Experiencing Interior Environments: New Approaches for the Immersive Display of Large-Scale Point Cloud Data

Ross Tredinnick* Markus Broecker† Kevin Ponto‡

Wisconsin Institute for Discovery - University of Wisconsin-Madison

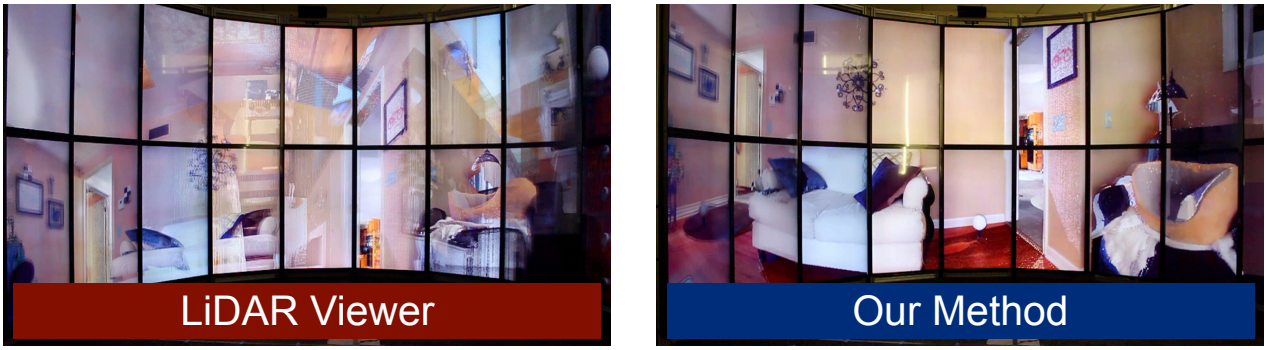

Figure 1: Shows comparison images between the proposed method and the method described in [2]. Note how objects in the left image seem transparent due to undersampling, while a similar view on the right appears solid.

ABSTRACT

This document introduces a new application for rendering massive LiDAR point cloud data sets of interior environments within high-resolution immersive VR display systems. Overall contributions are: to create an application which is able to visualize large-scale point clouds at interactive rates in immersive display environments, to develop a flexible pipeline for processing LiDAR data sets that allows display of both minimally processed and more rigorously processed point clouds, and to provide visualization mechanisms that produce accurate rendering of interior environments to better understand physical aspects of interior spaces. The work introduces three problems with producing accurate immersive rendering of LiDAR point cloud data sets of interiors and presents solutions to these problems. Rendering performance is compared between the developed application and a previous immersive LiDAR viewer.

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Virtual reality; I.3.8 [Computer Graphics]: Applications—

1 INTRODUCTION

The advent of LiDAR technology has enabled rapid acquisition of 3D environments via scanning of point cloud data sets.LiDAR scanners exist in two forms, airborne laser scanners (ALS) for scanning large-scale geographical locations, and terrestrial laser scanners (TLS) for ground-based scanning of environments. About the same time as the rise of the LiDAR scanner, immersive VR display systems gained popularity.Architectural walk-through of spaces became a common application for these display systems, yet, despite this application, and the ability for LiDAR to capture environments, there have been few examples of VR applications that allow walk-through of massive LiDAR data sets. When adapting the lone previous application from the literature to display interior point cloud data sets [2], three problems were encountered.

As file sizes of LiDAR point clouds are often larger than available graphics memory, many previous methods have utilized out-of-core rendering mechanisms. One common method to handle out-of-core rendering is to use a hierarchical approach which sorts points into spatial data structures. This method enables a "preview" subsampling of the data while information is being read from disk via a multi-resolution data structure [4]. With interior environments, this approach introduces a *see-through surfaces* problem where a reduction in information from subsampling can cause difficulty understanding the visualization of the data due to underdraw. As display systems increase in resolution, the issue of underdraw is exacerbated. This issue is particularly problematic for interior spaces as walls may be transparent during times of interaction while loading from disk. As shown in Figure 1, this can lead to confusion as a user may think they are in one room, only to find when the data is fully loaded that they are actually in another.

Along with this *see-through surface* problem, two additional issues when rendering LiDAR point clouds of interiors within immersive VR systems have been identified. First, households often contain reflective surfaces, such as mirrors, particularly in bathrooms and living rooms. LiDAR scanning works by line-of-sight and creates false (reflected) points that can end up in other parts of a home. Removing these false points during the manual registration and clean-up process leaves holes in locations of the original mirrors. This *mirror problem* can cause a disruption in presence while navigating the virtual environment.

Finally, TLS capture point clouds where the range of depth across points and number of directions points are captured varies more than an ALS point cloud. Depending on the hardware setup, a single PC's physical displays in an immersive VR cluster may cover a large field of regard. Therefore, with interior point cloud environments, an enormous number of points at varying depths from the viewer in several directions may need to be rendered by a single PC. This *varying depth problem* leads to exceeding memory limits quickly for a single PC and can quickly reduce overall interactivity in a VR display cluster.

*e-mail: rdtredinnick@wisc.edu
†e-mail: broecker@wisc.edu
‡e-mail: kbponto@wisc.edu

2 METHODS

Data Pipeline: Directly rendering points affords the ability to draw cluttered spaces and complex objects that are often found in home environments. First, raw LiDAR data is converted from the scanner's proprietary software, to Point Cloud Library's PCD format. From a single PCD file, an octree is written to disk as nested folders with each node as a folder and leaf nodes comprised of a PCD file of octant points and metadata file in the deepest folder. A tool then randomly rearranges the points in each octant to provide improved screen coverage Additionally, all PCD files are optionally written into a single binary file, via the same traversal, with an accompanying index file that contains offsets for starting points of octant point memory. The single binary file reduces file I/O operations thereby improving reading speed when loading points.

Rendering: The application, implemented with C++, OpenGL and GLSL, has rendered data sets consisting of up to one billion points. The resultant size of these data sets cannot fit into graphics or system memory. Therefore, the application streams octant points via multiple reading threads with priority based on octant distance from the user. Workload of the application is controlled by command line parameters for setting limits on physical and graphics memory. When either memory limit is exceeded, octants farthest from the viewer are unbuffered from the GPU or deleted from physical memory. Once occlusion visibility is fully resolved, the points are drawn with a light weight shader used in [1]. The GPU program adjusts point size within the vertex shader based on a point's radius projected to the near plane scaled to image coordinates.

Occlusion Culling: A common characteristic of building and household interiors is the significant amount of occlusion due to walls, ceilings and floors. The application uses this fact to help alleviate the *varying depth problem* and *see-through surfaces problem* by implementing two different occlusion culling strategies. The application runs a GPU implementation of hierarchical z-buffer based occlusion culling using OpenGL's transform feedback functionality. A GPU program performs frustum and occlusion culling on octants and returns the area of each visible octant's screen-projected bounding box. If the projected area is less than the number of octant points, the area is directly used as the number of points issued to the graphics card as a level of detail approximation during rendering. A second occlusion culling strategy is executed via hardware accelerated occlusion queries on child octants. For each child octant that passes frustum culling, an occlusion query is issued when rendering the octant's bounding box. If the occlusion query returns a pixel count above zero, the application considers all octant points visible until the octant moves offscreen or is recycled out due to exceeding memory limits. The application performs occlusion queries for the main screen, and uses the z-buffer approach when rendering mirrors to remove the need to check occlusion query results across potentially multiple frame buffer objects.

Mirrors: During the registration process, points located within mirrors are erased. Next, convex polygons are manually inserted where the original mirrors were located. These polygonal shapes are defined in world coordinates and exported to a basic ASCII text format for loading in-engine. Mirrors are reactive in the application but each mirror requires a separate render pass and two additional scene draws for correct stereo. Therefore, mirror contents are only updated if the mirror is visible via an occlusion query test and facing towards the user. Mirrors are rendered last to take advantage of depth testing against the static point clouds. Visibility of octants and loading of points is still determined when viewing a mirror.

3 EVALUATION

The application was tested on the DSCVR twenty panel LCD 3DTV immersive VR display system as it provided an environment useful for testing sampling issues [3]. The system is able to render a total of over 41 megapixels per frame. A point cloud environment of a 1400 square foot two-story house consisting of 264 million points was used to evaluate performance. Average frame rates were calculated for the proposed application and compared against rates from the other known immersive point cloud renderer by navigating the same continuous path through the first floor of the environment for one minute in each application. The movement traveled through two rooms and two hallways while loading points for three additional rooms that came into view through doorways. During the test, the graphics memory limit was set to 1.5 GB while system memory limit was intentionally set high to twelve GB to avoid any recycling of octants. To evaluate the performance with mirrors, navigation was performed along the same pattern from the previous test with a single mirror added to the scene.

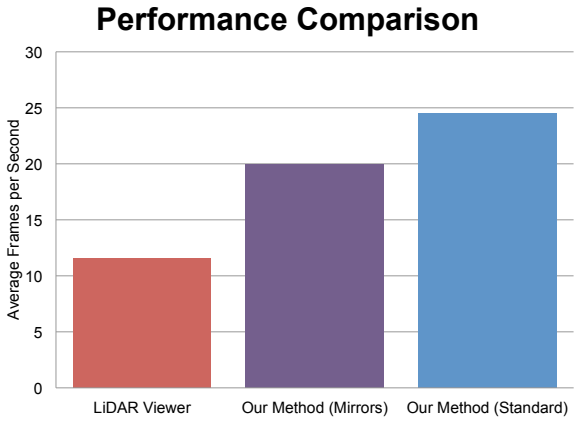

Figure 2: Compares average frames per second between the proposed method and the method described in [2] based on the procedure from Section 3.

4 RESULTS

As shown in Figure 2, the proposed method achieves twice the draw rate compared to the previous LiDAR viewing application. The authors believe this is due to the occlusion culling strategies employed to reduce points submitted to the GPU, something the other LiDAR viewer does not implement. Furthermore, Figure 1 shows differences between the proposed method and the other method for a single frame of the animation path. In the previous method, objects and rooms on the other side of walls and the ceiling can be seen given their sampling techniques. The proposed method renders the walls and ceilings such that they appear solid. While the inclusion of mirrors effectively creates correct reflected projections; a single mirror in a scene reduces the frame rate by approximately 20%. Due to the efficient occlusion culling algorithm run for each mirror (less than 0.2 ms to create the hi-z map), the reduction in frame rate mainly depends on the number of additional points sent to the GPU.

Future work will aim to better understand the effect of these techniques on the experience for the user. As an example, future studies may attempt to determine the effect of mirrors on a user's sense of presence. These studies will help to guide design decisions for immersive LiDAR rendering software.

REFERENCES

[1] C. Dachsbacher, C. Vogelgsang, and M. Stamminger. Sequential point trees. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 657–662. ACM, 2003.

[2] O. Kreylos, G. W. Bawden, and L. H. Kellogg. Immersive visualization and analysis of lidar data. In *Advances in visual computing*, pages 846–855. Springer, 2008.

[3] K. Ponto, J. Kohlmann, and R. Tredinnick. Dscvr: designing a commodity hybrid virtual reality system. *Virtual Reality*, pages 1–14, 2014.

[4] M. Wand. Point-based multi-resolution rendering. 2004.