Democratizing Rendering for Multiple Viewers in Surround VR Systems

Jürgen P. Schulze[1], Daniel Acevedo[2], John Mangan[1], Andrew Prudhomme[1], Phi Nguyen[1], Philip Weber[1]
[1] University of California San Diego, La Jolla, CA, USA
[2] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

ABSTRACT

We present a new approach for how multiple users' views can be rendered in a surround virtual environment without using special multi-view hardware. It is based on the idea that different parts of the screen are often viewed by different users, so that they can be rendered from their own view point, or at least from a point closer to their view point than traditionally expected. The vast majority of 3D virtual reality systems are designed for one head-tracked user, and a number of passive viewers. Only the head tracked user gets to see the correct view of the scene, everybody else sees a distorted image. We reduce this problem by algorithmically democratizing the rendering view point among all tracked users. Researchers have proposed solutions for multiple tracked users, but most of them require major changes to the display hardware of the VR system, such as additional projectors or custom VR glasses. Our approach does not require additional hardware, except the ability to track each participating user. We propose three versions of our multi-viewer algorithm. Each of them balances image distortion and frame rate in different ways, making them more or less suitable for certain application scenarios. Our most sophisticated algorithm renders each pixel from its own, optimized camera perspective, which depends on all tracked users' head positions and orientations.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

1 INTRODUCTION

The vast majority of existing immersive virtual reality (VR) systems use a tracking system to track one user's head, as well as a hand-held 3D input device. The images on the screens are computed for the tracked user's eye positions, and the input device (wand) is used to interact with the 3D environment. Larger VR systems, such as CAVEs, were designed to be used by more than one user, and are often used by groups of two to 10 users in a variety of scenarios, ranging from scientific meetings of experts to group demonstrations. If more than one user are present in a VR environment which only tracks one, the non-tracked (passive) users see the images on the screen rendered from the perspective of the tracked user. Depending on the geometry of the VR system, the type of data being viewed, and most of all the positions of tracked and non-tracked users, the visual quality of the rendered image will range from acceptable to very distorted. The distortion comes from the fact that the 3D world gets projected onto the display surfaces, and a key parameter for this projection is the center of it, which is determined by the viewer position or, more precisely, the user's eye position. Depending on the relative positions and orientations of the users, the distortions can include bent lines between non-coplanar screens, pseudo-stereo (inverted left and right eye images), or incorrect size perception. Besides image distortion, another problem is that when users point at features in the data set with their fingers, the location pointed at is generally not the same for the other users.

Popular workarounds to these problems include passing the head tracker around the group so that everybody gets to see the correct view at some point. Pointing works correctly for everybody when done with a virtual pointer controlled by the wand. And in certain scenarios, special rendering techniques can eliminate the pseudo-stereo effect. Solutions have been presented which render completely distinct images for two or more users, but they require specialized hardware.

Our approach does not require additional display or viewing hardware, and there is no theoretical limit to the number of tracked users, except that with more users each user's view tends to get more distorted. The goal of our method is to make the images for multiple viewers more perspectively correct than it is traditionally the case. We require that each user be head-tracked. There are a variety of tracking technologies, such as electro-magnetic, ultrasonic, mechanical, optical, and combinations of the above. While typically only two points in space are tracked by VR systems (head and hand), most systems can be extended to track additional points. With many optical tracking systems it is particularly easy and inexpensive to add additional tracker targets. Each of them reports a precise position and orientation to the rendering system.

We present three implementations of our algorithm, all of which are based on the same basic idea of using different camera positions when rendering different parts of the screens, depending on what users look at. The algorithms have different performance characteristics and work differently with different types of data. Our algorithms have in common that they work best in VR systems fully surrounding the users, or at least cover 180 degrees.

The three implementations we present are all based on the same idea, and are designed to work around various hardware limitations in different ways. Our most sophisticated, but also most computationally expensive algorithm recalculates the optimum camera position for each pixel, depending on positions and orientations of all tracked users.

In the following sections, we are going to present prior work done in this field, describe our three multi-viewer algorithms, present and discuss visual and performance results, and finally conclude with pointers to future work.

2 RELATED WORK

Previously published approaches for multi-viewer systems can be distinguished by how many viewers an approach scales to, what usage scenarios are supported, and how specialized the required hardware is.

The most closely related work was done by Jonathan Marbach. He introduced the notion of rendering different users' views in separate parts of the screen, based on their position and head orientation [4]. These views are blended together in a blend area between them. When multiple users look at nearby points, their viewpoints are averaged and one image is rendered for those users. In subsequent work, Marbach applies geometry shaders to improve the rendering rate [3], and runs an extensive user study to compare various multi-viewer approaches [5]. Our approach differs from Marbach's in that we don't just render one image per user, but gradually vary the viewpoint across the screen, causing a continuous transition between the users' views. Also, we distinguish users' views not only

along the horizontal axis, but also the vertical axis.

Earlier approaches to support multiple viewers were often based on the idea of rendering completely distinct views for each user, utilizing specialized or modified hardware. Agrawla et al. [1] built a two-viewer display table which time slices the required four views (two eyes each for two users) and utilizes modified shutter glasses to generate images at 36Hz per eye and per user.

Froehlich et al. [2] further improve on the shutter glasses approach by adding polarization filters, which doubles the number of users supported at a given shuttering rate.

McGinity et al. [6] created a cylindrical VR theater which is 10m in diameter and fully immerses up to 20 users at once. It uses Omnistereo with discretized viewing strips: 6-8 views per eye and projector. Because the system uses the Omnistereo approach it does not require head tracking, and hence sacrifices the ability for individual users to look around objects in the 3D world.

To our knowledge, the idea of rendering each pixel from a different, optimized view point, has not previously been published. Our approach does not require specialized display or viewing hardware, and is thus a step towards a solution to a common problem with almost all VR systems.

3 Approach

Our new multi-viewer rendering approach is designed for a typical immersive VR system with stereo displays surrounding the user, with the ability to head track every user. Many VR systems which currently track only one user can easily and inexpensively be extended to track multiple users by adding additional tracking targets or sensors.

The issue we address with our approach is that in order to render a correct image in a VR environment, each screen has to be rendered with the same camera and projection parameters. The camera parameters are typically obtained by tracking one user who is head-tracked. This user's head position is used to calculate the eye positions, which is done by adding an offset from the head position for each eye. This concept generates a perspectively correct image for the head-tracked user only. All other users will see the generated imagery from the head-tracked user's perspective, which leads to artifacts such as distortion and bent lines for the non-tracked users. This distortion gets worse, the more distant the users are from the head-tracked user, and the more their head orientation differs from this user.

Figure 1 illustrates our approach. Two people use a surround virtual environment. Each user has a position and a head orientation. For practical reasons we consider viewing direction to be identical to head orientation (hardly any VR systems use eye tracking). Our approach aims at rendering the point on the screen a user looks at (look-at point) from that user's eye point whenever possible, while in a user's peripheral view it is acceptable to have some degree of image distortion. The algorithm allows for greater distortion at greater distance from the look-at point. This distortion is caused by the camera location used to render a certain pixel not being co-located with the user's eye point.

Our analogy for the calculation of where to place the camera point with respect to the users is a scenario where each user carries a flashlight on their heads, pointing along their viewing direction. Each user's flashlight uses a different color, but has the same radial brightness fall-off from the center point. Our algorithm calculates the equivalent of what light intensity each user's flash light has at a given point P_s on the screen. The center point of the flash light's cone have a value of 1, while areas unlit by the flashlight have a value of 0, all other points within the flash light's cone of light will have values between 0 and 1. Then each user's brightness contribution is divided by the sum of all users' brightness contributions at point P_s. The resulting value is interpreted as a generalized barycentric coordinate to determine a position between all users' head positions.

For example: in a two-user scenario such as that shown in Figure 1, let C_1 be user 1's contribution to the camera position, and C_2 user 2's. Let a and b be user 1 and user 2's respective flashlight brightness values at point P_s. Then $C_1 = a/(a+b)$ and $C_2 = b/(a+b)$. The camera position P_c will be somewhere on the line connecting user 1 and 2's head positions P_1 and P_2. P_c is the location from which P_s will be rendered. P_c is calculated as $P_c = P_1 + C_2 * (P_2 - P_1)$.

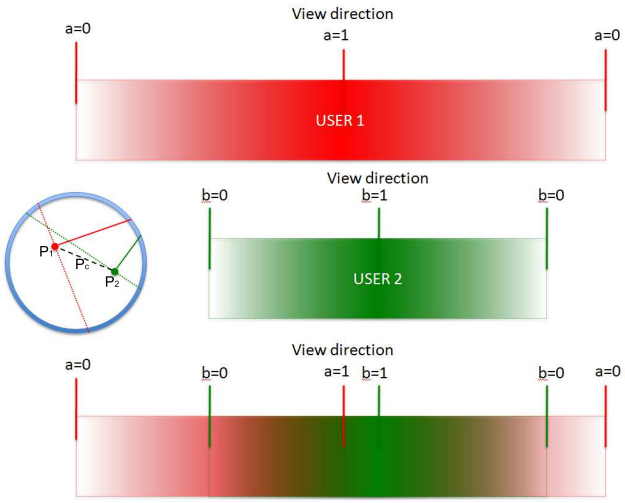

Figure 1: The area on the screens each user looks at is colored in red and green, respectively. The screen has been unwrapped, and is shown from the perspective of the users. The topmost line shows user 1 (red), the middle line shows user 2 (green). The bottom line shows the blended flashlight contributions. On the left is a top-down view of the viewer positions and directions.

Our approach is independent of the number of head-tracked users. It does, however, produce better results with fewer users, as each user has, on average, more screen area available for a good camera perspective.

4 New Algorithms

We created three implementations of the above described algorithm. All of them scale, in theory, to as many users as can be head-tracked, but we only implemented them for two users. The three implementations vary in the granularity of the size of the screen area rendered by the same camera position.

4.1 Dynamic Zoning

The first and simplest implementation we created subdivides the screen surface into a fixed number of zones, based on which viewer looks at them more directly. We created two versions of the algorithm: one in which the zones have vertical boundaries, and another in which the boundaries tilt with the user's head. The zone boundaries are calculated with respect to each user's viewing direction. A parameter for the algorithm is the horizontal field of view angle, which determines the area on the screen a user looks at, the core viewing area. If the core viewing areas of the users do not overlap, each user sees a perspectively correct image within the entire core viewing area. If the areas do overlap, the overlapping area is rendered from a camera position exactly in the center of the users. For two users, this means that four viewing areas exist: the two core viewing areas of the users which do not overlap, the overlapping area, and the area outside of both users' core viewing areas.

This approach is similar to Marbach's, only that we render directly to the screen using OpenGL's stencil buffer, and that we do not blend the views at their boundaries. We also add the ability to tilt the edges between the users' images, depending on their head tilt.

Our algorithm consists of two steps: first, for each viewer the stencil buffer is set up. In order to do this, the viewer's core view frustum is being intersected with the screen. Then, after setting the stencil mask, the area covered by the core view frustum is rendered to initialize the stencil buffer. Each bit in the stencil buffer represents a unique viewer (which limits this algorithm to the number of bits in the stencil buffer). After initialization, stencil buffer pixels with more than one set bit have more than one viewer looking at it.

After setting up the stencil buffer, each user's view is rendered from its correct eye position with the assigned stencil bit set, so that it renders only into those pixels which are part of the user's core viewing area.

The final step is to render the area in which the users' core viewing areas fully or partially overlap. For rendering this area, the camera is set to the point half-way between the eyes (for two users). Figure 2 illustrates the Dynamic Zoning concept for two users.

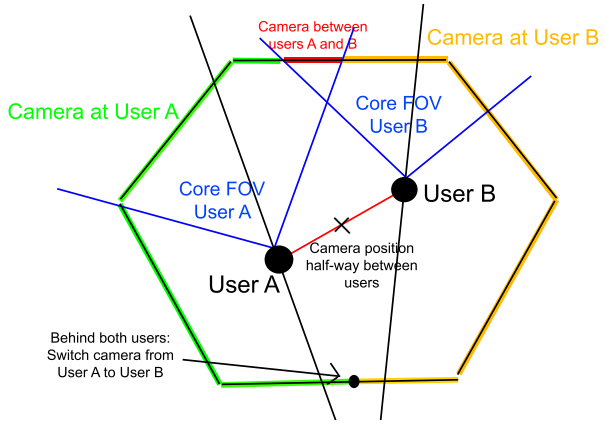

Figure 2: Our Dynamic Zoning algorithm for two users, A and B.

4.2 Dynamic Tiling

Our dynamic tiling implementation subdivides each screen into a regular 2D grid of equally sized zones. Each grid cell is rendered from one camera location, calculated by a weighted average between the users (weighted based off of how directly they are looking at the zone). More grid cells generate smoother images but take longer to render. This algorithm has the following options:

- Auto Adjust mode: adjusts the grid size dynamically to achieve a target frame rate range.
- 3D or 2D orientation calculation: allows the contribution of each user to a zone to be determined either in 3D vector calculations, or only in 2D, by ignoring any vertical angle.
- Orientation/contribution cutoff angle: affects the angle of the "view cone", starting at the user and projected out towards the screen. In general, any zone directly in front of the user will give that user a weight of 1 in contribution functions, and as a zone approaches the cutoff angle, the weight will approach 0.

We implemented three different functions to calculate the contribution of each user to the camera position for a given part of the screen. Each uses an algorithm based on two angles. The first is the contribution angle created by the vector of the user's viewing orientation (from the viewer's head position) and the vector from the viewer (eye position) to the center of the zone currently being calculated. The second angle is the orientation cutoff angle defined by the user (always greater than 0). For all three functions, the final weights are divided by the total of the weight for all users, creating a total sum of 1.0. For instance, if one user has a weight of .75 and the other has a weight of 0.5, the final weights are adjusted so that the first user has .6 and the second user has a weight of .4. The following three functions determine the pre-adjusted orientation contribution, or weight values per user:

- Linear: $weight = 1 - (contribution\_angle / cutoff\_angle)$
- Cosine: $weight = cos(contribution\_angle / cutoff\_angle)$. This function achieves a good balance between efficiency and effect.
- Gaussian: The weight is given by a cumulative distribution function (CDF) of a normal Gaussian distribution (non-standard) interpolated by the $orientation\_angle$ (mean of the distribution would be hit when $orientation\_angle = 0$, implying the zone center is directly in the path of the viewer's orientation, and the standard deviation of the distribution is given by the user defined cutoff angle divided by 3, thus approaching the cutoff angle will result in a value just slightly greater than 0, although being 0 or less is impossible). This algorithm is the most complex of the three, but it does prevent "hard edges" in a viewer's view spectrum, since they have some contribution on every zone (even those directly behind them, due to the Gaussian distribution).

Figure 3: Dynamic Tiling algorithm, using extra large tile size.

Figure 3 shows the Dynamic Tiling algorithm with a protein data set and extra large tiles to illustrate how the algorithm works.

4.3 Per Pixel Cameras

Our most sophisticated implementation, which generates the most accurate result for our algorithm, albeit at the cost of rendering performance, is based on GLSL shaders and renders the interpolated view for each pixel. The algorithm renders each triangle through a shader program consisting of a vertex, geometry, and fragment shader.

In the vertex shader, the input model-view matrix is set to transform the vertices to world space. The world space points are passed on with their associated normals, colors and texture coordinates.

In the geometry shader, the view and projection matrices for each viewer are present in four of the texture matrices. The triangle points are transformed through the view and projection matrices from the viewers. Lighting is calculated for each point separately. A number of triangles are emitted by the shader to cover all the

possible screen fragments between the triangle for the two viewers. We support two methods: a brute force method with 7 emitted triangles, and one with the minimal triangle output of 1 through 4 triangles. We did not find a noticeable performance gain with the second method so we use the first method by default. Per vertex color, world space position, texture coordinates and diffuse lighting multipliers are passed through as flat output values.

The fragment shader first computes the rendered pixel's screen position in world space. Then it uses uniform viewer positions and directions to find the pixel's weight. Then it finds the line segment from the near to far planes that intersects the fragment for the interpolated viewer position in world space. After that, the algorithm determines if this segment intersects the fragment's triangle using the world space points passed from the geometry shader. This process also results in the barycentric coordinates for the fragment, which are used to determine color and lighting based on values from the geometry shader, as well as the texture coordinates if applicable. The depth value of the fragment is set to its true depth value. This approach is computationally intensive, but the GPU pipeline is still much faster than a CPU implementation could be. The frame rate varies depending on how complex the geometry is and how many fragments are being processed.

5 Results

Figures 4 and 5 illustrate the Dynamic Zoning approach, showing how the camera positions are selected for different viewer positions. On the left are photographs of two users in our VR environment, the images to the right show top-down views of the users' locations within the VR environment and their head directions.

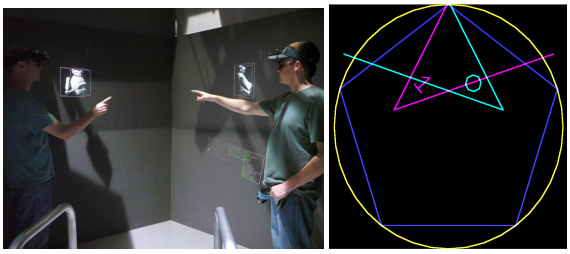

Figure 4: Two users looking in different directions: each user gets their own, perspectively correct view of the data set.

Figure 5: Two users looking in the same direction: scene rendered from midpoint between users' head positions.

The rendering rates for our Dynamic Zoning algorithm are close to those of the standard case of a single head-tracked user. Dynamic Tiling adjusts the zoning automatically depending on what target frame rate the user selects. As an example: rendering a 100k polygon data set with 400 zones per screen happens at interactive frame rates of about 20 frames per second (fps), using Nvidia Quadro 5600 graphics cards. The Per Pixel approach is the slowest at approximately 5 fps for a 100k polygon data set.

6 Discussion

In comparison, each of our three algorithms has advantages and disadvantages. Dynamic Zoning renders the fastest with little impact on rendering performance compared to a single user scenario. It also works the best when the users are not looking at the same screen area, because in this case they each get their own dedicated screen surfaces, and the images get rendered without distortion.

If the users' viewing directions overlap then the other two algorithms work better because they do not cut the views off at a certain angle, but instead create a smoother transition between core and peripheral fields of view. The per pixel algorithm is the ultimate solution for the highest image quality, but until the graphics hardware for shader calculations are fast enough for it to run at true real-time frame rates of 20 fps and up, our Dynamic Tiling algorithm is a better compromise between real-time frame rates and image quality.

7 Conclusion and Future Work

We presented a new approach for how multiple users' views can be rendered in a surround virtual environment without using special multi-view hardware. We discussed three different implementations of the approach which achieve the same goal, each with their own advantages over the other two. Our multi-viewer solution works better for scenarios with small data sets (in terms of size on the screen), but does not work as well for architectural models and similar data sets which surround the user and have large amounts of lines and rectangular structures in them, because the distortion aspect of our approach can be confusing.

Although our algorithms are not limited in the number of supported viewers, in practice the number of tracking targets supported and the size of the VR environment limit the number of users. We would expect that three users would still benefit from our multi-viewer approach in a surround environment.

Finally, we would like to conduct a formal user study to evaluate the effectiveness of our multi-viewer approach.

Acknowledgements

This publication is based in part on work supported by Award No. US 2008-107, made by King Abdullah University of Science and Technology (KAUST).

References

[1] M. Agrawala, A. C. Beers, I. McDowall, B. Fröhlich, M. Bolas, and P. Hanrahan. The two-user responsive workbench: Support for collaboration through individual views of a shared space. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 327–332, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[2] B. Frhlich, R. Blach, O. Stefani, J. Hochstrate, J. Hoffmann, K. Klger, and M. Bues. Implementing multi-viewer stereo displays. In *WSCG (Full Papers)'05*, pages 139–146, 2005.

[3] J. Marbach. Gpu acceleration of stereoscopic and multi-view rendering for virtual reality applications. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, VRST '09, pages 103–110, New York, NY, USA, 2009. ACM.

[4] J. Marbach. Image blending and view clustering for multi-viewer immersive projection environments. In *Proceedings of the 2009 IEEE Virtual Reality Conference*, pages 51–54, Washington, DC, USA, 2009. IEEE Computer Society.

[5] J. Marbach. Supporting multiple users in single-stereo-pair immersive virtual reality environments. In *Doctoral Dissertation*, page 104. University of Colorado, 2010.

[6] M. McGinity, J. Shaw, V. Kuchelmeister, A. Hardjono, and D. D. Favero. Avie: A versatile multi-user stereo 360deg interactive vr theatre. In *Proceedings of the 2007 Workshop on Emerging Display Technologies*, EDT '07, New York, NY, USA, 2007. ACM.