An Easy-to-Use Pipeline for an RGBD Camera and an AR Headset

Hanseul Jun
Jeremy N. Bailenson
Stanford University

Henry Fuchs
University of North Carolina at Chapel Hill

Gordon Wetzstein
Stanford University

Introduction

The contribution of this article lies in providing the code for a working system running with off-the-shelf hardware, but not in advancing theory in computer vision or graphics. The current work presents a system which uses one RGBD camera (Microsoft Kinect v2) to capture people in places, and an AR headset (Microsoft HoloLens) to display the scene. While the fidelity of the system is relatively low compared to others which utilize multiple cameras (i.e., Orts-Escolano et al., 2016), it displays with a high frame rate, has low latency, and is mobile in that it does not require a render computer.

Previous Work

Maimone and colleagues (2013) previously examined the connection between RGBD cameras and AR headsets. Their system used two Kinect devices to scan and render a space in an AR headset for avatar-based telepresence, and projectors to augment the brightness of the AR headset. Kowalski, Naruniec, and Daniluk (2015) also examined the combination of RGBD cameras and AR headsets, using multiple Kinect v2 devices to convert a space into a stream of point clouds. Compared to these systems, the current work is different in that it is designed to work in a minimalist situation—without a render machine and using a single camera.

Valorem Reply (2016) introduced HoloBeam, a proprietary system that connects users through Kinect v2 devices and HoloLens devices. Vimeo also introduced a proprietary streaming system for holograms (Fleisher, 2018). While the terms the companies used differ, the visual outcomes of these systems, including the system of this article, do not largely differ from each other.

It should be noticed that there is continuing work for streaming spatial information based on more sophisticated models with better visual quality. As a recent example, Wei and colleagues (2019) introduced a technique that reconstructs the face of a VR headset user in a form it can be rendered in a virtual environment, allowing realistic rendering of people in virtual environments without the headsets they are wearing.

Presence, *Vol. 27, No. 2, Spring 2018*, 202–205
https://doi.org/10.1162/PRES_a_00326

Description of the System

The hardware part of this system consists of two parts: laptop with a Kinect v2 and a HoloLens. The software also consists of two parts: sending the pixels from a Kinect v2 to a HoloLens and rendering the pixels in the HoloLens. This process faced three challenges which emerged from three particular characteristics of the devices.

The device characteristics were:

1. Kinect v2 produces large amounts of data.
2. HoloLens requires wireless transmission.
3. HoloLens has low computational power.

The challenges were:

a. Due to (1) and (2), compression is required.
b. Due to (a) and (3), a computationally efficient decompression is required.
c. Due to (3), an efficient rendering technique is required.

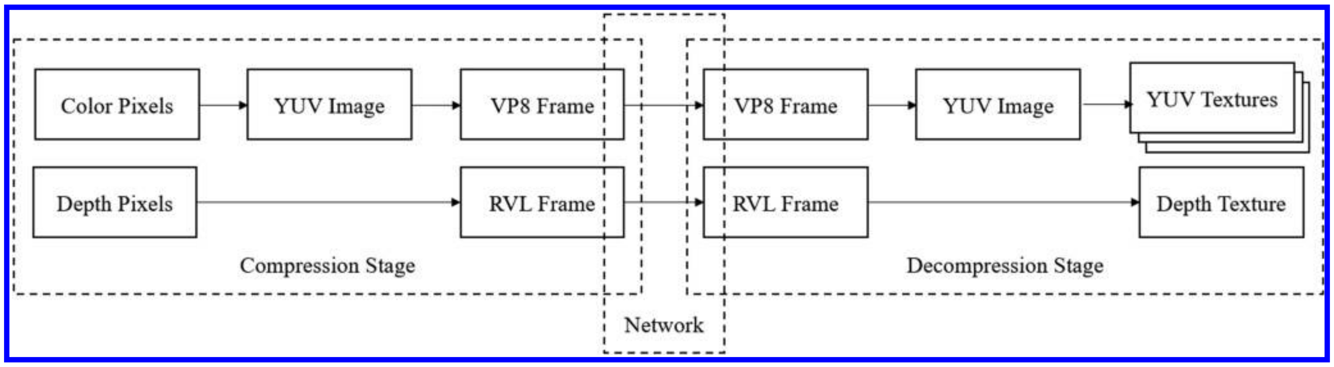

Figure 1. *Overview of compression and decompression of our system. The compression stage is for the laptop connected to a Kinect v2. After the compression, the VP8 frame and the RVL frame get transmitted to a HoloLens. The decompression stage is for the HoloLens that received the frames.*

Compression and Decompression

Kinect v2 devices have color pixels and depth pixels. For the color pixels, we adopted the VP8 codec, using libvpx and FFmpeg for encoding and decoding. While H.264 has shown at least equivalent performance, we instead chose VP8 since libvpx is licensed under the revised BSD license that is consistent with our goal to provide the source code of this system. For depth pixels, as we wanted to avoid the uncertainty of lossy compression, RVL (Wilson, 2017) has been chosen for compression as it is computationally inexpensive and lossless. Figure 1 provides an overview of the compression and decompression process of our system.

Compression Stage

Using a laptop connected to a Kinect v2, the color pixels are encoded by libvpx into VP8 frames and the depth pixels are RVL compressed into RVL frames. While performing this function, the width and height of the resolution of color pixels were halved since their resolution (1920 x 1080) was redundantly large compared to the resolution of the depth pixels (512 x 424), especially for our visualization technique which pairs each depth pixel to a color value. Both pixels get sent to the HoloLens through a wireless network.

Decompression Stage

Our system decodes the encoded color pixels using FFmpeg and converts the decoded pixels into three 8-bit single channel Direct3D textures, each of them belonging to a color channel of YUV. To avoid conversion from YUV420—a dominant color space for video streaming since it allows 4x compression in U and V channels—to RGB that is computationally expensive, we chose to create textures in the YUV color space. We used three textures instead of one to avoid realigning the decoded pixels into a single texture. The pixels compressed with RVL were decompressed into a 16-bit single channel Direct3D texture.

Rendering Stage

With the YUV textures and the depth texture, a HoloLens can render each pixel of the depth texture into a quad floating on the space and color the quads using the color values from the YUV textures. Our rendering technique requires a mesh precomputed with the intrinsic variables of the Kinect v2. In a vertex shader, with the depth texture, the precomputed mesh turns into a group of points that reflects the depth values of the depth texture. In a geometry shader, these points turn into quads. The size of the quads was chosen to be the maximum size that does not intrude adjacent quads. Finally, in a fragment shader, the quads are

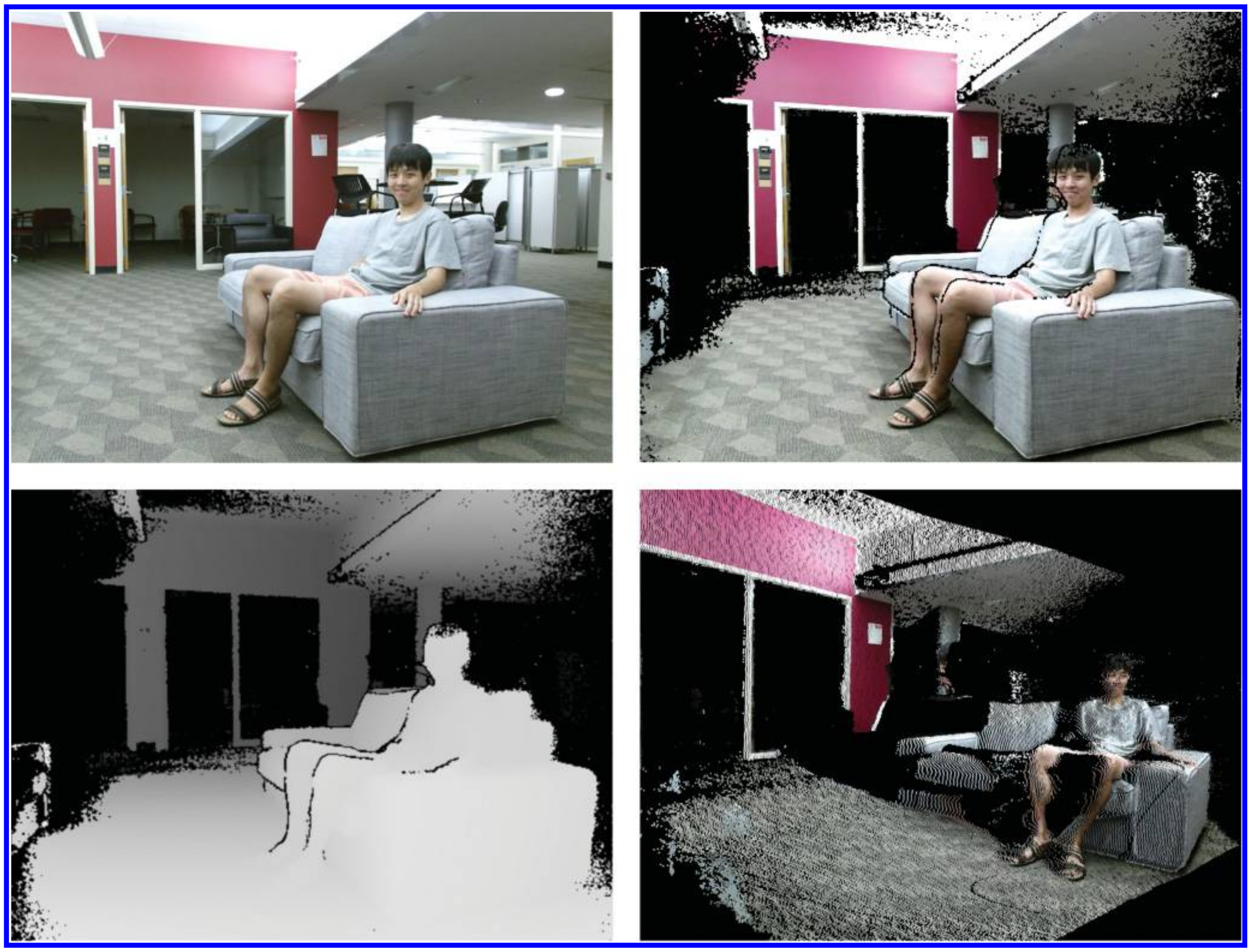

Figure 2. *An example of our rendering technique. The top-left and the bottom-left are the color and depth pixels from a Kinect v2. The top-right is the rendered scene captured from a front-view, and the bottom-right is the rendered scene captured from a side-view.*

colored based on the YUV textures. Figure 2 is an example of our rendering technique.

Our goal in publishing the code for this system is to allow others to use it as a building block for ongoing development of applications for AR headsets, in particular, telepresence systems. While the quality of the rendering is far from perfect, it is our hope that the system will help developers, researchers, and consumers by providing a verified method that is portable—it does not require a machine for rendering—and works on hardware that is readily available for consumers.

Source Code

The source code and the additional instructions for this system is available at https://github.com/hanseuljun/kinect-to-hololens .

References

Fleisher, O. (2018). How Vimeo can power live streaming holograms. Retrieved from https://vimeo.com/blog/post/how-vimeo-can-power-live-streaming-holograms

Kowalski, M., Naruniec, J., & Daniluk, M. (2015). Livescan3d: A fast and inexpensive 3d data acquisition system for multiple Kinect v2 sensors. *2015 International Conference on 3D Vision*, 318–325.

Maimone, A., Yang, X., Dierk, N., State, A., Dou, M., & Fuchs, H. (2013). General-purpose telepresence with head-worn optical see-through displays and projector-based lighting. *2013 IEEE Virtual Reality*, 23–26.

Orts-Escolano, S., Rhemann, C., Fanello, S., Chang, W., Kowdle, A., Degtyarev, Y., et al. (2016, October). Holoportation: Virtual 3D teleportation in real-time. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 741–754.

Valorem Reply (2016). *HoloBeam Tech* [Computer Software]. Retrieved from https://www.microsoft.com/en-us/p/holobeam-tech/9nblggh555zf

Wei, S., Saragih, J., Simon, T., Harley, A. W., Lombardi, S., Perdoch, M., Hypes, A., Wang, D., Badino, H., & Sheikh, Y. (2019). VR facial animation via multiview image translation. *ACM SIGGRAPH 2019*.

Wilson, A. D. (2017). Fast lossless depth image compression. *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces*, 100–105.