2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR) | 978-1-6654-5325-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISMAR55827.2022.00017

Complex Virtual Environments on Thin VR Systems Through Continuous Near-Far Partitioning

Voicu Popescu* Purdue University
Seung Heon Lee† Purdue University
Andrew Shinyoung Choi‡ Purdue University
Sonia Fahmy§ Purdue University

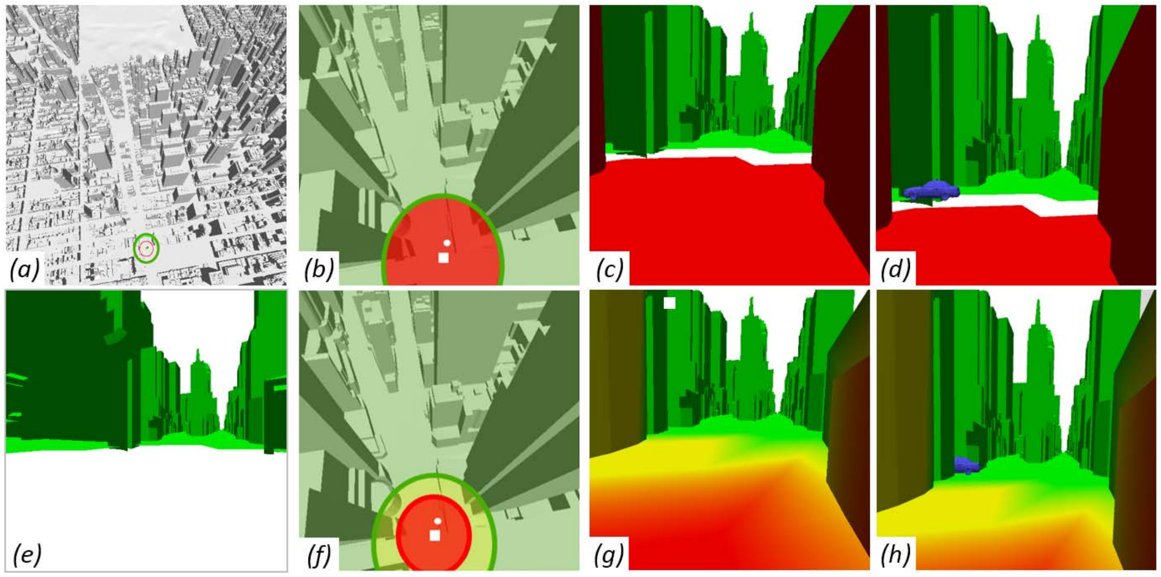


Figure 1: An urban virtual environment (VE) with 3.7M triangles *(a)* is too large to be rendered on a self-contained VR system. The VE is partitioned *(b)* into a near region of 30m radius (red), to be rendered as usual, from geometry, and a far region (green), to be rendered from an environment map *(e)*. Since the near region contains only 371 triangles, this conventional near-far partitioning greatly reduces the rendering load, but it produces a visualization discontinuity *(c)* between the near (red) and far (green) regions. Our method uses an intermediate region (yellow in *f*) rendered with a morph that preserves visualization continuity between near and far *(g)*. Our method also supports dynamic far objects with correct depth compositing (car in *h* vs *d*).

Abstract

This paper describes a method for reducing rendering load such that complex virtual environments (VEs) can be deployed on ”thin” VR systems with limited rendering power. The method partitions the VE into four regions: a near region, an intermediate region, a stationary far region, and a dynamic far region. The stationary far region is replaced with an environment map, which brings a substantial rendering load reduction. The other three regions are rendered from geometry: the near region is rendered from the user viewpoint, the dynamic far geometry is rendered from the center of the environment map, and the intermediate region is rendered with a morph that switches viewpoint gradually from the user viewpoint to the center of the environment map. The intermediate region connects the near and far regions seamlessly. Furthermore, the environment map is enhanced with per pixel range which allows depth compositing the dynamic and stationary far geometry. An IRB approved user study ($N = 22$) found significant advantages for our method over conventional near-far partitioning.

*e-mail: popescu@purdue.edu
†e-mail: lee3072@purdue.edu
‡e-mail: choi681@purdue.edu
§e-mail: fahmy@purdue.edu

Index Terms: Computing methodologies—Computer graphics—Graphics systems and interfaces—Virtual reality;

1 Introduction

Advances in virtual reality (VR) technology have produced self-contained VR systems with on-board rendering, tracking, and power (e.g., Meta Quest 2 [1]). The user freedom brought by untethered VR headsets comes at a cost. Compared to desktop workstations, self-contained VR headsets cannot render complex virtual environments (VEs) at the high frame rates needed for a comfortable VR user experience. The power consumption of workstation-grade graphics processing units (GPUs) precludes their use in battery-powered VR systems. What is needed is to reduce rendering load while preserving output frame quality, such that complex VEs can be deployed on ”thin” VR systems with limited capabilities. Since reducing rendering load inherently reduces the amount of data needed to render output frames, reducing rendering load is also beneficial in a distributed VR context, where it reduces bandwidth requirements as well as download times at startup and after teleportation.

Reducing rendering load is a long standing problem in computer graphics research with no perfect solution. One approach is to compute the part of the VE that the user could potentially see from the current region, and to limit rendering to this potentially visible set [9]. However, computing the potentially visible set in a complex VE is challenging. Furthermore, the size of the potentially visible set varies greatly with the user location within the VE, and it can become intractably large when the user has line-of-sight "deep" into the VE. Another approach is to adapt the level-of-detail of the VE [12], for example by reducing the geometric complexity of distant parts of the VE that have a small output frame footprint. Like for visibility computation, adapting the geometric level-of-detail of a complex VE is challenging, and the rendering load remains unbounded.

A third approach is to partition the VE into a near and a far region, and to replace the far region geometry with a precomputed environment map [3]. Computing the partition based on distance from the user is straight forward, and the rendering load is reduced to rendering the near region. However, such a conventional near-far partitioning approach has two fundamental limitations that have to be overcome for it to be suitable to the VR context.

The first limitation is a visualization discontinuity between the near region, which is rendered from the user viewpoint, and the far region, which is rendered from the center of the environment map. Whereas this discontinuity is acceptable when the environment map approximates parts of the VE that are very far away, such as a cloudy sky or mountains at the horizon, the discontinuity creates objectionable artifacts when the environment map is called upon to approximate medium distance parts of the VE, as needed to meet the rendering budget of thin VR systems. In Fig. 1, near-far partitioning reduces the rendering load of the *Manhattan* urban VE by four orders of magnitude, down to a level that can be comfortably handled by any VR system. However, the near and far regions are disconnected in the output frame *(c)*, since one is rendered from the user viewpoint (white dot in *b*) and one from the center of the near region (white square in *b*). The second limitation is that, since the environment map is precomputed, the far region cannot be dynamic. Rendering a far dynamic object from the user viewpoint places the object at the wrong location and with incorrect visibility (*d*).

In this paper, we describe a method for the near-far partitioning of a VE that maintains visualization continuity between the near and far regions and supports dynamic far region objects. The method partitions the VE into four regions: a near region, an intermediate region, a stationary far region, and a dynamic far region. The near region is rendered conventionally, from geometry. The stationary far region is rendered through environment mapping. The intermediate region is rendered from geometry, with a 3D morph that changes viewpoint gradually from the user viewpoint to the center of the environment map, maintaining visualization continuity. The dynamic far region is rendered from geometry, which allows updating its position at each frame. The dynamic far region is rendered from the center of the environment map and the environment map is enhanced with per-pixel range, which allows for accurate per-pixel depth compositing of the dynamic and stationary parts of the far region. In Fig. 1, our method uses the intermediate region to connect the near and far regions *(g)*. Furthermore, our method renders the blue car at the correct location within the environment map, and with correct per-pixel depth compositing *(h)*.

In summary, our method brings the significant rendering load reduction of conventional environment mapping *without* its significant loss in visual quality. On a Quest 2 VR headset, the frame rate for the *Manhattan* VE is restored from 12fps to a full 70fps, while the environment mapped part of the frame remains well integrated with the part rendered from geometry, and not just a disconnected and inert background.We have conducted a user study ($N = 22$) with our Institutional Review Board's (IRB) approval. The study confirmed that our approach has a significant advantage over conventional near-far partitioning in terms of avoiding a gap between the near and far regions, of conveying directions consistently across the near-far boundary, and of projecting dynamic far geometry correctly relative to stationary far geometry rendered through environment mapping. Furthermore, in a majority of cases (56%), participants did not notice the curvature of trajectories crossing the intermediate region. We also refer the reader to the accompanying video.

2 Prior work

Computer graphics research has approached the problem of rendering load reduction from two directions. One is through visibility computation, which aims to find all and only the parts of the scene that are visible from a given view region. Visibility algorithms fall into two categories. One category is that of sampled-based visibility algorithms, which probe the scene with rays that originate in the view region and accumulate the visible set from the visible triangles found by each ray [9]. Sampled-based algorithms are fast, but the resulting visible set might miss some visible triangles, because ray probing is heuristic, and because no triangle that is not in the visible set can be *confirmed* as hidden, as that would require an infinite number of rays. A second category of visibility algorithms analyze the space of rays originating in the view region continuously, for example by beam tracing [28]. The resulting visible set is complete, but continuous visibility algorithms are slow. The camera offset space approach formulates a hybrid visibility solution by investigating the camera translations under which an image plane sampling location is covered by a triangle. The resulting visible set contains most and not only visible triangles, i.e., false negatives and false positives are possible. The visibility approach to rendering load reduction has the advantage of quality: when all visible triangles are found, the output frames are indistinguishable from those obtained from the entire scene. However, even when the visible set contains only and all visible triangles, the visible set is unbounded and can be large. For the example in Fig. 1, the avenues and streets that intersect the near region provide line-of-sight deep into the urban VE, and the resulting visible set can have a large number of triangles. Another disadvantage is that visibility computation is complex, which requires pre- or co-processing.

A second direction from which rendering load reduction has been approached is level-of-detail (LoD) adaptation. Given a view region, the goal is to compute an alternate, simpler representation of the scene geometry that produces similar results to the original, full-complexity representation [12]. LoD adaptation on a continuum scale, that meets a given triangle budget, and that minimizes output image quality loss is a challenging problem.

Despite decades of research, the only LoD adaptation scheme used in practice is *environment mapping*, which approaches its fiftieth anniversary [3]. Environment mapping approximates the far region with a panoramic image that captures the scene appearance in all directions, using, for example, a spherical, equirectangular, or cube map ray parameterization. Environment mapping has the advantages of: (1) versatility, as it is guaranteed to work with any scene, no matter how complex; (2) fast construction, as it can be rendered with the help of graphics hardware; and (3) bounded complexity, as it replaces the far region with an image of fixed resolution. However, environment mapping suffers from limitations that preclude its use outside rendering reflections on curved surfaces, or rendering very distant geometry. The limitations are: (1) lack of motion parallax in the output frame in response to user viewpoint translates, as the 3D geometry of the far region was replaced with a 2D image; (2) lack of continuity between the near and far regions, which are rendered from different viewpoints; (3) lack of support for dynamic scenes, as the far region is frozen in time at the moment when the environment map was constructed. In this paper we propose a near-far partitioning scheme that addresses the lack of continuity and lack of dynamic scene support of conventional environment mapping.

Leveraging the environment mapping premise that distant parts of the scene have an approximately constant projection under small viewpoint translations, researchers have developed a method that reduces rendering load in VR by reserving the stereoscopical rendering to the near parts of the scene, and by rendering the distant parts monoscopically [5]. Since most of the scene complexity is in the far region, the method can achieve a rendering load reduction of up to a factor of two, which is not sufficient for complex scenes such as the urban environment from Fig. 1. The frame discontinuity at the transition between the cyclopean eye, from where the far region is rendered, and the user eyes, from where the near region is rendered, is easily handled with a small overlap between the three frusta, leveraging the small viewpoint translation caused by the interpupillary distance. Our method has to maintain near-far continuity for large translations between the user and the environment map viewpoints, which is done by morphing the intermediate region geometry.

Reducing rendering load has also been investigated in the distributed VR context, where the VE is stored on a server and the application is deployed on clients connected to the server via a wireless network. In addition to making VR applications tractable on thin clients such as phones, tablets, and VR headsets, reducing rendering load and thereby the amount of data needed to render output frames also reduces the bandwidth requirements over the unpredictable wireless networks. In 360° VR, reducing the data that has to be transferred to and rendered at the client is implemented through view-frustum culling, i.e., the transfer is limited to the part of the 360° frame encompassed by the predicted user view [7, 18]. View-frustum culling provides sufficient data reduction in this context since, like a conventional video frame, a 360° video frame has a depth complexity of one. In the case of volumetric video streaming, the pixels are 3D points with color, which also allows the user to change viewpoint and not just view direction. Since depth complexity is not one anymore, view frustum culling was enhanced hierarchical LoD schemes (e.g., an octree) to reduce the detail of occluded and distant parts of the scene [6, 11, 17, 27].

When the VE geometric model contains triangle meshes, which is the standard for modeling 3D VEs, view frustum culling and simple LoD schemes are insufficient. One solution is to restrict the user to a set of possible viewpoints, e.g., by discretizing the ground plane with uniform 2D grid, and to precompute environment maps for the set of viewpoints surrounding the current user position on the server [10]. Since the VE geometry is replaced by an environment map, rendering load at the client is reduced to the few objects with which the user interacts. The shortcomings are lack of support for continuous viewpoint translation, as the user has to jump from one available viewpoint to the next, and reliance on a server to compute environment maps. As the number of clients increases, so does the bandwidth required to transfer their environment maps from the server. This bottleneck is alleviated by increasing the similarity–and therefore the compression efficiency–of nearby environment maps [14]. The near-far discontinuity is avoided by again clamping the user viewpoint to a discrete set of possible locations.

Our method bridges the gap between the user viewpoint and the environment map center, and the resulting frame is multiperspective. Previous VR research has used multiperspective images to increase VE exploration efficiency by allowing the user to see around the corner [26], to improve collaboration in VR by alleviating the difference in occlusion between the collaborators' view of the VE [24], and to facilitate selection by increasing the frame footprint of selection candidates [23]. Prior work [25] has established that user disorientation and cybersickness are alleviated if the part of the VE close to the user is rendered conventionally, responding as expected to the user's view changes. Our method adheres to this rule as it renders the near region conventionally, from the user viewpoint.

One approach developed by VR research for controlling the rendering time of complex VEs is based on discretizing the VE to facilitate level-of-detail (LoD) adaption. One method represents the VE as a point cloud with a continuous level-of-detail [20]. The method leverages state of the art GPUs to handle over 17 billion points per millisecond. Another server-client streaming solution uses voxels for transmission, which are then converted to meshes using marching cubes for rendering [21]. A technique specifically aimed at large urban environments represents the VE with a collection of images [16]. The temporal and spatial coherence of the VR output frames has been exploited by 3D image-warping to improve frame rate and to generate the left and right eye frames at a cost smaller than that of rendering the VE twice [19]. These VE discretization approaches are complex and therefore not suitable for thin VR systems. Furthermore, although the VE discretization brings data uniformity which aids with LoD adaptation, the discretization also makes the output frame reconstruction more challenging.

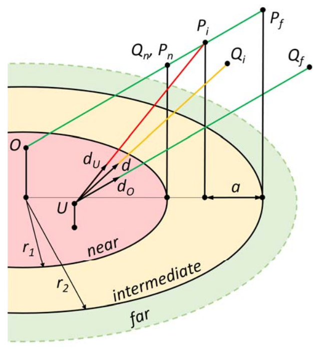


Figure 2: Intermediate region vertex morphing.

3 Continuous Near-Far VE Partitioning

We have developed a method that provides good control of the rendering load in a VR application. The VE is partitioned into a near, an intermediate, and a far region using two co-axial cylinders of radius r_1 and r_2 (Fig. 2) . In a preprocessing step, a cube map is rendered from a viewpoint O on the cylinder axis using the stationary far region triangles . Then, at run time, each output frame is initialized from the cube map and the VE triangles not used to render the cube map are rendered as follows: (1) near region triangles are rendered as usual; (2) intermediate region triangles are rendered with a vertex morph that ensures continuity between the near and far regions, as described in Sec. 3.1; (3) far region dynamic triangles are rendered as described in Sec. 3.2.

3.1 Near to Far Morphing

Given a 3D vertex P_i in the intermediate region (Fig. 2), its position is first morphed to 3D point Q_i, and then Q_i is projected conventionally. Q_i is computed with the steps shown in Eq. 1.

$$\begin{aligned} Q_i &= U + dt \\ d = d_U w_U + d_O w_O;\;\; d &= d/\|d\| \\ w_U = a/(r_2 - r_1);\;\; w_O &= 1 - w_U \\ t &= \|UP_i\| w_U + \|OP_i\| w_O \end{aligned} \tag{1}$$

Q_i is placed at distance t from the user viewpoint U along direction d. d is computed by interpolating the directions d_U and d_O in which P_i is seen from U and O. The interpolation weights w_U and w_O are defined by the relative position of P_i in between the two cylinders. Distance t is computed similarly by interpolating the distances from U and O to P_i. The morph switches gradually from a projection from U, when P_i is close to the near region, to a projection from O, when P_i is close to the far region, providing continuity across the near-to-intermediate and intermediate-to-far boundaries. Indeed, the morph does not displace a vertex P_n on the near-to-intermediate boundary, i.e., $Q_n = P_n$, which ensures continuity with the near region geometry that is rendered conventionally. Furthermore, the morph displaces a vertex P_f on the intermediate-to-far boundary to Q_f, which maps to P_f in the cube map, ensuring continuity with the far region geometry that is rendered through environment mapping.

3.2 Dynamic Far Region Support

The triangles of the far region that are dynamic have to be rendered from geometry, such that their position can be updated for every frame. These dynamic triangles belong to the same VE region as the stationary triangles from which the cube map was rendered, so the dynamic triangles have to be rendered in a way that is consistent with the cube map. Consistency requires meeting two conditions. The first condition is that a dynamic far region vertex P be projected as if it were looked up in the cube map, by displacement to Q followed by conventional projection. The displacement is given in Eq. 2, and it is a special case of the intermediate region vertex morph from Eq. 1 with $w_U = 0$ and $w_O = 1$.

$$Q = U + P - O \tag{2}$$

The second condition is that the stationary and dynamic far region triangles be depth composited accurately, as it can happen that stationary triangles are closer than dynamic triangles, so it is no longer the case that the cube map background can always be safely overwritten. The first step is to enhance the cube map with per pixel range. Range is defined as the Euclidean distance between a viewpoint and a surface point. Unlike z which is measured perpendicularly to the image plane, range is independent of view direction and it allows for direct depth comparisons along the same ray. The z-buffers obtained when the cube map is pre-rendered are converted to range buffers and provided as a depth cube map to the fragment program that renders the far region dynamic triangles. The program culls a fragment whose range r_U is greater than the range r_O in the cube map at the same location, as shown in Eq. 3.

$$r_U = \|Q - U\|;\ d = (Q - U)/r_U;\ r_O = CubeMap(d).range \tag{3}$$

The range r_U of the current fragment is the distance from the viewpoint U to the displaced world position Q of the fragment passed down from the vertex shader that implements Eq. 2. The cube map range r_O is looked up using the direction from the user viewpoint U to the displaced fragment Q.

4 Results and Discussion

We have implemented our method using Unity [2] and deployed it on a Quest 2 all-in-one VR headset [1]. Our implementation processes all run-time geometry, i.e., the near, intermediate, and dynamic far geometry, with a unified pipeline: a vertex is classified in the near, intermediate, and far region, near vertices are not displaced ($w_U = 1$ and $w_O = 0$ in Eq. 1), intermediate vertices are morphed ($w_U > 0$, $w_O > 0$, $w_U + w_O = 1$), far vertices are displaced for environment mapped projection ($w_U = 0$ and $w_O = 1$). All fragments are compared with the environment map depth channel and culled if hidden by the environment mapped geometry (Eq. 3).

4.1 Rendering Load Reduction

The recommended upper limit on the number of triangles for Quest 2 is 750k-1M [15]. Quest 2 cannot render our *Manhattan* model that has 3M triangles–the sluggish frame rate and the flickering induces cybersickness even after a short exposure. Since our method renders the parts of the VE in the near and intermediate regions from geometry and the parts in the far region from an environment map, rendering load is controlled through parameter r_2 that defines the intermediate to far boundary. r_2 is selected based on the rendering capabilities of the VR system: the smaller r_2, the fewer triangles have to be processed, but also the more of the VE is approximated with environment mapping. Fig. 3 shows the number of triangles that have to be rendered as a function of r_2 for our *Manhattan* VE, as well as the rendering load reduction. The rendering load reduction is the percentage of the VE triangles that are in the far region and

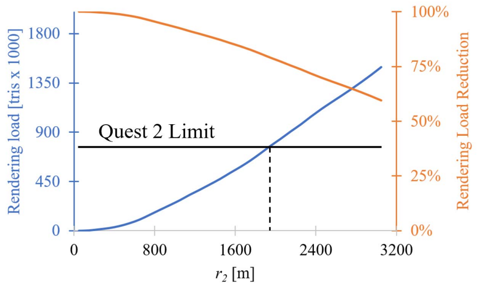


Figure 3: Rendering load dependence on the radius r_2 of the far region boundary.

thus are not rendered. The user region is in *Times Square* as shown in Fig. 1, where $r_2 = 50$m and the rendering load is below 1,000 triangles. Rendering load remains well within the capabilities of the Quest 2 headset even when the far region starts at $r_2 = 1,000$m. Conversely, for $r_2 = 50$m, near-far partitioning allows rendering a much higher density model.

Given a value for r_2, the VE geometry is partitioned into two bins: the static triangles in the far region (*SF* bin), and all the other VE triangles (*NSF* bin, $NSF = VE - SF$). To avoid hairline fractures between the environment mapped part of the frame and the part rendered from geometry, both *SF* and *NSF* contain the triangles with vertices on both side of the r_2 boundary. The *SF* triangles are prerendered to a cube map conventionally. The *NSF* triangles are rendered with a vertex shader and a fragment shader that implement the morph and the depth compositing with the cube map. The vertex shader displaces the input vertex based on its location within the near, intermediate, or far regions of the VE. Near vertices are not displaced. Far vertices are displaced with a simple translation (Eq. 2). The main computational cost of the morph comes from the displacement of the intermediate region vertices using Eq. 1, which implies a small number of 3D vector operations. The extra computational cost brought by the fragment shader is small, and it is dominated by the additional cube map range lookup (Eq. 3). Whereas rendering the entire *Manhattan* VE from geometry runs at 12fps, with constant flickering due to missing frames, with our continuous near-far partitioning scheme the frame rate is 70fps.

4.2 Visual Quality

Our method brings the rendering load reduction of conventional near-far partitioning while improving the output visual quality.

Near-far continuity. A first aspect of visual quality is achieving continuity between the part of the frame rendered from geometry and the part rendered with environment mapping. Conventional near-far partitioning leaves a gap between the near and far regions which results in objectionable visual artifacts (Fig. 1 *c* and *d*). Of course, one cannot hide the gap with an environment map that is rendered from *all* the triangles in the VE and not just from those in the far region, as this would fill in the gap by repeating parts of the VE, resulting in a redundant visualization. Our intermediate region morph guarantees continuity between the near and far regions, without any redundancy (Figs. 1, 4, and 5).

Support for dynamic far geometry. A second aspect of visual quality is whether dynamic far geometry is supported, and how. Conventional near-far partitioning does not have any good option for rendering dynamic far geometry: one option is to omit the dynamic geometry; another option is to render the dynamic far geometry from the environment map viewpoint, but then geometry that moves from near to far will have a trajectory discontinuity as the viewpoint switches abruptly at the near-far boundary; a third option is to render the dynamic far geometry from the user viewpoint, which provides a continuous trajectory, but which projects the dynamic far geometry

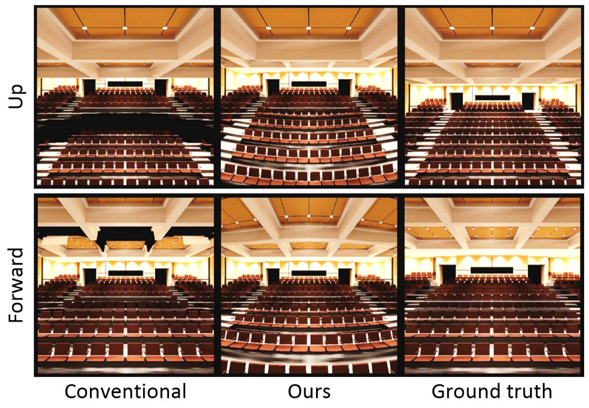


Figure 4: Comparison between the three methods (columns) for two viewpoint translations (rows), on an auditorium VE.

at a frame location inconsistent with that of the static far geometry rendered through environment mapping, and with no depth compositing (Fig. 1 *d*). With our method, geometry moving from near to far has a continuous trajectory in the frame, and the projection of the moving geometry is consistent with the static geometry it passes by. Furthermore, with our method the environment map is not just a 2D background image, but rather a range image that allows for accurate depth compositing at pixel level (Fig. 1 *h*).

Geometric distortion. A third aspect of visual quality is the distortion introduced by our morph at the intermediate region. The morph switches visibility from the user to the environment map viewpoint. The gradual viewpoint change distorts the intermediate region geometry, i.e., straight lines have a non-linear projection in the output frame. The distortion introduced by the morph is controlled by the intermediate region size $r_2 - r_1$. The larger the intermediate region, the slower the pace of the viewpoint change, but also the larger the region where distortions occur. For the images in Fig. 1, $r_1 = 30m$ and $r_2 = 50$m, which provides a sufficiently large transition region of 20m.

Approximations introduced by environment mapping. A fourth aspect of visual quality are the various approximations introduced by environment mapping. While our method integrates seamlessly the environment mapped part of the frame with the part rendered from geometry, the limitations of environment mapping remain.

One is that the environment mapped part of the frame is not rendered from the user viewpoint, but rather from the center of the environment map, so the environment mapped geometry appears in the frame at an approximate location. The approximation error is minimized by rendering the environment map from a viewpoint O at a typical VE viewing height. This is done such that the cube map approximates the user's view of the far region as well as possible, and it does *not* restrict the user to the height of O.

A second environment mapping limitation is the lack of motion parallax as the user viewpoint translates. When the user view change is a pure translation, i.e., without any rotation, the part of the frame rendered by environment mapping does not change. However, such a pure translation is atypical in VR where the user selects the view by moving their head and by walking. The rotation present in such view changes helps hide the environment mapping approximation better than, for example, when using a ”WASD” keyboard interface of a desktop interactive visualization application.

A third limitation is that the environment mapped part of the frame does not provide correct depth cues when viewed stereoscopically, i.e., the left and right eye image disparity is not correctly modulated by the variable depth of the various parts of the environment map. Although our method enhances the environment map with per pixel range, the additional channel is only used for depth compositing and not to render the environment map in 3D. Doing so would increase the rendering load, as each environment map pixel becomes a vertex, and would create disocclusion errors as at least one of the eyes would see VE surfaces not captured by the environment map.

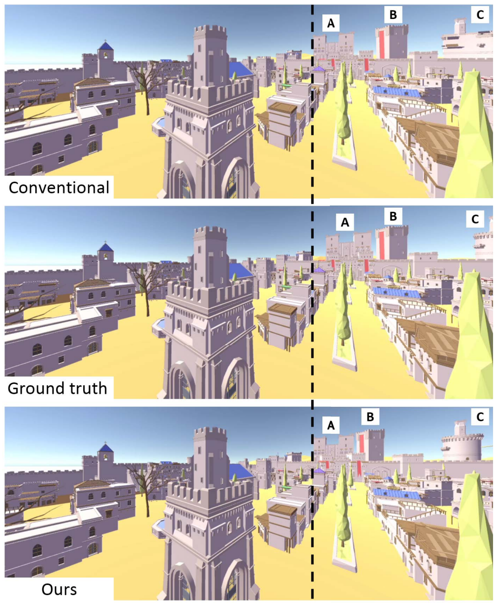


Figure 5: Comparison between the three methods on a medieval city. The near-far partitioning boundary passes through building C, breaking it apart in ”Conventional”. Furthermore, ”Ours” conveys the relative position of buildings A and B more accurately than ”Conventional” because B is in the intermediate region and is morphed towards A, which is in the far region.

4.3 Image Quality Metrics

To compare the conventional, discontinuous near-far partitioning to our continuous near-far partitioning, the first step is to select a suitable image quality metric. The field of image and video processing has developed several image quality metrics that quantify the error introduced by various processing pipelines, such as compression. Examples include PSNR [8], RMSE [4], structural similarity [8], or the more recent fovvideovdp [13], which differentiates between the focus and context regions of the frames. However, these quality metrics are suitable for in-place image transformations, whereas the discontinuous and continuous near-far partitioning methods amount to transformations that displace entire regions of the image, i.e., the far region is rendered from the center of the environment map. These conventional quality metrics assume an identity mapping between the truth and processed images, and do not discern between two different ways of perturbing this mapping. In Fig. 6, a truth image a is partitioned into a top and a bottom partition, and the bottom partition is shifted up and right, which results in a discontinuous image (b). Image c uses a piece of the bottom partition to define an intermediate partition that is then warped to connect the bottom and top partitions continuously. Since the bottom partition is identical to ground truth, and since the bottom partition shrinks from b to c, a metric that does not penalize for discontinuities will prefer b to c. Indeed, the PSNR values for *a-b* and *a-c* in Fig. 7 are 34.76 dB and 32.30 dB, respectively. Similarly, fovvideovdp reports a Q_JOD of 4.125 for conventional and of 3.466 for our method.

We compare the quality of the two near-far partitioning schemes with a discerning metric that quantifies image continuity. Image continuity is based on projection continuity (Eq. 4): as the distance $\|h\|$ between two 3D points $P+h$ and P decreases to 0, so does the

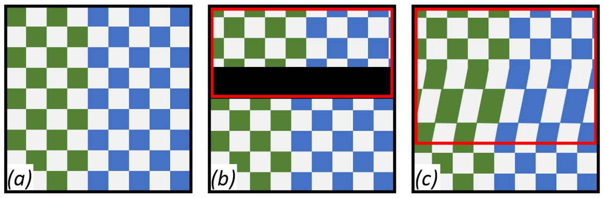


Figure 6: Truth image (*a*), discontinuous partitioning (*b*), and continuous partitioning (*c*). Metrics that assume an identity mapping from truth to processed image will incorrectly judge *b* as better than *c* because *c* differs from *a* over a larger area than *b* (red rectangle).

distance between their image projections $\Pi(P+h)$ and $\Pi(P)$.

$$\lim_{\|h\|\to 0} \|\Pi(P+h)-\Pi(P)\| = 0 \qquad (4)$$

Given a processed image I, rendered either with the conventional or with our near-far partitioning method, and a ground truth image I_0, rendered from geometry, we estimate the discontinuity at pixel p of I with Eq. 5.

$$\delta(p) = \frac{1}{|N_p|} \sum_{q\in N_p} \frac{\|p-q\|}{\|\Pi_0(\Pi^{-1}(p)) - \Pi_0(\Pi^{-1}(q))\|} - 1 \qquad (5)$$

$\delta(p)$ is the average discontinuity over the neighborhood N_p defined in Eq. 6: a pixel q of I is in N_p if its projection $\Pi_0(\Pi^{-1}(q))$ on the ground truth image I_0 is within ε of that of p.

$$N_p = \{q \in I \mid \|\Pi_0(\Pi^{-1}(q)) - \Pi_0(\Pi^{-1}(p))\| < \varepsilon\} \qquad (6)$$

Referring to Eq. 5 again, a pixel q in N_p contributes to the discontinuity at p the ratio between the distance between p and q in I and the distance between p and q in I_0. If p and q project at nearby locations in I_0 but far apart in I, I is discontinuous at p. Since I_0 has a ratio of 1 over all its pixels, the average is decreased by one for a discontinuity of 0 for I_0.

Fig. 7 shows that conventional near-far partitioning (*b*) has a large discontinuity value close to the boundary between near and far; the boundary splits the ground plane (blue) and the left building (yellow), and samples that should project close to one another appear at distant locations in the image. Our continuous near-far partitioning (*c*) projects nearby scene 3D points to nearby image locations, resulting in small discontinuity values at the yellow and blue locations; the red location is close to the near region but still in the intermediate region resulting in a small residual discontinuity value; the green location is in the far region, resulting in the same small discontinuity value of *b*. In the ground truth image *a* δ is 0 at all four locations, and all four neighborhoods are perfect disks with the radius equal to the parameter ε chosen to define the neighborhood (Eq. 6, here ε = 17 pixels). Fig. 8 illustrates the discontinuity metric over entire images: compared to conventional near-far partitioning (*a*), our method (*b*) reduces discontinuity by an order of magnitude.

4.4 User Study

Participants. We have recruited $N = 22$ participants from our university campus community. There were 7 participants aged 30 or more, and the average age of the participants younger than 30 was 23. Five participants were women; 3 participants had never used a VR system before, 5 once, 9 occasionally, and 5 frequently.

Conditions. The study has a within-subject design where all participants performed all tasks under all conditions. There were three conditions. Conventional near-far partitioning was used as a control condition (CC). Our continuous near-far partitioning was the experimental condition (EC). Rendering the entire VE from geometry served as a ground truth condition (TC).

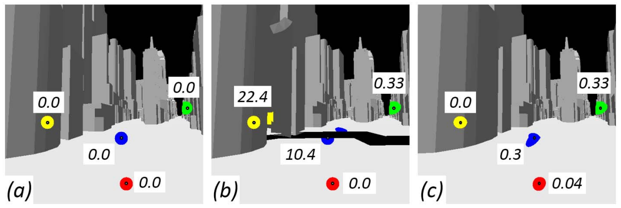


Figure 7: Discontinuity δ values at four image locations, for ground truth (*a*), for conventional near-far partitioning (*b*), and for our method (*c*). The location is shown with a black dot and the neighborhood over which discontinuity is computed is shown with color, i.e., red, green, blue, and yellow.

Tasks and Hypotheses. Each participant performed three tasks. The **first task** asks the participant to count red spheres in the *Manhattan* VE (Fig. 9). The number of spheres is between 11 and 13. The participant stands in a default position from where all spheres are visible, which avoids the need of teleportation or redirection. Counting spheres has only the role of making the participant inspect the VE, and is not a differentiating factor between the three conditions. Once the participant selects the number of spheres from three possible choices, the participant is asked the yes/no question "Is there anything wrong with the environment?" (*Q1*). This first task was run only in the CC and EC conditions as rendering the entire VE from geometry had a low frame rate (12fps) and flickering due to missing frames, which triggered cybersickness even after a short exposure. The CC condition shows a gap between near and far, and the related experimental hypothesis is that participants will find that there is something wrong for CC and not for EC (*H1*).

The **second task** uses a VE with one near red rectangle and with three distant rectangles, one blue, one green, and one yellow. The distant rectangles are in the far region, and are thus rendered with environment mapping. The near rectangle ends short of the far region, and is thus rendered with the morph for EC and from the user viewpoint for CC. The participant is asked the question: "Where does the red tape point?" (*Q2*). The possible answers are three colored squares corresponding to the far rectangles. In the CC condition, the distant and near rectangles are misaligned because they are rendered from different viewpoints. The misalignment is sufficient for the near rectangle to appear not aligned with any of the distant rectangles, e.g., CC_1 in Fig. 10, or even to appear well aligned with an incorrect rectangle, e.g., CC_2 in Fig. 10). CC also magnifies the gap between the near and distant rectangles, e.g., CC_1 vs. TC. In the EC condition, the morph bends the near rectangle over the intermediate region, aligning it with the correct far rectangle. The related experimental hypothesis is that participants will choose the correct far rectangle for EC and TC, and that their answers for CC will be consistently wrong (*H2*). A participant repeats Task 2 three times for each of the three conditions, each time with a different orientation of the near rectangle.

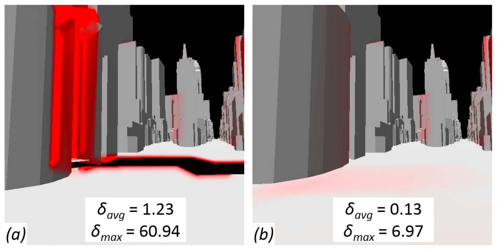


Figure 8: Discontinuity δ for conventional (*a*) and our (*b*) near-far partitioning highlighted in red.

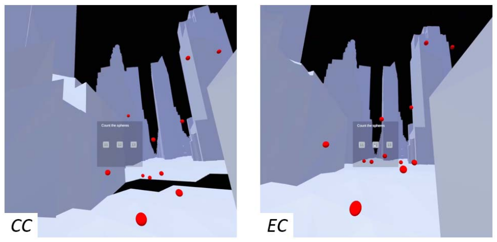


Figure 9: Task 1 frames in the control and experimental conditions.

The **third task** uses a VE with distant boxes with unique colors, which are in the far region of the VE and are thus rendered via environment mapping for EC and CC (Fig. 11). A white sphere flies from the near region, in a straight line, to land on one of the boxes in the far region. After it lands, the participant is asked two questions. The first question is a yes/no question: ”Did the sphere move on a straight line?” (*Q3*). The related experimental hypothesis is that participants will indicate that the CC and TC trajectories are linear and that the EC trajectories are curved (*H3*). The second question is: ”Where did the sphere land?” (*Q4*), with the possible answers given by small squares of the color of the boxes (similar to Task 2, see Fig. 10). The related experimental hypothesis is that participants will answer correctly for EC and TC, and incorrectly for CC (*H4*). A participant repeats Task 3 three times for each of the three conditions, each time with a different sphere trajectory.

Data collection and analysis. In addition to the demographic data collected via a pen and paper questionnaire, the system saves the participant's answers to each question along with the correct answer in a file stored on the VR headset. The data is tabulated and proportions of ”yes” (*Q1, Q3*) or of correct (*Q2, Q4*) answers are computed for each of the three conditions, over all participants.

Our data is paired, i.e., each participant answers questions in all conditions, and it contains values of a dichotomous variable, i.e., ”yes”/”no” for Q1 and Q3, and correct/incorrect for Q2 and Q4. Therefore we investigate the statistical significance of the difference between pairs of proportions using McNemar's test . Furthermore, since the number of participants is rather small, we do not *approximate* the distribution of our data with the χ^2 distribution, but rather perform the exact binomial test. We compute McNemar's exact test from the 2x2 contingency table, i.e., (condition1, condition2) $\times$ (answer1, answer2), using the Omni calculator [22].

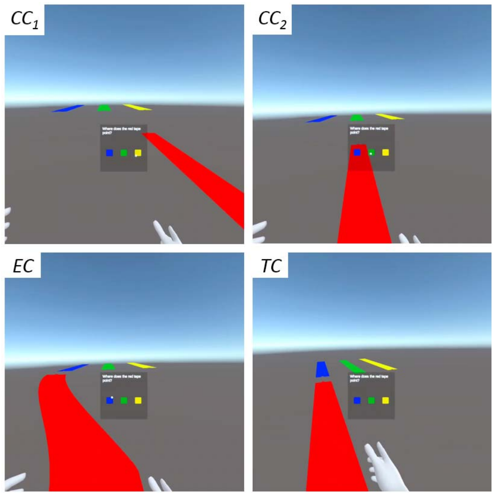


Figure 10: Task 2 frames in the control (CC_1 and CC_2), experimental (EC), and ground truth (TC) conditions.

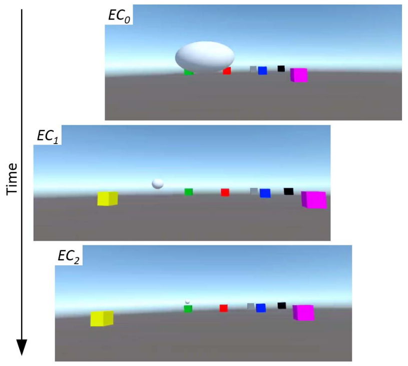
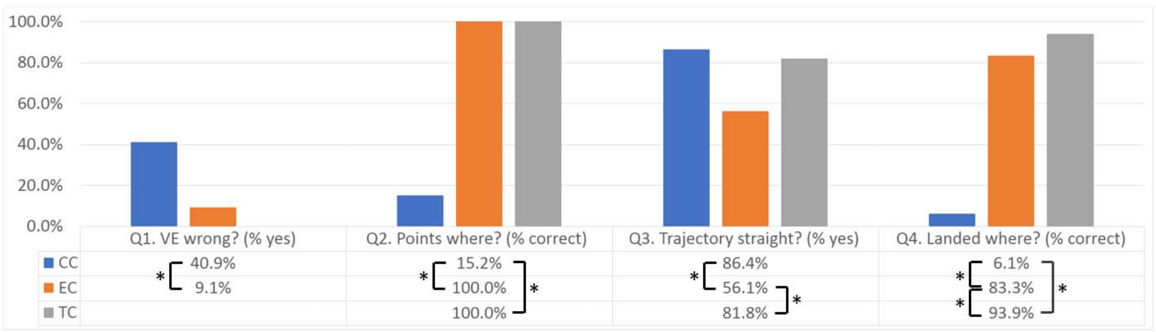


Figure 11: Task 3 frames in the experimental condition, in chronological order. The white sphere moves in a straight line in the VE. The frames are shifted left-right in the figure to align their azimuths for a better illustration of the curved image space trajectory of the white sphere, which starts out *right* of the green box (EC_0), then moves *left* of it (EC_1), to finally land *on* the green box (EC_2).

Results and Discussion. The proportions of ”yes” answers for *Q1* and *Q3* and of correct answers for *Q2* and *Q4* are given in Fig. 12. For *Q1* there is no TC data as the headset cannot render the entire VE from geometry. Only 9% of EC participants found that there is something wrong with the environment, a broad question that encompasses all aspects of visual quality, while for CC the proportion is 41%. We ran the McNemar's test for paired proportions on the 2 $\times$ 2 contingency table. The table values are: 1 participant answered ”yes” for both CC and EC, 12 participants answered ”no” for both CC and EC, 8 participants answered ”yes” for CC and ”no” for EC, and only 1 participant answered ”no” for CC and ”yes” for EC. The discordant, off-diagonal values of 8 and 1 yield the McNemar's test results of $\chi^2 = 5.44$ and $p = 0.0196$, so the user perception advantage of EC over CC is statistically significant ($\alpha = 0.05$), confirming the experimental hypothesis *H1*.

For TC and EC all participants answered *Q2* correctly, whereas only 15.2% did so for CC. The difference EC-CC is significant (McNemar's test $\chi^2 = 18$ and $p < 0.001$), confirming the experimental hypothesis *H2*. For TC, the near rectangle has straight edges, which are aligned with the straight edges of the far rectangle, which is a strong cue pointing the participant in the correct direction (Fig. 10 TC). For EC, the projection of the near rectangle is curved but it was sufficient to point the participant in the right direction(Fig. 10 EC). For CC, performance is worse than chance level, i.e., 33%, because the visualization consistently points the participant in the wrong direction.

The responses to *Q3* were somewhat surprising. First, not all CC and TC trajectories were judged as straight, which indicates that it is difficult to judge trajectories with a great z change as perspective foreshortening yields a highly non-linear image plane motion. Indirectly, this is positive for our method which bends trajectories over the (r_1, r_2) z range. The second surprising aspect of the answers to *Q3* is that a majority (56%) of EC image trajectories were judged as *straight*. Digging deeper into the answers to *Q3*, two of the three EC trajectories were consistently judged as straight (82% and 82% ”yes”), while one was consistently judged as not straight (4.5% ”yes”). The explanation for this is that the curvature of the trajectory is easily detectable when the VE line of the trajectory passes close to the user viewpoint, who then has the vantage point necessary to see the bend of the trajectory. For example, in Fig. 11, the sphere has an image plane velocity vector whose horizontal component switches sign. A trajectory that stays one side of the view direction might have no velocity component sign changes, and only have acceleration sign changes which are harder to detect. Overall, significantly more CC ($p < 0.001$) and TC ($p < 0.001$)



	Q1. VE wrong? (% yes)	Q2. Points where? (% correct)	Q3. Trajectory straight? (% yes)	Q4. Landed where? (% correct)
🟦 CC	40.9%	15.2%	86.4%	6.1%
🟧 EC	9.1%	100.0%	56.1%	83.3%
⬜ TC		100.0%	81.8%	93.9%

Figure 12: User study results analyzed with McNemar's test for paired proportions. Significant differences are indicated with an asterisk.

trajectories were judged as straight compared to EC, which confirms the experimental hypothesis *H3*, althoughparticipants detected the curvature of EC trajectories in fewer than half of the cases.

The answers to *Q4* indicate an even worse performance for CC than for Task 2 (*Q2*). Whereas there were only three far rectangles for Task 2, there were 7 landing boxes for Task 3, so the chances that in the CC image the sphere lands close enough to the correct box are low. The differences between EC and CC ($p < 0.001$), and between TC and EC ($p < 0.001$) are significant. CC projects the sphere from the user viewpoint, which results in an image location substantially different from the image location of the nearby landing box that is projected from the environment map center. Answer correctness in EC is about 10% lower than for TC. 7 of the $3 \times 22 = 66$ landing locations were judged incorrectly for EC and correctly for TC, and none were judged correctly for EC and not for TC. McNemar's test indicates that the difference between TC and EC is significant ($\chi^2 = 7$, $p = 0.008$). Like TC, EC renders the sphere precisely above the correct landing box, but, unlike TC, EC does not provide accurate depth cues. A participant might see the horizontal and vertical (x, y) image plane alignment but suspect a misalignment in z, hence the incorrect answers for EC. In conclusion, the experimental hypothesis *H4* is confirmed. Although EC answers were 83.3% correct, the lack of depth perception for EC leads to a significantly lower proportion of correct answers compared to TC.

5 Conclusions. Limitations. Future Work

We have presented a method for reducing rendering load to increase the complexity of the virtual environments that can be rendered on thin VR systems. Our method is based on the tried and true approach of rendering the far region from environment mapping, but instead of limiting the far region to a disconnected background image, our method integrates seamlessly the environment mapped part of the frame with the part rendered from geometry. Continuity is guaranteed by construction and dynamic far geometry is composited with the environment mapped part of the frame with pixel-level accuracy. Empirical evidence shows that the visualization continuity provided by our method translates to significant advantages over conventional environment mapping in terms of overall perception of the VE and in terms of answer accuracy to spatial queries.

Our approach is ready to be integrated with VR applications. Our approach was tested in texture-less VEs which best reveal geometric discontinuities, as well as the geometric distortions introduced by our method. We expect that in textured, rich environments, which do not have the long straight lines of the city blocks of our VE, the distortion introduced by the morph is even less noticeable. To facilitate integration into complex VE's, the morph and depth compositing should not be added to all vertex and fragment shaders in the VE, but our method should be integrated into an advanced cube map construct with parameters O, r_1, and r_2 that applies the morph and the depth compositing to all VE geometry.

One limitation of our approach is that, since the dynamic far region is rendered from geometry, the rendering load might not be reduced sufficiently if too much of the far region is dynamic. Similarly, our method might not reduce rendering load sufficiently in a scene with highly variable complexity. Another limitation is that the far region is still not at par with the parts of the VE rendered from geometry. For example, the headlights of a car moving in the far region at night should illuminate the stationary far geometry, which future work could investigate doing by enhancing the environment map with additional channels such as normals and material properties.

Our method works for a "user region" (i.e., a near region) of considerable size, e.g., $r_1 = 30$m in the *Manhattan* VE. It is unlikely that the physical space available to the VR application exceeds the near region, so future work will have to examine the integration of our method with methods for alleviating the virtual/physical world size mismatch, such as teleportation and redirection. An important advantage of our method is that it does not require complex preprocessing. Getting ready for a new user region only requires rendering the cube map and collecting the triangles within r_2 of the region center, and it does not require complex geometric processing such as visibility or LoD computation. For example, since the Quest 2 can render the *Manhattan* VE from geometry at 12fps, rendering a cube map takes 500ms, and even less if a single pass approach is used to distribute the triangles to the six faces with a geometry shader. Getting ready for a new user region in a fraction of a second is sufficient to support fade-out/fade-in relocation in the VE.

Another direction of future work is to integrate our method into a distributed VR system where it promises to reduce the amount that has to be transferred from the server to the client. To reduce startup times, the server could first transfer the geometry of a small near region and then increase it progressively, transferring the newly included triangles to the client. Once additional triangles arrive at the client, the triangles are moved from the environment mapped to the geometry-based part of the frame by gradually increasing r_1 and r_2. Such a progressive refinement could be hidden from the user by taking advantage of saccades and of favorable view directions.

Thin VR clients have made rendering load reduction once again an active area of research. As it was the case for desktop workstation GPUs, the rendering capability of VR headset GPUs is likely to continue to increase. As this implies a commensurate increase in power consumption, we foresee that rendering capability will not increase as fast for untethered VR systems that are powered by batteries. Finally, even if battery capacity increases keep up, limiting rendering to only what is relevant to the application will remain important such as to avoid wasteful power consumption, towards alleviating environmental concerns of growing urgency.

Acknowledgments

The authors wish to thank the anonymous reviewers, the study participants, and the Purdue University Computer Graphics and Visualization Laboratory for their contributions to this paper. This work was supported in part through a gift from Meta.

References

[1] Quest 2. *Meta Platforms, Inc. https://store.facebook.com/quest/.*

[2] Unity real-time development platform. *Unity Software, Inc. https://unity.com/.*

[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

[4] T. Chai and R. R. Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[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

[6] 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

[7] 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.

[8] A. Hore and D. Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th international conference on pattern recognition*, pp. 2366–2369. IEEE, 2010.

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

[10] 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

[11] 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.

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

[13] R. K. Mantiuk, G. Denes, A. Chapiro, A. Kaplanyan, G. Rufo, R. Bachy, T. Lian, and A. Patney. Fovvideovdp: A visible difference predictor for wide field-of-view video. *ACM Transactions on Graphics (TOG)*, 40(4):1–19, 2021.

[14] J. Meng, S. Paul, and Y. C. Hu. *Coterie: Exploiting Frame Similarity to Enable High-Quality Multiplayer VR on Commodity Mobile Devices*, p. 923–937. Association for Computing Machinery, New York, NY, USA, 2020.

[15] Oculus. Performance and optimization. In *Oculus Developer Center, https://developer.oculus.com/documentation/unity/unity-perf/.*

[16] 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

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

[18] 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.

[19] 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

[20] 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

[21] 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.

[22] A. Szczepanek, W. Sas, and J. Bowater. Mcnemar's test calculator. In *https://www.omnicalculator.com/*, Retrieved May 2022.

[23] L. Wang, J. Chen, Q. Ma, and V. Popescu. Disocclusion headlight for selection assistance in vr. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 216–225, 2021. doi: 10.1109/vr50410.2021.00043

[24] L. Wang, W. Wu, Z. Zhou, and V. Popescu. View splicing for effective vr collaboration. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 509–519, 2020. doi: 10.1109/ISMAR50242.2020.00079

[25] M.-L. Wu and V. Popescu. Anchored multiperspective visualization for efficient vr navigation. In *International Conference on Virtual Reality and Augmented Reality*, pp. 240–259. Springer, 2018. doi: 10.1007/978-3-030-01790-3_15

[26] M.-L. Wu and V. Popescu. Efficient vr and ar navigation through multiperspective occlusion management. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3069–3080, 2018. doi: 10.1109/TVCG.2017.2778249

[27] 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.

[28] 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