Progressive Feedback Point Cloud Rendering for Virtual Reality Display

Ross Tredinnick* Markus Broecker† Kevin Ponto‡

Wisconsin Institute for Discovery - University of Wisconsin-Madison

Abstract

Previous approaches to rendering large point clouds on immersive displays have generally created a trade-off between interactivity and quality. While these approaches have been quite successful for desktop environments when interaction is limited, virtual reality systems are continuously interactive, which forces users to suffer through either low frame rates or low image quality. This paper presents a novel approach to this problem through a progressive feedback-driven rendering algorithm. This algorithm uses reprojections of past views to accelerate the reconstruction of the current view. The presented method is tested against previous methods, showing improvements in both rendering quality and interactivity.

Keywords: Point-based graphics, virtual reality, 3D scanning

Index Terms: K.6.1 [Management of Computing and Information Systems]: Project and People Management—Life Cycle; K.7.m [The Computing Profession]: Miscellaneous—Ethics

1 Introduction

Traditionally, there are two problems to overcome when rendering large point clouds: Physical or GPU memory limits, and high primitive counts that cause the GPU to become *geometry limited*. The traditional solution to these problems has been to involve an *out-of-core* aspect into the rendering application. Out-of-core solutions load data on demand as it enters the field of view of the current viewpoint. This involves incorporating an octree or similar three-dimensional spatial subdivision data structure into the application. Nonetheless, this approach does not fully solve the geometry limited problem as visibility of points inside of octants can only be naively estimated. Common estimation methods assume a uniform distribution of points within an octant and determine visibility of points based on distance metrics; however, as points for physical spaces, such as data gathered from LiDAR scans, are rarely uniform in distribution, these approximations are inaccurate. The presented application incorporates a new algorithm that takes advantage of the temporal coherency of point clouds between frames. Overall the algorithm provides three major contributions:

Interactive Progressive Rendering: While the view in a VR system changes continuously between frames, the change is small in nature, and thus, large parts of a previously rendered frame can be re-used to provide an initial best guess for the current frame's render output. As additional points are added to the feedback loop, view quality is progressively improved over time.

User Definable Interactivity Independent of Quality: In traditional point cloud rendering techniques, higher quality rendering output came at the cost of interactivity. In the presented method, interactivity remains constant with the trade off simply being a slightly longer time to converge to a high quality rendering output.

*e-mail: rdtredinnick@wisc.edu

†e-mail: broecker@wisc.edu

‡e-mail: kbponto@wisc.edu

Unlimited Data size: The required memory for geometry is constant. Combined with out-of-core rendering techniques, the method supports the visualization of data sets of potentially unlimited size, thereby elegantly handling the physical and GPU memory limits.

2 Method

The desired maximum number of points that can be rendered per frame without overdraw is equal to the number of pixels on screen with each point sampled by a unique pixel; however, such an ideal case rarely exists. The 3D projection of point clouds causes multiple points to be projected onto the same pixel, thus resulting in overdraw. Because it is impossible to know a-priori which points will be visible and which will not, z-buffer depth testing, occlusion, and view frustum culling are usually employed; however, this still requires the submission of many points to the graphics card.

The presented method works on the assumption that while the view changes between frames, the overall majority of points will be visible in two consecutive frames. As such, the previous frame can be reprojected using the current frame's camera information using a feedback loop. In order to fill the image in, additional points are added into the feedback loop on a per-frame basis. In this regard, the number of points rendered per frame can be described as: $P = V(R + S)$ with V being the number of viewpoints (for stereo rendering, this would be two), R being the resolution of the display system, and S being the number of points being streamed in a single frame. While V and R are predefined and static, S can be modified dynamically, thus providing a means to control frame rate. Drawing more points results in faster image convergence with the trade-off of a lower refresh rate. We note that with little movement between frames, this algorithm achieves a high reuse rate of points, and therefore, the number of reprojected points is low.

Algorithm: This algorithm selectively renders points from a source point cloud over several frames, which are accumulated each frame into the feedback loop, thereby maintaining interactive frame rates due to the spread of source point cloud rendering across frames. Furthermore, the algorithm uses little to no run-time memory with only an initial single dynamic memory allocation to allocate a single vertex buffer object (VBO) to upload to the GPU. A section of this VBO is updated each frame with newly streamed points, with the amount of points automatically determined by the application each frame based on the user's chosen frame rate. Two render target frame buffer objects are employed in a ping-pong configuration. The render loop consists of four distinct parts: 1) visibility determination, 2) reprojection, 3) drawing of additional points, and 4) display of the result. The first part traverses the octree and determines the visible nodes using view frustum culling. The visible nodes are then sorted front-to-back based on euclidean distance to the current camera. The point density estimate function determines how many points are potentially visible to the user for each visible octant and determines how many points are loaded from file. Note that unlike previous object space approaches, this estimation is only for the number of points to read, rather than the number of points to draw for the octant.

During the reprojection step, a screen-filling point grid with one point per pixel is drawn. The previous frame's render target buffer is bound as a read texture and the result is drawn to the second render target. For each point, its coordinates are used as texture coordi-

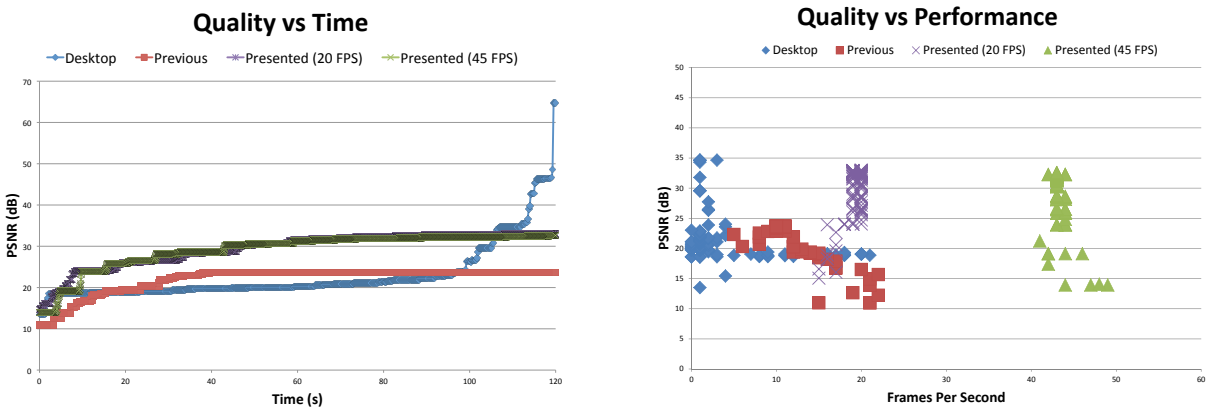

Figure 1: Results analyzing quality (left) and performance (right)

nates into the color texture. If the value read is different from the cleared depth value of the depth attachment of the frame buffer object, the point is considered valid and the RGB colors are treated as world coordinates. Its clip space position is calculated from these coordinates and the fragment shader writes the point's world position at the appropriate texel in the target render buffer. Next, a predetermined number of new points (Streamcount *S*) are drawn by drawing the points of visible octants. These points are drawn in a sequential manner over multiple frames and transformed similarly as above. While some of these points will fail the depth test due to the already drawn points, those that pass will converge the render target to the correct solution. Finally, the result is rendered to the screen and the render targets are swapped for the next pass.

3 Evaluation

To evaluate our approach, we compare it against a commercially available *desktop* viewer and a *previous* immersive point cloud viewer [3]. The desktop system subsamples the point cloud while interacting with the system before filling the data when motion is stopped. The previous viewer uses a level of detail scheme to determine which points to display, with the overall number of points drawn limited by available GPU memory. The test model consisted of 1.9 billion points within a single thirty GB binary file.

These applications were evaluated both for performance and for quality. Performance was measured by evaluating the number of frames drawn on a per-second basis. We chose to use a common quality metric in the field of image compression: peak signal to noise ratio (PSNR) [1]. PSNR is generally used by comparing a compressed image to a ground-truth uncompressed image with typical values falling between 20-40 dB [2]; however, it is not our intention to quantify a PSNR value for our render as "good," but simply to use the metric as a comparison between quality levels.

Ground-truth images for each application were generated by allowing the applications to converge for fifteen minutes. Viewpoints matching the ground-truth were loaded in a "cold-start" fashion, with each rendered frame recorded to video for 120 seconds. Additional metrics, such as frame rate were also monitored. The test was run using a mono rendering of a 1920x1920 resolution display at 120 Hz using an nVidia Quadro 5000 GPU with 2.5 GB of RAM with vertical sync off. Four different conditions were used consisting of the desktop viewer, the previous point cloud viewer [3], and two configurations of the presented method optimized to run at 20 and 45 frames per second, respectively.

Figure 1 demonstrates the results of these tests. The left image maps the achieved quality (PSNR) over elapsed time of a non-moving view into the scene. The images show that the presented method converges to a higher quality much more quickly than existing approaches. While previous techniques plateau once they reach their memory limit, the presented method is able to continually improve in the quality of the rendered image throughout the trial period. The desktop approach is able to eventually produce a higher quality image compared to the reference applications.

The right image shows achieved quality (PSNR) over a sustained frame rate as scatter plots. Not only does the presented method achieve high quality, the vertical columns highlight that a fixed refresh rate can also be guaranteed. The average frame-rate for a trial period was approximately three frames per second when using the desktop viewer. The previous approach for immersive environments improves the interactivity, with an average frame rate of about ten frames per second , but also produces a lower quality output in comparison. The presented method achieves an average of 19, and 43 frames per second with significant improvements in quality compared to the previous approach, shown in the right image in Figure 1. We note that the presented method achieves frame rates within 10% of the target frame rate. In all cases, the quality for the newly presented method is greater than in the previous method.

4 Conclusion

This paper presents a novel solution to rendering large point cloud data sets inside of virtual environments. While previous approaches have created a trade-off between quality and interactivity, the presented solution is able to overcome these limitations through a progressive feedback driven rendering scheme. Future work will aim to better understand the experience of the end user, develop ways to measure the quality of the rendering system during motion, and explore alternative sampling strategies for accelerated convergence.

Acknowledgements

Ponto and Tredinnick were supported by grant number R01HS022548 from the Agency for Healthcare Research and Quality. The content is solely the responsibility of the authors and does not necessarily represent the official views of the Agency for Healthcare Research and Quality. Broecker and Tredinnick were supported in part by the VCRGE at the University of Wisconsin - Madison.

References

[1] A. Hore and D. Ziou. Image quality metrics: Psnr vs. ssim. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2366–2369, Aug 2010.

[2] D. Salomon. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[3] R. Tredinnick, M. Broecker, and K. Ponto. Experiencing interior environments: New approaches for the immersive display of large-scale point cloud data. In *Virtual Reality (VR), 2015 IEEE*, pages 297–298, March 2015.