Immersive OSPRay: Enabling VR Experiences with OSPRay

JUNG WHO NAM, GREGORY D. ABRAM, FRANCESCA SAMSEL, and PAUL A. NAVRÁTIL,
Texas Advanced Computing Center (TACC), the University of Texas at Austin, USA

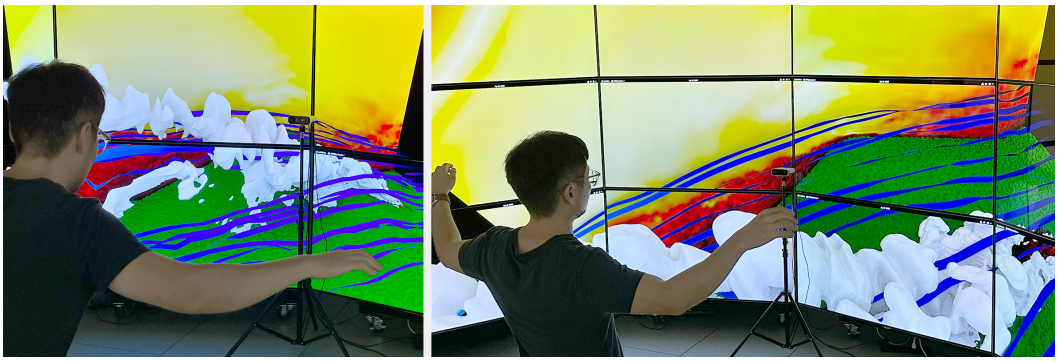

Fig. 1. Gesture is used to fly through a visualization of HIGRAD/FIRETEC simulation model data [1] rendered by an open-source, real-time ray tracing application running on a cluster of eighteen displays.

We present an extension to OSPRay Studio, Intel's open-source ray tracing application, to support immersive virtual reality experiences in public exhibition settings. This extension enables one to display a single, coherent 3D virtual environment on tiled display walls and use gesture-based interaction techniques to navigate the environment. Additional mechanisms are provided to configure the application for different display arrangements and integrate other motion-tracking technologies.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; **Systems and tools for interaction design**; • **Computing methodologies** → **Virtual reality**; **Ray tracing**.

Additional Key Words and Phrases: Interactive Installations, Gesture-based Interaction, Tiled Displays, Ray Tracing, Virtual Reality

ACM Reference Format:
Jung Who Nam, Gregory D. Abram, Francesca Samsel, and Paul A. Navrátil. 2023. Immersive OSPRay: Enabling VR Experiences with OSPRay. In . ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recent advances in graphics hardware have opened the door to move ray tracing rendering techniques previously used in non-interactive mediums, e.g., films, into applications that benefit from interactivity, e.g., data visualizations and games. Hardware ray tracing acceleration has been added to the current generation of GPUs [8, 12] and hardware vendors have released optimized high- and low-level libraries for ray tracing operations [13, 17, 18]. Ray tracing is now incorporated

PEARC '23, July 23–27, 2023, Portland, OR

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

into popular visualization toolkits [2, 15, 16, 20] and it is part of the new Khronos Group standard for analytical rendering [7].

We are interested in exploring the potential of these new, widely accessible rendering techniques for public exhibitions. Inspired by CAVE VR systems [3, 4] that provide immersive experiences with minimal body-worn devices, we introduce a system that extends an open-source ray tracing application [14, 17] to display a single, coherent 3D virtual environment on tiled display walls and support gesture-based interaction techniques (Figure 1).

The system is further extended to account for possible constraints imposed in public installation spaces. To adapt to different display configurations, we provide an interactive tool with graphical assistance for setting displays and their arrangements in a 3D space. To integrate a wide range of sensing technologies, the part that handles gesture-based interaction is implemented as a separate server application and a plugin to the rendering application to decouple the part from the original application. Finally, containerized using Docker and Singularity/Apptainer [6, 11] the system can be deployed in exhibition platforms with pre-existing demos that might require different dependencies.

This paper contributes an example of a real-time, interactive ray tracing application used in interactive installation settings, which may indicate the recent advances in graphics hardware and hint at incoming paradigm shifts in real-world computer graphics applications. Additionally, this paper introduces design concepts and implementations that could be used in creating gesture-based interactive installations for different space configurations.

2 DESIGN CRITERIA AND CORE DESIGN CONCEPTS

Our inspiration comes from various display and interaction technology combinations to provide engaging approaches to exploring 3D content. For instance, in CAVE VR systems [3], a 3D virtual environment is displayed on a surrounding display system to provide an immersive worldview with support for 3D interaction techniques for object manipulation and scene navigation. A fish-tank VR system provides a smaller virtual reality environment on a system that is a simple extension of a desktop environment [19]. The setup choice depends on various factors [5, 10], such as visualizing data, display availability, and sensing hardware. There exist frameworks for configuring the tiled display setups, such as MinVR and UniCave, but these are for non-raytracing solutions. Our vision is a ray tracing framework with the flexibility to support these different virtual reality setups. To achieve this vision, we formulate the following design criteria:

- C1 From a fish-tank VR to a CAVE VR, we need to support an arbitrary number of displays and spatial layouts, and we should be able to display a single, coherent virtual environment on these multiple displays.
- C2 As VR setups promote different interaction styles, we should be able to extend the system to support additional gestures and possibly integrate other motion-tracking technologies.
- C3 As exhibition spaces have different sets of available hardware and space constraints, we should be able to configure the system with minimum effort, e.g., without recreating or rebuilding an application.

2.1 Design Concepts

As shown in Figure 2, our core concept is **defining a view by arranging multiple, smaller camera views**, and each view is shown in a separate window to have the ability to arrange these windows/views to form a larger, single, even non-rectangular view (C1). For cluster-driven displays, the application can be extended to run on multiple nodes as separate processes, and workflows and content in these **processes are synchronized** to display a single, coherent 3D virtual environment

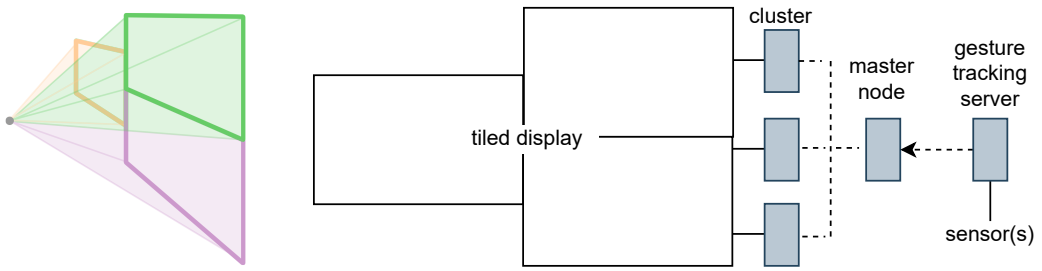

Fig. 2. Immersive OSPRay core concepts; (left) a view is divided into smaller views to display a 3D environment on non-planar or even non-rectangular displays; (right) the application can be extended to support cluster-driven displays, and the part that integrates sensor data is decoupled from the rendering application.

on these multiple displays. Also, to ensure user interactions do not conflict, **a master process handles all interactions**.

We also introduce additional concepts to configure the application for different exhibition settings. The application is **containerized** so that it can be run on the widest range of host systems possible. To help with modifying the arrangement for different installation spaces, we provide **display configuration files** and **a 3D tool for specifying spatial locations of displays** (C1, C3). The part that handles user inputs is decoupled from the main rendering application to be able to integrate a wide range of motion-tracking technologies (C2, C3). As some vendors' SDKs only run on specific platforms, the part that reads sensor data is implemented as **a separate server application** that sends body tracking data to the rendering client on the master node. Also, the part that interpreted the tracking data is written as **a plugin to the rendering application** so that a developer can easily interchange different sets of gestures for interaction techniques.

3 DETAILED DESIGN AND IMPLEMENTATION

To implement the concepts, we build upon Intel's open-source ray tracking application, OSPRay Studio [14], and use Microsoft Kinect for motion-tracking sensors. We chose OSPRay as a starting point as it has built-in support for MPI and plugins so that gesture-based interaction techniques can be add-ons to default keyboard-mouse interactions. First, we extend their default camera system to be able to show a 3D virtual environment by arranging multiple, smaller camera views (3.1). Each camera view is shown in a separate window; we provide another mode of running the application with the ability to open multiple windows and coordinate these windows (3.2). We provide gestured-based interaction techniques by integrating a separate server application sending tracking user data to a plugin to the rendering application (3.3). Finally, we provide mechanisms for setting displays (3.4) and deploying to various settings (3.5). See our implementation in https://github.com/jungwhonam/ImmersiveOSPRay.

3.1 Extending the default camera system

We extend OSPRay Studio's perspective camera class so that its projection plane does not have to be orthogonal to the viewing direction (Figure 2 (left)). In the previous version, a camera's projection plane is computed from the camera position, viewing direction, field of view, and aspect ratio. However, as we no longer assume that the camera is placed on the symmetry axis of the projection plane, we only use the camera position and require three corner positions of a projection plane. In our case, a physical display replicates a camera's projection plane. Thus, the three corners are the corners of a physical display.

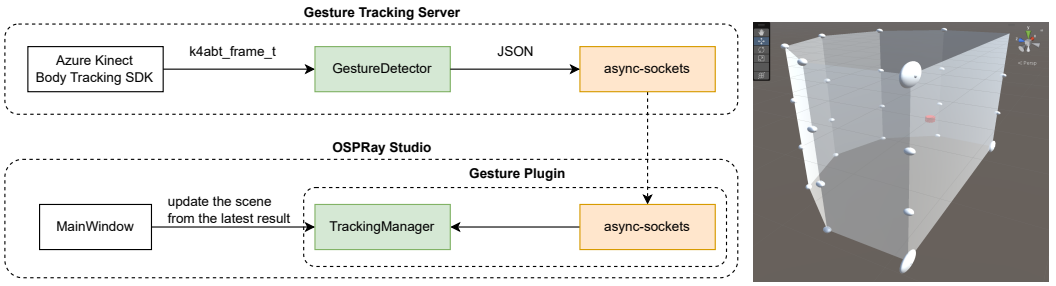

Fig. 3. Integrating sensors and display layouts; (left) Gesture Tracking Server sends processed body tracking data to the rendering application; (right) The configuration generator is built on Unity game engine to provide 3D object manipulation techniques for arranging display objects.

3.2 Opening and running multiple windows

We extend the built-in MPI support of OSPRay Studio to open and synchronize multiple windows. At the start, the MPI application creates a process per node, and each process opens a window and defines an off-axis projection camera based on values specified in the display configuration file (3.4). We take additional steps to run these multiple processes in a synchronized fashion. After processing user inputs, the master process updates values in a sharing object. Then, at the beginning of the next frame, the object is broadcast to other processes, and each process updates its objects and application states based on the shared values. Also, to ensure windows display rendering results simultaneously, processes wait for others to complete the rendering processes before swapping buffers. This is done by calling *MPI_Barrier(…)* before *glfwSwapBuffers(…)*.

3.3 Gesture Tracking Server and OSPRay Studio Plugin

The server gets body tracking data from a Kinect sensor, parses the data (removing unnecessary information, e.g., orientations of joints), and sends the data to connected clients (Figure 3 (left)). The OSPRay Studio plugin handles the connection with the server, computes gestures from received data, and keeps track of the latest state. When the plugin receives a message from the server, it derives additional information from the body tracking data (see 4 for our implementation of a flying navigation technique). The underlying scene is not updated immediately; OSPRay Studio initiates the process of updating the scene from the latest tracking data. When the application is in the phase of processing user inputs, e.g., key-pressed events, it calls a poll event method from the plugin to get the latest tracking result and uses it to update corresponding 3D objects, e.g., changing camera locations.

3.4 Display Configuration File and Generator

At the start, the application reads a configuration file to set its cameras and arrange windows. The JSON configuration file comprises an array of JSON objects, and each object represents a display containing information about an off-axis projection camera and the window that shows the camera view. Display Configuration Generator helps create the JSON file by providing a graphical assistant in specifying and arranging displays in a system (Figure 3 (right)). The tool is built on Unity game engine, allowing users to specify the positions of the three display corners and the eye by referencing Unity objects in a 3D view. Additionally, this tool provides features for placing these display objects around an axis and scaling a master display to fit all other displays.

3.5 Containers

The application is containerized so we can run our application on a host system that also supports DisplayCluster [9], a legacy application that relies on several well out-of-date supporting packages. Since our primary target system is a Linux cluster driving a display wall, we use Apptainer[6] to run the application container; this makes access to the cluster nodes' displays and interconnect transparent. The only requirements on the host are to have Apptainer and a version of MPI that matches that used inside the container. We use OpenMPI 4.1.1 over Ethernet on our target system; high-speed interconnects (when available) can be used by substituting the appropriate MPI support in the application container. The definition of the container is included in our repo (section 3).

4 APPLICATION CASES

Based on the implementation, we created a proof-of-concept prototype that shows a 3D virtual environment on tiled display walls driven by a cluster of nineteen PCs (Figure 1). Each node runs an MPI-process that shows a window on display, and all the displays form a hemisphere providing a surrounding view. A user moves around a 3D virtual environment by lifting both hands - pretending to be a bird - and leaning the body to fly in that direction. When multiple users are presented in the area, the user closest to the sensor is considered the primary user.

To come up with reliable flying interaction techniques, we used a demo application to test the tracking capability [1]: 1) Tracking of *HANDTIP* and *THUMB* joints becomes unreliable when hands are moving, below a belly, or far away (more than 1.5 meters away from the sensor), 2) When a user is far away, tracking of *HAND* joints becomes unreliable; confidence-levels become *NONE* or *LOW*, meaning that the joints are out of range or not observed, and 3) Joints on large body parts, e.g., *WRIST* and *CHEST*, can reliably be tracked even when a user is far away from the sensor, meaning the joints' confidence-levels are at least *MEDIUM*. Based on these observations, we decided to design our interaction techniques using joints on large body parts, e.g., *WRIST* joints, and ignored using joints such as *HAND*, *HANDTIP*, and *THUMB*.

Based on this decision, we created an interaction technique for flying through a 3D virtual environment. Lifting both hands above a belly button triggers a flying mode. Once in the mode, a camera moves into a body-leaning direction. When y-position of *WRIST* is above that of *SPINE_NAVEL*, the hand is considered as lifted. The leaning direction is computed by projecting the vector from *NECK* to *SPINE_NAVEL* to a x-z plane. The projected direction vector is then used to move the camera. As the vector was projected to a x-z plane, the camera does not move up or down.

5 OUTLOOK

Immersive OSPRay provides a series of tools for building interactive installations using an open-source ray tracing application and off-the-shelf motion tracking sensors. Our initial implementation could be helpful to the community as it provides a starting point for creating installations that could be used in different exhibition scenarios. These include: 1) using a fish-tank VR to show 3D models smaller in scale, such as medical data, and 2) integrating gesture-based interaction techniques to showcase visualizations in other toolkits such as ParaView and VisIt. Additionally, we plan to implement the concept with other ray tracing frameworks, such as OptiX, and support additional gestures by upgrading our server and plugin.

ACKNOWLEDGMENTS

Thanks to Bruce Cherniak, Will Usher and Kittur Ganesh for their guidance and support with OSPRay Studio. This work is funded in part by an Intel oneAPI Center of Excellence award.

[1] Kinect Body Tracking SDK tracks 32 joints, and each joint data object has position, orientation, and confidence-level values.

REFERENCES

[1] Divya Banesh, Rodman Ray Linn, and John M. Patchett. 2021. Vorticity-Driven Lateral Spread Ensemble Data Set. https://www.lanl.gov/projects/sciviscontest2022/report.pdf Accessed: 2023-04.

[2] Tim Biedert and Dave DeMarle. 2019. ParaView and VTK add GPU-accelerated ray tracing with NVIDIA RTX. https://www.kitware.com/paraview-and-vtk-add-gpu-accelerated-ray-tracing-with-nvidia-rtx/. Accessed: 2023-04.

[3] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon, and John C. Hart. 1992. The CAVE: Audio Visual Experience Automatic Virtual Environment. *Commun. ACM* 35, 6 (jun 1992), 64–72. https://doi.org/10.1145/129888.129892

[4] Thomas DeFanti, Daniel Acevedo, Richard Ainsworth, Maxine Brown, Steven Cutchin, Gregory Dawe, Kai-Uwe Doerr, Andrew Johnson, Chris Knox, Robert Kooima, Falko Kuester, Jason Leigh, Lance Long, Peter Otto, Vid Petrovic, Kevin Ponto, Andrew Prudhomme, Ramesh Rao, Luc Renambot, Daniel Sandin, Jurgen Schulze, Larry Smarr, Madhu Srinivasan, Philip Weber, and Gregory Wickham. 2011. The future of the CAVE. *Open Engineering* 1, 1 (2011), 16–37. https://doi.org/10.2478/s13531-010-0002-5

[5] Cagatay Demiralp, Cullen D. Jackson, David B. Karelitz, Song Zhang, and David H. Laidlaw. 2006. CAVE and Fishtank Virtual-Reality Displays: A Qualitative and Quantitative Comparison. *IEEE Transactions on Visualization and Computer Graphics* 12, 3 (may 2006), 323–330. https://doi.org/10.1109/TVCG.2006.42

[6] David Godlove. 2019. Singularity: Simple, Secure Containers for Compute-Driven Workloads. In *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (Learning)* (Chicago, IL, USA) *(PEARC '19)*. Association for Computing Machinery, New York, NY, USA, Article 24, 4 pages. https://doi.org/10.1145/3332186.3332192

[7] The Khronos Group. 2021. Khronos ANARI Registry. https://registry.khronos.org/ANARI/ Accessed: 2023-04.

[8] Intel. 2023. Intel ARC Graphics Processing Units. https://www.intel.com/content/www/us/en/products/discrete-gpus.html Accessed: 2023-04.

[9] Gregory P. Johnson, Gregory D. Abram, Brandt Westing, Paul Navratil, and Kelly Gaither. 2012. DisplayCluster: An Interactive Visualization Environment for Tiled Displays. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER '12)*. IEEE Computer Society, USA, 239–247. https://doi.org/10.1109/CLUSTER.2012.78

[10] J.J. LaViola, E. Kruijff, R.P. McMahan, D. Bowman, and I.P. Poupyrev. 2017. *3D User Interfaces: Theory and Practice*. Pearson Education. https://books.google.com/books?id=fxWSDgAAQBAJ

[11] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239, Article 2 (mar 2014). https://dl.acm.org/doi/10.5555/2600239.2600241

[12] NVIDIA. 2023. GeForce RTX - Ultimate Ray Tracing. https://www.nvidia.com/en-us/geforce/rtx/ Accessed: 2023-04.

[13] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.* 29, 4, Article 66 (jul 2010), 13 pages. https://doi.org/10.1145/1778765.1778803

[14] Isha Sharma, Dave DeMarle, Alok Hota, Bruce Cherniak, and Johannes Günther. 2021. OSPRay Studio: Enabling Multi-Workflow Visualizations with OSPRay. In *VisGap - The Gap between Visualization Research and Visualization Software*, Christina Gillmann, Michael Krone, Guido Reina, and Thomas Wischgoll (Eds.). The Eurographics Association. https://doi.org/10.2312/visgap.20211086

[15] Theoretical and Computational Biophysics Group UIUC. 2016. VMD 1.9.3. https://www.ks.uiuc.edu/Research/vmd/vmd-1.9.3/ Accessed: 2023-04.

[16] Ben Trumbore. 2017. ParaView Advanced Virtual Workshop. https://cvw.cac.cornell.edu/ParaViewAdv/ospray Accessed: 2023-04.

[17] I Wald, GP Johnson, J Amstutz, C Brownlee, A Knoll, J Jeffers, J Gunther, and P Navratil. 2017. OSPRay - A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (jan 2017), 931–940. https://doi.org/10.1109/TVCG.2016.2599041

[18] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.* 33, 4, Article 143 (jul 2014), 8 pages. https://doi.org/10.1145/2601097.2601199

[19] Colin Ware, Kevin Arthur, and Kellogg S. Booth. 1993. Fish Tank Virtual Reality. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands) *(CHI '93)*. Association for Computing Machinery, New York, NY, USA, 37–42. https://doi.org/10.1145/169059.169066

[20] Qi Wu, Will Usher, Steve Petruzza, Sidharth Kumar, Feng Wang, Ingo Wald, Valerio Pascucci, and Charles D. Hansen. 2018. VisIt-OSPRay: Toward an Exascale Volume Visualization System. In *Proceedings of the Symposium on Parallel Graphics and Visualization* (Brno, Czech Republic) *(EGPGV '18)*. Eurographics Association, Goslar, DEU, 13–24. https://dl.acm.org/doi/10.5555/3293524.3293526

Received 21 April 2023; revised 12 May 2023; accepted 12 May 2023