·Article·

Cloud-to-end rendering and storage management for virtual reality in experimental education

Hongxin ZHANG[1*], Jin ZHANG[1], Xue YIN[1], Kan ZHOU[1], Zhigeng PAN[2], Abdennour EI RHALIBI[3]

1. *State Key Lab of CAD&CG, Zhejiang University, Hangzhou* 3100578, *China*

2. *Alibaba Business College, Hangzhou Normal University, Hangzhou* 310036, *China*

3. *Liverpool John Moores University, UK*

*** Corresponding author,** zhx@cad.zju.edu.cn

Received: 14 April 2020 **Accepted:** 8 July 2020

Supported by the National Key Research and Development Project of China (2018YFB1004904); National Natural Science Foundation of China (U1909204).

Citation: Hongxin ZHANG, Jin ZHANG, Xue YIN, Kan ZHOU, Zhigeng PAN, Abdennour EI RHALIBI. Cloud-to-end rendering and storage management for virtual reality in experimental education. Virtual Reality & Intelligent Hardware, 2020, 2(4): 368—380
DOI: 10.1016/j.vrih.2020.07.001

Abstract **Background** Real-time 3D rendering and interaction is important for virtual reality (VR) experimental education. Unfortunately, standard end-computing methods prohibitively escalate computational costs. Thus, reducing or distributing these requirements needs urgent attention, especially in light of the COVID-19 pandemic. **Methods** In this study, we design a cloud-to-end rendering and storage system for VR experimental education comprising two models: background and interactive. The cloud server renders items in the background and sends the results to an end terminal in a video stream. Interactive models are then lightweight-rendered and blended at the end terminal. An improved 3D warping and hole-filling algorithm is also proposed to improve image quality when the user's viewpoint changes. **Results** We build three scenes to test image quality and network latency. The results show that our system can render 3D experimental education scenes with higher image quality and lower latency than any other cloud rendering systems. **Conclusions** Our study is the first to use cloud and lightweight rendering for VR experimental education. The results demonstrate that our system provides good rendering experience without exceeding computation costs.

Keywords Clout-to-end render; Cloud storage; Virtual reality; Experimental education

1 Introduction

China is a vast country with unbalanced education opportunities[1]. Teaching equipment and facilities are generally inadequate, especially in the western regions and rural areas of China. For this and other reasons, online education is an effective method of implementing education equality. Moreover, since the outbreak of the COVID-19 pandemic, many schools have been forced to provide online education.

Traditional online education methods (e.g., multimedia and live instruction) are insufficient for modern

experimental education needs. Compared with other classroom techniques, experimental education aims to improve observation and operation through practice. Students can remotely perform experiments at the educational facilities.

Virtual reality (VR) is a simulated environment in which computer graphics are used to create a realistic-looking virtual world that can respond to a user's input[2]. VR in experimental education provides immersive, realistic, and intuitive experiences. Existing technologies include Lila[3], the Virtual Chemical Vapor Deposition (CVD) Learning Platform[4], the Virtual Computer-integrated Manufacturing (CIM) Laboratory (VCIMLAB)[5], and RoboUALab[6,7].

Previous work has focused on specific knowledge-transfer areas. None, however, have built a universal system to support all kinds of experimental education. Current solutions use preset models to render scenes at end terminals. However, there may be many complicated experimental scenes in a fully capable education system. It then becomes implausible to place all of the computational burden on the end terminals. A truly dynamic universal system must be able to support mobile devices and terminals with small storage capacities and low computing power.

The aim of this study is to provide a cloud-to-end rendering and storage system to provide high-quality educational experiences with low latency. We divide scenes of experimental education into two parts: background and interactive models. Cloud rendering[8] is used to render the background computations. The rendering results are then transferred to the end terminal using the real-time message protocol (RTMP)[9]. Lightweight rendering is used for the interactive models. Finally, the rendering results are combined at the end terminal. Lightweight rendering leverages a terminal-oriented adaptive algorithm to transfer rendered models based on computing power and network latency. We also propose an improved 3D-warping and hole-filling algorithm that significantly improves image quality when the user's viewpoint changes.

2 Related work

2.1 VR experimental education system

There are many useful VR experimental education systems. Past research has focused on improving knowledge transfer. For example, scholars have applied traditional real-time 3D rendering technology at the end terminal. However, this can tax the end computers' computing ability. In the introduction, several extant systems were listed. The virtual CVD learning platform[4] simulates the CVD process to teach science and statistics fundamentals. The 3D scenes of this platform are very simple, and a 2D interface is used for key interactions. The VCIMLAB[5] is an educational software application consisting of models of common CIM hardware, robots, machines, and computer systems. Students can operate the VR models using real-time operating principles. RoboUALab[6,7] is a virtual remote laboratory (Java applet) used to execute a simulated manipulator that allows students to practice industrial robotics commands.

Notwithstanding the advances made by these studies, traditional 3D rendering capabilities that only handle specific educational areas is considerably limiting. A universal VR experimental education system is needed.

2.2 Cloud rendering

Cloud rendering uses cloud-computing clusters to render 3D model data and to pass results to the end user. The idea was first proposed as WireGL[8], which rendered client submissions via multiple commands. Humphreys et al. proposed the Chromium stream-processing framework based on the WireGL system. Cloud rendering is widely used for cloud-based games[10]. OnLive[11] was the first cloud game platform. There are many more, including Gaikai[12], Ubitus[13], and Ciinow[14]. In China, Aliyun and Zhejiang

University developed the Ali cloud-rendering service, and Tencent launched Tencent Instant Play. To date, few studies have investigated this type of cloud rendering for experimental education.

2.2.1 Cloud-to-end video streaming

With cloud rendering, large amounts of high-definition (HD) image data must be transferred. To achieve a smooth display experience, the frame rate should be greater than 30fps. If the frame-buffer resolution is 1920×1080, the transfer speed will be greater than 10Mbps. This puts great pressure on the network.

Video streaming can transfer HD image data quickly and more efficiently. The major encoding standards are H.261, H.263, and H.264. In- and inter-frame compression are widely used with these. OnLive[11] uses video streaming to transfer rendering results. GamingAnyWhere[15] uses a highly optimized H.264 advanced video encoder to provide better video quality with less latency. This technology provides a mature cloud-rendering solution between the server and the end terminal. The end terminal only decodes the video stream and displays a 2D image frame by frame. With this method, extensive graphics computing power is no longer required.

2.2.2 Lightweight 3D modeling

Lightweight rendering includes lightweight 3D modeling and image-based rendering (IBR). 3D lightweight modeling simplifies 3D models having large numbers of faces and complex details. It filters unnecessary redundant information while retaining necessary structural information. Hoppe[16] proposed an algorithm that obtained 3D model level-of-detail via edge folding and point splitting. Ma et al. proposed a geometric simplification algorithm based on the improved loop subdivision method, and a generated progressive mesh was rapidly reconstructed on mobile devices[17]. Liang et al. proposed a display resolution algorithm based on the moving least-squares method for mobile devices and real-time rendering and interaction[18]. These algorithms reduced the required bandwidth and improved the efficiency of rendering complex 3D models on mobile devices.

IBR[19] uses depth and color maps. Shi proposed a real-time remote rendering system based on IBR[20]. With this, the end terminal only received 2D images and 3D-depth images. When the rendering viewpoint changes within a limited range, the end terminal executes 3D warping to get a new rendering result.

2.3 Storage management

Cloud and point-to-point (P2P) distributed storage technologies are not the same. Cloud storage is divided into public, private, and hybrid cloud storage. Almost all cloud-storage companies use a master-slave replication model and write-time replication technology to ensure high availability. To ensure data consistency and high concurrency, most use data-consistency protocols (e. g., Paxos[21], Raft[22], Zookeeper[23]). Although cloud storage is quite mature now, there is no guarantee that the data will remain 100% safe and reliable.

P2P distributed storage is famous for high bandwidth utilization and low failure rates. There are many such distributed storage products (e.g., the interplanetary file system (IPFS) and the transparent cryptographic file system). Compared with cloud storage, each node in a P2P distributed storage system can contribute space. However, it is difficult let users contribute space. A combination of cloud and P2P distributed storage will, therefore, provide a more reliable storage system for cloud-to-end rendering.

3 Implementation

3.1 Overall architecture

This research provides a cloud-to-end rendering and storage system for VR experimental education. The

overall architecture is shown in Figure 1. Scenes of experimental education can be divided into background and interactive models. Generally, the background part is complicated, and the model files are large. Interactive models are the experimental instruments. They are relatively small and must be operated and rendered frequently. Thus, we use cloud rendering for the background and lightweight rendering for the interactive models.

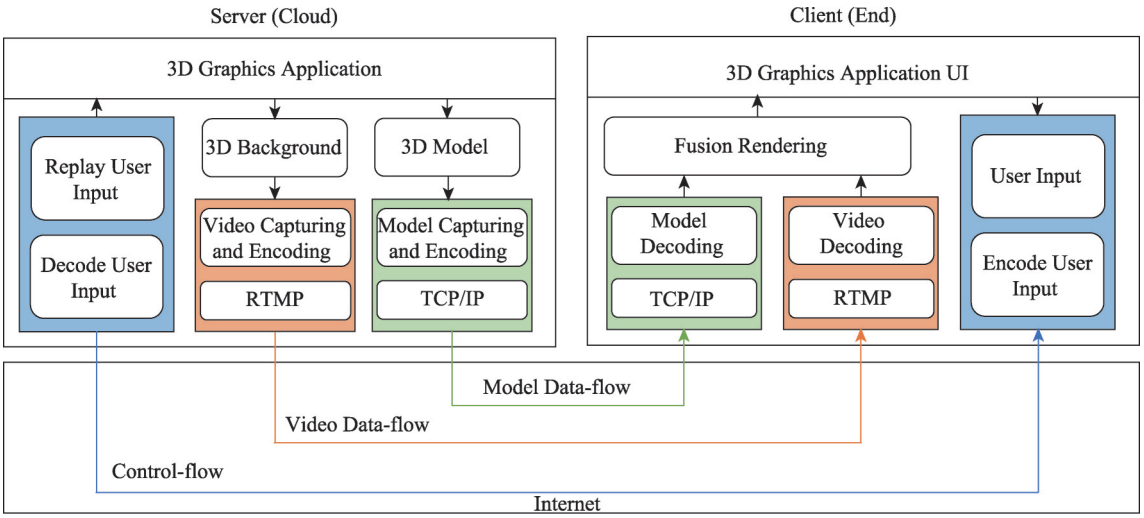

Figure 1 Overall architecture of our proposed cloud-to-end rendering framework.

In the cloud, a graphic application renders the background and captures the rendered results as video. The collected video is then transferred via RTMP[9] to the end terminal after being compressed and encoded. Interactive 3D models are prerendered and adaptively collected as images or grids and compressed based on network status and the rendering capability of the end terminal.

At the end terminal, the client receives the background video stream and model data via the transmission-control protocol/internet protocol (TCP/IP). A video-decoding module decodes video into image frames. For the model data stream, a model decompression module decompresses the data first. Then, the end terminal renders content based on data format. Finally, the video and model images are combined to display complete 3D scenes.

When users translate, rotate, or zoom the interactive models, our system uses an improved 3D-warping methodology to provide new rendering results without network retransmissions. A hole-filling algorithm based on multiple viewpoints and double-warping is also used to improve quality. When the background changes, the end terminal sends the changes to the cloud. The cloud application renders it again and returns the results to the end terminal.

The scenes are decomposed and standardized using the Collaborative Design Activity interchange file format. We, therefore, build a distributed storage system that integrates cloud storage and IPFS to improve reliability and flexibility. Users obtain scenes via the Restful application programming interface (API).

3.2 Cloud-to-end rendering

Cloud-to-end rendering reduces graphic computing requirements for 3D rendering. Via this method, the cloud server converts background imagery into a video stream and prerenders the interactive models. The end terminal then lightweight-renders the interactive models and combines all the results.

3.2.1 Video streaming and decoding

(1) Connection. One handshake occurs between the server and the client. The server creates a socket on a

predefined port, receives the client's connection application, establishes the network connection, and sends the success message.

(2) Video acquisition. After the handshake, the server collects the video stream. The first step is to determine the frame rate. If the application captures 30 frames per second, the video-stream frame rate is 30fps. We use a graphics device interface as the screen-capture device. The images for each frame and timestamp are saved into a red-green-blue (RGB) cache array.

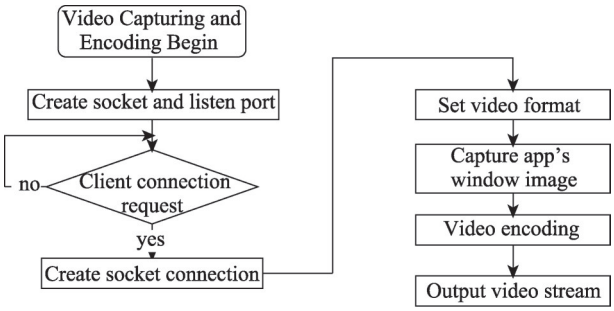

Figure 2 Video streaming for the cloud-to-end rendering framework.

(3) Video compress encoding. FFmpeg is used to compress and encode the captured images into a formatted video. All raw video data are stored in a cache array during acquisition. YUV4:2:0 is used to sample the original RGB image to reduce the transmission bit rate. We use an H.264 video encoder, which is a mixed coding framework based on 16×16 macroblocks. Each block in the current frame, F_n, is encoded with intra- or inter-frame prediction to obtain the prediction block, P. In the intra-frame prediction mode, current frame F_n is encoded, decoded, and reconstructed as uF'_n. The prediction block, P, is then obtained from the intra-frame prediction of uF'_n. In the inter-frame prediction mode, the prediction block, P, is calculated from one or more coded reference frames, uF'_{n-1} by motion estimation and compensation. The residual block, D_n, is obtained by subtracting the predicted block, P, from the current macro block. After transformation and quantization of D_n, the quantization coefficient, X, is obtained. After reordering and entropy-coding X, the encoding result and requisite additional information (e.g., prediction mode, motion vector information, and quantization step size) is formed into the final code stream, which is transmitted and stored in a network abstraction layer (NAL) format.

At the end terminal, we use the H.264 video decoder shown in Figure 3. It receives an encoded code stream via the NAL. Then, it decodes and reorders the data to obtain the quantization coefficient, X, and performs anti-quantization and-transformation on X to obtain the same residual block, D'_n, as in the encoder. Finally, the decoder uses intra- or inter-frame prediction to obtain the same prediction block, P, as the encoder based on the additional information in the code stream. After adding the predicted block, P, and the residual block, D'_n, to obtain the reconstructed block, uF'_n, output-video frame F'_n can be acquired by using the filter to remove the image block effect.

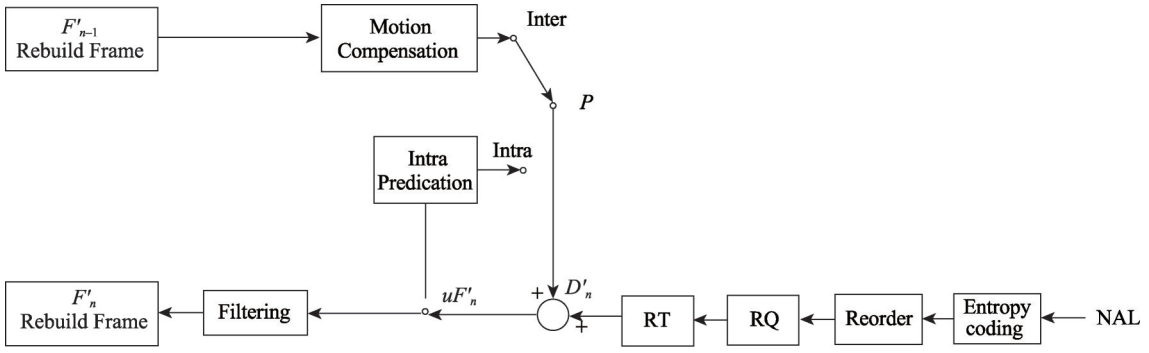

Figure 3 Video encoding and decoding in our cloud-to-end rendering framework.

3.2.2 Model collection and transmission

Interactive models are split out from 3D scenes and pre-rendered based on depth image-based rendering

(DIBR) at the cloud server. After the connection is established, the cloud application loads the related interactive models. The initial position of the camera is (0, 0, 0), and it is then shifted to the left and right by L units, $2L$ units, $3L$ units, $\cdots$, nL units, as shown in Figure 4. The camera position is $V = \{v_i \mid i \in [0, 2n + 1]\}$. Models are rendered for each viewpoint, and the rendering results are as follows:

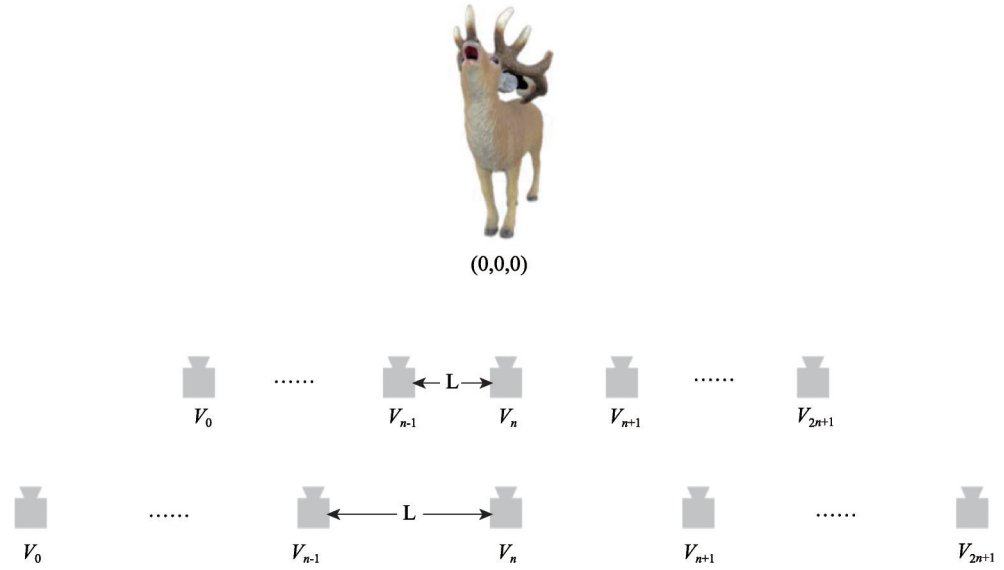

Figure 4 Example of camera shifting for interactive model rendering.

$$T = \{t_i = \langle I_i, D_i \rangle \mid i \in [0, 2n + 1], I_i = render(S, v_i)\},$$

where D_i is the depth image for I_i.

When L is sufficiently small and n is sufficiently large, the reference image sets contain more model details. Because lightweight rendering is used at the end terminal, the reference image sets must be transmitted from the cloud to the end terminal over the internet. Thus, the number of image sets cannot be too large. From experiments, only a few scattered reference-image sets are needed to achieve high quality when using our hole-filling algorithm, which is described in Section 3.2.4. L is set to 4 and n is set to 1 in our study.

Because raw image sets are large, the cloud application uses H.264 to encode the color map sequence, and zlib is used to compress the depth-map sequences. The end terminal then receives model data and uses H.264 and zlib to decompress the data.

3.2.3 Mixed-reality scenario

The 3D background video and interactive model data streams are mixed using scene description language (SDL) at the end terminal. The whole process is shown in Figure 5.

For the background video stream, each YUV420 video frame is decoded into one AVFrame. One SDL_texture is then created from the AVFrame. The end-terminal application then copies each SDL_texture into a shader to complete the background rendering.

We use the improved 3D-warping algorithm described in Section 3.2.4 to lightweight-render the interactive model. When users operate the models, the camera information is updated and rendered again based on ReferenceImage[]. The rendered image adopts the SDL_Surface format and is transformed into SDL_Texture by SDL_CreateTextureFromSurface(). SDL_SetColorKey() removes the background color, and SDL_RenderCopy() copies SDL_Texture into the shader.

We use an event-driven approach to display the final image. A thread is then created to send the SDL

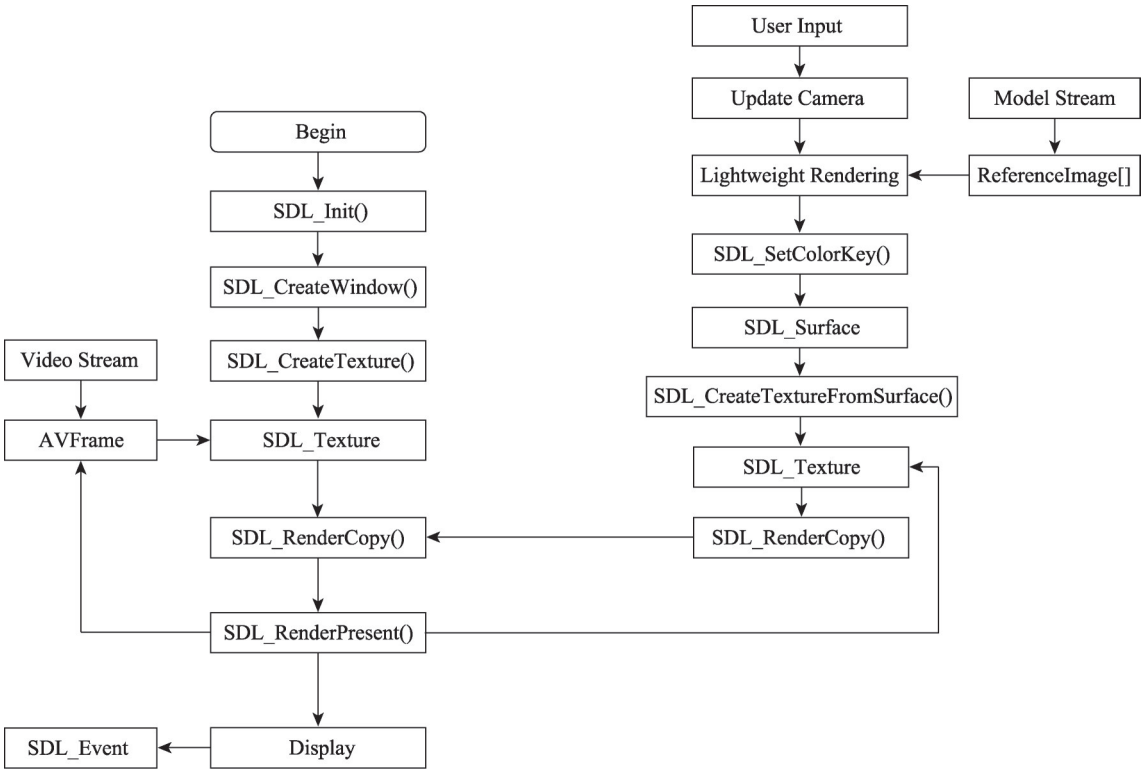

Figure 5 Program diagram for mixed-reality scenario.

notification every 33.3ms. This notification informs the main function to display the appropriate frame. For every refresh, a new AVFrame is read from the video stream, and the rendering results of the interactive models are updated. Finally, the SDL_RenderPresent() mixes the two parts and presents the blended image to users.

3.2.4 Improved 3D warping algorithm

The 3D-warping algorithm uses depth information and camera parameters to project the image points to the 3D space. It then projects the 3D-space points to the virtual imaging plane according to the camera parameters from different viewpoints. The process can be described by the following equation:

$$\boldsymbol{v}_2 = \boldsymbol{M}_2\boldsymbol{V} = \boldsymbol{M}_2\boldsymbol{M}_1^{-1}\boldsymbol{v}_1 \tag{1}$$

where $\boldsymbol{M}_1$ is the transformation matrix from the world space coordinates to the screen coordinates for $\boldsymbol{v}_1$, and $\boldsymbol{M}_2$ is transformation matrix from the world space coordinates to the screen coordinates for $\boldsymbol{v}_2$. To get $\boldsymbol{v}_2$, 16 floating-point multiplications and 12 floating-point additions are needed for the multiplication operation of a 4×4 matrix and a 4×1 vector. The costs are high for 1280×720 or even 1920×1080 high-resolution 2D images. Thus, we use three additions to improve the warping algorithm. We set A = [$\boldsymbol{a}$ $\boldsymbol{b}$ $\boldsymbol{c}$ $\boldsymbol{d}$], and the multiplication can be written as:

$$\boldsymbol{pixel}(i,j) = A\begin{bmatrix} i \\ j \\ k \\ 1 \end{bmatrix} = A\begin{bmatrix} i \\ 0 \\ 0 \\ 0 \end{bmatrix} + A\begin{bmatrix} 0 \\ j \\ 0 \\ 0 \end{bmatrix} + A\begin{bmatrix} 0 \\ 0 \\ k \\ 0 \end{bmatrix} + A\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = i\cdot\boldsymbol{a} + j\cdot\boldsymbol{b} + k_{i,j}\cdot\boldsymbol{c} + \boldsymbol{d} \tag{2}$$

where $i\cdot\boldsymbol{a} = (i-1)\cdot\boldsymbol{a} + \boldsymbol{a}, j\cdot\boldsymbol{b} = (j-1)\cdot\boldsymbol{a} + \boldsymbol{b}$. Then, we can get $\boldsymbol{pixel}(i-1,j)$:

$$\boldsymbol{pixel}(i-1,j) = (i-1)\,\boldsymbol{a} + j\cdot\boldsymbol{b} + k_{i-1,j}\cdot\boldsymbol{c} + \boldsymbol{d}. \tag{3}$$

Based on Equations (2) and (3), we can get the relationship between $pixel(i,j)$ and $pixel(i-1,j)$:

$$\boldsymbol{pixel}(i,j) = pixel(i-1,j) + (k_{i,j} - k_{i-1,j})\cdot\boldsymbol{c} + \boldsymbol{a}, \tag{4}$$

where $k_{i,j}$ is the depth value of (*i, j*), which OpenGL stores in the depth buffer, Z-buffer. Assume $g_{i,j} \in [0,255]$ is the grey value of (*i, j*). Then, $g_{i,j} = \delta(k_{i,j})$. Define $varray[i] = i \cdot \boldsymbol{c}, i \in [0,255]$. Then, Equation (4) can be simplified as follows:

$$\begin{aligned} \boldsymbol{pixel}(i,j) &= \boldsymbol{pixel}(i-1,j) + varray\left[\delta\left(k_{i,j} - k_{i-1,j}\right)\right] + \boldsymbol{a} \\ \boldsymbol{pixel}(i,j) &= \boldsymbol{pixel}(i,j-1) + varray\left[\delta\left(k_{i,j} - k_{i,j-1}\right)\right] + \boldsymbol{b}. \end{aligned} \tag{5}$$

Because $\boldsymbol{pixel}(0,0) = varray\left[\delta\left(k_{0,0}\right)\right] + \boldsymbol{d}$ is constant, *pixel*(*i, j*) can be calculated recursively based on Equation (5) while 3D-warping a pixel in a row or a column. Then, only three additions are needed to calculate the warp.

After 3D warping, there are holes from missing information because of occlusion and insufficient sampling. To improve image quality, the hole-filling algorithm leverages the multiple viewpoints and provides double-warping. The end terminal gets scattered image sets from different viewpoints for each interactive model, as described in Section 3.2.2. From the image sets, we find the two nearest viewpoints, $\boldsymbol{v}_{left}$ and $\boldsymbol{v}_{right}$, for the target viewpoints, $\boldsymbol{v}_{dst}$ and $\boldsymbol{v}_{left} \leqslant \boldsymbol{v}_{dst} \leqslant \boldsymbol{v}_{right}$. Then, we calculate the transform image, W_{left}, from $\boldsymbol{v}_{left}$ to $\boldsymbol{v}_{dst}$ and transform image W_{right} from $\boldsymbol{v}_{right}$ to $\boldsymbol{v}_{dst}$. The final target image is calculated based on Equation (6):

$$W(i,j) = \begin{cases} (1-\alpha)W_{left}(i,j) + \alpha W_{right}(i,j), W_{left}(i,j) \neq 0 \wedge W_{right}(i,j) \neq 0 \\ W_{left}(i,j),\ W_{left}(i,j) \neq 0 \wedge W_{right}(i,j) = 0 \\ W_{right}(i,j),\ W_{left}(i,j) = 0 \wedge W_{right}(i,j) \neq 0 \\ 0,\ W_{left}(i,j) = 0 \wedge W_{right}(i,j) = 0 \end{cases} \tag{6}$$

where $W(i, j)$ is the pixel value of the final image at point (*i, j*), and $W_{left}(i, j)$ and $W_{right}(i, j)$ are the pixel values of the left target image and the right target image at point (*i, j*) respectively. α is the weight coefficient, and t is the translation parameter of the camera.

$$\alpha = \frac{|t - t_{Left}|}{|t - t_{Left}| + |t - t_{Right}|} \tag{7}$$

3.3 Storage management

The architecture of the storage system is shown in Figure 6. It can be divided into four parts:

(1) The microservice system consists of an API gateway, a load balancer, and other components. Its major responsibility is unifying the external interface and load balancing.

(2) The distributed storage subsystem includes the IPFS component, an external storage server, and external cloud storage. It provides basic storage services for the entire system.

(3) The distributed retrieval subsystem includes the IPFS network, a retrieval server, and a knowledge map. Additionally, stored content metadata is required to expand the knowledge map. It returns the model based on the user's request.

(4) The user system includes the IPFS network and a user-permission server. It provides user-level security for the entire system.

The IPFS file system is the cornerstone of the efficient operation of our storage module, and its performance directly determines the overall performance of the storage module.

4 Results

4.1 Application

We used the cloud-to-end system to build the VR experimental education sample. We created two scenes to

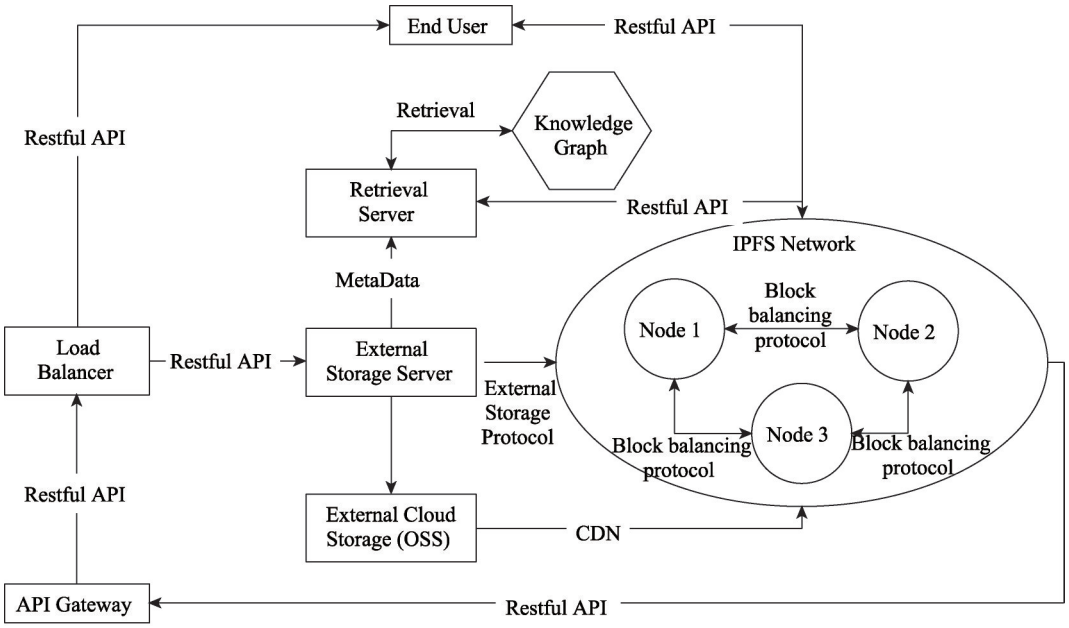

Figure 6 Architecture of storage system.

compare the rendering results between server and client (Figures 7 and 8). Compared with the original cloud-rendered results, the image quality at the end terminal was excellent.

Figure 7 Lab scene of the server (left) and the client (right).

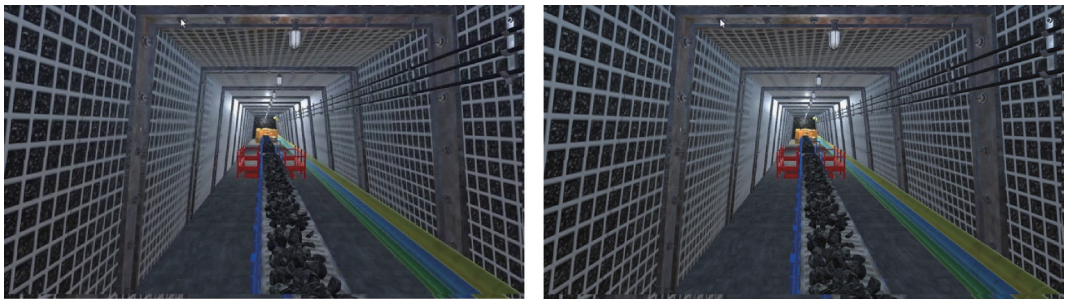

Figure 8 Mine scene of the server (left) and the client (right).

Because these two education scenes are simple, we built a more complicated scene: a forest. There is no apparent difference between the client-rendered image and that of the cloud (Figure 9). Our system can obviously support complicated scenes with high image quality.

4.2 Image quality

We collected data every 50 frames and obtained 40 images for each scene. The peak signal-to-noise ratio (PSNR) and structural similarity (SSIM) index results are shown in Figures 10 and 11. Our system's image

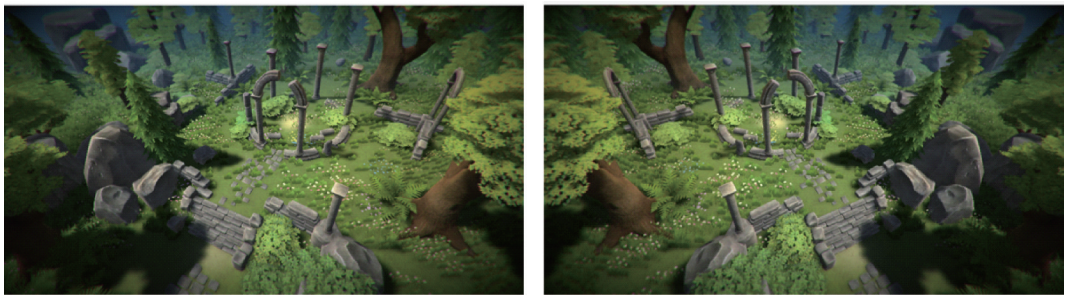

Figure 9 Forest scene of the server (left) and the client (right).

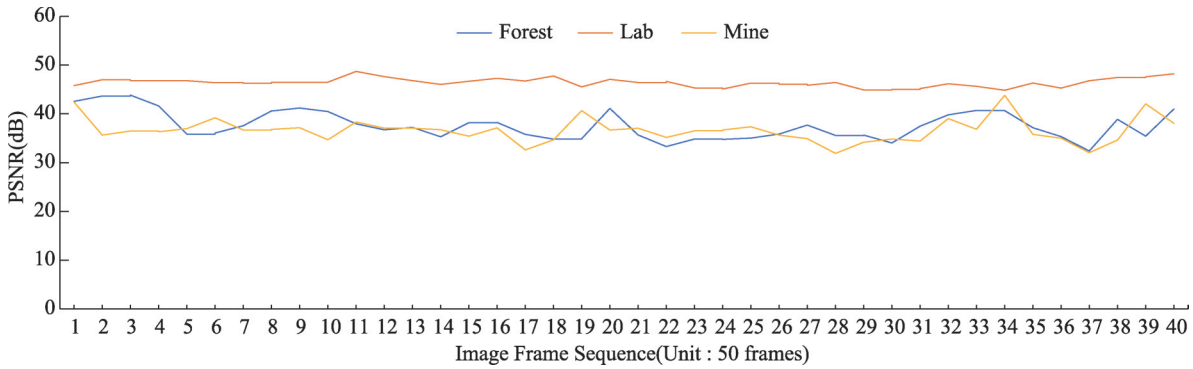

Figure 10 PSNR of 3D background.

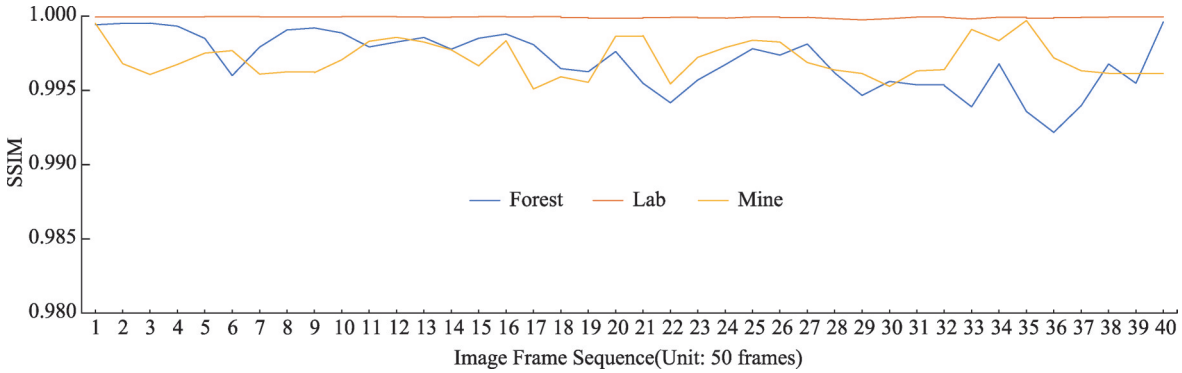

Figure 11 SSIM of 3D background.

quality is comparable with that of OnLive (Table 1).

Table 1 Average PSNR and SSIM comparison with OnLive

	PSNR	SSIM
Ours	37	0.99
OnLive	35	0.95

For the model part, we collected results from different viewpoint using different hole-filling algorithms. The initial position of the camera was (0.0, 0.0, 2.0). We used our improved hole-filling algorithm for positions at (0.4, 0.0, 2.0), (0.0, 0.0, 2.0), and (0.4, 0.0, 2.0). We use a single-viewpoint hole-filling algorithm for (0.0, 0.0, 2.0). The results are shown in Figures12 and 13. Traditional hole-filling algorithms draw a better image when the viewpoint is close to the target object. However, the quality gets worse when the camera moves farther away. The image quality of our method is better, even in this case.

4.3 Latency

Latency is defined as the time from when an interactive event occurs at the terminal to the time a new image is rendered. Because we use lightweight rendering for interactive models, users can directly interact

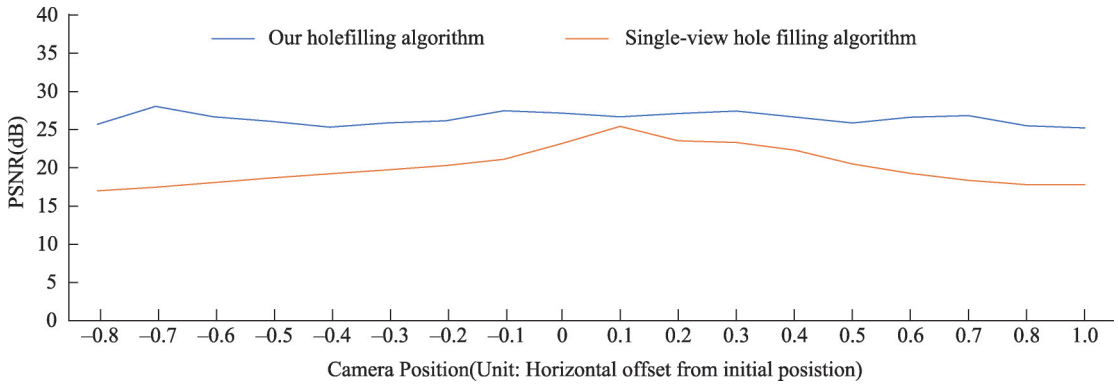

Figure 12 PSNR of 3D model.

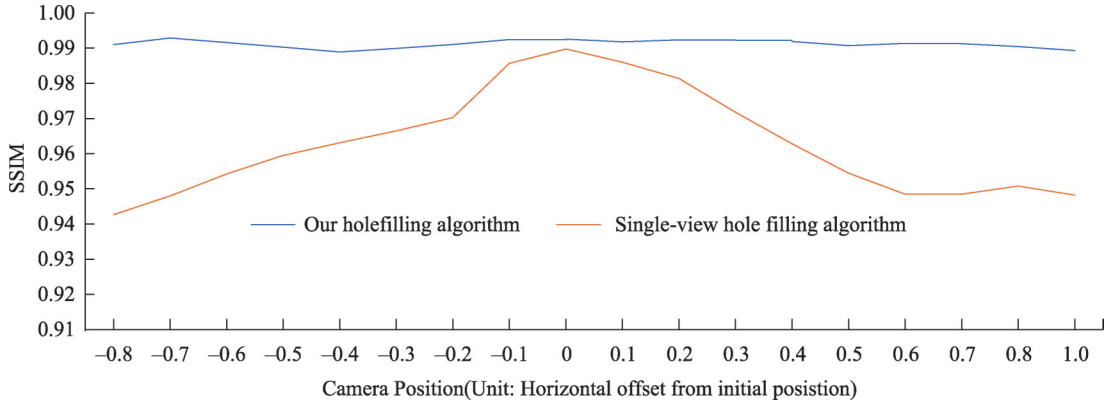

Figure 13 SSIM of 3D model.

with them at the terminal without waiting for a cloud response. Therefore, the latency for our interactive models equals the time needed for lightweight rendering.

We compared the latency of our implementation with OnLive, and the results are shown in Table 2. For the background part, the latency was similar. However, we used lightweight rendering for the interactive models, which can be directly rendered after user operates it. Thus, only the time required for 3D warping and hole-filling needed to be considered. OnLive treats models and backgrounds the same. Thus, the latency equals that of the background. For the same network environment, our system gives the user a better experience with low latency.

Table 2 Comparison with GamingAnyWhere

	Model response delay (ms)	Background response delay (ms)
Ours	38	134
GamingAnyWhere	141	141

5 Conclusion

We proposed a cloud-to-end rendering and storage system to provide a high-quality 3D experience with low latency for experimental education. We analyzed scenes used by typical experimental educational applications and cloud- and lightweight-rendered different aspects of the scenes separately. For the lightweight-rendering part, we used an improved 3D-warping and hole-filling algorithm. The results of this study indicate that our system rendered 3D experimental education scenes with high image quality and low latency. This was the first study to use cloud rendering and lightweight rendering for VR experimental education. We believe our method is generic enough to be adapted to many other application domains, including those of mixed reality.

The major limitation of this study is that it was not easy to apply the tool to arbitrary 3D scenes. On one hand, background and interactive models need to be separated manually in advance. On the other hand, DIBR has limitations. For example, it only allows users to pan, rotate, and scale models in a fixed direction. It is, therefore, difficult to meet all possible user interaction requirements.

Further work should focus on applying the algorithm to automatically identify and separate background and model images and allow users to conduct more interactive operations.

References

1 Yang J, Huang X, Liu X. An analysis of education inequality in China. International Journal of Educational Development, 2014, 37: 2−10
DOI:10.1016/j.ijedudev.2014.03.002

2 Burdea G, Coiffet P. Virtual Reality Technology, New York, Wiley, 1994

3 Richter T, Boehringer D, Jeschke S. LiLa: A European project on networked experiments. In: Automation, Communication and Cybernetics in Science and Engineering 2009/2010. Berlin, Heidelberg, Springer Berlin Heidelberg, 2010, 307−317
DOI:10.1007/978-3-642-16208-4_27

4 Koretsky M D, Amatore D, Barnes C, Kimura S. Enhancement of student learning in experimental design using a virtual laboratory. IEEE Transactions on Education, 2008, 51(1): 76−85
DOI:10.1109/te.2007.906894

5 Hashemipour M, Manesh H F, Bal M. A modular virtual reality system for engineering laboratory education. Computer Applications in Engineering Education, 2011, 19(2): 305−314
DOI:10.1002/cae.20312

6 Jara C A, Candelas F A, Puente S T, Torres F. Hands-on experiences of undergraduate students in Automatics and Robotics using a virtual and remote laboratory. Computers & Education, 2011, 57(4): 2451−2461
DOI:10.1016/j.compedu.2011.07.003

7 Torres F, Candelas F A, Puente S T, Pomares J, Gil P, Ortiz F G. Experiences with virtual environment and remote laboratory for teaching and learning robotics at the university of Alicante. International Journal of Engineering Education, 2006, 22(4): 766−776

8 Eldrdgem H G. A scalable graphics system for clusters. In: Proc. of ACM SIGRAPH 2001. ACM Press, LosAngeles, 2001

9 Lei X H, Jiang X H, Wang C H. Design and implementation of streaming media processing software based on RTMP. 2012 5th International Congress on Image and Signal Processing, 2012, 192–196
DOI:10.1109/cisp.2012.6469981

10 Humphreys G, Houston M, Ng R, Frank R, Ahern S, Kirchner P D, Klosowski J T. Chromium: a stream-processing framework for interactive rendering on clusters. 2002, 21(3): 693−702

11 OnLive. http://www.onlive.com, 2012

12 Gaikai. http://www.gaikai.com, 2012

13 Ubitus. http://www.ubitus.com/, 2013

14 Ciinow. http://www.ciinow.com/, 2014

15 Huang C Y, Hsu C H, Chang Y C, Chen K T. GamingAnywhere: an open cloud gaming system. In: Proceedings of the 4th ACM Multimedia Systems Conference on-MMSys'13. Oslo, Norway, NewYork, ACM Press, 2013, 36−47
DOI:10.1145/2483977.2483981

16 Hoppe H. Progressive meshes. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques-SIGGRAPH '96. New York, ACM Press, 1996, 99−108
DOI:10.1145/237170.237216

17 Hoppe H, DeRose T, Duchamp T, McDonald J, Stuetzle W. Mesh optimization. In: Proceedings of the 20th annual conference on Computer graphics and interactive techniques-SIGGRAPH '93. New York, ACM Press, 1993,19−26
DOI:10.1145/166117.166119

18 Ma J P, Luo X N, Chen B, Chen H H. A geometry simplification method for mobile 3D graphics. Journal of Computer Research and Development, 2008, 45(8): 1395−1401

19 Jr. L M. An image-based approach to three-dimensional computer graphics. Thesis (Ph.D.)-University of North Carolina at Chapel Hill, 1997

20 Shi S, Nahrstedt K, Campbell R. A real-time remote rendering system for interactive mobile graphics. ACM Transactions on Multimedia Computing, Communications, and Applications, 2012, 8(3s): 1−20
DOI:10.1145/2348816.2348825

21 Lamport L. The part-time parliament. ACM Transactions on Computer Systems (TOCS), 1998, 16(2): 133−169
DOI:10.1145/279227.279229

22 Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14). 2014, 305–319

23 Hunt P, Konar M, Junqueira F P, Reed B. ZooKeeper: wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference. 2010

24 Dawson-Howe K M, Vernon D. Simple pinhole camera calibration. International Journal of Imaging Systems and Technology, 1994, 5(1): 1−6
DOI:10.1002/ima.1850050102

25 Wang Y, Ostermann J, Zhang Y Q. Video processing and communications. 2002

26 Wang Z, Lu L G, Bovik A C. Video quality assessment based on structural distortion measurement. Signal Processing: Image Communication, 2004, 19(2): 121−132
DOI:10.1016/s0923-5965(03)00076-6