Stereoscopic augmented reality with pseudo-realistic global illumination effects

Francois de Sorbier and Hideo Saito
Keio University, Department of Information and Computer Science
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, Kanagawa, 223-8522, Japan
Tel: +81-45-563-1141
{fdesorbi,saito}@hvrl.ics.keio.ac.jp

ABSTRACT

Recently, augmented reality has become very popular and has appeared in our daily life with gaming, guiding systems or mobile phone applications. However, inserting object in such a way their appearance seems natural is still an issue, especially in an unknown environment. This paper presents a framework that demonstrates the capabilities of Kinect for convincing augmented reality in an unknown environment. Rather than pre-computing a reconstruction of the scene like proposed by most of the previous method, we propose a dynamic capture of the scene that allows adapting to live changes of the environment. Our approach, based on the update of an environment map, can also detect the position of the light sources. Combining information from the environment map, the light sources and the camera tracking, we can display virtual objects using stereoscopic devices with global illumination effects such as diffuse and mirror reflections, refractions and shadows in real time.

Keywords: Augmented Reality, Depth Camera, Global Illumination, Real Time, Pseudo-Realism, Stereoscopic Rendering

1. INTRODUCTION

For years, many researchers have focused on augmented reality trying to efficiently mix virtual objects with real environments. Augmented reality is now widely available for many applications, such as information, game or advertisement, using smartphones or PCs. Even if augmented reality starts to be largely accepted as a new communication mean, it still lacks realism. Virtual objects are often added without any consideration of the surrounding environment.

One reason is the difficulty to automatically and accurately extract the geometry and the illumination properties from this environment. Moreover, it is such a complex problem that most of the realistic augmented reality applications do not run in real-time. Nowadays, researches in augmented reality have strongly progressed thanks to the availability of several consumer level depth cameras. Besides the color image, the depth camera is able to capture the corresponding depth information with real-time frame-rate. The data captured by this camera can then be processed for clearly and quickly detecting the surfaces.

A first category of depth cameras, such as the SwissRanger SR4000[1] or the DepthSense[2], uses the time of flight technology (TOF). Its advantage over traditional stereo cameras is that TOF cameras provide accurate depth maps in real time on a practical distance and under normal illumination. However, this category of device suffers of low resolutions (176x244) and very high prices.

Few years ago, Microsoft has released the Kinect, a depth camera based on a structured light approach combining a coded infrared light projector and an infrared camera [1]. Kinect has quickly become very popular, mainly because of its low price. More than being a cheap device, Kinect has also the advantage of capturing higher resolution depth maps compared with TOF cameras and also appears to be slightly more accurate [2,3].

[1] http: //www.mesa-imaging.ch

[2] http://www.softkinetic.com/

Kinect has been a great stimulus for the research in human machine interactions but also for augmented reality. In their seminal work, Newcombe *et al.* introduced KinectFusion [4,5] that demonstrated extended capabilities of depth cameras. Several other researches [6,7,8] have been performed considering the use of depth cameras for augmented reality. Even if the interactions among the real and the virtual elements are now more accurate, very few methods are considering the surrounding environment for a realistic rendering of the virtual objects in real time.

In this research, we propose a system for rendering pseudo-realistic virtual objects in a real environment by using a Kinect-like depth camera and display it with stereoscopic devices. We do not use any assumption on the global scene geometry and we want to have to perform an offline processing. We then considered a method based on environment maps updated continuously with the color and depth data from the surrounding environment. These storing structures are also used for estimating the position of the light sources in the 3D space. By knowing the local geometry of the scene and the position of the lights, we are able to render in real-time effects related with global illumination like reflections, Lambertian reflectance, refractions and shadows. Since we do not consider all the effects related to global illumination (multiple reflections, caustics, BRDF, translucent materials), we refer to our rendering process as credible or pseudo-realistic. Having a local reconstruction of the scene, we also demonstrate that it can be used to generate new virtual views from slightly different viewpoints and produce the input images of stereoscopic or auto-stereoscopic displays.

The main contributions of our paper are then:

- Automatic detection of light sources
- Real time rendering of reflections and refractions
- Soft shadow rendering onto the scene
- Use only one Kinect (no fish-eye camera, no tracker, no mirror ball)
- No pre-computation or marker are required
- Real-time processing and rendering on stereoscopic displays

The rest of the paper is organized as follows: In the next section, we will review the previous works in relation with our goal. In section 3, we give an overview of our framework. In section 4, we will detail our approaches for the stereoscopic pseudo-realistic rendering of virtual objects. In Section 5, we present and discuss our results**.**

2. PREVIOUS WORKS

In this section, we give an overview of the previous works that were aiming at environment aware augmented reality. Even if few interesting solutions have been proposed based on a single color camera like [9], [10] and [11], we will mainly focus on works taking advantage of depth cameras.

One problem that often arises with single camera based augmented reality is the resolution of occlusions. With the release of depth camera, several works [12,13] took advantage of the depth information for an easier computation of the occlusions. Especially, with computer graphics techniques, in become easy to perform a depth test based on the data provided by the depth camera. Even if the scene is dynamic, virtual objects can be easily and quickly occluded by real surfaces.

Koch *et al.* [14] extended the use of depth cameras by taking into account the illumination of the scene for a better coherence. This work required to define manually the 3D position of the light sources. From each of the point lights, a shadow map is computed and applied on a reconstruction of the scene. However, if the number of lights becomes important the performances may decrease, and only point lights are considered.

Considering the illumination of the scene, Frahm *et al.* proposed an augmented reality application [15] with an automatic detection of light sources. The depth data were used for detecting a target planar surface onto which the virtual object will be added. In addition, a fish eye camera is used for detecting the saturated colors that might represent point light sources. Those light sources are then used for computing the new shadows casted in the scene by the virtual objects.

In previous works [16,17], we proposed an interactive augmented application using a depth camera coupled with a color camera. From the depth information, we performed a reconstruction of the scene and linked it with a physic engine. A virtual object could then be released in the scene and directly reacted to the changes of the surfaces. For instance, It was possible to push a virtual object with the hand. The application also included several effects like depth of field, shadows with a predefined static light and it was also possible to render the result on auto-stereoscopic displays. Besides being not realistic, this application suffered of the flickering of the depth map that made the pose of virtual objects unstable.

Some techniques were focusing on reducing the instability of the depth map. Usually, they consist in enhancing the quality of the depth map with a filter like the bilateral filter [18]. Several approaches are trying to detect planar surfaces or specific objects [19-21] for correcting the depth map. For example, Lensing and Broll [22] have proposed several steps for improving the depth map for a mix reality application. They first apply a Gaussian blur and then use the edges extracted from the color image for refining the depth values over local patches.

The goal of the system presented in this paper is mainly related to the work introduced by Kan and Kaufmann [23,24] on realistic augmented reality with a single color camera. The authors use an AR Toolkit marker for both tracking the camera position and superposing a pre-reconstructed model of the scene. By applying a ray casting technique, authors propose to create a realistic depth of field effect but also show that it can be extended for also considering the global illumination. However, this work suffers of many drawbacks. First, the application requires a pre-computed model that is manually created and then implies that objects cannot be moved. Second, markers are required. Finally, this approach can handle occlusions only between the reconstructed scene and the virtual objects.

In our knowledge, there is no previous work using a depth camera for augmented reality that is able to render convincing global illumination effect such as reflections, refractions and shadowing in real time.

3. OVERVIEW OF OUR AUGMENTED REALITY SYSTEM

Our purpose is to create an application that analyzes the scene for detecting light sources and the geometry, and render the virtual objects according to the result of this former analyze. The overall approach is summarized in Figure 1.

In this section, we will describe the different stages composing our approach.

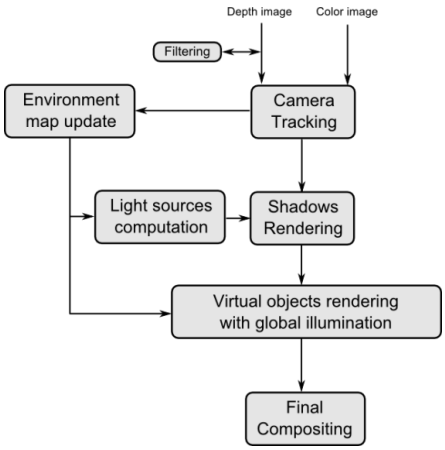


Figure 1. Overview of the different steps of our pseudo-realistic augmented reality system.

3.1 Tracking of the depth camera

An augmented reality application usually involves being able to freely move the viewpoint around a target surface. For this reason, a key-point of our method is to estimate in real-time the pose of the depth camera for further processing. We will then detail the choice of the tracking technique we have retained for our approach.

An advantage of a depth camera is to be able to provide the corresponding 3D data thanks to a pre-calibration stage. Using these data, several methods have been proposed to compute the linear rigid transformation between a source and a target point clouds and then estimated the pose of the camera. The most common concept is the Iterative Closest Point (ICP) algorithm introduced by Zhang [25]. The naive version consists in finding correspondences between points of the source and the target based on the shortest distance. From these sets of correspondence, we evaluate the rigid transformation thanks to a least-square fitting and remove outliers. This process is repeated until we are able to found the best transformation that fits the source with the target. When applied on an unknown and dense dataset, the algorithm tends to be really slow and does not always converge, especially when data are noisy like with Kinect-like cameras.

Several extensions of the ICP algorithm have been proposed and are mainly summarized in [26]. To improve the accuracy and the speed, those methods assume small movements between two captures and different metrics for

estimating the correspondences between the both point clouds. However, those approaches remain inaccurate against noise and still slow.

For reducing the computational time, some methods have introduced the use of key-point matching techniques and then rely on the choice of good descriptors. Since the Kinect captures 3D points, possible descriptors can be Normal Aligned Radial Feature (NARF) [27] or the Fast Point Feature Histograms (FPFH) [28]. In that case, the normals have to be computed, which can be done quickly with Kinect-like camera since the data are organized [29]. However, the noise can still be a problem during the registration, even if data are smoothed.

Kinect-like cameras have the advantage to be calibrated in a way that there is a known relation between depth data and pixels from the color image. It is then possible to use both the color and the depth for estimating the pose of the camera. With feature detectors and descriptors like SIFT [30] or SURF [31], we can quickly find out correspondences between two consecutive frames. The 3D coordinates related with the descriptors are then extracted from the depth map and the rigid transformation is finally computed. An example of such approach has been proposed by Huang et al. in the context of visual odometry [32] with the FAST feature detector. The correspondences are found with a KD-tree nearest neighbor search (FLANN) and the 3D data extracted. Finally, the rigid transformation is estimated by applying a least square fitting on correspondences as described by Arun [33]. However, this color-based approach is highly dependent to color variations and can lead to an inaccurate tracking.

Recently, the Microsoft's Kinect Fusion [4,5] has applied the point-to-plane ICP algorithm on GPU for proposing a real-time Simultaneous Localization And Mapping (SLAM) of the Kinect. Data are progressively stored in a voxel map and compared with the last capture of 3D points. Advantage of this approach is that it decreases the influence of the noise since point clouds are merged based on a Truncated Signed Distance Function (TSDF).

This latest approach appears to be the most suitable for our case, since it is accurate and fast. Besides getting the pose estimation of the camera, we also use the result of Kinect Fusion for improving the quality of the depth map. Basically, Kinect Fusion results in a limited voxel grid that does not cover the entire space and that is reset every time the field of view is outside of this 3D volume[3]. If we directly use the data from this voxel grid for generating a better quality depth map then some data might be unavailable and the processing time will be higher since a ray casting has to be performed.

We have then decided to accumulate the frames based on the pose estimation. For each new capture, we transform the previous depth map onto the current frame. For each pixel, we compare the depth values *D* as follows:

$$D_t = \begin{cases} D_{t-1}, & if\ \mathrm{D_t}=\emptyset \\ D_t, & if\ \left\|\mathrm{D_t}\text{-}\mathrm{D_{t\text{-}1}}\right\|>\delta \\ \mathrm{D_t}\times 0.2+\mathrm{D_{t\text{-}1}}\times 0.8, & otherwise \end{cases} \quad (1)$$

where *t* is the frame's time of capture. Advantages of this approach are a smoothing of the data over the time, holes are filled, and we can take dynamic objects into account by adjusting the parameter δ.

3.2 Capture and storage of the surrounding environment

The goal of our augmented reality application is to add virtual objects in the image captured by our Kinect-like camera. An important point is that the inclusion has to be credible, and not only be synthetic objects displayed independently of their surrounding environment. Our approach should then be able to render effects such as reflection, refraction and illumination conditions. The whole surrounding scene needs to be stored with the constraint of using only one single depth/color camera. Solutions involving mirror balls or fish eye lenses are not considered.

Our proposal is to associate an axis aligned bounding box (AABB) [34] to each virtual object. Each face of the box is a texture that holds the color information corresponding to one axis direction. For every captured frame, we map the color image onto the face of the cube according to the pose of the camera. We preferred a bounding box rather than a bounding sphere since this latter has important loss of precision at the poles.

[3] At least in the implementation available at http://www.pointclouds.org

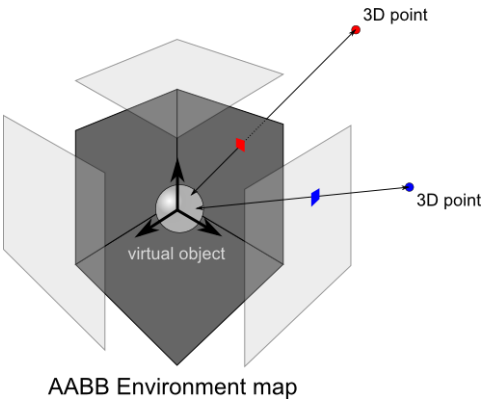


Figure 2. Demonstration of the update of the environment map where the box is centered on the virtual object. For each 3D point in the global referential, we compute the intersection of the ray and the faces of the cube. If there is an intersection then the content of the map is updated. The computation is speeded up by using an axis aligned bounding box.

For updating the faces of this environment map, we use the 3D coordinates obtained from the depth camera. The coordinates are then transformed into a global referential by applying the current pose estimation of the camera. For each of these 3D points, we generate a segment delimited by the center of the box and we check if there is an intersection with one of the faces of the box. If an intersection between the segment and a face is found, we update the color C of the face at the position (u, v) of the intersection and store the distance d of the point in the alpha channel as depicted in Figure 2. The update of each face is given by three conditions defined as follows:

$$C_{(u,v)} \begin{cases} C_{(u,v)}, & if\ \alpha \leq \left\| d_{input} - d_{(u,v)} \right\| \leq \beta \quad (1) \\ \frac{C_{input} + C_{(u,v)}}{2}, & if\ \left\| d_{input} - d_{(u,v)} \right\| \leq \alpha \quad (2) \\ C_{input}, & otherwise \quad (3) \end{cases} \tag{2}$$

The case (1) is applied for resolving occlusion on small objects. If the camera is observing the back face of an object then we should keep the current value that might be the opposite face observed from the center of the cube map. Note that this update only works on objects smaller than $\beta - \alpha$. The case (2) is used to smooth the color when similar 3D coordinates are processed. We add a tolerance of α for the difference of depth for also taking into account small re-projection errors and noise. The last case (3) has been added for yet unavailable data, but also considering dynamic objects that might drastically change the depth value. Example of an ongoing update of a bounding box is presented in Figure 3.

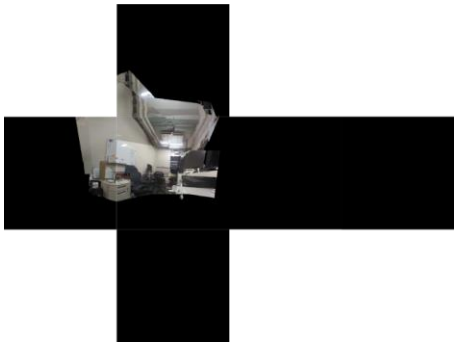


Figure 3. Example of an on-going capture of the scene into the environment map. Each face of the cube is represented by a square texture. The rest of the map will be filled over the time considering black areas will be covered by the depth camera.

3.3 Estimation of the position of lights

The estimation of the position of the light sources is an important step since it will affect the rendering of shadows the computation of illumination. Several methods have been proposed for estimating the position and the characteristics of the light sources.

A traditional approach is to use a mirror sphere [10] located at a known position to be able get information about the environment. Using this approach, Nowrouzezahrai *et al.* [35] proposed a light factorization model with spherical harmonics to compute precisely the characteristics of the light sources. Moreover, the approach is real time thanks to the GPU and can be applied with dynamic lights.

For avoiding the use of a mirror ball, Ikeda *et al.* [36] proposed an improvement of the illumination from shadows method [37]. Instead of using a pre-computed model, the authors suggest to use Kinect for detecting a specific object in the color image. For each shadow pixel, caused by the presence of this object, a ray is emitted in the direction of a hemisphere and intersections are checked with this particular object. Finally a high dimension system is resolved for estimating the position of the lights. However the complexity of the computation avoids the algorithm to run in real time, even on GPU.

Recently, Gruber *et al.* [38] proposed an estimation of the lighting environment thanks to the reconstruction of the scene obtained with Kinect Fusion and spherical harmonics are used to store the information. Unfortunately, the result is running at interactive frame rate, even with intensive use of the GPU.

For our case, we did not want to use specific devices like a mirror ball and we want to be able to estimate the position of the light sources in real time, so none of those methods were applicable. We then propose an alternate approach, less accurate on the estimation of light sources' characteristics, but very fast.

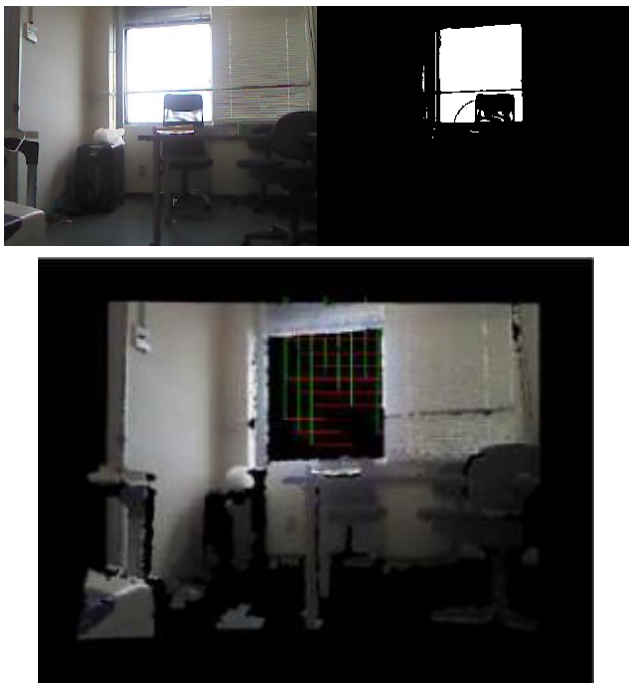


Figure 4. Result of the light sources detection. In this example, the window is the only light source (upper left). The image is then segmented based on saturated areas (upper right) and light sources are extracted into the 3D space from the sampled binary light map (bottom). Even if the can be interpreted as a single point light, our approach allows to sample it into several light sources.

For our application, we adapted the light source detection detailed in [15]. The authors proposed to use two fish-eye cameras to triangulate the position of the light sources. For each input image, the pixels with a high saturation level are

retained as a potential candidate for belonging to a source light. Finally the source lights are deduced by applying a region growing technique. The position is deduced by taking the center of gravity of each of those regions. This implies that light sources are reduced to single points even if they represent large areas.

Our light source detection approach is also based on the localization of saturated elements in the scene. For this, we convert each face of our environment map into a HSV color representation. If the saturation level is over some threshold then it might be a candidate for being a light source. To avoid misdetection like reflections on surfaces, we add another criterion based on the distance. For each color stored in the box, we also have access, in the alpha channel, to the corresponding 3D position. For now, we keep the saturated pixels only if they are at minimum of 1.50 meters from the center of the box, which is enough for detecting lights on the ceiling and coming from windows. The result of this part is six binary images corresponding to the six faces of the box.

It might be possible to associate a light source to each saturated pixel, but the processing time for the computation of shadows will be too much high. We then propose to subdivide the faces into bigger sized cells. For each cell, if the number of included saturated pixels is over a given threshold φ, then it will be considered as a light source. Next, we compute the center of gravity of the saturated pixels and back project it using the distance stored in the alpha channel for finding its 3D coordinate in the scene. Besides the 3D position, we also store the ratio between the total number of pixels in the cell and the actual number of saturated pixels for further computation of soft shadows. Our light extraction process is depicted in Figure 4.

Using this approach, we can approximate lights coming from windows or fluorescent tubes with multiple light sources, but also remove the noise. This approach is coupled with to the real-time update of the environment map, but even if we are theoretically able to detect moving light sources in the field of view, we are considering static light sources.

4 STEREOSCOPIC RENDERING OF CREDIBLE VIRTUAL EFFECTS

In this section, we introduce all the visual effects we are able to perform based on our environment map data structure and our light detection approach. We will first detail how we can generate Lambertian + Phong illumination with shadow casting. Then describe our approach for mirror reflections and refractions. Finally, we will introduce our stereoscopic rendering.

4.1 Illumination and shadows

Each light we have detected in the environment will influence the illumination of our virtual objects. In our approach, we limit the illumination coming directly from light sources and we do not consider the indirect illumination. We based our illumination rendering on the Lambertian and Phong light models defined for each light as follows:

$$\mathrm{I} = \sum_{n=0}^{lights} I_n \quad (3)$$

$$I_n = (K_d L_d(\vec{l}.\vec{n}) + K_s L_s(\vec{r}.\vec{v}) + K_a L_a) \times V_n/lights \quad (4)$$

where *lights* is the number of detected lights, *K* is the object's color and *L* the light color for the diffuse term *d*, specular term *s* and ambient term *a*. $\vec{l}$ is the vector from the surface of the object to the light, $\vec{n}$ the normal, $\vec{r}$ the reflected vector (half-vector) and $\vec{v}$ the vector from the surface of the object to the viewpoint. *V* is the binary term depicting the visibility of the light source for shadows. The term V is estimated by comparing the distance of the current light source with the depth stored in the environment map. If values are similar then V is set to 1, otherwise 0.

The presence of virtual objects in the scene modifies the appearance of the real surfaces by casting shadows on it. Knowing the position of the light sources, one possibility is to apply computer graphics techniques for rendering the shadows, such as shadow mapping or shadow volume. However, if many light sources are detected, those techniques will not be suitable since the result will not be real-time.

For this reason, we preferred the use of a ray caster. For each pixel of the current captured color image, we find the corresponding 3D position in the global coordinate system by using the depth map and the pose estimation of the depth camera. From this 3D position, we emit a ray in the direction of each light source. We then compute the intersection between those rays and the virtual object. If an intersection is found, then the pixel corresponding to the origin of the emitted ray has to be shadowed.

Figure 5. Example of Lambertian reflectance with the Phong Model and shadows. Positions of light sources used for the computation of the illumination are extracted from the environment map.

The change of appearance of the pixel is computed as follows. We evaluate the ratio between the total number of lights and the number of intersections. This ratio is then applied to the color of the pixel to decrease its luminosity. As presented in the previous section, the large light sources are drastically sampled. Therefore, the shadows might appear sharp. For reducing this effect, we propose to emit extra rays when checking for intersections with the virtual objects. The direction of the rays is based on a circle perpendicular the original ray direction. The radius of the circle is defined by the distance between the origin of the ray and the position of the light. The number of rays is empirically defined by using the ratio of saturated pixels we found in a cell. For instance, we emit 12 extra rays if the ratio is higher than 0.7, and only 1 if the ratio is between 0.5. We can then render large light sources as well as point light sources. This is illustrated in Figure 6.

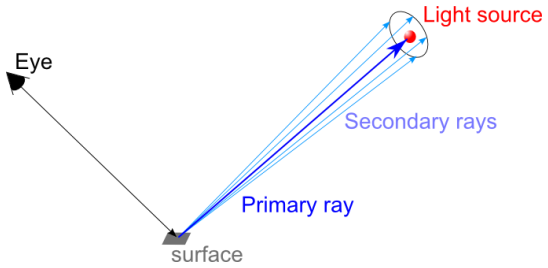


Figure 6. For each 3D point related to a pixel of the color image, we emit several rays in the direction and close neighborhood of the sources lights and we check for intersections with the virtual objects. The effect of sending several rays is a smoothing of the appearance of shadows.

4.2 Mirror reflections and refractions

The bounding box that we previously computed and updated is an important part for generating the mirror reflections and the refractions on the virtual objects. We define the mirror reflections as light that this partially or completely reflected by the surface, and refraction as light going through the surface of the object. A common approach to render the reflections and the refractions is to use the principle of environment map, also called skybox that is actually a 3D texture.

For each vertex of the virtual object, we need to compute the normal and the ray from the vertex going in the direction the view position. From those data, we compute the vector resulting from the reflection on or the refraction through the surface. Finally this vector is used to get the color from the environment map and information is added on the virtual object. An example of result is depicted in figure 8.

4.3 Stereoscopic rendering

For the stereoscopic rendering of our scene, we have to generate from 1 to *n* extra views at slightly different viewpoints. For this, we use the same approach we presented in [16,17].

We take advantage of the 3D reconstruction provided by our Kinect-like depth camera for generating the new images. This changes our image-based problem into a standard OpenGL stereoscopic rendering. We build a triangular mesh from the point cloud captured by the depth camera and projected the corresponding color image onto it. With this mesh representation, colors are automatically interpolated. We then translate the position of the virtual viewpoint as many times as we require input images for our stereoscopic device, and render the results in textures.

5 RESULTS AND DISCUSSION

In this section, we first introduce our setup for our experiments, describing our system configuration, but also the values of some of the parameters. Next, we present our results qualitatively and finally discuss some of our choices with possible improvements.

5.1 Details about the experiments

Our experiments were conducted in an indoor environment without any special geometry or content. This test room is illuminated by three groups of fluorescent tubes and the outside light coming from a window. There are also several reflections coming from surfaces like white boards.

Figure 7. Our depth camera mounted on the Sony's 3D HMD.

Our virtual object is a sphere with a radius of five centimeters. The position of the object is defined at the position of Kinect when starting the program. We tried to set the position of the object over the center of a planar surface, but using the origin of the Kinect provide more control on the position and the orientation of the virtual object.

The resolution of the faces of the environment map was defined at a resolution of 640×640. The range used for the update of the environment map was defined with $\alpha = 1\text{cm}$ and $\beta = 5\text{cm}$. The size of the cells for the light sources estimation was 10×10 with a threshold φ of 0.45.

The framework was implemented on an Intel Core i7 3.20GHz with 16Go of RAM running under the distribution Ubuntu 12.04. The graphic card is a *nVidia GeForce GTX 580* with 1.5Go of RAM. Our depth camera is an *ASUS Xtion Pro Live* that we access using the library OpenNI v1.5. For real-time computing, we are making intensive use of CUDA for the processing (tracking, back-projection, ray/plane intersections, light sources estimation) and OpenGL 4.0 with GLSL for the rendering.

For the stereoscopic rendering, we used a Tridility auto-stereoscopic display with 5 views as input, and the 3D HMD HMZ-T1 from Sony. In the latter case, our depth camera was mounted on the top of it, as illustrated in Figure 7.

5.2 Results

While running our framework, we can achieve a frame-rate between 26 and 30 frames per second with the update of the environment map and the computation of the lights' position activated. An example of result is presented in Video 1 and the average of computational time is summarized in table 1.

We extracted approximately 115 light sources from our indoor test room. The precision of the locations of these light sources is strongly related with the accuracy of the tracking. Figure 4 shows the relation between the light sources position and the saturated pixels. Note that even if the light is coming from one large area in the image, we are able to sample it in several point lights. However, we were unable to disable the white balance of depth camera, so the detection of saturated pixels became sometime unstable.

Table 1. Results of the average speed for each step of our process.

Steps	Average speed (msec)
Tracking	15
Update environment map	5
Compute light position	6
Compute shadows	9
Render background with correct depth map	4
Render virtual object	3
Total	42

5.3 Discussion

Using Kinect Fusion[4] for tracking the camera is enough for accurately estimating the pose of the camera and generating the content of the environment maps. The voxel-based storing approach of Kinect Fusion is yet requiring large memory space with a standard grid's resolution of $256 \times 256 \times 256$. As described in [4], a frame-to-frame tracking is not accurate since it accumulates errors over the time. A solution we will try to investigate in the future is to use our frame accumulation approach to perform the pose estimation. Compared with a simple frame-to-frame approach, we might be able to track the camera with good accuracy while the amount of required memory will remain small.

In our rendering model, we consider only the direct illumination coming from the light source we detected. For this reason, we are speaking of pseudo-realistic rendering since several effect related to global illumination are still missing. If we want to take into account the indirect illumination, we might need to extend our ray tracing to the neighborhood of each virtual object, or apply a technique like photon mapping. In that case we will also be able to render effect such as caustics that should be visible with translucent virtual objects.

For updating the environment map of each object, the user needs to move the camera in all the directions in order to capture the whole environment. For a better coverage of the environment, we are thinking about adding some guides that will help the user to know where the camera should be directed or which areas have not been covered yet.

6 CONCLUSION

We have presented our framework designed for achieving convincing augmented reality. We proposed to use a depth camera for updating the color of an environment map. This approach avoids the use of 3D reconstructions that use large amount of memory and are suitable for being quickly updated during the rendering process.

The environment map is also used for finding the position of light sources by detecting high saturated areas. We proposed to sample those areas for sampling the large light area like windows into multiple light sources.

Using the light sources and the environment map, we were able to render virtual objects in real time with various effects related with global illumination. For instance, we presented results with Lambertian reflectance, mirror reflection, refractions and shadows.

[4] http://www.pointclouds.org

We will also improve the realism of the rendering by probably applying a ray tracing method not only for the shadows. Using this approach might give a better control on the propagation of light in the scene, but needs to be coupled with GPU processing for real time rendering.

ACKNOWLEDGMENTS

This work is partially supported by National Institute of Information and Communications Technology (NICT), Japan.

REFERENCES

[1] J. Kramer, M. Parker, D. Herrera, N. Burrus, and F. Echtler, [Hacking the Kinect], Apress, (2012).

[2] J. Smisek, M. Jancosek, and T. Pajdla, "3d with kinect," Proc. IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 1154–1160 (2011).

[3] K. Khoshelham and S. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications," Sensors 12(2), 1437-1454 (2012).

[4] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," Proc. 10th IEEE International Symposium on Mixed and Augmented Reality, 127-136 (2011).

[5] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison et al., "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," Proc. 24th annual ACM symposium on User interface software and technology, 559–568 (2011).

[6] S. Kahn, "Reducing the gap between augmented reality and 3d modeling with real-time depth imaging," Virtual Reality 17(2), 111-123 (2013).

[7] L. Cruz, D. Lucio, and L. Velho, "Kinect and rgbd images: Challenges and applications," Proc. 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials, 33-49 (2012).

[8] A. Kolb, E. Barth, R. Koch, and R. Larsen, "Time-of-flight cameras in computer graphics," Computer Graphics Forum 29(1), 141-159 (2010).

[9] K. Agusanto, L. Li, Z. Chuangui, and N. Sing, "Photorealistic rendering for augmented reality using environment illumination," Proc. Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 208-216 (2003).

[10] P. Debevec, "Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography," Proc. ACM SIGGRAPH 2008 classes, 189-198 (2008).

[11] K. Karsch, V. Hedau, D. Forsyth, and D. Hoiem, "Rendering synthetic objects into legacy photographs," Proc. 2011 SIGGRAPH Asia Conference, 157-169 (2011).

[12] J. Fischer, B. Huhle, and A. Schilling, "Using time-of-flight range data for occlusion handling in augmented reality," Proc. Eurographics Symposium on Virtual Environments, 109-116 (2007).

[13] T. Franke, S. Kahn, M. Olbrich, and Y. Jung, "Enhancing realism of mixed reality applications through real-time depth-imaging devices in x3d," Proc. 16th International Conference on 3D Web Technology, 71-79 (2011).

[14] R. Koch, I. Schiller, B. Bartczak, F. Kellner, and K. Koeser, "Mixin3d: 3d mixed reality with tof-camera," Proc. Dynamic 3D Imaging, 126-141 (2009).

[15] J. Frahm, K. Koeser, D. Grest, and R. Koch, "Markerless augmented reality with light source estimation for direct illumination," Proc. Conference on Visual Media Production, 211-220 (2005).

[16] F. de Sorbier, Y. Takaya, Y. Uematsu, I. Daribo, and H. Saito, "Augmented reality for 3d tv using depth camera input," Proc. 16th International Conference on Virtual Systems and Multimedia, 117-123 (2010).

[17] F. de Sorbier, Y. Uematsu, and H. Saito, “Depth camera to generate online content for auto-stereoscopic displays,” Proc. 20th International Conference on Artificial Reality and Telexistence, 184-188 (2010).

[18] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” Proc. Sixth International Conference on Computer Vision, 839-846 (1998).

[19] S. Olesen, S. Lyder, D. Kraft, N. Krüger, and J. Jessen, “Real-time extraction of surface patches with associated uncertainties by means of kinect cameras,” Journal of Real-Time Image Processing, 1–14 (2012).

[20] M. Vieira and K. Shimada, “Surface mesh segmentation and smooth surface extraction through region growing,” Computer aided geometric design 22(8), 771–792 (2005).

[21] S. Kwon, F. Bosche, C. Kim, C. Haas, and K. Liapi, “Fitting range data to primitives for rapid local 3d modeling using sparse range point clouds,” Automation in Construction 13(1), 67-81 (2004).

[22] P. Lensing and W. Broll, “Fusing the real and the virtual: A depth-camera based approach to Mixed Reality,” Proc. 10th IEEE International Symposium on Mixed and Augmented Reality, 261-262 (2011).

[23] P. Kán and H. Kaufmann, “Physically-based depth of field in augmented reality,” Proc. Eurographics 2012 Short Papers, 89-92 (2012).

[24] P. Kán and H. Kaufmann, “High-Quality Reflections, Refractions, and Caustics in Augmented Reality and their Contribution to Visual Coherence,” Proc. International Symposium on Mixed and Augmented Reality, 99-108 (2012).

[25] Z. Zhang, “Iterative point matching for registration of free-form curves and surfaces,” International journal of computer vision 13(2), 119-152 (1994).

[26] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” Proc. Third International Conference on 3-D Digital Imaging and Modeling, 145-152 (2001).

[27] B. Steder, R. Rusu, K. Konolige, and W. Burgard, “Narf: 3d range image features for object recognition,” Proc. International Conference on Intelligent Robots and Systems, 2010.

[28] R. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (fpfh) for 3d registration,” Proc. Robotics and Automation, 3212-3217 (2009).

[29] S. Holzer, R. Rusu, M. Dixon, S. Gedikli and N.Navab, “Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images,” Proc. International Conference on Intelligent Robots and Systems, 2684 -2689 (2012).

[30] D. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” International Journal of Computer Vision 60(2), 91-110 (2004).

[31] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” Proc. European Conference on Computer Vision Lecture Notes in Computer Science Volume 3951, 404-417 (2006).

[32] A. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Visual odometry and mapping for autonomous flight using an rgb-d camera,” Proc. International Symposium on Robotics Research, 647-663 (2011).

[33] P. J. Schneider and D. Eberly, [Geometric Tools for Computer Graphics], Elsevier Science Inc., (2002).

[34] K. Arun, T. Huang, and S. Blostein, “Least-squares fitting of two 3-d point sets,” Proc. IEEE Transactions on Pattern Analysis and Machine Intelligence (5), 698-700 (1987).

[35] D. Nowrouzezahrai, S. Geiger, K. Mitchell, R. Sumner, W. Jarosz, and M. Gross, “Light factorization for mixed-frequency shadows in augmented reality,” Proc. 10th IEEE International Symposium on Mixed and Augmented Reality, (2011).

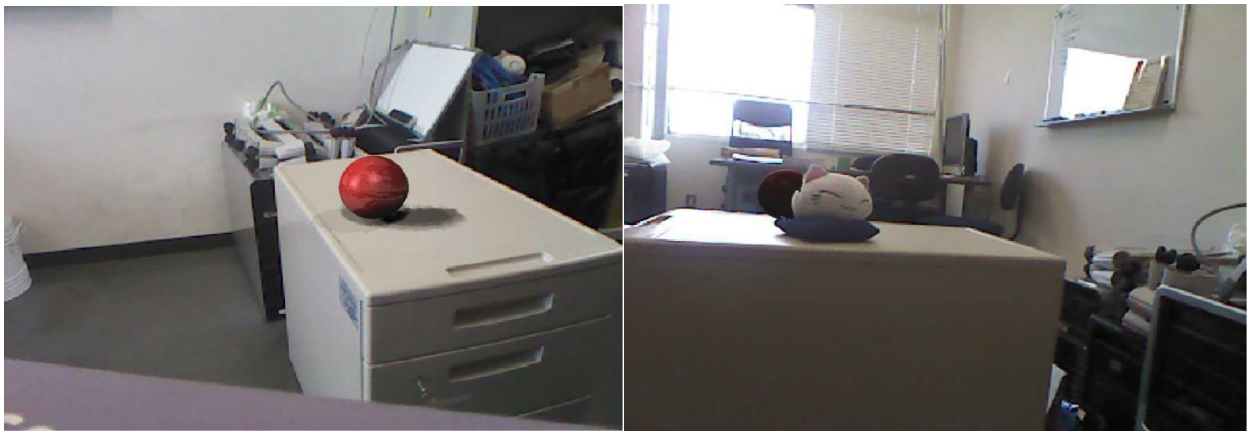
[36] A. Ikeda, Y. Oyamada, M. Sugimoto, and H. Saito, “Illumination estimation from shadow and incomplete object shape captured by an rgb-d camera,” Proc. 21st International Conference on Pattern Recognition, (2012).

[37] I. Sato, Y. Sato, and K. Ikeuchi, “Illumination from shadows,” Pattern Analysis and Machine Intelligence 25(3), 290-300 (2003).

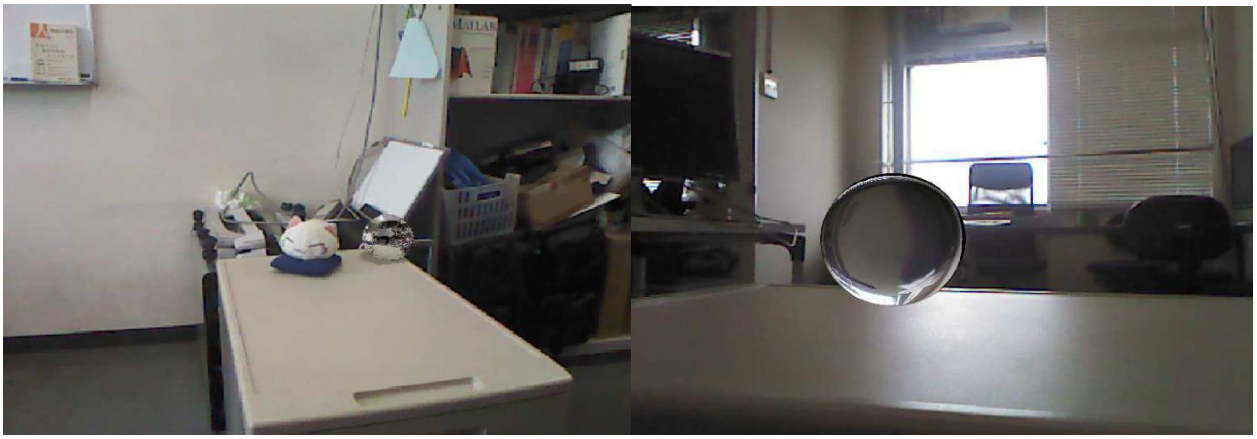
[38] L. Gruber, T Richter –Trummer, and Dieter Schmalstieg, “Real-time photometric registration from arbitrary geometry,” Proc. IEEE International Symposium on Mixed and Augmented Reality, 119-128 (2012).

Video1. Video demonstrating the steps of our approach. http://dx.doi.org/doi.number.goes.here

Examples of lambertian reflectance and shadows



Lambertian reflectance with shadows and reflections (left) and occlusions handling (right)



Mirror reflection (left) and refractions (right)

Figure 8. Extra results from our proposed approach.