Occlusion of Virtual Objects for Augmented Reality Systems using Kinect

João Vitor de Sá Hauck
Computer Science Department
Universidade Federal de Juiz de Fora
Email: jhauck@ice.ufjf.br

Matheus R. F. Mendonca
Computer Science Department
Universidade Federal de Juiz de Fora
Email: matheus.mrfm@gmail.com

Rodrigo Luis de Souza da Silva
Computer Science Department
Universidade Federal de Juiz de Fora
Email: rodrigoluis@ice.ufjf.br

***Abstract*—This work presents a novel approach to creating realistic occlusions of virtual and real objects placed inside a real environment in an augmented reality system. In this work we use the Kinect depth sensor together with AVRLIB library in order to position and occlude virtual objects into the scene. This framework creates a realistic and robust system capable of rendering and effectively occluding a virtual object inside a real environment in real time, without the need of any preprocessing.**

I. Introduction

Augmented reality is an interesting technique and it has several applications, such as architectural planning, gaming and even technological devices like Google glass. Despite this there are few works that concerns with occlusion in virtual objects. The use of occlusions leads to a much more realistic system as the virtual objects interacts better with the real ones.

The main problem when dealing with occlusion is that usually there is no depth information of the scene. Estimation of the depth information is computationally expensive. So, in our work, rather than estimate the depth of the scenario, we use the Microsoft Xbox Kinect^{TM}sensor to get this information.

Augmented reality may also be with or without markers. The ideal system is fast, robust, precise and not intrusive. Commonly, systems that use markers are more robust, simpler and faster than the ones without. However you have to include a strange object into the scene. In this work we use markers due it is simplicity and robustness.

II. Related Works

In an effort to create realistic augmented reality systems, it has been created different approaches to tackle the problem of creating a precise occlusion of virtual objects placed in a real environment. In this section, we describe some of these approaches.

In [1] it is presented two approaches to create realistic interactions between real and virtual objects. The first, called model-based method, requires a virtual object to be registered to a real object, in an effort to recreate a real object based on a virtual object with a similar shape. It then converts the object position to the camera coordinates, calculating the occlusion of each object. As for the second approach, called depth-based method, uses Weng's algorithm [2] to generate a dense depth-map of the environment. The depth of each pixel is then used to determine the occlusion of the virtual objects placed in the scene.

A more recent work, described in [3], tackles the occlusion problem by using a SLAM tracking system to acquire a set of feature points of the real environment and then determines which of these points belongs to a plane, in an effort to find every planar object in the scene. The planes encountered are used to determine the position of real foreground objects, such as a hand. It then considers that every virtual object is positioned between a plane and a foreground object. Thus, only foreground objects can occlude a virtual object, constraining this technique to environments where planes constitute a background and virtual objects are always between a background and a foreground object. Furthermore, the method described requires an environment scanning before the occlusion can actually take place.

A different approach is presented in [4], in which is used a marking system that allows users to mark the background and the real object that will occlude the virtual object. The marking process is made directly upon the video. Through this technique, it is extracted a set of feature points of the real object, followed by the retrieval of its exact border, allowing the program to track the object throughout the video. A virtual object, placed in the real environment, will then be occluded by the real object and occlude the background. Although this technique is capable of tracking the real object and performing the occlusion process during the video capture in real time, it suffers from unrealistic effects when the real object is placed behind the virtual object, since the first will still occlude the last even when it belongs to the background.

In the current work, we use the Microsoft Kinect to aid in the process of acquiring the depth map of the environment. This system has been largely used in academic research due to its efficiency in capturing the depth value of a scene. Some examples of systems created using the Kinect are as follows: a controller-free system for medical images visualization [5], a multi-purpose system that allows users to control avatars or virtual characters in remote environments [6], human detection using depth information [7], hand detection and gesture recognition using depth information [8] and *Kinect Fusion* [9], a system capable of building a 3D model of the environment in real-time [10] based on the depth map of the objects present in the scene.

III. Proposed method

In most augmented reality systems, the virtual objects are rendered over the scene, disregarding the depth of the real

objects. This way, real objects are always placed in background, while the virtual object compose the foreground of the resulting scene. There has been several attempts to extract depth information of a real environment, as described in II, but there are always drawbacks regarding the proposed method. In an attempt to create a robust, fast and realistic augmented reality system, we use a very powerful device, the Microsoft Kinect, that is capable of retrieving the depth information of an environment in a simple and efficient fashion, eliminating the need of any preprocessing operation.

Although the depth information retrieval composes a critical process for creating a realistic augmented reality system, the insertion of a virtual object also presents its own difficulties, such as the location, depth, scale, rotation and tracking of the artificial object. To tackle these problems we make use of the *AVRLib* [11], an augmented reality library based on the well known *ARToolkit* library [12], that is specialized in markers detection. With this tool, the positioning of the virtual objects becomes practical and efficient.

In this section, we describe the tools used for this work, as well as the methods used to integrate them in a single project.

A. Depth Information

To obtain the image and depth information of the environment, we use the Kinect. This way, we can recreate the scene with more than just its visual properties (obtained as an RGB image), making use of the depth information, which is crucial to creating realistic occlusions of virtual objects.

Kinect uses a depth camera, capable of obtaining the depth value for each pixel of the scene. The depth value varies from 0 to 10000, where 0 is attributed to the pixels that represents the closest object and 10000 to the ones that represents the farthest objects. The interface between the Kinect system and the computer was made using Freenect, an open source library for Kinect.

The Kinect system has two different streams: the first one has the RGB image and the second has the depth information. Due to physical limitations the depth sensor and the camera are not exactly in same position. Thus, the depth stream and the RGB image must be aligned. The Kinect depth camera also presents distance limitations, as it can only acquire the depth information of objects within a distance between 800 and 5000 millimeters.

B. Virtual Object Positioning

A very important aspect of a augmented reality system is the virtual object positioning method used to display the object in the correct spot of the scene. Placing a virtual object inappropriately in the environment leads to an unrealistic and non-immersive augmented reality system, thus, it is of extreme importance to adopt a reliable method to place these objects in the scene.

In this work, the virtual object must be placed according to the user's desire, since it is the latter that interacts with the system and determines the correct position of the object. To add more immersion to the resulting system, the user must be able to determine the virtual object's position in real time through an intuitive method, otherwise, the practical value of the system is affected.

The virtual object placement was solved by using the AVRLib, a library that offers all the basic tools for marker detection. With this tool, the user must place a marker in a visible place to allow the system to detect it and create a virtual object in its spot. The object to be rendered can be changed to any object with an appropriate object file that describes its properties, thanks to a specialized object handler developed, described in subsection III-C.

C. Object Handler

The resulting system proposed in this work may be used in any application focused on evaluating objects positioning in a real environment before the real objects are actually moved. Thus, the system must allow the user to choose any virtual model to be rendered in the scene, and not only a predetermined set of virtual models.

To accomplish this task, it was developed an object handler capable of loading and rendering objects described in *.off*, *.ply* and *.obj* formats. The object handler loads the object shape, geometry, scale and texture from the file, while the marker system deals with the object's rotation and position.

D. Recreating the Environment

When recreating a real environment, it is usually used a simple plane with an image of the scene as a texture. This method, yet simple and efficient for a great deal of augmented reality applications, does not take into account the depth information of the environment. To recreate a scene using the depth information, it is necessary to render it in a three dimensional space. Thus, we create the scene by drawing points in a 3D space, where each point represents one pixel in the output RGB image. The scene rendering is made using *OpenGL*.

After drawing the points in a 3D space, we create the virtual object in the scene, where its position and rotation are determined by the marker used, as described in III-B. Since the resulting virtual environment is entirely created using the depth information, there is no distinction of virtual and real objects, hence, the occlusion occurs naturally when the scene is rendered by the *OpenGL* library.

With this scene rendering method, a projection problem arises, since the set of points that represents the environment must be rendered in orthographic projection so the points are aligned in the X, Y plane, obscuring the depth differences between each one of them, and the virtual objects must be rendered in perspective, in an effort to give a realistic view of the object. But, as mentioned before, the virtual representation of the real environment and the virtual objects are blended in the same scene. Thus, we need to render the entire scene using a single projection and apply a projection change matrix to either the set of points or the virtual objects.

We naturally chose the perspective projection, as it minimizes the changes in the AVRLib library. In order to render a pixel on the screen using a perspective projection we draw a rectangle in such a way that after the projection it will be positioned on the correct pixel and measure just like a pixel

measure. To do so we first must correct the position of the pixel that we will draw, which leads us to an transformation that is exactly the inverse of the projection, where points that are closer to near plane have to shrink, while those that are distant from the near plane must be magnified. The process is described in the Equation 1:

$$(X, Y, Z) = \left(X \cdot tan\left(\frac{fov}{2}\right) \cdot Z \cdot As, X \cdot tan\left(\frac{fov}{2}\right) \cdot Z, Z \right), \quad (1)$$

where the vector (X, Y, Z) is the point's position, fov is the aperture angle used in the projection matrix and AS is the aspect ratio. However there is another problem. We must use the same fov used in the AVRLib library, but we do not know it. So we have to infer it from the projection matrix P used by the library. As the first element of the matrix P is defined by Equation 2, so $fov = cotan^{-1}(P_{1,1} \cdot As)$. This leads us to a fov equals to 37.8^o degrees.

$$P_{1,1} = \frac{cotan\left(\frac{fov}{2}\right)}{As} \quad (2)$$

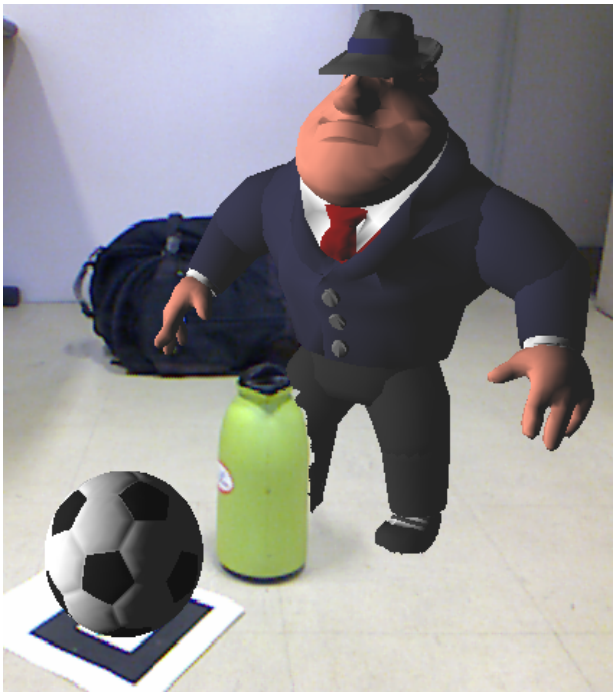
IV. Results

In this section we discuss the generated results of the proposed method. It was implemented using C++ programming language and compiled using GCC 4.8.2. All tests were performed in an Intel Xeon(R) CPU E31220 @ 3.10GHz x 4 computer with 8 GBs of RAM. The graphic card was an AMD Radeon HD 5700 series.

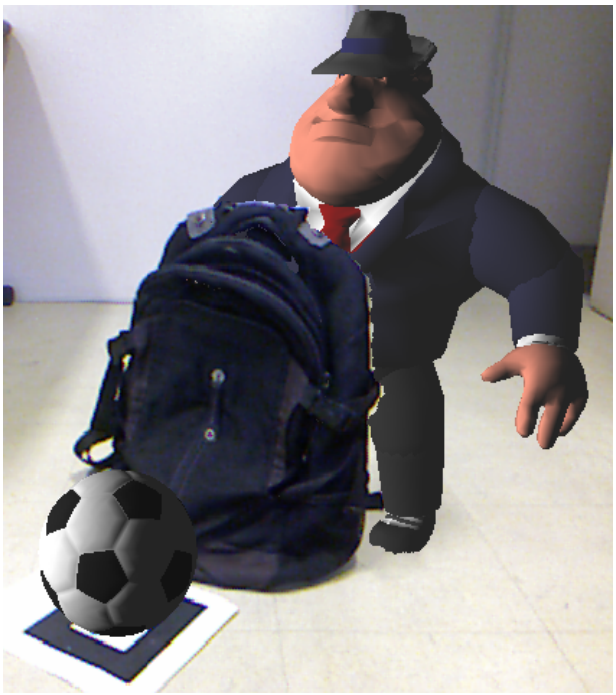
The resulting system is an augmented reality system capable of rendering virtual objects in a real environment in a realistic and efficient fashion, performing the occlusions between virtual and real objects. The test shows that the system described in this paper is a robust and reliable method able to solve the occlusion problem without the need of any preprocessing and without any major drawbacks, as seen in other works described in II.

In Figure 1, we demonstrate the output image of different objects layout. In Figure 1(a), two virtual objects were created associated to a single marker, where one of the objects is placed at the same location as the marker and the other one is translated in the X axis. It is also placed two real objects in the scene: a bottle, placed between the virtual objects, and a backpack, placed behind them. As shown in the illustration, the bottle occludes the furthest virtual object but suffers occlusion from the nearest one and the backpack is occluded by both of the virtual objects. Figure 1(b) shows a similar result, but this time, the backpack is placed between the virtual objects.

The accuracy of the real objects boundaries is directly related to the precision of Kinect depth camera. To demonstrate this precision, it is shown in Figure 2 how the resulting system deals with small objects, such as fingers. In Figure 2(a), the hand is placed over the virtual character. The resulting image shows that the depth camera is not capable of acquiring precisely the finger boundaries (although it is capable of detecting them), resulting in aliasing problems. Figure 2(b) demonstrates the virtual character occluding the hand, when the latter is placed further in the scene.



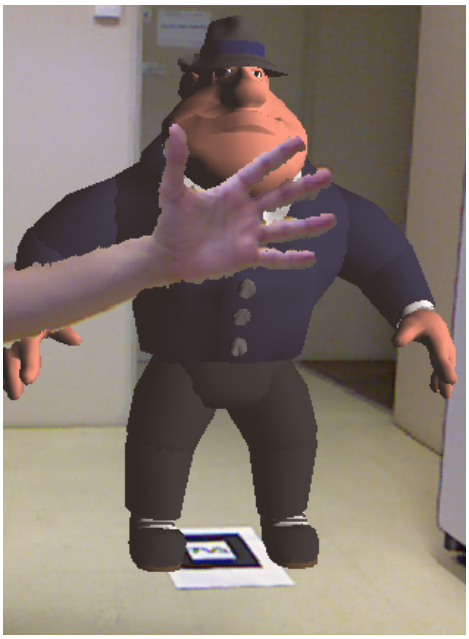
(a) Real objects behind and between the two virtual objects.



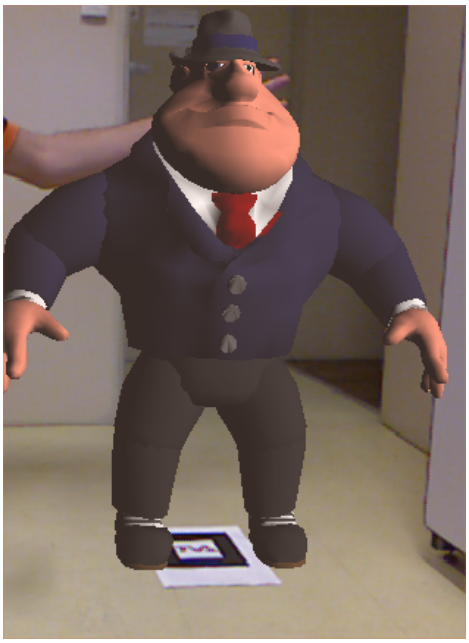
(b) Real object between the two virtual objects.

Fig. 1: Simple occlusion example.

In Figure 3, it is demonstrated a case where a virtual object is placed under a chair. Thus, the further chair legs are occluded by the virtual car, while the nearest legs occludes it. In another scenario, we place a virtual table in the environment, with a real chair behind it and a real box between the table's legs, as shown in Figure 4, where the occlusions all occur in a realistic manner.



(a) Hand over the virtual object.



(b) Hand behind the virtual object.

Fig. 2: Hand occlusion example.

One of the drawbacks presented in the system is the constant presence of the marker in the scene, as shown in the Figures 1, 2, 3 and 4, where it has to be visible for the correct execution of the system. The results also highlight another issue: the imprecision of the real objects detections, resulting in aliasing problem when a real object occludes a virtual object, as shown in Figures 2(a) and 3. At last, the system is constrained to small real environments due to the range capabilities of the Kinect depth camera.

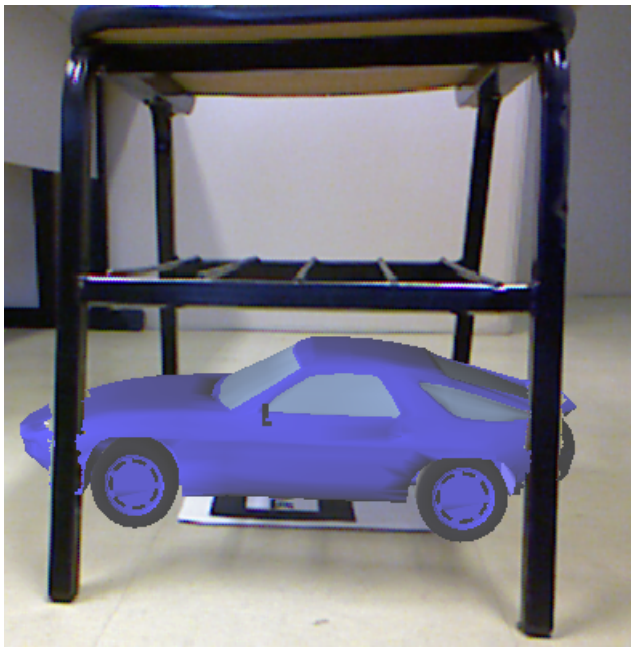


Fig. 3: Virtual object between real objects.

Fig. 4: Virtual table rendered over a real chair and a real object between the virtual object.

V. Conclusion and Future Works

We presented an augmented reality system capable of performing the occlusion of virtual objects by real or other virtual objects using the Kinect depth camera and the *AVRLib*. The real environment is virtually recreated with dots in a three dimensional space, where the X and Y coordinates and color are taken from the RGB video taken from the Kinect and the Z coordinate from the depth camera. Then, the virtual object is placed in the scene in the position obtained by the marker detection system. At last, the final scene is rendered in a perspective projection, where, for each point representing the real environment, is applied a transformation to its coordinates to simulate a orthographic projection. This way, the final render uses different projections for the environment and the virtual objects. The system is capable of rendering any virtual model in the scene, given there is a correct model file describing the object. Furthermore, the system renders everything in real-time.

Our system can be used in a great deal of different applications, ranging from design analyses, games, interactive systems and any other application that requires a realistic virtual object occlusion simulation. Although it presents some advantages, it still suffers some issues: the systems must be used in small areas, ranging from 800 to 5000 millimeters, the obligatory presence of the markers and the aliasing problem when a real object occludes a virtual object.

Based on the drawbacks highlighted in this paper, some future works are proposed. First, the application of an image processing algorithm in the resulting image in an effort to diminish the aliasing problem detected in the occlusion of virtual objects by real objects. As a final proposal, the application of *inpainting* algorithms to remove the marker from the resulting image might be interesting, in order to increase the realism and immersion of the system.

ACKNOWLEDGMENT

The authors would like to thank FAPEMIG and CAPES for the financial support.

REFERENCES

[1] D. E. Breen, R. T. Whitaker, E. Rose, and M. Tuceryan, "Interactive Occlusion and Automatic Object Placement for Augmented Reality," *Computer Graphics Forum*, vol. 15, no. 3, pp. 11–22, 1996.

[2] J. Weng, T. S. Huang, and N. Ahuja, "Motion and Structure From Two Perspective Views: Algorithms, Error Analysis, and Error Estimation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 5, pp. 451–476, May 1989.

[3] J. Ventura and T. Hollerer, "Online environment model estimation for augmented reality," in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, Oct 2009, pp. 103–106.

[4] Y. Tian, T. Guan, and C. Wang, "Real-time occlusion handling in augmented reality based on an object tracking approach," *Sensors*, vol. 10, no. 4, pp. 2885–2900, 2010. [Online]. Available: http://www.mdpi.com/1424-8220/10/4/2885

[5] L. Gallo, A. Placitelli, and M. Ciampi, "Controller-free exploration of medical image data: Experiencing the kinect," in *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, June 2011, pp. 1–6.

[6] A. Nagendran, R. Pillat, A. Kavanaugh, G. Welch, and C. Hughes, "Amities: Avatar-mediated interactive training and individualized experience system," in *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '13. New York, NY, USA: ACM, 2013, pp. 143–152. [Online]. Available: http://doi.acm.org/10.1145/2503713.2503731

[7] L. Xia, C.-C. Chen, and J. Aggarwal, "Human detection using depth information by kinect," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, June 2011, pp. 15–22.

[8] Z. Ren, J. Meng, J. Yuan, and Z. Zhang, "Robust hand gesture recognition with kinect sensor," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 759–760. [Online]. Available: http://doi.acm.org/10.1145/2072298.2072443

[9] S. Izadi, R. A. Newcombe, D. Kim, O. Hilliges, D. Molyneaux, S. Hodges, P. Kohli, J. Shotton, A. J. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time dynamic 3d surface reconstruction and interaction," in *ACM SIGGRAPH 2011 Talks*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 23:1–23:1. [Online]. Available: http://doi.acm.org/10.1145/2037826.2037857

[10] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '11. New York, NY, USA: ACM, 2011, pp. 559–568. [Online]. Available: http://doi.acm.org/10.1145/2047196.2047270

[11] D. C. B. Oliveira, F. A. Caetano, and R. L. S. Silva, "Avrlib - an object oriented augmented reality library," *Workshop de Realidade Virtual e Aumentada (WRVA)*, pp. 54–59, 2013.

[12] H. Kato, "Artoolkit 2.33 documentation (alpha version)," *Human Interface Technology Laboratory, University of Washington*, 2005. [Online]. Available: http://www.hitl.washington.edu/artoolkit/documentation/