Supplemental Materials

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

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

1. Composable autoencoder

The encoder f_{enc} was a multi-layer perceptron (MLP) with 1 hidden layer. The hidden layer's dimension was twice the input dimension. The bottleneck layer dimension d was adapted to the class of assets. For instance, for control rig parameters we used a maximum of 80 components whereas a larger number of components was required for blendshapes defined on the vertices. The decoder f_{dec} had the same architecture in reverse. All activation functions were Rectified Linear Units (ReLU). Masks were randomly selected from a uniform distribution. An Adam optimizer was used with a learning rate of 0.0001. The model was trained for 50 epochs. L2 loss, $||\mathbf{x} - \hat{\mathbf{x}}||$, was used as objective function.

One consequence of using masking during training is that the last units in the bottleneck layer get fewer updates on their weights than the first units, since they are masked out most of the time. To counteract this, we devised two strategies. First, we pretrained the composable autoencoder without masking for 5 epochs such that all weights would get trained. Second, we used a non-uniform sampling distribution. To this end, we defined a multinomial distribution where the dimension d was selected with a probability that was proportional to either a quadratic or a cubic function. This assured that the trailing components would get updated more often. We thus performed an ablation study with different modifications of the base model.

Results are depicted in Fig. 1. We used an example dataset consisting of control rig of a humanoid character with 455 dimensions and $d_{\text{max}} = 80$. 19 different motion sequences (crouching, walking, jumping, dancing, sword attacks) representing a total of 6154 frames were used for training and 2 held out sequences (high spin sword swing and a combo move) representing 644 frames were used for testing. The legend specifies which modifications were added:

- `uniform`. Base model without pretraining where the masks are selected from a uniform distribution.
- `multinom`. Instead of a uniform distribution, a multinomial distribution was used for selecting d. The probability was proportional to a quadratic function of d.
- `pretrain`. Before the training loop, the model was pretrained for 5 epochs without masking.
- `multinom3`. Instead of a quadratic function, a cubic function was used for sampling the mask.
- `reg`. In addition to the L2 loss, a regularization loss was added on the activations in the bottleneck layer. The loss was $0.0001\,(i-1)\,||\mathbf{a}_i||^2$ where $\mathbf{a}_i$ was the activation of the i-th neuron in the bottleneck layer. In other words, activations in later neurons were increasingly penalized, encouraging the model even more to concentrate information in the first few units.

As the figure illustrates, the regularization term had a detrimental effect, with the later components adding little information. Using pretraining and a multinomial instead of uniform distribution both improved overall results. Using a cubic multinomial function (multinom3) instead of a quadratic one led the lowest overall error, but this came at the expense of worse performance for a smaller number of components. We therefore chose the model with pretraining and a quadratic multinomial probability distribution (`multinom pretrain`, red line) because it offered a good trade-off of performance across the whole component spectrum.

2. Optimality of the REVI algorithm

Optimality of Algorithm (1) requires that $\forall a \in \mathcal{A}_{\text{view}} : b_{\text{enc}}(a) < b_{\text{vid}}(a)$, i.e., streaming an asset as encoded control weights should incur a lower bitrate cost than streaming an asset as video. Here, we will discuss two scenarios involving distant assets and occlusion wherein the requirement can be violated and the encoded bitrate exceeds the video bitrate.

In the first scenario, the asset is far away from the camera and contributes a relatively small number of pixels to the server-rendered video. In this case, encoding as a video can be cheaper than naively sending control weights. How-

---

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

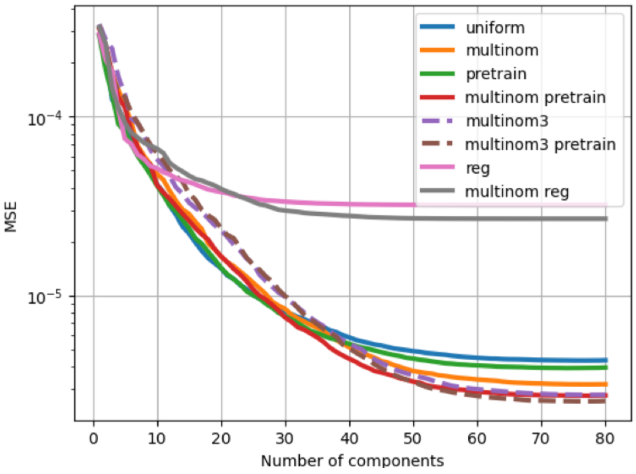

Figure 1. Ablation study for the composable autoencoder. We compared different sampling strategies for the mask (uniform vs multinomial), pretraining, and an additional regularization loss (reg).

ever, the scalability our composable autoencoder provides a possible solution. Unlike standard autoencoders, which are trained on a specific dimensionality, the composable autoencoder allows for on-the-fly tradeoffs between bitrate and perceptual quality. For instance, starting from a given number d of components used, bitrate can be reduced at the expense of perceptual quality by decreasing d. Fig. 2 shows that assets that are further away from the camera indeed require less components to attain a target visual quality. In other words, by scaling down d we can assure that $b_{\text{enc}}(a) < b_{\text{vid}}(a)$ while maintaining perceptual quality.

The second scenario involves occlusions. If one asset completely occludes another asset then rendering the asset on the server will not contribute video bitrate, i.e., $b_{\text{vid}}(a) = 0 < b_{\text{enc}}(a)$. This can be solved by using occlusion awareness in streaming: control weights do not need to be streamed if the asset is occluded. An approximate but simple way for detecting occlusions between assets is to measure the intersection of their bounding boxes. We did not implement occlusion awareness in our experiments and leave its development for future work. Our results show consistent bitrate savings despite between occlusion unaware.

3. REVI manifest file

In the REVI streaming framework, the server provides the client with sufficient information to run Algorithm (1) and select which assets are to be rendered and which are to be sent as compressed scene control weights. To this end, we envision that the server sends a manifest file before each decision time point. The file provides a list of asset IDs followed by their rendering time and bitrate estimates from Random Forest estimators. In the following example we assume that assets are provided from front to back:

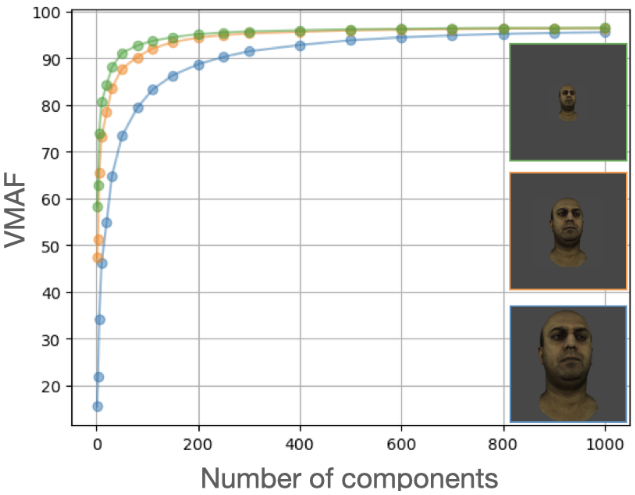

Figure 2. Perceptual optimization. Dimensionality d of the composable autoencoder is plotted against VMAF for an exemplary asset from the Multiface dataset for three different distances to the camera: far (green), middle (orange), and near (blue). The further the asset is away the less components are required to attain a specific quality level.

```
#vf-actor1 10 23.2
#vf-actor2 5 19.1
#vf-actor3 3 15.5
#vf-actor4 1 11.01
#vf-actor5 11 12.1
#background 3 189.1
```

If for instance the client has a rendering time budget of 20%, it could request the server to send the first 3 assets as control weights (at a total rendering time cost of 18%) and receive the remaining assets as server-rendered video. For adequate usage of the server's rendering time estimates, meta information on the server's CPU and GPU capabilities needs to be provided to enable the client to translate server rendering time cost into its own rendering capabilities.

4. Random Forest predictions

Fig. 3 shows Random Forest (RF) predictions of normalized rendering time and bitrate (x-axes) vs. ground truth for third-person view scenes. Each line corresponds to a different scene segment, that is, different set of characters, background, and camera path. The RF is evaluated on the features extracted from the characters.

Fig. 3a shows *cumulative* rendering time predictions. That is, for rendering time, the first datapoint at 0 is for zero assets rendered on the client, first datapoint to the right for the first (closest to camera) asset, second datapoint for the

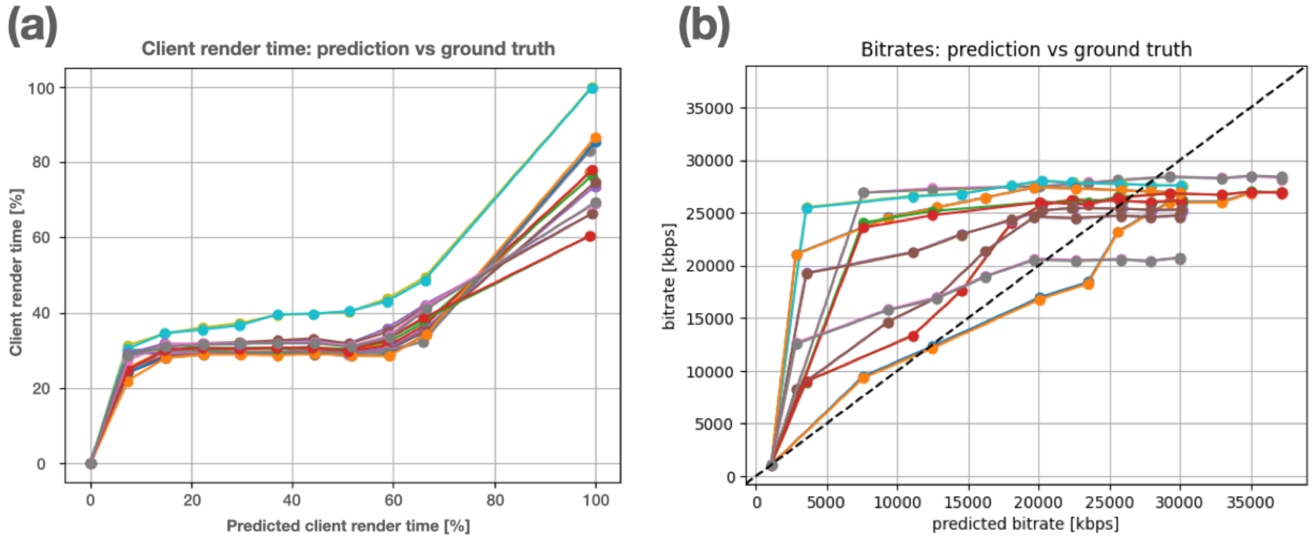

Figure 3. Random Forest predictions of normalized rendering time and bitrate. (a) Real rendering time vs. predicted rendering time. (b) Real bitrate vs. predicted bitrate.

first 2 assets, and so on. The last datapoint corresponds to the background asset. It is more costly to render the background as opposed to a single character, this explains the jump for the last data point.

Fig. 3b shows *cumulative* bitrate predictions. This is best interpreted from the server perspective. For the first datapoint (bottom left), the server only streams the control weights. The second datapoint corresponds to streaming the background as video plus all other assets as control weights. Since the background fills most of the image it incurs a high bitrate. This explains the big jump in bitrate. We can see that while the predicted bitrate (x-axis) inreases steadily, the real bitrate (y-axis) levels off in many cases and even reduces in some cases. In particular, we can identify different patterns for scenes with a static camera vs. scenes with a dynamic camera. In scenes with a static camera, the background does not move and is easily compressed by the video codec. It is the moving characters that add dynamics and hence bitrate. Static scenes correspond to the lower lines in Fig. 3b that most closely follow the diagonal. In scenes with a dynamic camera, the scene background itself is moving and hence incurs a high bitrate cost. Here, dynamic characters do not incur much additional bitrate since the character just occludes the background (which is moving too). In some cases, such as when the texture of the character is simpler than the texture of the background, adding a character can even slightly reduce bitrate (small dips in the line). Dynamic scenes correspond to the upper lines in Fig. 3b that do not follow the diagonal very well.

If the RF predictions were highly accurate, the line of predicted vs. real rendering time and bitrate should be close to the diagonal. This is not the case for many scenes. The reason is that implicitly the RF prediction assumes additivity, that is, the total cost for, e.g., 5 assets should be the sum of the costs of each asset in isolation. For rendering time, we find that rendering a single asset incurs an additional constant rendering cost that is likely related to steps in the rendering pipeline such as initializations. Adding a second asset adds a lower cost since these steps have already been performed. For bitrate, we find that additivity does not always hold (e.g., the total bitrate for three characters is not necessarily the sum of their individual bitrates), due to mutual occlusions of characters or characters superimposed on a moving background.

5. HEVC encoding

All video encodings were carried out with HEVC, using the low-latency oriented recipe:

```
ffmpeg -vsync 0 -i <input> -c:v libx265 -preset medium -tune psnr -crf 23 -maxrate 20000k -bufsize 40000k -x265-params keyint=120 :min-keyint= 120: bframes=0: rc-lookahead=0: sync-lookahead=0: no-mbtree=1: scenecut=0 -an <output>.
```

6. Example videos

We provide a set of videos for an example scene with 7 characters. It illustrates the rendering on the client, server, and the composite client+server rendering. The filename describes how many characters were rendered. For instance, `eevee_5server_2client.mp4` indicates that 5 assets were rendered on the server and 2 on the client. Note that for this visualization the closest assets were determined on a frame-by-frame basis. Furthermore, the view used for the distance calculation was larger than the camera view, so the closest asset may not be actually visible in a given camera view, leading to less characters displayed in the client view. The videos can be played with ffmpeg's `ffplay`.

7. Results for FPV and Metaverse/VR

Fig. 4 shows additional results of Experiment 1 for first-person view (FPV) and Metaverse/VR scenes. Just like for third-person view (TPV), bitrate saving increases with client rendering time. The REVI w/o RF model consistently outperforms the REVI w/RF model. For FPV scenes (Fig. 4a) we furthermore find that REVI w/o RF consistently outperforms the fixed baseline methods, yielding higher bitrate savings for the same render time.

For Metaverse/VR scenes (Fig. 4b), the results are more mixed. We observe a non-monotonic relationship between render time and the number of assets (see digits in figure). For instance, going from rendering 1 character to rendering 2 characters, average render time increases as expected, but going from 2 to 3 it decreases. We believe that this is due to occlusion effects: the head avatars might be simpler to render that some of the background elements. Therefore, adding them to the scene might occlude more complex background elements in some cases and therefore decrease total rendering time. For the Client-Foreground setting (left panel), REVI w/o RF again outperforms the adhoc methods. For the Client-Background setting, we found that REVI outperforms the fixed baseline for low bitrate savings but underperforms for higher bitrate savings.

8. Minimizing rendering time

The optimization problem considered in the main paper is

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

In other words, we minimize bitrate subject to a rendering time constraint. Let us now consider the complementary optimization problem whereby we minimize client rendering time subject to a bitrate constraint, given by flipping the roles of the terms,

$$\begin{aligned} \text{minimize } & c(\mathcal{A}_{\text{client},t}) \\ \text{subject to } & b_{\text{vid}}(\mathcal{A}_{\text{server},t}) + b_{\text{enc}}(\mathcal{A}_{\text{client},t}) \leq \Theta_t \end{aligned} \tag{1}$$

To test this, we used the third-person view data in the Client-Foreground scenario. Fig. 5 shows the results. We observe a mixed pattern of results. The REVI w/RF model performs en par with fixed baselines that always use 1, 2, or 4 characters (respective digits in the figure). The model fails to target higher bitrates, which is due to limited accuracy in bitrate and rendering time predictions. The REVI w/o RF model does not rely on the bitrate predictions. For lower bitrates, it is outperformed by the 1, 2, and 4 character fixed baseline (which requires lower rendering time for an equivalent bitrate), but it out performs the 6, 8, and 9 character baseline for higher bitrates.

Concluding, this illustrates that REVI can be successfully applied to the complementary problem of minimizing rendering time subject to bitrate constraints. However, in this problem REVI outperforms a fixed baseline for higher bitrates but not for lower bitrates.

9. Equivalent bitrate encoding

Previous analyses have shown that REVI enables bitrate savings by streaming both video and compressed control weights and distributing rendering time between the server and client. Here, we provide evidence that when bitrate is fixed, REVI provides better visual quality vis-à-vis a full-cloud rendering solution. As shown in Fig. 6, at an equivalent bitrate, the REVI solution provides more crisp details and high-frequency content. This is particularly evident for irregularly textured surfaces such as the grass in Fig. 6a and clothing details such as the pants in Fig. 6b.

10. Global Illumination

In the main paper, we used Blender's EEVEE real-time rasterization pipeline with light probes. For comparison, we show a visualization of one our scenes using Blender's CYCLES engine [2] in Fig. 7. In this case, High-Definition Range (HDR) maps [1] are synced with the client. HDR maps allow for termination of rays without too many bounces by embedding the environment within a sphere upon which a HDR map is projected. The path tracing results show deeper shadows and a richer dynamic range. Implementing it into our REVI framework would require either offline delivery or online streaming of HDR maps.

References

[1] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 189–198, New York, NY, USA, 1998. Association for Computing Machinery. 4

[2] Blender Docs. Cycles, 2024. Accessed: 2024-03-20. 4

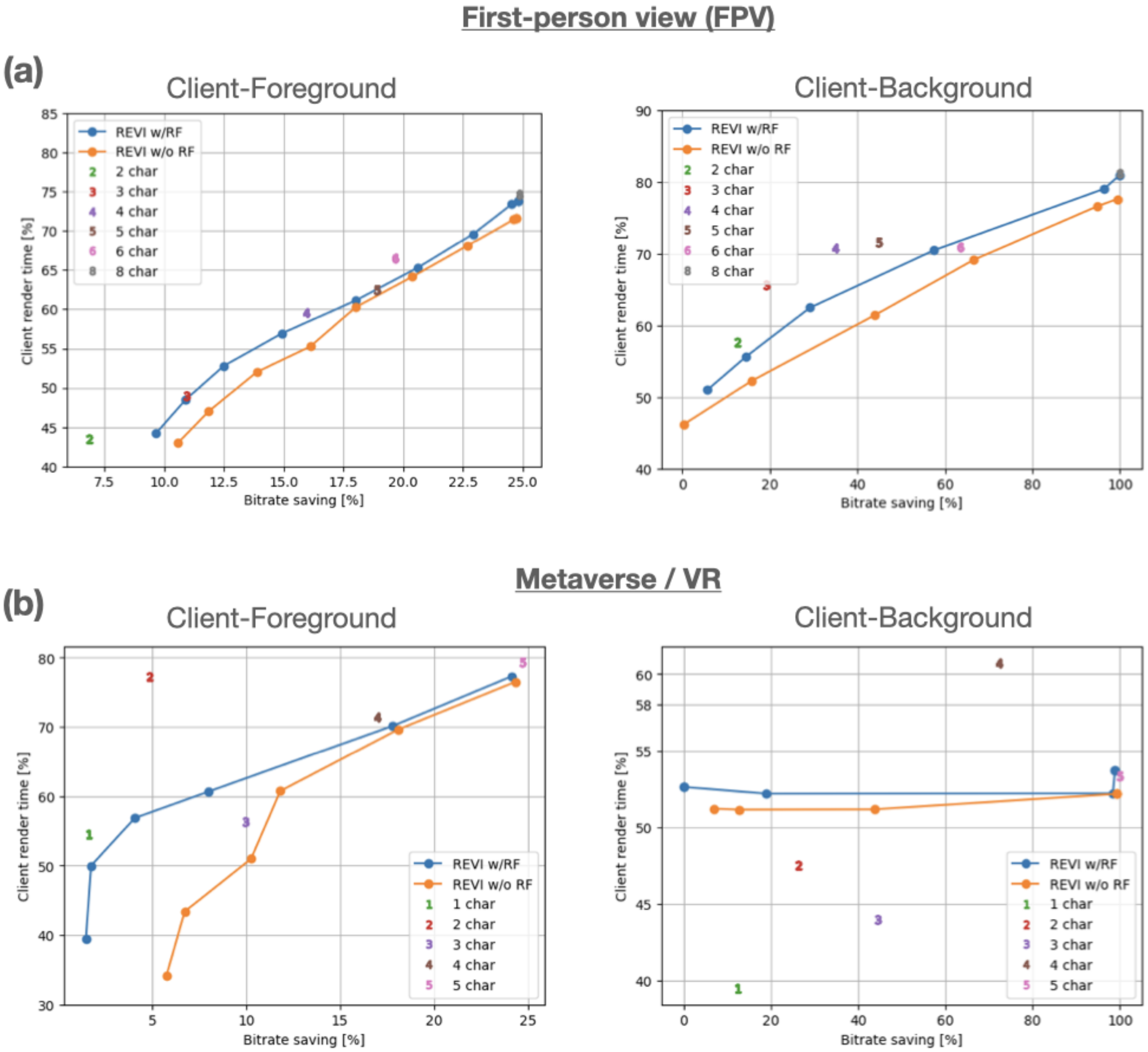

Figure 4. REVI result for FPV and Metaverse/VR. Points closest to the bottom right are optimal. (a) Bitrate savings for FPV scenes. The y-axis shows the relative client rendering time, averaged across all clips. The x-axis shows the bitrate saving obtained with the corresponding method when compared to the full cloud rendering solution. Each data point represents a different rendering time target. Left: Client-Foreground. Right: Client-Background. (b) Bitrate savings for Metaverse/VR scenes.

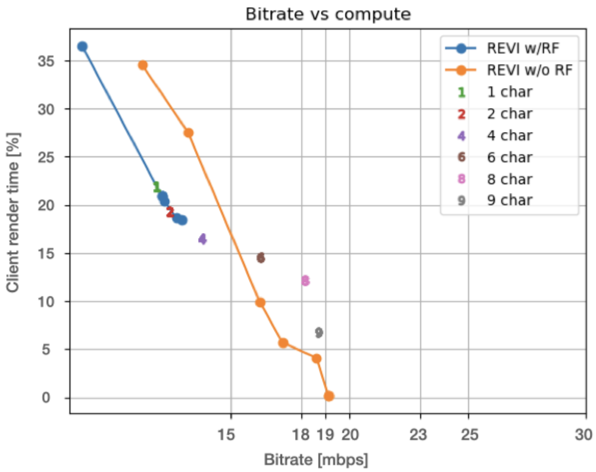

Figure 5. AIMD results for minimizing rendering time subject to bitrate constraints for the third-person data in the Client-Foreground scenario. The x-axis shows the average total bitrate (sum of video bitrate and scene control weights bitrate) and the y-axis show the average rendering time percentage on the client. For the low bitrate range < 15, REVI does not outperform the fixed baseline which sends a fixed number of 1, 2, or 4 characters as control weights for rendering on the client. For the higher bitrate range > 15, however, REVI requires requires significantly less client rendering time than the fixed baseline for the same bitrate target.

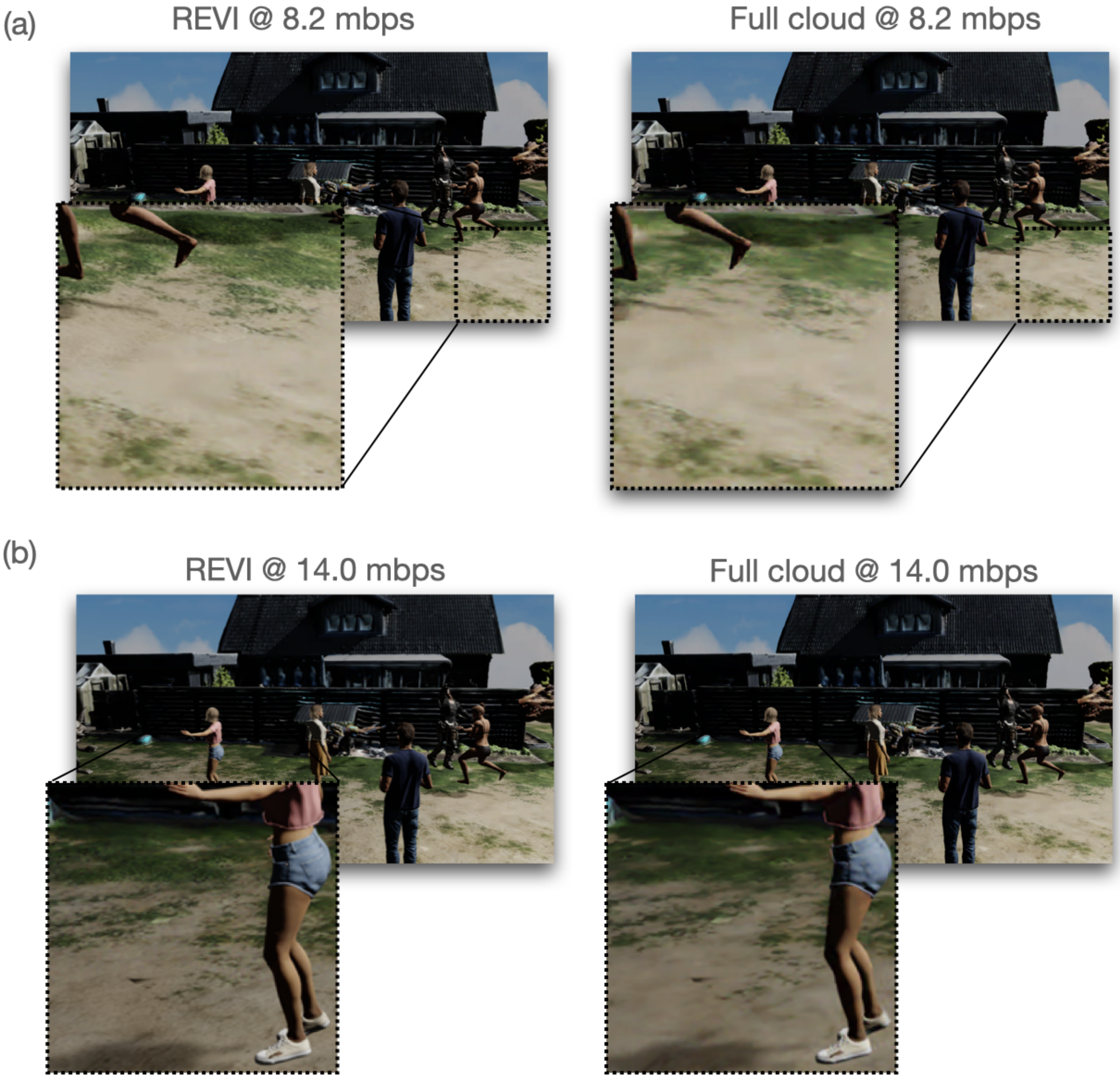

Figure 6. Example frames for bitrates obtained with REVI (background is rendered on server, all characters are rendered on client) vs. cloud rendering (everything rendered on server) at equivalent bitrates. (a) REVI vs. full-cloud at 8.2 mbps. (b) REVI vs. full-cloud at 14.0 mbps. Compared to REVI, cloud-based rendering has losses in texture crispness and reduced level of detail.

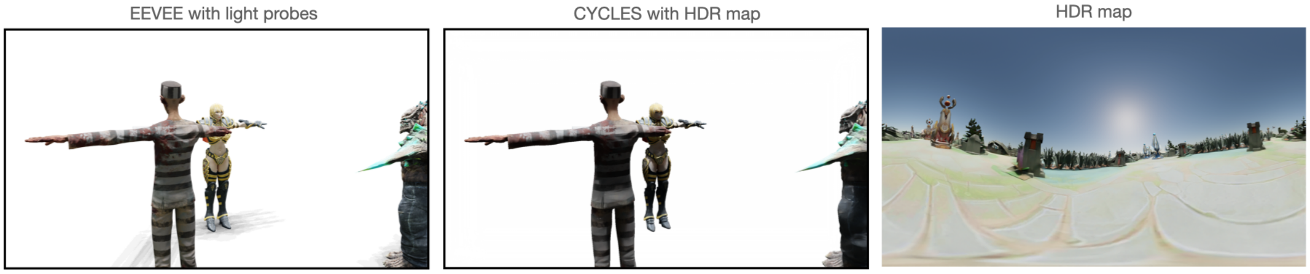

Figure 7. EEVEE rasterization pipeline with light probes (used in the main paper) compared to a path-tracing pipeline CYCLES with HDR maps.