Visualization of Dynamic Hidden Areas by Real-Time 3D Structure Acquistion Using RGB-D Camera

Siim Meerits and Hideo Saito
Department of Information and Computer Science
Keio University
Yokohama, Japan

***Abstract*—**We present a system that can accurately visualize changing (dynamic) areas captured by an RGB-D camera at an alternative viewpoint. Our system does not make assumptions about the scene 3D structure and works in real time.

***Keywords**-free-viewpoint video; RGB-D camera*

I. Introduction

In recent years, mediated reality research has become increasingly active within the field of computer vision. One of the subfields of mediated reality is diminished reality (DR), which is concerned with removing i.e. diminishing objects from scenes. Various possible applications exist for such systems in TV broadcasting [1], smartphone applications [2] and robotics [3].

To remove an object from image, one must overlay it with its background image. Hence, the core issue with DR systems is how to visualize the backgrounds of objects being diminished. Most systems in DR research rely on color cameras to gather information about the scene where objects are to be made invisible.

RGB-D cameras have become increasingly popular among computer vision researchers in recent years. Some industry projects, such as Microsoft Kinect are being used in consumer living rooms. New projects, such as Google Tango, are bringing the technology to smartphones and tablets.

An RGB-D camera enables us to observe the 3D structure of a scene in real-time. With fast and accurate 3D reconstruction and rendering it is possible to visualize the scene at a different camera viewpoint.

We present a general method of visualizing hidden areas using an RGB-D camera. This system could be used in specific diminished reality applications to hide objects. Conceptual overview of the proposed system for a sample scene from the system setup to the final results can be seen in Fig. 1.

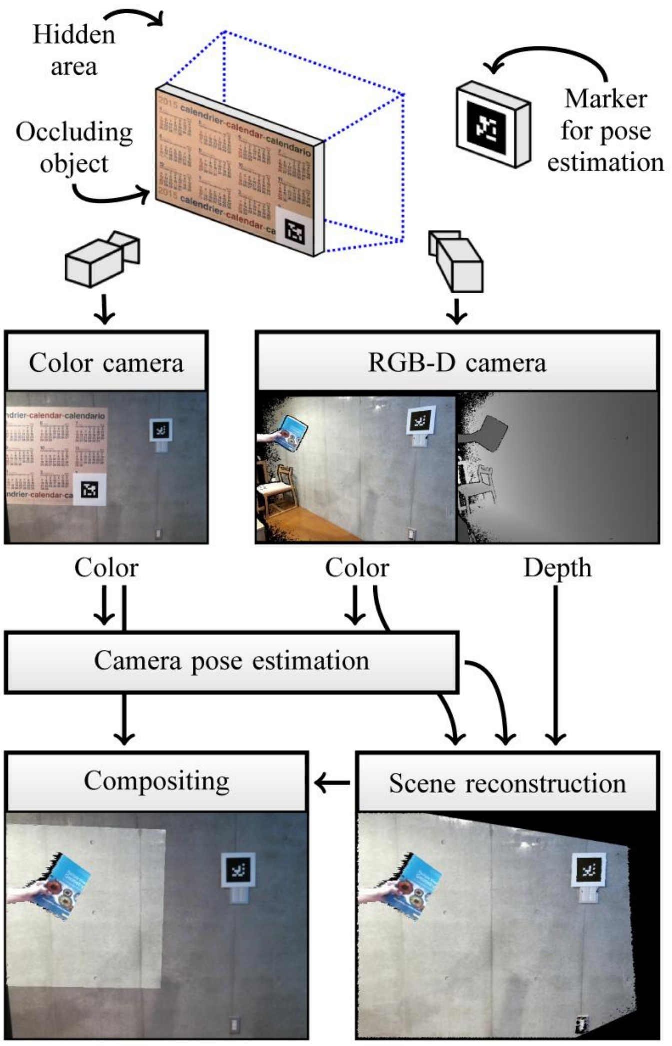


Figure 1. Proposed system overview. The diagram shows how an occluding object, in this case a beige calendar, is hidden from a scene. Final system result without the calendar is shown in bottom left corner.

II. Related Works

Numerous different systems in the field of diminished reality have attempted to achieve goals similar to ours. The approaches, however, have varied wildly from work to work. The most closely related methods use multiple cameras at different viewpoints to hide objects in real time.

Most multi-camera methods require the scene background behind objects to be made invisible to be planar [1-2] or piecewise planar [4]. This is a rather strong requirement and excludes any scenes where objects move in the background. Nevertheless, some interesting applications have been presented. For example [2] demonstrated method to make human writing on a whiteboard to disappear. Because the whiteboard was planar and remained same in the structure, the system worked. Hence the system was capable of handling color changes, but not structure changes in the scene.

Couple systems not making planar background assumption have also been published. [5] made use of multiple color cameras to remove colorful objects in limited size. Their approach used plane sweep algorithm to get a coarse reconstruction of a scene. With a reconstructed 3D model of a scene it is easy to define a region of space, which will not be included in rendering the scene model and effectively hides all objects in the region. Unfortunately, the utilized reconstruction method is both computationally expensive and has quality issues. This is not surprising as the more general scene reconstruction using multi-view stereo is also computationally very expensive.

One way of solving the scene reconstruction issues is to use depth-sensing RGB-D cameras to help in the reconstruction process. One such method using three RGB-D cameras to hide a robotic hand of a rescue robot was demonstrated in [3]. This work does not attempt to completely hide objects, but leaves them half-visible. Additionally, this work requires multiple RGB-D cameras to be placed in a rigid configuration.

III. Proposed Method

The system diagram has been given in Fig. 1. As previously mentioned, our system uses two cameras. First one is an RGB camera and the second is an RGB-D camera. The overall objective is to use the RGB-D camera to see some hidden areas not visible in color camera.

After estimating the pose of cameras we reconstruct the scene seen from the RGB-D at the color camera viewpoint. The reconstruction is then used together with color camera image in compositing step to get the final system result.

A. *Camera Pose Estimation and Hidden Area Detection*

We need to determine pose of the cameras in relation to each other. In our demonstrations we utilize AR markers for this task, but generally it does not matter what method is used as long as we can retrieve rotation and translation information between cameras. With markers, both cameras estimate the pose to the marker object. We combine the coordinate space transformation from RGB-D camera to the AR marker with the inverse transformation from the marker to color camera to get the needed RGB-D camera – color camera pose relation.

Second thing that needs to be done before scene reconstruction is to figure out the outlines of (occluding) objects that should be hidden in the scene. The method of figuring out those areas depends on particular application. For example we can track known objects with attached markers or find objects with known shape and color.

The outlines should be specified in all camera images. In the case of the RGB-D camera, the object outline defines an image region that should be excluded from scene reconstruction. Technically we just set the RGB-D camera depth map pixels to invalid values, which are correctly ignored in reconstruction phase. Object outline in color camera frame selects the image region that should be overlaid with scene background. The exact process is later described in the compositing section.

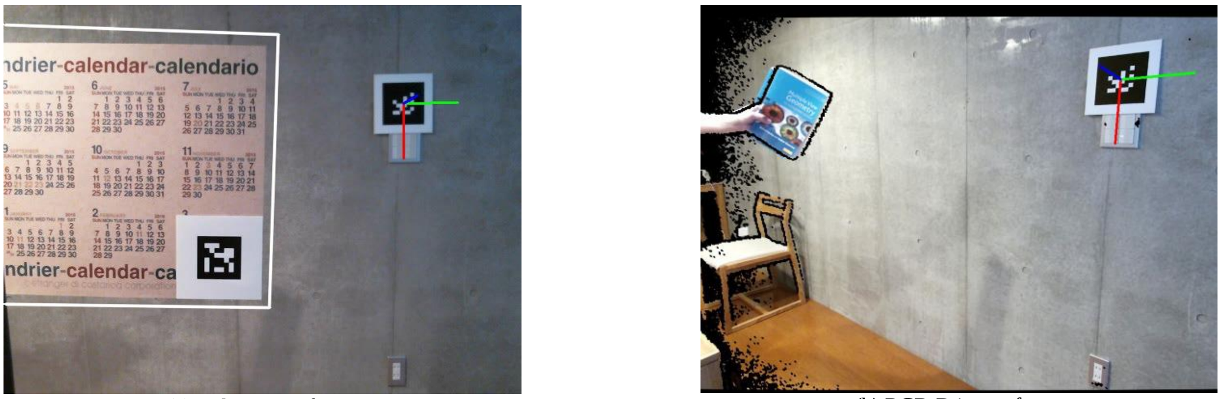
Illustration of the pose estimation and hidden area detection can be seen in Fig. 2.

B. *Scene Reconstruction*

The scene visible from an RGB-D camera must be reconstructed, rotated and translated to target camera viewpoint, and finally rendered. Since we wanted to develop a real-time system, we had to choose our approach accordingly.

The core concept for scene reconstruction is adapted from step discontinuity constrained triangulation (SDCT) method [6]. SDCT can directly be applied to depth maps to get a triangle mesh of scene background without using any point cloud data structures. Additionally, the method is very fast and well parallelizable. SDCT works by forming triangles between neighboring depth pixels. This process is illustrated in Fig. 3. There are some issues that need to be considered with this approach.

One issue is that separate objects in the scene should not be



(a) Color image frame (b) RGB-D image frame

Figure 2. Example of camera pose estimation and hidden area detection. The white box in (a) shows occluding object contour that should be hidden in the final system result. Both images (a) and (b) also show coordinate axes drawn on top of the pose estimation marker. The red, green and blue lines correspond to X, Y and Z axes of a marker centric coordinate system.

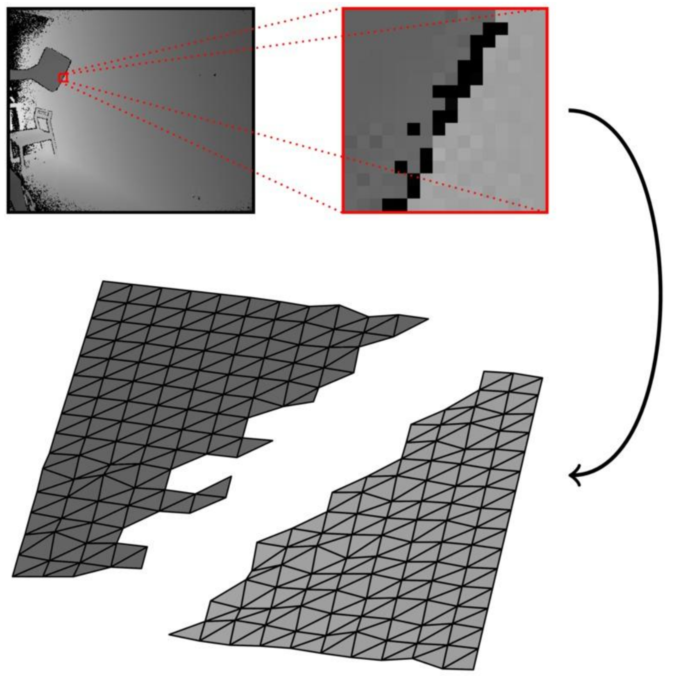


Figure 3. Illustration of triangular mesh generation in scene reconstruction. Depth map from the RGB-D camera is shown in the top-left corner. Small 16x16 pixel region has been cut out of the depth map and is shown in the top-right red box. Neighboring depth pixels are turned into triangles as shown in the bottom image.

joined together by triangles. The solution is to exclude any triangles that connect depth pixels with depth value differences larger than some constant value. Empirical experiments showed that 5 cm is a good choice for such segmentation in indoor scenes.

Second issue is to deal with invalid depth pixels from the RGB-D camera. This can be handled in the similar manner as the previous case by removing any triangles connected to invalid pixels.

In some cases our depth image might include regions of scenes that should not be rendered at the target camera viewpoint. For example the source camera partially sees some object and that region ends up being the object back side when looked from target camera viewpoint. To avoid rendering such scene parts, we carry out triangle orientation checking. At the start of reconstruction, all triangle vertices are oriented in clockwise manner. After scene rotation and translation, the invalid triangles end up having vertices in counter-clockwise order which makes them easy to detect and remove.

C. *GPU Acceleration*

In our system, both scene reconstruction and rendering process has been implemented in OpenGL shaders to speed up calculation on GPU. Only standard features from OpenGL version 3 core profile were used. In the following we describe the vertex, geometry and fragment shader steps in detail.

As a first step in processing every frame, the depth map from source camera is uploaded to the GPU memory. Every depth pixel is to be considered as a vertex to be processed with

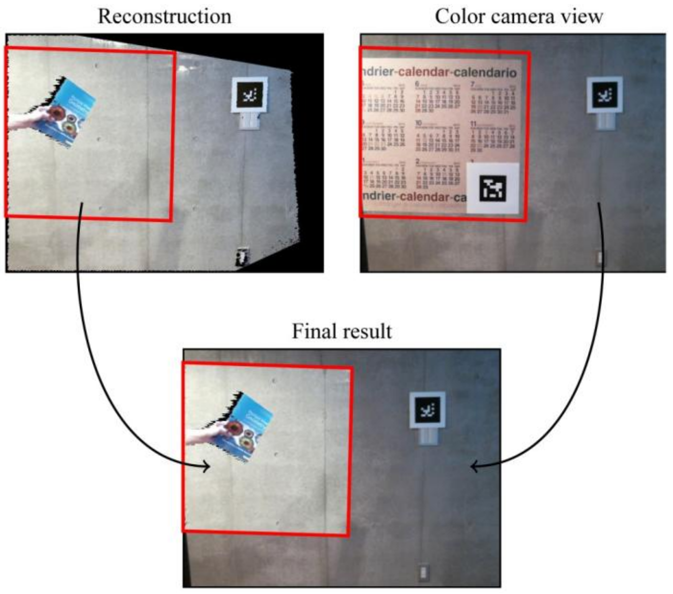


Figure 4. Illustration of the compositing process. Initial image for the result is copied from color camera view. The region of the occluding object is highlighted in red boxes. The red box content from reconstruction is copied to the final result image.

vertex shader. To achieve better computational efficiency, we do scene reconstruction and its transformation to target camera viewpoint in reverse order i.e. we start from rotation and translation transformation and then move to reconstruction and rendering. Hence the first step in vertex shader is to use the previously estimated pose between cameras to calculate the position of every vertex in target camera viewpoint. Additionally, we calculate the corresponding source camera color frame coordinates for every vertex and store them for later processing.

The geometry shader is responsible for forming triangles between vertices. The shader takes in a set of vertex indices corresponding to any possible triangles and outputs all triangles deemed valid. The validation process follows rules outlined in the previous scene reconstruction section.

Finally, the generated triangles are rendered using fragment shader. This process is very straightforward as we only need to load texture values in triangles using texture coordinates calculated in the vertex shader step. Texture coordinate interpolation between triangle vertices and depth testing is taken care by built-in OpenGL functionality.

D. *Compositing*

The last task left to us is to compose the final result from color camera image and the scene reconstruction image. Both input images should align exactly as long as camera poses were correctly estimated. Hence the compositing process just selects pixels from either image depending on the occluding object contour as previously discussed. This process is shown in Fig. 4.

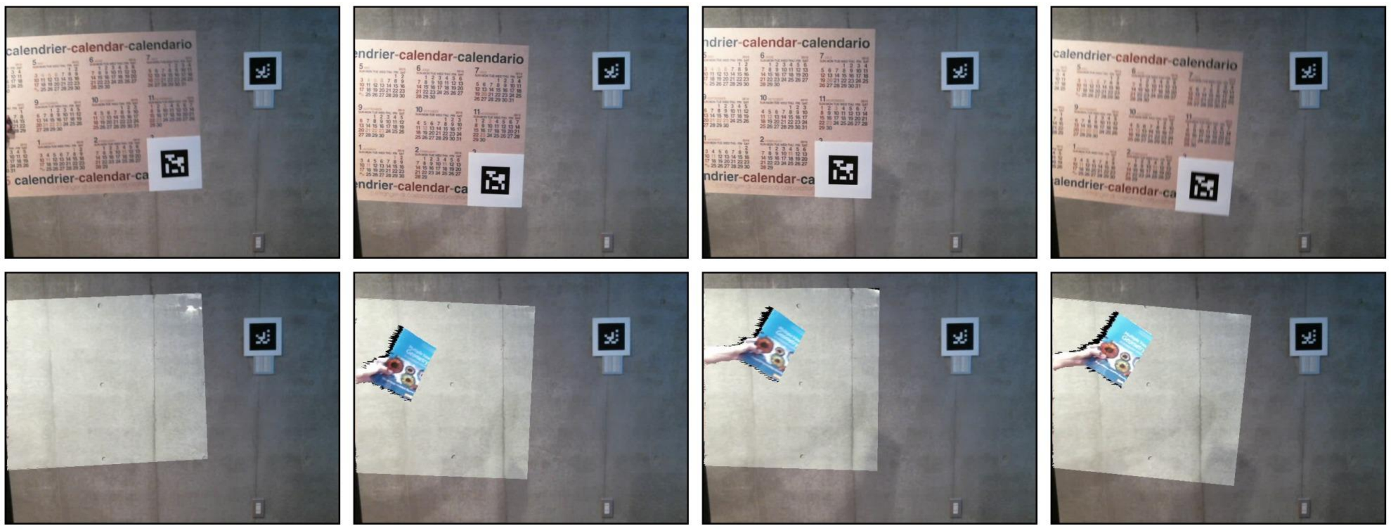


Figure 5. Video outtakes from the calendar scene experiment. Top row shows frames from color camera and the bottom row shows corresponding frames after processing by the system.

IV. Experimental Results

We tested our system in a scene called "calendar". In this scene, the objective was to hide a calendar object from user view. Since estimating the calendar region on the screen just from known color can be unreliable, we fixed an extra AR marker on top of this calendar. The dimensions of the calendar were known so the occluded object region could be calculated just by knowing the pose of the fixed marker.

All illustrations in this paper show frames taken from this calendar scene. Additional result images taken from recorded experiment video can be seen in Fig. 5.

Tests showed similar performance characteristics across different scenes. In the case of calendar scene, the pose estimation of AR markers took 8.2ms on average. The scene reconstruction took 12.8ms and compositing had negligible computational cost somewhere in the millisecond range. The whole process was fast enough to run in real time and was limited by camera frame rates to 30fps.

V. Conclusion

This paper presented a novel method of visualizing hidden areas behind objects. Our method requires minimal setup and can work even with a single color and single RGB-D camera. Importantly non-planar scene backgrounds are supported and movement in hidden area is immediately visible in system output.

References

[1] T. Hashimoto, Y. Uematsu, H. Saito, "Generation of see-through baseball movie from multi-camera views," International Workshop on Multimedia Signal Processing (MMSP2010), pp.432-437, 2010.

[2] A. Enomoto, H. Saito, "Diminished Reality using Multiple Handheld Cameras," ACCV'07 Workshop on Multidimensional and Multiview Image Processing, pp.130-135, 2007.

[3] K. Sugimoto, H. Fujii, A. Yamashita, and H. Asama, "Half-diminished reality image using three rgb-d sensors for remote control robots," International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp. 1–6, 2014.

[4] S. Zokai, J. Esteve, Y. Genc and N. Navab, "Multiview Paraperspective Projection Model for Diminished Reality," International Symposium on Mixed and Augmented Reality (ISMAR 2003), pp. 217 - 226, 2003.

[5] S. Jarusirisawad, T. Hosokawa, and H. Saito, "Diminished reality using plane-sweep algorithm with weakly-calibrated cameras," Progress in Informatics, vol. 7, pp. 11–20, 2010.

[6] A. Hilton, A.J. Stoddart, J. Illingworth and T. Windeatt, "Reliable surface reconstruction from multiple range images," Computer Vision (ECCV'96), Springer Berlin Heidelberg, pp. 117-126, 1996.