Real-time Augmented Reality with Occlusion Handling Based on RGBD Images

Xiaozhi Guo, Chen Wang, Yue Qi
State Key Laboratory of Virtual Reality Technology and Systems, Beihang University
Beijing 100191, China
Beihang University Qingdao Research Institute
Qingdao 266100, China
xiaozhi@buaa.edu.cn, vr_wangchen@buaa.edu.cn, qy@buaa.edu.cn

***Abstract*—Augmented Reality (AR) is one of the latest developments in human-computer interaction technology. It aims to generate illusions from seamless fusion of virtual objects and real world. Typical AR system requires two basic parts: three-dimensional registration and real-virtual fusion. Occlusion handling is crucial for visual realism. To optimize visual realism, we generated a real-time systematic architecture to operate occlusion handling. The architecture is based on RGBD images, and it consists of three parts: real-time camera tracking system, 3D reconstruction system and AR fusion system. Specifically, we used a two-pass scheme strategy to execute the AR system. The first pass tracks camera poses timely at video rate, which allows the reconstruction results be updated and visualized correspondingly during the scanning. The second pass takes place simultaneously to handle occlusion between virtual objects and real scene according to camera pose. Finally, the render results of virtual objects and the color images are fused to generate AR contents. Our results indicate that this method is stable and precise for occlusion handling, and can effectively improve realism in AR system.**

Keywords-augmented reality; occlusion handling; scene reconstruction; rgbd

I. INTRODUCTION

Augmented Reality (AR) has been a hot spot in research for a long time [1]. It combines virtual items produced by PC with real scene. AR system can produce more semantic implications than either virtual or real world by orchestrating them to be a whole. The primary challenge in generating convincing augmented reality is to project 3D models onto a user's view of the real world and create a spatial sustained illusion that the virtual and real scene coexist.

For a system to meet Azuma's definition of augmented reality system [2], it must fulfill two fundamental necessities: occlusion handling and 3D register. They play a very important role in convincing augmented reality system. Current AR system can be generally characterized into two groups according to its tracking method [3]. One is based on the sensor tracking technology that rotation and position of the camera are acquired by the data from accelerometer, GPS and compass. The cost of such system is generally high. The other one is based on the technology of computer vision. In this AR system, marker or scene features is tracked by means of computer vision technology. Such as QR-codes Based [4], Edge Snapping Based [5], 3D Line Segment Based [6], and convex polygon marker Based [7]. Since Davison [8] and other researchers proposed simultaneous localization and mapping (SLAM), it has been widely considered in the field of augmented reality [9][10][11].

One of the main problems of current augmented reality is the lack of reliable depth information. It simply overlays virtual 3D objects on real world imagery [12][13]. Such overlay is not fantastic when displaying data in three dimensions because the occlusion between real and computer-generated objects is not addressed. Hauck JDVS et al. utilized single depth image to handle Occlusion [14], but its performance is not good enough in detail due to the unstable depth value (Figure 1). As we can see,occlusion handling is a key issue of AR realism.

To handle the occlusion between virtual objects and reality scene, and improve the accuracy of Camera tracking, we adopt a method of computer vision for camera tracking, and a model-based approach for occlusion handling. In this paper, we proposed a robust marker-less AR architecture based on RGBD images.

II. SYSTEM OVERVIEW

The goal of our system is to generate a convincing model-based augmented reality system, which can handle occlusion correctly and track camera precisely. In order to achieve the goal, we adopt a specific method to rebuild the real scene while tracking the camera simultaneously. We take advantage of the kinect sensor, a conventional and low-cost RGBD camera.

In our system, we adopted a two-pass scheme. The first pass performs parallel camera tracking in real-time, and it allows the reconstruction results to be updated and visualized during the scanning process. The second pass handles the occlusion between virtual object and real scene model according to camera pose, and fuses the render result with the color image.

The data processing in our system consists of three major components. The flowchart of our system is shown in Fig.2.

- Camera tracking. Camera tracking needs to run at the start of the system. We utilize a fast bilateral filter to pre-process raw depth image. Then, we exploit camera parameters to transform the depth image into a point cloud in the 3D space, and

978-1-5386-2636-8/19/$31.00 ©2019 IEEE
DOI 10.1109/ICVRV.2017.00069

produce the current camera pose through a weighted Iterative Closest Point (ICP) algorithm.

- Model integration and extraction. We make use of the tracking state to fuse these depth images into a volumetric model, and we extract the isosurface from the current estimated camera pose by a ray tracing algorithm. For each voxel, we shoot a ray from the center along the negative gradient of the distance field, which is approximately the surface normal. If the ray hits the isosurface, we add the intersection point to the point cloud.
- Fusion and rendering. In this section, we handle the occlusion between virtual object and real scene model according to camera pose. Finally, we fuse the render result and the color image.

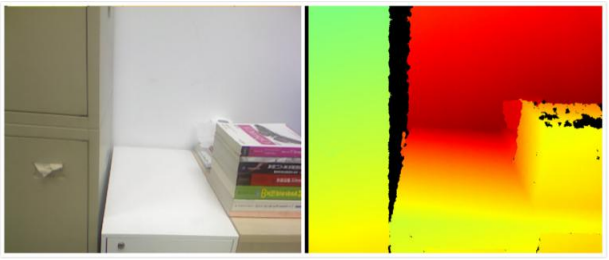

Figure 1. Raw images. Left: raw color image. Right: the raw depth image contains black spots due to the invalid depth values.

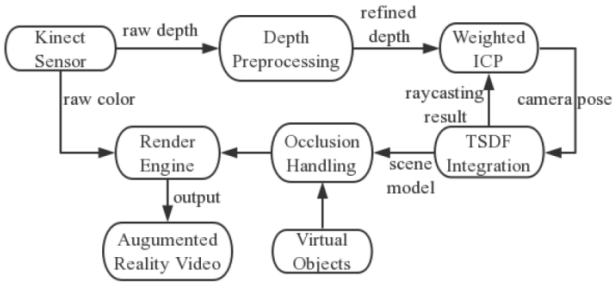

Figure 2. System flow chart

III. System Architecture

This section details the architecture and algorithms we used in our AR system.

A. Preliminaries

The kinect outputs a stream of RGBD frames marked as ($Color_i$, $Depth_i$) where $Color_i$ and $Depth_i$ denote the i_{ith} color image and depth image respectively. To reconstruct the 3D scene, we have to estimate the 6 degrees of freedom (6DOF) pose of camera. For each incoming frame, we need to estimate the transformation matrix T_i

$$T_i = \begin{bmatrix} R_i & t_i \\ 0^t & 1 \end{bmatrix} \tag{1}$$

Where R_i is the matrix of rotation, t_i is the vector of translation. An point position *worldPos(x, y, z)* in world coordinate is transformed into the camera's coordinate frame as follow

$$camPos = R * worldPos + t \tag{2}$$

According to calibrated intrinsic parameters of kinect sensor and the principle of pinhole camera model, we will suppose c_x, c_y are the image centers of the camera, and f_x, f_y are focal lengths. A 3D point *camPos = (x, y, z)* in the camera coordinate system is projected to the image plane under perspective projection

$$\pi(camPos) = (f_x \frac{X}{Z} + c_x, f_y \frac{Y}{Z} + c_y)^T \tag{3}$$

In the image coordinate system, a pixel coordinate *(u, v)* with depth data *Z = depth(u, v)* can be back projected conversely through

$$\varphi(u, v, z) = (\frac{u - c_x}{f_x} Z, \frac{v - c_y}{f_y} Z, Z)^T \tag{4}$$

B. Camera Tracking

We adopt a method extending the voxel hashing based volumetric fusion [15][16] approach to achieve robust real-time camera tracking. Our reconstruction results can be visualized instantly to guide the scanning.

In order to perform real-time 6DOF camera tracking, we utilize the framework of volumetric fusion with voxel hashing, which can handle large-scale reconstruction of fine-grained detail. The reconstructed scene is represented by Truncated Signed Distance Field (TSDF) on the voxel grid with voxel hashing data structure. In each voxel, we store the signed distance to the closest surface point, the color, and the weight for a weighted running average in real-time fusion into the existing structure.

The voxel hashing structure allows real-time access and update using GPU. While the camera is moving, data can easily flow into and out of the hash table with the streaming algorithm. The volumetric fusion framework stores the TSDF value $\psi(p)$ and an associated weighting factor $w(p)$ in a voxel whose center locates at point p. And it has two important components, namely, frame-to-model registration and model integration.

For frame-to-model registration, we utilize the weighted TSDF tracking method, which performs better than the original projection ICP method [15]. It registers each entered depth image to the reconstructed TSDF model, rather than the rendering depth image generated from the ray casting. For each pixel *(u, v)*, we suppose the corresponding inhomogeneous 3D position(a 3-vector) is $camPos_{(u, v)}$. To calculate the camera transformation *{R, t}*, we need to minimize the objective function of the form

$$E_{R,t} = \sum_{u,v} w_{u,v} \psi(R * camPos_{u,v} + t)^2 \tag{5}$$

To solve equation (5), we adopt a Gaussian Newton nonlinear minimization method. At iteration $k + 1$ $(k \geq 0)$, we start to linearize M^{k+1} around M^k by a linear function.

$$M^{k+1} \approx \begin{bmatrix} 1 & -\gamma & \beta & a \\ \gamma & 1 & -\partial & b \\ -\beta & \partial & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} M^k \quad (6)$$

We set up a 6-dimensional vector $\xi = (\alpha, \beta, \gamma, a, b, c)$, which is a twist coordinate, representing the incremental transformation relative to M^k at iteration k. (a, b, c) and (α, β, γ) are the translation vector and the angular velocity respectively.

By denoting $Y = M^k * camPos$, ξ^{k+1} is computed from equation

$$\sum \omega \varphi(\tilde{Y}) \nabla(\tilde{Y}) = -\sum \omega \nabla \varphi(\tilde{Y}) \nabla(\tilde{Y})^T \xi^{k+1} \quad (7)$$

Where $\tilde{Y}$ represents Y's inhomogeneous version, $\nabla\varphi(\tilde{Y})$ is the derivative of the TSDF function evaluated at point Y. By solving this $6*6$ linear equation. ξ^{k+1} can be computed efficiently. The incremental transformation can be gained from ξ^{k+1}. We update T^{k+1} and iterate this process until get convergence. We terminate when either the change of ξ between iterations is small enough or a certain number of iteration is reached. In our implementation, the maximum number of allowed iterations is set to 18. To get a real-time performance, we parallelize each pixel calculation using GPU.

For model integration, we utilize the non-uniform weighting strategy to integrate the i_{th} depth image with the previously reconstructed model (ψ_{i-1}, W_{i-1}). We suppose that p is the center of a voxel, which is close to the underlying surface of the incoming depth image. We can compute its distance to the i_{th} camera through projection, denoted by $C_i(p)$, and the depth value on the projected pixel $depth(\pi(p))$. The distance of p to the implied surface is $f_i = C_i(p) - depth_i(\pi(p))$. Then update the voxel with center p

$$\varphi_i(p) = \frac{W_{i-1}(p)\varphi_{i-1}(p) + \lambda_i(p) f_i(p)}{W_{i-1}(p) + \lambda_i(p)} \quad (8)$$

$$W_i(p) = \lambda_i(p) + W_{i-1}(p) \quad (9)$$

The parameter $\lambda_i(p)$ makes the model integration favor points from the near range, which usually has better accuracy than points from a far distance. In our experiments, we utilize a linear weighting function

$$\lambda_i(p) = 1 - (depth_i(\pi(p)) - d_{\min}) / (d_{\max} - d_{\min}) \quad (10)$$

$$d_{\max} = 3.5m, d_{\min} = 0.5m \quad (11)$$

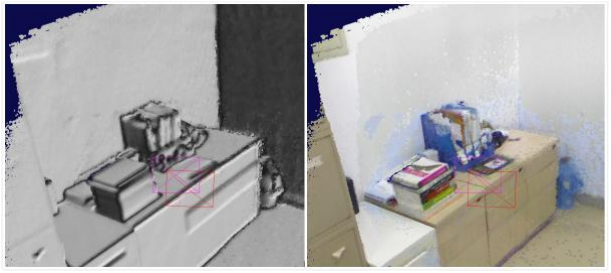

Figure 3. The corner of the room, no loop closure. Left: triangular mesh without texture. Right: triangular mesh with texture.

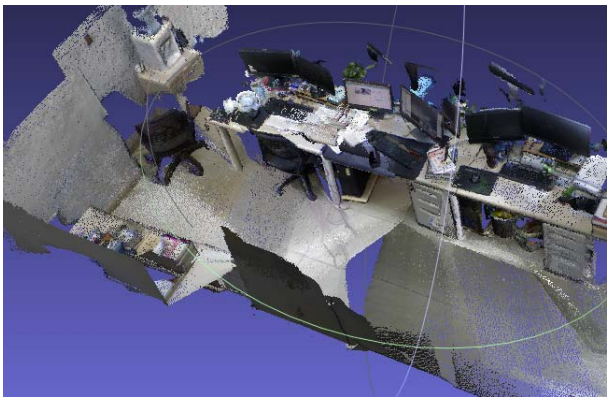

Figure 4. The results of reconstruction algorithm with texture

C. Fusion and Rendering

The AR illusion can be implemented using the following contents: a background video of the real environment obtained by camera, a virtual camera positioned by the camera tracking and spatial position relationship between virtual objects and real surroundings.

Before the start of the rendering and fusion, the primary task is to obtain the initial position of virtual object. We achieve this by a plane detecting algorithm. Specifically, we adopt the ray casting depth map as input, and then utilize least squares method to fit the plane, and then calculate the plane normal and center coordinates as the virtual object's initial position.

When extracting the model surface from the TSDF, a surface point is created at the zero crossing point along a casting ray. Once the zero crossing point is calculated, the associated color is determined by the voxel where the point resides.

In our system, the specific work of scene fusion is done by Open Scene Graph (OSG). Making use of the kinect sensor, we can get RGB view of the current frame drawn as the background, virtual 3D model rendered on top. As the reconstructed scene is built from multiple views, it can eliminate the error caused by single depth image when handling occlusion.

Compared to the other typical augmented reality applications, one significant improvement in our proposed method is the ability to perform realistic occlusion of AR contents. With the kinect sensor's depth data, we are able to

determine on a per-pixel basis whose areas of virtual objects should not be drawn when they lie behind real-world objects.

In order to achieve our purpose, we wrote a custom fragment render shader to discard fragments with a greater distance from the kinect camera than the real objects at the same pixel. And then, the 32-bit floating point depth values are passed to our shader in a texture and packed into four 8-bit colour channels (RGBA). This allows the full depth resolution to be recovered in the fragment shader, which typical only permits a 8-bit colour resolution.

IV. RESULT

In this section, we analyzed the results and performance of our approach. Our experiment was implemented using C++ programming and compiled with visual studio 2013. All the tests were performed in an Intel(R) Core(TM) CPU i7-4790k@ 4.00GHz computer with 16 GB of RAM. The graphic card was from NVIDIA GTX970 series.

To test the robustness of camera tracking, we placed a textbook on the floor, and initialized the position of virtual dinosaur to be there. We looked at the dinosaurs in different perspectives and see if the location of the dinosaurs has changed. We conducted a 360 degree view of the dinosaurs, and found it is still near the textbook. Our camera tracking is robust and accurate. The overall performance of our system is about 25 fps. The concrete results are shown in Fig.6.

Figure.7 shows dense reconstruction of real scene and the spatial relationship between virtual object and the scene before the fictitious fusion, which is the key to dealing with occlusion problem.

We also compared our experimental results with others [14]. As described in Figure.8. The experimental results show that our system can deal with the problem of occlusion stably and effectively, and can also effectively improve realism in AR system. Besides, our system can guarantee real-time performance under GPU acceleration.

Figure 5. Virtual dinosaur in a real scene with occluded by table.

V. CONCLUSION

In this paper, we presented a real-time augmented reality system with occlusion handling based on RGBD Images, a system for online dense reconstruction of indoor scene using data from a hand-held RGBD camera. To improve the accuracy and robustness of camera tracking, we combined dense reconstruction with SLAM, and designed a real-time rendering system to produce AR content. We described all the key components and implementation details here, including real-time frame-to-model tracking, surface model integration and fusion. As is shown in our experimental results (Figure.5), our system maintains a realistic illumination of AR system.

Our future work will focus on calculating shadows for the virtual models and interactive AR that full of challenge and great value.

Figure 6. Virtual dinosaur in our laboratory observed in different perspectives. The blue book right blew the dinosaur's foot is used to detect whether the dinosaur has moved or not.

Figure 7. The picture vividly reveals our model-based method of occlusion handling. The white background is the 3D model of real scene, which is updated at video rate.

Figure 8. Contrasting with other methods. Left: the effect of our method. Right: the effect of raw depth data [14].

ACKNOWLEDGMENT

The work is supported by National Key R&D Program of China (No. 2017YFB1002602), the National Natural Science Foundation of China (No. 61572054) and Applied Basic Research Program of Qingdao (No. 16-10-1-3-xx).

REFERENCES

[1] Azuma R T. A survey of augmented reality[M]. MIT Press, 1997.

[2] Azuma R. Tracking requirements for augmented reality[J]. Communications of the Acm, 1993, 36(7):50-51.337.

[3] Liang Z M, Yuan Y, Liang J Y, "A Survey on Augmented Reality". Journal of Image & Graphics, Vol.9, No. 7, pp.767-773, July 2004.

[4] Gherghina A, Olteanu A C, Tapus N. A marker-based augmented reality system for mobile devices[C] Roedunet International Conference. IEEE, 2013:1-6.12.

[5] Du C, Chen Y L, Ye M, et al. Edge Snapping-Based Depth Enhancement for Dynamic Occlusion Handling in Augmented Reality[C]// IEEE International Symposium on Mixed and Augmented Reality. IEEE, 2016:54-62.0.

[6] Augmented Reality Framework Using On-Site 3D Line-Segment-based Model Generation[J]. Journal of Imaging Science & Technology, 2016.3.

[7] Gao Y F, Wang H Y, Bian X N. Marker tracking for video-based augmented reality[C] International Conference on Machine Learning and Cybernetics. IEEE, 2017:928-932.

[8] Davison AJ,Reid ID,Molton ND "MonoSLAM Real-time single camera SLAM[J]" Pattern Analysis and Machine Intelligence□IEEE Transactions on Pattern Analysis and Machine Intelligence Vol.29,No. 6,pp.1052—1067,June,2007.

[9] Liu H, Zhang G, Bao H. Robust Keyframe-based Monocular SLAM for Augmented Reality[C]// IEEE International Symposium on Mixed and Augmented Reality IEEE, 2016:1-10.

Article in a conference proceedings:

[10] Whelan T, Kaess M, Johannsson H, et al. Real-time large-scale dense RGB-D SLAM with volumetric fusion[J]. International Journal of Robotics Research, 2015, 34(4-5):598-626.40.

[11] Li S, Handa A, Zhang Y, et al. HDRFusion: HDR SLAM Using a Low-Cost Auto-Exposure RGB-D Sensor[J]. 2016:314-322.

[12] Vera L, Gimeno J, Coma I, et al. Augmented Mirror: Interactive Augmented Reality System Based on Kinect[M] Human-Computer Interaction – INTERACT 2011. Springer Berlin Heidelberg, 2011:483-486.22.

[13] Sato H, Cohen M. Using motion capture for real-time augmented reality scenes[C] International Conference on Humans and Computers. 2010:58-62.

[14] Hauck J V D S, Mendonça M R F, Silva R L D S D. Occlusion of Virtual Objects for Augmented Reality Systems using Kinect[C] Workshop De Realidade Virtual De Aumentada. 2014.0.

[15] M Nießner, M. Zollh¨ofer, S. Izadi, and M. Stamminger.Real-time 3d reconstruction at scale using voxel hashing.ACM Transactions on Graphics, 32(6):169, 2013.

[16] Wang H, Wang J, Wang L. Online Reconstruction of Indoor Scenes from RGB-D Streams[C] IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2016:3271-3279.1.