An Intuitive Interface for Visualizing Numerical Data in a Head-Mounted Display with Gesture Control*)

Yuichi TAMURA, Hiroaki NAKAMURA[1)] and Susumu FUJIWARA[2)]

Konan University, Kobe 658-8501, Japan
[1)]*National Institute for Fusion Science, Toki 509-5292, Japan*
[2)]*Kyoto Institute of Technology, Kyoto 606-8585, Japan*

(Received 30 November 2015 / Accepted 22 February 2016)

This study aims to create an interface for visualizing numerical data on a head-mounted display (HMD) and introduce functions to allow control of this visualization via hand gestures. HMDs have the advantage of providing a user with a 360-degree field of view without taking up a lot of space. However, it is difficult to control visualized numerical data intuitively with this type of display because the user cannot see his/her own hand. We therefore introduced functions allowing the user to control the virtual scene with visualized virtual hands. We developed a system in which a virtual menu is presented to the users and they can change the visualization method by pushing virtual panels. The user can move freely in the visualized virtual scene. Moreover, to help users share their thoughts, we have introduced a drawing function that enables users to indicate points and areas of interest in the visualized scene.

Keywords: visualization, numerical simulation, virtual reality, head-mounted display, gesture recognition, virtual menu

DOI: 10.1585/pfr.11.2406060

1. Introduction

Virtual reality technology has been widely used in visualization research. This includes the visualization of numerical data, in particular, using a CAVE type system (Fig. 1). In this virtual reality system, a user wears stereoscopic glasses and watches stereoscopic images. The user can therefore see himself/herself and control objects with his/her own hands. However, this device requires a large amount of space and is difficult to use readily. As an alternative, a head-mounted display (HMD) (Fig. 2) can also be used for observing stereoscopic information. In this system, the user wears a display and can see a three-dimensional (3D) scene. The advantage of this system is that the user can observe a full 360-degree scene and can use the device in a restricted space. However, the user cannot see his/her own hands. It is therefore inconvenient for the user to control the virtual scene using his/her hands or hand-held devices. We have therefore developed an interface for the intuitive control of an HMD environment.

Considerable research has been devoted to virtual menus and their utilization in HMD environments [1–3]. These researches have focused on the usability of virtual menus and their applications in training tasks, rather than on scientific visualization. Moreover, in previous researches, the user was required to use a special device such as a data glove when gesturing. Research on virtual menus for scientific visualization has also been conducted as VFIVE [4–6]. However, this research has focused on visualization on large immersive display environments such as the CAVE system. A visualization system combining the VFIVE and HMD system has been proposed [7]. However this system does not include a gesture recognition function. One study [8] proposed a visualization system for an HMD for visualizing computational flow dynamics. However, this system is only applicable to a specialized field.

Fig. 1 A CAVE-type 3D display system.

Fig. 2 A head-mounted display (HMD) system.

author's e-mail: tamura@konan-u.ac.jp

*) This article is based on the presentation at the 25th International Toki Conference (ITC25).

The purpose of this study is to develop a general-purpose and intuitive visualization for use in an HMD environment with gesture control. The interface we developed allows interaction with visualized objects by pushing virtual panels using visualized virtual hands. The user can draw lines and points using his/her own hands.

2. System Configuration

2.1 Hardware configuration

Figure 3 shows the hardware configuration of our proposed system. The system consists of an HMD (Oculus Rift [1]) and a leap motion sensor [2]. The leap motion sensor is attached to the front of the HMD. The user wears the HMD and watches a 3D visualized scene through lenses for both eyes, attached inside the HMD. A motion sensor is attached to the HMD which can detect the user's head rotation. Hence, the user can look around the scene by rotating his/her head. The oculus rift's angle of view is approximately 100 degrees. The leap motion sensor detects the positions and angles of the user's hands and fingers. This allows the user to control the virtual scene. The effective range of the sensor is approximately 3 cm to 60 cm from the device. Figure 4 shows the effective range and angle for detecting the user's hands.

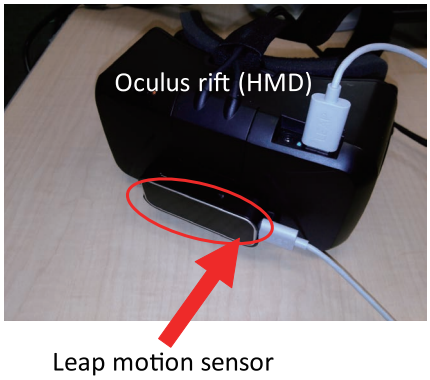

Fig. 3 HMD hardware configuration: A leap motion sensor is attached to the front of the Oculus Rift. The direction of the z axis of the leap motion is forward, i.e., away from the user.

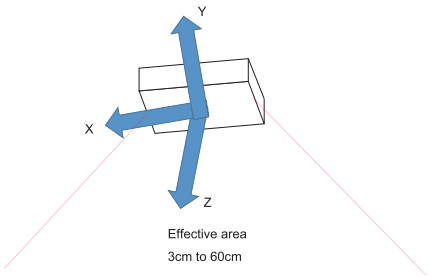

Fig. 4 The coordinate system and the effective area of the leap motion sensor.

[1] https://www.oculus.com
[2] https://www.leapmotion.com

2.2 Software

Our proposed system is based on the Unity 3D engine [3], which was originally developed for designing computer games, specifically in 3D. One of the advantages of this engine is its support for various devices, particularly virtual reality technology. Virtual reality devices have been proposed by many developers. In visualization research, they are used to control the virtual scene. However, it is difficult to connect an existing 3D scene to a new device because of hardware incompatibility and differences in programming environments among other reasons. To bridge these gaps, we used the Unity 3D engine to link the HMD and the leap motion sensor with visualized numerical data.

3. Visualization and Virtual Menu

The virtual menu we developed consists of several virtual panels. The user can select either "static mode" or "dynamic mode". Figure 5 shows the virtual menu presented in front of the user's eye in static mode. The user's hand is also displayed in the virtual scene, and the user can select functions by pushing panels using this virtual hand. If a virtual panel is pushed, the color of the panel changes. In this example, temperature values from a numerical simulation result is visualized using contour function, and lines have been added using the "draw function."

3.1 Virtual menu modes

Figure 5 shows the "static virtual menu". In this mode, the virtual menu is always located in front of the user and does not move. However, this menu causes disruption when the user wants to observe the visualized numerical data. Therefore we added an "on/off" button for the virtual menu. For viewing visualized objects without the virtual menu, the user can switch off most of the virtual pan-

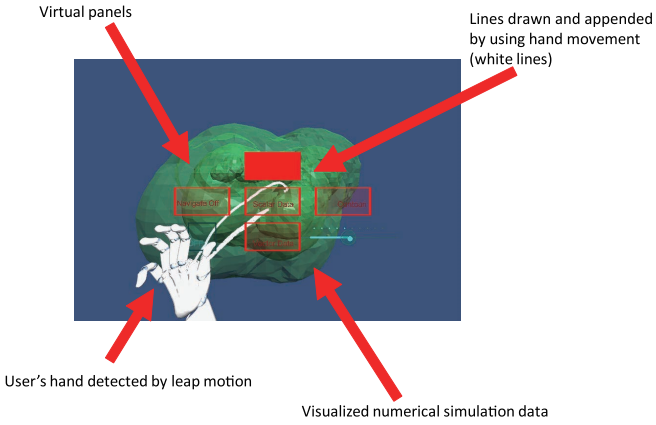

Fig. 5 The virtual menu (static virtual menu).

[3] http://unity3d.com/

els, except the "on/off" panel itself by pushing the "on/off" button. In static mode, the user's inability to freely change the position of the virtual menu causes difficulties when changing parameters. We therefore introduced a "dynamic virtual menu" mode. Figure 6 shows an example of this mode being used. In dynamic mode, the virtual menu is located at the fingertips of the user's left-hand as shown in Fig. 7. The virtual panels are situated at a short distance from the fingertips. This is done because the user selects the virtual panels with their other (right) hand. If the panels were located at the fingers, the fingers of the right hand would overlap those of the left hand, causing misrecognition or non-recognition of the fingers. An advantage of the dynamic menu mode is that it allows the user to hide the virtual panels in an intuitive manner. Figure 6 shows the panels when the user extends all his/her fingers, and Fig. 8 shows the panels when the user extends one finger. The

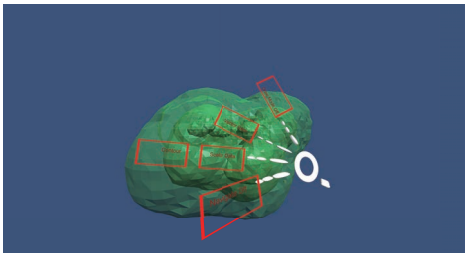

Fig. 6 The dynamic virtual menu when the user extends all fingers All panels are present.

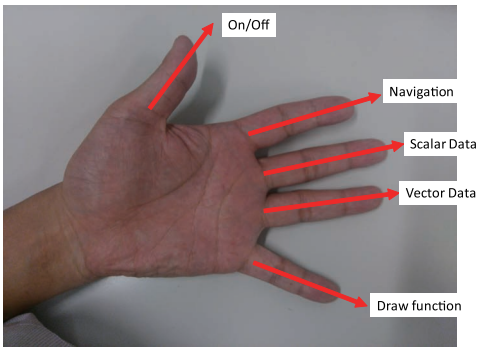

Fig. 7 The coordinate system of the dynamic virtual menu.

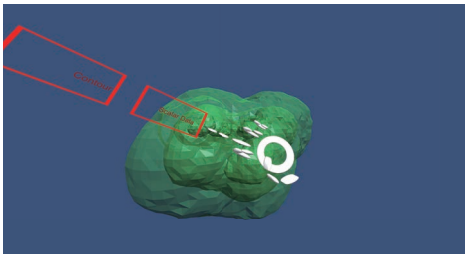

Fig. 8 The dynamic virtual menu when the user extends one finger (middle finger). Only one "scalar data panel" is available.

former shows all the panels, but the latter shows just one panel. Our system can detect whether each finger is extended or not, and if a finger is bent, the panel assigned to that finger is hidden. With this system, the user can intuitively and dynamically use the virtual menu.

The panels are assigned to the fingers as shown in Fig. 7. The menu appears when the sensor detects the user's left hand. The user pushes a panel using a right-hand finger. For example, if the user pushes the "Scalar Data" panel, one or several subpanels open. In Fig. 8, the "contour" panel opens. In this case, there is only a "contour" panel because there is only one set of scalar data in the simulation results. To add a function, it is necessary to create a panel in the virtual scene and write a script for using the function. The panel is easily set by dragging and dropping a panel into the virtual scene using the Unity engine's application development environment. For adding the function, the user needs to write scripts using the ActiViz library. However, since various initialization processes have been already completed, the user can implement functions by simply writing a few lines of script. Moreover, the system uses Visual Tool Kit (VTK) [9], a widely used visualization library, and it is easy to find implementations of functions on the web.

The user needs to select the static or the dynamic menu before starting up the application. After startup, it is not possible to change the menu mode. Although the user select the mode before startup, the user does not need to write a different script for visualization in each mode. The script is shared in both modes. One reason for not removing the static menu mode is that the accuracy of finger detection is a little low. Another is that there are cases where the static menu is more suitable, for example, if the user does not change parameters frequently.

3.2 Visualization function

For visualizing numerical data, the system uses VTK, a widely used visualization library. However, it is impossible to use VTK directly with the Unity 3D engine because the Unity 3D engine does not support C++ while VTK is written in C++. We therefore use the ActiViz library, which is a C# wrapper tool for VTK, developed by the VTK development team.

The first step a user performs is to convert his/her numerical data into a VTK format file. Once the data have been converted, all the VTK functions can be easily implemented. In Fig. 5, an isosurface of temperature is visualized. This illustrates a simulation of the self-assembling behavior of amphiphilic molecules [10–14]. In this case, the simulation results consist only of temperature data; hence, only an isosurface (scalar data) is visualized. Certainly, vector data can also be visualized using VTK functions.

3.3 Draw function

In a CAVE-type system, multiple users can simulta-

neously watch visualized scenes and discuss the results of a numerical simulation. However, this is impossible in an HMD system, because the HMD is designed as a personal device. One solution to this problem is to connect several HMDs via a network. Another is to save scene memories, or "memos", that record an interesting areas or phenomena in a simulation result. If the user indicates the area they are interested in, another user can watch this later and discuss it. For this purpose, we introduced a draw function that enables a user to select a region of the scene by drawing lines and points. The lines and points are drawn using the location of the palm of the hand. Although it is more natural to draw using a finger since the accuracy of finger locations detected by the system is low, palm positions were used. Figure 9 shows the result of drawing a triangle five times. This result shows the lower accuracy resulting from using a finger rather than the palm. The upper side of the triangle is clearly observed in both cases, but the lower side is not successfully drawn when using a finger. This is because the user's finger is hidden by his/her own hand when drawing the lower side due to the position of the leap motion sensor mounted on the user's head. In addition, we introduced a point reduction function to prevent memory overflow. If the user indicates more than 20,000 points, this function deletes the points and instead produces a 3D line.

3.4 Navigation

We introduced a navigation function that works via gesture recognition. Figure 10 shows how visualized virtual objects are moved and rotated. To use the navigation function, the user touches the "Navigate" panel and spreads out the fingers of the right hand. Then, the object moves in the direction that the arm indicates and rotates in accordance with the rotation angle of the arm. If the user bends any finger, the virtual object stops.

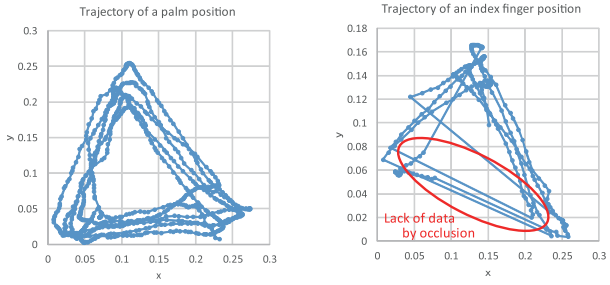

Fig. 9 A comparison of the accuracy of drawing triangles using a palm or a finger. On the left are lines drawn using a palm and on the right using an index finger. The coordinate system used for this result is shown in Fig. 4.

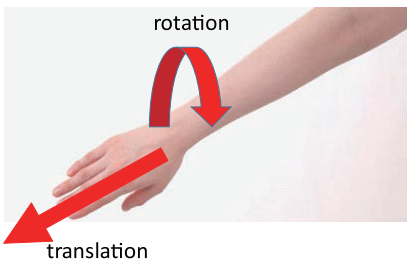

Fig. 10 Navigation in the virtual environment using an arm.

4. Conclusion

We propose an intuitive visualization system for use with an HMD system. Our proposed system consists of a visualization function that uses a VTK library and a draw function selected via a virtual menu. The virtual menu is located in front of the user, and the user can switch between a "static mode" and a "dynamic mode". In the static mode, the virtual menu is fixed in the user's display and does not move. In the dynamic mode, virtual panels are shown at the fingertips of the user's left hand and the user controls the virtual scene by pushing these panels with the right-hand fingers. The panels appear when the user extends his/her fingers. This allows the user to naturally control the virtual scene. Moreover, we introduced a draw function with which the user can select parts of the virtual scene for storage for later viewing. Using the proposed system, a user can observe and intuitively control simulation results.

As the measurement accuracy of the hand recognition sensor is low, the draw function is not fully effective for intuitive representation. To overcome this limitation, we used functions to smooth curves and automatically connect points.

Acknowledgment

This study is partly funded by a Grant-in-Aid for Scientific Research KAKENHI (26330237 and 15K06650), MEXT, Japan and the Hirao Taro Foundation of the Konan University Association for Academic Research, Japan.

[1] D.A. Bowman and C.A. Wingrave, Proc. Virtual Reality, pp. 149-156 (2001).
[2] S. Jayaram *et al.*, Comput. Graph. Appl. **19**, 6 (1999).
[3] J.M. Ritchie *et al.*, Virtual Reality **11**, 4 (2007).
[4] A. Kageyama *et al.*, Prog. Theor. Phys. Suppl. **138**, 665 (2000).
[5] N. Ohno *et al.*, J. Plasma Phys. **72**, 6 (2006).
[6] A. Kageyama *et al.*, Int. J. Model. Simulat. Sci. Comput. **4**, 1340003 (2013).
[7] S. Kawahara and A. Kageyama, Plasma Fusion Res. **10**, 1201087 (2015).
[8] V.J. Shahnawaz *et al.*, Proc. ASME Design Engineering Technical Conferences, pp. 1-7 (1999).
[9] Kitware, Inc., *VTK User's Guide 11th edition* (Kitware, Inc., 2010).
[10] S. Fujiwara *et al.*, J. Plasma Phys. **72**, 1011 (2006).
[11] S. Fujiwara *et al.*, Mol. Simul. **33**, 115 (2007).
[12] S. Fujiwara *et al.*, J. Chem. Phys. **130**, 144901 (2009).
[13] S. Fujiwara *et al.*, Plasma Fusion Res. **6**, 2401040 (2011).
[14] S. Fujiwara *et al.*, Plasma Fusion Res. **10**, 3401029 (2015).