Real-Time Virtual Reality Streaming with Neural Super-Resolution

Xingyuan Sun, Jimmy Wu, Runzhe Yang

Abstract

In this work, we tackle the problem of real-time virtual reality streaming in low-bandwidth network conditions. In low-bandwidth settings, VR streaming faces an inherent trade-off between responsiveness and image resolution, both of which are crucial to a high quality user experience. To overcome this trade-off, we propose to send low-resolution frames over the network and then apply neural super-resolution on the client to generate high-resolution frames. We implement a client/server setup for a 3D virtual environment as a proof of concept, and demonstrate that even under low-bandwidth (12 Mbps) network conditions, our system is still able to deliver a fast, responsive, and high-resolution VR experience to the end user.

1 Introduction

Recently, virtual reality (VR) has been gaining popularity as a technology of human-machine interaction. By simulating virtual scenes in a 3D space similar to that of real physical environment, virtual reality is revolutionizing the way we experience gaming, movies, specialized training [6], and other immersive applications.

In this work, we are interested in real-time virtual reality streaming in low-bandwidth network conditions. Streaming of high resolution image frames in real-time has very heavy bandwidth requirements, and high bandwidth connections are not always affordable or feasible. This limitation stands as one of the main technological bottlenecks to real-time VR streaming. Unfortunately, VR streaming in a low bandwidth environment is infeasible due to either poor latency/throughput or low resolution, both of which would severely limit the quality of experience for the end user.

When streaming real-time VR in a low-bandwidth environment, there is an inherent trade-off between the responsiveness of the user experience and the resolution of the image feed. Attempting to send high-resolution image frames when there is not enough network bandwidth results in each frame taking a very long time to send. This results in high latency and a very low frame rate for the end user.

Alternatively, we can get around the low-bandwidth limitation by simply streaming low-resolution image frames to the end user. Such an approach does not suffer from poor latency or low frame rate, but will still result in a worsened user experience since the frames will be low-resolution and very grainy.

In this work, we propose a method to enable high quality, real-time VR experiences in low bandwidth environments. We focus on maintaining a high quality of experience for the end user while greatly reducing the bandwidth requirement of the system. Our approach is to send low-resolution frames from the server to the client, and then use neural super-resolution on the client to improve the quality of the image frames. Low-resolution frames are less bandwidth intensive than high resolution frames, and allow for low latency and high throughput while they are being sent to the client. This keeps the VR experience fast and responsive, both of which are crucial components of a good user experience.

On the client side, rather than directly showing low-resolution frames, which would degrade the user experience, we use a very lightweight neural super-resolution model to improve the quality of the image before showing it on the screen. In this way, we are able to maintain a high quality of experience for the end user while drastically cutting down on the network bandwidth required by the system.

As a proof of concept, we built a real-time virtual reality client and server for a 3D photorealistic in-

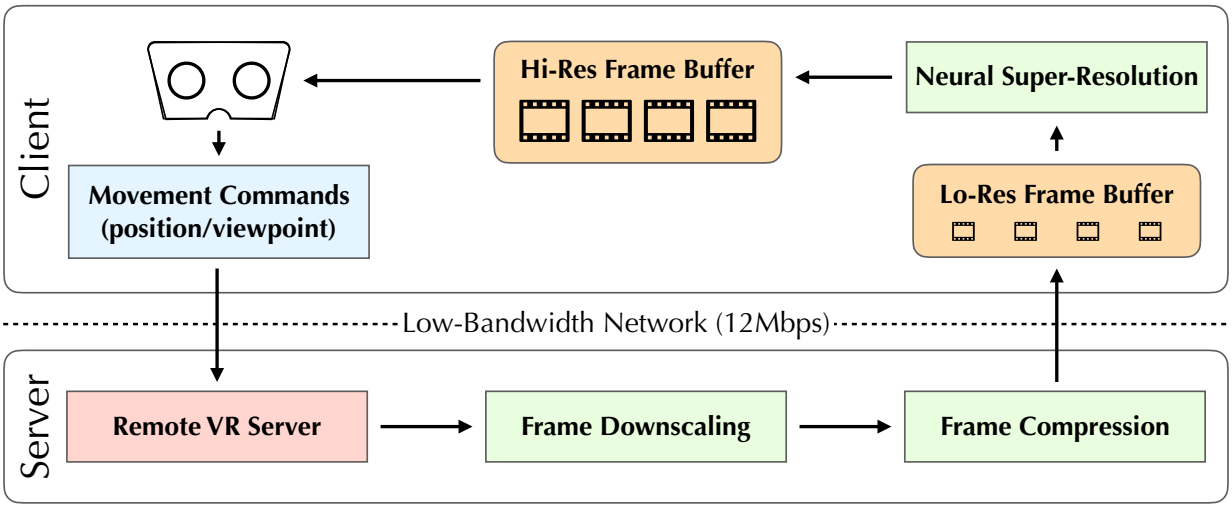


Figure 1: An overview of our system design for real-time VR in a low-bandwidth networking environment. We run a photorealistic 3D virtual environment on the remote VR server, which processes movement commands from the client and sends compressed, low-resolution frames to the client. The client then performs neural super-resolution on the received frames to generate high-resolution image frames, which are then shown to the user. This design allows our system to deliver high quality VR to the end user while using very little network bandwidth.

door virtual environment, and ran the system under a simulated low bandwidth network to demonstrate the effectiveness of our approach. With a connection between the client and server of only 12 Mbps, the client is able to navigate inside a high-resolution (1440x1080) indoor virtual environment resolution at over 10 frames per second.

We measured the amount of network bandwidth required by our approach and compared with a baseline that directly sends high-resolution frames over the network. Our approach significantly reduces the network bandwidth required by the system, but does not sacrifice. Under low-bandwidth network conditions, we are still able to provide an experience for the end user that is fast, responsive, and high-resolution.

In the following sections, we will describe our detailed system design in Section 2, present qualitative and quantitative evaluation of our system in Section 3, describe related work in Section 4, and conclude in Section 5.

2 Design

We designed and implemented a client/server setup as a proof-of-concept of our approach. Our system is illustrated in Figure 1. We run a photorealistic 3D virtual environment on the remote server, which maintains the current state of the environment. The server processes movement commands from the client and renders low-resolution frames to send back to the client over the low-bandwidth network. The client acts as an user-friendly interface between the end user and the virtual reality engine on the server. The client also performs neural super-resolution of the low-resolution frames it receives, allowing it to present high quality frames to the end user even though the system is operating in a low-bandwidth environment. We now proceed to describe each component in more detail below.

2.1 Server

The remote VR server is mainly responsible for running the 3D virtual environment and communicating with the client. We use the open-source AI2-THOR [2] framework, which provides a photorealistic 3D virtual environment, complete with a Python API for interaction with the environment. Please see Figures 3 or 4 for example screen captures of the 3D environment as visualized by the client. The environment consists of 120 3D indoor scenes across four

different scene types: kitchens, living rooms, bedrooms, and bathrooms.

The server periodically receives movement commands from the client, expressing what actions the user wishes to perform inside the environment. The user is initially placed into a canonical position and viewpoint inside a scene, and is able to move in all four directions and rotate the viewpoint left or right. The user is also able to freely switch among the 120 different scenes in AI2-THOR.

Whenever the server processes a movement command from the client and updates the player state, it will then proceed to send an image frame of the updated 3D environment to the client. In order to overcome the low-bandwidth constraints of the network, the server will send a compressed, low-resolution version of the frame over the network, which the client will process with super-resolution to maintain a high quality of experience for the user.

The rendering engine on the server can render image frames in either low-resolution (480x360) or high-resolution (1440x1080). As shown in Figure 3, rendering frames directly in low-resolution results in lots of image artifacts. In our system, we found that it worked best to run the VR engine in high-resolution mode, and then use bilinear interpolation to downscale the high-resolution frames to 480x360, as this resulted in much fewer image artifacts.

Once the image frame has been downscaled to 480x360, we do not send the raw array image data directly to the client, as the size of an RGB images can be greatly reduced using JPEG compression at almost no perceptible cost to image quality. For example, a 480x360 array image frame is 507 KB, whereas the JPEG compressed version is only 36 KB, over 14 times smaller. The server thus uses JPEG to further reduce the bandwidth usage of our system, compressing the 480x360 downscaled image frame to a JPEG bytestring before sending the frame over the network to the client.

2.2 Client

The client is the main interface of our system with the end user, and could be a real-time VR headset worn by the user. In our case, as a proof of concept, we implemented a desktop client in which the user can interact with the 3D AI2-THOR environment as if he/she was playing a photorealistic computer game. The 3D virtual environment is streamed in real-time to the client machine, allowing the user to interact as if the engine was running directly on the local machine. The most recent frame from the VR server is displayed on the computer screen, and the user is able to use keyboard commands to navigate inside the 3D environment.

The keyboard commands input into the system, such as move forward or turn right, are sent over the network to the VR server, where the state of the 3D environment is update accordingly. The server then sends a compressed, low-resolution 480x360 image frame back over the network, reflecting the updated environment state. The client will then run neural super-resolution on the received frame to generate a detailed, high-resolution 1440x1080 frame that is shown to the end user. We describe the setup and implementation details of the neural super-resolution model we used in Section 2.4.

2.3 Low-Bandwidth Network Emulation

In order to test how well our system works in low-bandwidth network conditions, we used the `mm-link` tool from the Mahimahi network emulation tool set to simulate a low-bandwidth connection, restricting the client-server connection to 12 Mbps. Once we established the low-bandwidth connection, we used the `speedtest-cli` tool to verify that the connection is indeed 12 Mbps each way. We then compare and benchmark how our system runs in various setups with and without network emulation enabled. We show such results in our evaluation in Section 3.2.

2.4 Neural Super-Resolution

In a low-bandwidth environment, sending low-resolution frames allows the client to be fast and responsive, but the low-resolution image feed degrades the otherwise high quality experience. One way to remedy the low resolution is to use super-resolution to enhance the resolution and content of the image frame.

Sub-Pixel CNN. For our application, we desire a lightweight, fast, and effective way to do super-resolution. We thus chose to adopt the Sub-Pixel CNN [5], which satisfies all of these criteria. The

Sub-Pixel CNN is an efficient and lightweight convolutional neural network (CNN) designed for real-time super-resolution of 1080p videos.

We use the implementation of Sub-Pixel CNN provided as part of the official PyTorch code examples [1]. The neural network model is very lightweight, with model weights that are only 245 KB (the size of only 7 compressed low-resolution images) and can easily be sent from the server to the client. When deployed on the client, the super-resolution adds a slight overhead of just 0.04 seconds per frame, while greatly enhancing the frame quality.

Model Training. We adapt the Sub-Pixel CNN for use in our system by training the model on image frames from the 3D virtual environment. We run the server in high-resolution mode (1440x1080) and collect 72 high-resolution screen captures from each of the 120 different scenes in the AI2-THOR environment, for a total of 8640 images. The 72 images per scene correspond to having the agent stand in place in each scene and take a screen capture at 72 different viewpoints in 5 degree increments. This collection of high-resolution image frames constitutes our training dataset for the Sub-Pixel CNN. Note that since our training data spans all 120 scenes in the AI2-THOR environment, we can train a single Sub-Pixel CNN model and use the same model for super-resolution in all 120 different scenes.

We also collect 5 additional screen captures for each of the 120 scenes for use in evaluating our trained model. We implemented a screen capture functionality on the client and manually moved around inside each scene to collect 5 representative images per scene. This collection of 600 images constitutes our evaluation dataset.

The Sub-Pixel CNN model is trained to apply a super-resolution that enlarges images by a factor of 3. During training, we bilinearly downscale each image by a factor of 3, and train the model to recover the original high-resolution image. We use the suggested hyperparameters of batch size 4 and learning rate 0.001, and train for 120 epochs on our custom dataset. We evaluate our trained model using our evaluation dataset and present our results in Section 3.4. Note that the model training is done entirely offline, so the training time has no effect on the end user experience.

Model Deployment. Once the Sub-Pixel CNN model is trained, we deploy the model on the client. This involves a one-time transfer of the model weights from the server to the client. The weights are only 245 KB in size and equivalent to 7 low-resolution image frames. Every time a low-resolution 480x360 frame is received over the network by the client, the client will run the image frame through the Sub-Pixel CNN, generating a high-resolution 1440x1080 frame that is shown to the end user.

3 Evaluation

In the following sections, we evaluate our system, showing performance measurements in Sections 3.1 and 3.2, qualitative comparisons with baselines in Section 3.3, as well as a quantitative evaluation of the neural super-resolution model in Section 3.4.

3.1 Bandwidth Usage

We measured the bandwidth usage of our system by recording the size in bytes of each frame sent by the server. As mentioned in Section 2.1, we use JPEG image compression across the board as it gives over an order of magnitude reduction in size with very small reduction in image quality. For high-resolution 1440x1080 frames, each frame is 263 KB in size, while for low-resolution 480x360 frames, each frame is 36 KB in size. We see that sending low-resolution frames over the network rather than high-resolution directly gives over 7x reduction in the bandwidth required per frame. Furthermore, by applying super-resolution to low-resolution frames, our system is able to deliver high quality image frames to the user while only using the bandwidth of a low-resolution system.

3.2 Latency and Frame Rate

We show in Figure 2 latency and frame rate measurements of our real-time virtual reality system. We take the measurements on the client, and run all measurements for about a minute, beginning measurement after a 3 frame warm-up period since the client takes

[1] https://github.com/pytorch/examples/tree/master/super_resolution

Network Bandwidth	System Setup	Latency (s)	Frame Rate (fps)
800Mbps	480x360	0.0286	24.86
	1440x1080	0.0939	9.51
	1440x1080 (ours)	0.0622	**14.49**
12Mbps	480x360	0.0617	13.76
	1440x1080	0.2825	3.40
	1440x1080 (ours)	0.0995	**9.38**

Figure 2: We measure the latency and frame rate of our system, which transfers low-resolution frames that become 1440x1080 after neural super-resolution. We consider both low-bandwidth (12 Mbps) and high-bandwidth (800 Mbps) network conditions, and compare with baselines that do not use super-resolution. We find that in low-bandwidth conditions, our method is able to deliver high-resolution 1440x1080 frames at 9.38 fps. This is comparable to the frame rate of 9.51 fps achieved by the baseline high-resolution setup, as measured in much better network bandwidth (800 Mbps) conditions.

some time to initialize. The latency is defined as the time it takes for the display to update after a key-press, and measures the per-frame round trip time to the server and back. The frame rate is simply computed as the number of frames received divided by the length of the time period over which those frames were received.

From our measurements in Figure 2, we can observe that the baseline setup for 480x360 always has a higher frame rate than the 1440x1080 setup, because (1) it takes longer to transfer the higher resolution frames over the network, and (2) it takes longer for the VR engine to render high-resolution frames. We note that over the 800 Mbps connection, the frame rate for streaming 1440x1080 is bottlenecked not by the network bandwidth, but by the rate at which the server can render the high-resolution frames.

When comparing our system, which applies super-resolution to 480x360 frames, to the baselines, we observe that the frame rate is lower than that of the 480x360 baseline. This is because the neural super-resolution adds an overhead of about 0.04 seconds per frame. However, in comparison with the low-resolution 480x360 frames, the frames generated by our system are of much higher quality, as shown in Figure 4. Compared to the 1440x1080 baseline, we observe that the high-resolution baseline is simply infeasible in 12 Mbps conditions, whereas our system is able to run at 9.38 fps, comparable to the 9.51 fps frame rate achieved by the high-resolution baseline in 800 Mbps conditions.

3.3 Qualitative Comparison with Baselines

We perform a qualitative evaluation by running our system in low-bandwidth network conditions and comparing with baselines. As indicated by the frame rate measurements in Figure 2, the high-resolution setup is infeasibly slow at 12 Mbps, so we qualitatively compare with the 480x360 baseline. We show several example comparisons in Figure 4.

Across all scenes we looked at, there was a noticeable improvement in image quality of our system compared with the low-resolution baseline. We also qualitatively compare our system with the high-resolution baseline running in high-bandwidth conditions. An example is shown in Figure 3, where (b) is compared with (d). Upon direct comparison, our generated high-resolution frames are not as finely detailed as the baseline 1440x1080 frames, owing to the limitations of super-resolution. However, our system has a much lower bandwidth requirement than the baseline 1440x1080 setup, and is able to run smoothly even in low-bandwidth conditions.

3.4 Evaluation of the Sub-Pixel CNN

As discussed in Section 2.4, we constructed a dataset to evaluate the super-resolution performance of the Sub-Pixel CNN model. The criteria most commonly used in evaluating super-resolution methods is the peak signal-to-noise ratio (PSNR). Our trained model achieves an average PSNR of 38.08 dB.

4 Related Work

Adaptive Bitrate Video Streaming. There have been a lot of recent work on adaptive bitrate video streaming schemes to address poor network conditions. These methods propose to use machine learning to dynamically adjust the video streaming bitrate based on the prevailing network conditions. For example, Pensieve [4] applies deep reinforcement learning to determine the best bitrate for video streaming. However, these methods do not directly enable real-time VR streaming, as they still must tradeoff between resolution and responsiveness in low-bandwidth conditions.

NAS [7] is another ABR system for improved video streaming that extends Pensieve. In particular, rather than simply doing bitrate selection, NAS also proposes the use of deep neural networks (DNNs) for super-resolution of lower quality frames. Their strategy of applying super-resolution to lower quality frames serves as the main inspiration for our work. In contrast to their work, we apply super-resolution to real-time virtual reality streaming rather than video streaming, and focus exclusively on delivering a high quality user experience in low-bandwidth network conditions.

Super-Resolution. Recently, there has been a lot of work in the field of deep learning on learned methods for image super-resolution [1, 3]. However, deep learning models are known to often be heavyweight and computationally expensive. Fortunately, with the development of deep learning approaches for many real-world applications, deployment of deep learning on mobile and low-power compute platforms has become an very active area of research. For our work, we adapt the Sub-Pixel CNN [5] since its efficiency and lightweight design is well-suited for our system.

5 Conclusion

We proposed a method to enable high quality VR streaming in low-bandwidth network conditions. Our approach is to send frames from the rendering server to the client in low-resolution, allowing our system to be fast and responsive. To also maintain high image quality for the user, we use neural super-resolution on the client machine to upscale the low-resolution frames to high-resolution before displaying them. Our system is able to deliver a VR experience that is fast, responsive, and high-resolution, even under low-bandwidth network conditions.

References

[1] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[2] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017.

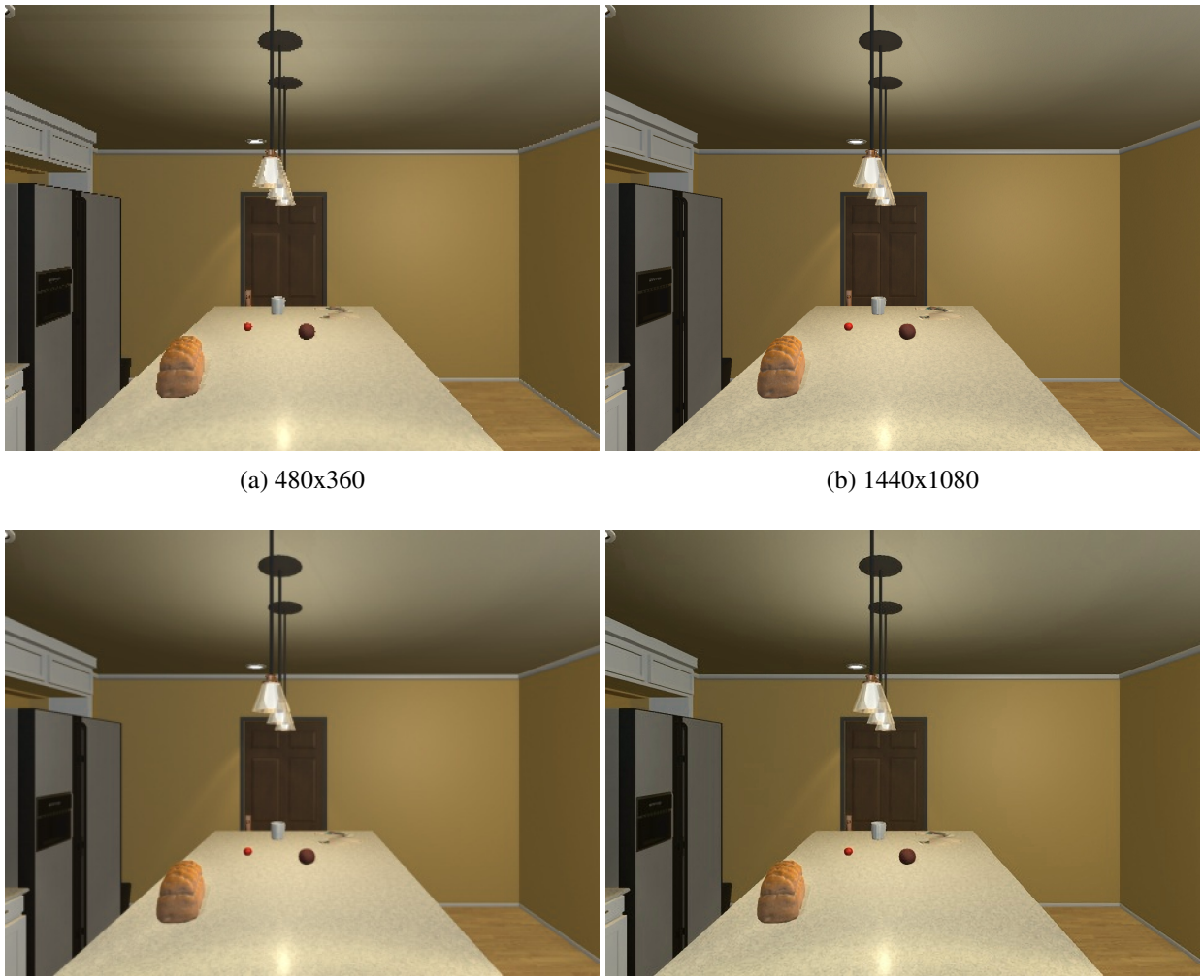
[3] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[4] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[5] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] N. Singh and S. Singh. Virtual reality: A brief survey. In *2017 International Conference on Information Communication and Embedded Systems (ICICES)*, pages 1–6, Feb 2017. doi: 10.1109/ICICES.2017.8070720.

[7] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 645–661, 2018.



(a) 480x360

(b) 1440x1080

(c) bilinear downsampling

(d) ours

Figure 3: A visual comparison of different setups. Running the 3D virtual environment directly in low-resolution (480x360) or high-resolution (1440x1080) mode gives frames that look like (a) or (b), respectively. However, the low-resolution frame from the 3D environment (a) has too many image artifacts. In our approach, we instead bilinearly downscale the high-resolution frame from the 3D environment (b) to get a smoother low-resolution 480x360 frame (c), which the server sends to the client. The client then applies neural super-resolution on (c) to get (d), the final upscaled high-resolution 1440x1080 frame. With this approach, our system is able to deliver high quality frames to the end user, while only using the low network bandwidth of a low-resolution setup.

(a) 480x360 (b) ours

(c) 480x360 (d) ours

(e) 480x360 (f) ours

Figure 4: Qualitative comparisons of our system with a low-resolution baseline. On the left, we show frames rendered from the 3D environment in low-resolution mode (480x360). This mode requires very low network bandwidth since the image frames are small. On the right, we show that our system presents much higher quality frames to the end user, while maintaining the same low network bandwidth requirement as the low-resolution baseline.