Adaptive Render-Video Streaming for Virtual Environments

Jia-Jie Lim* Matthias S. Treder* Aaron Chadha Yiannis Andreopoulos

Sony Interactive Entertainment, London, UK
{first dot last}@sony.com

Abstract

In cloud-based gaming and virtual reality (G&VR), scene content is rendered in a cloud server and streamed as low-latency encoded video to the client device. In this context, distributed rendering aims to offload parts of the rendering to the client. An adaptive approach is proposed, which dynamically assigns assets to client-side vs. server-side rendering according to varying rendering time and bitrate targets. This is achieved by streaming perceptually-optimized scene control weights to the client, which are compressed with a composable autoencoder in conjunction with select video segments. This creates an adaptive render-video (REVI) streaming framework, which allows for substantial tradeoffs between client rendering time and the bitrate required to stream visually-lossless video from the server to the client. In order to estimate and control the client rendering time and the required bitrate of each subset of each scene, a random-forest based regressor is proposed, in conjunction with the use of AIMD (additive-increase/multiplicative-decrease) to ensure predetermined average bitrate or render-time targets are met. Experiments are presented based on typical sets of G&VR scenes rendered in Blender and HEVC low-latency encoding. A key result is that, when the client is providing for 50% of the rendering time needed to render the whole scene, up to 60% average bitrate saving is achieved versus streaming the entire scene to the client as video.

1. Introduction

Gaming and Virtual reality (G&VR) frameworks are increasingly-popular forms of entertainment and social activity. At the moment, G&VR typically takes place either via client-side rendering [10], or via server-based rendering, where the game or VR scene is fully-rendered on the server and the client device receives video [28]. The latter approach emerged more than ten years ago in order to cater for thin clients that do not support the appropriate CPU or GPU hardware [28]. However, within the last few years, many commodity laptops, smartphones, and G&VR platforms have emerged with powerful graphics, central and neural processing-unit (GPU, CPU and NPU) hardware and ample storage to retain complex game assets [35]. Consequently, approaches that distribute the rendering effort between client and server are increasingly being explored [6, 12, 21, 22, 26, 27, 30].

Render-video (REVI) based streaming is proposed as a novel distributed rendering solution. REVI is initiated with the offline distribution of gaming or VR assets between server and client. The server streams both video (of assets rendered on the server) and compressed scene control weights (of assets to be rendered on the client) as well as auxiliary information required for shadows and illumination. The client renders the assets corresponding to the control weights and then composes the output frame. At regular time intervals, the server: *(i)* sends a manifest file that describes the current assets in the camera view and their estimated rendering time and bitrate costs; *(ii)* controls (and synchronizes) the state of the entire REVI streaming process, with requests for video streams and scene control weights made by the client based on the provided manifest. This entails the following contributions.

- Assuming that the client device has storage to *a-priori* receive the offline assets of all G&VR scenes of interest, a perceptually-optimized compaction of scene control weights is proposed. To this end, a *composable autoencoder* is introduced: unlike a standard autoencoder, it does not only offer optimized compaction for a given target dimension d, but also for any smaller dimension, similar to principal component analysis (PCA) [34]. Unlike PCA, the composable autoencoder is sensitive to non-linear dependencies and hence offers better compaction. It is used for: *(i)* compaction with no impact on perceptual quality of the rendered frames at the client side and *(ii)* inherent rate/accuracy scalability, i.e., increased quality with increased number of retained dimensions.
- Since local rendering of the entire scene cannot be guar-

*Equal contribution. Listing order of the two authors is random.

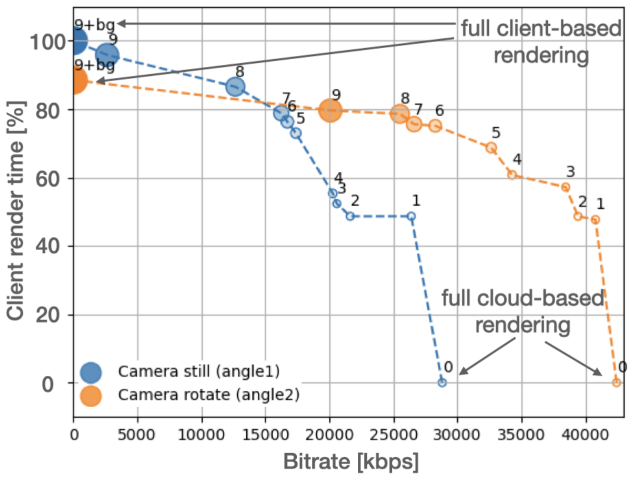

Figure 1. Bitrate vs. normalized client rendering times for an example scene with a total of 9 characters and background. Bitrate and rendering time are an average across all frames of a video clip. The number next to each data point specifies the number of characters rendered on the client, where 9+bg = full client-based rendering and 0 = full server-based rendering. Different camera paths are represented with different colors (blue=still camera, orange=rotating camera).

anteed, it is proposed to carry out offline estimation of the encoding bitrate and the rendering time of all scene assets. This allows for the derivation of rendering time vs. bitrate estimates, which can be used during real-time REVI streaming.

- Based on these estimates, dynamic bitrate-based or render-time based adaptation is proposed based on the use of AIMD (additive-increase/multiplicative decrease). The server can then deliver each component as compact scene control weights, or as video. This allows for significant benefits in bitrate by transferring some of the rendering time to the client device in an optimized manner, while keeping the state synchronization entirely at the server side. Fig. 1 showcases the obtained bitrate-vs.-render time tradeoffs. The plot shows that splitting the rendering effort between client and cloud allows for significant tradeoffs between bitrate and client rendering time (numbers 1 through 9).
- It is shown that the client can adapt its estimates based on adaptive resource estimation, which leads to increased compliance to rendering time or bitrate constraints.
- Experiments based on: *(i)* Blender assets that are representative of G&VR scenes and *(ii)* HEVC [32] low-latency encoding, show how the estimated client rendering time versus bitrate is approximated by the proposed estimates. When allowing for 50% of the rendering GPU time to take place on the client, bitrate savings of up to 60% are achieved.

2. Related Work

Multiserver rendering or server/edge computing: Approaches for multiserver rendering assume there are multiple machines that are capable to render parts of the game and decide which parts to do on which hardware [15, 37]. It is assumed that bitrate is not a constraint and each server can send high-volume visual representations to others, in order to be stitched together for delivery to the client.

Rate-quality optimization in video encoding: At the other extreme, optimization of G&VR encoding assumes that rendering is performed entirely on a remote server and the client only receives a video stream of the rendered scene. The client is only tasked with decoding the video stream and transmitting player input commands back to the server. Recent work on game encoding optimization typically aims to modify the video encoding process only rather than optimizing rendering [1, 11, 13, 20]. These approaches promote handcrafted or learnable region-of-interest (ROI) based encoding. More recently, Le *et al.* propose GameCodec [17], the first end-to-end neural video codec designed for cloud gaming, which leverages camera pose and depth information in order to generate compressed frames. Such approaches require substantially-higher compute cost than HEVC [32], which is a high-performance video coding standard used for cloud gaming.

Render-vs-quality and bitrate/quality optimization: In all approaches of this category, the device under consideration is either carrying out rendering, or receives encoded video, and the essence of all proposals is to introduce optimizations within each of the two cases for rendering time or bitrate/quality. For example, tradeoffs between rendering detail and rendering time have been used in games for decades. Recent approaches explore speculative rendering or ROI/field of view rendering [29] in order to reduce rendering effort for areas not expected to be attended to by the viewer. On the other hand, traditional adaptive bitrate (ABR) ladders and bitrate optimizations in encoders [4, 33] have been developed significantly since their original introduction. Overall, none of the approaches here attempt render-video streaming or attempt adaptive bitrate optimization under render time constraints as considered by this paper.

Distributed rendering: Stengel *et al.* [30] compute global illumination on the server and serve light probes dynamically to the client, which then composes the final frame. The authors encode the environment maps as textures, but neural representations have been explored, too [27]. Shading Atlas Streaming performs shading on the server and then streams a texture atlas with baked in lighting as a video along with geometry information [21]. Quadstream streams quadrilateral proxies that allow for efficient rerendering for nearby viewpoints, with applications in VR [12]. Adaptive residual streaming runs a low-res version

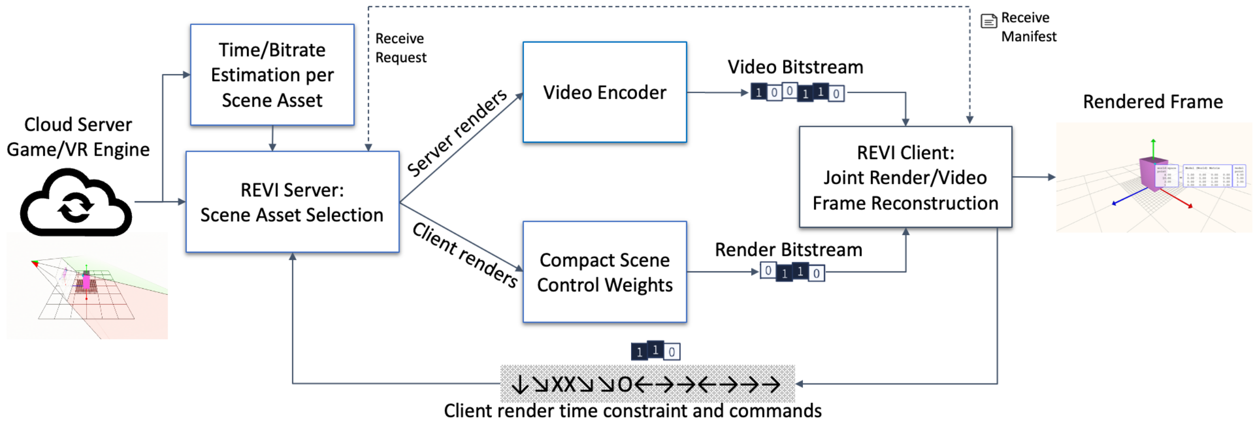
Time/Bitrate Estimation per Scene Asset
Receive Request
Receive Manifest
Cloud Server Game/VR Engine
REVI Server: Scene Asset Selection
Server renders
Client renders
Video Encoder
Video Bitstream
Compact Scene Control Weights
Render Bitstream
REVI Client: Joint Render/Video Frame Reconstruction
Rendered Frame
Client render time constraint and commands

Figure 2. Schematic of the proposed REVI streaming framework.

of a game on the client and a high-res version on the server [6]. The difference between low-res and high-res versions is streamed to the client. The main conceptual difference between these approaches and the proposed approach is that the former partition the rendering pipeline globally, distributing some tasks to the client (e.g., direct lighting) and some to the server (e.g., global illumination). In contrast, in the proposed approach the assets themselves are distributed between client and server, such that each side renders a subset of assets. This allows for a dynamic allocation of rendering time on server and client in real-time based on rendering time and bitrate targets.

3. System Model

The overall schematic of the REVI framework is shown in Fig. 2. The server runs the Game/VR Engine including both the game logic and a renderer. It also includes the REVI server that provides a manifest describing options for assets to be included in the video stream or to be sent in the form of control weights. The client also runs a renderer, but it does not run the game logic. Instead, it is remote-controlled by the server via the streamed control weights. The client can subscribe to the server manifest and, per game scene, request different configurations for the game assets.

Direct and indirect illumination is realized within Blender's real-time rendering engine EEVEE [9]. For accurate shadow casting between assets, topologies (e.g., meshes) are loaded into GPU memory on both server and client sides, but Physically Based Rendering (PBR) textures [24] are only loaded on the side that actually renders the asset. For global illumination effects, light probes are synced between server and client. This avoids the high computational requirements of full path tracing [16, 23], which are unsuitable for many VR and consumer-level client devices. Initial results are shown in the Supplemental Materials.

A widely used representation for 3D assets in computer games and animations is a mesh with corresponding texture maps. A mesh is formalized as a set of v vertices $\mathcal{V} = \{(x_i, y_i, z_i) \in \mathbb{R}^3 \mid i = 1, ..., v\}$ and a corresponding topology defining the edges connecting the vertices. The background is considered as an asset with the largest distance (e.g., a large textured box encompassing the visible area). In addition to meshes, particle-based systems [25] are used for modelling transient (e.g., explosion) or dynamically changing assets (e.g., fire, water). Although the formalism developed in the next sections focuses on mesh-based assets, these alternate assets are typically also controlled by a set of parameters (e.g., position of each particle). Hence, the utilized formalism can be extended to these cases.

4. Compact Scene Control Weights

Scene control weights involve per-frame parameters such as character motions and positions of dynamic light sources. Two techniques used specifically for animating meshes are blendshapes and rigging. Blendshapes are frequently employed in facial animation [18]. A target expression $\mathbf{f} \in \mathbb{R}^{3v}$ with v degrees of freedom in each of the three dimensions can be formulated as a linear combination of blendshapes. An alternative to blendshapes that links the animation to physical constraints is rigging. It involves the creation of a virtual skeleton wherein the model's geometry is bound to a hierarchical structure of bones and joints [3]. Bones are related to mesh vertices via another set of weights that describes the influence each bone has on each vertex. For notational simplicity, the blendshape or rig control weights are collected in a vector $\mathbf{w} \in \mathbb{R}^n$, where $n = 3v$ in the case of blendshapes and $n < v$ in a control rig [3].

To animate an asset on the client device, the blendshapes or control rig needs to be transferred only once, but $\mathbf{w}$ needs to be streamed for every frame. This incurs an ongoing bitrate cost, which must be minimized for efficient delivery. Autoencoders [2] achieve dimensionality reduction by mapping vectors to a lower dimension $d < n$. To enable *real-time adaptive streaming*, dynamic control of the value of d is required. However, autoencoders are always trained for a single target dimensionality and need to be re-trained when d changes. Principal Components Analysis (PCA) [34] provides a single decomposition for all values of d but is limited to linear relationships.

The non-linear compression of autoencoders is combined with the composability of PCA by defining a *composable autoencoder*. Let $\mathbf{m}_d \in \mathbb{R}^{d_{\max}}$ be a binary mask of the form $\mathbf{m}_d = [1, ..., 1, 0, ..., 0]$ where $d_{\max}$ is the size of the bottleneck layer and the subscript indicates the index of the last "1" (e.g., $\mathbf{m}_2 = [1, 1, 0, 0, ...]$). The encoder is a function of two arguments, $f_{\text{enc}}(\mathbf{w}, \mathbf{m}_d) = \mathbf{y} * \mathbf{m}_d = \mathbf{y}_{\text{enc}}$, where $\mathbf{y}$ is the full activation in the bottleneck layer, $\mathbf{y}_{\text{enc}}$ is the masked activation, and $*$ represents point-wise multiplication. Crucially, the effective dimension of $\mathbf{y}_{\text{enc}}$ is $d < d_{\max}$, providing the desired compression. The decoder $f_{\text{dec}}(\mathbf{y}_{\text{enc}}) = \hat{\mathbf{w}}$ performs zero padding and then outputs the reconstructed input. Encoder and decoder are implemented as multi-layer perceptrons with a single hidden layer with ReLU activations and trained using L2 loss. During inference, the desired value for d can be set on the fly. During training, to make sure that the model learns a composable function, d is randomly selected in each training iteration. This forces the model to concentrate as much information as possible in the first dimensions. For more details see the Supplemental Materials.

Fig. 3a shows the MSE on test data as a function of the number of components, for a control rig dataset comprising $n = 455$ parameters. The standard autoencoder performs best for $d = 80$ (the dimensionality it has been trained on) but its performance collapses quickly for all $d < 80$. PCA outperforms the autoencoder for $d < 80$ with the sharpest slope for the first 10 components. The composable autoencoder shares the composability with PCA but yields better performance due to sensitivity to non-linear dependencies. Therefore, by setting its maximum dimensionality high enough, it can be used for the dynamic adjustment of the autoencoder-based compaction with accuracy that significantly surpasses that of PCA. Using this approach, perceptually-optimized blendshape/rig control weights can now be derived by solving the constrained rate optimization problem

$$\begin{aligned} &\text{minimize } d \\ &\text{subject to VMAF(render}(\mathbf{f})) \geq q \end{aligned} \tag{1}$$

where: minimizing d corresponds to minimizing the bitrate, VMAF is a perceptually-tuned quality score [19] for the rendered asset $\mathbf{f}$, and q is a perceptual quality threshold (typically $q = 95$) in order to allow for visually-lossless representation according to VMAF [19]. This optimization is performed on a separate dataset to avoid overfitting. To solve this optimization problem, a logarithmic search for d is proposed. Fig. 3b shows the rate-perceptual quality tradeoff for an example asset.

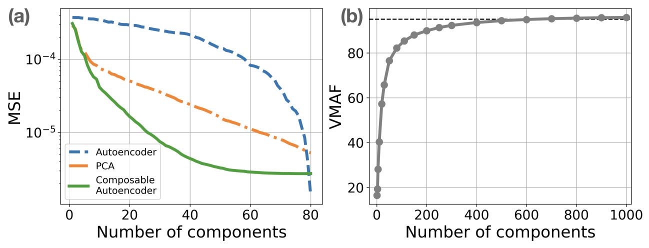

Figure 3. (a) Composable autoencoder results for a control rig for an example asset of a humanoid character with 455 dimensions and $d_{\max} = 80$. 19 different motion sequences are used for training and 2 held out sequences for testing (lower MSE is better). (b) Perceptual optimization. Number of components is plotted against VMAF for an exemplary asset from the Multiface dataset [38] with n=21918 vertex coordinates (higher VMAF is better). The dashed line indicates the VMAF=95 threshold (attained for about d=600), which corresponds to a compression factor of 36x.

5. REVI Server: Scene Asset Selection

The linchpin of an efficient render-video streaming system is an accurate control of transmission bitrate and rendering time. Two scenarios are considered: one whereby the client rendering time is bounded and bitrate is minimized, and, vice versa, the scenario whereby bitrate is limited and client rendering time is minimized. In both scenarios, the following two options exist for each asset:

- Render the asset on the server and stream it as a video (along with other rendered assets).
- Stream the asset's encoded control weights and render it on the client device.

To formalize this problem, let $\mathcal{A}$ be the space of all assets and $\mathcal{A}_{\text{view},t} \subset \mathcal{A}$ the assets contained in the camera view at time t. The space is assumed to include information about the asset, its location, rotation, as well as camera parameters. Since these features can change over time, each asset is time dependent and will be denoted as $a_t \in \mathcal{A}$. The goal is to partition $\mathcal{A}_{\text{view}}$ into assets rendered on the server $\mathcal{A}_{\text{server}}$ and assets rendered on the client $\mathcal{A}_{\text{client}}$. Let $b_{\text{vid}} : \mathcal{A} \to \mathbb{R}$ return the estimated future bitrate for a given asset if it were rendered alone on a neutral background and streamed as a video. More concretely, $b_{\text{vid}}(a_t) = \text{bitrate}(a_{t+1}, a_{t+2}, ..., a_{t+N} \mid a_t, a_{t-1}, ..., a_{t-M+1})$, that is, the future bitrate for asset a for the next N frames when conditioned on the past M frames. Analogously, $b_{\text{enc}} : \mathcal{A} \to \mathbb{R}$ returns the bitrate for an asset if its encoded control weights are streamed to the client. Let $c : \mathcal{A} \to \mathbb{R}$ be the function that returns the rendering time it takes to render the asset on the client device after receiving the control weights. The main objective function can now be formalized as

$$\begin{aligned} &\text{minimize} \;\; b_{\text{vid}}(\mathcal{A}_{\text{server},t}) + b_{\text{enc}}(\mathcal{A}_{\text{client},t}) \\ &\text{subject to} \;\; c(\mathcal{A}_{\text{client},t}) \leq \Theta_t \end{aligned} \tag{2}$$

where $b_{\text{vid}}(\mathcal{A}_{\text{server},t})$ is shorthand notation for $\sum_{a_t \in \mathcal{A}_{\text{server},t}} b_{\text{vid}}(a_t)$, and analogously for b_{enc} and c. In other words, the aim is to minimize the total bitrate consisting of the video bitrate incurred by assets rendered on the server plus the encoded bitrate for streaming assets to the client device, subject to the constraint that the client has a limited and potentially time-varying computation budget amounting to Θ_t.

The constrained minimization of (2) involves a discrete combinatorial optimization problem. It consists of partitioning the set $\mathcal{A}_{\text{view}}$ into two subsets $\mathcal{A}_{\text{server}}$ and $\mathcal{A}_{\text{client}}$. The optimization search space is finite, but with $2^{|\mathcal{A}_{\text{view}}|}$ possible partitions it grows exponentially with the number of assets. However, it is significantly reduced by taking into account a practical necessity: To avoid occlusion effects, it must be guaranteed that asset rendering must take place from furthest to closest to the view camera, otherwise occlusion effects would need to be considered[1]. This ordering reduces the search space to $|\mathcal{A}_{\text{view}}|$. An example is depicted in Fig. 4a.

Algorithm 1 presents a water-filling algorithm that solves the optimization problem in (2). Assets are sorted in ascending order according to their distance to the camera. If the expected rendering time of the first asset does not exceed threshold θ_t, it is added to $\mathcal{A}_{\text{client}}$ and removed from $\mathcal{A}_{\text{server}}$. The process proceeds in the same fashion with the next closest asset until total client rendering time exceeds the threshold. The algorithm requires that total bitrate decreases monotonically as assets get removed from $\mathcal{A}_{\text{server}}$, i.e., $b_{\text{enc}}(a) < b_{\text{vid}}(a)$. If the requirement is met, the optimality of the solution is guaranteed. Violations of the optimality requirements are explored in the Supplemental Materials.

Evaluation of b_{vid} and c requires estimation of the future bitrates and render times. As a consequence of using estimators, bitrate and rendering time estimates will not be exact. Consistent underestimations of c can prohibit the client device from finishing all computations in time, leading to jitter or frame drops. To address this challenge, a relaxation of the constraint in (2) is proposed. An additional series of adaptive thresholds θ_t is introduced in order to deal with systematic biases in the estimators. For instance, if c systematically underestimates the rendering time, this is compensated by enforcing $\theta_t < \Theta_t$. The optimization problem is therefore modified to:

[1]Note that the background assets can also be rendered on the client (again from furthest to closest to the view camera) as long as the server streams a segmentation mask for the foreground assets along with their rendered video.

Algorithm 1 REVI water-filling algorithm

```
Require: $\forall a \in \mathcal{A}_{\text{view}} : b_{\text{enc}}(a) < b_{\text{vid}}(a)$
  $\mathcal{A}_{\text{server}} \leftarrow \mathcal{A}_{\text{view}}$
  $\mathcal{A}_{\text{client}} \leftarrow \emptyset$
  total_compute $\leftarrow 0$
  for $a \in$ sorted($\mathcal{A}_{\text{view}}$) do                    $\triangleright$ sort by distance
      if total_compute + $c(a) \leq \theta_t$ then
          total_compute $\leftarrow$ total_compute + $c(a)$
          $\mathcal{A}_{\text{server}} \leftarrow \mathcal{A}_{\text{server}} \setminus \{a\}$
          $\mathcal{A}_{\text{client}} \leftarrow \mathcal{A}_{\text{client}} \cup \{a\}$
      else
          break
      end if
  end for
```

$$\begin{aligned} &\text{minimize} \;\; b_{\text{vid}}(\mathcal{A}_{\text{server},t}) + b_{\text{enc}}(\mathcal{A}_{\text{client},t}) \\ &\text{subject to} \;\; c(\mathcal{A}_{\text{client},t}) \leq \theta_t \\ &\qquad\qquad \mathbb{E}_{t-1,t-2,\ldots}[\, c(\mathcal{A}_{\text{client},t}) \,] \leq \Theta_t \end{aligned} \tag{3}$$

where it is assumed that exact rendering times for past frames are available and the expectation is taken over the past frames. Exact rendering times can be obtained with a simple timer function on the client side. $\mathbb{E}_{t-1,t-2,\ldots}[\, c(\mathcal{A}_{\text{client},t}) \,] \leq \Theta_t$ assures that the rendering time target is met in expectation, whereas using θ_t for the optimization deals with estimator bias. If the estimator bias is unknown, then the initialization is simply $\theta_0 = \Theta_0$. The bitrate minimization is enforcing a tight approximation of the rendering time bound Θ_t, since a decrease in bitrate will involve an increase in rendering time at the client side (i.e., more assets are rendered on the client).

6. Adaptive REVI streaming

A client-server REVI streaming system is now examined. It is assumed that G&VR assets, light probes etc. are delivered offline, e.g., shipped together with the client software or in the form of an offline software update, or streamed asynchronously. During online operation, Algorithm 1 runs on the client device along with a scheduling algorithm for θ_t, whereas instantiations of the estimators b_{vid} and c run on the server.

In terms of estimators for b_{vid} and c, random forests (RF) [5] and support vector machines (SVMs) with a radial basis function kernel [8] were used. Initial testing revealed that RF provided for better performance than SVMs, and was therefore selected for further investigation. The asset's: bounding box, location, rotation, blendshape or rigging weights, as well as: camera location and rotation for

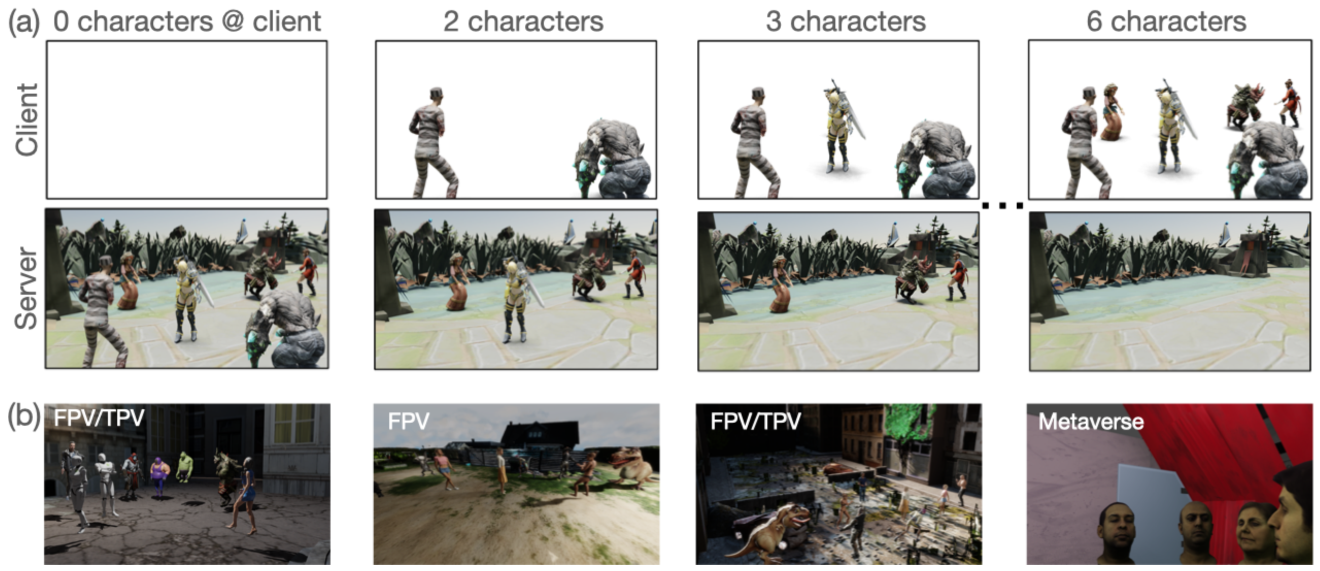

Figure 4. (a) Example illustration of the breakdown of assets between server and client using one of the scenes for the Client-Foreground setting. First row shows the assets rendered on the client, second row the complementary assets rendered on the server. Each column corresponds to a different choice of the number of characters rendered on the client. (b) Example views of the other scenes used in the experiments. FPV: first-person view, TPV: third-person view.

the past 5 frames of several training scenes, were used as features. The target values to be inferred are the mean bitrate (kbps) and rendering time (seconds) for the next 120 frames. The training scenes comprised the asset in front of a neutral background at various camera locations, distances, paths, and multiple asset animations. The animations are exported as videos and encoded using low-latency HEVC [32] encoders in order to estimate bitrate (see experiments for details on encoding recipes).

To derive θ_t, the additive-increase / multiplicative-decrease (AIMD) algorithm was used. AIMD is a feedback control mechanism utilized by diverse biological systems [31] and has achieved widespread usage in feedback control of networked systems, e.g., TCP congestion control [7] and resource adaptation in distributed computing [36]. It is adapted for the control of θ_t as follows:

$$\theta_t = \begin{cases} \theta_{t-1} + \alpha & \text{if } \mathbb{E}[\, c(\mathcal{A}_{\text{client},t})\,] < \Theta_t \\ \theta_{t-1} \cdot \beta & \text{if } \mathbb{E}[\, c(\mathcal{A}_{\text{client},t})\,] \geq \Theta_t \end{cases} \tag{4}$$

with the initialization $\theta_0 = \Theta_0$. The hyperparameters $\alpha > 0$ and $0 < \beta < 1$ determine the magnitude of the additive increase and multiplicative decrease, respectively. The expectation is estimated over a window of past frames $t-1, t-2$, ... using a moving average.

The current values for b_{vid} and c are provided by the server. In terms of practical implementation, taking inspiration from client-driven adaptation in MPEG-DASH [14], a manifest file is periodically sent from the server to the client, with the period of manifest refresh depending on the use case, e.g., per change of G&VR scene or every few seconds, as detailed in the Supplemental Materials. The client makes a request to the server, specifying which assets should be sent as encoded weights and which should be rendered in the cloud. These client requests are assumed to be sent once for a given interval, e.g., once every 120 frames. At the beginning of each interval, the client cycles through the following REVI streaming steps:

1. Receive manifest file from server.
2. Update θ_t using AIMD in (4).
3. Run Algorithm 1 to determine $\mathcal{A}_{\text{server}}$ and $\mathcal{A}_{\text{client}}$.
4. Send request to server to render assets in $\mathcal{A}_{\text{server}}$ and to send assets in $\mathcal{A}_{\text{client}}$ as encoded control weights.
5. Receive video frames for in $\mathcal{A}_{\text{server}}$ and render assets in $\mathcal{A}_{\text{client}}$ locally.
6. Combine video frames with locally rendered assets and display them on the screen.

Note that the scene control weights include updates for other dynamic elements in the scene such as the position of a moving light source. The runtime complexity is estimated as follows. The update of θ_t involves a single addition or multiplication. For Algorithm 1, addition and set insertion / removal operations cost $\mathcal{O}(1)$. The for loop is executed at most $A = |\mathcal{A}_{\text{view}}|$ times leading to $\mathcal{O}(A)$. The evaluation of c via a random forest with T trees and depth D and yields $\mathcal{O}(ATD)$. The dominating factor is this $\mathcal{O}(ATD)$. Since T and D are fixed hyperparameters, the runtime complexity increases linearly with the number of assets and is negligible in comparison to encoding/decoding and rendering of the assets.

7. Experiments

To test the proposed REVI framework, several G&VR scenes were created, consisting of a background asset and multiple characters created by 3D artists, or head avatars created from 3D scans of actual people. Characters were animated using control rigs, whereas head avatars were animated using blendshapes. The following settings were explored:

- *Third-person view (TPV)*: This setting emulates a third-person view of a scene. 3 different backgrounds were paired with 18 different characters. The following camera settings were used: still camera, rotating camera with at least 90 degrees in-plane rotation, and rotation with zoom. Assets were downloaded from Mixamo[2] and Actorcore[3]. An example rendering can be seen in Fig. 1.
- *First-person view (FPV)*: This scenario was created from the third-person views and an additional scene by placing the camera on the head of a character. The camera thus followed the head movements of a character.
- *Metaverse/VR*: This scenario emulates a conversational setting in virtual room and communication via character avatars. Two different sets of characters were used: full-body talking avatars based on the Mixamo and Actorcore assets, as well as five realistic head avatars from the Multiface dataset [38].

Fig. 4b depicts example renderings of the different scenes (a full video is provided as Supplemental Material). Scenes were animated and exported as 5–10 seconds clips at 60 fps. In total, 26 different clips were created across the three scenarios. The Python API of Blender 3.6 was used for arranging the scenes, invoking the rendering, and timing the rendering time per scene asset. Specifically, a render function that involves moving cameras, animating actors and rendering the scenario of interest (1 character with/without background, 2 characters, etc.) was written to facilitate the experiments; the rendering times were measured using the Python built-in timing module. All experiments were executed on Ubuntu 20.04 with an Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz and an NVIDIA RTX A6000 GPU. All video encodings were carried out with HEVC [32] using a low-latency oriented recipe. Encoded weights were obtained from training the composable autoencoder on a separate training set and then projecting the test data onto the PCs. Weights were perceptually optimized to yield visually-lossless reconstruction (VMAF>95). For each individual asset including the backgrounds, random forest regressors were trained on data not used in the streaming experiments in order to predict bitrate and rendering time.

[2]www.mixamo.com

[3]actorcore.reallusion.com

7.1. Experiment 1: bitrate savings for different rendering time targets

This experiment investigated bitrate savings that can be obtained with REVI as compared to a full-cloud rendering baseline. Two versions of REVI and a number of baseline approaches were used, wherein the number of characters was fixed:

- **REVI w/RF**. REVI based on rendering time predictions from RFs.
- **REVI w/o RF**: REVI without rendering time predictions. Instead, the model simply uses the rendering time statistics for previous time segments.
- **2 char, 4 char, 6 char, ...**: Given no other known distributed rendering approaches that allow for an adaptive tradeoff between bitrate and client compute resources, non-adaptive baselines are created, wherein a fixed number of assets is streamed. For instance, in *2 char*, the client always renders the closest two characters and the remaining assets are rendered in the cloud.

Splitting the assets for rendering can involve the client rendering the foreground assets and the server rendering the background assets, or vice versa. Both scenarios were considered:

- **Client-Foreground**: clients renders foreground characters, server renders background plus background characters.
- **Client-Background**: reversed roles whereby the server renders the foreground characters and streams them along with a segmentation mask (which is encoded with the same encoder and recipe). The client renders the background plus background characters and uses the decoded segmentation mask to merge its results with the foreground-character video sent by the server.

Bitrate was minimized subject to a rendering time constraint as shown in (3). The experiments were run for constant client rendering time targets, i.e., $\forall t : \Theta_t = \Theta_{\text{target}}$, and repeated for different values of Θ_{target} between 5% and 85%. Client rendering time was quantified relative to the server side. That is, it was given as fraction of the rendering time the server needs to render the full scene (assuming that client and server have identical compute capabilities). For each of the three settings (TPV/FPV/Metaverse), 100 clips were randomly sampled. Bitrate and total rendering time were averaged across all clips for each target. AIMD was run with $\alpha = 0.1$ and $\beta = 0.9$.

7.2. Experiment 2: time-varying rendering time targets

A key feature of the proposed approach is real-time adaptivity to current rendering time and bitrate constraints. This

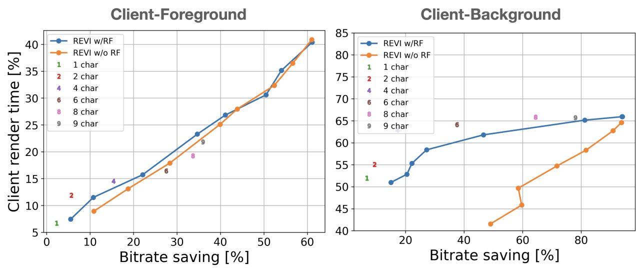

Figure 5. Bitrate savings in Experiment 1 for third-person view. Points closest to the bottom right are optimal. The y-axis shows the relative client rendering time, averaged across all clips. The x-axis shows the bitrate saving obtained with the corresponding method when compared to the full cloud rendering solution. Each data point represents a different rendering time target. Left: Client-Foreground. Right: Client-Background.

experiment investigated whether the proposed approach is able to track moving rendering time targets. To this end, the experiments started from the third-person view data in the Client-Foreground scenario with the REVI w/o RF model. As target values for Θ_t time-varying functional forms were considered, namely a step function, a triangular function, a sine, and a sawtooth function. To track the rendering time average $\mathbb{E}[\, c(\mathcal{A}_{\text{client},t})\,]$ a Kalman filter was used.

8. Results

Fig. 5 shows the results of Experiment 1 for third-person view. As expected, bitrate saving increases with client rendering time, since more assets are generated locally. For the Client-Foreground setting (left panel) REVI without RF slightly outperforms the predictive one, i.e., REVI with RF. This is attributed to RF prediction performance not transferring well enough to the test set and is further discussed in the Supplemental Materials. For REVI w/o RF, up to 60% bitrate savings are obtained with up to 40% of client rendering time usage. For the non-adaptive case, REVI outperforms streaming 2 or 4 characters, although streaming a fixed number of 6, 8, or 9 characters yields slightly better bitrate savings than REVI. Importantly, REVI allows for significantly higher bitrate savings when more client rendering time is available.

The right panel of Fig. 5 depicts the bitrate saving for the Client-Background scenario. Again, REVI w/o RF outperforms REVI w/RF. Crucially, it also outperforms all non-adaptive approaches with a fixed number of characters. In this case, 60% bitrate saving is achieved at 50% client render time, and bitrate saving can reach up to 90% if additional client render time is available. The increased savings mainly stem from the rendering of the background on the client without having to stream it from the server. Further experiments on the other scene types (FPV and VR) are included in the Supplemental Materials.

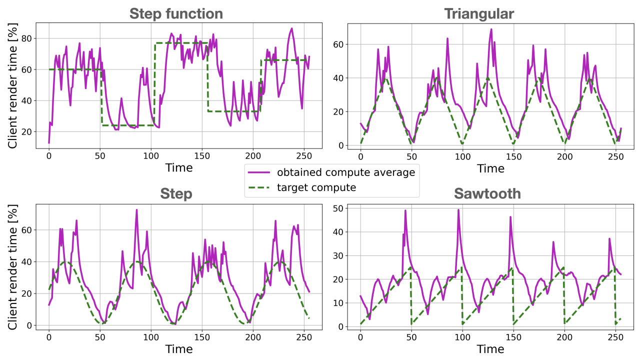

Figure 6. Time-varying rendering time targets Θ_t (green dashed line) in Experiment 2. The actual rendering times obtained with the proposed model (purple solid line) are able to track the rendering time targets albeit with spikes and oscillations. The spikes are partially due to the fact that the Θ_t is continuous whereas the optimization problem is discrete, so the proposed algorithm alternates between overshooting and undershooting in order to meet the rendering time target on average.

Fig. 6 shows the results of Experiment 2. The AIMD algorithm successfully tracks the varying rendering time budget over time. The tracking is found to not be entirely smooth, but rather involves spikes and oscillations. This is due to the fact that the optimization space is discrete whereas the rendering time targets are continuous so the algorithm tends to oscillate around the target value.

9. Conclusion

This paper introduces a render-video streaming approach that adaptively selects which assets are rendered on the server and which are streamed as control weights and rendered locally in G&VR environments. This enables a dynamic and adaptive tradeoff between bitrate and client rendering time at a level of granularity that was not previously possible. This tradeoff is envisioned to be performed at regular intervals (e.g., every 120 frames), taking the past frames as a basis for prediction. The REVI proposal covers how occlusions of assets and lighting effects can be handled for typical cases of resource-efficient rendering in gaming and VR environments. Further work can cover cases where applications have hard constraints on rendering time and bitrate (instead of stochastic ones), occlusion-aware resource prediction, and more advanced cases of global illumination and ray tracing. Integration of the proposed approach within the Unreal engine will also enhance the possible usage of the framework in G&VR. Finally, there is a tradeoff in the REVI framework between efficiency and generality of the framework. The present work targets efficiency, but further can attempt to generalize it for any G&VR pipeline at the cost of some efficiency loss.

References

[1] Hamed Ahmadi, Saman Zad Tootaghaj, Mahmoud Reza Hashemi, and Shervin Shirmohammadi. A game attention model for efficient bit rate allocation in cloud gaming. *Multimedia Systems*, 20:485–501, 2014. 2

[2] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 37–49, Bellevue, Washington, USA, 2012. PMLR. 3

[3] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3D characters. *ACM Transactions on Graphics (TOG)*, 26(3):72–es, 2007. 3

[4] Nabajeet Barman and Maria G Martini. H. 264/mpeg-avc, h. 265/mpeg-hevc and vp9 codec comparison for live gaming video streaming. In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2017. 2

[5] Leo Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001. 5

[6] De-Yu Chen and Magda El-Zarki. A framework for adaptive residual streaming for single-player cloud gaming. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2s), 2019. 1, 3

[7] Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(1): 1–14, 1989. 6

[8] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 5

[9] Blender Docs. EEVEE, 2024. Accessed: 2024-03-20. 3

[10] Jason Gregory. *Game engine architecture*. A K Peters/CRC Press, 2018. 1

[11] Mohamed Hegazy, Khaled Diab, Mehdi Saeedi, Boris Ivanovic, Ihab Amer, Yang Liu, Gabor Sines, and Mohamed Hefeeda. Content-aware video encoding for cloud gaming. In *Proceedings of the 10th ACM multimedia systems conference*, pages 60–73, 2019. 2

[12] Jozef Hladky, Michael Stengel, Nicholas Vining, Bernhard Kerbl, Hans-Peter Seidel, and Markus Steinberger. Quadstream: A quad-based scene streaming architecture for novel viewpoint reconstruction. *ACM Transactions on Graphics*, 41(6), 2022. 1, 2

[13] Gazi Karam Illahi, Thomas Van Gemert, Matti Siekkinen, Enrico Masala, Antti Oulasvirta, and Antti Ylä-Jääski. Cloud gaming with foveated video encoding. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 16(1):1–24, 2020. 2

[14] ISO/IEC 23009-1. Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. Standard, International Organization for Standardization, Geneva, CH, 2014. 6

[15] Iryanto Jaya, Wentong Cai, and Yusen Li. Rendering server allocation for mmorpg players in cloud gaming. In *Proceedings of the 49th International Conference on Parallel Processing*, pages 1–11, 2020. 2

[16] J. T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986. 3

[17] Hoang Le, Reza Pourreza, Amir Said, Guillaume Sautiere, and Auke Wiggers. Gamecodec: Neural cloud gaming video codec. In *33rd British Machine Vision Conference 2022, BMVC 2022, London, UK, November 21-24, 2022*. BMVA Press, 2022. 2

[18] J. P. Lewis, Ken Anjyo, Taehyun Rhee, Mengjie Zhang, Fred Pighin, and Zhigang Deng. Practice and Theory of Blendshape Facial Models. In *Eurographics 2014 - State of the Art Reports*, pages 199–218. The Eurographics Association, 2014. 3

[19] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward A Practical Perceptual Video Quality Metric. In *Netflix Technology Blog*, page Jun 6, 2016. 4

[20] Omar Mossad, Khaled Diab, Ihab Amer, and Mohamed Hefeeda. Deepgame: Efficient video encoding for cloud gaming. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 1387–1395, 2021. 2

[21] Joerg H. Mueller, Philip Voglreiter, Mark Dokter, Thomas Neff, Mina Makar, Markus Steinberger, and Dieter Schmalstieg. Shading atlas streaming. *ACM Transactions on Graphics*, 37(6), 2018. 1, 2

[22] Thomas Neff, Brian Budge, Zhao Dong, Dieter Schmalstieg, and Markus Steinberger. PSAO: Point-Based Split Rendering for Ambient Occlusion. In *High-Performance Graphics - Symposium Papers*. The Eurographics Association, 2023. 1

[23] Y Ouyang, S Liu, M Kettunen, M Pharr, and J Pantaleoni. Path resampling for real-time path tracing. *Computer Graphics Forum*, 40:17–29, 2021. 3

[24] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (4th ed.)*. The MIT Press, 2023. 3

[25] William Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *ACM Transactions on Graphics*, page 91–108, 1983. 3

[26] Bernhard Reinert, Johannes Kopf, Tobias Ritschel, Eduardo Cuervo, David Chu, and Hans-Peter Seidel. Proxy-guided image-based rendering for mobile devices. *Computer Graphics Forum*, 35(7):353–362, 2016. 1

[27] Carlos Rodriguez-Pardo, Javier Fabre, Elena Garces, and Jorge Lopez-Moreno. Nenv: Neural environment maps for global illumination. *Computer Graphics Forum*, 42(4): e14883, 2023. 1, 2

[28] Ryan Shea, Jiangchuan Liu, Edith C-H Ngai, and Yong Cui. Cloud gaming: architecture and performance. *IEEE Network*, 27(4):16–21, 2013. 1

[29] Rahul Singh, Muhammad Huzaifa, Jeffrey Liu, Anjul Patney, Hashim Sharif, Yifan Zhao, and Sarita Adve. Power, performance, and image quality tradeoffs in foveated rendering. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 205–214. IEEE, 2023. 2

[30] Michael Stengel, Zander Majercik, Benjamin Boudaoud, and Morgan McGuire. A distributed, decoupled system for losslessly streaming dynamic light probes to thin clients. In *Proceedings of the 12th ACM Multimedia Systems Conference*,

pages 159–172, New York, NY, USA, 2021. Association for Computing Machinery. 1, 2

[31] Jonathan Y. Suen and Saket Navlakha. A feedback control principle common to several biological and engineered systems. *Journal of the Royal Society Interface*, 19(188): 20210711, 2022. 6

[32] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. 2, 6, 7

[33] Liyang Sun, Tongyu Zong, Siquan Wang, Yong Liu, and Yao Wang. Tightrope walking in low-latency live streaming: Optimal joint adaptation of video rate and playback speed. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 200–213, 2021. 2

[34] Jolliffe Ian T. and Cadima Jorge. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A*, 374:20150202, 2016. 1, 4

[35] Tianxiang Tan and Guohong Cao. Efficient execution of deep neural networks on mobile devices with npu. In *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*, pages 283–298, 2021. 1

[36] L. Vlahakis, N. Athanasopoulos, and S. McLoone. Aimd scheduling and resource allocation in distributed computing systems. In *60th IEEE Conference on Decision and Control (CDC)*, pages 4642–4647, 2021. 6

[37] Maoli Wang, Kunlun Yang, Yining Zhao, Yalin Wang, Jinan Guo, and Xu Ma. Multi-server offloading based on game theory in cloud-edge collaborative computing. In *Proceedings of the 2021 ACM International Conference on Intelligent Computing and its Emerging Applications*, pages 128–133, 2021. 2

[38] Cheng-hsin Wuu, Ningyuan Zheng, Scott Ardisson, Rohan Bali, Danielle Belko, Eric Brockmeyer, Lucas Evans, Timothy Godisart, Hyowon Ha, Alexander Hypes, Taylor Koska, Steven Krenn, Stephen Lombardi, Xiaomin Luo, Kevyn McPhail, Laura Millerschoen, Michal Perdoch, Mark Pitts, Alexander Richard, Jason Saragih, Junko Saragih, Takaaki Shiratori, Tomas Simon, Matt Stewart, Autumn Trimble, Xinshuo Weng, David Whitewolf, Chenglei Wu, Shoou-I Yu, and Yaser Sheikh. Multiface: A Dataset for Neural Face Rendering. *arXiv*, page 2207.11243, 2022. 4, 7