Received: 16 January 2018 | Accepted: 2 February 2018
DOI: 10.1002/cav.1805

RESEARCH ARTICLE

A shared augmented virtual environment for real-time mixed reality applications

Yu Zhu[1,2,3] | Shiying Li[2] | Xi Luo[2] | Kang Zhu[1,2] | Qiang Fu[2] | Xilin Chen[4] | Huixing Gong [5] | Jingyi Yu[2]

[1]Shanghai Institute of Microsystem and Information, Chinese Academy of Sciences, China
[2]ShanghaiTech University, China
[3]University of Chinese Academy of Sciences, China
[4]Institute of Computing Technology, Chinese Academy of Sciences, China
[5]Shanghai Institute of Technical Physics, Chinese Academy of Sciences, China

Correspondence
Jingyi Yu, Center for Virtual Intelligence, ShanghaiTech University, 393 Middle Huaxia Road, Pudong, Shanghai, China.
Email: yujingyi@shanghaitech.edu.cn

Funding information
Science and Technology Commission of Shanghai Municipality, Grant/Award Number: 17XD1402900, 17JC1403800

Abstract
Headsets for virtual reality such as head-mounted displays have become ubiquitous and bring immersive experiences to individual users. People who stand outside the virtual world may want to share the same scenes that are shown on the screen of the headset. It is therefore of great importance to merge real and virtual worlds into the same environment, where physical and virtual objects exist simultaneously and interact in real time. We propose shared augmented virtual environment (SAVE), a mixed reality (MR) system that overlays the virtual world with real objects captured by a Kinect depth camera. We refine the depth map and exploit a Graphics Processing Unit (GPU) based natural image matting method to obtain the real objects from cluttered scenes. In the synthetic MR world, we can render real and virtual objects in real time and handle the depth from both worlds properly. The advantage of our system is that we connect the virtual and real worlds with a bridge controller mounted on the Kinect and need to calibrate the whole system only once before use. Our results demonstrate that the proposed SAVE system is able to create high-quality 1080p live MR footage, enabling realistic virtual experiences to be shared among a number of people in potential applications such as education, design, and entertainment.

KEYWORDS
calibration, Kinect, mixed reality, natural image matting, real time

1 | INTRODUCTION

By wearing a headset for virtual reality (VR), users may enjoy a live music performance by their favorite musician from an arbitrary viewpoint, make a virtual tour to the Great Wall of China, Eiffel Tower, and Statue of Liberty in a single day, or virtually try on clothes from online shops.[1–4] Such devices produce photorealistic 3D contents and offer immersive experiences for people who are actually using the apparatus.

The virtual world, however, is blocked within the headset. In many cases, sharing VR experiences among a number of people such as colleagues and friends is of significance.[5] This desire still remains challenging. First, the user immerses in the virtual world in first-person view (FPV), whereas the others expect to watch how the user interacts with virtual objects in third-person view (TPV). Viewpoints need to be transformed from FPV to TPV. Second, the virtual world with the real objects need to be merged seamlessly in the mixed reality (MR) world, to ensure that everything exists in the same world naturally. Third, because both real and virtual worlds are 3D, depths in the MR world need to be handled consistently

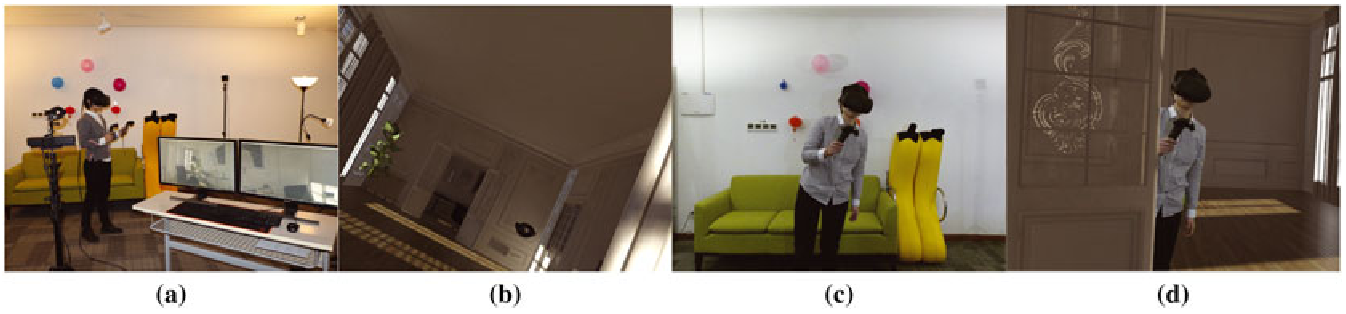


FIGURE 1 Our shared augmented virtual environment. (a) A user wears a headset and interacts with the virtual objects in the virtual scene. (b) The virtual scene displayed on the screen of the headset (in first-person view). (c) The real scene captured from Kinect. (d) The mixed reality footage integrated the virtual scene and the real objects (in third-person view)

according to the depth from both worlds. Finally, real and virtual worlds need to be rendered to create a high-quality live MR footage with tolerable latency.

Milgram et al.[6] introduced a reality–virtuality continuum that encompasses all possible variations and compositions of real and virtual objects between completely real and virtual environments. At the demand of sharing VR experiences, MR[7] aims at creating augmented virtuality, where real objects (such as the user) are augmented into the virtual environment. In this paper, we propose shared augmented virtual environment (SAVE), an MR system that overlays the virtual world with real objects to facilitate the sharing of VR experiences in real time, as shown in Figure 1. Our system overcomes the existing difficulties in creating high-quality MR at a low cost. Virtual and real worlds are connected by a bridge controller mounted on top of a Kinect depth camera, and the real objects can be image mattered from arbitrary real environments. We verify our system on several 1080p MR examples in which human–computer interaction is implemented in the MR world smoothly.

Our contributions in this paper are as follows.

- We introduce a novel method of mounting a third controller on a Kinect to bridge the virtual world displayed in the headset and the real world. Coordinate systems in both worlds can then be transformed easily for the final MR synthesis.
- We propose a stand-alone calibration step to characterize the entire SAVE system. The calibration step needs to be done only once before use, which makes our method flexible and easy to implement.
- We develop a GPU-based natural image matting method for automatically extracting real objects and use an effective depth fusion step in the merging of real and virtual worlds into the MR world with proper depth arrangement.
- We design a multi-thread in multi-process (MTMP) strategy to speed up the rendering of high-quality 1080p live MR footage.

2 | RELATED WORK

In the reality–virtuality continuum, there are two extremes, reality and virtuality. Between the two, we have augmented reality (AR) and augmented virtuality (AV). AR augments the real world with virtual data synthesized, whereas AV augments the virtual environment with data from the real world. In MR, we merge real world with virtual world, which actually covers AR and AV. In the earlier years, people used a 3D live recording room to reconstruct a 3D person and mix them into the virtual world.[8,9] These two methods are complicated, and the cropped person cannot interact with the virtual world. Suma et al.[10] mounted a heavy PrimeSensor with light-emitting diode (LED) markers in front of a head-mounted display (HMD) to capture a depth map, cropped people and objects from the real world, and then mixed them with the virtual world. Günther et al.[11] used chroma-keying techniques to segment hands from background and mixed them into the virtual world. However, they were unable to get the depth of the hands.

Recently, for AR and MR applications, many researchers used depth cameras, such as Kinect, to get depth maps of the real world and merge contents from real and virtual worlds. This approach is able to get accurate depth maps at a low cost and makes it easy to build the sharing VR system. Ventura et al.[12] presented a method to handle occlusion in AR. The method obtained depth maps of the real world by computing stereo image pairs captured in a stereo camera system. However, they had limited accuracy in stereo matching and their AR system.

To tackle the problems of seamless integration of the real world and the virtual world in MR applications, as well as to acquire real-time lighting and interaction in a dynamic environment, Lensing et al.[13] presented a method using a low-cost depth camera to acquire an original depth image with a series of filters and optimization operations. Schiller et al.[14] presented an MR system based on a time-of-flight camera that captures depth maps of the real world in real time. They used the captured depth maps to generate a real-world model and exploited the geometric information for real-world segmentation and occlusion handling. Three cameras, including a fish-eye camera, a time-of-flight camera, and a perspective camera, are integrated into a single system.

Another critical issue in MR is to extract objects of interest from the background. Image matting techniques are developed for many years to solve this problem. There are mainly two classes of image matting techniques, natural image matting and green/blue screen matting. For natural image matting, sampling-based methods such as in the work of Shahrian et al.[15] pick many samples from both foreground and background. The samples are selected according to the weighted sum of pixels. Singaraju et al.[16] assumed that a smooth transition exists between foreground and background layers and carried out image matting by selecting the transition areas. Deep learning-based methods were more recently proposed to achieve the most accurate results[17,18]; however, they cannot obtain results in real time. With the aid of depth information from red, green, blue, and depth (RGBD) cameras, some researchers proposed methods for image matting in real time.[19–21] Inspired by the previous work, we develop a robust method to extract real objects from cluttered scenes.

Real-time tracking devices have been implemented commercially in the recent years. We may need one or more kinds of Microelectromechanical systems gyroscope, accelerometer and laser position sensors to track the pose of a device.[22] HTC Vive integrated these sensors in a single system to track the device pose, called Lighthouse technique. Controllers and HMD can be easily and accurately tracked in the system in real time.

3 | SYSTEM OVERVIEW

Our SAVE system is designed to utilize off-the-shelf hardware with a customized algorithm to build an MR system to share high-quality VR contents among multiple people, as shown in Figure 2. A user wears a VR headset and plays in FPV in the virtual environment. At the same time, the user's actions are captured and transferred along with the VR content to the rendering algorithm. Our algorithm synthesizes an MR world that merges the interaction of the user and the virtual objects. Others can therefore share the experience of the user in the virtual world in TPV.

The system configuration is shown in Figure 2. It has two main aspects, hardware subsystem and software subsystem. The hardware subsystem is composed of off-the-shelf devices, including an HTC Vive for VR display to a user, a Kinect to capture the user's actions, and a host computer to process the data and render the MR world. A third controller included in the HTC Vive, called bridge controller, is mounted on top of the Kinect to connect the real and virtual worlds. Sensors and monitors in the hardware subsystem are used for signal generation, interactive parameters transferring, and results

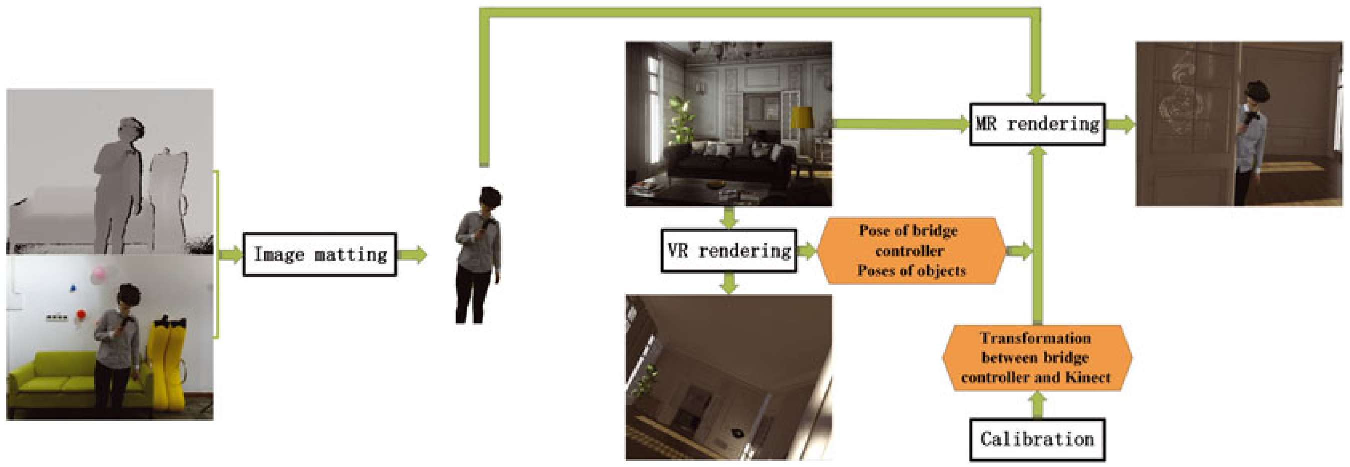


Image matting
MR rendering
VR rendering
Pose of bridge controller
Poses of objects
Transformation between bridge controller and Kinect
Calibration

FIGURE 2 The shared augmented virtual environment system configuration. The system consists of three blocks: virtual reality (VR) rendering to the head-mounted display in the first-person view, communication in and between the VR process and mixed reality (MR) process, and MR integration to the screen in the third-person view

displaying. The software subsystem includes algorithms of system calibration, image matting, and MR rendering. They are designed for connecting all the hardware, detecting the status of devices, collecting data from HTC Vive and Kinect, merging the virtual and real worlds, and displaying the final results.

The hardware subsystem has two types of sensors, positioning sensors on Vive and depth, and image sensors on Kinect. The positioning sensors detect and confirm geometric poses of the Vive, three controllers, and two base stations. The sensors on Kinect capture color images and depth maps of the real world. Meanwhile, there are two types of monitors, HMD on Vive and screen along with the host computer. The HMD shows the virtual world to the user, and the screen displays the MR world in TPV to the audience.

We have three steps in the software subsystem. In the initial step of data preparation, OpenVR software development kit (SDK) is used to connect HTC Vive and Kinect SDK is used to connect Kinect. Positioning of the HMD and controllers, color images, and depth maps of the real world are collected accordingly. In the second step, calibration is performed to transform the coordinate systems between the virtual and real worlds. After calibration, we render the VR and MR videos with the computed transformation between Kinect and the bridge controller. The rendered MR footage can therefore be displayed on the screen in TPV.

In following sections, we introduce the design and implementation of these subsystems in detail.

4 | SYSTEM CALIBRATION

In our SAVE system, the ultimate goal is to render a synthetic MR world in TPV, consisting of both virtual objects and real objects that are extracted from the background. As shown in Figure 3, the system encompasses four coordinate systems: virtual world coordinate system (WCS), HMD coordinate system (HCS), bridge controller coordinate system (CCS), and Kinect coordinate system (KCS). In order to synthesize the MR world, we need to transform virtual objects from WCS to KCS in real time.

Unfortunately, it is difficult to determine the pose of Kinect in WCS. In particular, Kinect may be moved from time to time. It is impractical to calibrate the system every time the configuration is changed. We mount a third controller, except the two used in the HTC Vive system, called bridge controller, on top of the Kinect as a device set, because it is convenient to obtain instant poses of the controller in WCS directly from the OpenVR SDK. The bridge controller is connected to our system via Universal Serial Bus interface.

Our proposed method results in a controller–Kinect (CK) module (see Figure 3). The advantage of this module is that the Kinect is either static or moved; we need not repeat the calibration as long as the bridge controller is rigidly fixed with the Kinect. This calibration step can therefore be stand-alone and is needed only once before use.

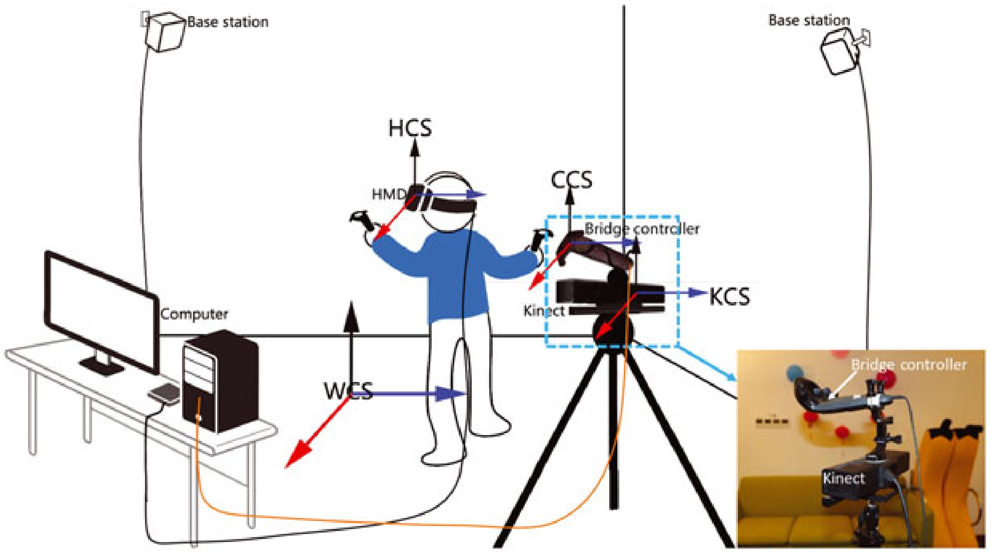


FIGURE 3 Coordinate systems for calibration. The device set of bridge controller and Kinect is shown in the small image. WCS = world coordinate system; HMD = head-mounted display; HCS = HMD coordinate system; CCS = controller coordinate system; KCS = Kinect coordinate system

We keep the CK's position unchanged during calibration. First, we need to get a sufficient number of HMD's 3D coordinates in both WCS and KCS. From these coordinates, we compute the static transformation matrix between WCS and KCS. Meanwhile, we can get directly the transformation matrix between WCS and CCS from OpenVR SDK. From these two transformation matrices, we then obtain the transformation matrix between KCS and CCS. When we render the dynamic scenes in the MR world, we can use this transformation matrix to connect the virtual world and real world effectively. We explain the mathematical relationship between the coordinate systems as follows.

4.1 | HMD coordinates in WCS

We denote the origin of the HCS as a point in WCS. The homogeneous coordinates can be represented as $\mathbf{p}_{wH} = (x_{wH}, y_{wH}, z_{wH}, 1)^T$, which are equal to a pure translation of the HMD origin from the WCS origin. From OpenVR SDK, we obtain the transformation matrix $\mathbf{T}_{W\to H} \in \mathbb{R}^4$ from WCS to HCS at an arbitrary HMD pose. This transformation matrix can be decomposed as the product of a rotation matrix $\mathbf{R}_{W\to H} \in \mathbb{R}^4$ and a translation matrix $\mathbf{Z}_{W\to H} \in \mathbb{R}^4$, that is,

$$\begin{aligned}\mathbf{T}_{W\to H} &= \mathbf{R}_{W\to H}\mathbf{Z}_{W\to H}\\ &= \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0\\ r_{11} & r_{12} & r_{13} & 0\\ r_{11} & r_{12} & r_{13} & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & x_{wH}\\ 0 & 1 & 0 & y_{wH}\\ 0 & 0 & 1 & z_{wH}\\ 0 & 0 & 0 & 1\end{bmatrix}.\end{aligned} \tag{1}$$

All the effective entries in the rotation matrix are exactly the corresponding entries in $\mathbf{T}_{W\to H}$. We then have the translation matrix by

$$\mathbf{Z}_{W\to H} = \mathbf{R}^{-1}_{W\to H}\mathbf{T}_{W\to H}. \tag{2}$$

Note that the HMD coordinates are the first three entries in the last column of the translation matrix. We thus compute the HMD origin coordinates in WCS.

4.2 | HMD coordinates in KCS

In order to get the homogeneous coordinates of the HMD origin in the KCS, we adopt the calibration method by Zhang.[23] Suppose that the HMD origin in KCS is $\mathbf{p}_{kH} = (x_{kH}, y_{kH}, z_{kH}, 1)^T$. The transformation from the KCS to image coordinate system can be represented as

$$(u, v, 1)^T = \frac{1}{z_{kH}}\mathbf{K}\,[\mathbf{I} \quad \mathbf{0}]\,(x_{kH}, y_{kH}, z_{kH}, 1)^T, \tag{3}$$

where (u, v) are the pixel coordinates in the image coordinate system. Here, $\mathbf{K} \in \mathbb{R}^3$ is the camera intrinsic matrix

$$\mathbf{K} = \begin{bmatrix} \alpha & 0 & u_0\\ 0 & \beta & v_0\\ 0 & 0 & 1\end{bmatrix}, \tag{4}$$

where (u_0, v_0) are the coordinates of the principal point, and α and β represent the scale factors in image u and v axes, respectively. Once we know the depth of each point, we can compute the coordinates of the HMD in KCS as follows:

$$\begin{cases} x_{kH} = z_{kH}\frac{u-u_0}{\alpha}\\ y_{kH} = z_{kH}\frac{v-v_0}{\beta}. \end{cases} \tag{5}$$

4.3 | Static transformation matrix from WCS to KCS

Having defined the HMD coordinates in both WCS and KCS, we are able to compute the static transformation matrix from WCS to KCS. In this case, we have the relationship between $\mathbf{p}_{wH}$ and $\mathbf{p}_{kH}$ by defining a transformation matrix $\tilde{\mathbf{T}}_{W\to K}$, such that

$$\mathbf{p}_{kH} = \tilde{\mathbf{T}}_{W\to K}\mathbf{p}_{wH}. \tag{6}$$

In order to compute $\tilde{\mathbf{T}}_{W \to K}$, we need at least 4 pairs of coordinates to solve the 12 unknowns. In practice, we choose $n = 15$ pairs of coordinates in both WCS and KCS. Suppose that the coordinates set in WCS is $S_w \in \mathbb{R}^{4 \times n}$, as follows:

$$\mathbf{S}_W = \begin{bmatrix} \mathbf{p}_{wH1} & \mathbf{p}_{wH2} & \cdots & \mathbf{p}_{wHn} \end{bmatrix}, \tag{7}$$

and that the coordinates set in KCS is $S_k \in \mathbb{R}^{4 \times n}$,

$$\mathbf{S}_K = \begin{bmatrix} \mathbf{p}_{kH1} & \mathbf{p}_{kH2} & \cdots & \mathbf{p}_{kHn} \end{bmatrix}. \tag{8}$$

Hence, we have

$$\mathbf{S}_K = \tilde{\mathbf{T}}_{W \to K} \mathbf{S}_W, \tag{9}$$

and the static transformation matrix can be computed by

$$\tilde{\mathbf{T}}_{W \to K} = \mathbf{S}_K \mathbf{S}_W^{\dagger}, \tag{10}$$

where $\mathbf{S}_W^{\dagger} = (\mathbf{S}_W^T \mathbf{S}_W)^{-1} \mathbf{S}_W^T$ is the pseudoinverse of $\mathbf{S}_W$.

4.4 | Transformation matrix from CCS to KCS

Now, we have the static transformation matrix $\tilde{\mathbf{T}}_{W \to C}$ between WCS and CCS in OpenVR SDK and are able to link the coordinates of an arbitrary point $\mathbf{q}_w$ in WCS, $\mathbf{q}_k$ in KCS, and $\mathbf{q}_c$ in CCS. We have

$$\begin{cases} \mathbf{q}_k = \tilde{\mathbf{T}}_{W \to K} \mathbf{q}_w \\ \mathbf{q}_c = \tilde{\mathbf{T}}_{W \to C} \mathbf{q}_w. \end{cases} \tag{11}$$

Therefore, the transformation matrix from KCS to CCS can be computed as

$$\mathbf{T}_{C \to K} = \tilde{\mathbf{T}}_{W \to K} \tilde{\mathbf{T}}_{W \to C}^{-1}. \tag{12}$$

Because the bridge controller is rigidly fixed on top of the Kinect, the transformation matrix $\mathbf{T}_{C \to K}$ from CCS to KCS remains the same wherever Kinect is moved.

4.5 | Dynamic transformation from WCS to KCS

In the rendering step, we need to transform the objects in the virtual world into KCS, that is, from WCS to KCS. From an arbitrary point $\mathbf{q}_w$ in WCS, its coordinate in KCS can be computed by

$$\mathbf{q}_k = \mathbf{T}_{W \to K} \mathbf{q}_w, \tag{13}$$

where the dynamic transformation matrix $\mathbf{T}_{W \to K}$ can be computed by

$$\mathbf{T}_{W \to K} = \mathbf{T}_{C \to K} \mathbf{T}_{W \to C}, \tag{14}$$

where $\mathbf{T}_{W \to C}$ is the instant transformation matrix obtained from OpenVR SDK in real time, and $\mathbf{T}_{C \to K}$ is the fixed transformation matrix we computed in the static calibration step.

5 | SAVE INTEGRATION

Our goal in the SAVE system is to generate high-quality 1080p MR footage in real time. To achieve high frame rate, we divide the process into two: VR process for rendering the virtual scenes into HMD, in which the user interacts with virtual world objects, and MR process for rendering the MR scenes on the screen in TPV, in which other people watch user interaction in the virtual world.

For fast and effective communication in the system, we design communication modules for communication management and data transfer in the VR and MR processes and also for the data exchange between them, as shown in Figure 4. We refine the depth map from Kinect for accurate depth information, exploit a GPU-based natural image matting technique to extract the user who wears the HMD from the background of the real scenes, and finally integrate all the aspects into our system, as described in details in the following subsections.

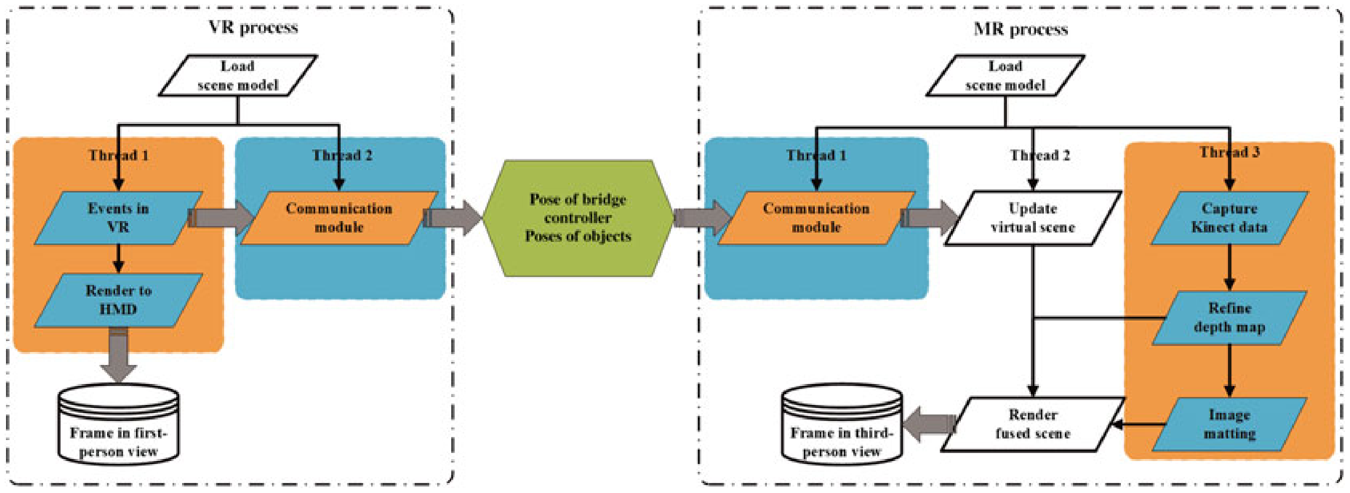


FIGURE 4 Communication pipeline. Communication management and data transfer in the VR process and MR process separately and between the communication modules in the two processes. VR = virtual reality; MR = mixed reality; HMD = head-mounted display

5.1 | Communication modules

According to the independency degree of each component in our system, a straightforward method is to communicate with single thread in a single process. Updating virtual objects (updating module), rendering in HMD (headset module), capturing in Kinect (Kinect module), and fusing virtual and real objects (fusion module) are executed sequentially. This method disrupts the output frame rate, leading to severe rendering latency in either the VR process or the MR process. To improve the performance, we used an alternative, which is multi-thread in a single process. Updating the module and headset module is thus implemented on Thread 1, and Kinect module and fusion module on Thread 2. Key parameters and data need to be exchanged in a swap pool. However, multi-thread in a single process also has defects that fusion module on Thread 2 has to wait for the updated parameters and data from the swap pool, which are actually generated on Thread 1. To further improve the performance, we design an MTMP that separates the VR process and the MR process in a higher layer of architecture as shown in Figure 4. Both processes are running separately and have their own communication modules to establish and terminate the communication and to transfer parameters and data.

In the VR process, the headset module uses the scene model and the view and projection matrices to render the scenes to HMD. These matrices may be affected by the user's interaction with the virtual objects in every frame, resulting in changes of geometry, viewpoint, and lighting. In the MR process, the main task is to handle Kinect module and fusion module and render the frame to the screen. These modules are separately implemented on their own threads and make frames in real time.

For real-time MR rendering, the scene model needs to be identical within the VR process, only except for the difference of view matrix. The communication modules in the two processes therefore adjust the scene model, which may vary partially with the user's interaction in each frame. In order to reduce the nonessential data communication, both processes only exchange data that include the poses of the devices (such as the bridge controller and HMD) and the poses of objects in the changed scene model. Moreover, in the work mode of MTMP, the VR process and the MR process render the scene models and acquire results separately. There is no relation between them regarding the frame rate. The two processes therefore need to load the same scene model and render it with model-view transformation matrix and projection matrix separately.

In the communication modules, we design the work mode of MTMP with TCP/IP socket communication, which makes the two processes implemented within the same network, instead of on the same machine, such that it can improve hardware performance significantly. There are two tasks of the communication modules: (a) Communication management is responsible for the establishment and termination of the socket pipeline, for listening and responding to the data requests, and for connecting, except for handling; and (b) transferring of data for exchanging data with the pipeline, parsing data with preset rules, and updating data to the swap pool.

We then synchronize the parameters and data in the swap pool. The communication modules are in charge of updating the tasks; all the other modules in our system work directly with the parameters and data whenever they are updated. The sending parts where the parameters and data are generated are responsible for actively sending relative parameters and data. The receiving parts get the transferred data from the socket pipeline to the parsing parts. Once the received data agree with the specific rules, the updated parts modify the backup of the corresponding data in the swap pool. Meanwhile, the VR and MR processes can be implemented with the respective backup of data independently, which guarantees the respective frame rate of outputs.

We also need to exchange data between the VR and MR processes. From the VR process, we send the beginning pose of each device and the difference between the beginning pose and the last pose of each device to the MR process. From the MR process, after receiving the data from the VR process, we render the virtual world combined with user interaction from Kinect on the screen in TPV.

5.2 | Depth map refinement

The depth data captured with Kinect are usually incomplete with no-measured depth (NMD) pixels, especially around the boundaries of objects of interest, mainly caused by the baseline between the depth sensor and laser emitter in Kinect and by the occlusion between the objects and the background in the real scene. It is time consuming to refine the depth data with elaboration and is inefficient for real-time MR. We develop a GPU-based method to fill adaptively the NMD pixels in parallel, which is inspired by Zhao et al.[20] We first segment the NMD pixels into shadow (due to foreground–object occlusion) and noise (due to out-of-range or reflective surface), as defined in the work of Yu et al.[24]

The size of the depth map from Kinect is 512×424, which is non-identical to the size of the red, green, and blue (RGB) image, we generate a $1,920 \times 1,080$ depth map using SDK in Kinect. Along each horizontal scanlines, we detect the NMD pixels whose neighbors on the left side are non-NMD pixels, and we list these NMD pixels as NMD line. We count the number of pixels in the NMD line and compare it with a shadow offset, which can be determined as follows:

$$d = bf\left(\frac{1}{z_o} - \frac{1}{z_b}\right), \tag{15}$$

where d denotes the length of the shadow offset, b denotes the length between the laser emitter and the depth sensor, f is the focal length of the depth sensor, and z_o and z_b are the depth values of the object and the background, respectively. We compute the difference between each NMD line and its corresponding shadow offset. If the difference is less than a threshold, the NMD line is considered as shadow; otherwise, it is considered as noise.

After NMD pixels segmentation, we then fill the shadow and noise with two different strategies. For a shadow pixel, because it is caused by foreground–object occlusion, it in fact belongs to the background. We therefore select eight neighboring non-NMD background pixels on the left side of the NMD line and compute the average of their depth values as the depth value of the shadow pixel.

For a noise pixel, as it is caused by out-of-range or reflective surface, we estimate its depth value using a joint bilateral filter with neighboring non-NMD pixels, as in the work of Gangwal et al.,[25]

$$\mathbf{I}(x) = \frac{1}{\mathbf{W}_p} \sum_{x_i \in \Omega} \mathbf{I}(x_i)\mathbf{G}_D\left(\|\mathbf{I}(x_i) - \mathbf{I}(x)\|\right)\mathbf{G}_S\left(\|x_i - x\|\right), \tag{16}$$

where $\mathbf{W}_p$ denotes the normalization term, as determined in the following equation:

$$\mathbf{W}_p = \sum_{x_i \in \Omega} \mathbf{G}_D\left(\|\mathbf{I}(x_i) - \mathbf{I}(x)\|\right)\mathbf{G}_S\left(\|x_i - x\|\right), \tag{17}$$

where $\mathbf{I}(x)$ is depth value of the noise pixel, and $\mathbf{G}_D$ and $\mathbf{G}_S$ are the kernel functions for smoothing depth values and coordinates, respectively.

Our hole-filled depth map can be used to automatically generate a trimap for image matting, in which white, black, and gray indicate *foreground*, *background*, and *unknown*, respectively. For precise matting, we enlarge the *unknown* area in the trimap by applying morphological operations. Also, the hole-filled depth map can be used in the final fusion step.

5.3 | GPU-based geodesic matting

We extract objects in the real world using a natural image matting method, instead of constant color matting such as green screening, so that our system is applicable in robust scenes while the objects can remain in high quality when integrated into the MR world.

In natural image matting, we assume an image I is a composite of a foreground image F and a background image B. The color of the *ith* can be assumed to be a linear combination of the corresponding foreground and background colors,

$$\mathbf{I}_i = \alpha_i \mathbf{F}_i + (1 - \alpha_i)\mathbf{B}_i, \tag{18}$$

where α_i is the pixel's foreground opacity.

The popular closed-form matting[26] relies on the local color–line model, which assumes that the values of F_i obey $F_i = \beta_i F_1 + (1 - \beta_i)F_2$ in a small window, where F_1 and F_2 are the known foreground values in the small window. Although the assumption can handle most situations, it can be easily violated when using large kernels, as pointed out by He et al.[27] For better matting and real-time performance, we improve the geodesic matting method[28] on GPU. We first divide the entire image of $1,920 \times 1,080$ into 3,600 blocks of 32×18. At the same CUDA kernel function, we ignore the image blocks without *unknown* pixels and record those blocks with *unknown* pixels into one set S. Second, for each image block with *unknown* pixels in set S, we compute the foreground probability density for each *unknown* pixel according to the following equation:

$$\mathbf{P}_{\mathcal{F}}(c_x) = \frac{\mathbf{P}_r(c_x|\mathcal{F})}{\mathbf{P}_r(c_x|\mathcal{F}) + \mathbf{P}_r(c_x|\mathcal{B})}, \tag{19}$$

where $\mathbf{P}_r(c_x|\mathcal{F})$ is probability density function (PDF) with the feature of color and depth values. We construct the feature value for each pixel as

$$\mathbf{C}_i = (C_c R, C_c G, C_c B, C_d D), \tag{20}$$

where C_C is the weight of color value to its feature, and C_d is the weight of depth. We also compute $\mathbf{P}_{\mathcal{B}}(c_x)$ for each *unknown* pixel. After the probability density computation for each *unknown* pixel, we calculate the geodesic distance to its background and foreground neighbors. The geodesic distance is considered as the minimum geodesic distance between two pixels, as defined in the work of Bai et al.,[28]

$$d(c_1, c_2) = \min C_{c_1,c_2} \int_{c_1,c_2} \left|\mathbf{W}(x) \cdot C_{c_1,c_2}\right| dx, \tag{21}$$

where C_{c_1}, c_2 denotes the path connecting the pixels c_1 and c_2, and $\mathbf{W}(x)$ is the gradient along the path in feature space of color and depth. We finally compute the alpha value as follows:

$$\mathcal{W}_l(c) = \mathbf{D}_l(x)^{-r} \cdot \mathbf{P}_l(c), l \in \mathcal{F}, \mathcal{B} \tag{22}$$

$$\alpha(c) = \frac{\mathcal{W}_{\mathcal{F}}(c)}{\mathcal{W}_{\mathcal{F}}(c) + \mathcal{W}_{\mathcal{B}}(c)}. \tag{23}$$

5.4 | Virtual–real fusion

Now, we have four images: virtual world color image, virtual world depth image, user color image, and user depth image. They are of the same resolution. We compare the values of the two depth images pixel by pixel and select the color pixel with smaller depth value, which means that we obtain the object closer to the third person.

6 | EXPERIMENTAL RESULTS

We implemented our SAVE system on a 64-bit desktop with Intel Eight Core 3.5GHz CPU, 16GB memory. and NVIDIA GeForce GTX 1080 with 8GB memory graphics card. The RGBD camera is Microsoft Kinect v2, which provides $1,920 \times 1,080$ RGB images and 512×424 depth maps with 30 Hz. We exploited HTC Vive for VR display with which the Lighthouse technique can accurately track in real time the VR devices (such as the controllers and HMD) in the system. For software implementation, we applied CUDA 8.0 library and its auxiliary libraries, including cuSparse and cuSolver. We have ensured that the SAVE system with C++ can satisfy our real-time performance and high-quality MR footage rendering in VR applications of museum tour and online shopping, as shown in Figure 5a,b. We transplanted and executed our system on the Unity platform because rich VR contents are available for application in, for example, entertainment (Figure 5c).

We calibrated the system with the device set of bridge controller and Kinect for the poses of Kinect and obtained satisfactory results that the controller grasped by the user (green) agreed with that rendered in the virtual scene (orange), as shown in Figure 6.

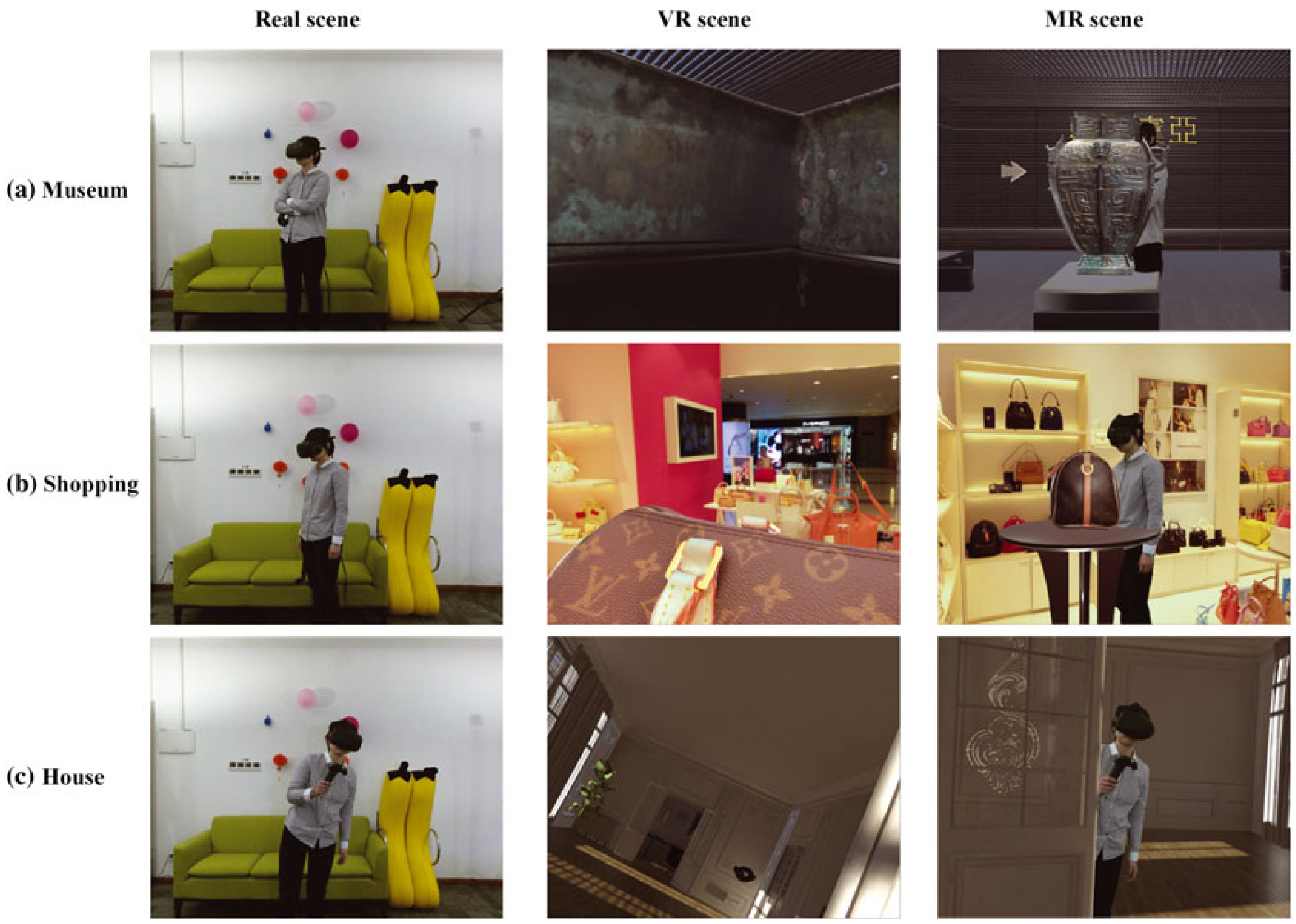


FIGURE 5 The shared augmented virtual environment applications. From left to right: real scene, virtual reality (VR) scene, and mixed reality (MR) footage in (a) museum visiting, (b) virtual shopping, and (c) house touring

In this paper, we extended the depth map of 512 × 424 in Kinect to the depth map of 1,920 × 1,080 and refined the depth map for significant improvement in our previous work [29], as shown in Figure 7. The boundary of the user is much clearer in the refined depth map, compared with the extended depth map (refer the boundary of the user to the RGB image).

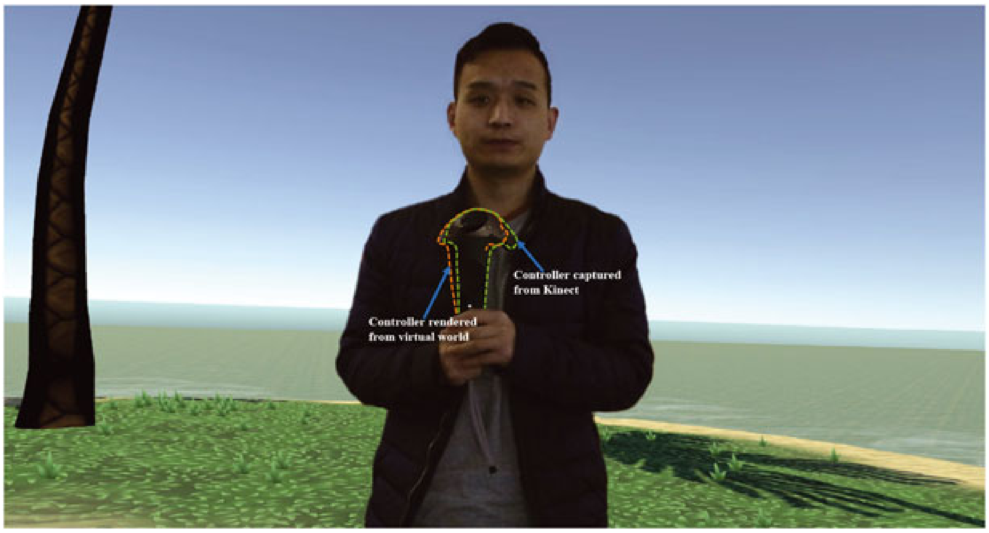

Controller captured from Kinect
Controller rendered from virtual world

FIGURE 6 Calibration results. The controller grasped by the user (green) agreed with that rendered in the virtual scene (orange)

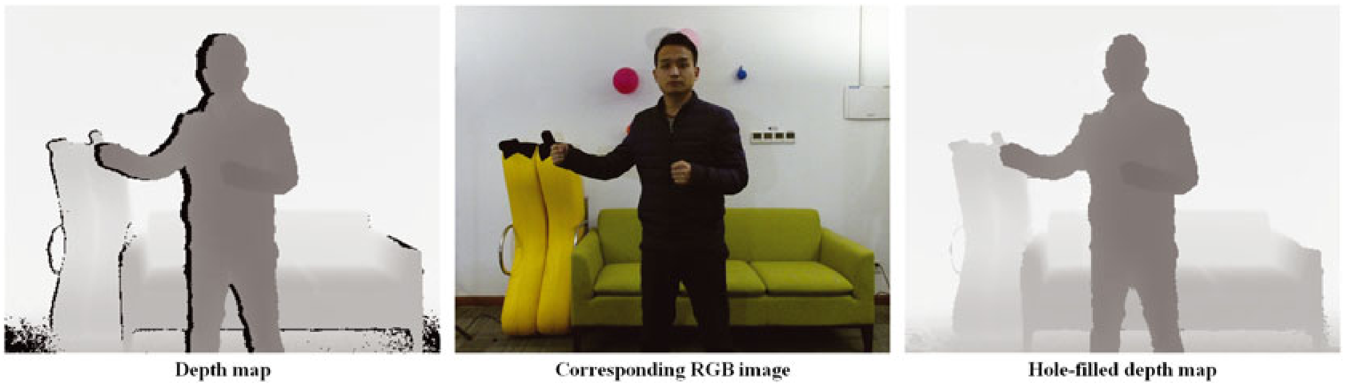


FIGURE 7 Depth map refinement. Left: extended depth map. Middle: corresponding RGB image. Right: hole-filled depth map

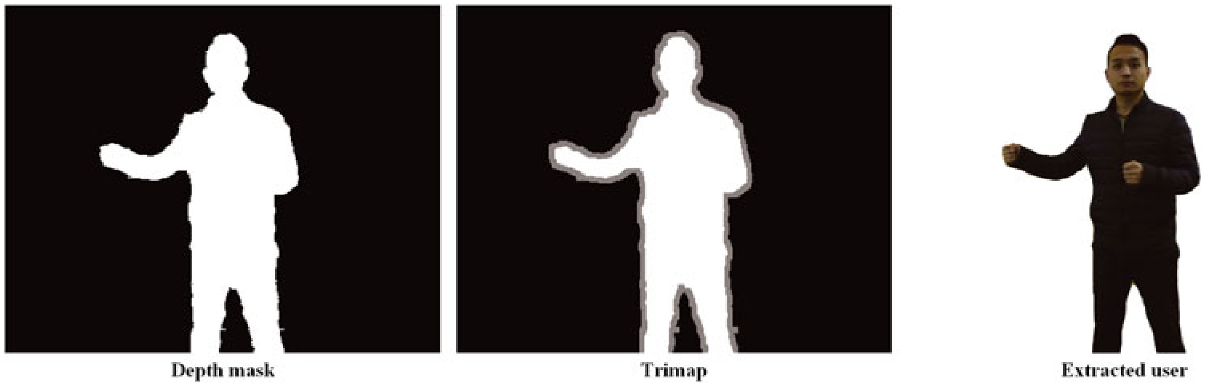


FIGURE 8 Depth map refinement. Left: mask from the refined depth map. Middle: trimap with extended *unknown* area. Right: extracted user

We also improved the green screening with a GPU-based natural image matting method. As shown in Figure 8, the trimap was automatically obtained from the depth mask, which was generated with the refined depth map. The user was extracted with sufficient quality in real time (30 FPS).

In addition, the MR footage with the refined depth map has few defects in Figure 9 (left), in contrast to those shown in fig. 9 (right) in our previous work.[29]

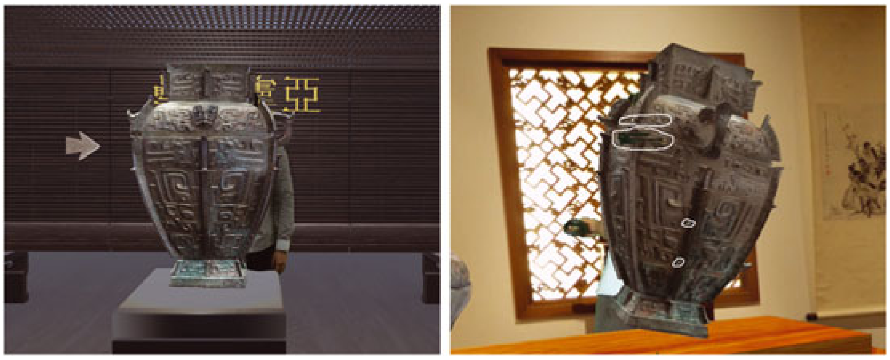


FIGURE 9 Improved mixed reality footage with our refined depth map. The mixed reality footage in our work (left) and that in the work of Zhu et al.[29] (right)

7 | FINAL REMARKS

We have demonstrated a SAVE system for real-time MR applications. We proposed a device set of bridge controller and Kinect to connect the real and virtual worlds. We calibrate the system transformation matrix from the KCS to the bridge controller system, such that we have a fixed relationship between the two, and we need only one calibration before use. We designed an MTMP strategy for the communication in and between the VR and MR processes. Moreover, we refined the depth map and developed a GPU-based natural image matting method for real objects extraction from the real scene. Our experimental results show that the SAVE system can provide VR experiences among multiple people in real time with high-quality 1080p live MR footage.

ACKNOWLEDGEMENTS

This work was supported by the Science and Technology Commission of Shanghai Municipality (17XD1402900, 17JC1403800) The authors thank Shanghai Museum for their support, and Zhong Li, Dapeng Sun, Menggeng Liu, Jiechao Wang, and Yisong Li for valuable discussion and help during the project.

ORCID

Yu Zhu http://orcid.org/0000-0002-5051-9308

REFERENCES

1. Wexelblat A, editor. Virtual reality: Applications and explorations. Cambridge, MA: Academic Press; 2014.
2. Guttentag DA. Virtual reality: applications and implications for tourism. Tour Manage. 2010;31(5):637–651.
3. Allard J, Gouranton V, Lecointre L, et al. FlowVR: a middleware for large scale virtual reality applications. Paper presented at: 10th International Euro-Par Conference; 2004 Aug 31–Sep 3; Pisa, Italy.
4. Altarteer S, Vassilis C, Harrison D, Chan W. Product customisation: virtual reality and new opportunities for luxury brands online trading. Paper presented at: Web3D '16. Proceedings of the 21st International Conference on Web3D Technology, ACM; 2016 Jul 22–24; Anaheim, CA. p. 173–174.
5. Biocca F, Levy MR, editors. Communication in the age of virtual reality. Abingdon-on-Thames, UK: Routledge; 2013.
6. Milgram P, Kishino F. A taxonomy of mixed reality visual displays. IEICE Trans Inf Syst. 1994;77(12):1321–1329.
7. Ohta Y, Tamura H, editors. Mixed reality: Merging real and virtual worlds. Berlin/Heidelberg, Germany: Springer; 2014.
8. Nguyen THD, Qui TCT, Xu K, et al. Real-time 3D human capture system for mixed-reality art and entertainment. IEEE Trans Vis Comput Graph. 2005;11(6):706–721.
9. Shujun Z. An engine of virtual reality mixing environment based on real-time modeling and interaction. Paper presented at: 2011 IEEE International Symposium on VR Innovation, IEEE; 2011 Mar 19–20; Singapore, Singapore. p. 155–159.
10. Suma EA, Krum DM, Bolas M. Sharing space in mixed and virtual reality environments using a low-cost depth sensor. Paper presented at: 2011 IEEE International Symposium on VR Innovation, IEEE; 2011 Mar 19–20; Singapore, Singapore. p. 349–350.
11. Günther T, Franke IS, Groh R. Aughanded virtuality - the hands in the virtual environment. Paper presented at: 2015 IEEE Symposium on 3D User Interfaces (3DUI), IEEE; 2015 Mar 23–24; Arles, France. p. 157–158.
12. Ventura J, Höllerer T. Depth compositing for augmented reality. Paper presented at: SIGGRAPH '08 ACM. SIGGRAPH 2008 Posters; 2008 Aug 11–15; Los Angeles, CA. p. 64.
13. Lensing P, Broll W. Fusing the real and the virtual: a depth-camera based approach to mixed reality. Paper presented at: 2011 10th IEEE International Symposium on Mixed and Augmented Reality, IEEE; 2011 Oct 26–29; Basel, Switzerland. p. 261–262.
14. Schiller I, Bartczak B, Kellner F, Kollmann J, Koch R. Increasing realism and supporting content planning for dynamic scenes in a mixed reality system incorporating a time-of-flight camera. Paper presented at: 5th European Conference on Visual Media Production; 2008 Nov 26–27; London, UK. p. 1–10.
15. Shahrian E, Rajan D. Weighted color and texture sample selection for image matting. Paper presented at: 2012 IEEE Conference on Computer Vision and Pattern Recognition, IEEE; 2012 Jun 16–21; Providence, RI. p. 718–725.
16. Singaraju D, Rother C, Rhemann C. New appearance models for natural image matting. Paper presented at: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE; 2009 Jun 20–25; Miami, FL. p. 659–666.
17. Cho D, Tai Y-W, Kweon I. Natural image matting using deep convolutional neural networks. Paper presented at: ECCV 2016. 14th European Conference on Computer Vision, Springer; 2016 Oct 11–14; Amsterdam, The Netherlands. p. 626–643.
18. Xu N, Price B, Cohen S, Huang T. Deep image matting. 2017. arXiv preprint arXiv:1703.03872.
19. Wang L, Gong M, Zhang C, Yang R, Zhang C, Yang YH. Automatic real-time video matting using time-of-flight camera and multichannel poisson equations. Int J Comput Vis. 2012;97(1):104–121.

20. Zhao M, Fu C-W, Cai J, Cham TJ. Real-time and temporal-coherent foreground extraction with commodity RGBD camera. IEEE J Sel Top Signal Process. 2015;9(3):449–461.

21. Kim YS, Yoon JC, Lee IK. Real-time human segmentation from RGB-D video sequence based on adaptive geodesic distance computation. Multimedia Tools Appl. 2017:1–13. https://doi.org/10.1007/s11042-017-5375-5

22. Comport AI, Marchand E, Pressigout M, Chaumette F. Real-time markerless tracking for augmented reality: The virtual visual servoing framework. IEEE Trans Vis Comput Graph. 2006;12(4):615–628.

23. Zhang Z. A flexible new technique for camera calibration. IEEE Trans Pattern Anal Mach Intell. 2000;22(11):1330–1334.

24. Yu Y, Song Y, Zhang Y, Wen S. A shadow repair approach for Kinect depth maps. Paper presented at: ACCV 2012. Asian Conference on Computer Vision, Springer; 2012 Nov 5–9; Daejeon, South Korea. p. 615–626.

25. Gangwal OP, Djapic B. Real-time implementation of depth map post-processing for 3D-TV in dedicated hardware. Paper presented at: 2010 Digest of Technical Papers International Conference on Consumer Electronics (ICCE), IEEE; 2010 Jan 9–13; Las Vegas, NV. p. 173–174.

26. Levin A, Lischinski D, Weiss Y. A closed-form solution to natural image matting. IEEE Trans Pattern Anal Mach Intell. 2008;30(2):228–242.

27. He K, Sun J, Tang X. Fast matting using large kernel matting Laplacian matrices. Paper presented at: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE; 2010 Jun 13–18; San Francisco, CA. p. 2165–2172.

28. Bai X, Sapiro G. Geodesic matting: a framework for fast interactive image and video segmentation and matting. Int J Comput Vis. 2009;82(2):113–132.

29. Zhu Y, Zhu K, Fu Q, Chen X, Gong H, Yu J. SAVE: shared augmented virtual environment for real-time mixed reality applications. Paper presented at: VRCAI '16. Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry-Volume 1, ACM; 2016 Dec 3–4; Zhuhai, China. p. 13–21.

Yu Zhu received his bachelor degree in Mechatronic Engineering from School of Mechanical Engineering, Northwestern Polytechnical University, Xi'an, China. He is a M.E. degree candidate in computer science, affiliated with Shanghai Institute of Microsystem and Information Technology, Shanghai, China and ShanghaiTech University, Shanghai, China. His current research interests are virtual reality, non-rigid reconstruction, computer version, and computer graphics.

Shiying Li received the degrees of M.E. and Ph.D from Nara Institute of Science and Technology, Nara, Japan, in 2004 and 2007. She is currently a Research Associate Professor with Schoold of Information Sciene and Technology, ShanghaiTech University, Shanghai, China. Her research interests include computer vision and image/video processing. She served as an Organizing/Program Committee Member for over 10 conferences, and a reviewer for many conferences and journals.

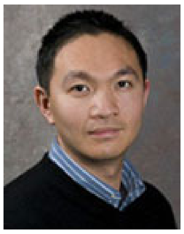
Xi Luo received her B.S. degree in Communication Engineering from School of Mechanical and Electrical and Information Engineering, Shandong University, Weihai, China, in 2016. She is a Ph.D. candidate in computer science, affiliated with ShanghaiTech University, Shanghai, China. Her current research interests are computer graphics, computer vision, and human-computer interaction.

Kang Zhu received his B.S. degree in instrument engineering from School of Optoelectronics, Beijing Institute of Technology, Beijing, China, in 2011. He is a M.E. degree candidate in computer science, affiliated with University of Chinese Academy of Sciences, Beijing, China and ShanghaiTech University, Shanghai, China. His current research interests include computer vision and computational imaging, in particular, hyperspectral light fields.

Qiang Fu received the Ph.D. degree in optical engineering from University of Chinese Academy of Sciences, China, in 2012. He works as a research associate professor at ShanghaiTech University from 2016-2017. His research interests include optical system design, multispectral imaging, computational imaging and nanofabrication.

Xilin Chen received the B.S., M.S., and Ph.D. degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 1988, 1991, and 1994, respectively. He was a Professor with the Harbin Institute of Technology from 1999 to 2005. He has been a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, since 2004. He has authored one book and over 200 papers in refereed journals and proceedings in the areas of computer vision, pattern recognition, image processing, and multimodal interfaces. He served as an Organizing Committee/Program Committee Member for over 80 conferences. He was a recipient of several awards, including the China's State Natural Science Award in 2015, the China's State Scientific and Technological Progress Award in 2000, 2003, 2005, and 2012 for his research work. He is currently an Associate Editor of the IEEE Transactions on Multi-Media, Journal of Visual Communication and Image Representation, a Leading Editor of the Journal of Computer Science and Technology, and an Associate Editor-in-Chief of the Chinese Journal of Computers. He is a Fellow of IEEE/IAPR/CCF.

Huixing Gong received the B.S. degree in Automation from University of Science and Technology of China, Beijing, China, in 1963, and the M.E. degree from Institute of Automation, Chinese academy of sciences, Beijing, China, in 1967. He works at Shanghai Institute of Technical Physics, Chinese Academy of Sciences, since 1968. He is currently the chief engineer of the Shanghai Institute of Technical Physics, Chinese Academy of Sciences, the academician of the Chinese Academy of Engineering, the dean of the School of Information Science and Technology of University of Science and Technology of China, the Standing Committee of the CPPCC National Committee, consultant of the Expert Committee of the State 863 Program for Aerospace and Aviation Technology, vice president of China Instrument and Control Society. He mainly engaged in the research field of infrared photoelectric technology for space applications.



Jingyi Yu received the B.S. degree from the California Institute of Technology, Pasadena, CA, USA, in 2000, and the Ph.D. degree from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2005.He is currently an Professor with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China, and an Associate Professor with the Department of Computer and Information Sciences and the Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, USA. His current research interests include computer vision and computer graphics, in particular, computational cameras and displays. Prof. Yu has served as the Program Chair of the 2011 Workshop on Omnidirectional Vision and Camera Networks, General Chair of the 2008 International Workshop on Projector-Camera Systems, and Area and Session Chair of the 2011 International Conference on Computer Vision. He was a recipient of the NSF CAREER Award and the AFOSR YIP Award. He is an Editorial Board Member of the IEEE Transactions on Pattern Analysis and Machine Intelligence, The Visual Computer Journal, and Machine Vision and Application.

How to cite this article: Zhu Y, Li S, Luo X, et al. A shared augmented virtual environment for real-time mixed reality applications. *Comput Anim Virtual Worlds*. 2018;29:e1805. https://doi.org/10.1002/cav.1805