CloVR: Fast-Startup Low-Latency Cloud VR

Yuqi Zhou, Voicu Popescu

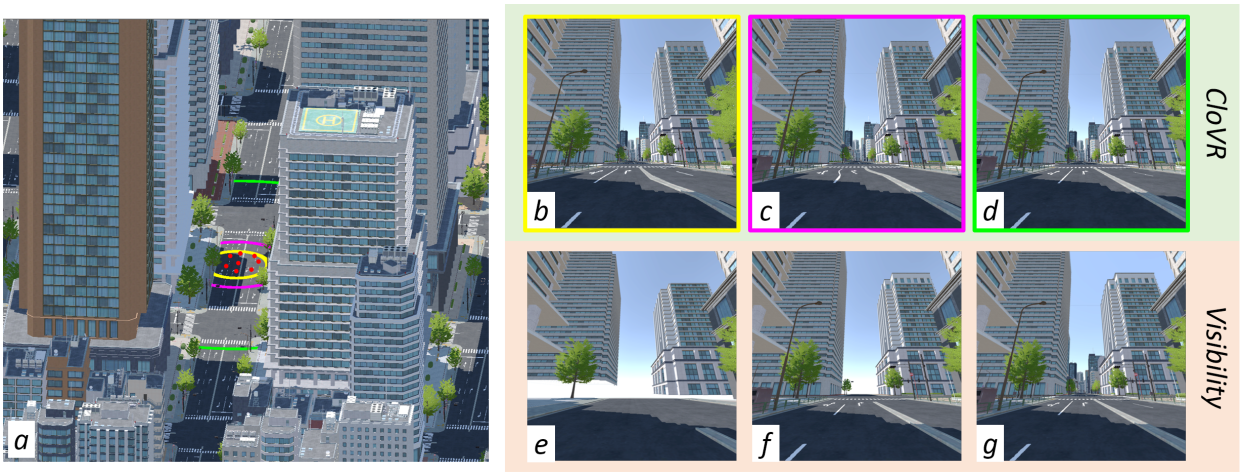

Fig. 1: (*a*) Overhead visualization of eight users (red dots) in a *City* virtual environment (VE) with 25 million triangles. The users are co-located in the same real world room and are wearing VR headsets that can render up to one million triangles. A server running on a laptop is reducing the VE either through our *CloVR* approach of near-far partitioning with continuous progressive refinement (*b*-*d*), or through conventional visibility computation (*e*-*g*). With *CloVR* the user quickly sees a complete view of the VE (*b*), which is then refined progressively by increasing the near region with strict visual continuity (from *b* to *c* to *d*). With the conventional approach the user is provided with an incomplete view (*e*) that is filled in as objects are received (*f* and *g*).

Abstract— VR headsets have limited rendering capability, which limits the size and detail of the virtual environment (VE) that can be used in VR applications. One solution is cloud VR, where the "thin" VR clients are assisted by a server. This paper describes *CloVR*, a cloud VR system that provides fast loading times, as needed to let users see and interact with the VE quickly at session startup or after teleportation. The server reduces the original VE to a compact representation through near-far partitioning. The server renders the far region to an environment map which it sends to the client together with the near region geometry, from which the client renders quality frames locally, with low latency. The near region starts out small and grows progressively, with strict visual continuity, minimizing startup time. The low-latency and fast-startup advantages of *CloVR* have been validated in a user study where groups of 8 participants wearing all-in-one VR headsets (Quest 2's) were supported by a laptop server to run a collaborative VR application with a 25 million triangle VE.

Index Terms—Cloud VR, Near-Far Partitioning

1 Introduction

Virtual Reality (VR) technology has rapidly advanced in recent years, offering powerful immersive experiences for users at accessible cost. Untethered VR headsets are now available with on-board rendering, tracking, and power. However, such all-in-one VR headsets have limited rendering capability, which limits the size and detail of the virtual environment (VE) that can be used in VR applications. Reducing the rendering load to make large VEs tractable on "thin" VR clients can take one of two fundamental approaches. One approach is visibility computation, which reduces rendering load by finding the subset of the VE visible to the user and by restricting rendering to this visible set. A second approach is level of detail (LoD) adaptation, which reduces rendering load by rendering distant parts of the VE from simplified representations. A straightforward and effective LoD method is to partition the scene into a near and a far region, rendering the near scene in full detail, from geometry, and rendering the far region in lower detail, from an environment map (i.e., a cubemap or a skybox). However, both the visibility and the LoD approaches to reducing rendering load require handling the VE at its full complexity, which could exceed the capabilities of the VR headset. What is needed is to completely insulate the VR client from the complexity of the VE.

Internet advances have allowed the virtualization of many computing applications. Interactive visualization is a challenging candidate for virtualization since it requires frequent communication between the server and the client as the user is free to change the view and to interact with the dataset as they desire. For each frame, the client has to send the view parameters to the server and then the server has to render and send the frame to the client, which incurs a latency at least equal to the network round-trip time between the server and the client. As network latencies have decreased, cloud interactive visualization is now starting to become practical on conventional, non-immersive displays, such as laptop, tablet, or phone screens. However, VR has strict latency requirements [19] that networks cannot yet meet. Consequently, a pure cloud VR approach where all rendering happens on the sever is not yet tractable. The solution is to partition the load between the server and the client, with the server reducing the VE to a more compact representation that the client can render locally with low latency. In addition to latency, a second cloud VR design concern is a fast VR application startup time. Once the user has initiated a VR application session or has teleported to a new location in the VE, the user should be able to see and interact with the VE as quickly possible, without

- *Yuqi Zhou is with Purdue University. E-mail: zhou1168@purdue.edu*
- *Voicu Popescu is with Purdue University. E-mail: popescu@purdue.edu*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

lengthy loading times.

In this paper we describe *CloVR*, a client-server system for cloud VR that includes a novel approach to progressive refinement for fast startup at session initialization and after teleportation. *CloVR* reduces the original VE to a size manageable by thin VR clients through near-far partitioning: the server renders the far region to an environment map which is sent to the client along with the geometry of the near region; the client renders output frames from the near region geometry and the far region environment map, with low latency. *CloVR* starts out with a small initial near region which is then grown progressively. In terms of transfer from the server, the near region is grown discretely, one ring at the time. In terms of visualization, the near region is grown with strict visual continuity: as a new ring arrives at the client, the ring is injected into the visualization gradually, maintaining frame to frame continuity. Fig. 1 illustrates the process. The *CloVR* approach starts with a 10m near region (yellow in *a*), which loads quickly yet provides the user with a complete view of the VE (*b*). The near region is grown by transferring from the server rings of increasing radii (magenta and green in *a*). When a ring arrives, the new geometry is deployed over several frames with visualization continuity that gradually morphs frame *b* into *c* and then into *d*. We also refer the reader to the video accompanying our paper. Fig. 1 compares *CloVR* to an approach based on visibility that completes the user frame one object at a time, as objects are received from the server (*e-g*). Here the objects are sent in decreasing order of their frame footprint, and even so early frames are far from complete (*e*).

We have evaluated *CloVR* both analytically and through a user study. The analytical evaluation confirms that *CloVR* provides a complete visualization of the VE early on in the loading process, unlike the visibility-based methods that render visible objects as they arrive from the server. We conducted a user study with two experiments. The first experiment ($N = 20$) confirmed the early visualization completeness advantage of *CloVR* over visibility-based methods, and it showed that *CloVR's* gradual increase of the near region with visual continuity does not come at a significant performance penalty. The second experiment ($N = 16$) confirmed that *CloVR* can support eight users simultaneously, allowing them to work together in a collaborative VR application with stringent multi-user synchronization requirements.

In summary, our paper contributes:

- A method that supports fast startup at session initialization and after teleportation, with progressive refinement that guarantees visual continuity.
- A cloud VR system that allows thin clients to explore complex VEs with low latency, with the help of a server running on a laptop.
- An analytical and empirical evaluation of the cloud VR system.

2 Related Work

Researchers have analyzed the feasibility of a pure cloud VR approach where all rendering happens at the server, which requires a network round-trip for the client to request and then receive the frame. The conclusions are that network latency remains a challenge [33], and that 5G cellular networks and beyond should have ultra-low latency, with an upper bound of 20ms [21].

To address the latency issue, researchers have examined hybrid approaches, where some of the VE is rendered locally, on the client. For example, the CloudVR [11] system renders most of the VE on a remote server while small objects close to the user are rendered on the VR client, noting that avoiding latency is particularly important for the objects with which the user interacts. Another approach for combating latency is to use the frame received from the server to reconstruct additional frames locally, on the client, through warping, which keeps the visualization responsive while waiting for the next server frame, at the cost of lower quality intermediate frames [13]. Instead of partitioning the VE geometry, researchers have also proposed partitioning the shading load between the server, in charge of expensive global illumination effects that require ray-tracing, and local illumination effects that can be handled by the client [28].

Reducing rendering load as a way of improving frame rate and frame quality is a long standing problem in computer graphics research. In complex 3D scenes, the user might have line of sight to only a small fraction of the scene due to occlusions. One approach for reducing the rendering load is to find and only render the visible set. Visibility has been studied extensively, with no complete solution. Some visibility algorithms search for visible triangles heuristically, by probing the dataset with visibility rays [14]. Such sampled-based algorithms have a low computational cost, but they can miss some visible triangles. Continuous visibility algorithms can produce a complete visible set, for example by probing for visible triangles not with rays but with beams [35], with the shortcoming of an increased computational cost. The camera offset space visibility algorithm [9] takes a hybrid, i.e., sample-based and continuous, approach that estimates the camera translation space region under which a pixel center is covered by a scene triangle; the resulting visible set is nearly complete, but it can miss some visible triangles and include some hidden triangles. At a fundamental level, even if the visible set is exact, the visibility approach to reducing rendering load has the disadvantage that the visible set could be too large, i.e., it does not allow the application to enforce a rendering budget.

Another approach for rendering load reduction is to compute an alternate representation of the 3D scene that has fewer triangles but that produces identical, or at least similar, frames to those obtained from the original representation. The idea is to reduce the level-of-detail (LoD) of distant objects with a small screen footprint, but LoD computation is challenging [18]. Interactive computer graphics applications often use a simple scheme that has two levels of detail: full detail for the objects close to the user, and low detail for distant objects, which are prerendered in an environment map [3]. Near-far scene partitioning and environment mapping has recently been used in the context of VR for rendering load reduction [23]. Whereas the environment map was previously reserved to the very distant parts of the scene, like the mountains at the horizon in an outdoor scene, the work introduces an intermediate region that maintains continuity between the near and far regions, which allows using environment mapping at closer distances. Our *CloVR* system uses this prior art approach [23] to maintain geometric continuity between the near and far regions.

Another effort for reducing rendering load in VR proposes rendering distant objects only once, i.e., monoscopically, to leverage the fact these objects contribute little to the disparity between the left and right eye frames [5]. A similar 2x load reduction is pursued by rendering the VE just once, from the midpoint of the interpupillary segment, and using the resulting depth image to reconstruct the left and right frames by 3D warping [26]. However, complex VEs require far more aggressive rendering load reduction factors.

The networking research community has also investigated distributing VR applications [4, 12, 17]. Due to the widespread availability of 360^o videos, one focus is saving bandwidth by not transferring the parts of the 360^o frame that the user does not see, i.e., to implement view frustum culling at the server [8, 25]. For free-viewpoint videos, also known as volumetric or 3D videos, the transfer and rendering load reduction provided by view-frustum culling was augmented with occlusion culling and with hierarchical LoD schemes, by taking advantage of the uniform structure of the RGBD frames [7, 16, 24, 32].

We target VEs modeled with triangle meshes, as needed by many VR applications. The triangle mesh representation of VEs is less uniform than that of videos, which complicates occlusion culling and LoD computation. One approach relies on the server to convert the VE to a uniform representation, by computing environment maps on a uniform 2D grid partitioning the ground plane [15]. The user is confined to a discrete set of possible viewpoints. The idea of simplifying LoD adaptation by discretizing the VE has also been used by a VR system where rendering is done on a workstation with a high-performance GPU [27]. Another example is a system that voxelizes the VE on the server and the client reconstructs and renders the VE from the voxels using marching cubes [29]. A method focusing on urban environments reduces the VE complexity by replacing it with a database of 2D images [22]. Our approach renders nearby objects from geometry,

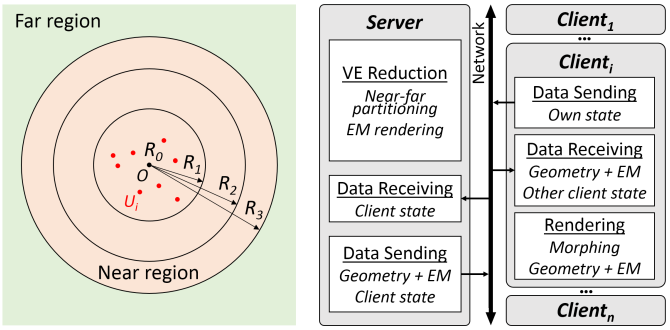

Fig. 2: *CloVR* approach (left) and system overview (right).

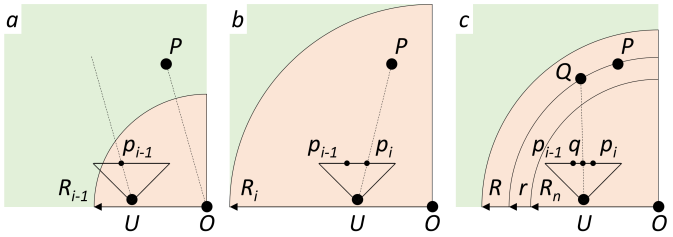

Fig. 3: Near region extension with visual continuity. Initially (panel *a*), the near region has radius R_{i-1}; a VE object P is in the far region and projects to p_{i-1}. If the near region is extended abruptly to R_i (panel *b*), the projection of P jumps from p_{i-1} to p_i. For a near region with radius $R \in [R_{i-1}, R_i]$ (panel *c*), our gradual extension projects P to the intermediate position q, as explained in Alg. 1.

avoiding the quality loss inherent to the discretization of the VE.

Researchers have also proposed differentiating between higher latency cloud servers and highly responsive edge servers, investigating optimal resource allocation [20], including with the help of deep learning [34]. In our work the server runs on a laptop, and the server and clients are co-located, which corresponds to the scenario of a low-latency edge server, exclusively dedicated to a single collaborative VR application.

3 *CloVR* Approach to Cloud VR

Fig. 2 gives an overview of our *CloVR* approach. The server partitions the VE into a near and a far region using a vertical cylinder. Partitioning proceeds at object level. Any object that has a vertex inside the cylinder is assigned to the near region. The server renders all geometry in the far region to an environment map from a viewpoint O, on the cylinder axis, at a typical viewing distance. The server transfers the environment map and the near region geometry to the client. The client renders output frames from the near region geometry and the far region environment map. The client never downloads or renders the VE at its original complexity. The client only renders geometry in the near region, which allows for tuning the rendering load to the client's rendering capabilities. The output frames are rendered locally, with low latency.

For a fast startup, the near region starts out small and is then grown progressively. The initial near region radius is 0 (R_0 in Fig. 2), so the server first sends an environment map that captures the entire VE. Then the initial region is grown to its first non-zero size (R_1), and the near region geometry and new environment map are sent to the client. Once the client receives the new data, the visualization switches gradually to the larger near region, with visual continuity, as described in Sec. 3.1. Once the user receives the first near region with non-zero radius, the user is not only provided with a complete visualization of the VE, but also with a functional VE: the user can see nearby geometry with correct depth perception, the user can interact with nearby geometry, and the user can see their own avatar and those of other nearby users. The near region is enlarged repeatedly until it reaches its final size (i.e., R_3 in Fig. 2).

In the case of a multi-user collaborative VR application, the clients have to communicate their state (e.g., avatar pose) to the other clients in real time. For this, each client reports its relevant state to the server, which relays it to the other clients. Furthermore, all users within the same near region (red dots in Fig. 2) rely on the same reduced version of the VE, amortizing server computational load by leveraging user locality.

3.1 Fast Progressive Startup with Visual Continuity

Once a client receives the geometry G_i of the additional objects needed to extend the near region from R_{i-1} to R_i and the new environment map EM_i, switching abruptly to the larger near region creates an abrupt, significant, and therefore highly objectionable change in the user's view. The cause of this change is illustrated in Fig. 3, panels *a* and *b*. U is the user viewpoint, and O is the environment map viewpoint, i.e., the near region center. When the near region is extended from R_{i-1} to R_i, an object P in G_i jumps in the user view from p_{i-1} to p_i; $U\,p_{i-1}$ is the user ray parallel to the environment map ray OP along which P is captured in EM_{i-1}; $U\,p_i$ is the user ray through the actual 3D position of P.

In order to inject geometry G_i gradually into the visualization, instead of rendering G_i from the user viewpoint right away, G_i is first rendered from O, where it was visible in EM_{i-1}, and then the viewpoint is gradually switched to U. Panel *c* in Fig. 3 shows the image projection of P to q for a frame that is part of the gradual deployment of the new geometry. In Fig. 1, rendering new geometry from the user viewpoint as soon as it becomes available creates a jump from *e* to *g*, whereas our gradual deployment morphs *e* into *g*, creating intermediate frames such as *f*.

The gradual deployment is implemented by projecting geometry vertices with Alg. 1. A vertex P is projected based on its distance r to the partitioning cylinder axis, which is here assumed to be vertical (line 1). To maintain continuity between the near and far regions, the near region is further subdivided into two subregions: a near subregion for $r < R_n$, and an intermediate subregion for $R_n < r < R$ (Fig. 3, *c*). The smaller the radius R_n, the larger the intermediate region width $R - R_n$, and the more gradual the switch between the environment map viewpoint and the user viewpoint. However, the smaller the radius R_n the smaller the distortion-free region surrounding the user. In practice we use $R_n = 0.7R$, which provides a good compromise.

There are three cases. If P is in the near subregion (line 2), then P is not displaced (line 3), and it is projected conventionally (line 12). If P is in the far region (line 4), then P is first displaced to point Q which is at the intersection of the vertical cylinder through P and a ray (line 6), after which it is projected (line 12). The ray starts at U and has the direction in which P is seen from O (line 5). This way, P appears in the output frame as if it were looked up in the environment map, which is where it was seen in the previous frames. If P is in the intermediate

Algorithm 1 Vertex projection for gradual near region extension with visual continuity.

Input: Vertex P, current near region radius $R \in [R_{i-1}, R_i]$, the near subregion radius R_n, user viewpoint U, user view MVP_U, environment map viewpoint O
Output: Vertex projection q

1: r = length$(O.xz - P.xz)$
2: **if** $r < R_n$ **then** // in near subregion
3: $\quad Q = P$
4: **else if** $r > R$ **then** // in far region
5: $\quad d$ = normalize$(P - O)$
6: $\quad Q$ = Ray$(U, d) \cap$ Cylinder(O, r)
7: **else** // in intermediate subregion $(R_n < r < R)$
8: $\quad w = R - r/(R - R_n)$
9: $\quad d_U$ = normalize$(P - U)$; d_O = normalize$(P - O)$
10: $\quad d$ = normalize$(d_U w + d_O(1 - w))$
11: $\quad Q$ = Ray$(U, d) \cap$ Cylinder(O, r)
12: **return** $q = MVP_U \times Q$

subregion (line 7), then P is first displaced to a point Q that is also at the intersection of the vertical cylinder through P and a ray (line 11). Like before, the ray starts at U, but now the ray direction is a linear interpolation (line 10) between the the direction d_U in which P is seen from U and the direction d_O in which P is seen from O (line 9). The weights of the linear interpolation w_U and w_O are determined by the ratio in which r splits the interval $[R_n, R]$ (line 8).

As R grows from R_{i-1} to R, the ring $[R, R_n]$ moves over a vertex P, changing its projection gradually from p_{i-1} to p_i, maintaining visual continuity (panel *c* of Fig. 3). Alg. 1 displaces vertices on their original cylinder. This comes at a cost of a ray-cylinder intersection, which requires solving a quadratic equation. This additional cost is warranted since it allows rendering geometry in the far region while waiting for the growing near region to incorporate it in the visualization.

The radii R_{i-1} and R_i are an input to Alg. 1. We select the sequence of radii that define the growth of the near region based on several considerations. One consideration is to keep the amount of data contributed by each ring constant. Assuming a uniform geometric density on the ground plane, this leads to decreasing radii changes, in order to keep the area of larger and larger rings constant. Another consideration is to take into account visibility–not all geometry inside farther rings is visible due to occlusion by the geometry of smaller, nearby rings, so one could choose thicker distant rings for the same amount of data. A third consideration is that farther rings have a smaller output image footprint due to perspective foreshortening, so the contribution of individual rings decreases with distance, which argues in favor of thicker farther rings. A fourth consideration is that each ring brings its own cubemap, which is not useful once the next next ring is loaded, overhead that increases with the number of rings. In practice, we take into account all these competing factors to start with rings of equal area (e.g., 314 m^2 for the *City* VE from Fig. 1), and then merge consecutive rings based on visibility and distance from user (e.g., resulting in rings of radii of 10m, 14m, 17m, 22m, 32m, and 63m for *City*).

4 Evaluation

We have evaluated *CloVR* both analytically and empirically. We compared our startup approach with *continuous* refinement (*CloVR*) to the same approach but with *discrete* visual refinement (*CloVRD*), as well as to two visibility-based startup approaches (*Vis0* and *Vis1*).

CloVR. This condition uses our approach as described in Sec. 3. The radius R of the cylinder starts at 0m and is increased discretely (i.e., to 10m, 14m, 17m, 22m, 32m, and 63m for *City*).

CloVRD. This condition is identical to *CloVR* except that the visualization switches abruptly to the larger near region once the next ring's data is received from the server.

Vis0. The first visibility-based approach partitions the VE into a near and a far region using a vertical cylinder of radius R (i.e., R = 63m for *City*). The objects in the far region are prerendered to an environment map from a reference viewpoint O which is on the cylinder axis, at a height h (e.g., h = 1.3m for *City*) above the ground plane. The objects in the near region are rendered from O to measure their visibility footprint. The objects in the near region whose visibility footprint is zero are discarded. The server sends the visible near region objects and the cubemap to the client. The client displays a blank screen while waiting for all the data to arrive.

Vis1. The second visibility-based approach proceeds like *Vis0*, except that the near objects are sent in decreasing order of their visibility footprint, and that the client displays each object as soon as it is received, without waiting for all objects to arrive first.

4.1 Analytical Evaluation

We have investigated the *CloVR* and the three approaches (*CloVRD*, *Vis0* and *Vis1*) in terms of three quantitative, objective metrics.

Object Visualization Completion (OVC). The *OVC* metric is the percentage of objects that are shown in a user frame F^*, out of the total of number of objects that are shown in the corresponding ground truth frame F. An object is shown in a user frame if it appears at least at one pixel in the frame. The ground truth frame is computed by rendering the entire VE from geometry. The *OVC* metric is given by Eq. 1, where

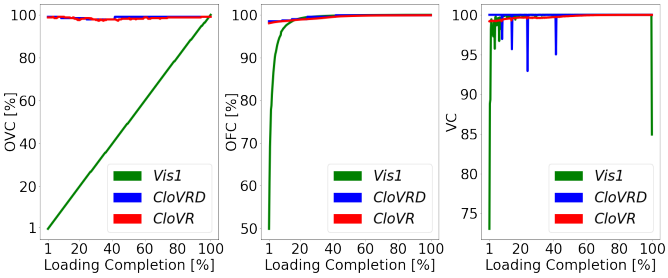

Fig. 4: Analytical comparison of loading approaches for the *City* VE in terms of object visualization completion (left), object footprint completion (middle), and visual continuity (right).

o_i are the n objects shown by the ground truth frame F, and $F^*(o_i)$ is the footprint of object o_i in the user frame F^*. *OVC* is 0 for *Vis0* until the end of the loading process where it jumps to 100%. Fig. 4, top, shows the *OVC* for the three approaches (excluding *Vis0*) for a user viewpoint U that is 1m to the left of the near region center O. The x axis is the loading completion in terms of object geometry for *Vis0* and *Vis1*, and in terms of the near region radius R for *CloVRD* and *CloVR*. As expected, for *Vis0* the *OVC* is 0% until the last object arrives from the server, when it jumps to 100%. For *Vis1*, *OVC* increases linearly from 0% to 100%. For both *CloVRD* and *CloVR*, *OVC* starts out high, because most objects are on screen from early on, either part of the environment map, or part of the near region. *CloVRD* and *CloVR* incorrectly omit an object on the rare occasion when the object is visible from the user viewpoint U but not from the environment map viewpoint O, or when the intermediate region morph alters the visibility from U.

$$OVC(F^*, F) = \frac{1}{n}\sum_{i=1}^{n}(F^*(o_i) > 0)\,?\,1 : 0 \tag{1}$$

Object Footprint Completion (OFC). The *OFC* metric is similar to the *OVC* metric, but the *OFC* metric doesn't just check whether an object is represented by at least a pixel in the frame, and instead it also checks whether the footprint of the object is the same as the footprint the object has in a ground truth frame. Given a frame F^* and its corresponding ground truth frame F, the *OFC* metric is given by Eq. 2, where $F^*(o_i)$ is the footprint of object o_i in F^* and $F(o_i)$ is the footprint of object o_i in F. n is the total number of objects shown in F. The footprint of an object in F^* is capped to its footprint in F, to avoid incorrectly skewing the frame *OFC* value when occluding objects are missing making the footprint of objects larger than what they should be. For *Vis0*, *OFC* is 0% until the loading process completes, and then it jumps to 100%. Fig. 4, middle, gives the *OFC* metric for the three approaches, excluding *Vis0*. The main difference with the *OVC* metric is that now *OFC* increases super-linearly for *Vis1* as the objects with the largest footprint are transferred first.

$$OFC(F^*, F) = \frac{\sum_{i=1}^{n} min(F^*(o_i), F(o_i))}{\sum_{i=1}^{n} F(o_i)} \tag{2}$$

Visual Continuity (VC). We investigate the visual continuity provided by the various approaches by computing the Structural Similarity Index Measure [30] between consecutive frames, with a window size equal to the entire image. Since the user view and the geometry do not change, there should be no changes between frames. The *VC* metric is given in Fig. 4, bottom. As expected, *Vis1* suffers from visual discontinuity, i.e., a drop of the *VC* value, every time an object arrives, with larger discontinuities earlier on as the objects with larger footprints arrive. The final visual discontinuity is due to the environment map's inclusion in the visualization. *CloVRD* suffers from visual discontinuity every time a new geometry ring is included in the visualization. *CloVR* provides good visual continuity throughout the loading process.

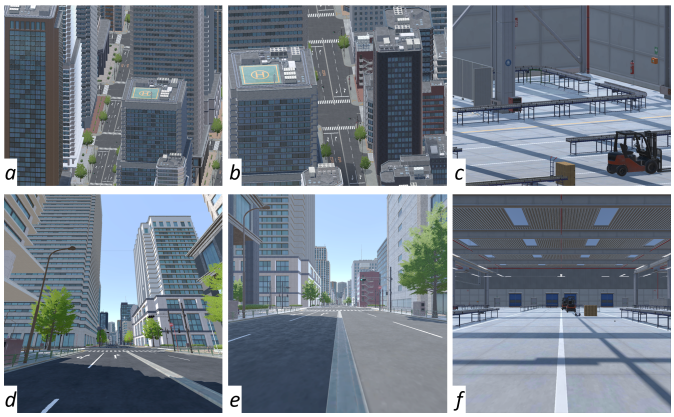

Fig. 5: Third-person (top) and first-person (bottom) views of the three locations used in Experiment 1: two locations in the *City* VE, i.e., *City1 (a, d)* and *City2 (b, e)*, and one location in the *Factory* VE (*c, f*).

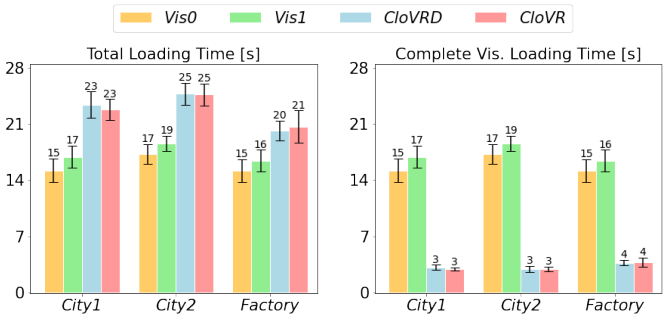

Fig. 6: Box-plots of total loading time (left) and of complete visualization loading time (right), for the four approaches and the three VE locations, over all participants.

4.2 Empirical Evaluation: User Study

We evaluated *CloVR* in a user study with two experiments. The user study was approved by our university's Institutional Review Board, the Purdue University Human Research Protection Program with the IRB number IRB-2023-622. The first experiment investigates using the *CloVR* approach to support a single user starting up (or teleporting) in a complex VE with unique geometries, requiring a substantial data transfer, as described in Sec. 4.2.1. The second experiment investigates using the *CloVR* approach to support multiple users engaged simultaneously in a collaborative VR application, co-located in the same region of the VE, as described in Sec. 4.2.2.

Setup. The server and the clients were located in the same real world room, i.e., a classroom on our university campus. The server and the client software were implemented using Unity 3D [2], version 2021.3.18f. The server was run on a Windows 11 Dell G7 7500 laptop with an Intel(R) Core(TM) i7-10750H CPU, 32.0 GB of RAM, a Killer 1650i wireless module, and an NVIDIA GeForce RTX 2070 graphics card. The clients were run on Meta Quest 2 headsets [1]. The client headsets were connected through WiFi to the laptop's hotspot. The laptop's upload speed is 2,400 Mbps and the Quest 2 maximum download speed is 1,200 Mbps.

Participants. We have recruited $N = 26$ participants; 6 self-identified as females, 19 self-identified as males, and 1 declined to answer the gender question; all were over 18 years of age, with an average age of 25.3 $\pm$ 3.35; regarding the VR experience question, 4 participants answered "never", 8 "once", 12 "occasionally", and 2 "frequently". 8 participants were assigned to session 1, 10 to session 2, and 8 to session 3. The sessions were run on consecutive days.

Pre-experiment activities. The session started with administering an eligibility questionnaire. The eligibility questionnaire was aimed to exclude participants who are prone to severe motion sickness, who have mobility or muscular skeletal issues that prevent them from wearing a headset, or who do not have normal or corrected vision. No participant was found to be ineligible. Participants then filled out the consent form. Consenting participants answered the demographics questionnaire. Each participant was then involved in both experiments. The total time involvement of each participant was at most 90 minutes, door to door.

4.2.1 Experiment 1: *CloVR* support for a single user

The first experiment investigates the ability of the *CloVR* approach to support fast startup at session initialization and after teleportation, with progressive refinement. The investigation is carried out for a complex VE with unique objects that amount to a significant data transfer at startup.

Experimental design. The first experiment exposed participants to the various approaches for loading the VE. We opted for a within subjects design with each participant being exposed to all four conditions *Vis0*, *Vis1*, *CloVRD*, and *CloVR*, in counterbalanced order. The within subjects design allows contrasting the conditions with fewer participants compared to a between subjects design.

Procedure. Participants worked individually. A participant put on the VR headset and read an initial instructions screen. Then a participant initiated loading a VE. We used two locations in the *City* VE and one location in the *Factory* VE, see Fig. 5, for each of the four conditions, totaling 12 trials. We used two locations in the *City* VE because the buildings are not of uniform complexity, which could translate in results variability. The *Factory* VE is compact and uniform, which does not warrant multiple locations. The participant had a virtual position 2m away from the center of the near region. Participants were seated for a consistent viewpoint, as needed for a consistent comparison of the various download conditions. Since participants are wearing a headset it is not possible to eliminate completely viewpoint and view direction discrepancies, as playing back pre-recorded paths immersively makes participants prone to cybersickness. Each loading trial took at most 25s. After each trial, participants answered a questionnaire in the VR application, as described below. A participant's total involvement in Experiment 1 lasted at most 20 minutes.

Data collection. The client headset recorded the time needed for the various loading stages and the frame rate. After each trial, the participant answered three or four questions about the VE loading they had just experienced, on a five-point Likert scale:

- Q1 (*all*): The loading time was acceptable.
- Q2 (*Vis0*): I found the blank screen during loading to be boring.
- Q2 (*Vis1*): I liked seeing the VE fill in as it loaded.
- Q2 (*CloVRD, CloVR*): I liked seeing a complete version of the VE being refined during loading.
- Q3 (*Vis1*): The pattern in which the VE filled in was confusing.
- Q3 (*CloVRD, CloVR*): The VE refinement was confusing.
- Q4 (*all*): Overall, the download process was acceptable.

Data analysis. We analyzed the loading time, frame rate, and Likert answer data using the non-parametric Friedman test [6], as our data did not have a normal distribution. When the Friedman test indicated a significant difference between conditions, a Wilcoxon [31] post-hoc analysis was conducted to investigate pairwise differences, with a Bonferroni corrected significance level of $\alpha = 0.05/6$, where 6 is the number of pairwise comparisons. We used the SPSS statistical software package [10].

Results and discussion.

We attempted to run all eight participants of the first session in parallel, which was over the capabilities of the network, resulting in long delays unrelated to the conditions tested. This made the data of six of the eight participants unusable. For sessions two and three we ran three participants in parallel, and the sessions completed as expected. Therefore, for Experiment 1, we report data for 2 + 8 + 10 = 20 of the 26 participants.

Loading times. The loading times for the four conditions and the

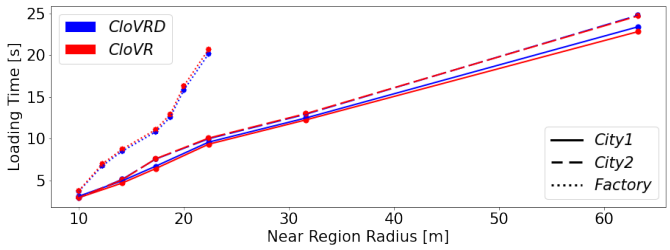

Fig. 7: Progressive loading times as a function of the growing near region radius, for the two near-far partitioning methods and the three VE locations, averaged over all Experiment 1 participants.

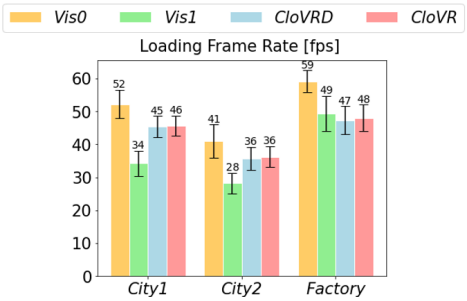

Fig. 8: Box-plots of average frame rate during loading, for the four approaches and the three VE locations, over all participants.

three VE locations are given in Fig. 6. The left panel gives the total loading time. For all four methods, the loading is complete when all the geometry of the (final) near region as well as the environment map of the far region are received. The statistical analysis of the differences is given in Tab. 1. There is a significant difference between the four conditions for all three VE locations. There is a small but statistically significant difference between *Vis0* and *Vis1*: *Vis0* is slightly but consistently faster than *Vis1* because for *Vis0* the headset does not render while waiting for the data to arrive, whereas for *Vis1* the headset renders the objects as they arrive. There is no significant difference between *CloVRD* and *CloVR*, which indicates that the gradual deployment of the new ring does not significantly slow down the download of the next ring. *City2* corresponds to a location in the urban VE that has a higher geometric density than the *City1* location, which is reflected by longer loading times for each of the four conditions (Fig. 6). The relative loading time increase from *City1* over *City2* is lower for the *CloVR* than for the *Vis* conditions, as the environment maps have similar sizes.

There is a significant difference between the loading times of any of the two visibility methods and any of the the two progressive near-far partitioning methods, i.e., for any of the four pairs in {*Vis0*, *Vis1*} x {*CloVRD*, *CloVR*}. The difference is due to the fact that *CloVRD* and *CloVR* transfer more data from the server, i.e., they transfer an environment map for each intermediate ring, whereas *Vis0* and *Vis1* only transfer the final environment map. For example, for *City1* the geometry data totals 113MB, the final environment map is 2.76MB, and the intermediate environment maps total 21.4MB. Therefore *Vis0* and *Vis1* transfer a total of 116MB whereas *CloVRD* and *CloVR* transfer a total of 134MB. At the cost of a delay of the completion of the loading process, the intermediate environment maps provide a complete visualization of the VE early on.

The right panel of Fig. 6 gives the time it takes to load a visually complete version of the VE; for the *Vis* methods this corresponds to the total loading time, while for the *CloVR* methods this corresponds to loading the geometry of the first ring with its environment map. The *CloVR* methods need 4s or less to download a functional and visually complete VE, whereas the *Vis* methods need at least 15s. The *CloVR* methods prioritize loading all geometry in a region of 10m radius, connected with visual continuity to an environment mapping visualization of the geometry beyond 10m. This initial region is sufficient (1) for a quality stereoscopic visualization, as the left-right eye disparity is highest for nearby objects and is very low beyond 10m; (2) for viewpoint translation by walking within the 10m radius region; (3) for the user to see correctly their own avatar as well as the avatar of nearby collaborators.

Fig. 7 shows that the rings selected for increasing the near region for the *CloVR* approaches do achieve an adequate transfer load balancing with an approximately linear growth of the near region as a function time.

Frame rate. In terms of frame rate, once loading completes, the frame rate is the same for all four approaches as they are rendering the same reduced version of the VE. The post-loading frame rates are 66.1 $\pm$ 3.98 fps, 57.7 $\pm$ 3.14fps , and 71.8 $\pm$ 0.16fps for *City1*, *City2*, and *Factory*. Rendering the original VEs on the headsets is prohibitively expensive, resulting in frame rates of 5.09 $\pm$ 2.78fps for *City*, and 26.3 $\pm$ 1.69fps for *Factory*. Both the visibility and the near-far partitioning approaches succeed at reducing the rendering load to enable comfortable frame rates.

The average frame rates during loading are given in Fig. 8, and the statistical analysis in Tab. 2. Like for total loading times, there is a significant difference between the four conditions, and the pairwise difference between *CloVRD* and *CloVR* is not significant, which again indicates that the gradual deployment of the new ring does not affect rendering performance. As expected, *Vis0* has the highest frame rate since it does not render anything during loading. Somewhat surprisingly however, for *City1* and *City2*, *Vis1* has a significantly lower frame rate than both *CloVRD* and *CloVR*. We attribute this difference to the greedy object transfer scheduling of *Vis1*, which transfers objects with larger screen footprint first. These objects are often also the more complex objects, which require the largest CPU computational effort to decode from the packets received from the server. On the other hand, *CloVRD* and *CloVR* send objects in distance order, i.e., ring by ring, which mixes more complex objects with simpler objects, achieving a better balancing of the decoding load.

Preference questionnaire. The answers to the preference questionnaire are shown in Fig. 9, which awards a score of 1 to 5 for the strongly disagree to strongly agree answers of the Likert scale. We have opted to have negative questions in the preference questionnaire to avoid mechanical answers. The negative questions are Q2 for *Vis0*

		Vis0 - *Vis1*	*Vis0* - *CloVRD*	*Vis0* - *CloVR*	*Vis1* - *CloVRD*	*Vis1* - *CloVR*	*CloVRD* - *CloVR*
City1		Friedman: χ^2 = 50.22, p <0.001*					
	Z	-3.25	-3.92	-3.92	-3.92	-3.92	1.12
	p	0.001*	<0.001*	<0.001*	<0.001*	<0.001*	0.263
City2		Friedman: χ^2 = 50.22, p <0.001					
	Z	-3.02	-3.92	-3.92	-3.92	-3.92	-0.53
	p	0.002*	<0.001*	<0.001*	<0.001*	<0.001*	0.601
Factory		Friedman: χ^2 = 49.5, p <0.001*					
	Z	-3.02	-3.92	-3.92	-3.92	-3.92	-0.97
	p	0.002*	<0.001*	<0.001*	<0.001*	<0.001*	0.332

Table 1: Friedman (χ^2, p) and posthoc Wilcoxon (Z and p) analysis of loading time differences between the four conditions at the three VE locations. Statistical significance indicated with an asterisk.

		Vis0 - *Vis1*	*Vis0* - *CloVRD*	*Vis0* - *CloVR*	*Vis1* - *CloVRD*	*Vis1* - *CloVR*	*CloVRD* - *CloVR*
City1		Friedman: χ^2 = 50.58, p <0.001					
	Z	-3.92	-3.73	-3.92	-3.92	-3.92	-0.37
	p	<0.001*	<0.001*	<0.001*	<0.001*	<0.001*	0.709
City2		Friedman: χ^2 = 45.78, p <0.001					
	Z	-3.92	-3.51	-3.44	-3.92	-3.92	-0.64
	p	<0.001*	<0.001*	<0.001*	<0.001*	<0.001*	0.526
Factory		Friedman: χ^2 = 33.9, p <0.001					
	Z	-3.77	-3.92	-3.92	-2.32	-1.12	-0.9
	p	0.002*	<0.001*	<0.001*	0.021	0.263	0.37

Table 2: Analysis of frame rate differences between the four conditions at the three VE locations.

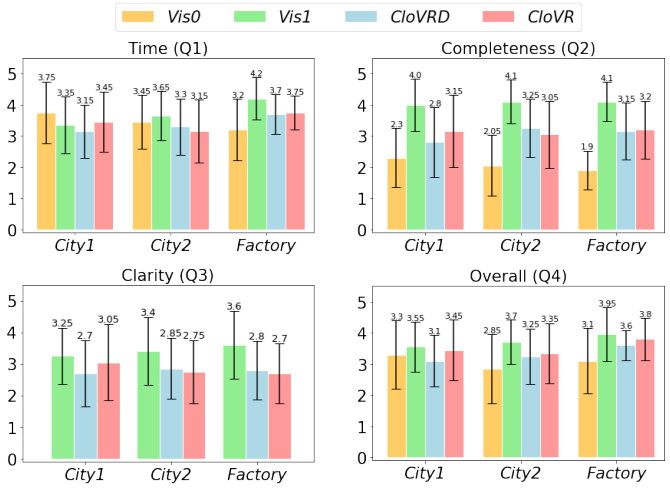

Fig. 9: Box-plots of answers to preference questionnaire for the four approaches and three VE locations, over all participants. Higher value always means better.

("blank screen boring"), Q3 for *Vis1*, *CloVRD*, and *CloVR* ("confusing intermediate visualizations"). For consistency, i.e., for higher values to always mean better, the answers to these negative questions were flipped, i.e., a value of x was replaced with $6-x$.

For download time acceptability (Time, Q1 in Fig. 9), the Friedman test indicates significant differences between the four conditions only for the *Factory* VE ($\chi^2 = 20.82, p < 0.001$). The Wilcoxon posthoc analysis reveals that the only pairwise difference that clears the Bonferroni corrected significance threshold of $0.05/6 = 0.008$ is *Vis0* versus *Vis1* ($Z = -3.26, p = 0.001$). Although *Vis0* is faster than *Vis1*, not seeing anything on screen during loading frustrates the participant, who has an exaggerated perception of the *Vis0* loading time. Although *CloVRD* and *CloVR* have both longer loading times than both *Vis0* and *Vis1*, the completeness of the intermediate visualization of the VE made the longer loading times acceptable to participants. For the *City* VE locations, no differences were significant, again, validating the hypothesis that slightly longer times are acceptable if they mean more complete intermediate visualizations.

For visualization completeness during loading (Completeness (Q2) in Fig. 9), *Vis0* received low scores, i.e., participants found the blank screen during loading to be boring. There are significant differences between the four conditions for all three VE locations, as shown in Tab. 3. *Vis1* had significantly higher scores than *Vis0*, i.e., participants preferred seeing the VE fill in as it loaded to seeing a blank screen. The participants' preference for seeing the VE fill in as it loaded (*Vis1*) was higher than their preference for seeing a complete visualization of the VE that is being refined during loading (*CloVR*). In other words, some indication of progress was more important to participants than seeing a complete VE early on. Some participants noted that computer games load objects individually rather than the entire scene once, like *Vis1*, and that they have never seen the environment load all at once and then refine through warping, like *CloVR* and *CloVRD*, which is a possible explanation for the higher scores given to *Vis1*. Furthermore, for *Vis1* the completeness question asked whether participants liked seeing the VE fill in as it loaded, and not whether the VE was complete early on. This suggests a comparison to *Vis0*, i.e., participants indicated their preference for seeing objects as they load, as opposed to waiting for all objects to arrive, hence the favorable score. *CloVRD* and *CloVR* were preferred to not seeing anything during loading (*Vis0*), and the difference was significant for *Factory*.

Participants found *Vis1* to provide the least confusing visualization during loading (Clarity (Q3) in Fig. 9), but no pairwise differences were significant. The concept of displaying objects one at a time as they arrive is simple and intuitive, hence the participant preference. *CloVRD* and *CloVR* show a complete visualization of the VE early on and then

		Vis0 - Vis1	*Vis0 - CloVRD*	*Vis0 - CloVR*	*Vis1 - CloVRD*	*Vis1 - CloVR*	*CloVRD - CloVR*
City1		Friedman: $\chi^2 = 19.78$, p <0.001*					
	Z	-3.35	-1.39	-1.8	-3.15	-2.68	-1.37
	p	<0.001*	0.163	0.071	0.002*	0.007*	0.169
City2		Friedman: $\chi^2 = 25.21$, p <0.001*					
	Z	-3.77	-2.59	-2.03	-2.62	-2.88	-1.16
	p	<0.001*	0.01	0.042	0.009	0.004*	0.248
Factory		Friedman: $\chi^2 = 32.77$, p <0.001*					
	Z	-3.98	-3.21	-3.27	-2.63	-2.8	-0.3
	p	<0.001*	0.001*	0.001*	0.009	0.005*	0.763

Table 3: Analysis of visualization completeness preference differences between the four conditions at the three VE locations.

manipulate it in a way unfamiliar to the user during the refinement that expands the near region. Participants noted the novelty of the *CloVR* refinement approach but found it more confusing than the "familiar" and "common" refinement implemented by *Vis1*. The differences in this highly subjective measure were not significant in our small-scale study.

In terms of overall preference (Overall (Q4) in Fig. 9), *CloVR* was always preferred to *CloVRD* and to *Vis0*. *Vis1* was always preferred to anyone of the three other conditions. However, the differences are small and the standard deviations large. The only pairwise significant differences were recorded for *City2* (Friedman $\chi^2 = 8.77, p = 0.032$) for *Vis1* versus *Vis0* (Wilcoxon $Z = -3.15, p = 0.002$), and for *Factory* (Friedman $\chi^2 = 9.66, p = 0.022$) for *Vis1* versus *Vis0* (Wilcoxon $Z = -2.94, p = 0.003$).

In conclusion:

1. unlike the visibility methods, *CloVR* provides a functional and visually complete initial version of the VE early on in the loading process;
2. unlike the visibility methods and unlike the discrete visual refinement *CloVRD* method, *CloVR* refines the VE progressively, with visual continuity; this did not come at the price of a significant loading time or frame rate penalty;
3. *CloVR* has a slightly higher total loading time than the visibility methods, and participants did not find the extended loading time to be objectionable;
4. *CloVR* has a high frame rate during loading;
5. participants had the lowest preference (2/5) for waiting until all data is loaded before any of the VE is displayed, a neutral preference (3/5) for growing the near region discretely or continuously, and the highest preference (4/5) for seeing objects being displayed as they arrived.

4.2.2 Experiment 2: *CloVR* support for a multiple users

The second experiment investigates the ability of the *CloVR* approach to allow multiple co-located users to explore complex VEs simultaneously, with low latency. Experiment 2 investigates multi-user interaction post download, where the emphasis is on the timely communication of low-volume but strictly-real-time data streams.

Experimental design and setup. We ran Experiment 2 in two sessions, each with eight participants (Fig. 10, left), for a total of 16 participants. All eight participants of a session were connected to the server simultaneously, and they were divided into two groups (i.e., teams) of four.

Task. The participants in each group had to collaborate to collect spheres that appeared in the *City* VE (Fig. 10, right). The VE was rendered using the *CloVR* near-far partitioning approach. Each participant used a handheld controller to aim a virtual laser. A sphere was collected if all four members of the group aimed their virtual laser pointer at the sphere for 2s. The only data that had to be transferred was the poses of the eight virtual laser pointers, so the campus WiFi had the necessary bandwidth to support the eight participants simultaneously. Each client sent the pose of their virtual laser pointer to the server and the server relayed all poses to all clients. A participant saw the virtual

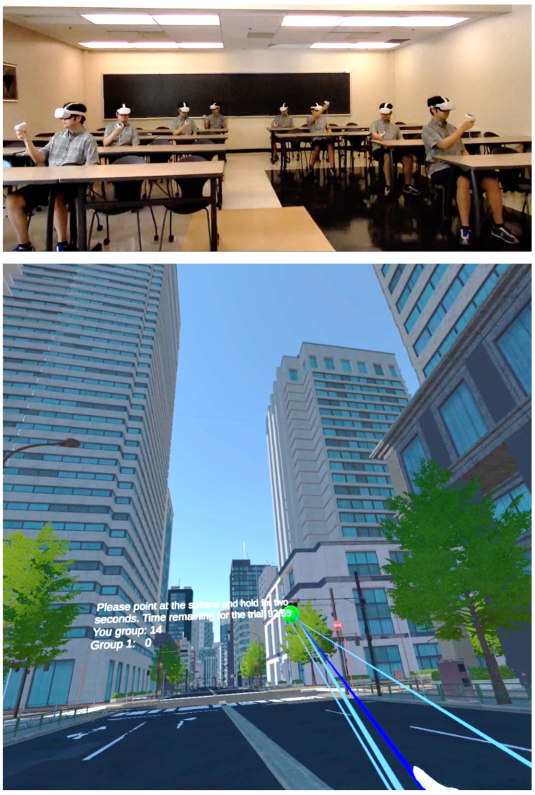

Fig. 10: *Top*: illustration of experiment room with eight participants (composite image shown to preserve participant privacy). *Bottom*: sphere collected collaboratively by a group of four participants with their virtual laser pointers.

laser pointers of the other three members of their group, but not of the other group. Furthermore, the spheres visible to one group were not visible to the other group. This way, groups were not competing for the same sphere. However, a group saw their own score, i.e., the number of collected spheres, as well as the score of the other group, which created indirect competition.

Procedure. Once participants put on the headsets, they were shown a screen with textual instructions that explained the collaborative task of collecting spheres, the goal of collecting as many spheres as possible, and the scores shown on screen. Then each client headset loaded the VE from the server. Whereas for Experiment 1 we tested loading performance assuming each object is unique, for Experiment 2 we leveraged object instancing in the *City* VE which greatly reduces the amount of data that has to be transferred. Once the VE loaded, participants performed the collaborative sphere collection task. The participants were located at random locations within a 5 m region concentric with the near region used by the *CloVR* approach.

A sphere appears in the VE one at a time. A sphere is visible until collected by the group, after which a new sphere appears. There were three types of spheres: *Easy*, i.e., stationary spheres, *Intermediate*, i.e., spheres moving with a 0.5m/s velocity, and *Hard*, i.e., spheres moving on with a 1.5m/s velocity. The position of each participant was chosen randomly in a 5m x 2m rectangular ground plane region. The participants were seated so they could not translate their viewpoint over large distances. The initial view direction of the participants was aligned. The *Intermediate* and *Hard* spheres moved on predetermined piece-wise linear trajectories that avoided intersections with the VE geometry and that kept them in front of the participants. Participants collected spheres in three trials. At the beginning of each trial, the score resets. A trial ends after 3min. The spheres alternated in difficulty from *Easy*, to *Hard*, to *Intermediate*, and then back to *Easy*. After each trial participants completed an experience questionnaire as described below. The questionnaire was administered in the VR environment so

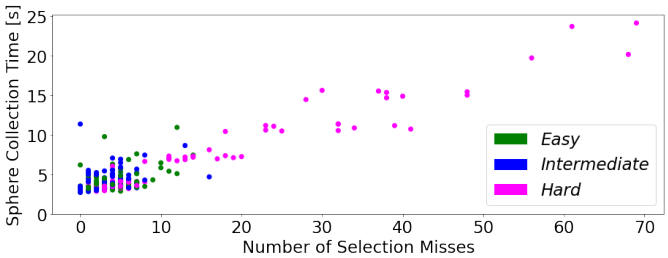

Fig. 11: The time needed to collect a sphere as a function of the number of selection misses, and of the sphere type.

participants did not have to remove the headset between trials.

Data collection. The objective metrics for Experiment 2 are the VE loading times, the frame rate for each client, the scores for each group and for each trial, the time needed to select each sphere, the type of each sphere selected, the number of misses when selecting each sphere, and the round-trip latency λ_i from participant i to the other members of the group. A miss is defined as a participant losing the selection of a sphere. The round-trip latency λ_i is given by Eq. 3, where τ_{is} (or τ_{js}) is the time to communicate from client i (or j) to the server, and τ_{si} (or τ_{sj}) is the time to communicate from the server to client i (or j).

$$\lambda_i = \tau_{is} + \max_{j \neq i}(\tau_{sj} + \tau_{js}) + \tau_{si} \tag{3}$$

λ_i is measured by sending a message M_1 from client i to the server, when the server receives M_1 it sends a message M_2 to each of the other clients j in the group, when a client j receives M_2 from the server it acknowledges it back to server with a message M_{3j}, and when the server receives all acknowledgments it sends a message M_4 back to client i letting it know that all other clients received its message M_1. λ_i is measured on the clock of client i as the total time elapsed between sending M_1 and receiving M_4. Since λ_i is measured on the same clock, no synchronized clocks are needed. $\lambda_i/2$ is a good estimate of time it takes for the pose of the virtual laser pointer of i to be known to all the other clients in the group.

The subjective data collected for Experiment 2 are the five-point Likert scale answers to an experience questionnaire with 4 questions:

- EQ1: It was easy to collaborate to select spheres.
- EQ2: The team worked well together.
- EQ3: The lasers of my teammates were updated without delay.
- EQ4: The system was laggy, that is it was slow to respond.

Results and discussion. Experiment 2 leveraged repeated objects in the VE to reduce download times. The average **loading times** were 6.38 $\pm$ 3.43s, which is about one fourth of the download times for Experiment 1 (Fig. 6, left, *CloVR*), which considered all VE objects unique. This demonstrates that the *CloVR* approach works for VEs with scene graphs that repeat objects. Of course, repeated objects alleviate the networking bottleneck, but not the rendering bottleneck, i.e., the VR headset cannot render all the instances of the repeated objects. The average **frame rate** for the clients was 50.9 $\pm$ 9.9fps. The **round-trip latency** λ_i had a maximum value of 195ms, and average of 58 $\pm$ 14ms.

Groups **succeeded in selecting** on average 26.3 $\pm$ 10.9 spheres in a 3min trial. Fig. 11 shows that there are more misses for more challenging spheres. The correlation coefficient is 0.927. The "hardest" sphere required over 24s and 69 misses. The "easiest" sphere was collected in 2.72s with zero misses. The average **collection time** is 4.07 $\pm$ 1.45, 4.37 $\pm$ 1.38, and 7.13 $\pm$ 5.14s, and the average number of misses is 3.56 $\pm$ 3, 3.73 $\pm$ 2.72, and 14.9 $\pm$ 17.0, for the *Easy*, *Intermediate*, and *Hard* spheres.

The **answers to the experience questionnaire** show that participants found it easy to collaborate to select spheres (3.94 $\pm$ 0.73 for EQ1), that the team worked well together (3.98 $\pm$ 0.6 for EQ2), and that the system was not laggy (3.9 $\pm$ 1.08 for EQ4, flipped for consistency). However, the participants did notice the delay in the update of the teammate virtual laser pointers (2.48 $\pm$ 0.9 for EQ3).

5 Conclusions, Limitations, and Future Work

We have presented *CloVR*, an approach for making complex virtual environments tractable on thin clients, such as a low-end all-in-one VR headset. A server deployed on a laptop reduces the VE through near-far partitioning, renders the far region geometry to an environment map, and transfers the environment map and near region geoemetry to the VR clients, where they are used to render quality frames locally, with low latency. The near region starts out small to reduce startup time, and is then increased gradually, with visual continuity. The advantages of *CloVR* are confirmed analytically, as well as in a user study with two experiments, one where participants worked independently, and one where eight participants worked collaboratively.

The *CloVR* approach reduces rendering load by only rendering geometry close to the user. The radius of this near region has to be chosen such that the amount of geometry it encompasses is tractable on the thin VR client on which the application relies. For the VEs we used, the near region radius starts at 10m and expands gradually to 63m, which provides ample space for the user to translate the viewpoint by walking. Before the user can walk to the edge of the near region, other limiting factors become relevant such as exertion or reaching the boundaries of the physical world hosting the VR application. These factors typically require teleportation, which is fully supported by the *CloVR* approach. Future work could examine providing support for VR applications that entail long distance viewpoint translation, e.g., flying a helicopter over an urban environment, which requires shifting the near region as the user viewpoint approaches its boundaries.

Another limitation is that the morph is implemented at vertex level, which can create intersection artifacts for large triangles that cross between the near and far regions. One solution is to subdivide all triangles that cross a near region growth ring, on the server, as a preprocess, but this could lead to a large number of additional triangles. A second solution, which we have adopted, is to rely on tessellation shaders to subdivide on the client, as needed, at run time. For our *City* and *Factory* VEs the subdivision was only needed for the few street, sidewalk, and floor triangles, but the solution could incur a large performance penalty for VEs that require extensive subdivision. The ultimate solution is to bypass triangle subdivision by applying the morph at fragment level, which could become practical in the future.

Our study shows that *CloVR* can quickly download a functional and visually complete version of the VE, which is then progressively refined by increasing the near region. Experiment 1 asked participants to observe passively the complete download process. Future experiments could ask participants to complete a task as quickly as possible, without waiting for the entire download to complete, which has the potential to confirm the benefit of *CloVR's* fast initial download both in terms of task completion and of participant preference metrics.

Our experiments involve a small number of participants and future work could work on gathering additional evidence for the differences between the methods. Instead of the Likert scales future work could resort to direct pairwise comparisons between methods forcing the user to choose one of the two options. Future work could also investigate the detectability thresholds for the intermediate region size and for the near region expansion speed in two-alternative forced choice (2AFC) experiments that measure the minimum size and maximum speed beyond which participants consistently identify the *CloVR* approach from a *CloVR* / ground truth pair.

Our approach supports dynamic VEs, but all dynamic objects, even those in the far region, have to be rendered through geometry. This works well when there are only a few dynamic objects in the far region, e.g., cars moving through the *City* VE or workers moving through the *Factory* VE, but could lead to an excessive rendering load in VEs where most or all objects are dynamic.

CloVR achieves scalability at the server with the number of clients in terms of computational cost by leveraging client locality. However, *CloVR* does not achieve scalability in terms of network bandwidth, as the server has to send the data to each client. Experiment 1 demonstrates that a simple laptop server can support three clients during the startup phase (i.e., download phase), and Experiment 2 demonstrates support for eight clients collaborating in real time. Future work should pursue scalability in terms of network bandwidth by leveraging the fact that all clients in the same region need the same data. One approach is to rely on clients who have received the data to send it to other clients, which requires direct client-to-client connections.

Our study placed the server and participants in the same real-world room, which corresponds, for example, to the scenario of a deploying a VR application as part of an on-campus laboratory session. Future work should investigate remote participants, who could each be at a different location, which is likely to increase the participant to participant latency. Such a geographically distributed collaborative VR scenario does require that each client have adequate network bandwidth. In addition to limiting the rendering load on the client, the near-far partitioning scheme employed by *CloVR* can also be used to adjust the amount of data that has to be transferred, and future experiments should investigate adapting the near region radius independently for each client based on the client's available bandwidth.

In order to make complex VEs tractable on thin VR clients, common practice is to over-simplify the VEs, which can lead to a cartoon-like appearance. Our work contributes to the effort of bringing to thin VR clients complex VEs without sacrificing visual detail, which is a prerequisite for VR applications in education, simulation, and training that rely on placing the user in authentic settings. The *CloVR* approach was developed and evaluated for immersive VR, but it promises benefits in the context of other forms of distributed interactive visualization on thin clients such as phones, tablets, or laptops.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 2212200, 2219842, 2309564, and 2318657.

References

[1] Oculus headset. https://www.oculus.com/. Accessed: 2023-10-01. 5

[2] Unity 3d engine. https://unity.com/. Accessed: 2023-10-01. 5

[3] J. F. Blinn and M. E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, oct 1976. doi: 10.1145/360349.360353 2

[4] T. Feng, H. Sun, Q. Qi, J. Wang, and J. Liao. Vabis: Video adaptation bitrate system for time-critical live streaming. *IEEE Transactions on Multimedia*, 22(11):2963–2976, 2019. 2

[5] L. Fink, N. Hensel, D. Markov-Vetter, C. Weber, O. Staadt, and M. Stamminger. Hybrid mono-stereo rendering in virtual reality. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 88–96, 2019. doi: 10.1109/VR.2019.8798283 2

[6] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937. 5

[7] B. Han, Y. Liu, and F. Qian. Vivo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3372224.3380888 2

[8] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of MobiSys*, 2018. 2

[9] J. Hladky, H.-P. Seidel, and M. Steinberger. The camera offset space: Real-time potentially visible set computations for streaming rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019. 2

[10] IBM Corp. IBM SPSS Statistics [Computer software]. Version 27.0, 2022. 5

[11] T. Kämäräinen, M. Siekkinen, J. Eerikäinen, and A. Ylä-Jääski. Cloudvr: Cloud accelerated interactive mobile virtual reality. In *Proceedings of the 26th ACM international conference on Multimedia*, pp. 1181–1189, 2018. 2

[12] V. Kelkkanen, M. Fiedler, and D. Lindero. Synchronous remote rendering for vr. *International Journal of Computer Games Technology*, 2021:1–16, 2021. 2

[13] J. Kim, P. Knowles, J. Spjut, B. Boudaoud, and M. Mcguire. Post-render warp with late input sampling improves aiming under high latency conditions. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 3(2):1–18, 2020. 2

[14] T. Koch and M. Wimmer. Guided visibility sampling++. *Proc. ACM Comput. Graph. Interact. Tech.*, 4(1), apr 2021. doi: 10.1145/3451266 2

[15] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, MobiCom '17, p. 409–421. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3117811.3117815 2

[16] K. Lee, J. Yi, Y. Lee, S. Choi, and Y. Kim. GROOT: A Real-time Streaming System of High-Fidelity Volumetric Videos. In *Proc. ACM MobiCom*, Sept. 2020. 2

[17] T. Liu, S. He, S. Huang, D. Tsang, L. Tang, J. Mars, and W. Wang. A benchmarking framework for interactive 3d applications in the cloud. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 881–894. IEEE, 2020. 2

[18] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., USA, 2002. 2

[19] M. Meehan, S. Razzaque, M. Whitton, and F. Brooks. Effect of latency on presence in stressful virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pp. 141–148, 2003. doi: 10.1109/VR.2003.1191132 1

[20] A. Mehrabi, M. Siekkinen, T. Kämäräinen, and A. yÏ J¨ślki. Multi-tier cloudvr: Leveraging edge computing in remote rendered virtual reality. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 17(2):1–24, 2021. 3

[21] Z. Nadir, T. Taleb, H. Flinck, O. Bouachir, and M. Bagaa. Immersive services over 5g and beyond mobile systems. *IEEE Network*, 35(6):299–306, 2021. 2

[22] J. Park, I.-B. Jeon, S.-E. Yoon, and W. Woo. Instant panoramic texture mapping with semantic object matching for large-scale urban scene reproduction. *IEEE Transactions on Visualization and Computer Graphics*, 27(5):2746–2756, 2021. doi: 10.1109/TVCG.2021.3067768 2

[23] V. Popescu, S. H. Lee, A. S. Choi, and S. Fahmy. Complex virtual environments on thin vr systems through continuous near-far partitioning. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 35–43. IEEE, 2022. 2

[24] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan. Toward practical volumetric video streaming on commodity smartphones. In *Proc. of ACM HotMobile*, 2019. 2

[25] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of MOBICOM*, 2018. 2

[26] A. Schollmeyer, S. Schneegans, S. Beck, A. Steed, and B. Froehlich. Efficient hybrid image warping for high frame-rate stereoscopic rendering. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1332–1341, 2017. doi: 10.1109/TVCG.2017.2657078 2

[27] M. Schütz, K. Krösl, and M. Wimmer. Real-time continuous level of detail rendering of point clouds. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 103–110, 2019. doi: 10.1109/VR.2019.8798284 2

[28] M. Stengel, Z. Majercik, B. Boudaoud, and M. McGuire. A distributed, decoupled system for losslessly streaming dynamic light probes to thin clients. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pp. 159–172, 2021. 2

[29] P. Stotko, S. Krumpen, M. B. Hullin, M. Weinmann, and R. Klein. Slamcast: Large-scale, real-time 3d reconstruction and streaming for immersive multi-client live telepresence. *CoRR*, abs/1805.03709, 2018. 2

[30] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 4

[31] F. Wilcoxon. *Individual comparisons by ranking methods*. Springer, 1992. 5

[32] A. Zhang, C. Wang, B. Han, and F. Qian. YuZu: Neural-Enhanced volumetric video streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 137–154. USENIX Association, Renton, WA, Apr. 2022. 2

[33] S. Zhao, H. Abou-zeid, R. Atawia, Y. S. K. Manjunath, A. B. Sediq, and X.-P. Zhang. Virtual reality gaming on the cloud: A reality check. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. IEEE, 2021. 2

[34] P. Zhou, Y. Xie, B. Niu, L. Pu, Z. Xu, H. Jiang, and H. Huang. Qoe-aware 3d video streaming via deep reinforcement learning in software defined networking enabled mobile edge computing. *IEEE Transactions on Network Science and Engineering*, 8(1):419–433, 2020. 3

[35] Y. Zhou, L. Wu, R. Ramamoorthi, and L.-Q. Yan. Vectorization for fast, analytic, and differentiable visibility. *ACM Trans. Graph.*, 40(3), jul 2021. doi: 10.1145/3452097 2