NSF GRANT # 0825818
NSF PROGRAM NAME: CMMI/CIS

Resolving Incorrect Visual Occlusion in Outdoor Augmented Reality Using TOF Camera and OpenGL Frame Buffer

Vineet R. Kamat
University of Michigan

Suyang Dong
University of Michigan

ABSTRACT: Augmented Reality (AR) has the potential of being an effective visualization tool for planning and operations design in construction, manufacturing, and other process-oriented engineering domains. One of the primary challenges in creating AR visualizations is to project graphical 3D objects onto a user's view of the real world and create a sustained illusion that the virtual and real objects co-exist across time in the same augmented space. However regardless of the spatial relationship between the real and virtual objects, traditional AR scene composing algorithm displays the real world merely as a background, and superimposes virtual objects in the foreground. This creates incorrect visual occlusion artifacts, that in effect breaks the illusion that real and virtual objects co-exist in AR. The research implements and demonstrates a robust depth sensing and frame buffer algorithm for resolving incorrect occlusion problems in outdoor AR applications. A high-accuracy Time-of-flight (TOF) camera capable of suppressing background illumination (e.g. bright sunlight) in ubiquitous environments is used to capture the depth map of real-world in real time. The preprocessed distance information is rendered into depth buffer, that allows the interpolation of visual or hidden elements in the OpenGL color buffer to generate the composite AR scene. An optimized approach taking advantage of OpenGL texture and GLSL fragment processor is also proposed to speed up sampling distance value and rendering into frame buffer. The designed algorithm is validated in several indoor and outdoor experiments using SMART AR framework. The AR space with occlusion effect enabled demonstrates convincing spatial cues and graphical realism.

1. **Introduction:** As a novel visualization technology, Augmented Reality (AR) has gained widespread attention and seen prototype applications in multiple engineering disciplines for conveying simulation results, visualizing operations design, inspections, etc. For example, by blending real-world elements with virtual reality, AR helps to alleviate the extra burden of creating complex contextual environments for visual simulations [3]. As an information supplement to the real environment, AR has also been shown to be capable of appending georeferenced information to a real scene to inspect earthquake-induced building damage [11], or in the estimation of construction progress [9]. In both cases, the composite AR view is composed of two distinct groups of virtual and real objects, and they are merged together by a set of AR graphical algorithms.

Spatial accuracy and graphical credibility are the two keys in the implementation of successful AR graphical algorithms. Spatial accuracy requires the virtual content to be registered with the real world, which means the virtual objects must always appear at their intended location in the real environment with correct pose. In their prior work, the authors have designed and implemented a robust AR mobile computing framework called SMART and ARMOR [6]. SMART is a generic software architecture including accurate registration and projection algorithms, and ARMOR is a modular mobile hardware platform tracking user's position and orientation in the outdoor environment. Graphical credibility implies a persistent illusion that the real and virtual content in AR co-exists in the augmented space and is merged seamlessly together. There are mainly two research branches aimed at minimizing the artifacts brought by virtual objects. One of them is photorealism, which mainly deals with lighting effects, like shadows, reflections, etc. The other one is occlusion [10].

The primary focus of this research is exploring a robust occlusion algorithm for ubiquitous AR environments. In an ideal scenario, AR graphical algorithms should have the ability to intelligently blend real and virtual objects in all three dimensions, instead of superimposing all virtual objects on top of a real-world

background as is the case in most current AR approaches. The result of composing an AR scene without considering the relative depth of the involved real and virtual objects is that the graphical entities in the scene appear to "float" over the real background rather than blending or co-existing with real objects in that scene. The occlusion problem is more complicated in outdoor AR where the user expects to navigate the space freely and the relative depth between involved virtual and real content is changing arbitrarily with time. Figure 1 [4] presents a snapshot of a simulated construction operation, where two real objects (tree and truck) are closer than the virtual excavator model to the viewpoint, and should be consequently blocked by the real objects. The right side image shows visually correct occlusion where the boom and bucket are partially hidden from the scene. However the left side image shows the scene in absence of occlusion, producing an incorrect illusion that the excavator was in front of the tree and truck.

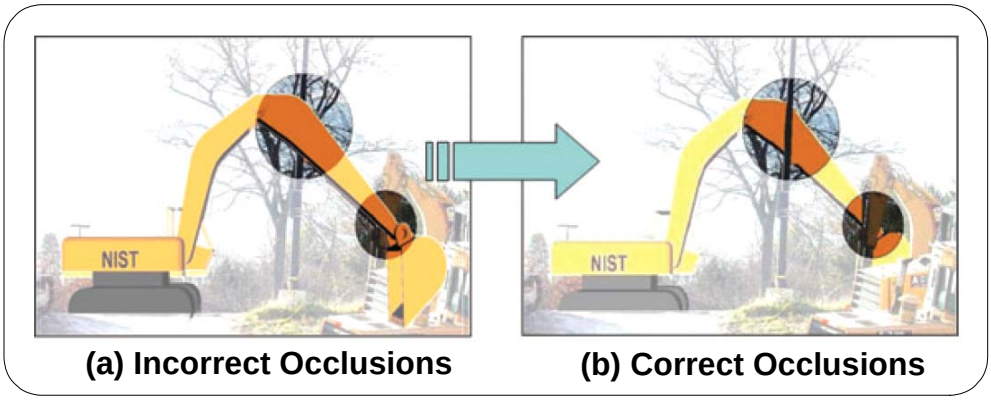

Figure 1: Example of occlusion in an outdoor AR scene

Several researchers have explored the AR occlusion problem from different perspectives: [22] implemented a fast-speed stereo matching algorithm that infers depth maps from a stereo pair of intensity bitmaps. However random gross errors blink virtual objects on and off and turn out to be very distracting; [5] proposed a contour based approach but with the major limitation that the contours need to be seen from frame to frame; [13] refined the previous method by a semi-automated approach that requires the user to outline the occluding objects in the key-views, and then the system automatically detects these occluding objects and handles uncertainties on the computed motion between two key frames. Despite the visual improvements, the semi-automated method is only appropriate for post-processing; [7] exhibited both model-based using bounding box and depth-based approach using stereo camera. The former one only works with static viewpoint, and the latter is subject to low-textured areas; [18] tried to increase the accuracy of depth map by region of interest extraction method using background subtraction and stereo depth algorithms, however only simple background examples were demonstrated; [20] also designed an interactive segmentation and object tracking method for real-time occlusion, but their algorithm fails in the situation where virtual objects are in front of real objects.

In this paper, the authors propose a robust AR occlusion algorithm that uses real-time Time-of-flight (TOF) camera data and the OpenGL frame buffer to correctly resolve the depth of real and virtual objects in AR visual simulations. Compared with previous work, this approach enables improvements in three aspects: 1) Ubiquitous: TOF camera capable of suppressing background illumination enables the algorithm and implemented system to work in both indoor and outdoor environments. It puts the least limitation on context and illumination conditions compared with any previous approach; 2) Robust: Due to the depth-buffering employed, this method can work regardless of the spatial relationship among involved virtual and real objects; 3) Fast: The authors take advantage of OpenGL texture and OpenGL Shading Language (GLSL) fragment shader to parallelize the sampling of depth map and rendering into the frame buffer. A recent publication [12] describes a parallelled research effort that adopted a similar approach for TV production in indoor environments with 3D model constructed beforehand.

2. Depth Buffer Comparison Approach: In this section, the methodology and computing framework for

resolving incorrect occlusion are introduced. This approach takes advantage of OpenGL depth buffering on a two-stage rendering basis.

2.1. Distance Data Source: Accurate measurement of the distance from the virtual and real object to the eye is the fundamental step for correct occlusion. In the outdoor AR environment, the distance from the virtual object to the viewpoint is calculated using Vincenty algorithm [21] with the geographical locations of the virtual object and the user. Location of the virtual object is documented in the data preparation phase. Meanwhile, location of the user is tracked by Real-time Kinematic (RTK) GPS. The ARMOR platform utilizes Trimble AgGPS 332 along with Trimble AgGPS RTK Base 450/900 to continuously track the user's position up to centimeter level accuracy.

On the other hand, TOF camera estimates the distance from the real object to the eye with the help of time-of-flight principle, which measures the time a signal travels, with well defined speed spends, from the transmitter to the receiver [2]. The chosen PMD CamCube 3.0 utilizes Radio Frequency (RF) modulated light sources with phase detectors. The modulated outgoing beam is sent out with a RF carrier, and the phase shift of that carrier is measured on the receiver side to compute the distance [8]. Compared with traditional LIDAR scanners and stereo vision, TOF camera possesses ideal features of being deployed in real-time applications: captures a complete scene with one shot and speeds up to 40 frames per second (fps). However TOF camera is vulnerable to background light, like artificial lighting and sun that also generates electrons and confuses the receiver. Fortunately the Suppression of Background Illumination (SBI) technology allows PMD CamCube 3.0 to work flexibly in both indoor and outdoor environment. [16]

2.2. Comparison of Heterogeneous Distance Data Source: The distance from the virtual and real object to the viewpoint cannot be compared directly since they are represented in heterogeneous forms. While the distances from the real object to the viewpoint are directly given for each pixel by TOF camera, the distance between the viewpoint and virtual object occupying a group of fragments cannot be retrieved until all the vertices are processed through the OpenGL graphics pipeline and reach the depth buffer. [19]

Depth buffering, also known as z-buffering, is the solution for hidden-surface elimination in OpenGL and is usually done efficiently in the graphics card or GPU. Depth buffer is a two-dimensional array that shares the same size with the viewport, and always keeps record of the closest depth value to the observer for each pixel. For a new candidate color arriving at a certain pixel, it will not be drawn unless its corresponding depth value is smaller than the previous one. If it is drawn, then the corresponding depth value in the depth buffer will be replaced by the smaller one. In this way, after the entire scene has been drawn, only those items not obscured by any others remain visible.

Depth buffering thus provides a promising approach for solving the AR occlusion problem, and Figure 2 shows the two rendering stage method: In the first rendering stage, the background of the real scene is drawn as usual but with the depth map retrieved from TOF camera written into the depth buffer at the same time. In the second stage, the virtual objects are drawn with depth buffer testing enabled. Consequently, the invisible part of virtual object, either hidden by real object or another virtual one, will be correctly occluded.

2.3. Problems with Depth Buffering Comparison Approach: Despite the simplicity and straightforward

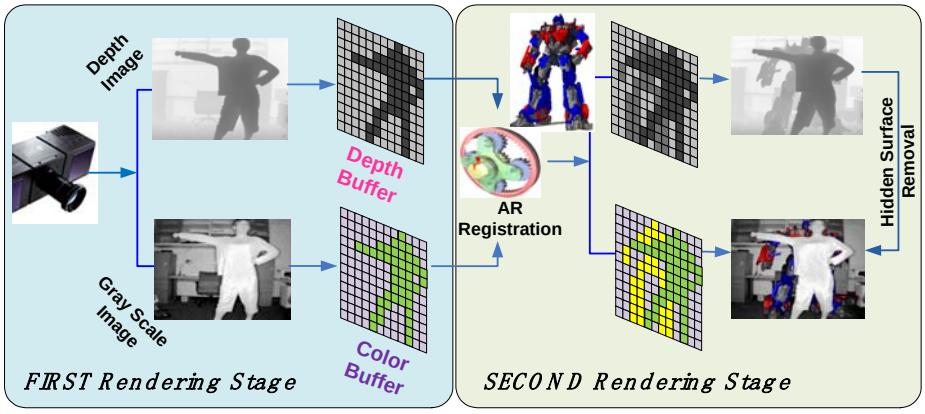

Figure 2: Two Stages Rendering

approach of depth buffering, there are several challenges when feeding depth buffer with TOF camera distance information:

1) Depth buffer value does not represent the actual distance between the object and the viewpoint but the distance after a projection transformation, division and normalization to the range [0, 1]. The conversion process from actual distance to depth buffering distance will be addressed in section 3.

2) Traditional glDrawPixels() command can be extremely slow when writing a two-dimensional array into the frame buffer. Section 4 introduces an alternative and efficient approach using OpenGL texture and GLSL.

3) The resolution of TOF depth map is fixed as 200*200 while that of the viewport and depth buffer can be arbitrary. This implies the necessity of interpolation between the TOF depth map and the depth buffer. Section 4 also takes advantage of OpenGL texture to fulfill interpolation task efficiently.

4) There are three cameras for rendering an AR space: Video camera captures RGB or intensity values of the real scene as the background, and its result is written into the color buffer; TOF camera acquires the depth map of the real scene, and its result is written into the depth buffer; OpenGL camera projects virtual objects on top of real scene with its result written into both color and depth buffer. To ensure correct registration and occlusion, all of them have to share the same projection parameters: aspect ratio and field of view. While the projection parameters of OpenGL camera are adjustable, video camera conflicts with TOF camera in three aspects: a) The centers of camera do not agree; b) video camera usually comes with aspect ratio of 640:480, but CamCube has its aspect ratio as 200:200; c) vertical field of view of ordinary video camera is around 32°, but CamCube reports wide field of view as 40°. The alignment of the TOF camera with the video camera is still an issue under study as shown in Figure 3. Currently the authors take advantage of gray scale image captured by TOF camera simultaneously with distance information, to work around the mismatching projection parameters problem.

3. Raw Data Preprocessing: Before feeding any gray scale image and depth map into the color buffer and depth buffer respectively, both of them need to be preprocessed to meet the data type and pixel format requirement of the frame buffer.

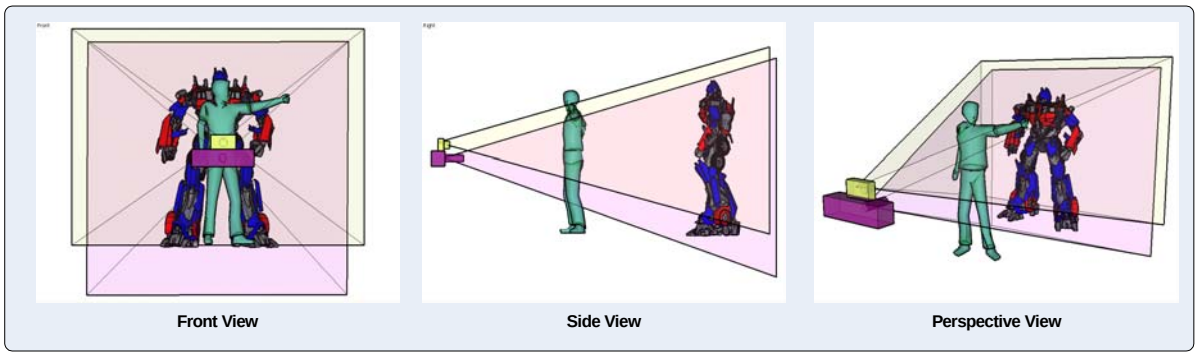

Figure 3: Projection parameters disagreement between TOF Camera (Purple) and Video Camera (Yellow)

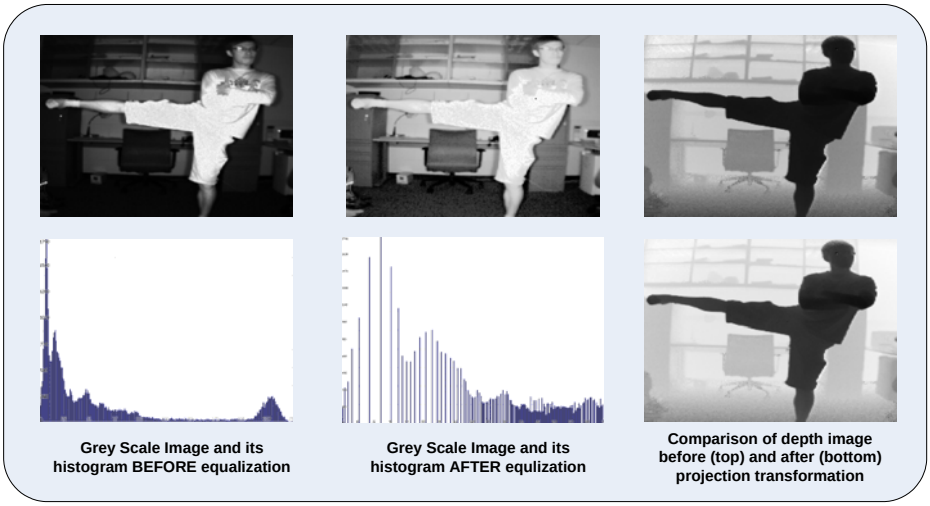

Figure 4: Preprocessing of the TOF gray scale and depth image

3.1. Preprocessing of Gray Scale Image: Since unsigned byte (8 bits represents 0 ~ 255) is the data type for showing intensity values, the raw gray scale image needs to be refined in two aspects. First of all, the raw intensity values spread from 1000 to 20,000 and thus have to be redistributed on [0,255]; secondly, the gray scale image is represented by close contrast values, and histogram equalization is helpful in spreading out the most frequent intensity values on the histogram for better visual effects. The basic idea of equalization is to linearize the cumulative distribution function (CDF) across the histogram from 0 to 255. The transformation can be described by the formula.1 [1]. CDF is the cumulative distribution function of a given gray scale image; v is the original intensity value of a given pixel, P(v) is the intensity value after equalization for that pixel; Level is the total number of gray scale after equalization.

$$P(v) = \frac{CDF(v) - CDFmin}{(width * height) - CDFmin} * (Level - 1)$$

3.2. Preprocessing of Depth Map: Since TOF camera is aligned with video camera (section 2.3), the distance value provided by TOF camera is treated within the eye coordinate system as Ze (actual distances from vertices to the viewer in viewing direction). In the OpenGL pipeline, several major transformation steps are applied on ze before its value is written into the depth buffer. Table 1 summarizes the transformation procedure, and more detailed information is available from [14]: 1) clip coordinate Zc (distance values in clip space where objects outside the view volume are clipped away) is the result of transforming vertices in eye coordinate by projection matrix; 2) Zc divided by Wc (homogenous component in clip space) is called perspective divide that generates Zndc; 3) Since the range of Zndc (distance values in normalized device coordinate (NDC) space that encompasses a cube and is screen independent) is [-1,1], it needs to be offset and scaled by the depth buffer range [0,1] before it is sent to the depth buffer.

Table 1: The transformation steps applied on the raw TOF depth image.

Name	Meaning	Operation	Range
Ze	Distance to the viewpoint	Acquired by TOF camera	$(0, +\infty)$
Zc	Clip coordinate after projection transformation	$M_{ortho} * M_{perspective} * [Xe\ Ye\ Ze\ We]^T$	[-n,f]
Zndc	Normalized device coordinate	*Zc / Wc* (*Wc = Ze, and* is the homogenous component in clip coordinate	[-1,1]
Zd	Value sent to depth buffer	*(Zndc+1) / 2*	[0,1]

4. Using Texture to Render to Frame Buffer: This section describes how to efficiently render to the color and depth buffer using texture and GLSL. After preprocessing of the gray scale image and the depth map, they are ready to be written into the frame buffer. However a challenging issue is how to write to frame

buffer fast enough so that real time rendering is possible: 1) the arbitrary size of the color and depth buffer requires interpolation of the original 200*200 image. While software interpolation can be very slow, texture filtering presents a hardware solution here since texture sampling is so common that most graphics cards implement it very fast; 2) even though glDrawPixels() command with GL_DPETH_COMPONENT parameter provides an option for writing array into depth buffer, no modern OpenGL implementation can efficiently accomplish this since data is passed from main memory

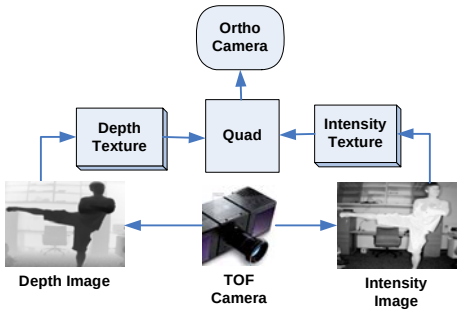

Figure 5: General structure of attaching multiple textures to the quad that is the real scene background

to OpenGL to graphics card on every single frame. On the other hand, texturing a QUAD and manipulating its depth value in GLSL fragment shader can be very efficient.

Texture is the container of one or more images in OpenGL [19], and is usually bound to geometry. Here the OpenGL geometric primitive type GL_QUADS is chosen as binding target, and two 2D textures are pasted on it (Figure. 5). One is gray scale image texture, and the other one is depth map texture. The quad shares the same size with the virtual camera's viewport, and projected orthogonally as the background.

4.1. Gray Scale Image Texture: Since modifying the existing texture object is computationally cheaper than creating a new one, it is better to use glTexSubImage2D() to replace repeatedly the texture data with new TOF images. [19] However the TOF image must be loaded to an initial, larger texture with size in both directions set to the next biggest power of two than 200, namely 256. Accordingly the texture coordinates are assigned as (0, 0), (200/256, 0), (200/256, 200/256), (0, 200/256) in counterclockwise order of the quad.

4.2. Depth Map Texture: The same sub image replacement strategy is applied on depth map texture. However even though internalformat of texture is set to GL_DEPTH_COMPONENT, the depth value written into depth buffer is not the depth map texture, but the actual depth value of the quad instead. Therefore the depth value of the quad needs to be manipulated in fragment shader according to the depth map texture. A fragment shader operates on every fragment which is produced by rasterization. One input for the fragment processor is interpolated texture coordinates, and the common end result of the fragment processor is a color value and a depth for the fragment [17]. These features make it possible to redefine polygon depth value so that the TOF depth map can be written into the depth buffer.

5. Validation: The indoor validation experiment is carried out in the authors' office. Optimus Prime (model accredited to mandun from Google 3D Warehouse community) is located approximately 2.5m away from the TOF camera, and the author is standing right behind Optimus Prime "grabbing" its waist and shoulders. A series of images (Figure. 6) shows correct occlusion effects where the author is largely hidden by Optimus Prime except his arms that are spatially in front of Optimus Prime.

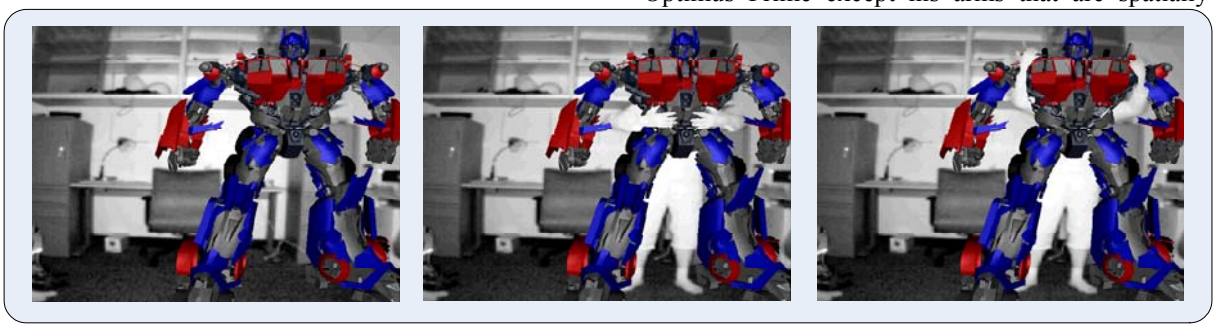

Figure 6: Indoor validation experiment

Despite the outstanding performance of TOF camera in speed and accuracy, the biggest technical challenge of it is the modular error, since the receiver decides the distance by measuring the phase offset of the carrier. Ranges are mod the maximum range, which is decided

by the RF carrier wavelength. For instance, the standard measurement range of CamCube3.0 is 7m. [16] If an object happens to be 8m away from the camera, its distance is represented as 1m (8 mod 7) on the depth map instead of 8m. This can bring incorrect occlusion in outdoor condition, where ranges can easily go beyond 7m. The authors have been looking into object detection, segmentation etc. to mitigate the limitation. For now, the experiment range is intentionally restricted to within 7m.

A layout of the experiment is shown in Figure 7. The TOF camera is positioned approximately 7m away from the wall of a building, which is surrounded by a flower bed. A small excavator model (accredited to J-m@n from Google 3D Warehouse community) is positioned about 3m away from the TOF camera, and the author is standing in front of the excavator. Scenarios for both occlusion function enabled and disabled are shown. It is obvious that occlusion provides much better spatial cues and realism for outdoor AR visual simulation.

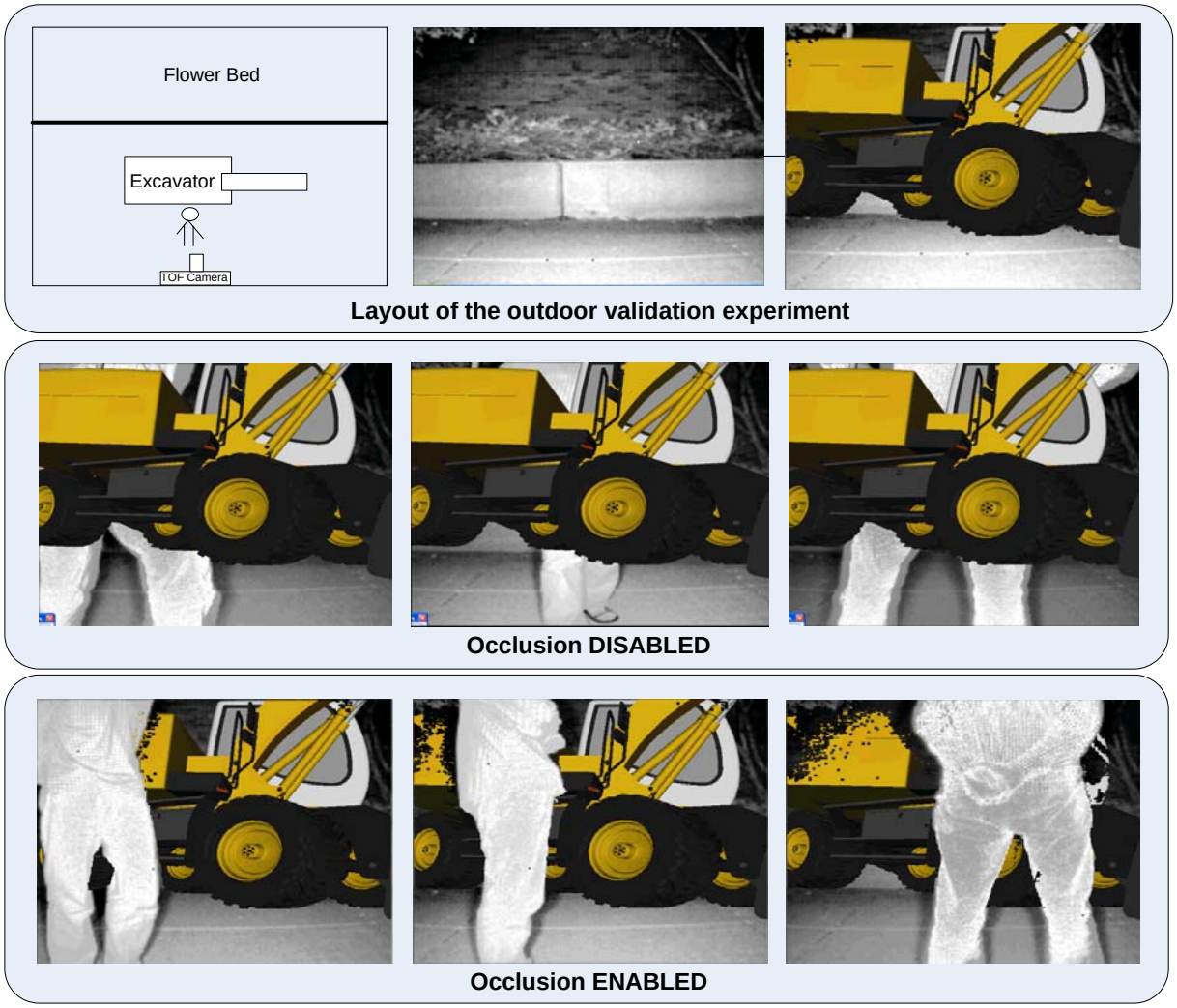

Figure 7: Outdoor Validation Experiment

6. Conclusion and Future Work: This paper described research that designed and implemented an innovative approach to resolve AR occlusion in ubiquitous environments using real-time TOF camera distance data and the OpenGL frame buffer. The first set of experimental results demonstrated promising depth visual cues and realism in AR visual simulations. However, several challenging issues remain outstanding and are currently being investigated by the authors. For example, the gray scale image captured by TOF camera is currently displayed as a background and thus no extra geometric transformation needs to be applied on distance values from TOF camera. The authors are attempting to align projection parameters of TOF camera with that of video camera so that the

background can be drawn in full color. Secondly, the authors acknowledge that the current 7m average operational range of TOF camera puts a limitation on fully outdoor simulation visualization. However the occlusion algorithm designed here is generic and scalable so that future hardware with improved range and accuracy can be plugged into the current AR visualization system with little modification to the core algorithm. Meanwhile, the authors are studying the feasibility of implementing hybrid methods, like stereo vision and object detection, to mitigate this limitation.

7. Acknowledgments: The presented work has been supported by the United States National Science Foundation (NSF) through Grants CMMI-0448762 and CMMI-0825818. The authors gratefully acknowledge NSF's support. Any opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF.

8. Reference

[1] Acharya, T., & Ray, A. K. (2005). Image processing : principles and applications, John Wiley & Sons, Inc.

[2] Beder, C., Bartczak, B., & Koch, R. (2007). A Comparison of PMD-Cameras and Stereo-Vision for the Task of Surface Reconstruction using Patchlets, Computer Vision and Pattern Recognition, Minneapolis, MN, 1-8. .

[3] Behzadan, A. H., & Kamat, V. R. (2009a). Scalable Algorithm for Resolving Incorrect Occlusion in Dynamic Augmented Reality Engineering Environments, Journal of Computer-Aided Civil and Infrastructure Engineering, Vol. 25, No.1, 3-19.

[4] Behzadan, H. A., & Kamat, R. V. (2009b). Automated Generation of Operations Level Construction Animations in Outdoor Augmented Reality, Journal of Computing in Civil Engineering, Vol.23, No.6, 405-417.

[5] Berger, M.-O. (1997). Resolving Occlusion in Augmented Reality: a Contour Based Approach without 3D Reconstructioin, Proceedings of CVPR (IEEE Conference on Computer Vision and Pattern Recognition, Puerto Rico.

[6] Dong, S., & Kamat, V. R. (2010). Robust Mobile Computing Framework for Visualization of Simulated Processes in Augmented Reality, Proceedings of the 2010 Winter Simulation Conference, Baltimore, USA.

[7] Fortin, P.-A., & Ebert, P. (2006). Handling Occlusions in Real-time Augmented Reality: Dealing with Movable Real and Virtual Objects, Proceedings of the 3rd Canadian Conference on Computer and Robot Vision, Laval University, Quebec, Canada.

[8] Gokturk, B. S., Yalcin, H., & Bamji, C. (2010). A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions, Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop, Washington, DC, USA, 35-44.

[9] Golparvar-Fard, M., Pena-Mora, F., Arboleda, C. A., & Lee, S. (2009). Visualization of construction progress monitoring with 4D simulation model overlaid on time-lapsed photographs, Journal of Computing in Civil Engineering, Vol. 23, No. 4, 391-404.

[10] Grau, O. (2006). 3D in Content Creation and Post-Production. In Oliver Schreer, Peter Kauff, T.S., ed.: 3D Videocommunication. 39-53.

[11] Kamat, V. R., & El-Tawil, S. (2007). Evaluation of Augmented Reality for Rapid Assessment of Earthquake-Induced Building Damage, Journal of Computing in Civil Engineering, 21(5), 303-310.

[12] Koch, R., Schiller, I., Bartczak, B., Kellner, F., & Koser, K. (2009). MixIn3D: 3D Mixed Reality with ToF-Camera, Lecture Notes in Computer Science, Vol. 5742, 126-141.

[13] Lepetit, V., & Berger, M.-O. (2000). A Semi-Automatic Method for Resolving Occlusion in Augmented Reality, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Hilton Head, South Carolina.

[14] Mcreynolds, T., & Blythe, D. (2005). Advanced Graphics Programming Using OpenG, San Francisco, CA: Dlsevier Inc.

[15] OSGART. (2010). Main Page. Retrieved from OSGART: http://www.osgart.org/wiki/Main_Page

[16] PMD. (2010). PMD CamCube 3.0. Retrieved July 2010, from PMD Technologies: http://www.pmdtec.com/fileadmin/pmdtec/downloads/documentation/datenblatt_camcube3.pdf

[17] Rost, R. J., Licea-Kane, B., & ginsberg, d. (2009). OpenGL Shading Language (3rd Edition), Addison-Wesley Professional.

[18] Ryu, S.-W., Han, J.-H., Jeong, J., Lee, S. H., & Park, J. I. (2010). Real-Time Occlusion Culling for Augmented Reality, 16th Korea-Japan Joint Workshop on Frontiers of Computer Vision, Hiroshima Japan, 498-503.

[19] Shreiner, D., Woo, M., Neider, J., & Davis, T. (2006). OpenGL Programming Guide (Fifth Edition).

[20] Tian, Y., Guan, T., & Wang, C. (2010). Real-Time Occlusion Handling in Augmented Reality Based on an Object Tracking Approach, Sensors , 2885-2900.

[21] Vincenty, T. (1975). Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations, In Survey Reviews, Ministry of Overseas Development, 88-93.

[22] Wloka, M. M., & Anderson, B. G. (1995). Resolving Occlusion in Augmented Reality. Symposium on Interactive 3D Graphics Proceedings ACM New York, NY, USA, 5-12.