Real-Time Diminished Reality for Dynamic Scenes

Siim Meerits[*]
Keio University

Hideo Saito[†]
Keio University

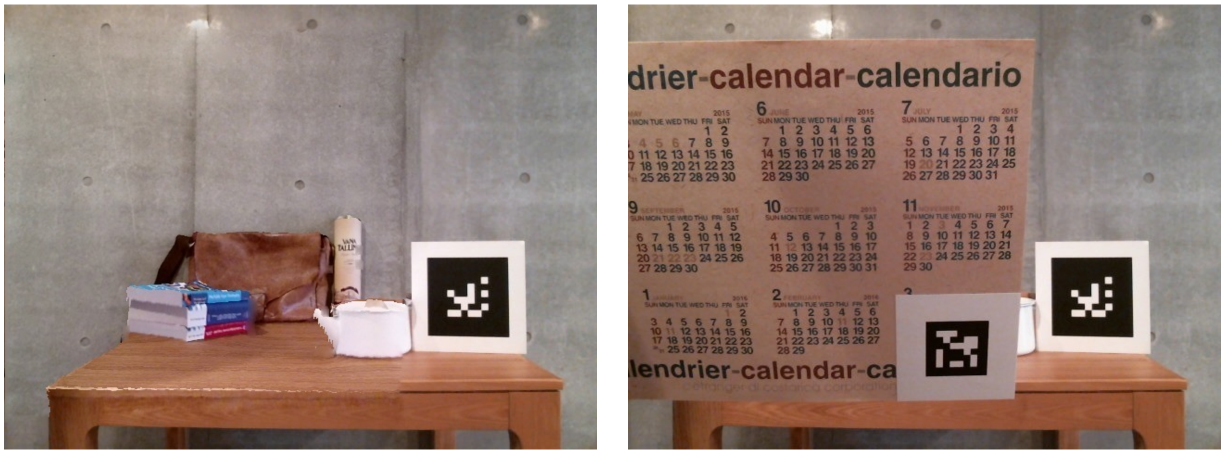


Figure 1: Diminishing a calendar from user view. Right side image shows original input from a color camera. A calendar on that image has been diminished using our system and the result can be seen on the left side image.

Abstract

Diminished reality (DR) research aims to develop methods to visually hide objects from real scenes. One or more cameras capture a scene where some objects, called the occluding objects, are 'diminished' by an DR system. To do this in real time we need to use multiple cameras. We introduce a novel multi-camera DR system utilizing an RGB-D camera to hide arbitrary trackable objects from a scene. In our case, the scene background does not have to be planar for the system to work and we can handle scene changes in real time. Additonally, we introduce color correction technique to merge image content originating from different cameras and also demonstrate method to repair images with partially missing content.

Keywords: Diminished reality, free-viewpoint video, RGB-D camera.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities;

1 Introduction

Diminished reality (DR) is a research field concerned with providing users an indirect view of the world where some objects have been made invisible. In other words, some device records user surroundings, then removes some objects from the view using image manipulation and finally presents the result to the user. Such device could be a wearable computer or hand–held device such as smartphone. The diminished reality concept is demonstrated in Figure 1 where we show one of the results of our system.

There are many approaches to diminished reality. Some methods, such as image inpainting, make guesses of possible image content based on patterns found on images. Another set of methods, based on videos, utilize series of images from camera to piece together backgrounds of objects to be removed from user view. We are interested in approaches which can achieve realistic diminished reality in real time. The inpainting methods cannot always give truthful results as there are limits to guessing image content and video based methods cannot always work in real time. To achieve realistic real–time diminished reality, we need to place multiple cameras into the scene to see hidden areas behind occluding objects.

Previously, most systems have attempted the use of multiple color cameras to develop understanding of the scene structure. This work introduces different approach using an RGB-D camera to see the hidden areas of the scene. In recent years, Kinect devices developed by Microsoft brought RGB-D cameras to consumer living rooms and new projects, such as Google Tango, are trying to introduce the RGB-D camera technology to smartphones. Therefore, we expect that RGB-D cameras will become more common in future.

In this paper we show that it is possible to achieve realistic diminished reality with only two cameras – a color camera and an RGB-D camera. While two camera DR systems have been developed before, our system importantly does not require the background of scenes to be planar. The background structure can not only be of any shape but it can also change and move freely.

We also address the issue of color calibration and relighting. By using different types of cameras at different viewpoints in the scene, we invariably have to deal with lighting differences of images obtained from cameras. Additonally, we show how to apply image inpainting techniques within our framework to fill in content of areas where scene reconstruction has failed. Finally, we demonstrate our system in two sample scenes where we diminish different objects.

The rest of the paper is organized as follows. After introducing related works, we explain our proposed method which includes sys-

[*]e-mail: meerits@hvrl.ics.keio.ac.jp
[†]e-mail: saito@hvrl.ics.keio.ac.jp

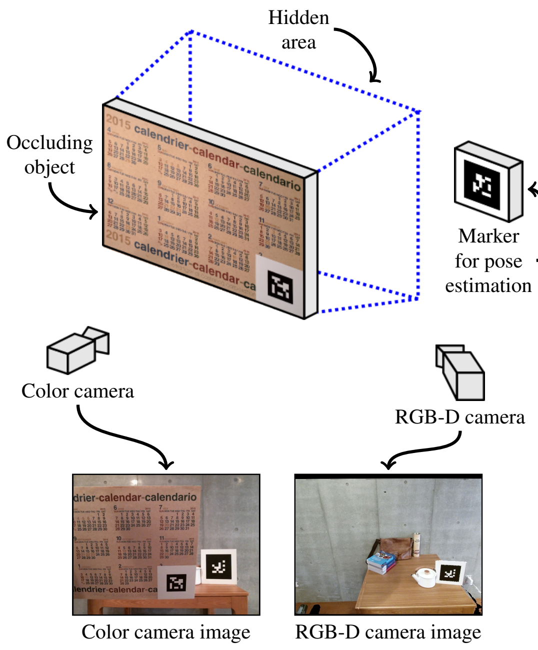


Figure 2: System setup diagram. The color camera sees some occluding object from the front. In this diagram, a calendar object was chosen as an example. The task of the RGB-D camera is to observe the hidden area behind the calendar highlighted in a blue box. Images from the actual scene are shown in the bottom of the diagram.

tem overview, camera pose estimation and object detection, scene reconstruction and result compositing. Before concluding, we show experimental results with system setup, results in two scenes and performance metrics.

2 Related works

The diminished reality term refers to systems capable of hiding objects from scenes. This definition is very broad and has resulted in articles with varying approaches to diminished reality to be published. Review of some earlier DR work can be found in paper by Mori et al. [8]. In the following subsections we summarize previous relevant research. All of the papers presented here can work in real time and use multiple cameras.

One important aspect of all DR works is the amount of scene complexity a DR system can handle. Hence, we start from works that can only handle scenes with planar backgrounds, and then move on to works which can handle more complex backgrounds.

Zokai et al. [12] introduced the main concepts of diminishing objects with planar and piecewise planar backgrounds by using a paraperspective background model.

Enomoto et al. [1] also demonstrate a system for diminishing objects in front of planar background. While previous work by Zokai et al. [12] considered the region of the occluding object to be known, Enomoto et al. deduce the occluding area by comparing pixels of different camera images. As a weakness, this sort of pixel-by-pixel comparison does not work if the occluding object has no color variation across its surface or if the number of cameras capturing scene is low.

Jarusirisawad et al. [6] demonstrate a DR system that can handle moving cameras. In this work the background model has to be created before system is put into use.

Another Jarusirisawad et al. [5] paper demonstrates a system capable of removing occluding objects from scene without requiring the background of the scene to be planar. The method in question

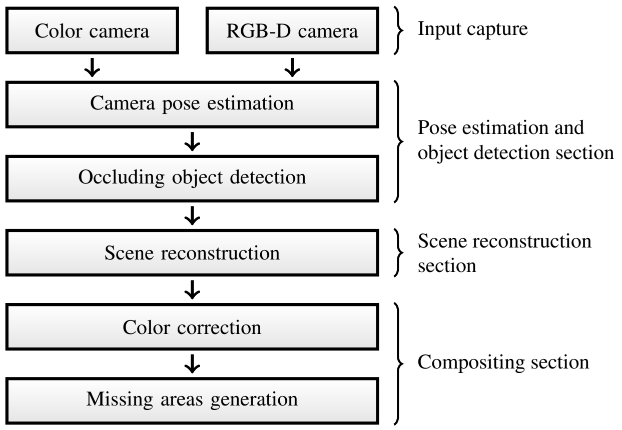


Figure 3: Overview of major system processes and their ordering by execution.

utilizes a large number of freely placed color cameras to gather information about the scene. The core method for object removal is plane–sweep algorithm (PSA). The PSA could be considered as a very simple form of multi–view stereo (MVS). As with most MVS algorithms, the computation is very expensive and the real-time requirements resulted in reconstruction quality tradeoffs – only smallish objects with strong contrast difference with background were effectively diminished.

Honda et al. [4] developed a system capable of diminishing objects in scenes with complex backgrounds. In their approach, the background is scanned before system use and stored as a 3D model. During system operation, the background of occluding objects is rendered using this model.

A recent work from Sugimoto et al. [10] realizes a DR system using three rigidly fixed RGB-D cameras to see behind a robotic arm. The approach is interesting for the use of multiple depth-sensing cameras. However, while the three dimensional structure of the scene is considered, the background model is not reconstructed. For every pixel, an evaluation function chooses best camera for showing background. As the robotic arm is only half–diminished, no color correction methods are used.

3 Proposed method

3.1 System overview

Reconstruction of the scene can be a very difficult task when only color cameras are available. The RGB-D cameras could be utilized to solve those problems. By obtaining a stream of depth maps, we can gather information about the scene structure in much easier manner. We construct our system on the idea that RGB-D cameras can help realize real–time diminished reality within dynamic scenes.

The setup of our system contains two cameras – a color camera and an RGB-D camera. The occluding objects seen by color camera are diminished using imagery seen from RGB-D camera. Essentially, the purpose of the RGB-D camera is to see hidden areas in the scene. This conceptual setup is illustrated in Figure 2.

Before explaining the way of utilizing the camera images, we discuss temporal issues of camera streams. It is tempting to utilize data from multiple frames of images, as in video based DR systems, to enhance the scene quality or fill in areas that cannot be seen from any camera at some moment. However, we argue that this approach will bend the reality of the scene. Some objects might have moved when cameras could not observe them. By utilizing data from previous frames, those objects remain visible in their prior locations, making the scene unrealistic. Returning to our concept of dimin-

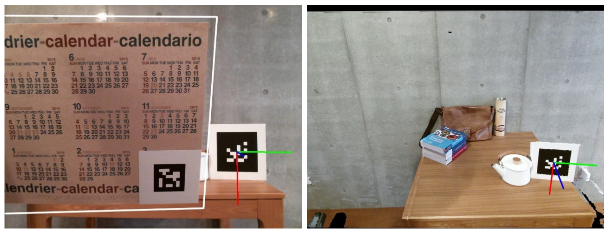

(a) Color camera view (b) RGB-D camera view

Figure 4: Camera pose estimation and occluding object detection. Camera poses are estimated using the marker attached to the concrete wall, visible in both images. The red, green and blue lines drawn on top of the marker show three Cartesian coordinate axes in marker coordinate system. Lastly, the white lines in (a) show the bounding box of a detected occluding object.

ished reality explained in introduction, the present authors argue that we want to see behind objects for some practical reasons and strongly prefer to see true representation of the world. Hence, we decide to only work with data available from cameras at present moment and not to generate any historical view of the scene.

All important system parts are given in Figure 3. This process is run once for every frame captured from both color and RGB-D camera.

3.2 Pose estimation and object detection

Before scene reconstruction can take place we need to obtain some preliminary information about the scene. Namely, we need to figure out the location of the cameras relative to each other and also find the occluding objects to be diminished from the scene. This section discusses how both of the operations are carried out in our system. Related illustration can be seen in Figure 4.

3.2.1 Camera pose estimation

In our system, we only need to estimate the poses of cameras relative to each other. Easiest way to do this is in real–time is to have a known object within the scene that can be tracked from all cameras simultaneously. For this reason, we use AR markers for camera pose estimation. Additionally, the advantage of markers is their good performance and accuracy. Our marker detection is implemented using system developed by Garrido-Jurado et al. [2].

It is important to note that the scene reconstruction process is carried out for every frame independent of other frames. This means that there is no need to track the scene pose. In turn, this implies that if we use a known object for pose estimation then this object can move freely within the scene as long as it is visible from all cameras.

3.2.2 Occluding objects detection

To remove objects from scenes we need to know where they are located. The occluding objects must be detected in both color and RGB-D camera images. In both cases the end result of this detection should be a binary mask over camera images indicating wheter particular pixels are part of an occluding object or not. We call this the occlusion mask. The actual detection method to derive it is highly dependent on what objects we are trying to diminish. In the following we demonstrate two detection methods – marker based detector and color blob based detector.

The marker based detector works, as the name suggests, by detecting AR markers that are attached on top of occluding objects. An example of this detection is seen in Figure 4. We assume that the 3D shape of an occluding object is known, including the position of the attached marker. Hence, we can project the 3D shape of the occluding object on the camera images. However, we still need to turn this shape information into final occlusion mask. In case of complex shapes we could render the whole 3D object to the occlusion mask using computer graphics techniques. However, some simple shapes, such as the calendar object in Figure 4, can be handled in a faster way. It can be seen that the outline of a rectangular object, such as the previously shown calendar, or any box-shaped object for that matter is always a convex polygon, regardless of the viewing angle. In case of convex shapes, we can ignore the internal structure of an object and only use its vertices to generate a convex hull i.e. the object outline. After obtaining a convex hull of the object we can fill it and obtain an occlusion mask.

Another detection method, the color blob based detector, works by finding objects with certain color and shape. First, we apply a watershed-based blob detection algorithm to find all blobs in image. Every found blob is then evaluated to see if it fits the object description. The process of checking object color is carried out as follows. The blob pixels are converted to HSV color space and average color values are calculated for all color channels. This process is also carried out on a set of training images of the object. We can measure variance of each color channel across the training data. This is useful when making actual evaluation of the blob colors – the average values of colors are allowed to vary around one variance distance of training data average color values. Finally, we can also apply some simple shape detection heuristics to further validate objects. For circle shapes, such as a ball used in experiments, we calculate the bounding box of every blob. If the width and height ratio is close to one, then the object might be a circular object. This process excludes many false-positive blobs that pass color checking.

The occluding object detection process is identical for both color and RGB-D camera. However, the way of using the occlusion masks from detection is completely different. In the following we describe both use cases. The RGB-D camera occlusion mask is used to exclude pixels from scene reconstruction. Essentially, the RGB-D camera depth map values are set to invalid values in areas where occluding objects were found. By removing occluding objects from RGB-D camera, it is guaranteed that such objects will not be included in scene reconstruction. The color camera occlusion mask is used after scene reconstruction, in compositing phase, and explained in corresponding section of this work in detail.

3.3 Scene reconstruction

The scene reconstruction method explained here expands on our previous work [7].

Very straightforward approach for projecting RGB-D images to the color camera view might be to project all RGB-D camera depth pixels separately to the color camera. This would, however, result in a sparse map of depth pixels in the other camera view. The problems with this approach are as follows. First, filling in the sparse map of depth pixels can be compuationally expensive as we need to search the neighborhood of every depth pixel for other pixels and then interpolate values between them. Secondly, the final result of our system should be a color image, not a depth map. This means that we need to carry out the proceedure of interpolating not only depth map, but also the color image between sparse depth map points in the target view image. Latest RGB-D cameras, such as Microsoft Kinect 2, tend to have color images many times the resolution of depth map. This process would essentially upsample the depth map to the color image resolution and hence inflate the transformation operations. Upsampling of the depth map does not increase the amount of information about the surroundings, so it would be reasonable to try to use a scene reconstruction method that works directly with the original depth map.

We decide to approach the problem from a different angle. To be able to render the scene at different viewpoint we first generate some other representation of the depth map that is easier and more reliable to process. Computer graphics research has brought

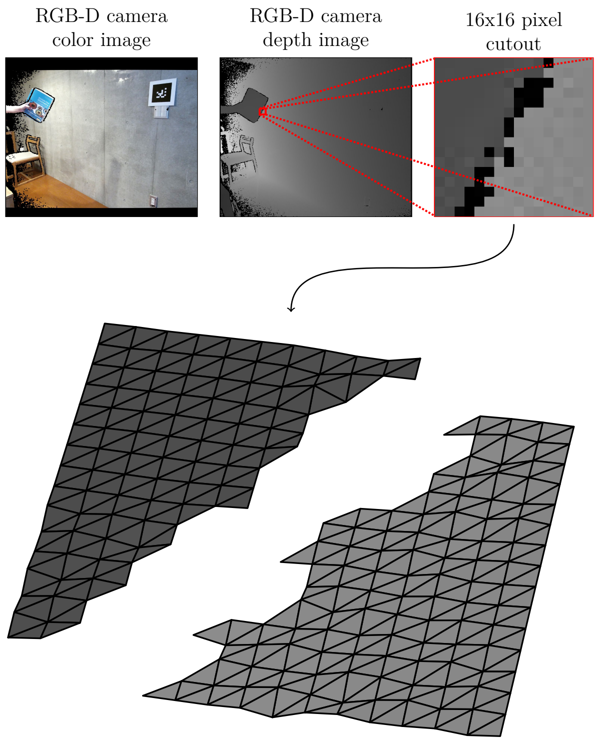


Figure 5: Step-discontinuity constrained triangulation (SDCT). A sample 16x16 pixel area has been cut out of RGB-D camera depth frame. The SDCT proceedure was applied to this cutout and the resulting triangles are shown in the lower part of the figure. Light grey pixels are further away from the camera than the dark grey pixels. As can be seen from the diagram, triangles corresponding to different objects have been correctly segmented to different triangle meshes.

us many powerful tools to deal with rendering of 3D data. It would make sense to leverage the available systems for our use. Typical computer graphics methods expect triangles for rendering. So it is important to realize a transition from RGB-D camera depth map to a set of triangles that make up the scene. The way we achieve that is later explained in scene reconstruction section in detail.

3.3.1 Step-discontinuity constrained triangulation

We have selected the so-called step-discontinuity constrained triangulation (SDCT) method [3] to generate triangle meshes out of RGB-D camera depth maps. This process is illustrated in Figure 5. The core idea in SDCT is that neighboring pixels in depth map typically correspond to close points in real scene. Therefore, triangles can be formed between space points defined by neighboring depth pixels to reconstruct an object's surface.

Triangle generation alone, however, is not enough. As seen in Figure 5, triangle mesh should be broken or segmented between different objects. In case we do not carry out the segmentation, any sort of viewpoint change will reveal triangles that join foreground and background objects, making the scene look incorrect.

The segmentation proceedure is implemented in very simple way in SDCT. Namely, any sort of depth change between neighboring triangles over certain constant discontinuity threshold will mean that the neighboring pixels are part of different objects. Hence, any triangles between such depth pixels should be deleted. Empirically values around 5 cm for this value were found to be producing good results.

3.3.2 Issues with SDCT

The segmentation method implies that surfaces with normals at very low angles relative to rays cast from camera might be removed by the procedure. In other words, if an object's surface is oriented such that the depth measurments of neighboring depth pixels differ more than the discontinuity constant then this surface is excluded from reconstruction. By knowing RGB-D camera properties we can calculate the angle of a surface at which it becomes invisible at different distances from the camera.

The calculation of minimum allowed surface angle in regions close to camera principal axis is illustrated in Figure 7. Writing down trigonometric relations of two triangles from the diagram we obtain

$$(x+d)\tan\alpha = d\tan\beta, \tag{1}$$

where β is the minimum allowed angle of the surface, α the angle between rays cast from two neighboring depth pixels of the camera, x the distance from camera to the object surface and d the discontinuity constant. We can extract the value of interest, β, and get

$$\beta = \arctan\left(\frac{x+d}{d}\tan\alpha\right). \tag{2}$$

The depth camera used in our experiments, Microsoft Kinect 2, has horizontal field of view of 70° and 512 pixels in horizontal direction. The angle α between rays cast from depth pixels depends on pixel coordinates in image plane. The value is highest near the principal axis, where $\alpha = 0.156°$, and lower elsewhere, such as at the edge of the depth map, where $\alpha = 0.103°$. The effect of surfaces becoming invisible is stronger at larger α values, so in the following we only consider the extreme case of $\alpha = 0.156°$ for estimation purposes. Given discontinuity constant of $d = 5$cm, we can calculate minimum surface angles at different distances. Our system is designed for indoor use and generally no scene in the experiments had dimensions over 5 meters. At 5 meters $\beta = 15.2°$ and at the much more common distance of 2.5 meters $\beta = 7.7°$. At those angles the surface is barely visible. Since so little is being seen of the surface, the reconstruction would also be very low quality. Hence, it is reasonable to exclude such areas from reconstruction in any case.

3.3.3 GPU acceleration

The SDCT reconstruction has been implemented in OpenGL shaders for accelerated execution on GPUs. The whole process has been divided between vertex, geometry and fragment shaders. In the following we describe the shaders in detail.

In the theoretical description, the triangle mesh generation, viewpoint transformation and rendering of the scene are considered as separate steps and carried out in this order. In practice, however, we can reorder some operations for the purpose of optimization. Also, we do not need to store any intermediate results such as the

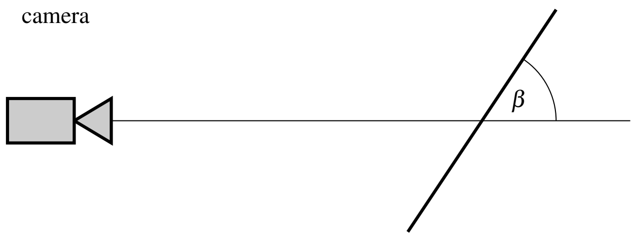


Figure 6: Minimum surface angle problem definition. Our task is to estimate the minimum allowed angle β where surface is still visible.

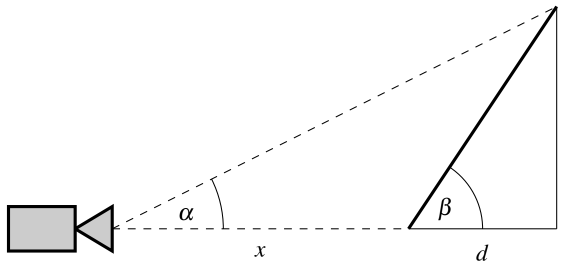


Figure 7: Minimum surface angle problem simplification. In this figure, α is the angle between rays cast from two neighboring depth pixels, d is the discontinuity threshold and x is the distance to surface from camera. This sort of simplification is possible by noting that the disparity threshold d is normally orders of magnitude smaller than the distance x between camera and surface.

triangle mesh data. Only the final color image of the reconstruction rendering is retained.

We start by uploading the depth map pixels to graphics card memory. The first stage of processing is done in vertex shader. Point coordinates (i.e vertices) in RGB-D camera space are calculated for every depth pixel. For even faster execution, the X and Y-axis projection coordinates in image plane for every depth pixel are stored in separate static buffer. The resulting point p in RGB-D camera space is then transformed to the target viewpoint using

$$p' = Rp + T, \tag{3}$$

where R and T are respectively the rotation and translation of viewpoint transformation.

Geometry shader is responsible for forming triangles between vertices calculated in vertex shader step. List of all theoretically possible triangle formations have been calculated in advance. Essentially, the geometry shader only has to go through the list and verify if a particular triangle is valid or not according to a set of rules and output them accordingly. The validation rules are as follows:

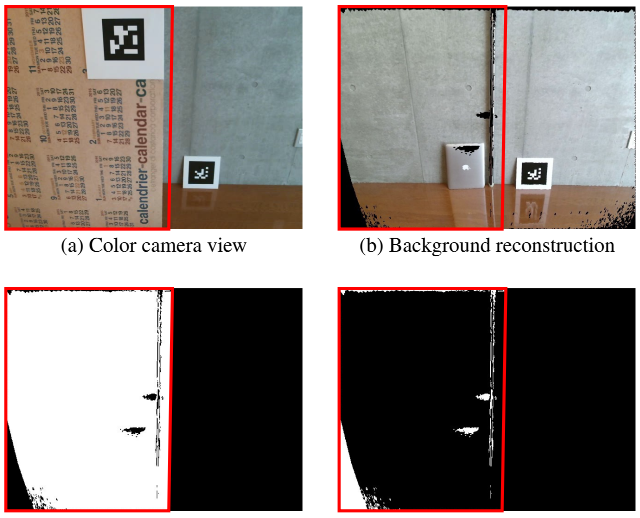
- **All triangle vertices must be valid.** Typically all real world depth maps contain at least some invalid depth points due to bad camera performance or occlusions.
- **Depth value difference between any two vertices of the triangle must be less than the discontinuity constant.** This rule hides triangles that would otherwise connect separate objects to a single mesh as discussed in previous subsections.
- **Triangle must be facing forward.** This rule makes sure that any back sides of objects are not made visible.

Final stage of the pipeline is using fragment shader. To output a color image we need to color in triangles generated in previous steps. We already have both the vertex coordinates and corresponding texture coordinates for every triangle. OpenGL automatically interpolates texture coordinates within triangles and performs color lookups.

3.4 Compositing

3.4.1 Ingredients for compositing

Figure 8 shows inputs to the compositing phase of our system. Our objective is to only hide occluding objects and therefore we only need to replace the image content of that region in color camera images. The main source for that replacement data comes from previously discussed scene reconstruction phase.



(a) Color camera view (b) Background reconstruction

(c) Valid pixel mask (d) Missing pixel mask

Figure 8: Compositing masks. The red rectangles highlight occluding object area. We need to use background reconstructon image to overlay occluding object in color camera view. The pixel mask below show succesfully reconstructed background pixels and the pixels that are missing from reconstruction.

It is not guaranteed that scene reconstruction always succeeds in all image areas. For example, due to the positioning of the RGB-D camera we cannot see some hidden regions completely, or the RGB-D camera is unable to read depth values in some image areas. The second issue is also visible in the Figure 8, where a metallic object causes holes in the RGB-D camera depth image and in turn results in some parts being missing from reconstrucion image.

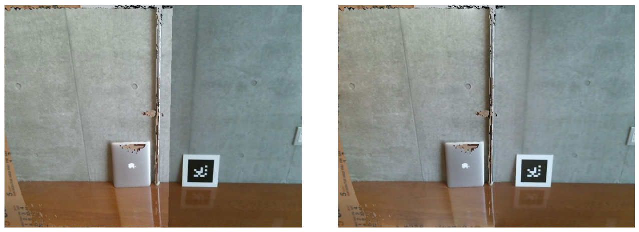
Fortunately, it is possible to develop an understanding of areas missing from reconstruction image. The scene reconstruction process, introduced in the previous section, saves final images in the 4-channel RGBA format. First three channels are the common RGB colors. However, the last channel, alpha, serves the purpose of indicating wheter a pixel was written during rendering. Therefore we can construct two masks from this alpha channel: a valid pixels mask and missing pixels mask. In both cases we constrain the masks to the regions of occluding objects.

In the following subsections we explain how the pixel masks are used to generate a composite result of of color camera and reconstruction images. In a nutshell, the valid pixels from reconstruction are copied using a color correction technique and the content of missing pixels is generated using image inpainting methods.

3.4.2 Color correction with seamless cloning

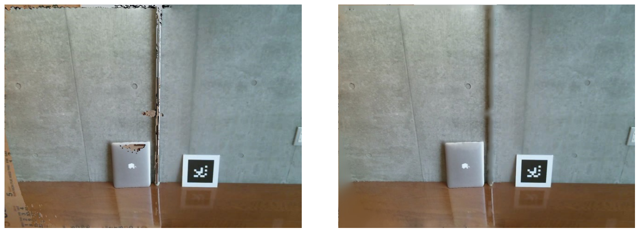
Most DR systems have opted to use homogenous set of cameras in demonstrations. This means that due to same lens and imaging sensors the colors across camera views are similar. In our case, however, there is a need to deal with very different cameras. This problem also exists in consumer applications where, for example, smartphone users have different device models and hence have cameras with wildly varying parameters. It is hard to expect that color calibration data would be available for all cameras. A good solution should work without calibration data.

It should be obvious that color differences can also arise from scene lighting – same cameras at various viewpoints percieve scene differently. This sort of problems have been considered in the field of relighting research.



(a) Simple compositing (b) Seamless cloning

Figure 9: Color correction example. (a) shows result created by copying pixels marked as valid from reconstruction image to color camera view without any processing. (b) shows merger of input images using seamless cloning to correct colors.



(a) Without inpainting (b) With inpainting

Figure 10: Image inpainting example. Missing areas on top of metallic surfaces have been repaired by inpainting. Lower left corner and top edge in the image have missing content due to the RGB-D camera field-of-view limits. These areas have also been filled in by inpainting.

Our solution to the problem at hand is to use seamless cloning technique introduced by Perez et al. [9]. Generally, this method seamlessly merges unrelated images. The effect of using this approach in our system can be seen in Figure 9. The method has number of good properties. First, no calibration process is necessary. Secondly, the color camera image colors are preserved, only inserted image is changed. Finally, small mismatches in alignment of input images are fixed automatically because of the seamless property of image insertion.

3.4.3 Guessing missing content with image inpainting

Image inpainting techniques are intensively studied and constitute a subfied of diminished reality on their own. Their purpose is to hide objects by procedurally generating image content by looking at image patterns. Generating content for complex backgrounds is difficult, and probably impossible when there are no repeating patterns. However, smaller areas with limited amount of variation could be filled with generated content. During experiments, we encountered situations where holes in scene reconstruction happened due to small specular objects, such as a steel pipes. To repair those image areas, we applied image inpainting method by Telea et al. [11]. The particular work was specifically advertised as effective in repairing damaged images, where inpainted areas are in limited size. The results of applying inpainting to the areas of missing pixels is shown in Figure 10.

4 Experimental results

4.1 Setup

The cameras used in the experiments are given in Table 1. The host computer had Intel Core i7-4770K CPU clocked at 3.5 GHz, 16 GB of RAM and GeForce GTX 780 graphics card. The system ran Gentoo Linux operating system.

Table 1: Camera hardware

Camera	Color camera	RGB-D camera
Model	Point Grey Flea3	Microsoft Kinect 2
Color resolution	1280x1024	1980x1080
Depth resolution	N/A	512x424
Framerate	30 fps	30 fps

The color camera had adjustable lens. It was necessary to open up the aperture to F1.6 to reduce motion blur in images. Otherwise, any moving AR markers would be hard to detect and result in incorrect pose estimation values. The lens settings were fixed and the camera intrinsic parameter calibration was carried out using a board with checkerboard pattern. The intrinsic parameters for RGB-D camera, both color and depth frames, were acquired from factory calibration data stored inside the Kinect 2 device.

4.2 Scenes

We set up two scenes to test our system – a ‘calendar’ scene and a ‘ball’ scene. Both are toy problems, but they should convey the generality of our approach. The fundamental difference of the scenes lies in the occluding object detection algorithm being used.

4.2.1 Calendar scene

The objective in calendar scene is to hide a calendar object with attached AR marker. The dimensions of the object are known, including the AR marker position. The detection method was previously explained in object detection section. Results are shown in Figure 11.

4.2.2 Ball scene

The target in ball scene is to detect a blue colored toy ball using blob detection and remove it from the scene. Again, the exact object detection method could be found in object detection section. Results can be seen in Figure 12.

4.3 Performance

System performance related data has been given in Table 2 for all major processing steps of our system. The necessary core operations for system operation are the pose estimation and scene reconstruction. These steps plus simple compositing can easily run in real time – the frame rate is limited only by camera speeds. Therefore we can achieve diminished reality in real time.

Table 2: System operations brakedown with computation time metrics

Operation	avg.	min.	max.
Pose est. and object det.	8.2 ms	8.1 ms	15.7 ms
Scene reconstruction	12.8 ms	6.6 ms	23.6 ms
Simple compositing	1.1 ms	1.1 ms	1.2 ms
Color correction	13256.1 ms	12517.3 ms	14038.3 ms
Guessing missing content	1871.5 ms	540.9 ms	3128.7 ms

However, if we want to add color correction step and guess missing image content then the computational cost is too high for real time execution. These steps are expensive partly due to the algorithmic complexity, but also due to the poor performance of third party implementation of seamless cloning and image inpainting methods.

5 Conclusion

In this paper we demonstated a novel diminished reality system. Our approach allows movement of scene background, occluding objects and all cameras – the system can handle fully dynamic scenes. Additonally, we take care of color differences across cameras to get visually pleasing results and repair image areas where RGB-D camera has failed to capture scene correctly.

Acknowledgements

This work has been supported in part by JSPS Grant-in-Aid for Scientific Research(S) 24220004, and JST CREST “Intelligent Information Processing Systems Creating Co-Experience Knowledge and Wisdom with Human-Machine Harmonious Collaboration”.

References

[1] A. Enomoto and H. Saito. Diminished reality using multiple handheld cameras. In *Proc. ACCV*, volume 7, pages 130–135, 2007.

[2] S. Garrido-Jurado, R. Muoz-Salinas, F. Madrid-Cuevas, and M. Marn-Jimnez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.

[3] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Computer VisionECCV'96*, pages 117–126. Springer, 1996.

[4] T. Honda and H. Saito. Realtime diminished reality based on 3d measurement of environment using an rgbd camera. *Transactions of the Virtual Reality Society of Japan*, 19(2):105–116, 2014.

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

[6] S. Jarusirisawad and H. Saito. Diminished reality via multiple handheld cameras. In *Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on*, pages 251–258. IEEE, 2007.

[7] S. Meerits and H. Saito. Visualization of dynamic hidden areas by real-time 3d structure acquistion using rgb-d camera. In *3D Systems and Applications Conference*, Aug. 2015. (accepted for poster presentation).

[8] S. Mori, R. Ichikari, F. Shibata, A. Kimura, and H. Tamura. Framework and technical issues of diminished reality: A survey of technologies that can visually diminish the objects in the real world by superimposing, replacing, and seeing-through. *Transactions of the Virtual Reality Society of Japan*, 16(2):239–250, 2011.

[9] P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM, 2003.

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

[11] A. Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004.

[12] S. Zokai, J. Esteve, Y. Genc, and N. Navab. Multiview paraperspective projection model for diminished reality. In *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, pages 217–226. IEEE, 2003.

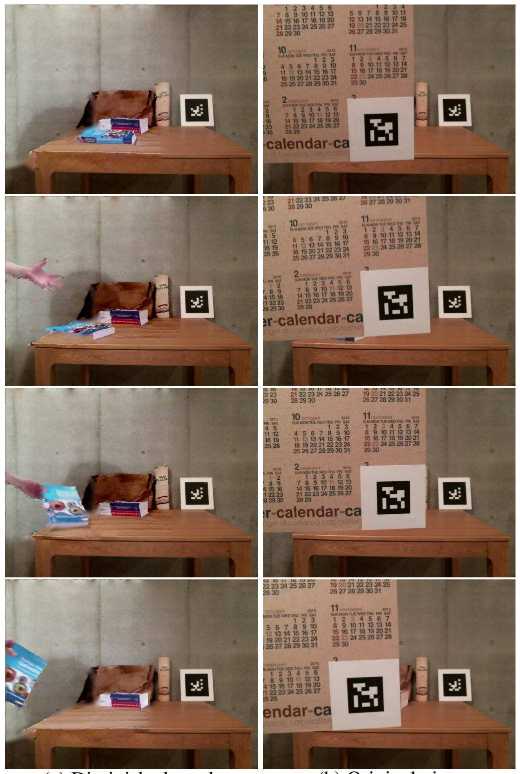

(a) Diminished result (b) Original view

Figure 11: Video outtakes from calendar scene experiment.

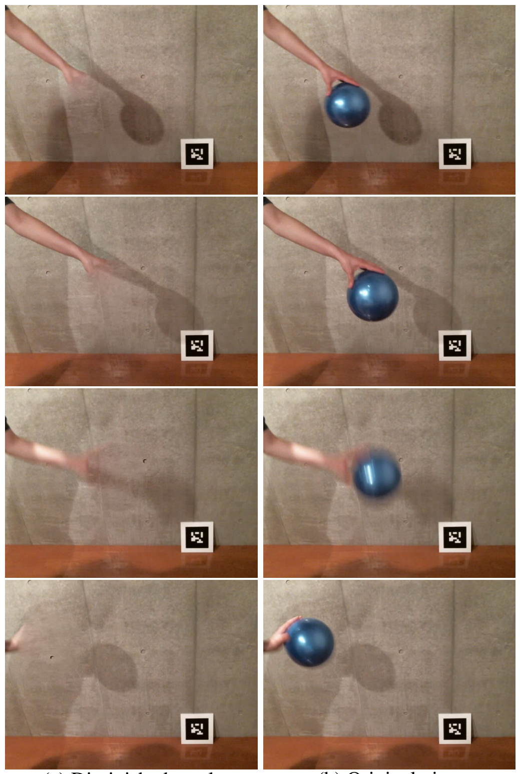

(a) Diminished result (b) Original view

Figure 12: Video outtakes from ball scene experiment.