A PHOTO-BASED AUGMENTED REALITY SYSTEM WITH LOW COMPUTATIONAL COMPLEXITY

Masaya Ohta[1], Ryuta Yokomichi[1], Masataka Motokurumada[1] and Katsumi Yamashita[1]

[1] Graduate School of Engineering, Osaka Prefecture University, Osaka, Japan

ABSTRACT

In this report, we propose an augmented reality (AR) system using photo images. Based on the position and orientation of the user's camera, the system selects the most appropriate image of an object from a set of photos taken at various angles and "pastes" it into the camera image at an appropriate location and scale. Since the technique requires no 3D model creation or 3DCG rendering during the AR processes, it is well suited to mobile devices with limited power and rendering capacity. The proposed system is also installed and evaluated on a practical device.

KEYWORDS

Augmented reality, Photo image, Low Computational Complexity.

1. INTRODUCTION

Augmented reality (AR) is a set of technologies by which a user's view of the real world is augmented with additional information from a computer or a mobile device. AR has been applied in a number of fields, including entertainment, medicine, manufacturing, repair, and training. Due to the recent spike in interest in AR, the need for simple, high-performance frameworks for developing AR systems has also risen sharply [1,2,3,4,5,6,7].

A typical AR system uses 3DCG to render the objects required for a given user's viewpoint. These objects are typically authored using 3D modeling software, such as AutoCAD, 3D Studio Max, Maya, Lightwave 3D, and Cinema4D. While highly expressive, these tools also tend to be complex and time-consuming. As a result, 3D reconstruction methods have been proposed [8,9,10] as an alternative to author-driven modeling. These methods attempt to reconstruct a 3D model, automatically or semi-automatically, based on photo images of a target object.

One of the goals of this study is to apply AR technology to the display of products on electronic commerce (EC) sites. AR using a marker is well suited to this purpose because it can render a product on screen at the correct size relative to the marker. There have been a number of studies on the application of AR in EC [11,12,13] and on the related notion of presenting photorealistic 3D objects through AR [14,15]. Also noteworthy is Object VR [16], which allows the user to drag the mouse to view a product from several different angles. However, Object VR is limited to an isolated review of a product, whereas AR can be used to express not only a given product but also its possible situation within the real environment surrounding the user. This gives the user a better sense of the object's scale and compatibility with his/her belongings. The main obstacle encountered in applying AR to EC sites in this way is that 3D models of all viewed products must be prepared in advance. If the number of viewed product is large, the cost of creating these models will be prohibitive in most cases.

This paper proposes a photo-based AR system that uses photo images taken at various angles around an object instead of rendering a 3D model corresponding to the object, at runtime. With this system, a photo image appropriate to the position and orientation of user's camera is selected from among images captured and stored ahead of time, then adjusted and simply “pasted” onto the camera view. This approach can significantly reduce the amount of rendering calculation, as it requires no 3DCG rendering. It also spares developers the effort of creating a 3D model for every object on display. Note that the system can be applied not only to photographed objects but also to pre-renderings of 3D models. This too helps to minimize the cost of rendering 3D on mobile devices with limited battery capacities, such as smart phones. Once selected, the system mounts the selected image using HTML5/JavaScript. These technologies were chosen for their widespread compatibility and applicability to internet-based resources.

Since the proposed system uses photo images taken in advance, it is generally not possible to accurately render the object as it would appear to the user’s point of view, neither is it possible to show variations due to lighting conditions (color, positions, and number of light sources) in the user’s environment. For prototyping purposes, we will assume that the distance between the marker and the object is between 20 cm and 60 cm, and that the lighting environment is composed of white light similar to what is found in offices and living rooms. These assumptions are generally appropriate for an AR application to be used with product display on EC sites.

Section 2 will detail the proposed system; Section 3 will provide an evaluation of its performance; and Section 4 we will summarize our findings.

2. PROPOSED SYSTEM

2.1. Imaging System

Fig. 1 shows the imaging system used in this study. Using this system, the object to be photographed is placed on a rotating base and is shot at a number of preset angles using a web camera. These initial images are stored in video format. To remove background from the images, blue-screen panels are set up around the object so their color can be cleared from the resulting images using video editing software. The final images are stored in png format file with an α channel.

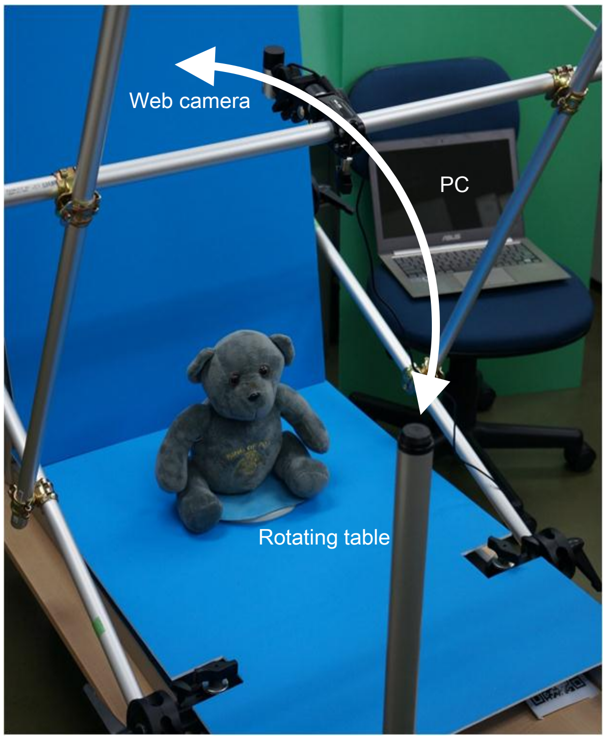


Fig. 1. Imaging system.

2.2. AR System

The AR system proposed in this paper consists of a marker detector and an AR renderer. The system is illustrated in Fig. 2. The marker detector recognizes a marker and estimates the position and orientation of the user's camera with respect to it. The FLARToolkit [17] provides us with marker detection functions. We modified the FLARToolkit by removing the default OpenGL renderer so that the results of marker detection are transmitted directly to our AR renderer (detailed below).

We also integrated the marker detection functions in ZXing [18], an open-source, QR-code scanning library implemented in Java, Objective-C, and ActionScript. With our modifications, the QR-code scanner assumes the position and orientation of the user’s camera based on the coordinates of the four corners of the QR-code, and transmits the result to the AR renderer.

The AR renderer in our system is implemented in HTML5/Javascript and runs on an HTML rendering engine. Our renderer receives input from the marker detector, selects the suitable photo image from a set of images captured and stored ahead of time, and draws the image into the rendering context. The following sections describe the specific image election method and the rendering method.

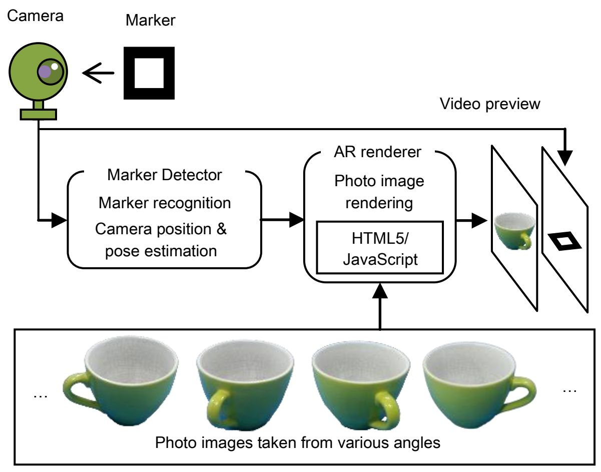


Fig. 2. AR system.

2.3. Image Selection

Fig. 3 (a) defines the marker coordinate system and the camera coordinate system. The marker detector calculates the conversion parameters $R_{3\times3}$ and $\mathbf{T}_c$ for the marker coordinate system $\mathbf{X}_m = (X_m, Y_m, Z_m)^T$ (T is transposed) and the camera coordinate system $\mathbf{X}_c = (X_c, Y_c, Z_c)^T$ based on the marker image captured by the camera. The specific calculation method is provided elsewhere in the literature [1].

The coordinate conversion between the marker coordinate system and the camera coordinate system is expressed by the following equation:

$$\mathbf{X}_c = R_{3\times3}\mathbf{X}_m + \mathbf{T}_c$$

Thus, the camera position $\mathbf{C}_m = (C_{mx}, C_{my}, C_{mz})^T$ within the marker coordinate system can be calculated by substituting the camera position $\mathbf{O}_c = (0,0,0)^T$ within the camera coordinate system in this equation:

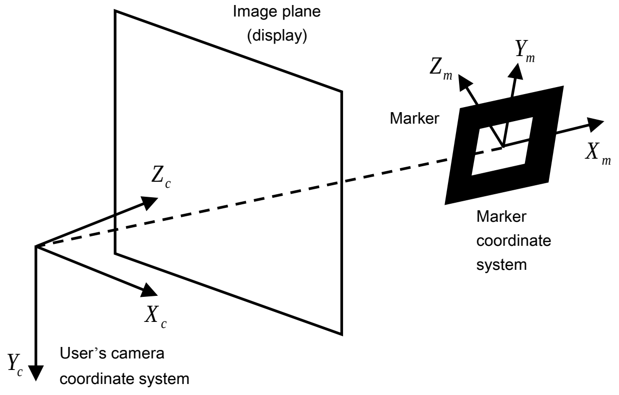
$$\mathbf{C}_m = R_{3\times3}^{-1}(\mathbf{O}_c - \mathbf{T}_c) = -R_{3\times3}^{-1}\mathbf{T}_c$$

Next, the center of the object within in the marker coordinate system is set as $(0,0,\varepsilon)^T$. As shown in Fig. 3 (b), the angle formed by the line-of-sight from the camera to the object center and the $+Z$ axis of the marker coordinate system is specified as depression ϕ, and the angle formed by the straight line as the projection of this line-of-sight onto the xy plane of the marker coordinate system and the y axis is specified as azimuth ψ. ϕ and ψ are calculated as follows:

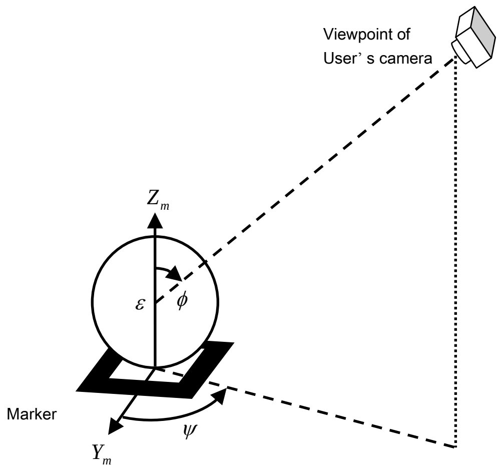
$$\phi = \tan^{-1}\left(\sqrt{C_{mx}^2 + C_{my}^2 + C_{mz}^2} \big/ \left|C_{mz} - \varepsilon\right|\right)$$

$$\psi = -\tan^{-1}\left(C_{mx} / C_{my}\right)$$

With the proposed method, the image shot from a position closest to both ϕ and ψ , as calculated above, is selected. If depression ϕ is small enough, the image shot from directly above is selected.



(a) Coordinate system of the AR system



(b) Depression and azimuth

Fig. 3. Image selection.

2.4. AR Rendering

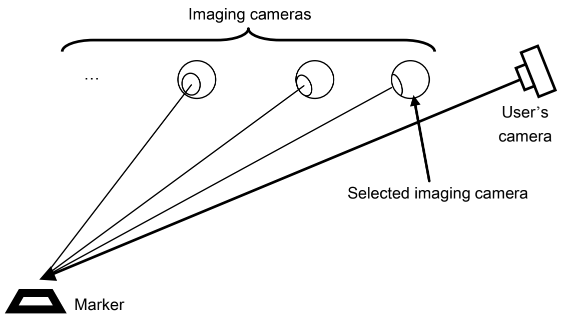
Only when the focal lengths of the camera for imaging and the user's camera are identical' and the position and the posture of the user's camera perfectly matches that of imaging, can our AR system use a selected photo without deformation. In all other cases, the exact appearance cannot be reconstructed by simple pasting.

Thus, we considered using a 3DCG rendering library as the first rendering method of our system Fig. 4 provides an outline of this method.

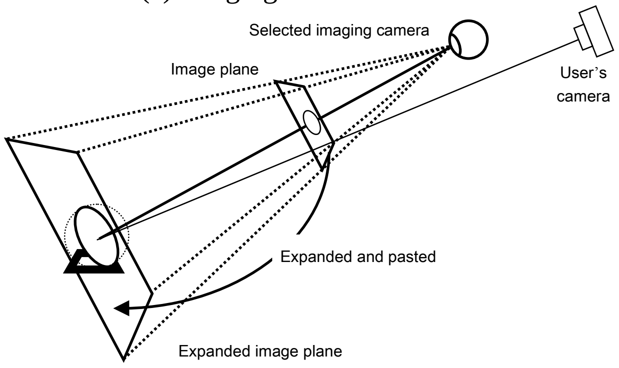
As shown in Fig. 4 (a), the imaging camera position for the photo image selected using the above method is calculated using the marker coordinate system. Next, the virtual plane that is parallel to the image plane of the imaging camera and that passes through the center of the object is considered, as shown in Fig. 4 (b). This plane is called the expanded image plane, as the selected photo image is expanded (i.e., scaled) and pasted onto its surface. Finally, as shown in Fig. 4 (c), the pixel is rendered at the point where the straight line connecting the viewpoint of user's camera at AR render time and each pixel on the expanded image plane crosses the image plane for the user's camera (corresponding to the display). This enables presentation of an image that feels “mostly right” to the user, despite mismatches in focal length or optical axis between the imaging camera and the user’s camera, as well as variation in distance to the marker.

A 3DCG rendering library can be used to render the image in this way. The marker coordinate system and the coordinate system for the rendering space of the library are matched, as are the position and orientation of the user’s camera and the library camera. Next, the expanded image plane is set up in this space and the selected image is applied to it as a texture. When the CG image is generated by the library’s rendering API after this process, an image equivalent to proposed method 1 is obtained. For our 3DCG library, we selected Three.js [19], an open-source library that performs 3DCG rendering from JavaScript.

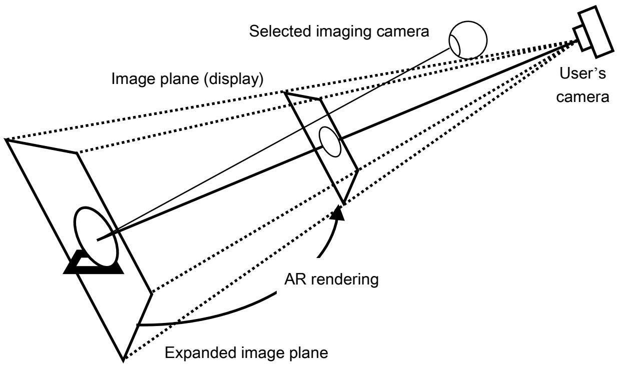
The first method incurs a calculation cost for rendering the virtual plane on which the texture is pasted using the 3DCG rendering library. To further reduce this calculation cost, we considered a second method to calculate the enlargement ratio and the rotation angle for the photo image, based on the position and orientation of the camera. Using this method, we could paste the image directly on the background and thereby avoid the overhead of a 3DCG rendering library. Although this introduces the cost of calculating an enlargement ratio and rotation angle for the pasted image, this new cost is far lower than that incurred by calculating the virtual plane and mapping the image as a texture.



(a) Imaging camera selection



(b) Expanded image plane



(c) AR rendering

Fig. 4. Rendering method 1 using 3DCG rendering library.

3. PERFORMANCE EVALUATION

To evaluate our system, we used a set of 721 photo images of 720 720 pixels each. This set comprises one photo shot from directly above the object, and photos shot from 10°, 20°, …, 60° depressions for every 3° azimuth, yielding a total of 120. The PC for this evaluation was an Intel Core-i5 3.00 GHz/8.00 GB RAM running Windows 7 Pro 64-bit. Our browser was Firefox 14.0.1 with an Adobe Flash Player 11.3 plug-in. All of the photo images were stored on the local HDD of the PC.

Fig. 5 shows screen dumps of the browser that loads the conventional FLARToolkit and the proposed system based on the FLARToolkit. We were able to verify that a suitable photo image corresponding to a position and orientation of the user's camera is selected, and rendered smoothly without dropped frames. It was also impossible to confirm the difference between proposed methods 1 and 2 by visual observation. Fig. 6 shows an example of AR rendering of food using the proposed method. Since there is no need to generate a 3D model as in the conventional method, we could confirm that rendering occurred smoothly regardless of the complexity of the object. Fig. 7 shows an example of mounting the AR system using the ZXing-based QR-code as our marker, for iOS- and Android-based mobile devices. We confirmed that this too operated smoothly on both of the major mobile platforms.

Next, we compared the performance between the conventional FLARToolkit using 3D models and proposed methods 1 and 2 using our modifications to FLARToolkit. In this experiment, AR rendering was tested by placing the marker on a rotating base that made a full rotation in 20 s and shooting this marker with a camera whose depression was fixed. The time it took to AR-render each frame was measured using FireBug starting immediately after browser start-up and ending 30 s later. The 3D model used in FLARToolkit was a texture image with 208 vertices, 228 faces, and a size of 75 KB. There were 721 images shot by the proposed method. We compared two methods of loading these images into memory: (1) up-front batch loading, and (2) lazy loading. The results of this comparison are presented in Fig. 8.

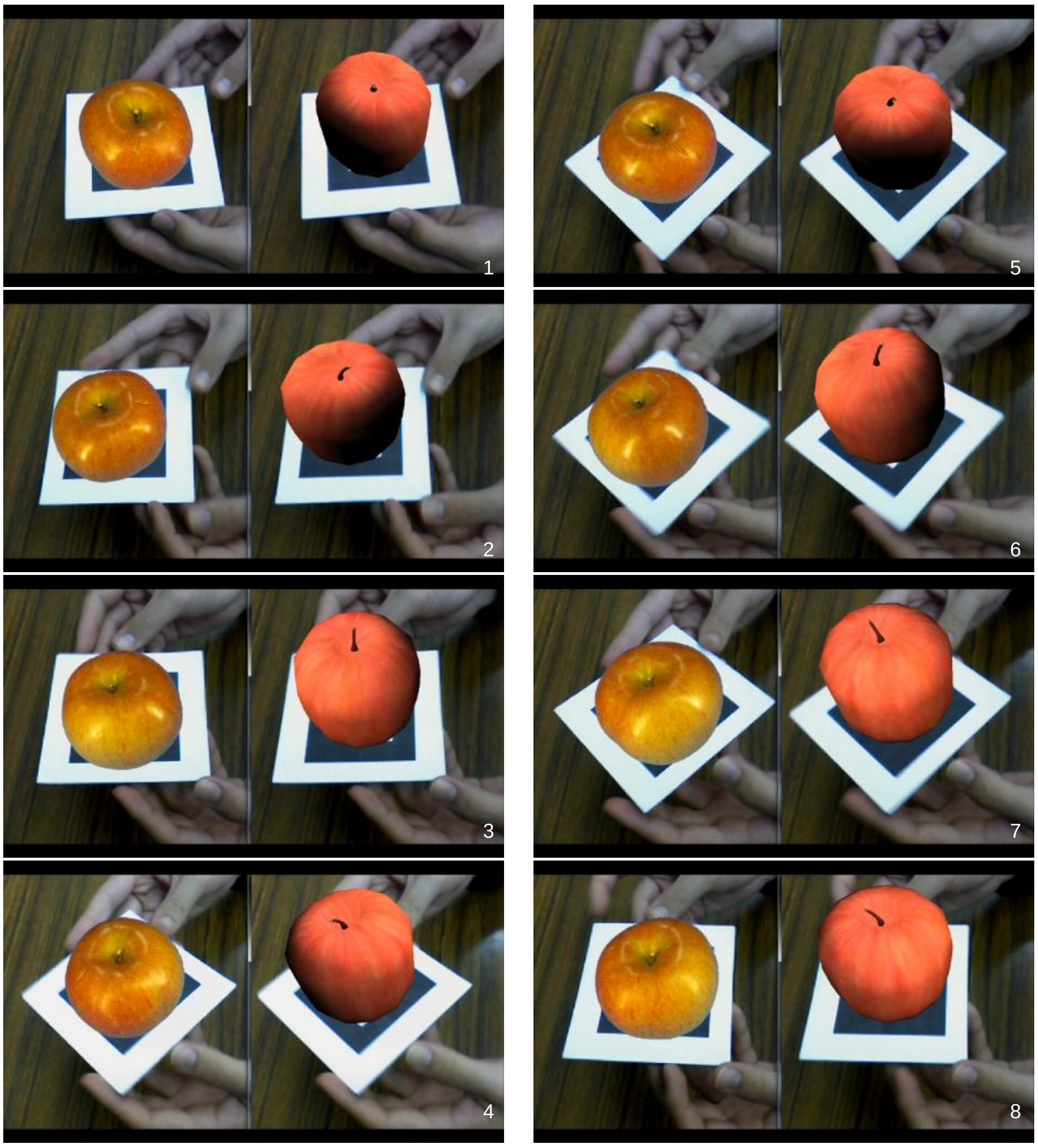


Fig. 5. Screen dumps of the conventional and proposed system.

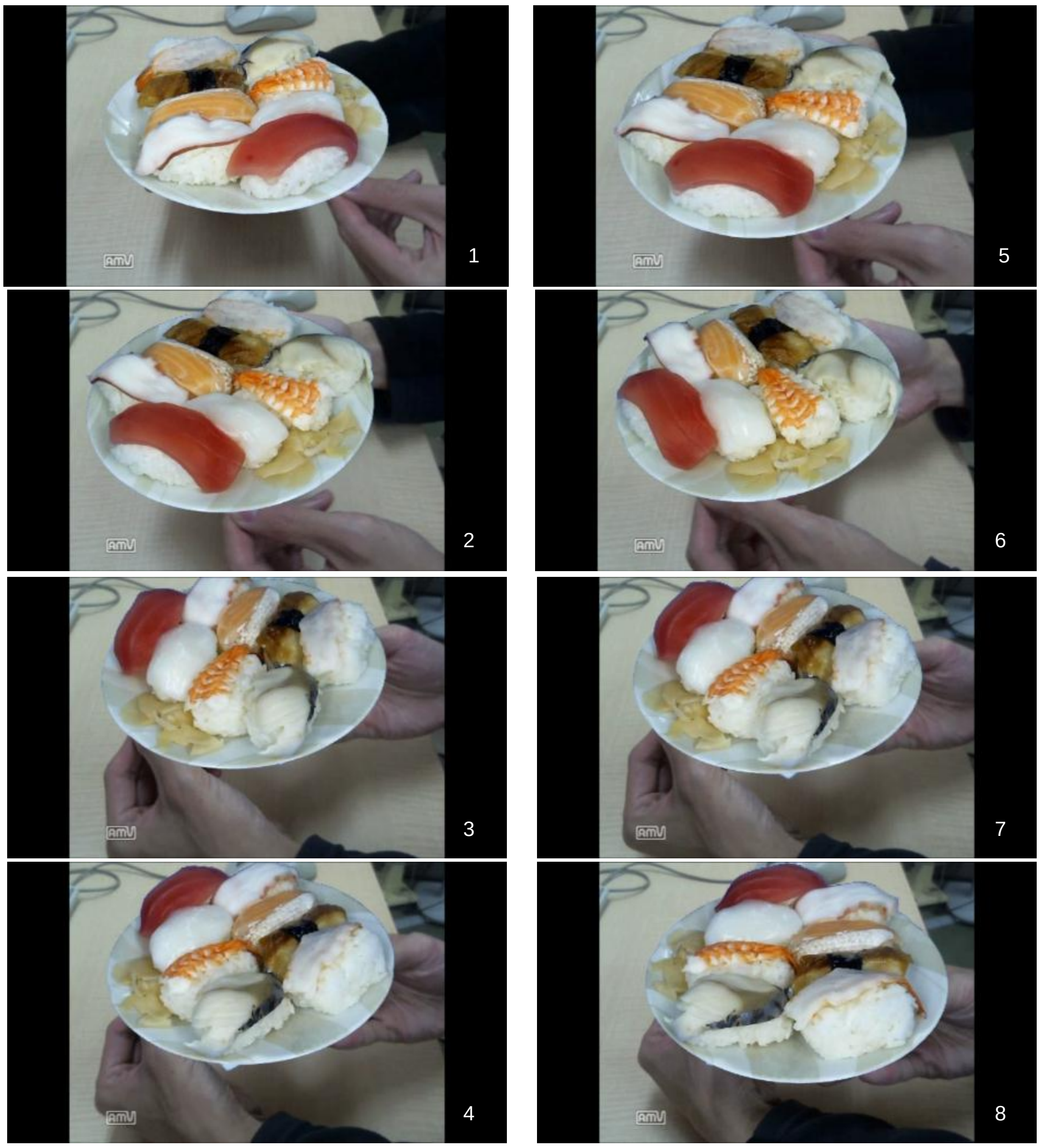
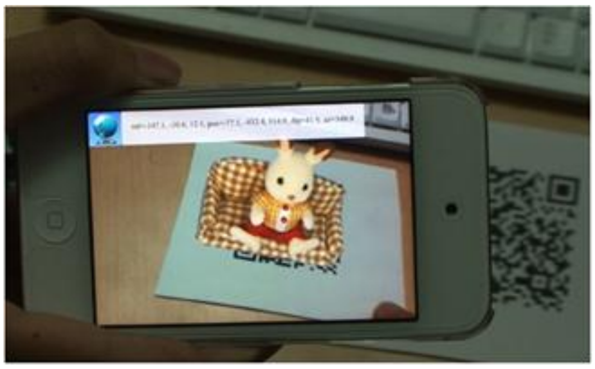
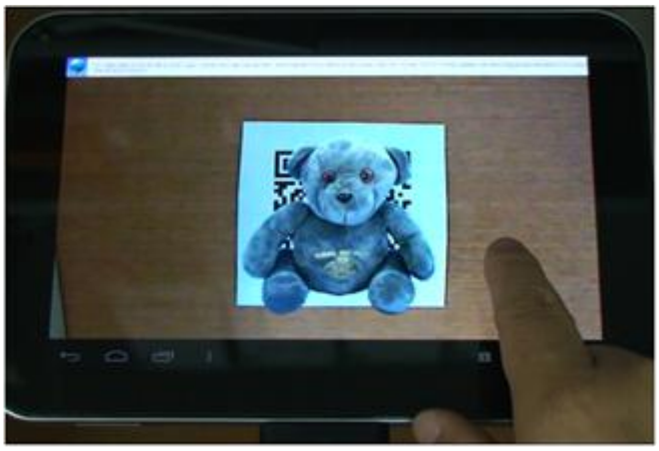


Fig. 6. AR rendering sample (sushi).



(a) iPod Touch (iOS 5.1)



(b)REGZA Tablet AT570 (Android 4.0)

Fig. 7. Proposed system based on ZXing implemented on iOS and Android.

Since the browser requires time to load data and images before AR rendering can begin, no graph starts from time 0. The rendering time for each frame was nearly constant with the conventional method as CG is rendered for each frame during the observation period. However, the rendering time became shorter after 20 s for the proposed methods. This is because the time to load and/or cache the images was included in the rendering time in this measurement. Since there was no more need to load a new image after the rotating base made a full turn, the time to load or cache was thereafter eliminated and the rendering time became shorter. Rendering started the latest using the batch loading method, because all images had to be loaded into memory beforehand. However, as expected, rendering time was shorter for batch loading than for lazy loading once rendering began.

On the other hand, the lazy loading required the shortest time until rendering began, indicating that the amount of data required in the beginning was the smallest. Proposed method 2 performed rendering in approximately 10% the time it took for proposed method 1, achieving a dramatic reduction in calculation cost even compared to the conventional method with the equivalent speed addressed until 20 s after start-up and the rendering time thereafter being about 20% that of conventional method. This is because rendering is completed by simply pasting the photo image on the display (using HTML5 canvas). Since the calculation cost does not vary with the proposed method, even when the rendered object becomes more complex, its advantage increases as object complexity grows.

We also measured the total number of images, memory usage, and start-up time using proposed method 2. The number of photo images used was 91, 181, 361, 541, and 721. Fig. 9 shows the memory consumption and start-up time from booting up a browser to rendering the first frame.

The proposed batch system consumes large amounts of memory, and takes longer to start-up compared to the lazy-loaded variation and to the system using conventional rendering methods. In the lazy-loaded system, memory consumption was only 10% higher than the conventional system and the start-up time remained constant regardless of the total number of images.

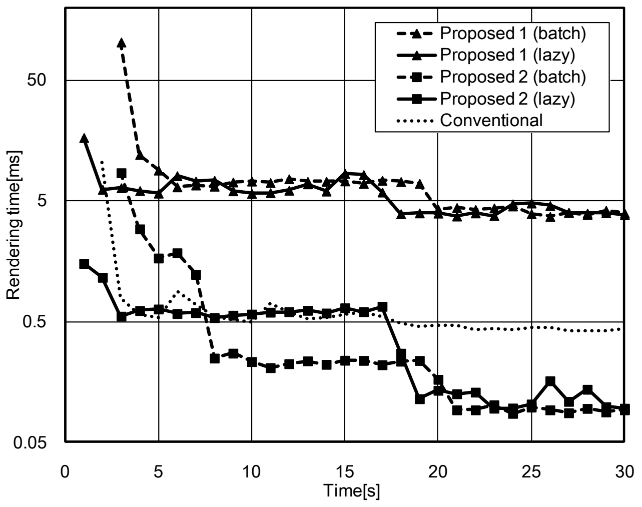
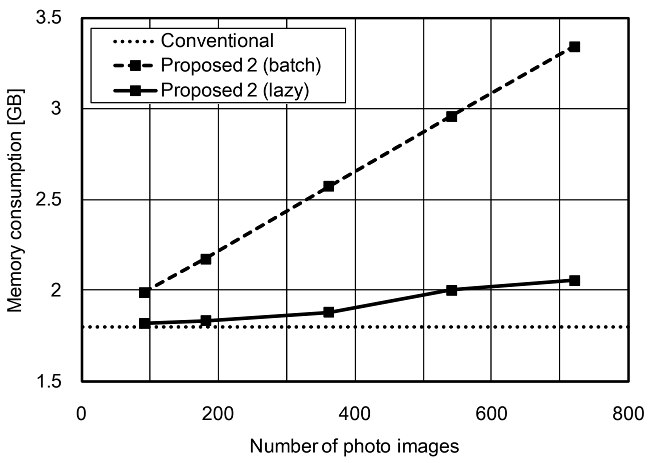
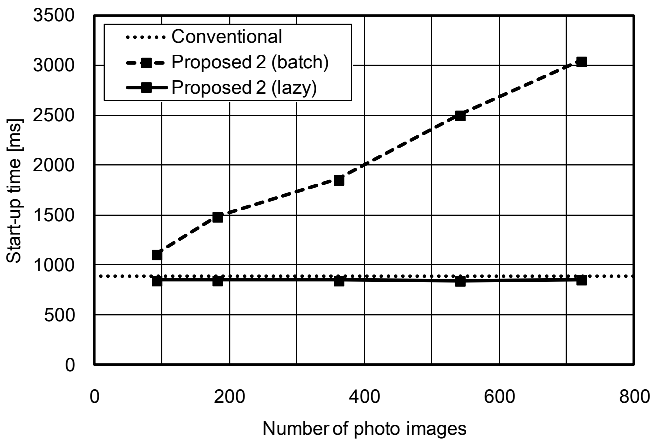


Fig. 8. Comparison of rendering times.



(a)Memory consumption



(b)Start-up time

Fig. 9. Performance comparison.

4. CONCLUSIONS

In this paper, we proposed a photo-based AR system that uses images taken in advance from various angles around an object. Our system simply pastes a selected photo image on the camera view instead of rendering using a 3DCG library. Using our modification of FLARToolkit, we were able to verify that a suitable photo image corresponding to the position and orientation of the user's camera was selected, and rendered smoothly without dropped frames. We also confirmed that smooth rendering was possible regardless of the complexity of the object, since there is no need to generate a 3D model as in the conventional method. Using QR-code markers on iOS- and Android-based mobile devices, we also noted smooth operation. Evaluations of rendering performance showed that it was possible to render in nearly the same time as the conventional method when rendering was conducted simultaneous to image loading, and in 20% of the time it took for the conventional method when using proposed method 2, after all the necessary images were loaded.

Finally, we measured the memory consumption and start-up time for different strategies for loading a set number of images under proposed method 2. Batch loading consumed large amounts of memory and took longer to start-up in comparison to lazy-loading and to the conventional method. Lazy-loading consumed only 10% more memory and the start-up time was constant regardless of the total number of images, making it nearly as fast as the conventional method.

REFERENCES

[1] H. Kato and M. Billinghurst, “Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System,” Proc. IWAR `99, pp. 85-94, 1999.

[2] Junaio, metaio, http://www.junaio.com/.

[3] Vuforia, Qualcomm, https://developer.qualcomm.com/mobile-development/mobile-technologies/augmented-reality.

[4] js-aruco, http://code.google.com/p/js-aruco/.

[5] AR-media, http://www.inglobetechnologies.com/.

[6] Unity3D, http://unity3d.com/.

[7] M. Ohta, R. Yokomichi, M. Motokurumada, and K. Yamashita, “A photo-based augmented reality system with HTML5/JavaScript,” Proc of the 1st Global Conference on Consumer Electronics, pp.425-426, Oct., 2012.

[8] M. Brown, T. Drummond, and R. Cipolla, “3D Model Acquisition by Tracking 2D Wireframes,” Proc. of the 11th British Machine Vision Conference, 2000.

[9] A. van den Hengel, A. Dick, T. Thormahlen, B. Ward, and P. H. S. Torr, "Video-Trace: Rapid Interactive Scene Modelling from Video,” ACM Transactions on Graphics, vol. 26 (3), 2007.

[10] Q. Pan, G. Reitmayr, and T. Drummond, “ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition,” Proc. of the 20th Brithish Machine Vision Conference, 2009.

[11] J. Park and W. Lee, “Augmented E-commerce: Making Augmented Reality Usable in Everyday E-commerce with Laser Projection Tracking,” Lecture Notes in Computer Science, vol. 3311, pp. 242-251, October 2004.

[12] X. Li and D. Chen, “Registration Correction for Augmented Reality in E-Commerce,” Proc. of International Conference on Management and Service Science, pp. 1-4, 2009.

[13] X. Li, D. Chen, “Augmented Reality in E-commerce with Markerless Tracking,” Proc. of International Conference on Information Management and Engineering, pp. 609-613, 2010.

[14] S. Gibson and A. Chalmers, Photorealistic Augmented Reality. Eurographics 2003 Tutorial, September 2003.

[15] S. Gibson, J. Cook, T. Howard, and R. Hubbold, Aris: Augmented Reality Image Synthesis. Tech. Rep. IST-2000-28707, University Manchester, July 2004.

[16] “QuickTime Toolkit: Advanced Movie Playback and Media Types,” Apple Computer, Inc., Science Direct, Amsterdam; Boston: Mongan Kaufmann, 2004, pp. 283-287.

[17] FLARToolkit, http://www.libspark.org/wiki/saqoosha/FLARToolKit/.
[18] ZXing, http://code.google.com/p/zxing/.
[19] Three.js, http://mrdoob.github.com/three.js/.

Authors

Masaya Ohta graduated from the Osaka Prefecture University in 1991, completed his doctoral studies in 1996, and became an assistant professor at the Osaka Electro-Communication University an assistant professor with the Graduate School of Engineering at the Osaka Prefecture University in 2002. He has been an associate professor since 2012. He is primarily pursuing research related to augmented reality and communications systems. He holds a D.Eng. degree, and is a member of IEEE, IEICE, IEEJ, and IPSJ.

Ryuta Yokomichi graduated from the Osaka Prefecture University in 2012, and is currently enrolled in the first half of the doctoral program there. He is primarily pursuing research related to augmented reality, and is a member of IPSJ.

Masataka Motokurumada graduated from the Osaka Prefecture University in 2012, and is currently enrolled in the first half of the doctoral program there. He is primarily pursuing research related to augmented reality, and is a member of IPSJ.

Katsumi Yamashita completed his doctoral studies from the Osaka Prefecture University in 1980. He became a lecturer at the Faculty of Engineering, University of Ryukyus, in 1982, an associate professor in 1988, and a professor in 1991. Since 2000, he has been a professor at the Graduate School of Engineering, Osaka Prefecture University. He has primarily been pursuing research on communications and adaptive signal processing. He holds a D.Eng. degree, and is a member of IEEE and IEEJ.