Augmented Reality System using a Smartphone Based on Getting a 3D Environment Model in Real-Time with an RGB-D Camera

Toshihiro Honda, Francois de Sorbier, Hideo Saito
Graduate School of Science and Technology
Keio University, 3–14–1, Hiyoshi, Kouhoku–ku, Yokohama–shi, 223–0061, Japan
Email: {t-honda,fdesorbi,saito}@hvrl.ics.keio.ac.jp

***Abstract*—In this paper, we propose a system to achieve Augmented Reality (AR) on a smartphone. In this system, we assume a fixed RGB-D camera connected to a server is installed in the environment, and perform AR based on the 3D environment shape got by the RGB-D camera in real-time. On the smartphone side, it sends the image captured by its camera, and receives the output image processed to perform AR by the server. On the server side, it sends back the image that virtual objects are superimposed on. At this time, by constructing the 3D environment model in real-time using the RGB-D camera, it is possible to calculate the smartphone pose and the interaction between the virtual objects and the environment properly. It is also possible to change the lighting virtually. Our experiment revealed it is possible to superimpose virtual objects on the smartphone in approximately 8fps.**

I. Introduction

As one of the application technology of computer vision, augmented reality (AR) has been attracting attention [1]. AR is a technique to superimpose virtual objects as if they existed in the real environment. By using the AR technology, it is possible to add some information which cannot be obtained just by simply looking at the real environment be attached. In the past, the system connecting a web camera to a PC [2], and the system using a head mounted display [3] are often proposed. However, the AR system which can be achieved on a mobile device such as a smartphone has been gathering attention [4]. For example, there is the system to estimate the pose of a smartphone camera from planar textures and superimpose virtual objects corresponding to the texture [5], and to superimpose 3D building models on a paper map if it is captured by a smartphone [6]. However, these AR systems are often intended for a planar place.

In this paper, we propose the system to perform AR in even a rough environment by getting the 3D environment model using a RGB-D camera, computing the relative position between the RGB-D camera and a smartphone, and grasping the 3D environment shape from the smartphone viewpoint. We assume a Kinect, which is developed by Microsoft, as a RGB-D camera, is connected to a server and it is fixed. If the area captured by the Kinect is captured by the smartphone camera, the relative position between the Kinect and the smartphone can be computed by matching natural feature points. Then virtual objects are superimposed on the smartphone screen considering geometric consistency. To superimpose virtual objects stably, we track the smartphone camera using its previous pose. Moreover, since we know the 3D environment shape, it is possible to observe how the environment changes when a virtual light source is put.

II. Proposed System

Figure 1 shows the flow of our proposed method. We describe the processing performed by the smartphone. The processing which sends an image to the server connected wirelessly to the smartphone starts after the smartphone captures the image. When the processing finishes, the processing which receives an output image created by the server starts. After receiving the output image, the processing which displays it on the smartphone screen starts. Then the processing which sends a captured image to the server starts again.

We describe the processing performed by the server. First, we get a 3D environment model using a 3D point cloud obtained by the Kinect depth image. Because it takes much time to perform this processing with only CPU, we use GPU to reduce the processing time. Then the positions of virtual objects are computed considering to the gravity and the collision with the meshes using a physics engine. Unlike the case of making the 3D environment model in advance, it is possible for the users to interact with the virtual objects because the model is created in real-time. Here, by computing normal vectors of the model, it is also possible to observe how the environment changes when a virtual light source is put. Next, we extract natural features of the Kinect color image and the smartphone image and match those feature points. 2D–3D correspondences between the smartphone image coordinates and the Kinect color image coordinates can be computed because the 3D coordinates of the Kinect feature points can be obtained using the Kinect depth image. By this 2D–3D correspondences, the projection matrix which projects the points in the Kinect color camera coordinate system to the smartphone image coordinate system is computed. Because it is considered that the user's smartphone moves constantly, we track the smartphone camera using the projection matrix of the previous frame and compute the matrix of the current frame stably. Virtual objects are superimposed on the smartphone image using this matrix and the output image is created.

III. Getting a 3D environment model and computing positions of virtual objects

We can get a color image and a depth image using a Kinect. The viewpoint of the camera for capturing the color image

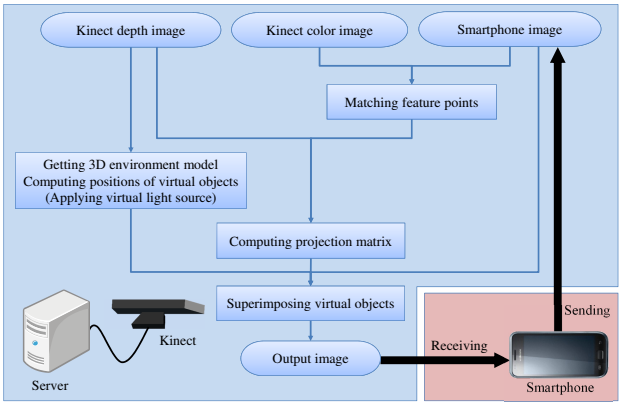


Fig. 1. Flow of the proposed method

and that for capturing the depth image are different, but we can match these viewpoints using a function included in the library called OpenNI[7].

A 3D environment model can be created by getting a 3D point cloud from the depth image and putting meshes on the point cloud. First, we describe the method to get the 3D point cloud whose origin is the position of the color camera from the depth image. Before the processing, we apply a bilateral filter[8] to the depth image to remove noise. It is possible to get the focal length $f\,[\mathrm{mm}]$ of the depth camera and the length $l\,[\mathrm{mm}]$ of the side of one pixel on the image plane at the focal length. Here, a coefficient a is defined as

$$a = \frac{l}{f} \tag{1}$$

The coordinate (x', y', z') whose origin is the position of the color camera can be calculated by

$$x'\,[\mathrm{m}] = a \times z \times x \tag{2}$$
$$y'\,[\mathrm{m}] = a \times z \times y \tag{3}$$
$$z'\,[\mathrm{m}] = z \tag{4}$$

where, (x, y) is a certain coordinate whose origin is the center of the depth image and $z\,[\mathrm{m}]$ is the depth value. It is possible to obtain the 3D point cloud by performing this processing for all pixels of the depth image. The pixels which not store the depth value are calculated as the depth value is $2\,\mathrm{m}$ for convenience. The processing for getting the 3D point cloud is speeded up by using GPU.

Next, we describe the method to put meshes. As shown in figure 2, we put triangle meshes by choosing 3 points regularly as

$$\{(x, y), (x+1, y), (x, y+1)\},$$
$$\{(x+1, y), (x, y+1), (x+1, y+1)\},$$
$$\{(x+1, y), (x+2, y), (x+1, y+1)\},$$
$$\ldots$$

where, (x, y) is a certain coordinate of the depth image. Meshes can be put fast because the processing is simple. This processing is speeded up to use the meshes of the previous frame as they are in the area where the positions of the meshes are not changed.

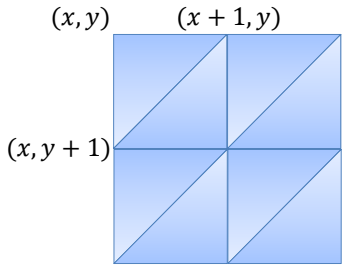


Fig. 2. Creating meshes

Virtual objects can be superimposed considering geometric consistency if the 3D environment model can be created because the collision detection between the virtual objects and the 3D environment model can be performed. Physics such as this collision detection and the gravity is performed using physics engine called Bullet [9].

It is possible to compute the position of the virtual objects in the world coordinate system whose origin is the color camera of the Kinect from the above processing.

IV. Superimposing Virtual Objects on a Smartphone Image

We have to compute where virtual objects in the world coordinate system correspond in the smartphone image coordinate system. In this system, the projection matrix which projects the points in the world coordinate system to those in the smartphone image coordinate system is computed. In this section, we describe the method to compute this projection matrix and superimpose virtual objects on a smartphone image.

A. Matching feature points

We match the pixels which represent the same point between a Kinect color image and a smartphone image to grasp the positional relationship between the Kinect and the smartphone. Matching is performed automatically using natural features. We use BRISK (Binary Robust Invariant Scalable Keypoints) [10] for the method to extract natural features. First, natural feature points are detected and describing what feature they have. Then, descriptors of the Kinect side and the smartphone side are compared and the most similar feature points are matched. The shorter the hamming distance is, the more similar descriptors the feature points have because the BRISK descriptor is represented as the binary format.

B. Computing projection matrix

It is possible to get the 3D coordinates of the feature points of the Kinect side corresponded to the feature points of the smartphone side because the alignment between the Kinect depth image and the Kinect color image is performed. Therefore, 2D–3D correspondences between the smartphone image coordinates and the Kinect color camera coordinates are obtained. The rotation matrix $\mathbf{R}$ and the translation vector $\boldsymbol{t}$ which convert the world coordinate system to the smartphone camera coordinate system can be computed using the library called OpenCV [11] from 6 or more corresponding points and the intrinsic parameters $\mathbf{A}$ of the smartphone camera which are measured in advance. Specifically, corresponding points

which are used to compute $\mathbf{R}, \boldsymbol{t}$ are selected by RANSAC (Random Sample Consensus) [12] to compute $\mathbf{R}, \boldsymbol{t}$ robustly if there are false corresponding points. Then, $\mathbf{R}, \boldsymbol{t}$ are computed by DLT (Direct Linear Transformation) method [13] using the corresponding points. The projection matrix $\mathbf{P}$ can be computed by

$$\mathbf{P} = \mathbf{A}\,(\mathbf{R}|\boldsymbol{t}) \tag{5}$$

Virtual objects in the world coordinate system can be projected to the smartphone image coordinate using this $\mathbf{P}$. This processing is performed by the library called OpenGL [14]. It is possible to superimpose virtual objects considering the occlusion because OpenGL can draw polygons using Z–buffer (memory area for storing the distance between the camera and the object).

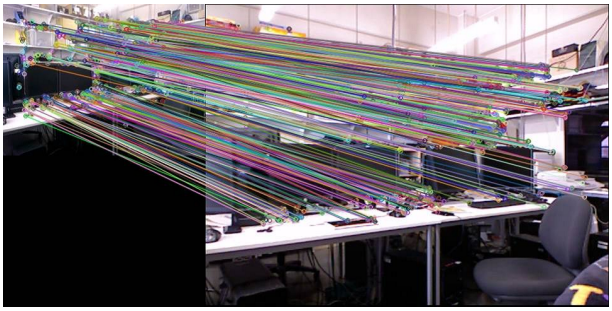
C. Tracking smartphone camera

In this system, the smartphone camera is tracked to compute the projection matrix $\mathbf{P}$ stably. The processing is divided into two cases, one is the key frame which is the starting point of the tracking and the other is the frame except for the key frame.

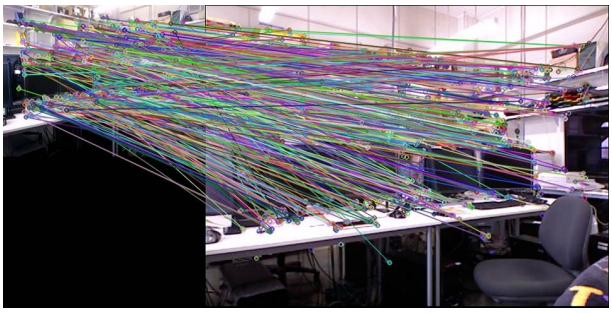
We describe the processing of the key frame. First, the 3D coordinates of the feature points and the descriptors are saved after extracting BRISK features from the Kinect color image. This is because it is believed that the descriptors of the Kinect which are extracted in the key frame can be used as they are in the frame except for the key frame because the Kinect is fixed. Then, the calculation of the desctiptor distance between one of the feature points of the Kinect side and all of the feature points of the smartphone side is performed, and the Kinect side point is matched to the smartphone side point whose distance is the shortest. $\mathbf{P}$ is computed by the corresponding points.

We describe the processing of the frame except for the key frame. First, the features of the smartphone image are extracted. We make a look-up table which store the number of the feature point in the pixel which has a feature point and -1 in the pixel which has no feature point in order to speed up the processing. Next, the feature points of the Kinect side which are saved before are projected to the smartphone image coordinates by $\mathbf{P}$ of the previous frame. The calculation of the descriptor distance is performed only to the feature points of the smartphone side surrounding the projected coordinate because it is believed that the smartphone camera does not move too much during one frame. In this system, we confirmed by experiments that $\mathbf{P}$ can be computed correctly by performing the calculation of the descriptor distance to the feature points in the range of 61 pixels square whose center is the projected point. We use 512 bits binary format descriptor, and the pair whose hamming distance is 200 or more is removed.

Figure 3 shows the state of the matching in the case of tracking a smartphone camera and not. The left side is a smartphone image and the right side is a Kinect color image, and corresponding points are connected by a line. As shown in figure 3, there are more correct pairs in the case of tracking a smartphone camera. We calculated the average number of the pairs which are actually used to compute $\mathbf{P}$ for some 10 frames when we applied RANSAC to 50 pairs whose hamming distance is short. It was 44.2 pairs in the case of tracking the smartphone camera, whereas it was 32.7 pairs in the case of not tracking. Therefore, it was shown quantitatively that tracking smartphone camera is effective.



(a) Tracking smartphone camera



(b) Not tracking smartphone camera

Fig. 3. Comparison of matching in the case of tracking smartphone camera and not

D. Evaluation of reprojection error

After computing a projection matrix $\mathbf{P}$, the evaluation of reprojection error of saved feature points of the Kinect side is performed using this $\mathbf{P}$. In this system, each feature point of the Kinect side keeps 5 BRISK descriptors and the value which judges whether to be used for computing $\mathbf{P}$ in the next frame or not. The evaluation of reprojection error updates these parameters.

First, all of the feature points of the Kinect side are projected to the smartphone image coordinates by $\mathbf{P}$. The feature point of the smartphone side located in the nearest neighbor of each projected point is searched. The nearest neighbor means not the descriptor distance, but the Euclidean distance between the feature points in the image space. The reprojection error $E\,[\mathrm{pixels}]$ is the Euclidean distance to the nearest neighbor point. The feature point of the Kinect side is judged as an outlier if E is more than the threshold $E_{th}\,[\mathrm{pixels}]$, and not used for computing $\mathbf{P}$ in the next frame. We define $E_{th} = 5.0$ in this system. The feature point judged an outlier continuously for 4 or more frames is removed and not used for computing $\mathbf{P}$ any more. The feature points of the Kinect side are updated with the BRISK descriptors of the nearest neighbor points using FIFO (First In, First Out) method if $\mathbf{P}$ is judged to be accurate according to the result of the evaluation of reprojection error and a fixed frame has passed since the feature points of the Kinect side were updated. This update enables the fixed Kinect to keep the BRISK descriptors of multiple viewpoints in a pseudo manner, and it can be expected to improve the matching accuracy. $\mathbf{P}$ is judged to be accurate if the number of $E < 1.0$ pairs is 50 or more,

and the update is performed after 10 or more frames since the previous update.

V. Sending and Receiving an Image between a Smartphone and a Server

Figure 4 shows the flow of processing in a smartphone and a server. There are the thread which receives an image from the smartphone and sends an image to the smartphone and the thread which performs AR in the server, and each thread runs independently. In the thread which sends and receives an image, the function to receive an image is called first, and then the thread enters standby state. The image is received when the function to send an image is called, and an input image is updated. At this time, the updated input image is used in the next frame if the thread which performs AR is in progress. An output image is sent if the function to send an image is called in the server side and the function to receive an image is called in the smartphone side. At this time, the output image of the previous frame is sent if the thread which performs AR is in progress. We calculated the 10 frames average time from calling the function to send and receive an image to actually finishing sending and receiving the image for each of the smartphone and the server. The experimental platform was implemented on 1.0GHz MSM8255 smartphone with 512MB RAM and 2.5GHz Intel Core i7-2860QM and GeForce GTX 560M laptop with 16.0GB RAM. The smartphone image size is 320×240pixels. Table I shows the result. The time from starting to receive image data to finishing receiving them was 15.5ms for the smartphone and 14.3ms for the server. Therefore, as shown in Figure 4, standby time occurs before receiving an image for both the server and the smartphone. In other words, the receiving function of the server is called before the sending function of the smartphone and the sending function of the server is called after the receiving function of the smartphone.

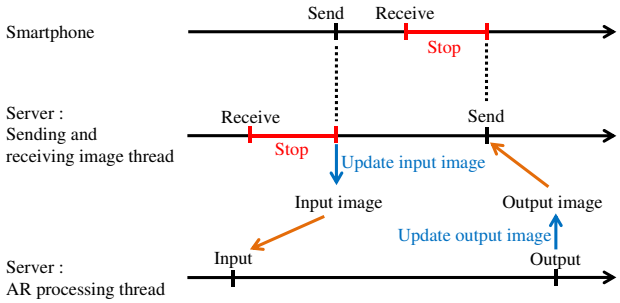


Fig. 4. Flow of processing of smartphone and server

TABLE I. Time to Send and Receive an Image for Smartphone and Server [ms]

	Receiving	Sending
Smartphone	33.4	1.2
Server	100.2	3.0

VI. Experiments

Figure 5 shows the experiment environment. A fixed Kinect is connected to a server and a user has a smartphone. The experimental platform was the same as section V. The smartphone image size is 320×240pixels and the Kinect color and depth image size are 640×480pixels.

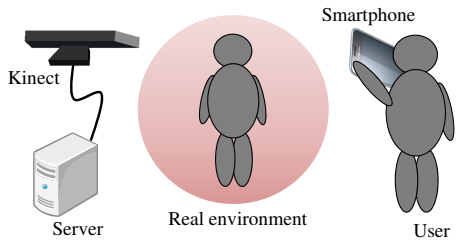


Fig. 5. Experimental environment

Figure 6 shows the result in the case of dropping virtual spheres from above and the case of installing a virtual white light source. The virtual light source is located at the 5m left from the user.

(a), (b) show it is possible to superimpose the virtual objects at the correct positions considering the 3D shape of the real environment. (d) shows the back side of the human in the image is bright, and the ventral side is dark. Therefore, it can be said that it is possible to compute the effect of the virtual light source considering the 3D shape. However, the positions of the virtual objects are still not stable when viewed as a movie although tracking the smartphone camera improves the stability. The method to compute the relative positional relationship between the Kinect and the smartphone accurately except for tracking the smartphone camera is needed. In addition, the surface is rough when the virtual light source is applied. This is because the depth image is coarse, so it is needed to improve. Table II shows the average time spent in the main processing for 10 frames. The frame rate is approximately 8 fps for superimposing virtual objects and 2 fps for applying a virtual light source. The time from taking an image with the smartphone to being displayed on the smartphone via the AR processing is about 170ms–350ms.

TABLE II. Time Spent in the Main Processing [ms]

Processing	Time
Create a 3D environment model	51.2
Compute positions of virtual objects	<1
Extract features of the Kinect side	31.3
Extract features of the smartphone side	6.5
Match feature points	13.8
Compute projection matrix	47.7
Evaluation of reprojection error	<1

VII. Conclusion

In this paper, we proposed the system to achieve AR on a smartphone in even a rough environment by getting a 3D environment model using a Kinect and computing the relative positional relationship between the Kinect and the smartphone by matching feature points.

As for our future work, we are planning to make the method to compute the relative positional relationship accurately. For example, use sensor information such as a gyro sensor and an acceleration sensor inside the smartphone, or track smartphone itself by capturing it with the camera

(a) Input image

(b) Output image which virtual objects are superimposed on

(c) Input image

(d) Output image which a virtual light source is applied to

Fig. 6. Result of the experiments

calibrated to the Kinect. In addition, we are also planning to improve a depth image. For example, approximate the area which can be approximated as a plane as the plane.

REFERENCES

[1] D. Van Krevelen, and R. Poelman, "A survey of augmented reality technologies, applications and limitations", International Journal of Virtual Reality, vol. 9, no. 2, pp. 1–20, 2010.

[2] J. Fischer, M. Neff, D. Freudenstein, and D. Bartz, "Medical Augmented Reality based on Commercial Image Guided Surgery", Proceedings of the 10th Eurographics Symposium on Virtual Environments, pp. 83–86, 2004.

[3] H. Kato, and M. Billinghurst, "Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System", Proceedings of the 2nd International Workshop on Augmented Reality, pp. 85–94, 1999.

[4] M. Gervautz, and D. Schmalstieg, "Anywhere Interfaces Using Handheld Augmented Reality", Computer, vol. 45, no. 7, pp. 26–31, 2012.

[5] N. Hagbi, O. Bergig, J. El-Sana, and M. Billinghurst, "Shape recognition and pose estimation for mobile augmented reality", IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 10, pp. 1369–1379, 2011.

[6] S. Martedi, and H. Saito, "Augmented Fly-through using Shared Geographical Data", International Conference on Artificial Reality and Teleexistence, pp. 47–52, 2011.

[7] Open Natural Interaction Library, http://www.openni.org/

[8] C. Tomasi, and R. Manduchi, "Bilateral Filtering for Gray and Color Images", International Conference on Computer Vision, pp. 839–846, 1998.

[9] Bullet Physics Library, http://bulletphysics.org/wordpress/

[10] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary Robust Invariant Scalable Keypoints", International Conference on Computer Vision, pp. 2548–2555, 2011.

[11] Open Computer Vision Library, http://sourceforge.net/projects/opencvlibrary/

[12] M.A. Fischler, and R.C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", Communications of the ACM, vol. 24, pp. 381–395, 1981.

[13] R. Hartley and A. Zisserman, "Multiple view geometry in computer vision 2nd edition", Cambridge University Press, pp. 88–93, 2003.

[14] Open Graphics Library, http://www.opengl.org/