Using Time-of-Flight Range Data for Occlusion Handling in Augmented Reality.

Conference Paper · January 2007

DOI: 10.2312/EGVE/IPT_EGVE2007/109-116 · Source: DBLP

3 authors, including:

Benjamin Huhle
University of Tuebingen
22 PUBLICATIONS **398** CITATIONS

SEE PROFILE

Andreas Schilling
University of Tuebingen
85 PUBLICATIONS **1,413** CITATIONS

SEE PROFILE

IPT-EGVE Symposium (2007)
B. Fröhlich, R. Blach, and R. van Liere (Editors)

Using Time-of-Flight Range Data for Occlusion Handling in Augmented Reality

Jan Fischer[1] † Benjamin Huhle[2] Andreas Schilling[2,3]

[1]Island Graphics Group, University of Victoria, Canada
[2]WSI/GRIS, University of Tübingen, Germany [3]Stanford University, USA (visiting scientist)

Abstract

One of the main problems of monoscopic video see-through augmented reality (AR) is the lack of reliable depth information. This makes it difficult to correctly represent complex spatial interactions between real and virtual objects, e.g., when rendering shadows. The most obvious graphical artifact is the incorrect display of the occlusion of virtual models by real objects. Since the graphical models are rendered opaquely over the camera image, they always appear to occlude all objects in the real environment, regardless of the actual spatial relationship. In this paper, we propose to utilize a new type of hardware in order to solve some of the basic challenges of AR rendering. We introduce a depth-of-flight range sensor into AR, which produces a 2D map of the distances to real objects in the environment. The distance map is registered with high resolution color images delivered by a digital video camera. When displaying the virtual models in AR, the distance map is used in order to decide whether the camera image or the virtual object is visible at any position. This way, the occlusion of virtual models by real objects can be correctly represented. Preliminary results obtained with our approach show that a useful occlusion handling based on time-of-flight range data is possible.

Categories and Subject Descriptors (according to ACM CCS): H.5.1 [Information Interfaces and Presentation]: Artificial, augmented, and virtual realities I.3.3 [Computer Graphics]: Display algorithms

1. Introduction

Augmented reality (AR) systems combine real images with three-dimensional graphical objects [ABB*01]. A significant proportion of augmented reality applications are designed as monoscopic video see-through systems. In these systems, a single digital video camera captures images of the real scene and uses them as background images when drawing the virtual objects. The main drawback of this approach to augmented reality with regard to the resulting graphical representation is the total lack of depth information. Since the distance of the real objects to the camera is not known, complex spatial interactions cannot be correctly displayed.

One of the most obvious artifacts resulting from this shortcoming is the lack of a correct handling of occlusions. Typcial rendering pipelines for AR simply superimpose the graphical models over the camera image. This produces output images in which virtual models continually occlude all objects in the real environment, neglecting the actual spatial relationships. The task of achieving a correct representation of the mutual occlusion of real and virtual objects in AR is called *occlusion handling*, and the occlusion of virtual models by real occluders can be identified as the most difficult and relevant problem [Kli04].

Here, we propose for the first time the introduction of a new kind of equipment into augmented reality. Time-of-flight range sensors deliver a 2D map of distances to objects in the environment in real-time. We believe that by combining this depth information with the color image delivered by a digital video camera, complex graphical effects can be added to monoscopic video see-through AR. In this paper, we describe how to use time-of-flight range data for occlusion handling in augmented reality.

The distance map delivered by the time-of-flight sensor is registered to high resolution color images of the real environment. This is achieved with a one-time calibration step. For camera pose estimation, our prototype system uses standard optical marker tracking based on the ARToolKit li-

† e-mail: jan@janfischer.com

brary [KB99]. When rendering graphical objects according to the currently estimated camera position and orientation, the information in the 2D depth map is taken into account. A specialized shader program running on the graphics processing unit (GPU) compares the absolute distance to the camera image plane computed for graphical object pixels with the measured real-world depth. If a real object is closer to the camera at a given position, the display of the virtual object pixel is suppressed. This way, the occlusion of graphical models by real-world objects is correctly represented.

We describe our prototype setup for augmented reality rendering using time-of-flight range data. The digital video camera is physically attached to the time-of-flight sensor in order to record a combined color and range image stream. We discuss the required steps to register color image and range data, as well as implementation details for making the comparison of absolute depths possible. Our current prototype design uses an offline process for generating the AR images. However, the processing speed of our occlusion handling method is fast enough for real-time image generation, and it could be easily adapted to be used in an interactive AR application.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 introduces the time-of-flight sensor technology and describes the hardware setup of our prototype system. The registration of the depth range data with the color images is described in Section 4. The actual occlusion handling method is discussed in Section 5, and Section 6 elaborates on some aspects of the implementation. Results obtained with our prototype system are presented in Section 7 and are discussed in Section 8. Finally, Section 9 concludes this paper with a summary and an outlook on possible future developments.

2. Related Work

The detection and handling of occlusion in augmented reality is an active research area. Breen et al. have suggested a model-based approach to handling occlusion in augmented reality [BWRT96]. The use of geometric models of known real objects for detecting the occlusion of virtual objects is called *static occlusion handling*. An extension of static occlusion handling for determining how virtual objects are hidden by the user's body was described by Fuhrmann et al. [FHFG99]. Fischer et al. have presented an application-specific optimization for using static occlusion handling in medical AR [FBS04]. *Dynamic occlusion handling* does not require geometric descriptions of real objects in the environment, but relies on image processing and computer vision techniques combined with certain assumptions or partial information about the real environment. Berger has described a method for resolving dynamic occlusion when overlaying virtual objects over recorded video sequences [Ber97]. Later, Lepetit and Berger presented another approach to handling occlusion in off-line augmented reality [LB00], which is based on the manual definition of the occlusion boundary of a real object as well as relevant key frames in the stored video sequence. An algorithm for detecting dynamic occlusion in front of planar, textured background objects was described by Fischer et al. [FRB03]. A similar, but more advanced, algorithm was later presented by Lin et al. [LSKS05]. Approaches for detecting occlusion in a stereo camera AR system have also been investigated. In some cases, an attempt to solve the occlusion problem using depth information delivered by stereo matching is made [KOTY00, WA95]. The method developed by Gordon et al. [GBB*02] can correctly render interaction devices into the scene.

Some researchers have used image-based object reconstruction techniques in order to incorporate real objects into a virtual environment. An example of this approach is described by Lok et al. [LNWB03]. However, this method typically requires multiple color video cameras (four in this case) and is computationally expensive.

Time-of-flight cameras, such as the *PMD* camera used in our system, can in principle deliver optical images along with the depth infromation. However, they normally provide only poor image quality compared to standard cameras, mainly due to the low resolution of the sensor. Therefore, it is a common approach to enhance the resulting 3D data by integrating a second (color) camera into the system [PHW*06, Reu06]. Since color images contain implicit information about the 3D geometry of a scene, it also becomes possible to enhance the quality of the distance data using a combination of a color camera and a time-of-flight camera, e.g., by optimizing the depth values in a Markov Random Field model that encodes the dependencies of edges in the image and the depth domain [HFS07]. However, constraining ourselves to real-time AR applications in this paper, we employed the depth data without further post-processing.

Another example of a depth camera is the Z-Cam™ produced by 3DV Systems [3DV07]. The use of this camera for depth-based occlusion handling has been demonstrated by Gvili et al. [GKOY03]. However, they only presented occlusion handling for purely image-based applications and for manually placed 3D objects. They did not deal with the registration of depth values from an AR camera tracking system with the acquired depth map, which is the core issue addressed in this paper (cf. Fig. 3).

3. The Time-of-Flight Range Sensor

Recently, time-of-flight sensors with a sufficiently high resolution for use in an AR context became available from different manufacturers at admissible costs. We chose the 3D time-of-flight camera from *PMD Technologies*, namely the *PMD [vision] 19k* [PMD07]. The camera uses a 160x120 pixel photonic mixer device (PMD) sensor array that acquires distance data using the time-of-flight principle with

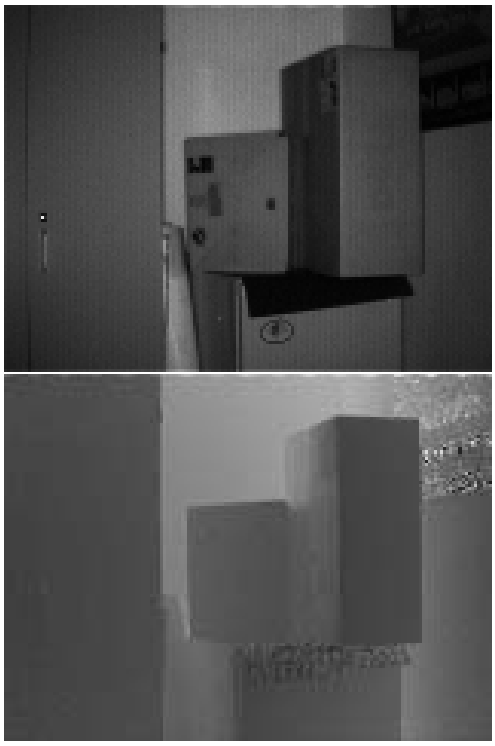

Figure 1: *The intensity (top) and depth images (bottom) delivered by the time-of-flight camera for an example scene.*

Figure 2: *Our two-camera setup (top). The high resolution color camera is attached to the top of the time-of-flight PMD camera. The bottom image shows the color image delivered by the color camera for the example scene.*

active illumination. An array of LEDs sends out invisible modulated near-infrared light. For each pixel, the PMD sensor delivers distance and intensity information simultaneously, where the distance data is computed from the phase shift of the reflected light directly inside the camera. Since both the intensity and the depth values are captured through the same optical system, they are perfectly aligned.

We use the PMD vision camera equipped with a 12 mm lens resulting in a horizontal field-of-view of about 30°. In contrast to devices such as most laser scanners and structured light range sensors, the time-of-flight technique with active illumination is able to capture 3D data of dynamic scenes. The camera works at a frame-rate of up to 20 fps and is therefore perfectly suitable for real-time augmented reality applications. Figure 1 shows the depth map and the intensity image as output by the camera for one example scene.

3.1. Hardware Setup of the Prototype System

As seen in Figure 1, the image data of the PMD camera itself is not adequate for augmented reality applications due to the low resolution and the limitation to grayscale images. In order to enhance the visual quality, we combined the time-of-flight camera with a standard high-resolution camera, namely a *Matrix Vision BlueFox* with a 1600x1200 pixel sensor and equipped with a 12 mm lens [Mat07]. The resulting horizontal field-of-view of about 34° is similar to that of the PMD camera. This ensures an easy calibration of both cameras and only a small loss of information. Moreover, due to the small dimensions of the Matrix camera, the two-camera-setup (see Figure 2) can be assembled in a way to provide a combined color and depth sensor with only small parallax effects.

4. Image Registration

To acquire colored 3D data from our two-camera setup, it is necessary to map the image data delivered by the color camera onto the depth map of the time-of-flight sensor. Since the intensity image and depth data coming from the PMD camera are perfectly aligned, it is possible to use standard algorithms for calibration and registration of the high-resolution camera with the depth data. This means that by registering two standard optical images - the PMD intensity image and the high resolution color image - we can effectively register two different modalities. To determine the relative position and orientation of both cameras, one can interpret the setup as a stereo system that needs to be calibrated. Using the stereo system calibration method of the *Camera Calibration Toolbox for Matlab* [Bou07] from Caltech, we calculate the extrinsic as well as the intrinsic parameters of this system.

By applying the resulting translation and rotation to the 3D data calculated from the depth data, one obtains 3D coordinates in the reference frame of the high resolution camera.

Note that the depth map entries originally delivered by the PMD camera do not represent the parallel projected distance to objects for each pixel. Instead, the absolute distance to the center of projection in the optical system is stored. In order to use the depth-of-flight data for advanced augmented reality rendering, however, Cartesian 3D coordinates are required.

To compute these 3D coordinates P from the depth map, i.e., from the distances between the PMD sensor and the real world objects, we use the simple pinhole camera model:

$$P = d\frac{p}{\|p\|},$$

where d is the depth value of the pixel (x_{img}, y_{img}) with projected 3D coordinates $p = (x_{img}, y_{img}, f)$ on the image plane (f being the focal length measured in pixels). This model is valid only after performing an undistortion step based on the estimated intrinsic parameters delivered by the calibration toolkit.

After back-projecting the 3D points from the color camera frame onto the image plane according to the intrinsic parameters of the color camera, a nearest-neighbor interpolation is performed to fill the whole area of the camera image with depth values. Finally, we supply the computed Cartesian depth values together with the already undistorted color images to the AR rendering pipeline.

5. Occlusion Handling

The occlusion handling scheme presented here is based on the comparison of the absolute depths of real and virtual objects. This is made possible since the aforementioned image acquisition pipeline delivers depth maps containing the parallel projected distance from the camera image plane to objects in the real world. These depths are calibrated so that an absolute depth value of 1.0 corresponds to one real millimeter. At the same time, the augmented reality camera pose estimation is also calibrated so that global coordinates corresponding to real millimeters are delivered.

Our protoype system for occlusion handling using time-of-flight range data consists of an offline image generation process. The depth maps and color images created by the process described above are stored on disk and then imported into our AR image generation software. We use the ARToolKit for marker tracking in order to estimate the position and orientation of the combined camera setup relative to a marker visible in the scene. The marker tracking is performed based on the color image obtained from the high resolution camera. Assuming that the correct camera calibration matrix is used and the correct marker dimensions are passed to the ARToolKit, it also delivers camera tracking information calibrated in real-world millimeters. This makes it possible to easily compare virtual objects depths with the time-of-flight depth maps.

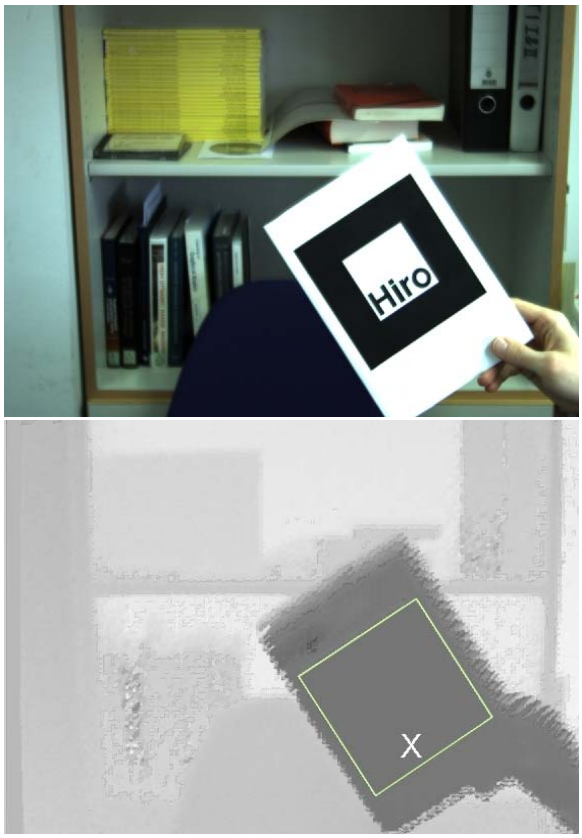

Figure 3: *Color and depth images for one example frame. The cross in the bottom image indicates the position where the depths were measured for comparison.*

Figure 3 shows the color and depth images for an example frame processed with our system. In the top image, the color data for a scene containing an ARToolKit marker is shown. The bottom image of Figure 3 shows the registered depth information for this frame displayed as pixel intensities. The brightly colored quadrangle in the depth image represents the outline of the marker projected according to the current camera pose estimation. The brightness on the inside of the marker rectangle correlates to the absolute depth of the marker computed using the estimated camera pose. In order to compare the depths delivered by the time-of-flight sensor and the marker tracking, we determined both depths at the position indicated by the white cross in the figure. The distance to the camera image plane computed at this position is 1156 mm, while the time-of-flight depth is 1176 mm. This is a deviation of only 2 cm, equivalent to an error of less than 2%. We have found that our system can deliver a similarly good correspondence between time-of-flight and marker tracking depths under most circumstances.

5.1. Modification of the Rendering Pipeline

When graphical objects are rendered in our system, an adapted rendering method is used in order to perform the occlusion handling. A special shader program is executed on the programmable graphics processing unit (GPU) available in modern graphics cards. Whenever a graphical primitive constituting a virtual model in the augmented environment is to be rendered, this shader program is activated. The shader compares the depth of each pixel of the graphical primitive with the depth information stored in the time-of-flight map. If the depth measured by the time-of-flight sensor is smaller than the primitive depth at this location, the display of the virtual object pixel is suppressed. This way, the occlusion of virtual objects by real objects is represented.

The OpenGL Shading Language (GLSL) was used for the implementation of the occlusion shader [Ros06]. In order to make a correct depth test possible, this shader cannot rely on the per-pixel depth information available in the graphics pipeline. The standard depth information available to the fragment shader, which controls the rendering of individual pixels, is provided in so-called OpenGL *window coordinates* (see [SWND03]). These window coordinates and the associated depth information are only indirectly related to the original virtual object vertices in the world coordinate system. Therefore, we use a special GLSL vertex shader, which computes the vertex coordinates transformed according to the current ARToolKit camera pose estimation. These transformed vertex coordinates reside in the so-called *eye coordinate* system and can be directly compared with the depth delivered by the time-of-flight sensor.

The z value of each vertex in eye coordinates is stored in a `varying` variable of the shader. This means that the z eye coordinate is linearly interpolated over the area of the graphical primitive. This principle is illustrated in Figure 4, where the z eye coordinates of the vertices of a triangle primitive are denoted as ec_*zn*. The fragment shader then can access the interpolated z coordinate for each pixel to be rendered. A correct comparison of this interpolated z value in eye coordinates with the time-of-flight depth is now possible.

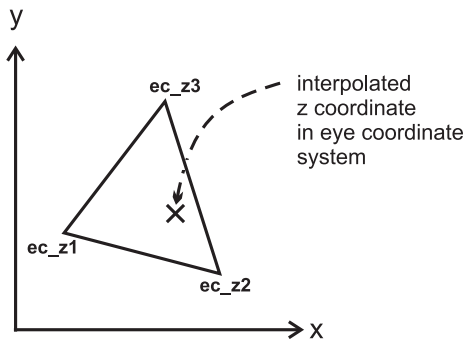

Figure 4: *Interpolation of the z value in eye coordinates over the area of a graphical primitive.*

An alternative to using a specialized shader for the depth test would be to convert the acquired depth range information into corresponding hardware z-buffer values and upload them into graphics card memory. This approach, however, would require a non-linear transformation executed on the CPU based on various parameters of the current OpenGL state. We believe that such a method would have a significant impact on the performance on the system, and would moreover require a considerably more complex implementation.

6. Implementation Details

The capturing of the sequences is done in the XGRT software framework developed in the WSI/GRIS graphics group of the University of Tübingen. With the flexible functionality of XGRT, we were able to acquire data from our two-camera system and perform the image registration. We could also show a real-time preview of the dynamic 3D scenes in order to confirm the validity of the acquired range information. As mentioned above, the color and depth images are then stored on disk for further processing in our occlusion-capable augmented reality rendering system.

Note that since undistorted images are already exported from the capturing software, we can forgo the undistortion step in the ARToolKit. This is achieved by setting the distortion factors in the ARToolKit camera parameter structure to the vector $(0,0,0,1)$. This way, the internal functions `arParamIdeal2Observ` and `arParamObserv2Ideal` return the unaltered identical input coordinates when performing image distortion or undistortion. As mentioned above, we determine the remaining internal parameters of the high resolution camera in a separate calibration step. In our augmented reality software, these parameters are passed as perspective matrix K to the ARToolKit library, ensuring a correct functioning of the marker tracking component.

In the central occlusion fragment shader in our AR rendering software, a comparison of interpolated virtual object z coordinates with time-of-flight depth data is performed. This test is implemented using conditional branching, i.e., with an `if` statement. If the real object depth is found to be less than the virtual object depth, the GLSL special function `discard` is called in order to prevent the fragment shader from altering the camera image pixel.

The time of flight depth map is supplied to the occlusion fragment shader as a 2D texture. This means that each depth map is uploaded into graphics card memory along with the color camera image before each AR frame is rendered. In our system, the depth information is represented as 16-bit unsigned integer data encoded in the R- and G- channels of a color image. The occlusion shader accesses this RGB texture and reconstructs the 16-bit depth information from it.

Note that since the occlusion shader replaces the fixed

functionality rendering pipeline of OpenGL, standard features like lighting are not automatically available anymore. In our prototype system, we have added a simple diffuse lighting computation to the occlusion shader. This way, the visible portions of virtual objects are rendered with a monochrome shading effect suggesting their three-dimensional structure. (Note that we have used a simple lighting model for demonstration purposes only. More advanced lighting could be easily integrated into the shader, and corresponding shader code is freely available, for example in [Ros06]).

7. Results

We have tested the occlusion handling method using time-of-flight depth data with several example scenes. Images from three tests are shown in Figures 5, 6, and 7. Although the color camera used in our system is capable of delivering higher resolutions, we scaled down the color images to a size of 640x480 pixels before feeding them into the AR image generation software. This was done to reduce the size of the color image video sequences on disk, which are stored as sets of individual image files. Moreover, the time-of-flight sensor only produces depth maps with a resolution of 160x120 pixels anyway.

Figures 5 and 6 depict example frames, in which a virtual easter island statue is penetrated by a real piece of cardboard. In these figures, the images on the left show the depth data as intensity images, i.e., brighter pixels represent a larger distance to the camera image plane. Next to the depth images, the resulting output images of our occlusion-capable AR rendering pipeline are shown. One of the limitations of the current generation of time-of-flight cameras becomes apparent in these figures. In particular in Figure 5, noise contained in the acquired depth information leads to unstable occlusion handling results near the intersection of real and virtual objects. In our experience, this problem typically depends on the material of the real occluder and its angle with respect to the time-of-flight camera. Figure 6, on the other hand, shows an example of a more stable occlusion handling result.

Figure 7 shows a virtual DC10 plane model partially occluded by a real piece of cardboard. In this figure, the output image rendered by conventional AR image generation without occlusion handling is additionally depicted on the right. This clearly illustrates the comparative benefit of occlusion handling based on real depth data.

8. Discussion

The examples described in the previous section, as well as a number of additional tests not presented here, demonstrate that our prototype system is capable of handling occlusion based on 2D depth information. Thanks to the design of our combined camera setup and the registration of the time-of-flight data with the high resolution color images, a useful combined stream of depth and color images is delivered. The adaptation of various parts of the augmented reality rendering pipeline (i.e., the modification of the internal camera parameters and the application of the occlusion shader) lead to a working occlusion handling system. Although the method is currently implemented as an offline processing system, the occlusion test executed on the GPU only has a minimal impact on the rendering speed and could be used in an interactive real-time application.

However, our tests have also highlighted some of the future challenges when introducing time-of-flight depth information into augmented reality. The data delivered by the time-of-flight sensor often contains noise, and can sporadically contain extreme outliers. (This can for example be caused by total reflections of the active illumination on certain materials.) The presence of noise is evident for instance in Figures 5 and 6, particularly near the intersection of real and virtual objects and near the ARToolKit marker. An additional shortcoming, which is not visible in the examples shown here, is the fact that the time-of-flight sensor and the color camera are not perfectly synchronized. This can lead to a mismatch of color and depth information in the case of fast-moving objects in the environment. Indeed, we consider a better synchronization of the depth range images with the color images to be one of the most important challenges for future work.

9. Conclusions and Future Work

In this paper, we have presented a system which introduces time-of-flight depth information into augmented reality. We have described a hardware setup combining the depth sensor with a color camera, and our approach to the registration of the 2D images. Moreover, a prototype implementation of occlusion handling in AR based on the additional depth information was presented.

The most important future developments will deal with the current shortcomings of the depth information. Methods for adaptive interpolation or smoothing could help to reduce the impact of noise present in the data (e.g., see [HFS07]). An improved hardware- or software-based synchronization of the two sensors could make it possible to use the system in environments with fast-moving objects. In addition to these improvements, the AR rendering pipeline could be directly integrated into the software framework used for capturing the time-of-flight and color images. This way, depth-enhanced augmented reality could be demonstrated in an interactive real-time application for the first time.

The main obstacle preventing a practical use in a mobile augmented reality scenario currently is the size of the combined camera setup. In particular, the time-of-flight sensor is relatively large and heavy compared to typical digital video

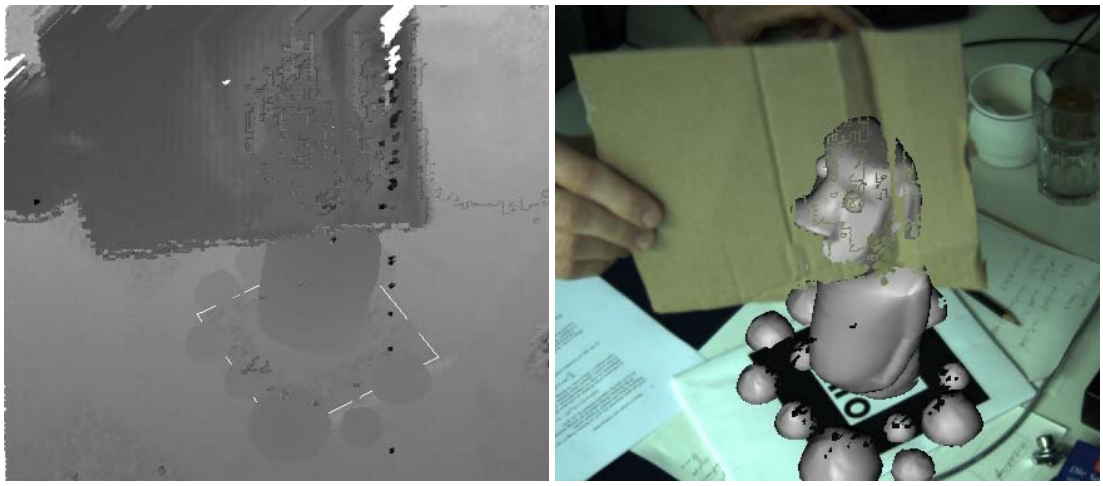

Figure 5: *Virtual easter island statue vertically penetrated by real piece of cardboard. (Left: depth data. Right: AR image with occlusion handling.)*

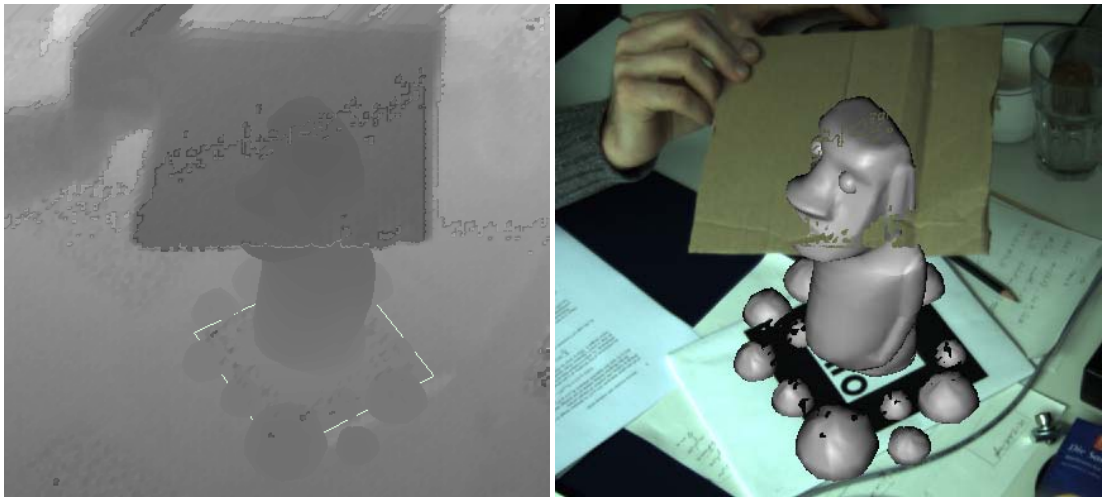

Figure 6: *Virtual easter island statue horizontally penetrated by real piece of cardboard. (Left: depth data. Right: AR image with occlusion handling.)*

cameras. However, we expect future types of depth cameras to be significantly smaller and lighter than the current generation. Moreover, an increased resolution and reduced tendency to noise can also be expected for future time-of-flight cameras.

The integration of depth information into monoscopic video see-through augmented reality can solve many of the basic problems of AR rendering. We believe that he utilization of depth data is not limited to occlusion handling, but can for instance also be useful for displaying shadows, for user interaction, and other tasks.

Acknowledgments

This work was supported by a Research Fellowship from the German Research Foundation (DFG). We would like to thank Rachel Gold for proofreading this paper.

References

[3DV07] 3DV SYSTEMS LTD: 3D Camera & 3D Video Solutions. http://www.3dvsystems.com/, 2007.

[ABB*01] AZUMA R., BAILLOT Y., BEHRINGER R., FEINER S., JULIER S., MACINTYRE B.: Recent Advances in Augmented Reality. *IEEE Computer Graphics and Applications 21*, 6 (November/December 2001), 34–47.

Figure 7: *Virtual DC10 plane model occluded by real piece of cardboard. (Left: depth data. Center: AR image with occlusion handling. Right: conventional AR display without occlusion handling.)*

[Ber97] BERGER M.-O.: Resolving Occlusion in Augmented Reality: A Contour Based Approach without 3D Reconstruction. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition* (1997), pp. 91–96.

[Bou07] BOUGUET J.-Y.: Camera Calibration Toolbox for Matlab. *available at www.vision.caltech.edu/bouguetj* (2007).

[BWRT96] BREEN D., WHITAKER R., ROSE E., TUCERYAN M.: Interactive Occlusion and Automatic Object Placement for Augmented Reality. *Computer Graphics Forum 15*, 3 (1996), 11–22.

[FBS04] FISCHER J., BARTZ D., STRASSER W.: Occlusion Handling for Medical Augmented Reality using a Volumetric Phantom Model. In *Proc. of ACM Symposium on Virtual Reality Software and Technology* (November 2004), pp. 174–177.

[FHFG99] FUHRMANN A., HESINA G., FAURE F., GERVAUTZ M.: Occlusion in Collaborative Augmented Environments. *Computers & Graphics 23*, 6 (1999), 809–819.

[FRB03] FISCHER J., REGENBRECHT H., BARATOFF G.: Detecting Dynamic Occlusion in front of Static Backgrounds for AR Scenes. In *Proc. of Eurographics Symposium on Virtual Environments* (May 2003), pp. 153–161.

[GBB*02] GORDON G., BILLINGHURST M., BELL M., WOODFILL J., KOWALIK B., ERENDI A., TILANDER J.: The Use of Dense Stereo Range Data in Augmented Reality. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality* (September 2002), pp. 14–26.

[GKOY03] GVILI R., KAPLAN A., OFEK E., YAHAV G.: Depth Keying. In *Proc. of SPIE Electronic Imaging* (2003).

[HFS07] HUHLE B., FLECK S., SCHILLING A.: Integrating 3d time-of-flight camera data and high resolution images for 3dtv applications. In *3DTV CON – The True Vision* (2007).

[KB99] KATO H., BILLINGHURST M.: Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proc. of IEEE and ACM International Workshop on Augmented Reality* (October 1999), pp. 85–94.

[Kli04] KLINKER G.: Introduction to AR - 3. OpenGL, Mixing Virtual and Real Objects. Lecture, Technische Universität München, campar.in.tum.de, November 2004.

[KOTY00] KANBARA M., OKUMA T., TAKEMURA H., YOKOYA N.: A Stereoscopic Video See-through Augmented Reality System Based on Real-time Vision-based Registration. In *Proc. of IEEE Virtual Reality* (March 2000), pp. 255–262.

[LB00] LEPETIT V., BERGER M.-O.: A Semi-Automatic Method for Resolving Occlusion in Augmented Reality. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition* (June 2000), pp. 2225–2230.

[LNWB03] LOK B., NAIK S., WHITTON M., BROOKS F.: Incorporating Dynamic Real Objects into Immersive Virtual Environments. In *Proc. of ACM Symposium on Interactive 3D Graphics and Games* (2003), pp. 31–40.

[LSKS05] LIN W., SENGUPTA K., KUMAR P., SHARMA R.: Occlusion Handling in Augmented Reality Using Background Foreground Segmentation. *Presence: Teleoperators and Virtual Environments 14*, 3 (June 2005), 264–277.

[Mat07] MATRIX VISION GMBH: mvBlueFox. http://www.matrix-vision.com/products/hardware/mvbluefox.php, 2007.

[PHW*06] PRASAD A., HARTMANN K., WEIHS W., GHOBADI S. E., SLUITER A.: First steps in ehancing 3d vision technique using 2d/3d sensors. In *Computer Vision Winter Workshop, Czech Pattern Recognition Society* (2006), Chum, Franc, (Eds.).

[PMD07] PMD TECHNOLOGIES GMBH: PMD-Cameras. http://www.pmdtec.com/e_inhalt/produkte/kamera.htm, 2007.

[Reu06] REULKE R.: Combination of distance data with high resolution images. In *ISPRS, Commission V Symposium, Image Engineering and Vision Metrology* (2006).

[Ros06] ROST R.: *OpenGL Shading Language*, 2nd ed. Addison-Wesley Publishing Company, 2006.

[SWND03] SHREINER D., WOO M., NEIDER J., DAVIS T.: *OpenGL Programming Guide*, 4th ed. Addison-Wesley Publishing Company, 2003.

[WA95] WLOKA M., ANDERSON B.: Resolving Occlusion in Augmented Reality. In *Proc. of ACM Symposium on Interactive 3D Graphics and Games* (1995), pp. 5–12.