Remote rendering for ultrascale data

Kenneth Moreland[1], Daniel Lepage[2], David Koller[2] and Greg Humphreys[2]

[1]Sandia National Laboratories, P.O. Box 5800 MS 1323, Albuquerque, NM 87185-1323
[2]University of Virginia, 151 Engineer's Way, P.O. Box 400740, Charlottesville, VA 22904-4740

E-mail: kmorel@sandia.gov

Abstract. The mission of the SciDAC Institute for Ultrascale Visualization is to address the upcoming petascale visualization challenges. As we move to petascale computation, we are seeing a trend not only in the growth but also in the consolidation of computing resources. As the distances between user and petascale visualization resources grow, the expected performance of the network degrades, especially with respect to latency. In this paper we will explore a technique for remote visualization that leverages unstructured lumigraph rendering. This technique will provide an interactive rendering experience regardless of the network performance to the remote visualization resource. The unstructured lumigraph rendering can also replace many of the other level-of-detail techniques currently used that have problems that are exasperated by petascale data.

1. Introduction

As visualization and simulation platforms continue to grow and change, so to do the challenges and bottlenecks. Although image rendering was once the limiting factor in visualization, rendering speed is now often a secondary concern. In fact, it is now common to perform interactive visualization on clusters without specialized rendering hardware. The new focus on visualization is in loading, managing, and processing data. One consequence of this is the practice of co-locating a visualization resource with the supercomputer for faster access to the simulation output [1, 2].

Most of us are not fortunate enough to have a supercomputer or associated visualization resource located in our office, yet we still require access to these resources. A remote visualization capability allows us to provide the widest access to our resources; anyone with network capabilities can perform large scale visualization from their desktop using remote visualization.

The distance between scientist and computing/visualization resources, as a trend, is growing. The US Office of Science, through programs like SciDAC, is encouraging scientists from different disciplines and different organizations to collaborate; the US Department of Defense and Department of Energy are consolidating resources like supercomputers; universities are reaching out to researchers around the world. As the distance between user and visualization resource becomes larger, remote visualization becomes more important and more challenging. One of the goals of the SciDAC Institute for Ultrascale Visualization is to ensure that remote visualization remains an interactive experience, even when accessing petascale data on the far side of the world.

2. Remote Rendering

Unlike most large-scale parallel processing, visualization tends to be an interactive process by nature. The parallel visualization must be controlled remotely to make its use convenient and, in some cases, possible. Thus, most large-scale turn-key visualization systems, such as ParaView, VisIt, and EnSight, provide some way to access the parallel visualization remotely. The architecture of these systems varies, but all comprise a client, run on a local desktop or laptop, that connects, via a socket, to one or more servers running on a remote parallel machine. For large scale problems, it is critical that all processing, including rendering, occurs on the server. The parallel rendering usually employ one of the "sort-last" algorithms for their scalability with respect to the geometry size [**?**, 3, 4].

Given enough computational resources, a sort-last parallel rendering algorithm should be able to render ultrascale data in a fraction of a second. However, there are times when this is not possible either because the physical resources are strained or because the constant overheads of the parallel rendering algorithm are too high. For these circumstances, some scalable visualization tools provide a special "interactive" render mode in which quality can be sacrificed for speed (although a high-quality "still" render mode is always available). For example, ParaView [5] provides geometric decimation [6], image decimation, and lossy image compression to achieve interactive rates.

When the client and server are connected over a wide area network, the latency of the network may make interative rendering from the server impossible. When rendering locally on the client, the visualization tool has a delicate balance between quality of the rendering and how much geometry it can store. Too often, the local client renderings either under-represent the data or require too much data to be pulled from the server.

We are investigating a new technique for approximate interactive rendering in remote visualization using image based techniques. As shown in Figure 1, high-quality still images generated by the server are cached on the client. This cache of images is used to synthesize images from new viewpoints during interactive rendering.

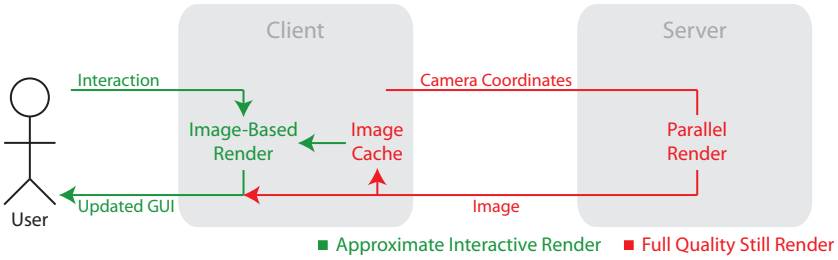


Figure 1. Update cycles for remote rendering with a local image-based approximate interactive rendering in the client.

Using images rendered from the server to synthesize other approximate images has two distinct advantages. First, it is scalable with respect to the size of the geometry being represented. Like the sort-last parallel rendering employed on the server, the overhead of the technique is independent of geometry size. Second, the image base rendering can leverage the rendering already being done on the server. Lengthy preprocessing of the data is not necessary, as is the case with geometric decimation.

3. Unstructured Lumigraph Rendering

We perform the client-side image-based rendering by extending the Unstructured Lumigraph Rendering technique [7]. This method constructs a blending field across the image plane

representing how the set of previously rendered input images will be combined. The color at each pixel of the new image is determined by projecting the input images onto a geometric proxy and merging the resulting images according to the blending weights.

3.1. The Proxy

We use a 3D triangle mesh proxy to map pixels from prior rendered images to pixels in the image plane of an interpolated view. The quality of the synthesized image increases when the proxy closely approximates the shape of the actual dataset. However, high quality interpolated views can be generated even when the proxy detail is low, as a coarse hull of the 3D model [8], or even a simple bounding volume such as a rectangular prism or a sphere will still produce high-quality renderings given a sufficiently dense set of known images (see Figure 2).

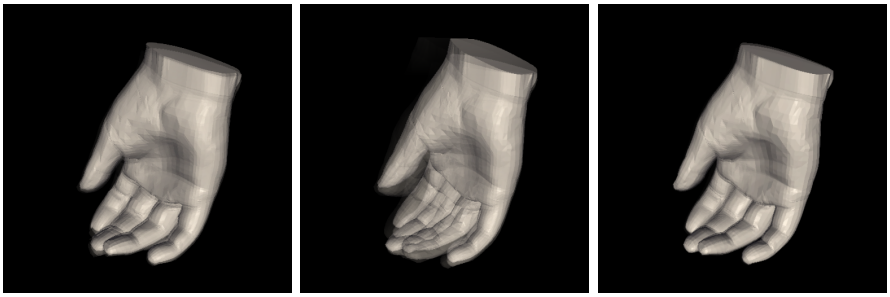


Figure 2. Left: An interpolated view of a 3D mesh model, projected onto a high-quality proxy. Center: The same view, using an axis-aligned bounding box as the proxy; the image quality is reduced. Right: Using the bounding box proxy again, but interpolating a greater density of sample images. The result is comparable with the high-quality proxy image, but does not require knowledge of the actual geometry.

3.2. Sampling the Blending Field

Rather than recomputing the blending field at every point in the image plane, we choose a discrete set of sample points and triangulate them, interpolating the blending weights linearly across each triangle as in [7]. We place a sample point at every vertex of the visible faces of the proxy geometry, at the center of projection for every known view, and in a regular grid across the image plane. We add the proxy edges and the square edges of the regular grid as constraints, and then perform a conforming Delaunay triangulation of the resulting 2D mesh in the image plane (Figure 3).

3.3. Projecting Back to the Proxy

Each triangle of this triangulation is either outside the bounds of the proxy geometry, or is a visible subsection of a face of the proxy. We project these triangles back onto the proxy by raytracing from the camera through the center of each triangle. If the ray does not intersect the proxy, then the triangle is discarded; otherwise, the ray identifies a unique face of the proxy that contains the triangle. In the latter case, we intersect rays through each of the triangle's vertices with the plane of the face to get the world-space coordinates of the sample triangle on the proxy.

One advantage of this method over that of [7] is that it allows concave proxies without repeating triangles - a sample triangle located in front of several proxy faces is only processed once, for the nearest face.

3.4. Blending Weights

We assign each pre-rendered input image a weight at each sample point. The client obtains an exact image from the server whenever the user stops moving, and so it is important to ensure that when our technique is used to synthesize a view that coincides with a portion of a known pre-rendered view, this known view has weight 1 within that portion, with all other views having weight 0 (the epipolar consistency property).

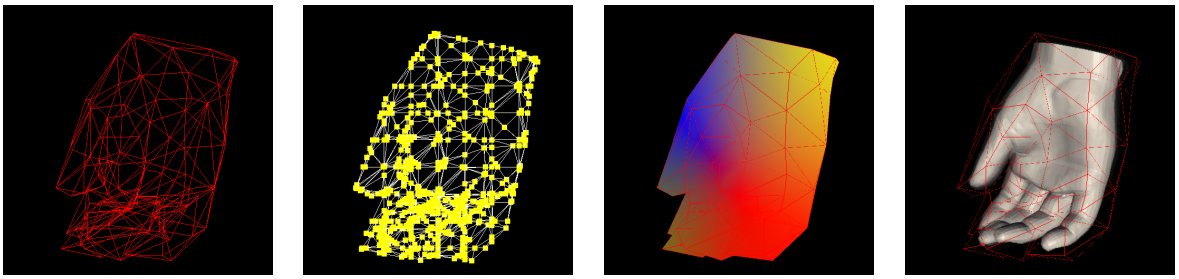
We view each input image as a collection of known rays in the lightfield, and compute the weights by comparing the rays out of the known viewpoints with those from the new view; this allows us to synthesize new views from a mix of perspective and orthographic cameras, as both are common in scientific visualization.

The system of penalties used by [7] does not provide true epipolar consistency, so we have modified the blending weights. We compute four weights for each known image at each sample point, ranging from zero to infinity. One weight is based on whether the sample point is occluded in the view of the known image, one on how close the sample point is to the edges of the known image's view frustum, one on the angle between the ray from the known image to the point and the ray from the new view to the point, and one based on the difference between the distances from the known and new views to the sample point. The total weight per camera per sample point is the product of these four weights.

For each sample point, we then find the k cameras with highest weights. We subtract the kth highest weight from the weights of the $k-1$ cameras with higher weights, then divide each of them by the sum of these $k-1$ nonzero weights so that they sum to one. This gives us a uniform blending field that sums to one at every visible point on the proxy, and that will vary smoothly when the synthesized viewpoint moves.

3.5. Rendering

We render each sample triangle m times, where m is the number of cameras that have nonzero weights for at least one of the three vertices. Each time, we enable the corresponding camera's texture and perform projective texturing using the blending weight as the alpha value. This results in an image that interpolates the k nearest known views to produce a new view. For the example images shown in Figures 2 and 3, $k = 4$.



A coarse geometric proxy for mesh

The sample points of the blending field

The resulting blending field

The final synthesized view, with the proxy overlaid

Figure 3. Stages of the unstructured lumigraph rendering.

3.6. Prefetching Images

The unstructured lumigraph rendering technique is most effective when the input images available for interpolation densely sample the nearby visualization space for the new view being rendered on the client. We attempt to roughly model and predict the user's navigation

path through the 3D environment using a number of heuristics, and then accurately prefetch corresponding rendered images from the server. Our prefetcher adapts the volume of its requests to the state of the network traffic and server load.

4. Conclusions

The basic algorithms for the unstructured lumigraph rendering as well as the heuristics for prefetching images are implemented. Our current work involves applying these techniques to large scale scientific data. Much of this involves integrating lumigraph rendering into the ParaView application, as demonstrated in Figure 4. Our current challenges include adding multithreading to the rendering routines to allow prefetching to happen during idle times and to generate proxy geometries of general data sets.

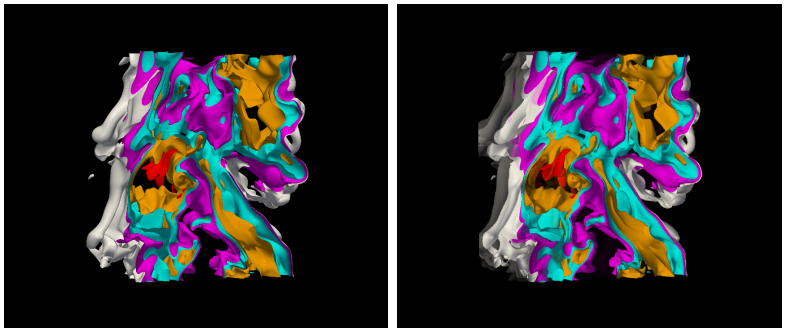


Figure 4. Turbulent combustion data rendered within ParaView (data courtesy of Jacqueline Chen, Sandia National Laboratories). Left: Full geometry render. Right: Lumigraph rendering with simple bounding box proxy.

Acknowledgments

This work was done in part at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] Ahern S 2007 Petascale visual data analysis in a production computing environment *Journal of Physics: Conference Series (Proceedings of SciDAC 2007)* vol 78

[2] Cedilnik A, Geveci B, Moreland K, Ahrens J and Farve J 2006 Remote large data visualization in the ParaView framework *Eurographics Parallel Graphics and Visualization 2006* pp 163–170

[3] Ma K L 1994 Parallel volume rendering using binary-swap image composition *IEEE Computer Graphics and Applications* **14** 59–68

[4] Wylie B, Pavlakos C, Lewis V and Moreland K 2001 Scalable rendering on PC clusters *IEEE Computer Graphics and Applications* **21** 62–70

[5] Squillacote A H 2007 *The ParaView Guide* ParaView 3 ed (Kitware, Inc.) ISBN-13: 978-1-930934-21-4

[6] Lindstrom P 2000 Out-of-core simplification of large polygonal models *Computer Graphics (Proceedings of SIGGRAPH 2000)* 259–262

[7] Buehler C, Bosse M, McMillan L, Gortler S and Cohen M 2001 Unstructured lumigraph rendering *Computer Graphics (Proceedings of SIGGRAPH 2001)* 425–432

[8] Buehler C, Bosse M, McMillan L, Gortler S J and Cohen M F 2001 Unstructured lumigraph rendering *Computer Graphics (Proceedings of SIGGRAPH 2000)* 425–432