Capturing a snapshot in VR space

Daiki *Yamada*[*] and Michael *Cohen*[**]

Computer Arts Lab, University of Aizu; Aizu-Wakamatsu, Fukushima; Japan

Abstract. We propose a new way to capture snapshots in virtual space. We created a demonstration of how this method could be incorporated into other VR (virtual reality) contents. Using an Oculus Quest HMD (head-mounted display), we created VR utility that runs on game engine Unity. We used bimanual controllers to take snapshots. The area framed by the bimanual controllers can be cropped and saved as a PNG image. It also has the functions to zoom in and out of the camera using the position of the player's HMD, to capture non-horizontal images, to dynamically apply graphical effects, and to create stereoscopic imagery. This system is a more intuitive and fun way to capture virtual snapshots.

1 Introduction

In recent years, many VR applications have been developed, ranging from job training [1] to entertainment [2]. VR contents that are mainly played for entertainment or simulations can often save the playing screen as a screenshot. However, for most applications, such functionality is not considered important. This project introduces a new virtual camera capturing function in VR environments. We developed a system that can capture a snapshot of a specified area in virtual space. This functionality can be deployed in many other VR applications. Unity was used for the development. In [3], which is one of the studies aimed at developing VR content, a type of game known as adventure games is described. Our project is most useful for this kind of exploration game, since the player moves around and explores a virtual space. On the other hand, it would not work well in an action game where players are constantly moving both hands, such as that described in [2]. This project is more effective for experiencing exploratory games and video content such as adventure games.

This study aims to capture trimmed snapshots in virtual space. Specifically, users express a rectangle with bimanual controllers and capture the area bounded by the rectangle. This is a new method of taking snapshots in a virtual space to provide more immersive and enjoyable experiences.

2 Method

We developed functionality that allows VR users to take snapshots in a new way while experiencing immersive content. Specifically, we allow players to move around a field created in virtual space and execute a new method of taking snapshots. The snapshot capture uses bimanual controllers. When the virtual camera is ready to shoot while experiencing VR content, a rectangle created based on the positions of the bimanual controllers appears. As seen in Fig.1, the aspect ratio of the rectangle can be transformed (into vertical or horizontal shape) by moving the position of the controllers in both hands. The user can see the image for the camera from the image in the area framed by the rectangle. When the user presses the shutter button on the right controller, the subjective view enclosed by the rectangle is automatically extracted and saved as a snapshot. One can also use other buttons or move one's head to apply effects to the snapshot, to zoom in, and to create stereoscopic images.

[*] yamituki.mimituki2023@gmail.com

[**] mcohen@u-aizu.ac.jp

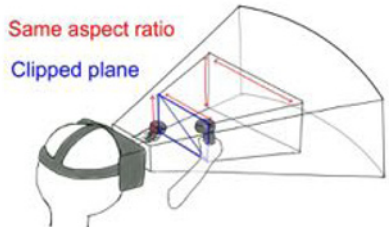

Figure 1: Simplified image of this system

Since this system is intended to be incorporated into other VR content, we used simple VR scene as a testbed. Using an environmental asset [4], an immersive field can be entered, around which a player can freely move. We added a function for taking snapshots to it, so that it can take snapshots in the way proposed in this study. The following explains the details of the image capture function.

The implementation process of this project is divided into four main parts: 1) drawing the frame of a rectangular area, 2) adjusting images in the framed area, 3) outputting the image, and 4) performing additional functions.

2.1 Drawing the frame of a rectangular area

First, we obtain the spatial coordinates of both hand-held controllers. Next, we create an empty object that has no mesh with the x and z coordinates of the right controller and the y coordinate of the left controller. Similarly, we create an empty object with the x, z coordinates of the left controller and the y coordinates of the right controller. By creating the above two objects, preparations for drawing the border are complete. To draw the lines, we used a Unity component called `LineRenderer` [5]. We draw the rectangle through a total of four objects: two controllers and two created empty objects, as shown in Fig. 2. In this case, we draw the frame clockwise from the right controller.

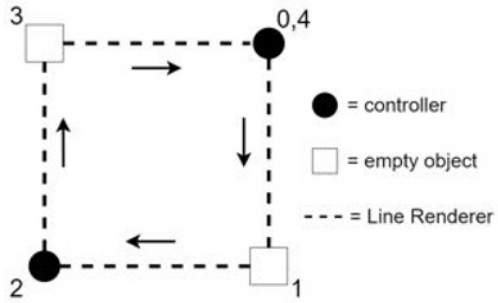

Figure 2: A rectangular area drawn by `LineRenderer` through four objects.

2.2 Adjusting images in the area

This step is divided into four more parts: 1) calculating resolution of the images in the area, 2) creating a frame, 3) calculating camera orientation, and 4) projecting camera images onto a plane.

2.2.1 Calculating resolution of images in the area

First, we calculate the aspect ratio of the rectangle from the distances of the objects at the four corners of the rectangular area created, as described above. Next, we calculate how large the currently expanded rectangle area is compared to the maximum value of each side of the rectangle area as set in advance. The maximum value of the sides of the rectangular area is based on the length of a human's arms when extended. The resolution of the image in the area is limited to no more than 500 pixels on each side. Finally, we obtain the height and width of the image and set the resolution of the video in the area based on the percentage calculated above. However, to dynamically display the image from the camera in an area of any size without distortion, it is necessary to render every frame, which can be very heavy if executed as is. To avoid this problem, resolution of the image in that area is reduced to half its original resolution.

2.2.2 Creating a frame

In this part, we describe creating a rectangle for projecting the image from the camera. The rectangular area created by the bimanual controllers is constantly deforming, so a quad ("quadrilateral") [6] is created every frame. In Unity, we create two coplanar triangles with a total of four vertices, which together can be considered as one quad. To create a specific rectangular frame, we first store the coordinates of the four points that are the vertices of the two triangles in an array. In this case, the vertices must be specified in the following ("rising zig-zag") order: lower left, lower right, upper left, upper right, as seen in the left of Fig. 3. Next, we determine where the four specified vertices are used as vertices of each triangle and store their numbers in a single array.

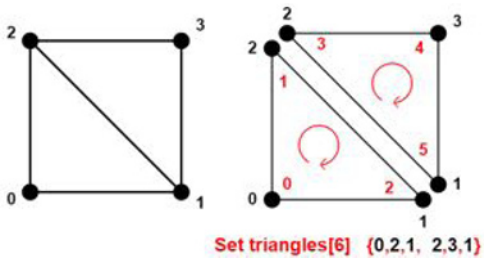

Figure 3: The quad is made up of two right triangles.

To ensure that the quad is displayed correctly from the player's point of view (subjective perspective), we set the normals of each vertex to the negative z-axis orientation on the quad's local coordinates in Unity. Finally, to display the texture correctly on the quad, the texture coordinates are established. Since texture coordinates are expressed in normalized scale as extending 0 to 1, we can display the texture for the entire quad object by setting the lower left vertex to (0,0), the lower right vertex to (1,0), the upper left vertex to (0,1), and the upper right vertex to (1,1), as seen in the right of Fig. 3.

2.2.3 Calculating the camera orientation

Orientation of the camera must always be perpendicular to the rectangular area created by the bimanual controllers. For this purpose, we need to calculate the vectors on the quad. We calculate the vector from the upper left vertex to the upper right vertex of the rectangular area. In addition, we calculate the vector from the lower left vertex to the lower right vertex of the rectangular area. By calculating the outer ("cross") product of the two vectors thus calculated, we can determine a vector that is perpendicular to the quad. The camera orientation is kept perpendicular to the quad by aligning the front direction of the camera with that vector.

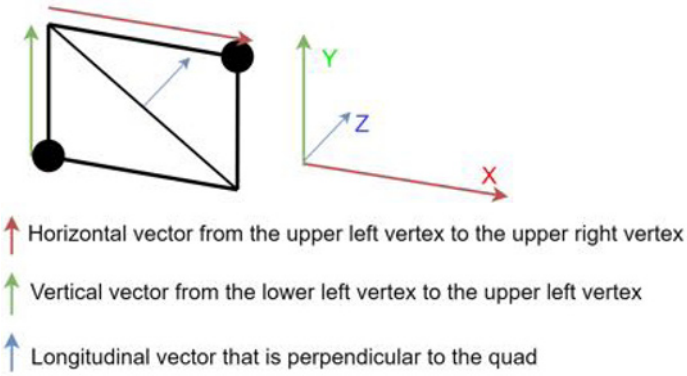

Figure 4: Calculated vectors (Unity uses left-handed coordinate system)

2.2.4 Projecting the camera images onto a plane

In this part, we create a material that projects the camera's image and apply it to the quad. The following steps are required. First, we create a small `RenderTexture` [7] with the same aspect ratio as the rectangular area. `RenderTexture` is a texture that can be rendered. It temporarily renders the image from the virtual camera. Normally, a square `RenderTexture` is desirable, but, if the height and width are not equal, as in this project, it will not work properly. The camera parameters are

changed to adjust to the aspect ratio of the `RenderTexture`, causing the image to be distorted and the field of view to change. There is no way to avoid this, so we adjust the camera parameters to project the correct image by changing them depending on the situation. The corrected image is projected by adjusting the field of view according to the change in height. When `RenderTexture` is applied to the quad's material, the image from the camera is projected onto the quad. By going through the above procedure, we can see the image as if the space has been cut out to fit a rectangular area that can be expanded vertically or horizontally.

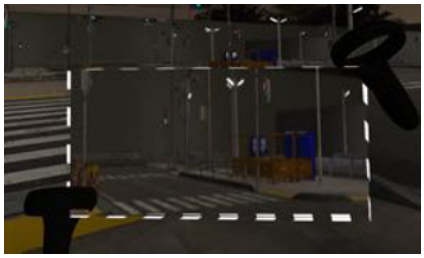

Figure 5 (left): Example of deforming an area horizontally
Figure 6 (right): Example of deforming an area vertically

2.3 Outputting the image

To output the camera image as a PNG image, the following steps are required. First, we create a large `RenderTexture` with the same aspect ratio as the rectangle area. There is no problem if the size of the `RenderTexture` is large because we only need to render once to save the image. Next, we copy the pixel data of that `RenderTexture` to a normal texture of the same size. Finally, we convert the texture to a PNG image and save it with a name we determined in advance.

These are the basic functions of this system. This system works by using the position data of three objects, two bimanual controllers and the HMD that the user is wearing. The data from the HMD is used to control the field of view, which parameterizes an additional feature. In this study, we have not set any limits on the location or orientation of those affordances, so we can move them freely in virtual space, with a total of 18 degrees of freedom (3 translational and 3 rotational for each of 3 objects). In addition, the following supplemental functions broaden the range of capture methods.

2.4 Performing other additional functions

There are several additional functions provided in this system: 1) Adjusting the camera using the player's HMD, 2) creating stereoscopic images, and 3) post-processing.

2.4.1 Controlling the camera using player's HMD

This function allows the player to control the capture using the player's HMD along with the bimanual controllers. The controls available include zooming and controlling the angle of the camera. The camera can be moved closer or further away depending on the distance between the HMD and the rectangular area framed by the bimanual controllers. When the user moves his or her head closer, the camera image is zoomed in, and when the head is moved away, the camera image is zoomed out. In addition, while this function is active, the camera angle is aligned with the angle of the player's HMD.

Normally, the user can only control the camera angle yawing. This head-tracking functionality allows the player to also control the camera angle rolling and pitching. An example of such framing can be seen in Fig.15. These two functions are activated by pressing the index trigger button on the left controller, and capture can be executed as usual.

2.4.2 Stereoscopic image

This function allows saving a captured picture as a stereoscopic image by outputting the image seen by the left eye and the image seen by the right eye to create a stereoscopic image pair. Two cameras were displaced from the position of the normal "cyclopean" camera — the camera for the right eye was displaced to the right and the camera for the left eye was moved to the left — to enable binocular parallax and stereopsis. This function can also be activated by a button. Pressing the X button

(lower thumb button on left controller) toggles between normal capture and stereo image creation modes. The frame line turns red while the stereo image creation function is activated.

2.4.3 Post-processing

Unity has a post-processing [8] function that allows application of filters and effects to a camera image before it is displayed. By adding post-processing-related components to the camera object, various effects can be used. In this study, we used color filters to process camera images. Each press of the B (upper thumb button on right controller) switches the effect. There are four different color effects currently available, as shown in Figs. 7–10.

Figure 7: Effect type 1 (“Grayscale”)

Figure 8: Effect type 2 (“Greenish”)

Figure 9: Effect type 3 (“Shift the hue”)

Figure 10: Effect type 4 (“Brighten the tint and increase the temperature”)

The player can see in real-time which effects are currently being applied. This function is highly extensible as additional effects can be added by straight-forward development.

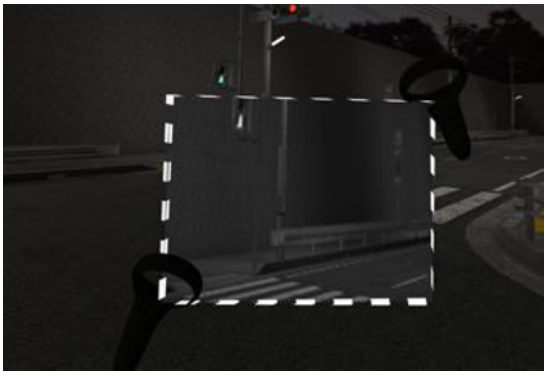

Figure 11: Only the enclosed area is grayscale

These three functions are orthogonal: they do not interfere with each other, and can be used together. For instance, one could capture a hue-shifted, rolled, stereogram.

3 Result

We could freely frame output images to any size and aspect ratio desired. Below in Figs. 12–15 shows some examples of captured images.

Figure 12: A very wide image

Figure 13: A very narrow image

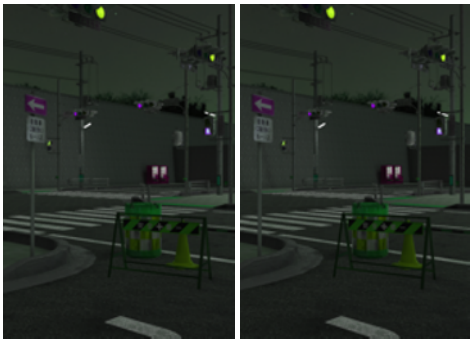

Figure 14: Stereoscopic image pair

Figure 15: A rolled “dutch angle” image

As originally intended, a system for taking snapshots by trimming out only a specific area was completed. More features that can be added in the future include the availability of a wider variety of effects and support for hand tracking.

4 Conclusion

In this study, we developed a system that can perform a new method of capturing images that works in VR using bimanual controllers and an HMD. If other VR contents implement the capture function, this system could be integrated to foster richer experience. Images within the rectangular area that the player sees while playing are displayed at a lower resolution than the actual output image, so they are less likely to degrade performance when incorporated into other VR content. In addition, this system can be expanded to include additional functions, so we can pursue further experiences.

5 References

1. Minhua Ma, Yeshwanth Pulijala, and Ashraf Ayoub. Oculus surgery–an Application of Oculus Rift and Stereoscopic 3D videos in training maxillofacial surgeons. (2017): 187-202. http://eprints.staffs.ac.uk/id/eprint/3587
2. Christopher Todd. Game Development in Unity Using Oculus Quest VR. (2020) https://www.cs.csustan.edu/~dkim/files/college_poster_session/2020/game_development_report.pdf
3. Jeanny Pragantha and Darius Andana Haris. Adventure Game "Detective Adventure" Using Unity Virtual Reality. *IOP Conference Series: Materials Science and Engineering*. Vol. 1007. No. 1. IOP Publishing, 2020.
4. 【3Dモデル】日本の道路パック / Japanese Street Pack【V2 Updated】 2020-11-27. https://booth.pm/ja/items/2442389
5. Unity Technologies. "Line Renderer". Unity User Manual 2019.4 (LTS). 2022-01-15. https://docs.unity3d.com/2019.4/Documentation/Manual/class-LineRenderer.html,
6. Unity Technologies. "Example - Creating a quad". Unity User Manual 2019.4 (LTS). 2022-01-15. https://docs.unity3d.com/2019.4/Documentation/Manual/Example-CreatingaBillboardPlane.html
7. Unity Technologies. "Render Texture". Unity User Manual 2019.4 (LTS). 2022-01-15. https://docs.unity3d.com/2019.4/Documentation/Manual/class-RenderTexture.html
8. Unity Technologies. "Post-processing". Unity User Manual 2019.4 (LTS). 2022-01-15. https://docs.unity3d.com/2019.4/Documentation/Manual/PostProcessingOverview.html