Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Space

XUESHI HOU, University of California, San Diego
YAO LU, University of California, San Diego
SUJIT DEY, University of California, San Diego

In this article, we explore the possibility of enabling cloud-based virtual space applications for better computational scalability and easy access from any end device, including future lightweight wireless head-mounted displays. In particular, we investigate virtual space applications such as virtual classroom and virtual gallery, in which the scenes and activities are rendered in the cloud, with multiple views captured and streamed to each end device. A key challenge is the high bandwidth requirement to stream all the user views, leading to high operational cost and potential large delay in a bandwidth-restricted wireless network. We propose a novel hybrid-cast approach to save bandwidth in a multi-user streaming scenario. We identify and broadcast the common pixels shared by multiple users, while unicasting the residual pixels for each user. We formulate the problem of minimizing the total bitrate needed to transmit the user views using hybrid-casting and describe our approach. A common view extraction approach and a smart grouping algorithm are proposed and developed to achieve our hybrid-cast approach. Simulation results show that the hybrid-cast approach can significantly reduce total bitrate by up to 55% and avoid congestion-related latency, compared to traditional cloud-based approach of transmitting all the views as individual unicast streams, hence addressing the bandwidth challenges of the cloud, with additional benefits in cost and delay.

CCS Concepts: • **Computing methodologies** → **Virtual reality**;

Additional Key Words and Phrases: Cloud-based mobile multimedia, virtual reality, virtual space, hybrid-cast

ACM Reference format:
Xueshi Hou, Yao Lu, and Sujit Dey. 2018. Novel Hybrid-Cast Approach to Reduce Bandwidth and Latency for Cloud-Based Virtual Space. *ACM Trans. Multimedia Comput. Commun. Appl.* 14, 3s, Article 58 (June 2018), 25 pages.
https://doi.org/10.1145/3205864

1 INTRODUCTION

In recent years, virtual reality (VR) has become increasingly popular and triggered great interest worldwide. A series of new head-mounted displays (HMDs), with the advancement of display and system on chip technology, are unlocking new applications in various fields, including gaming, education, enterprise, entertainment, manufacturing, media, and transportation. However, due to the high computational requirement of VR applications, current HMDs are either tethered to a PC (like Oculus Rift [26] and HTC Vive [18]), or attach to a smartphone (like Samsung Gear VR

Authors' addresses: X. Hou, Y. Lu, and S. Dey are with Mobile Systems Design Lab, Department of Electrical and Computer Engineering, University of California-San Diego, La Jolla, CA 92093 USA; emails: x7hou@ucsd.edu, luyao@ucsd.edu, dey@ece.ucsd.edu.

https://doi.org/10.1145/3205864

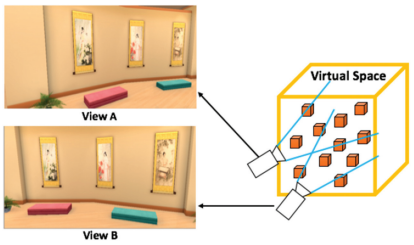

Fig. 1. Two views in the same virtual space application (museum).

[32]), thus decreasing their mobility. To enable a truly portable and mobile VR experience, cloud-based approach is of great interest, where views are rendered and encoded in the cloud, and then transmitted to the end user as video streams over the wireless network. In this case, with mobile cloud computing techniques [14, 42], the HMDs only need to decode the video stream, with the possibility of a dramatically simplified and lightweight HMD design, not tethered to PCs or smartphones.

In this article, we explore the possibility of enabling an emerging category of VR applications, virtual spaces, for better computational scalability and easy access from any end device. Specifically, we consider two virtual space applications—i.e., virtual classroom and virtual gallery, where a teacher/guide and students/visitors from different geographic locations can participate and communicate in the same classroom/gallery session, rendered as avatars in the same virtual space. A key challenge in enabling such cloud-based virtual space applications is the high bandwidth requirement to stream multiple views to each of the users, especially in the case of wireless VR applications, where all the users may share a limited bandwidth. Hence, we seek to minimize the total bit rate needed to transmit video streams of all users' views without compromising video quality. Note that though we develop and demonstrate our approach on two specific virtual space applications, virtual classroom and gallery, our algorithms and approach can be applied to other virtual space applications, such as virtual conference, stadium, campus, and so on.

As opposed to general cloud-based streaming applications, where each user's view is unique, in a virtual space application, users are usually close by with views overlapping. For example, in Figure 1, view A and view B are captured from two virtual cameras (i.e., visitors' views) in a virtual museum application, with a large portion of shared pixels. By taking advantage of this observation, in this article, we propose a novel hybrid-cast approach in which not every pixel of each user's view rendered on the cloud needs to be encoded and streamed from the cloud to each user separately. Instead, we broadcast only one copy of common pixels (common view), and unicast the rest of the pixels (residual pixels or residual views) to each individual user.

Another challenge lies in the selection of common view among multiple users. For better bandwidth reduction, users should be grouped with a large portion of shared pixels so that common view can be maximized and residual view can be minimized. However, an optimal algorithm for grouping is computationally expensive and impractical for real-time use due to the frame-to-frame view change of each user. In this article, we propose a fast and effective smart grouping algorithm with novel metrics to evaluate the quality of grouping. The main contributions of our work can be summarized as follows.

—We propose a novel and efficient hybrid-cast approach to reduce the total bandwidth to stream all the user views in cloud-based virtual space applications by broadcasting a single common view and unicasting only the residual views of individual users, without loss of video quality. Thus, our approach is able to enhance the user experience (i.e., higher video

quality and less latency) by alleviating network congestion. We show that we can reduce up to 55% total bit rate needed for a virtual classroom and a virtual gallery application, with latency improvement in a bandwidth-limited wireless network.
—We develop a common view extraction approach to calculate common view and residual view between any two user views. To the best of our knowledge, we are the first to perform common view extraction by exploring the pipeline of 3D virtual space rendering.
—We propose an automated and smart grouping algorithm to assign multiple users to different groups to minimize the total bit rate needed to transmit all user views.
—We propose a new metric to enable fast and accurate calculation ofa common view between two views. Compared to the optimal algorithm, we achieve similar solution quality with hundreds of times speedup.

Note that we have published a conference paper [17] about this work, where we report on the hybrid-cast approach and some preliminary results. In this article, we extend our approach by proposing a smart real-time grouping algorithm, improving our proposed metrics and conducting experiments on various virtual space applications.

The remainder of the article is organized as follows. In Section 2, we review related work. In Section 3, we introduce virtual space applications and describe the architecture of our proposed hybrid-cast approach. In Section 4, we first describe the methodology for common view extraction, problem statement, optimization metrics, and grouping algorithm. Section 5 presents our experimental setup and we analyze the results. Section 6 concludes our work.

2 RELATED WORK

For cloud-based 3D virtual space applications, the main difference compared to a standard local rendering pipeline [39] is the use and transmission of the entire screen as a video stream [24]. Therefore, reducing total bit rate of the video streams is the key to avoid congestion-related latency and improve user experience, as stated in Section 1. In this section, we review the previous works addressing the bandwidth challenge in (i) codec development, (ii) joint optimization for computer-generated view streaming, and (iii) other techniques.

Upgrading video codec can be the most direct way to reduce bit rate consumption. Video coding standards have evolved primarily through development by ITU-T and ISO/IEC standardization, from H.261 [11] and H.263 [12] by ITU-T, MPEG-1 [1], and MPEG-4 Visual [2] by ISO/IEC, to jointly produced H.262/MPEG-2 [3], H.264/MPEG-4 Advanced Video Coding (AVC) [4], and H.265/MPEG-H High Efficiency Video Coding (HEVC) [19] standards. In recent years, VP9 [31] and AV1 [35] also appear as strong functional video coding formats due to their open and royalty-free features. However, even if the bit rate can be reduced by using new video encoding standards, with the increasing resolution of the devices, the bit rate requirements of the video streams are expected to keep increasing, and particularly in the case of multi-stream VR applications.

In terms of streaming computer-generated views, joint optimization for both video codec and graphics engines has proved to be more effective [23, 25, 34]. Wang et al. [34] propose a rendering adaptation technique to dynamically adjust the richness and complexity of graphic rendering, depending on the network and cloud computing constraints. Liu et al. [23] derive a content-aware adaptive rendering algorithm to adjust rendering factors depending on current network conditions, so as to obtain an "optimal tradeoff" between rendering and encoding quality. Lu et al. [25] propose a joint asymmetric graphics rendering and video encoding approach for automatic selection of texture details or view distance settings for the left view and right view. However, all these approaches compromise video quality to achieve smaller bit rate and are therefore inappropriate to be used for HMDs due to proximity to the eyes.

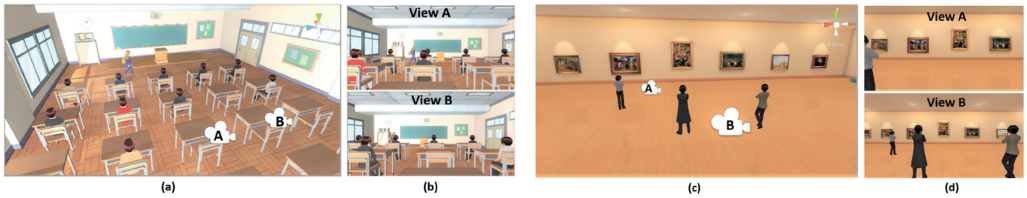

Fig. 2. (a) Two users' positions in a virtual classroom. (b) Corresponding views of two users. (c) Two users' positions in a virtual gallery. (d) Corresponding views of two users.

Alternatively, Cai et al. [9] propose a peer-to-peer cooperative video sharing solution in a multiplayer scenario to substantially reduce the total bit rate from cloud server to game clients. However, their approach is only applicable to scenarios such as 2D third-person games, where game players may share the same bird-view. Hence, their method cannot be applied to 3D virtual spaces.

In summary, codec-only optimization is limited by not exploring any additional benefits in computer-generated view streaming. Joint optimization usually trades off quality for bandwidth, which is inappropriate to be applied for HMDs since users are more sensible to the video quality in VR virtual space applications. Other works explore peer-to-peer streaming, with the limitation of fixed views of all users. Our work is distinguished from all previous works in that (i) we propose a hybrid-casting approach to explore the benefits of computer-generated views in virtual space VR applications to reduce the total bit rate needed while not sacrificing video quality, (ii) we develop a common view extraction algorithm by exploring the 3D rendering pipeline, and (iii) we propose an automated and smart grouping algorithm, with fast and accurate real-time evaluation with empirically validated experimental results.

3 OVERVIEW

In this section, we first present virtual classroom and gallery applications. Next, we describe the architecture of our proposed hybrid-cast approach.

3.1 Virtual Space Applications

We have developed a prototype implementation of the virtual classroom application using Oculus [26] and Unity [33]. Compared to the Massive Open Online Course [15], where students can only watch video without interaction with the teacher and each other, the virtual classroom offers students a more immersive and interactive experience. Figure 2(a) illustrates the virtual classroom, where each student has a unique view of the classroom. Figure 2(b) shows two views from two students as an example. In our implementation, we place cameras to represent the students' views and more views can be easily obtained by placing more cameras. In such a virtual classroom application, neighboring students may share a common view (the set of pixels corresponding to the same coordinates in the object world between two views), as shown in Figure 2(b). As is stated in Section 1, with our proposed approach, we can identify and broadcast the common pixels shared by multiple users, while unicasting the residual pixels for each user. In this way, our proposed approach can address the bandwidth challenges of the cloud, with benefits in cost and delay.

Similarly, we have developed a virtual gallery application using Unity and conducted the related experiments on it. Figure 2(c) illustrates the virtual gallery, where each visitor has a unique view of the gallery. Figure 2(d) shows two views from two visitors as an example. As is shown in Figure 2(d), two visitors also share a common view.

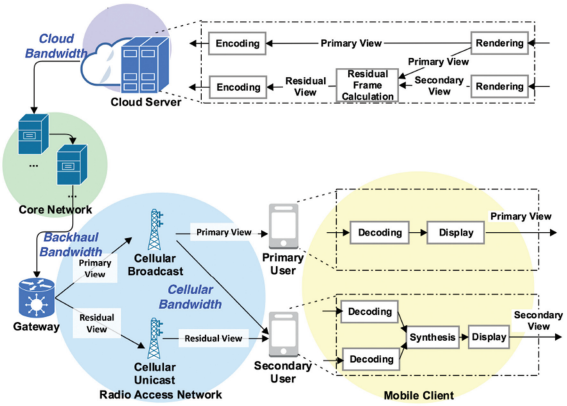

Fig. 3. Hybrid-cast approach.

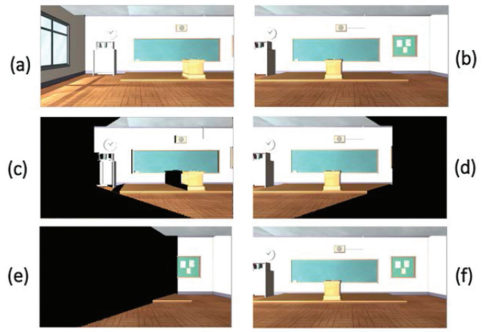

Fig. 4. Illustration of common view extraction: (a) *primary view* A, (b) *secondary view* B, (c) common pixels from view A, (d) common pixels from view B, (e) *residual view* in B, (f) generated view B from *common view* and *residual view*.

3.2 Hybrid-Cast Approach

Figure 3 illustrates the architecture of the hybrid-cast approach. On the cloud, multiple views are rendered in response to commands from multiple users, and then captured, encoded, and transmitted as video streams. To enable our hybrid-cast approach, we cluster the users into different groups based on their views. Within each group, only one user's view is assigned to be the *primary view*, and all the other views are *secondary views*. For transmission, we broadcast the entire *primary view* with full pixel information to all group members, but only unicast the residual pixels (non-existing in the *primary view*) for each *secondary view*. Figure 4 gives an example. Figure 4(a) shows a *primary view* A and Figure 4(b) presents a *secondary view* B. *Secondary view* B consists of two parts, *common view* and *residual view*. The *common view* is part of the view that shares the common pixels between A and B. Figure 4(d) shows the *common view* of B shared with A and Figure 4(c) presents the corresponding shared view from view A's perspective. For *secondary view* B, *residual view* is defined as the pixels non-existing in *primary view* A. Figure 4(e) shows the *residual view*, which can be used to recover *secondary view* B (Figure 4(f)) by combining with *common view* (Figure 4(d)). We explain the common view extraction and our grouping algorithm in Sections 4.1, 4.3, and 4.4. For instance, in a virtual classroom like Figure 2(a), user A is a student in a back row within the class, sharing a subset of the view from the students in front of him. With our proposed

Table 1. Three Types of Bandwidth Savings

Bandwidth Savings	***Feasibility***
Cloud	Always
Backhaul	For users associated with base stations connected by the same gateway
Cellular	For users associated with the same base station

Table 2. Several Cases for Applications

Cases	***Description***	***Bandwidth Savings***
1	*Students in campus join a virtual gallery application* (associated with the same base station)	Cloud, backhaul, cellular
2	*Users in the same urban region attend a virtual classroom (associated with base stations connected by the same gateway)*	Cloud and backhaul
3	*Users in distant places participate in a virtual space* (e.g., belonging to different networks)	Only cloud

grouping algorithm, user A will be clustered into a group, where another student within this group is assigned as the user with *primary view* (since only one user is assigned to have *primary view* in each group). The *primary view* may include the blackboard, teacher, walls, and so on, while the *residual view* of user A may consist of the back of other students in front of him or her, and so on. After doing synthesis of *primary view* and *residual view* on the client side, user A can obtain his own *secondary view*.

The bandwidth savings achieved by our hybrid-cast approach may depend on the location of the users. Figure 3 shows the dataflow from the cloud server (*cloud*) through core network and gateways (*backhaul*) to base stations and end users (*cellular*), consuming different types of bandwidth—*cloud*, *backhaul*, and *cellular bandwidth*—at different stages of the network. The bandwidth savings achieved by our approach (as reported in Section 5.3) are fully translated to the cloud bandwidth savings achieved, irrespective of the location of users, since transmitting a single *primary view* will suffice from the cloud servers. However, the backhaul and cellular bandwidth savings will depend on the location of users. For users associated with the same base station, clearly the bandwidth savings translate to both the cellular and backhaul, besides cloud bandwidth savings, as a single *primary view* can be broadcast to all base stations through the backhaul, and broadcast to all primary users through the access network (*cellular*). For users associated with different base stations that share the same gateway, although cellular bandwidth cannot be saved through cellular broadcast, backhaul bandwidth savings can be achieved since a single *primary view* can be transmitted (multi-cast) through the backhaul to the base stations for these users. However, for users associated with different base stations that do not share same gateway, while the savings do not translate to the backhaul and access bandwidth, cloud bandwidth savings can still be achieved as explained earlier. We summarize the three types of bandwidth savings achieved by our approach under different user locations in Table 1.

We provide an estimation of bandwidth savings for several cases in Table 2. For case 1, when students in campus join a virtual gallery application, we can obtain *cloud, backhaul,* and *cellular* bandwidth savings. In case 2, for users in the same urban region attending a virtual classroom, *cloud*

and *backhaul* bandwidth savings can be gained with our proposed approach. Since the gateway connected to core network can support from 100 base stations for medium size urban network (5x5 km coverage) to over 1,000 base stations (15x15 km coverage) [13, 20], the virtual application users within this coverage area will share the same gateway, and hence will result in *backhaul* bandwidth savings, though not *cellular* bandwidth savings as the latter will depend on the distribution of the users across the base stations. For case 3, where users may belong to different networks, only *cloud* bandwidth savings can be achieved.

Note that to maximize gains in bandwidth savings, our proposed method requires multicast protocols to be employed on the entire end-to-end paths (from the cloud center, through the core network, and all the way to users). Since multicast protocols are not widely employed in today's networks, our approach is developed based on the assumption that multicast protocols will be well deployed and utilized in the future networks. Next, we briefly discuss the existing data transmission mechanisms that can be used for access links (cellular) and backhaul links. As for wireless access links, broadcast and unicast can be realized by using existing LTE Evolved Multimedia Broadcast Multicast Services [22] standards, and broadcast/multicast point to multipoint [30] being developed for future 5G access networks. In terms of transmissions in cloud and backhaul links, advanced IP multicast [36] protocols can be used, including Pragmatic General Multicast [37], Multicast File Transfer Protocol [21], Real-Time Transport Protocol [38], and Resource Reservation Protocol [40]. Data routing, forwarding, and associated transmission protocols should be further studied in future work.

Our approach achieves best performance in reducing bandwidth when users are associated with the same base station, and only *cloud* bandwidth savings (no *backhaul* and *cellular*) can be guaranteed when users are distributed in distant places (since users do not share the same cellular gateway). Moreover, the limitations of our approach mainly come from (a) the difficulty of large-scale deployment of multicast protocols in current networks and (b) our approach only supports an indoor 3D environment (where views aimed toward the same wall) and does not support a fully tridimensional environment such as an outdoor environment. We propose the following to address these limitations. For limitation (a), apart from expecting multicast protocols widely deployed in the future, we can first apply our proposed approach in scenarios where users are associated with same base station or base stations connected by the same gateway, and take the conventional method (i.e., unicasting the view directly) for transmitting views for users associated with other distant base stations. For example, the community in a campus or company is such a good scenario. In terms of limitation (b), in our future work, we will further develop corresponding metrics to apply our approach in fully tridimensional environments such as outdoor environments.

4 OUR APPROACH

In this section, we first describe the methodology for common view extraction and give the problem statement. Next, we describe our optimization metrics and provide empirical validation. We then present our grouping algorithm.

4.1 Common View Extraction

Figure 5 illustrates the common view extraction flow between two users. In the flow, we assume that user A is assigned the *primary view* and user B is assigned the *secondary view*. To extract the *common view*, we perform a series of transformations from window space of user A to the object space, and further to the window space of user B. A *common view* is the set of pixels within the window space of A that falls into the window space of B after the above transformation. To better illustrate our approach, we define two series of transformations: (i) *forward transformation* and

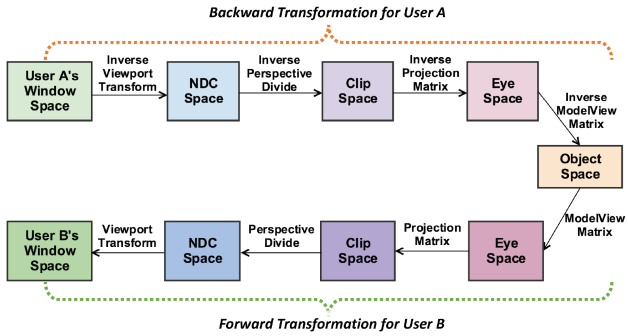

Fig. 5. Architecture of common view extraction.

(ii) *backward transformation*. For a pixel with coordinate $(x, y, z)_{object}$ in the object space, the *forward transformation* calculates the corresponding coordinate $(x, y, z)_{window}$ in the window space and vice versa for the *backward transformation*.

Specifically, the OpenGL [28] pipeline performs the *forward transformation* from object space to window space, through four steps [5, 6]. The first step is to transform from the object space to the eye space using the ModelView matrix $M_{ModelView}$, then we transform from the eye space to the clip space using projection matrix $M_{Projection}$; next we transform from the clip space to the Normalized Device Coordinate (NDC) [16] space using perspective dividing M_{Divide}; last, we transform from the NDC space to the window space by performing viewport transformation $M_{Viewport}$. (The details of OpenGL pipeline are omitted due to space constraints.) Overall, for a pixel with coordinate (x, y, z) in the object space, the procedure of transformation to the window space can be described as the following:

$$(x, y, z)_{window} = M_{Viewport} \cdot M_{Divide} \cdot M_{Projection} \cdot M_{ModelView} \cdot (x, y, z)_{object}.$$

To check the common pixels between two window spaces, as described in Figure 5, we propose to use the inverse transformation matrices for *backward transformation* (i.e., calculating corresponding coordinates from the window space to the object space), as described in the following equation:

$$(x, y, z)_{object} = M^{-1}_{ModelView} \cdot M^{-1}_{Projection} \cdot M^{-1}_{Divide} \cdot M^{-1}_{Viewport} \cdot (x, y, z)_{window}.$$

Therefore, by utilizing both the *backward* and the *forward transformations* in serial for only those pixels in window space A, we can tell if the pixel falls in window space B. Thus, we can extract the *common view*. Furthermore, we can also recover the *secondary view* by using the *primary view* and *residual view* based on the same theory. This procedure is named as synthesis in the Figure 3.

4.2 Problem Statement

We introduce the problem statement in this subsection. The basic notation used in our formulation is provided in Table 3.

Given: All views in the virtual classroom; dimensions of the virtual classroom, including the width and length.

Table 3. Notations Used in Our Formulation

Notation	Meaning
V	view
V_{com}	common view
V_{res}	residual view
n	number of users
$P[\cdot]$	number of pixels
$R[\cdot]$	pixel ratio
P_{frame}	number of pixels in one frame
P_{total}	total number of pixels transmitted
R_{total}	total pixel ratio transmitted
p	primary view
q	secondary view
S_p	set of primary views
S_q	set of secondary views
B	binary indicator matrix
$b_{p,q}$	element of binary indicator matrix B
D	distance matrix
$D(p,q)$	element of distance matrix D

Find: An optimal strategy to minimize the total number of pixels transmitted P_{total} for all views.

$$\begin{aligned}
\min P_{total} &\Leftrightarrow \min\left\{\sum_{q\in S_q} P[V_{res}(q)] + \sum_{p\in S_p} P[V(p)]\right\} \\
&\Leftrightarrow \min\left\{\sum_{q\in S_q} R[V_{res}(q)] + \sum_{p\in S_p} R[V(p)]\right\} \\
&\Leftrightarrow \min\left\{\sum_{q\in S_q} R[V_{res}(q)] + |S_p|\right\} \\
&\Leftrightarrow \min\left\{\sum_{p\in S_p}\sum_{q\in S_q} b_{p,q}\cdot D(p,q) + |S_p|\right\}, \qquad (1)
\end{aligned}$$

where

$$b_{p,q} = \begin{cases} 1, & \text{if } p,q \text{ are in the same group,} \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

$$D(p,q) = R[V_{res}(q)] = 1 - R[V_{com}(q)]. \qquad (3)$$

Equation (1) demonstrates our objective to minimize the sum of total number of pixels transmitted P_{total} for all views. S_p and S_q represent the set of *primary views* and *secondary views*, respectively. $P[V_{res}(q)]$ and $P[V_{com}(q)]$ represent the number of residual and common pixels within *secondary view* q, respectively. Note that for each group, there is one user with *primary view* and the rest of the users with *secondary views*. For each *secondary view*, only one *primary view* is corresponding to calculate $P[V_{com}(q)]$ and $P[V_{res}(q)]$. $P[V(p)]$ denotes the number of pixels within *primary view* p.

Table 4. Notations Used to Build Metrics

Notation	Meaning
xVL	length of front wall
yVL	length of side wall
$VLength$	metric to represent the range of a view in x-axis
x_i, y_i	location for user i in x-axis and y-axis
Δx	distance between a user and left side wall in x-axis
Δy	distance between a user and front wall in y-axis
cVL	metric to evaluate the common ratio between two views
$cnVL$	normalized form of cVL
$xfactor$	length of front and side wall within both views
$yfactor$	squared ratio of distance to the front wall for both views
RmW	room width
k, b	parameters in linear regression

Moreover, we define the residual pixel ratio of a *secondary view* q as

$$R[V_{res}(q)] = \frac{P[V_{res}(q)]}{P_{frame}},$$

and the common pixel ratio of a *secondary view* q as

$$R[V_{com}(q)] = \frac{P[V_{com}(q)]}{P_{frame}},$$

where P_{frame} is the number of pixels per frame. Since the number of pixels in *primary view* p is equal to P_{frame}, the pixel ratio $R[V(p)] = 1$. $|S_p|$ indicates the number of *primary views*.

In Equations (2) and (3), the elements of binary indicator matrix B and distance matrix D are defined as $b_{p,q}$ and $D(p, q)$, respectively. Specifically, the value of $D(p, q)$ equals the residual pixel ratio of a *secondary view* q when the *primary view* p is selected. The larger the $D(p, q)$ is, the more the difference between view p and view q is.

4.3 Metrics and Evaluation

In our approach, to assign multiple users to different groups and minimize the sum of bit rate across multiple users, we want to develop a strategy to group the views that have a lot of common parts together. To make the grouping technique fast, we need to avoid the time-consuming process of conducting common view extraction between all pairs of different views. Instead, we develop and use an easy to calculate metric to represent how much is common between two views; we term this metric as *common normalized VLength (cnVL)*, which we define next. Though we consider the virtual classroom application to develop the metric below; the same definition will be applicable to other virtual spaces like virtual gallery.

The basic notation used in our proposed metrics is provided in Table 4. To be specific, we define $VLength$ as the sum of xVL and yVL, as shown in Equation (4). In this definition, xVL and yVL denote the length of the front wall and side wall within the current view, respectively. Figure 6(a) illustrates from the top of the classroom an example of xVL and yVL with the view of seat #9. Figure 6(b) illustrates from the side of the classroom another example of xVL and yVL of one student view, which is shown as the first view on the right. Figure 6(b) also presents four actual student views from four different positions in the second row of the classroom. We can also see the position of the far plane, which exceeds the wall of virtual classroom. $VLength$ also indicates

(a) Looking from the top.

(b) Looking from the side.

Fig. 6. The model of the virtual classroom and camera view when looking from the top and side. Lengths of *xVL* and *yVL* are shown. (a) presents the boundary of the classroom, seat positions as well as a demonstration of users' views. (b) exhibits views (on the right) as illustrations for student views of four different positions on the second row.

Table 5. Four Different Cases

Case	a	b	c	d
Left wall visible	✓			✓
Front wall visible	✓	✓	✓	✓
Right wall visible			✓	✓

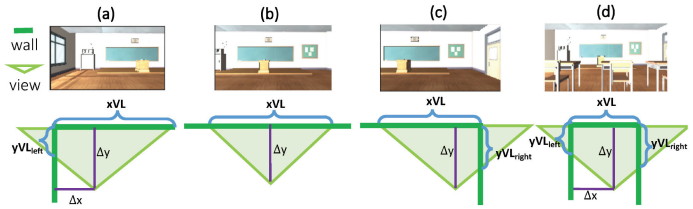

Fig. 7. (a)–(d) show four types of relative positions between classroom boundary and view boundary. And *RmW* denotes the width of the classroom in *x*-axis.

the range of view in *x*-axis and is defined in Equation (4):

$$VLength = xVL + yVL. \tag{4}$$

To calculate *VLength*, we summarize four different types of relative positions between classroom boundary and view boundary, as shown in Table 5. Figure 7 also shows these four types of relative positions as in cases (a)–(d). Especially, we note that for case (d), *VLength* is obtained separately according to *yVL* from the left wall (yVL_{left}) and right wall (yVL_{right}), respectively, as shown in Equation (5):

$$VLength = xVL + yVL_{left} + yVL_{right}. \tag{5}$$

We define the *Common VLength* (cVL) as a metric to evaluate the common ratio between two views *p* and *q*. Equation (6) describes *cVL*, with *xfactor* and *yfactor* defined in Equations (7) and (8), respectively. *xfactor* is defined as the length of the front and side walls within both views. Specifically, we assign two users with subscripts 1 and 2 to distinguish them. Subscript 1 represents the user on the left while subscript 2 denotes the user on the right. Condition 1 indicates that two views are of case (a) and (c), respectively; condition 2 indicates that both views are of case (b); and condition 3 indicates all other case combinations. *yfactor* is defined as the squared ratio of the

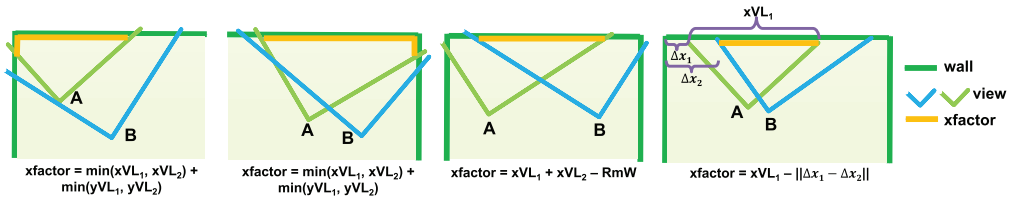

Fig. 8. Four cases of two user views, where the yellow line denotes $xfactor$.

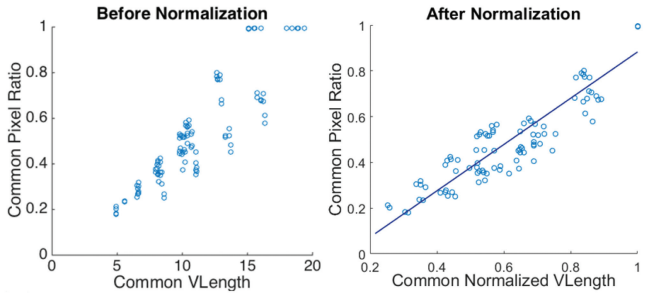

Fig. 9. Validation of model parameters, showing the relationship between the common pixel ratio and *cVL* as well as *cnVL*.

distance to the front wall for both views.

$$cVL(p,q) = xfactor \cdot yfactor, \tag{6}$$

where

$$xfactor = \begin{cases} xVL_1 + xVL_2 - RmW \quad \text{if condition 1;} \\ xVL_1 - |\Delta x_1 - \Delta x_2| \quad \text{if condition 2;} \\ \min(xVL_1, xVL_2) + \\ \min((yVL_{left})_1, (yVL_{left})_2) + \\ \min((yVL_{right})_1, (yVL_{right})_2) \\ \text{otherwise (condition 3);} \end{cases}, \tag{7}$$

$$yfactor = \begin{cases} (\Delta y_1/\Delta y_2)^2, \text{ if } \Delta y_1 < \Delta y_2 \\ (\Delta y_2/\Delta y_1)^2, \text{ if } \Delta y_1 \geq \Delta y_2 \end{cases}. \tag{8}$$

Figure 8 shows four cases of two user views (for users A and B), where the yellow line denotes *xfactor*. The proposed metric *cVL* is used to represent common pixel ratio between two views, and our approach can handle arbitrary viewing direction. For instance, when two users have small or even no common view, the corresponding *xfactor* and calculated *cVL* are small or even zero, representing that the common pixel ratio is small. As described in the next section, when a user A's viewing direction may be very different than user B's, user A may be clustered into a different group than user B due to their small common pixel ratio.

To evaluate our proposed metric, we compare *cVL* (Equation (6)) to the common pixel ratio (defined in Section 4.2) for every pair of views in a virtual classroom with a seat pattern of 2x5 (10 views, 100 pairs of views). Each view has a resolution as 1080p. Figure 9(a) illustrates the correlation between *cVL* and the common pixel ratio values for the 100 pairs of views, with an overall

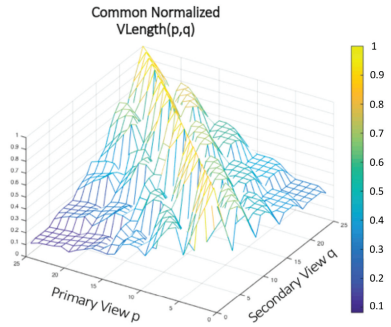

Fig. 10. The *cnVL(p,q)* between every two views in a 5x5 seat pattern, where the p and q represent the primary view index and secondary view index, respectively.

correlation of 0.8828. To increase the correlation with common pixel ratio, we define *common normalized VLength (cnVL)* as follows:

$$cnVL(p,q) = \frac{cVL(p,q)}{cVL(p,p)}. \tag{9}$$

Figure 9(b) illustrates a higher correlation, 0.9207, between *cnVL* (Equation (9)) and the common pixel ratio. We also conduct a linear regression and obtain the results as follows:

$$R[V_{com}(q)] \approx k \cdot cnVL(p,q) + b, \tag{10}$$

where $k = 1.012$ and $b = -0.1291$.

For instance, Figure 10 is a distribution of $cnVL$ in a virtual classroom with a 5x5 seat pattern (shown in Figure 6(a)); the 25 students (views) are indexed from 1 to 25, p refers to primary view and q is the secondary view. In Figure 10, we can see that the distribution of $cnVL(p,q)$ reflects the distribution of common ratio between two views p and q. Specifically, if two views are captured in closer positions, the $cnVL$ tends to be larger; otherwise, $cnVL$ will decrease. Within these 25 views, $cnVL$ attains 1 when views p and q are selected as the same view, while the minimum is attained as 0.1 when *primary view* and *secondary view* are assigned as view #25 and view #1, respectively.

The calculation of $cnVL$ values is much simpler and faster compared to obtaining the actual *Common Pixel Ratio* between every two views using graphic rendering. Therefore, due to the high correlation between $cnVL$ and common pixel ratio, we can approximate the common pixel ratio with the metric $cnVL$, which greatly saves runtime. Subsequently, we update the calculation of $D(p,q)$ in our problem formulation (Equation (1)) as follows:

$$\min P_{total} \Leftrightarrow \min \left\{ \sum_{p \in S_p} \sum_{q \in S_q} b_{p,q} \cdot D(p,q) + |S_p| \right\}, \tag{11}$$

where

$$\begin{aligned} D(p,q) &= R[V_{res}(q)] \\ &= 1 - R[V_{com}(q)] \\ &\approx 1 - (k \cdot cnVL(p,q) + b). \end{aligned} \tag{12}$$

In this way, we can use the newly defined metric $cnVL$ to estimate the common pixel ratio $R[V_{com}]$ between any pair of views. We can also denote the residual pixel ratio as $D(p,q)$ between the *primary view* p and the *secondary view* q, and calculate D based on metric $cnVL$ as Equation (12).

4.4 Grouping Strategy

In a virtual space with a large number of users, if we have all the users in the same group (i.e., one primary view and all others have secondary views), then the number of residual pixels needed may become very large, with some residual views almost equaling the size of the primary view. On the other hand, if we divide users into too many groups (e.g., each one as a unique group), then the size of residual views may greatly decrease (e.g., even to 0) but the number of common pixels may become very large (e.g., even equal to the total pixel number for all views), thus leading to an overall high bit rate needed. Hence the challenge is to identify the most appropriate partitioning (groups) of the users of the virtual space so the overall bit rate needed to transmit their views is minimized (Equation (11)). For example, for better bit rate reduction, users should be grouped with a large portion of shared pixels so that common view can be maximized and residual views can be minimized.

Since finding the optimal groups to minimize the overall bit rate (Equation (11)) is NP-Hard, we first propose a heuristic algorithm $VS-GRP$ (presented in Algorithm 1), which we show in Section 5.4 to have linear complexity in terms of number of users. So that we can evaluate the performance of $VS-GRP$, we subsequently present an optimal but exponential time complexity algorithm $VS-OPT$ (shown in Algorithm 2), and compare their performances in Section 5.

We next describe the steps of Algorithm $VS-GRP$. In Lines 2–11, we make an implement to take the minimum value of evl^* (i.e., the value of P_{total} in Equation (11)), traversing all potential optimized grouping strategies for K groups for I_w iterations. In Lines 4–5, we use the procedures *Kmeans* and *Evaluate* to obtain optimized groups and calculate their value of evl^*. The *Kmeans* procedure consists of three parts: *Initialize*, *Optimization* and *Update*.

We use the $Initialize(k, i_w)$ procedure to initialize k cluster centers for k groups (i.e., each group has a cluster center). In iteration number $i_w = 1$, we will give a priority index to each user. The numbering method is giving the user with smaller y a smaller priority index, and if two users have equal y, we give the user with smaller x a smaller priority index. In this way, each user will be numbered with a priority index, from 1 to n. In our algorithm, we initialize k cluster centers as $\lfloor \frac{n}{k} \rfloor, \lfloor \frac{2n}{k} \rfloor, \ldots, n$. After that, we will carry out the Optimization and Update procedures. Otherwise, if iteration number $i_w \neq 1$, we initialize k cluster centers randomly by picking k different users from n users.

Then, in Lines 3–11 of *Kmeans* procedure, we implement the optimization by updating the users with *primary* and *secondary views* continuously until the set of primary views S_p and the set of secondary views S_q do not change anymore. The number of the iterations in Lines 3–11 are denoted as I_u, which will be discussed further in Section 5.4. In *Optimization* procedure (Lines 5–7 of *Kmeans* procedure), we will find out which view p in the set of primary views S_p has the minimum $D(p, q)$ and then select it as the *primary view* within this group. In *Update* procedure, we will update the S_p, S_q by assigning the nearest one to the average location of all group members as the user with *primary view*.

In Lines 2–6 of *Evaluate* procedure, we calculate the value of evl^* (i.e., the value of P_{total} in Equation (11)) by adding up each item in Equation (11). With all these procedures, we can finally obtain the optimized grouping strategy with our algorithm.

As stated earlier, since algorithm $VS-GRP$ is heuristic, so that we can evaluate its performance, we also present an optimal algorithm $VS-OPT$, which considers forming all possible groups by using all possible assignments of primary views and secondary views in an exhaustive manner. Then by comparing the value of P_{total} in Equation (11) for every possible assignment and finding out the groups for which is minimum, we can obtain the optimal groups. Specifically, we present this algorithm with MATLAB implementation. The function $combntns(set, n_0)$ returns a matrix whose rows are the various combinations that can be taken of the elements of the vector *set*

ALGORITHM 1: VS-GRP Algorithm

Inputs: Number of users n, locations (x_i, y_i) for each user i and distance matrix D.
Output: Binary indicator matrix B, such that bit rate needed for all user views is minimized (Equation (11)).

$evl \leftarrow +\infty$
for $k = 1 : K$ **do**
  for $i_w = 1 : I_w$ **do**
    $B^* \leftarrow Kmeans(k, i_w)$
    $evl^* \leftarrow Evaluate(B^*)$
    if $evl > evl^*$ **then**
      $B \leftarrow B^*$
      $evl \leftarrow evl^*$
    end if
  end for
end for
return B

Procedure $Kmeans(k, i_w)$:

$S_p, S_q \leftarrow Initialize(k, i_w)$
$S_p^* \leftarrow [\,]$
while $S_p^* \neq S_p$ **do**
  $B^* \leftarrow zeros(n, n)$
  for $q \in S_q$ **do**
    $p \leftarrow \arg\min_p \{D(p, q), p \in S_p\}$
    $b^*_{p,q} \leftarrow 1$
  end for
  $S_p^* \leftarrow S_p$
  $S_p, S_q \leftarrow Update(B^*)$
end while
return B^*

Procedure $Evaluate(B^*)$:

$evl^* \leftarrow 0$
for $p \in S_p$ **do**
  for $q \in S_q$ **do**
    $evl^* \leftarrow evl^* + (b^*_{p,q} \cdot D(p, q) + k)$
  end for
end for
return evl^*

of length n_0 while the function $setdiff(A, B)$ returns the data in A that is not in B, with no repetitions. In this way, we can do a traversal of all possible combinations of users assigned with primary view or secondary view. For every possible combination, we will calculate its value of evl^*. Finally, we will obtain the user's assignment with the minimum evl^*, which is the optimal strategy for grouping.

Next, we briefly analyze the time complexity of the optimal exhaustive algorithm $VS - OPT$ as well as our proposed grouping algorithm $VS - GRP$. Let n be the number of users in the virtual space, and k be the desired number of groups. For the optimal exhaustive algorithm $VS - OPT$,

ALGORITHM 2: VS-OPT Algorithm

Inputs: Number of users n, locations (x_i, y_i) for each user i, and distance matrix D.
Output: Binary indicator matrix B, such that bit rate needed for all user views is minimized (Equation (11)).

$evl \leftarrow +\infty$
$v \leftarrow [1:n]$
for $k = 1:n$ **do**
  $S_{s_p} \leftarrow combntns(v, k)$
  $n_{comb} \leftarrow size(S_{s_p}, 1)$
  for $i = 1:n_{comb}$ **do**
    $B^* \leftarrow zeros(n, n)$
    $S_p \leftarrow S_{s_p}(i, :)$
    $S_q \leftarrow setdiff(v, S_p)$
    for $j = 1:(n-k)$ **do**
      $q \leftarrow S_q(j)$
      $p \leftarrow \arg\min_p \{D(p, q), p \in S_p\}$
      $b^*_{p,q} \leftarrow 1$
    end for
    $evl^* \leftarrow Evaluate(B^*)$
    if $evl > evl^*$ **then**
      $B \leftarrow B^*$
      $evl \leftarrow evl^*$
    end if
  end for
end for
return B

there are C_n^k possibilities of choosing k primary views (for k groups). After deciding k users with primary views, we have $(n-k)$ users to be assigned with secondary views. For each of the unassigned users, there are k possible choices (each view can be assigned to either of the k groups), leading to a multiplicative factor of k^{n-k} for each C_n^k. Therefore, the time complexity of the optimal exhaustive algorithm $VS-OPT$ is $O(C_n^k k^{n-k})$.

In contrast, our proposed grouping algorithm has a computation complexity of $O(nKI_u)$, where n is the number of users, I_u is the number of iterations in *Kmeans* procedure, and K is the maximum number of groups we traverse. Next, we empirically estimate upper bound for K and I_u to validate our complexity analysis in Section 5.4.

5 EXPERIMENTAL RESULTS

In this section, we describe our experimental setup and results. We use an Intel Core i7 Quad-Core processor with 32GB RAM and implement our approach in MATLAB. We demonstrate the performance of our grouping algorithm ($VS-GRP$) with our proposed evaluation metrics side by side to an optimal, exhaustive grouping algorithm ($VS-OPT$). We demonstrate experiments by using (i) the virtual classroom application with a regular student's seat (view) pattern and (ii) the virtual gallery application with the location of visitors (views) randomly distributed. To further demonstrate the robustness, we also show experiments for a virtual classroom, with vacant seats (views). Then we present a complexity analysis, and a congestion-related latency study in this

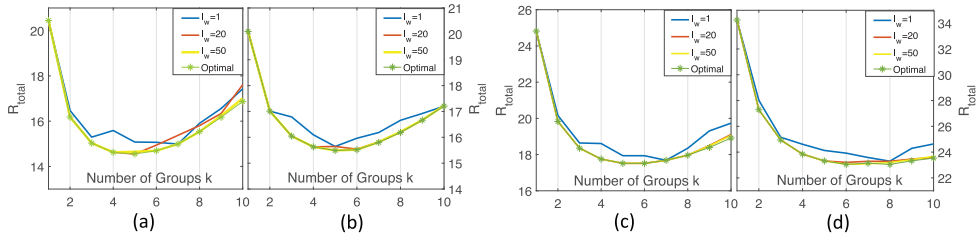

Fig. 11. Total number of transmitted pixels divided by number of pixels in one frame, versus number of groups k for four different seats pattern virtual classrooms. Sub-graphs present seat patterns of (a) 7x5, (b) 5x7, (c) 7x6, and (d) 7x7, respectively.

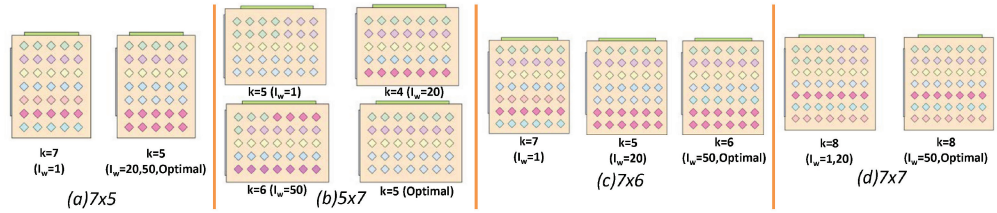

Fig. 12. Grouping results of $I_w = 1, 20, 50$, and *Optimal* cases for four different seat pattern virtual classrooms. Sub-graphs (a)–(d) present seat patterns of 7x5, 5x7, 7x6, and 7x7, respectively.

section. Note that we select virtual classroom and virtual gallery applications to validate the effectiveness of our approach, since they have different characteristics: the users in the classroom have a regular distribution pattern while the users in the gallery are more randomly distributed in the virtual space. For the virtual classroom, we explore two scenarios, without and with vacant seats, since vacant seats can affect the user distribution pattern and hence the grouping results.

5.1 Virtual Classroom

5.1.1 Without Vacant Seats. To evaluate the effectiveness of our approach, we perform our experiments using the virtual classroom application with different seating patterns and all seats occupied.

We first consider a virtual classroom with a seating configuration having the same horizontal and vertical seat spacings ($Spacing_V = Spacing_H = 2$). Figure 11 shows the total number of pixels (normalized to frame size) to be transmitted with different number of groups. R_{total} is the ratio of total number of pixels to be transmitted over total number of pixels in one frame. For example, in a 20-user scenario, $R_{total} = 14.99$ means that we only need $14.99 \times P_{frame}$ (e.g., 1920x1080 pixels) instead of $20 \times P_{frame}$. We use $I_w = 1, 20, 50$ for our proposed algorithm $VS - GRP$, and compare to the optimal algorithm $VS - OPT$ (*Optimal*), using four different seat patterns—7x5, 5x7, 7x6, and 7x7. For a given number of groups k, we can see that there is a tradeoff between solution quality (total number of pixels to be transmitted) and I_w. Larger I_w benefits the grouping result (closer to optimal R_{total} as well as P_{total}) but also consumes more runtime, as shown in Table 6.

From Figure 11, we can also find a preference for k as well. Too few groups may result in many distinct secondary views, thus there is not much common view. Too many groups may result in too many primary views, leaving insufficient secondary views. Figure 12 shows examples of some grouping results for the four seat patterns. Each diamond denotes a seat and each color represents a unique group. We can see that as I_w increases, our grouping algorithm performs closer to the

Table 6. Experimental Results for Four Different Seat Pattern Virtual Classrooms

Space	*Alg.*	*Runtime*	k	R_{total}	*Pixel Savings*
7x5	$I_w = 1$	0.0035s	7	14.99	57.2%
	$I_w = 20$	0.0707s	5	14.55	58.4%
	$I_w = 50$	0.1660s	5	14.55	58.4%
	Optimal	$>1h$	5	14.55	58.4%
5x7	$I_w = 1$	0.0036s	5	15.65	55.3%
	$I_w = 20$	0.0683s	4	15.63	55.3%
	$I_w = 50$	0.1715s	6	15.56	55.5%
	Optimal	$>1h$	5	15.49	55.7%
7x6	$I_w = 1$	0.0044s	7	17.68	57.9%
	$I_w = 20$	0.0774s	5	17.55	58.2%
	$I_w = 50$	0.1917s	6	17.51	58.3%
	Optimal	$>1h$	6	17.51	58.3%
7x7	$I_w = 1$	0.0046s	8	23.30	58.4%
	$I_w = 20$	0.0950s	8	23.30	58.4%
	$I_w = 50$	0.2349s	8	23.20	58.6%
	Optimal	$>1h$	8	23.20	58.6%

optimal solution. For instance, Figure 12(a) shows that by using $I_w = 1$, we obtain a grouping strategy (dividing users into 7 groups) while by employing $I_w = 20, 50$ or optimal algorithm $VS - OPT$, we receive the same grouping result (five groups). Similarly, we present the grouping results for the other three seating patterns in Figure 11(b)–(d), respectively.

Table 6 summarizes the quality of results in terms of runtime, R_{total}, and pixel savings for the four seat patterns. Our proposed approach consumes up to 0.24s for grouping, compared to 1 hour of the optimal exhaustive grouping algorithm, while giving at most 0.2% degradation of bit rate savings when $I_w = 50$. For a more real-time-critical scenario, we only need 5ms for grouping when $I_w = 1$, with a maximum degradation of 1.2% in pixel savings. Overall, our proposed approach can achieve more than half of the pixel savings for all four seat patterns within milliseconds. To be specific, as seen from Table 6, our proposed algorithm $VS - GRP$ (such as $I_w = 1$) with hybrid-cast approach reduces the pixels needed by 57.2%, 55.3%, 57.9%, and 58.4% compared to the conventional approach of transmitting all the 7x5, 5x7, 7x6, and 7x7 views as individual unicast streams. Also, our proposed algorithm $VS - GRP$ is able to reduce the pixels needed only by a marginal 1.2%, 0.4%, 0.4%, and 0.2% less than the optimal algorithm $VS - OPT$, while ensuring that it can be run in real time (runtime of 0.0040 seconds on average when $I_w = 1$) compared to more than an hour of runtime for the optimal algorithm. The results when $I_w = 20, 50$ are really similar to optimal results but the runtime will be a little more than the runtime when $I_w = 1$.

Since the above grouping results present a tendency for horizontal grouping (Figure 13), we then perform experiments with a virtual classroom with different seating configuratiosn, where the spacing between vertical seats is less than the spacing between horizontal seats ($Spacing_V = 1$, $Spacing_H = 2$). Figure 13(a) and (b) show grouping results for two seat patterns (6x7 and 7x7), respectively. Each diamond denotes a seat and each color represents a unique group. We can observe that for the new seat configuration, our proposed grouping algorithm produces more groups in vertical direction, and, in general, groups consisting of both horizontal and vertical neighbors.

Besides showing the real-time performance of our proposed grouping algorithm, we analyze the end-to-end latency consumed for our cloud-based virtual space approach in Appendix B. Our

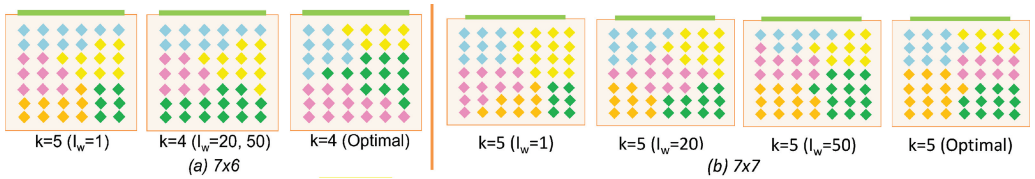

Fig. 13. Grouping results of $I_w = 1, 20, 50$ and *Optimal* cases for two different seat pattern virtual classrooms. The sub-graphs (a), (b) present seat patterns of 7x6 and 7x7, respectively.

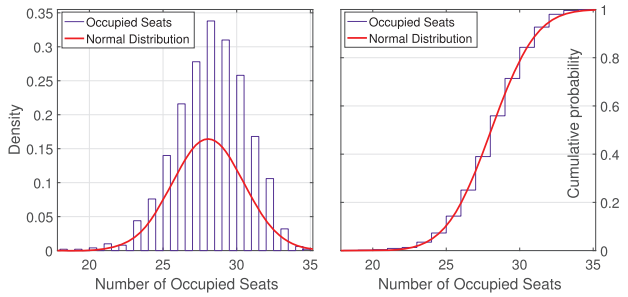

Fig. 14. The probability density function figure (*left*) and cumulative distribution function figure (*right*) of occupied seats covering all 1,000 samples. The red line shows the normal distribution fitting.

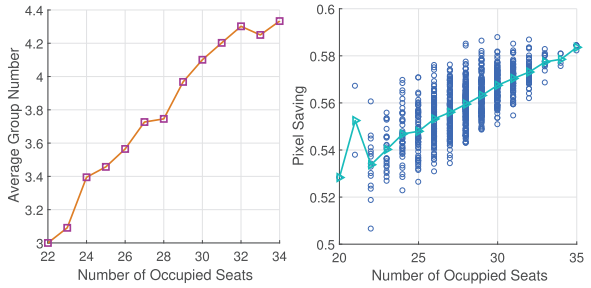

Fig. 15. The left sub-graph shows the average number of groups versus number of occupied seats while the right sub-graph demonstrates the pixel savings versus the number of occupied sets. Specifically, in the latter, the green line represents the average pixel savings.

analysis shows server-side latency of 9.5–19.5ms and client-side latency of about 5.5ms, per rendered frame.

5.1.2 With Vacant Seats. To demonstrate the robustness of bit rate savings, we consider a more irregular seat pattern in the virtual classroom. In our experiment, we use a 7x5 seat pattern, and each seat has a vacant probability of 20%. (This scenario can match to a virtual class where each student may choose to drop the class with a fixed probability.) We randomly generate 1,000 different vacancy patterns. Figure 14 shows the probability density function and cumulative distribution function for the 1,000 vacancy patterns. For each vacancy pattern, we apply our proposed grouping algorithm $VS-GRP$. Figure 15 shows the results in terms of the number of groups and pixel savings obtained. The empirical results demonstrate that our algorithm performs well with vacant seats. We have achieved similar pixel savings ratio compared to a virtual classroom without vacancy.

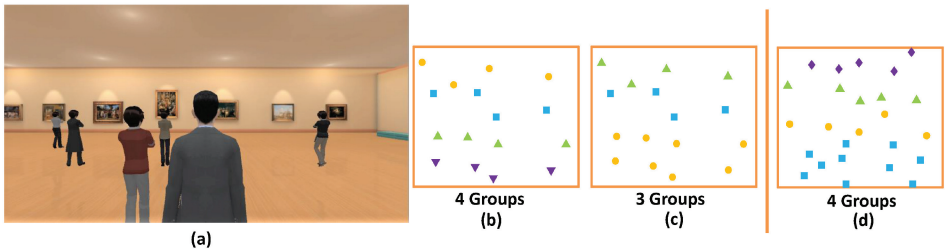

Fig. 16. A virtual gallery scene where the visitors are randomly distributed is demonstrated in (a), grouping results for 16 users are presented in (b), (c) while results for 25 users are shown in (d).

Table 7. Experimental Results for Multiple Users in Gallery Scene

Scene	*Alg.*	k	*Pixel Savings*	*Runtime*
$Scene_1$	$I_w = 1$	4	55.0%	0.0043s
	$I_w = 20$	3	56.9%	0.0454s
	Optimal	3	56.9%	$>1h$
$Scene_2$	$I_w = 1$	4	56.8%	0.0056s
	$I_w = 20$	4	56.8%	0.0533s
	Optimal	4	56.8%	$>1h$

Interestingly, as shown in Figure 15, the average group number grows approximately linearly with the increase in the number of occupied seats. Also, we can achieve more pixel savings with more occupied seats, while the exact location of those vacancies does not impact much the pixel savings. Overall, we still achieve more than half of pixel savings for all configurations, indicating the robustness of our algorithm. Specifically, in the right sub-figure of Figure 15, the x-axis is the number of occupied seats and y-axis is the pixel savings. Every blue point represents the number of occupied seats and the pixel savings achieved by using our proposed approach. The green line and the triangle point within it represent the average pixel savings corresponding to the number of occupied seats. For example, the point (20, 0.53) indicates we can achieve approximately 53% pixel savings to transmit all 20 views (for 20 occupied seats) with our proposed approach. It is demonstrated that the pixel savings will increase linearly as the number of occupied seats increases. We can observe that the pixel savings will not be affected by the position of vacant seats, but rather the number of vacant (as well as occupied) seats.

5.2 Virtual Gallery

Next, we show results of applying our approach to the virtual gallery application. In the virtual gallery shown in Figure 16(a), visitors (views) are randomly distributed. We conduct two sets of experiments, with 16 and 25 users, respectively. Two different grouping results for 16 users are presented in Figure 16(b) and (c), while one grouping result for 25 users is shown in Figure 16(d). Table 7 summarizes the experimental results in terms of the parameters, pixel savings obtained, and runtime, compared to the optimal algorithm $VS - OPT$. We can see that for 16 users ($Scene_1$), our proposed algorithm $VS - GRP$ ($I_w = 1$) can obtain pixel savings of 55.0%, with only 1.9% degradation compared to the optimal result, while consumes only 4.3ms. The corresponding assignment for $I_w = 1$ and $k = 4$ is shown in Figure 16(b). The grouping assignments for $I_w = 20$ and *Optimal* are the same, and are demonstrated in Figure 16(c). For 25 users ($Scene_2$), the grouping assignments

Table 8. High Correlation Between Savings in Bit Rates and Savings in Pixel Ratio

Space	*Items*	*Conv. Approach*	*Prop. Approach*	*Savings*
VC	*Total Bit Rate*	43.7Mbps	26.7Mbps	39.0%
	Avg. Pixel Ratio	2.00	1.18	41.0%
VG	*Total Bit Rate*	27.6Mbps	23.3Mbps	15.7%
	Avg. Pixel Ratio	2.00	1.63	18.5%

for $I_w = 1, 20$ and *Optimal* are the same, shown in Figure 16(d). The above results again demonstrate the effectiveness of our proposed grouping algorithm in a virtual space application with randomly distributed views. Overall, the pixel savings are similar compared to a virtual classroom application, while suitable for real-time-critical applications.

5.3 High Correlation Between Pixel Ratio and Bit Rate Savings

In this experiment, we validate the correlation between pixel ratio and bit rate savings. In our hybrid-cast approach, the total bit rate needed for the video streams is different from the raw pixel savings in that video streams are encoded in fixed frame size, and then transmitted. Note that, in our case, this is equivalent to exploring whether there is high correlation between pixel ratio savings and bit rate savings. The reason is that in our experiments, we obtain pixel ratio and bit rate savings for two approaches (proposed approach and conventional approach) with the same setting of video resolution and frame rate. The pixel ratio savings equals pixel savings divided by number of pixels in one frame (e.g., 1920x1080 pixels) while the bit rate savings equals the savings in number of bits times the value of frame rate. Thus, high correlation (between pixel ratio savings and bit rate savings) and high correlation (between pixel savings and number of bit savings) are two equivalent statements in our discussion.

To demonstrate the above, we perform live experiments for the two virtual space applications. For the virtual classroom application, we assume that there are two students in the same row, sitting close to each other, and a teacher is walking fast at the front of the classroom. The view centers of the students keep moving and are always towards the teacher, i.e., focusing continuously with movement of the teacher. We record 750 frames (30s assuming 25fps). Then we encode the video using H.264 after applying our approach. In our implementation, we only need to broadcast primary views and unicast residual views (instead of secondary views). Residual views are encoded full frame size, with common pixels replaced by black pixels. (Thus, we don't explore any further pixel savings by downscaling.) We also perform the live experiment using the virtual gallery application. In our setup, parameters are set the same except that two visitors stand farther away from each other, and both watch a slowly moving tour guide.

Table 8 summarizes the experimental results for the above two scenarios, in terms of the average pixel ratio of the rendered frames, the total bit rate needed for the resulting encoded video frames, and the savings achieved by using the proposed approach for both the average pixel ratio and the total video bit rate. For the virtual classroom application, the savings in average pixel ratio is 41.0% while the savings in the total bit rate is 39.0%. For the virtual gallery application, our proposed approach produces 18.5% savings in average pixel ratio while the savings in total video bit rate is 15.7%. The results show that there is a high correlation between pixel ratio savings and bit rate savings for both the virtual space applications. And as shown in Appendix A, the bit rate savings using our approach also leads to substantial savings of cloud costs.

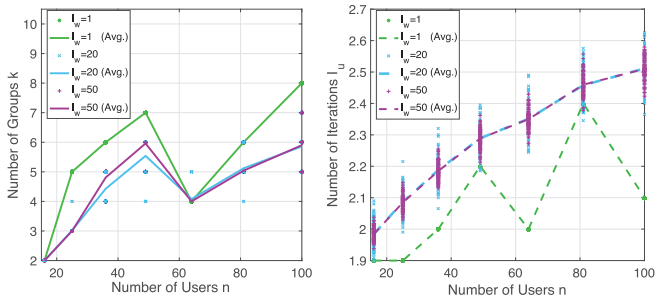

Fig. 17. (a) Number of groups versus the number of users. The line represents the average number of groups given the number of users. (b) Number of iterations as I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users.

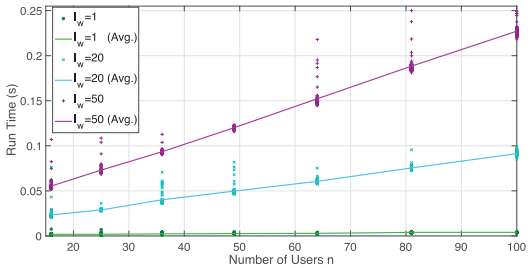

Fig. 18. The runtime versus the number of users when $I_w = 1, 20, 50$, respectively. The line represents the average runtime given the number of users.

5.4 Empirical Results Validating Complexity Analysis

As is mentioned in Section 4.4, our proposed grouping algorithm has a computation complexity of $O(nKI_u)$, where n is the number of users, I_u is the number of iterations in *Kmeans* procedure, and K is the maximum number of groups we traverse. Next, we empirically estimate upper bound for K and I_u, and show that the complexity of our proposed algorithm can be practically simplified as $O(n)$. We perform the experiments using the virtual classroom application, with seat patterns setting from 4x4 to 10x10. For a given seat pattern, we generate 100 different seat pattern configurations and implement our grouping algorithm. We also calculate the average of parameters (i.e., k, I_u, runtime) for these seat pattern configurations.

Specifically, Figure 17(a) demonstrates the optimal number of groups from our algorithm versus total number of users. The three lines represent the optimal, average number of groups across all seat patterns. We observe that the number of groups show similar trends and are generally smaller than $\sqrt{n}$, across all seat patterns. In our implementation, we empirically bound the maximum number of groups $K = 10$.

Figure 17(b) shows the number of iterations I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users. We can see an increase in I_u with the increase in n when the $I_w = 20$ and 50, and similarly for $I_w = 1$ with some fluctuations. The results indicate a steady, if not increasing I_u.

In addition, we explore the relation between the runtime and the total number of users. Figure 18 shows the runtime versus the number of users when $I_w = 1, 20, 50$. The lines represent the average runtime across all seat patterns. The results show that the average runtime grows linearly with the total number of users, i.e., $O(nKI_u)$ can be simplified as $O(n)$ in that (i) K is bounded to be 10 and

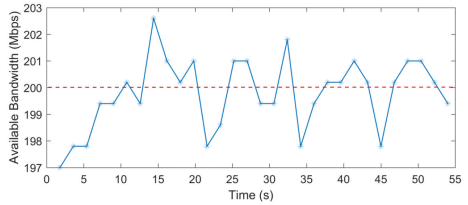

Fig. 19. The available bandwidth of the network in our experiment.

Table 9. Congestion-Related Latency in Bandwidth-Limited Network

Approaches	***Settings***	***Latency***
Conv. Approach	10 users (bit rate: 23Mbps*10)	Min≈600ms Max>1s
Prop. Approach	10 users (bit rate: 11Mbps*9+23Mbps)	Min<1ms Max≈5ms

(ii) I_u is practically a small constant. Overall, our proposed algorithm $VS-GRP$ can be executed in real time with linear time complexity, with similar solution quality to the optimal algorithm, while consuming negligible runtime. As presented in Appendix C, we also perform experiments on the virtual gallery application to validate our complexity analysis.

5.5 Congestion-Related Latency

In this experiment, we show the congestion-related latency improvement by utilizing our hybrid-cast approach. For the server side, we deploy our model in an Ubuntu 16.04 TLS system hosted on the Amazon Web Service (AWS) [8] server, equipped with a 2.6GHz Intel Xeon processor, 16GB RAM, and a NVIDIA GRID GPU. For the client side, we simulate a 10-user scenario by deploying to 10 nodes, each with the same, above-mentioned machine configuration. We assume that there is only one group (i.e., one user with a primary view and nine users with secondary views). We use DummyNet [10] to emulate the wireless network, specifically network bandwidth profiles experienced by the virtual space data transmitted from the AWS server.

To measure the latency from AWS cloud server to the user's client, we performed experiments to record the round-trip delay needed with different network bandwidth profiles. Figure 19 shows a fluctuating bandwidth profile (for 55s), with an average available bandwidth of approximately 200Mbps. We emulate two cases: using conventional method (all user views unicast) and proposed method (one group with one primary view broadcast and nine residual views unicast). We record a 55s video of one user view (23Mbps) and a 55s video of its corresponding residual view (11Mbps) separately. We assume that within the 55s period, a user will receive approximately 11Mbps video as residual view with our proposed hybrid-cast approach while the user needs to receive 23Mbps with the conventional method. Table 9 reports the latency measured under these two different settings. In the setting using the conventional method, since the realistic bit rate needed is larger than available bandwidth, the latency varies from a low of around 600ms to a high of larger than 1s. However, with our proposed hybrid-cast approach, the bandwidth needed is significantly decreased and thus the latency achieved is much smaller (i.e., <5ms).

Note the above analysis assumes all users are associated with the same base station (or at least the same cellular gateway) and cloud server. Hence the bandwidth savings using our approach can help in reducing the congestion-related latency. However, as described in Section 3.2, if the

users are not associated either with the same base station or the same cellular network gateway, latency reduction may not be achievable using the proposed approach. With the above assumption, the results demonstrate the significant advantage our proposed hybrid-cast approach may have to alleviate congestion-related latency in fluctuating and bandwidth-limited wireless networks.

6 CONCLUSION

In this article, we propose a multi-user hybrid-cast approach to significantly reduce the total bit rate needed to stream high-quality videos to multiple users in a virtual space application. Instead of unicasting the video of each user view, we introduce the novel approach that allows unicasting much lower-bandwidth residual views, together with one or more common view(s). Then we propose an efficient way of identifying common and residual views. To minimize the total bit rate, we develop a smart real-time algorithm for grouping the users of the virtual space, using a novel grouping metric. Our experimental results demonstrate the effectiveness of our proposed grouping algorithm both in terms of optimal performance and speed. Furthermore, the results show that the total bit rate needed to transmit multiple user views can be significantly reduced by up to 55%, and thus provide better user experience (less delay) under a constrained network.

Our future research interests include: (i) integrating our hybrid-cast approach with a real network (e.g., Wi-Fi or cellular); (ii) studying data routing, forwarding and related protocols for data transmission in hybrid-cast approach; (iii) considering more complex virtual spaces, including irregular shapes and topologies; and (iv) analyzing the case of having multiple primary views in the same group and the corresponding performance benefit.

REFERENCES

[1] ISO/IEC JTC 1. 1993. Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s-Part 2: Video. ISO/IEC 11172-2 (MPEG-1) (1993).
[2] ISO/IEC JTC 1. 1999. Coding of audio-visual objects—Part 2: visual. ISO/IEC 14496-2 (MPEG-4 Vis. Version 1) (Apr. 1999).
[3] ITU-T ISO/IEC JTC 1. 1994. Generic coding of moving pictures and associated audio information part 2: Video. ITU-T Rec. H.262 ISO/IEC 13818-2 (MPEG-2 Video) (Nov. 1994).
[4] ITU-T ISO/IEC JTC 1. 2003. Advanced video coding for generic audio-visual services. ITU-T Rec. H.264 ISO/IEC 14496-10 (AVC) (May 2003).
[5] Song Ho Ahn. 2018. OpenGL projection matrix. Retrieved 2018 from http://www.songho.ca/opengl/gl_projectionmatrix.html.
[6] Song Ho Ahn. 2018. OpenGL transformation. Retrieved 2018 from http://www.songho.ca/opengl/gl_transform.html.
[7] Amazon. 2018. Amazon EC2 pricing. Retrieved 2018 from https://aws.amazon.com/ec2/pricing/on-demand.
[8] Amazon. 2018. AWS. Retrieved 2018 from https://aws.amazon.com.
[9] Wei Cai and Victor C. M. Leung. 2012. Multiplayer cloud gaming system with cooperative video sharing. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE, Taipei, 640–645.
[10] Marta Carbone and Luigi Rizzo. 2010. Dummynet revisited. *ACM SIGCOMM Computer Communication Review* 40, 2 (2010), 12–20.
[11] Recommendation H.261 CCITT. 1990. Video codec for audiovisual services at px 64 kbit/s. *Draft Revision* (Mar. 1990).
[12] Recommendation H.263 CCITT. 1995. Video coding for low bit rate communication. *Draft* (Nov. 1995).
[13] Aleksandra Checko, Henrik L. Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S. Berger, and Lars Dittmann. 2015. Cloud RAN for mobile networks—A technology overview. *IEEE Communications Surveys & Tutorials* 17, 1 (2015), 405–426.
[14] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. 2013. A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Communications and Mobile Computing* 13, 18 (2013), 1587–1611.
[15] edX. 2018. MOOC. Retrieved 2018 from https://www.edx.org.
[16] Cass Everitt. 2001. Projective texture mapping. White Paper, Nvidia Corporation (April 2001).
[17] Xueshi Hou, Yao Lu, and Sujit Dey. 2016. A novel hyper-cast approach to enable cloud-based virtual classroom applications. In *2016 IEEE International Symposium on Multimedia (ISM'16)*. IEEE, San Jose, CA, USA, 533–536.
[18] HTC. 2018. HTC Vive. Retrieved 2018 from https://www.htcvive.com.

[19] ITU-T and ISO/IEC JTC. 2013. High efficiency video coding. ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC) (Jan. 2013).
[20] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. 2013. Softcell: Scalable and flexible cellular core network architecture. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 163–174.
[21] StarBurst K. Miller, StarBurst K. Robertson, Cisco A. Tweedly, and StarBurst M. White. 1998. StarBurst multicast file transfer protocol (MFTP) specification. Retrieved 1998 from https://tools.ietf.org/html/draft-miller-mftp-spec-03.
[22] David Lecompte and Frédéric Gabin. 2012. Evolved multimedia broadcast/multicast service (eMBMS) in LTE-advanced: Overview and Rel-11 enhancements. *IEEE Communications Magazine* 50, 11 (2012).
[23] Yao Liu, Shaoxuan Wang, and Sujit Dey. 2014. Content-aware modeling and enhancing user experience in cloud mobile rendering and streaming. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 4, 1 (Mar. 2014), 43–56.
[24] Yao Lu and Sujit Dey. 2016. JAVRE: A joint asymmetric video rendering and encoding approach to enable optimized cloud mobile 3D virtual immersive user experience. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 4 (2016), 544–559.
[25] Yao Lu, Yao Liu, and Sujit Dey. 2015. Cloud mobile 3D display gaming user experience modeling and optimization by asymmetric graphics rendering. *IEEE Journal of Selected Topics in Signal Processing* 9, 3 (April 2015), 517–532.
[26] Oculus. 2018. Oculus rift. Retrieved 2018 from https://www.oculus.com.
[27] Oculus. 2018. Performance head-up display. Retrieved 2018 from https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-hud.
[28] OpenGL. 2018. OpenGL. Retrieved 2018 from https://www.opengl.org.
[29] Abhijit Patait and Eric Young. 2018. High performance video encoding with Nvidia GPS. https://goo.gl/Bdjdgm.
[30] 5G-Xcast Project. 2018. PTM. Retrieved 2018 from http://5g-xcast.eu/about.
[31] WebM project. 2018. VP9 video codec. Retrieved 2018 from http://www.webmproject.org/vp9.
[32] Samsung. 2018. Samsung gear VR. Retrieved 2018 from http://www.samsung.com/global/galaxy/gear-vr.
[33] Unity. 2018. Unity. Retrieved 2018 from https://unity3d.com.
[34] Shaoxuan Wang and Sujit Dey. 2013. Adaptive mobile cloud computing to enable rich mobile multimedia applications. *IEEE Transactions on Multimedia* 15, 4 (June 2013), 870–883.
[35] Wikipedia. 2018. AV1. Retrieved 2018 from https://en.wikipedia.org/wiki/AOMedia_Video_1.
[36] Wikipedia. 2018. IP multicast. Retrieved 2018 from https://en.wikipedia.org/wiki/IP_multicast.
[37] Wikipedia. 2018. Pragmatic general multicast. Retrieved 2018 from https://en.wikipedia.org/wiki/Pragmatic_General_Multicast.
[38] Wikipedia. 2018. Real-time transport protocol. Retrieved 2018 from https://en.wikipedia.org/wiki/Real-time_Transport_Protocol.
[39] Wikipedia. 2018. Rendering pipeline. Retrieved 2018 from https://en.wikipedia.org/wiki/Graphics_pipeline.
[40] Wikipedia. 2018. Resource reservation protocol. Retrieved 2018 from https://en.wikipedia.org/wiki/Resource_Reservation_Protocol.
[41] Finn Wong. 2016. Performance analysis and optimization for PC-based VR applications: From the CPU's perspective. *Intel Virtual Reality Documentation*. Retrieved 2016 from https://software.intel.com/en-us/articles/performance-analysis-and-optimization-for-pc-based-vr-applications-from-the-cpu-s.
[42] Yi Xu and Shiwen Mao. 2013. A survey of mobile cloud computing for rich media applications. *IEEE Wireless Communications* 20, 3 (2013), 46–53.

Received August 2017; revised March 2018; accepted April 2018

ONLINE APPENDIX

A CLOUD COST SAVINGS

Based on the experiments described in Section 5.5, we perform two experiments: (i) estimating the savings obtained by our proposed approach in terms of cloud bandwidth and the consequent cloud cost for different number of days, and (ii) calculating total monthly cost for different number of users. Specifically, we estimate the cloud cost charged using conventional approach and proposed approach when the service provider of the virtual space application uses AWS. We experimentally choose to have 10% users with primary views (23Mbps) and 90% users with residual views (11Mbps). We calculate corresponding cloud cost using the AWS pricing model in Table 10 [7].

Figure 20(a) shows the accumulative data transfer and total cost for 10 days (with assumption of 100 users in the virtual space). We can observe that the total cost (denoted by orange lines) increases sublinearly when the accumulative data transfer increases linearly due to segmented pricing in AWS pricing model. The total cost savings are the difference between two orange lines. We can see that the accumulative data transfer reaches around 240TB and 125TB respectively by employing conventional and proposed approaches, for 100 users in 10 days. The corresponding total cost saving is around $5000 for 100 users in 10 days using our proposed approach.

Figure 20(b) presents the total monthly cost versus different number of users. The blue bar and green bar denote the total monthly cost using conventional approach and our proposed approach respectively. We can observe that total monthly cost grows continuously with the increase in the number of users. For 1000 users, the total monthly costs are up to $367,800 and $170,000 respectively using conventional and proposed approaches, which can translate to a substantial cloud cost saving of $2.04M annually for the virtual space service provider. Note the above analysis assumes existence of multicast protocols in the cloud network connecting cloud servers to core network gateways. While such multicast protocols are being researched and developed, the service provider savings discussed in Appendix A will need to wait deployment of the such protocols in the future.

Table 10. Pricing model per month on Amazon Web Service.

Data Transfer	***Price***
First 1GB	$0 per GB
Up to 10TB	$0.09 per GB
Next 40TB	$0.085 per GB
Next 100TB	$0.07 per GB
Next 350TB	$0.05 per GB

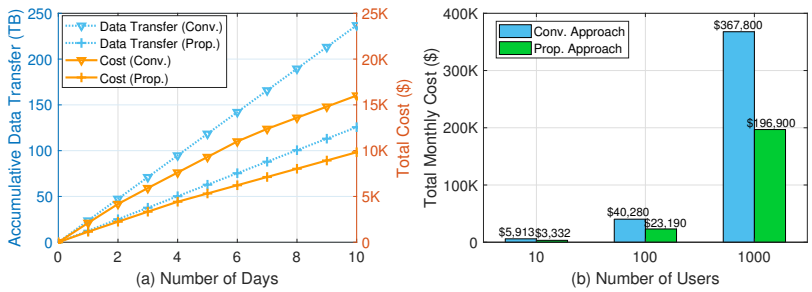

Fig. 20. (a) Accumulative data transfer (left y-axis) and total cost (right y-axis) versus the number of days, for 100 users in the virtual space; (b) Total monthly cost versus different number of users in the virtual space.

B LATENCY ON THE SERVER AND CLIENT SIDES

In this section, we show the latency for tasks on the server side and client side. In terms of latency for the rendering, encoding and decoding tasks, we give the estimated values according to their

Table 11. Latency for procedures on the server side.

Procedures	*Latency*
Rendering	4-9ms
Residual View Calculation	≈2.5ms
Encoding	3-8ms

Table 12. Latency for procedures on the client side.

Procedures	*Latency*
Decoding	≈ 3ms
Synthesis	≈ 2.5ms

current advanced implementations [27, 29, 41]. Then we measure the latency for residual view calculation and synthesis by ourselves.

Table 11 and Table 12 present the latency for the various tasks performed on the server side and client side respectively, besides the latency of our proposed grouping algorithm discussed earlier. Specifically, on the server side, the tasks performed are real-time rendering, residual view calculation and encoding, in sequence. Meanwhile, the server will cluster users into different groups at short intervals (e.g. every 100ms) with our proposed grouping algorithm. On the client side, decoding task will be performed for primary users while decoding and synthesis (of primary and secondary views) tasks will be performed for secondary users.

Since the fundamental limitation of dumping the frame information in real-time from Unity [33], we demonstrate the ability of real-time residual view calculation using a separate program. Our program is written in C++ using 64-thread on a Xeon 2-CPU server. We calculate the residual view calculation for 1080p frame 100 times and report the average latency in Table 11 and Table 12. We can observe the average latency as 2.5ms. When using GPU parallel implementation (i.e. computation is executed for pixels in a frame in parallel instead of sequentially), we can expect smaller latency for these two tasks.

C COMPLEXITY ANALYSIS FOR VIRTUAL GALLERY

To analyze the complexity in the various scenarios, we also perform experiments on the virtual gallery application. We empirically estimate upper bound for maximum number of groups traversed K and the number of iterations in K-means procedure I_u, and validate that the complexity of our proposed algorithm can be practically simplified as $O(n)$. We conduct experiments with number of users (i.e. 16, 25, 36,. . . 100), randomly located in the virtual gallery space as explained before. For a given number of users, we generate 100 different user topologies and implement our grouping algorithm on every user topology. We also calculate the average of parameters (i.e. k, I_u, $runtime$) for these different user topologies.

Figure 21(a) shows the optimal number of groups k selected by our algorithm versus total number of users. The dashed lines represent, for different values of I_w used, the average number of groups across all user topologies, considering the number of users. We observe that the number of groups demonstrate similar trends and are generally smaller than $\sqrt{n}$, across all user topologies. In our implementation, we empirically bound the maximum number of groups $K = 10$. Figure 21(b) presents the number of iterations needed by our algorithm, I_u, versus the number of users. The dashed lines demonstrate the average I_u considering the number of users. We can see a slow increase in I_u with increase in number of users n. The values of number of iteration I_u are also identical when $I_w = 20$ and 50, and similarly for $I_w = 1$ with some fluctuations.

Figure 22 presents the runtime versus the number of users in virtual gallery applications. Compared with the empirical results in virtual classroom, we can see that the average runtime grows linearly with the total number of users. The difference is that the runtime for gallery application

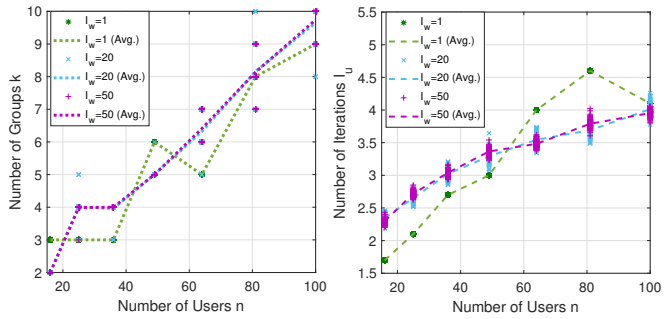

Fig. 21. (a) Number of Groups versus the number of users in virtual gallery. The dashed line represents the average number of groups given the number of users; (b) Number of Iterations as I_u versus the number of users. The dashed line demonstrates the average I_u considering the number of users.

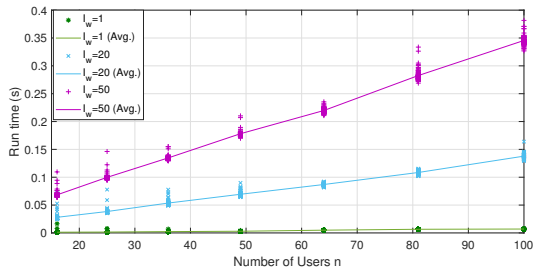

Fig. 22. The run time versus the number of users when $I_w = 1, 20, 50$ respectively in virtual gallery. The line represents the average run time given the number of users.

scenarios is slightly larger than for virtual classroom due to more randomness in user locations (topologies) in the former. The number of iterations I_u for virtual gallery cases is also slightly larger than for virtual classroom scenarios, as shown in Figure 17(b) and Figure 21(b).