereo matching algorithm. We list five possible areas of future research.

First, making the depth computation for rectangular, featureless image areas a special case might improve accuracy considerably.

Second, taking advantage of interframe coherency is imperative. This could considerably improve algorithm speed and accuracy, while simultaneously resolving the object-blinking problem discussed in Section 6.1.

Third, since object blinking seems to result mainly from noisy camera images, anti-aliasing or smoothing the original video images before they are processed deserves investigation. In particular, how does such a preprocessing step influence algorithm speed and accuracy?

Fourth, our current algorithm is static; in particular, the edge-detection algorithm used to implement phase two of the basic algorithm (see Section 4) does not adapt to changing intensity value distributions, for example, when increasing or decreasing total illumination of the augmented world. Thus, we rely on the cameras to automatically adjust their apertures to maintain apparent constant illumination. Performing this function in software would obviate this dependency.

Fifth and finally, if only a few computer graphics objects augment the video images, a considerable performance gain is possible. Instead of stereo matching the whole video image pair, only the video image areas that are covered by the bounding boxes of the computer graphics objects require depth values. Thus, depending on number, size, and distribution of the computer graphics objects, computation requirements might decrease drastically. In addition, the disparity of the left- and right-view renderings of each object might guide the stereo-matching algorithm, thus further improving performance.

Acknowledgements

We thank the following people for their help: Kevin Arthur and Ted Camus for providing us with sample stereo images and enduring all our questions about video frame grabbers, cameras, and stereo matching algorithms; Richard Szeliski and Takeo Kanade for guiding us through the stereo matching literature; Martin Friedmann, Ken B. Russel, and Thad Starner for letting us use their hardware, in particular the Sirius video card; and finally, our advisors Andy van Dam and John Hughes for their support.

This work was sponsored in part by NSF/ARPA Science and Technology Center for Computer Graphics and Scientific Visualization, Sun Microsystems, Autodesk, Taco Inc., Microsoft, NASA, NCR, and ONR Grant N00014-91-J-4052, ARPA Order 8225.

References

[1] Arthur, Kevin. Private Communications (August 1994).

[2] Azuma, Ronald and Gary Bishop. Improving Static and Dynamic Registration in an Optical See-Through HMD. Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994). In Computer Graphics Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, New York, 1994, pp. 197–204.

[3] Bajura, Michael, Henry Fuchs, and Ryutarou Ohbuchi. Merging Virtual Objects with the Real World: Seeing Ultrasound Imagery within the Patient. Proceedings of SIGGRAPH '92 (Chicago, Illinois, July 26–31, 1992). In Computer Graphics 26, 2 (July 1992), 203–210.

[4] Bajura, Mike and Ulrich Neumann. An Improved Model for Augmented Reality Systems. Technical Report TR-94-022, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, 1994.

[5] Barnard, Stephen T. and Martin A. Fischler. Computational Stereo. *ACM Computing Surveys*, 14(4):553–572, December 1982.

[6] Carlbom, Ingrid, William Freeman, Gudrun Klinker, Wiliam E. Lorensen, Richard Szeliski, Demetri Terzopoulos, and Keith Waters. Computer Vision for Computer Graphics. Course Notes of Course 03 of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994).

[7] Caudell, Thomas P. and David W. Mizell. Augmented Reality: An Application of Heads-Up Display Technology to Manual Manufacturing Processes. *HICSS*, pages 659–669, 1992.

[8] Dhond, Umesh R. and J. K. Aggarwal. Structure from Stereo — A Review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, November 1989.

[9] Feiner, Steven, Blair MacIntyre, and Doree Seligmann. Knowledge-Based Augmented Reality. *Communications of the ACM*, 36(7):52–63, July 1993.

[10] Fuchs, Henry, Gary Bishop, Kevin Arthur, Leonard McMillan, Ruzena Bajcsy, Sang Lee, Hany Farid, and Takeo Kanade. Virtual Space Teleconferencing Using a Sea of Cameras. Technical Report TR-94-033, University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, June 1994.

[11] Kanade, Takeo. Development of a Video-Rate Stereo Machine. In *Proceedings of '94 ARPA Image Understanding Workshop*, November 1994.

[12] Koch, Reinhard. Automatic Reconstruction of Buildings from Stereoscopic Image Sequences. In R. J. Hubbold and R. Juan, editors, *Eurographics '93*, pages 339–350, Oxford, UK, 1993. Eurographics, Blackwell Publishers.

[13] Matthies, Larry. Stereo Vision for Planetary Rovers: Stochastic Modeling to Near Real-Time Implementation. *International Journal of Computer Vision*, 8(1):71–91, 1992.

[14] Nishihara, H. K. Real-Time Implementation of a Sign-Correlation Algorithm for Image-Matching. Technical Report 90-2, Teleos Research, February 1990.

[15] Ross, Bill. A Practical Stereo Vision System. In *Proceedings of Computer Vision and Pattern Recognition '93*, 1993.

[16] Stereographics. CrystalEyes Video System, 1994.

[17] Wloka, Matthias M. Lag in Multiprocessor Virtual Reality. *Presence*, 4(1), 1994. To appear.