Point-based VR Visualization for Large-scale Mesh Datasets by Real-time Remote Computation

Jinghua Ge, Daniel J. Sandin, Andrew Johnson, Tom Peterka, Robert Kooima, Javier I. Girado, Thomas A. DeFanti

Electronic Visualization Laboratory
University of Illinois at Chicago
jinghua@evl.uic.edu

Figure 1: Point-based view construction of experimental datasets

Abstract

High speed interactive visualization of large-scale mesh datasets for desktop VR facilities is still a challenge because of the slow geometry setup and rasterization for huge number of small triangles. This paper presents a point-based virtual reality (VR) visualization pipeline for large-scale mesh datasets in a client-server architecture. Remote server computation which samples the triangle mesh into discrete 2D grids is steered by the client-end interactive frustum request. A point-based geometry is built up incrementally during run time for both server and client. By organizing the point model into a multi-resolution octree-based space partition hierarchy, the client-end visualization ensures fast view reconstruction by splatting the available points onto the screen with efficient occlusion culling and view-dependent level of detail (LOD) control. The combination of the high-priority client side local splatting and server side low-speed view updating decreases the dependence on remote computation performance and network requirements for an interactive VR visualization.

Keywords: remote computation, point splatting, occlusion culling, view-dependent LOD, view reconstruction.

I. Introduction

VR systems with head-tracking that provide a stereo or auto-stereo visualization experience to interactive end users require fast view-construction speed, at least 15 frames per second (fps) for each eye. Current scientific visualizations usually contain hundreds of millions of primitives, with gigabytes of texture information. Typical rendering software cannot fulfill VR performance requirements because of memory and processor limitations. To address this problem, a 3D **P**oint sample **P**acking and **V**isualization **E**ngine (PPVE) with remote computation is presented. The PPVE system operates in a client-server architecture, with a server side process performing point sampling of the original dataset, and a client side process generating fast local point splatting for view reconstruction. The point splatting technique is first introduced in [Westover90] by representing a 3D point as a 2D footprint in view reconstruction. In PPVE, a most-recent point-based partial geometry is dynamically built up during run time based on client's navigation path. The server and client processes are loosely linked, so the server's process frame rate entirely depends on its computation performance while the client's view reconstruction frame rate is determined by its local point splatting speed. The server's computation is steered by client interaction and the result is sent to the client over a high-speed network for new view updating. For the client, each view update frame adds more useful point samples into its cached point geometry, thus providing a more accurate view reconstruction.

Since the PPVE server is completely steered by the client's navigation in real-time, the application needs no pre-processing for geometry simplification and texture segmentation, and only the visible part of the original dataset needs to be retrieved and sampled. Other graphics techniques such as occlusion culling and view-dependent LOD control are used to ensure fast and smooth view reconstruction at the client side. Figure 2 shows the workflow of the PPVE application.

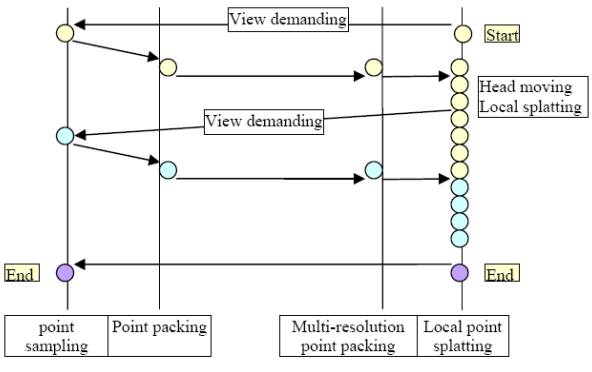


Figure 2: workflow of a PPVE system

VRCIA 2006, Hong Kong, 14–17 June 2006.
© 2006 ACM 1-59593-324-7/06/0006 $5.00

The following section describes related work in the field of large-scale dataset visualization. Section 3 outlines the architecture of the PPVE framework. Section 4 explains the demand-driven perspective point sampling techniques. Section 5 elaborates on the multi-resolution point model organization and local point splatting. Results are given in Section 6, and the paper concludes with some remarks on future activities.

2. Related work

Local real-time visualization of large-scale datasets must typically address two key problems: view-dependent adaptation of the model resolution, and efficient occlusion culling [Funkhouser93]. These methods usually involve long preprocessing time to build multi-resolution geometry and texture hierarchies with space partitioning. Out-of-core memory loading algorithms are also critical [Borgeat05]. In the case of remote data resources, the entire dataset may need to be downloaded in advance.

Grid-based visualization [Shalf03] techniques are developed to enable the visualization of scientific data using remotely available high-end visualization architectures from any internet-connected desktop computer. The parallel servers provide some combination of 2D images, triangulated surfaces, and 3D subvolumes on demand to every visualization frame of the interactive client. The client uses modern programmable graphics hardware to provide combined geometry and volume rendering displays [Engel00a, Engel00b]. In these techniques, real-time interaction fully depends on server computation performance, network bandwidth and latency, and load balance. Special algorithms are used to address these problems [Bowman04, Xiong05]. Moreover, the client is limited by memory and CPU performance for viewing and interacting with the data.

Recently, point based rendering (PBR) techniques have been used to represent complex objects in a manner that avoids the large rendering time and computational costs of polygons. Using points as a modeling and display primitive has certain advantages such as compact model representation, straightforward simplification mechanism, and freedom from texture mapping [Gross01]. Point based rendering is now used widely in very complex scene rendering where each polygon may only occupy less than one pixel [Rusikiewicz00], and in volume rendering. [Zwicher01b]. When point based rendering is used to visualize large-scale mesh dataset, current PBR techniques usually need to have a complete point geometry representation of the original mesh in advance with detailed point sample information such as sample density and shape. Some applications [Dachsbacher03] assume uniform point sampling.

In PPVE, we present a real-time point sampling and splatting algorithm for large-scale triangle meshes in a client-server architecture. In the application environment, the client is a head-tracked autostereoscopic VR desktop, and the server is a single Linux computer connected by the CAVEwave [CAVEwave] gigabit network. When visualizing large-scale scientific datasets, the client can achieve at least 15 fps for real-time autostereo drawing, while the server rendering rate may be only 1 fps. The PPVE method uses a loosely linked client-server computation architecture, whereby the client caches a point model for local splatting in high speed, while new view updates from the server can be much slower. PPVE enables fast VR view reconstruction and decreases the dependence on server computation performance and on network bandwidth and latency for real-time interaction. PPVE essentially de-couples client performance from server and network constraints.

3. Functional framework

Figure 3 is a diagram showing the functional framework of the PPVE system. The framework includes a remote computation center (server) and a VR desktop (client). The server and client can be connected by a local area network or high-speed internet, depending on the location of the data source.

Both server and client processes are multi-threaded so that the functional modules, such as data communication, server-end point sampling and client-end point splatting can be running concurrently, taking advantage of multi-processors if applicable.

Given a frustum request, server side computation includes sampling the visible part of the original mesh into the discrete 2D grids of proper resolution and eliminating the redundant sampling before transmitting the new view to client. Client's main purpose is to splat visible 3D points which are stored in the geometry cache into seamless 2D views. Also for each new view update, client will extract multi-resolution 3D points from the sampled view and pack them into the geometry cache.

To keep the most-recent point model in both server and client's main memory, obsolete points will be deleted the same time as new points are added. Obsolete point deletion is originated by client and needs to be synchronized between server and client to ensure correct redundancy elimination.

Networking between the server and client provides data communication, such as view-frustum demanding, map transmitting and obsolete data notification.

In Sections 4 and 5, each functional module of the server-end and client-end processes will be discussed in detail.

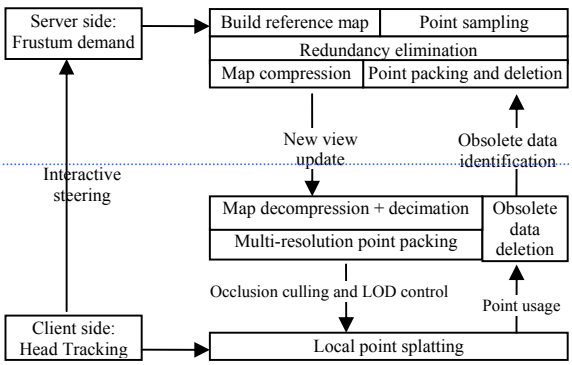


Figure 3: Framework of a PPVE system

4. Server: remote computation

The remote server waits for frustum demand from the client and accordingly provides a discrete sampling at a proper resolution of

the original continuous triangle mesh. Server also keeps a compact packing of the previous samplings for redundancy elimination's purpose. Obsolete sample deletion which can prevent memory over-use should be synchronized between the server and client for correct redundancy elimination.

4.1. Point sampling

The server has direct access to the dataset and provides point sampling of the dataset in response to the client's interaction. Some point sampling techniques [Grossman98] assume uniform point distribution by sampling orthographic views on an equilateral triangle lattice. In our point-based visualization pipeline, the server usually performs perspective sampling according to the client's perspective viewing demand. Perspective sampling causes non-uniform point distribution and irregular surface coverage. The output point samples from the server computation are of multiple resolutions by nature.

For each frustum demand, the server samples the visible part of the dataset in perspective by projecting it onto discrete 2D grids of color map and depth map. The color map and depth map are collectively henceforth termed "depth-image". The depth image-based representations (DIBR) as a new family of 3D geometry representation [Ignatenko03] have been adopted into MPEG-4 Part16: Animation Framework eXtension (AFX).

4.2. Redundancy elimination

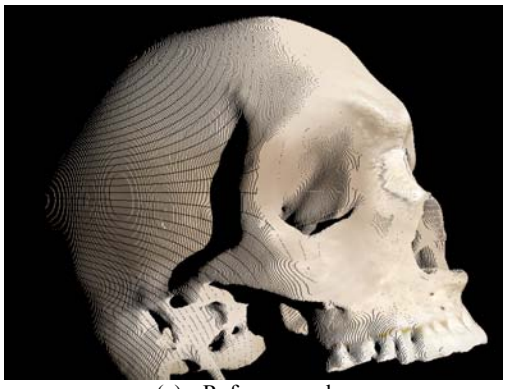
To maintain point geometry compactness, repetitive sampling needs to be deleted among multiple sampling frames. A pixel in a depth-image becomes redundant when the 3D geometry it represents gets a higher resolution sampling in another depth-image. Obviously a complete redundancy elimination within one depth-image is related to all the sampling frames of an overlapping 3D geometry coverage. As a real-time point packing and visualization pipeline, data deletion of an existing point packing can break the integrity of the current packing and involve complicated data identification scheme. For simplicity, an approximate redundancy elimination algorithm is applied in the current PPVE framework by only deleting redundant pixels in current sampling maps. This simple redundancy elimination algorithm is introduced below:

First, build reference maps by rendering the available point samples without splatting.
Assume there exists a current depth map Z and reference depth map *refZ*.
for every pixel i in Z and *refZ*
{
  if ($Z(i) < refZ(i) - \varepsilon$) keep pixel i in current sampling maps
  else discard pixel i in current sampling maps
}

Here ε is a small threshold to remedy the possible difference of depth maps sampled from triangle meshes and previously extracted 3D points.

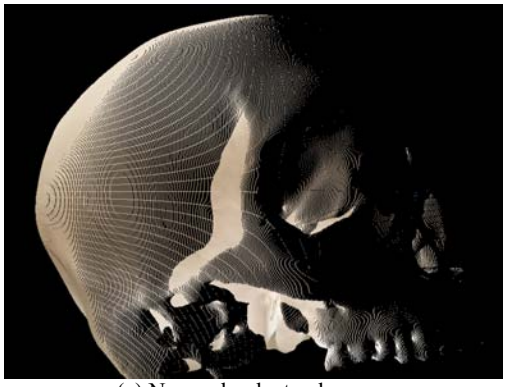
This algorithm is very simple because it only needs one map comparison operation per pixel. Also it ensures data integrity after data is actually packed, because it only deletes redundancy in the current sampling map. The tradeoff is its inability to delete previous low resolution data. In the case of previous samplings in low resolution and a current sampling in high resolution, instead of deleting the redundant low resolution data in the old data packing, some new high resolution data is deleted. The resulting problems in the client-end splatting process and an approximate solution will be discussed in Section 5.

Figure 4 is an example of new color map, reference color map and non-redundant color map for one server-sampling frame. From the maps we can see that all the pixels in the current sampling map are deleted if they already appeared in the reference map with same depth value.



(a) Reference color map

(b) Current sampling color map



(c) Non-redundant color map

Figure 4: Example of color maps before and after the redundancy elimination.

4.3. Map compression and network requirements

After redundancy elimination, the updated maps (depth-image) are compressed and sent to the client over the network. A lossless LZW compression [Nelson89] is used for map compression. For a depth-image with resolution of 1280*800, the original data size is about 7MB; compression reduces the size to about 2MB. The compressed maps are sent to the client using reliable TCP/IP protocol. Because the server sampling rate is expected to be low, e.g. one frame per second, the network traffic is small and bursty.

4.4. Point extraction and packing

The server samples the triangle mesh into a depth-image according to a frustum demand from the client. One 3D point can be extracted from each valid pixel of the sampled depth-image by un-projection. The server maintains a most-recent point model as a record of the previous samplings in order to eliminate redundancy in a new sampling. A basic 3D point has attributes of its coordinates and color. An approximate normal could be calculated from the depth map, but this is not included in the current implementation. Even though point samples do have different shapes and sizes, these attributes are not saved for each point at the current point extraction stage.

Because of the conceptual simplicity of using points as the modeling primitives, the extracted point samples can be packed together without topological connection enforcement to form a compact and photorealistic representation of the original 3D geometry. The point-based geometry is composed of point patches and reconstructed incrementally and dynamically during run time. A ***point patch*** is a collection of points extracted from each server sampling frame after redundancy elimination. Compared to existing point patches, a new point patch has either new geometry coverage or higher sampling resolution. The client-side navigation path determines how the point patches are defined. The dynamic (real-time) patch definition is different from the pre-processed geometry segmentation because it is completely view-dependent and will be different every time the application runs.

Figure 5 shows the composing point patches of a point-based geometry. Points belonged to the same point patch are signified by same color. Even though a point patch is extracted from a uniform sampling grid under a certain viewing transformation and resolution, its irregular distribution can be seen clearly at different viewing conditions. The point packing stage incrementally constructs a point based representation of a continuous triangle mesh from multiple perspective projections.

For local storage, the point model is partitioned into ***point clusters*** and organized into a level-limited octree, with each leaf node representing a point cluster inside its bounding box. The original point patches extracted from every computation frame are partitioned into mosaics to fit into the octree leaf nodes bounding boxes, and every mosaic has sequential memory storage for fast data access. For a point cluster, its constituent point patch mosaics from different server-sampling-frames are linked together. The octree-based space partition hierarchy of the point geometry enables fast occlusion culling and efficient data access. Figure 6 shows a space partition of current point geometry.

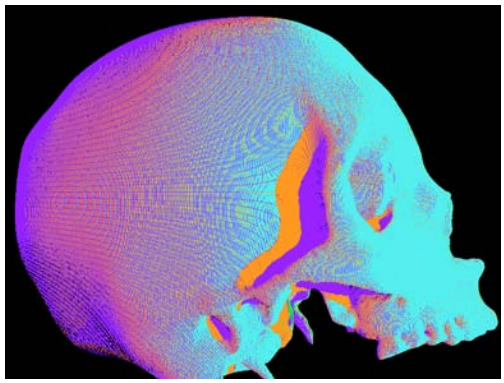


Figure 5: Point patches of a point-based geometry, signified by different colors

Figure 6: A space partition of current point geometry

4.5. Obsolete data deletion

The server side obsolete data deletion is steered by the client. When the client wants to delete some point clusters, the identification number of the corresponding octree leaf node is sent to the server over network. An octree leaf node's identification number is solely decided by it's bounding box position relative to the whole tree. By maintaining the same octree structure as the client, the server's obsolete data deletion is assured to be synchronized with the client. Once a point cluster is marked to be deleted, all of its point patch mosaics are to be deleted.

5. Client: VR visualization

The purpose of client-end visualization is to achieve fast and seamless 2D view reconstruction from the irregularly distributed point samples. To serve this purpose, the client side software introduces its own data updating and point splatting techniques. The data updating techniques include multi-resolution point generation from existing samples, point clustering in the level-limited octree structure, and obsolete data deletion. The point splatting algorithm assigns a proper splat size for each point sample in a view-dependent manner to get a view-reconstruction as seamless as possible.

5.1. Map decompression and decimation

At the client side, a data receiving thread is always waiting for incoming data from the remote server. After a non-redundant depth-image is received, it is decompressed first and 3D points are extracted from those maps. To get multi-resolution point samples, the depth-image can be decimated.

There are two ways to decimate a depth-image:

- Re-render the originally extracted 3D points in the same frustum but with smaller viewport. Decimated maps with arbitrary resolution can be produced in this way.
- Decimate the maps directly: use a linear filter for the color map re-sampling and a minimum point filter for the depth map re-sampling.

The direct decimation technique is used in current implementation.

5.2. Multi-resolution point packing

At the client side, the most-recent point model is organized into the same octree structure as the server side, except that from each view-sampling-frame we now get a set of multi-resolution point patches re-sampled by decimation. The multi-resolution point patches extracted from one view-sampling frame represent the same data with different level of detail.

Points are clustered based on the octree leave nodes' bounding boxes. For one point cluster, each of its composing point patches has an attribute called ***pixel range*** indicating its data resolution. The term ***pixel range*** refers to the projected pixel coverage of a point cluster's bounding box under a certain viewing condition. ***Pixel range*** is used as a quantified indication of point sample resolution for a point patch. Another attribute of a point patch is the point number it holds. The octree leaf node keeps the record of memory storage of its point cluster.

Figure7 shows the data structure of an octree leaf node at client side.

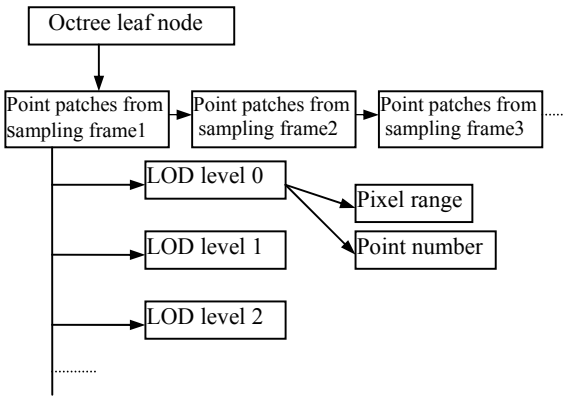


Figure7: data structure of an octree leaf node at client side

5.3. Obsolete data deletion

An obsolete data deletion mechanism is used based on a point cluster's most recent access time, so that the point model can always fit into the client's main memory. To facilitate the obsolete data identification, the ***access time*** attribute of the chosen point cluster's octree leaf node will be updated every view-construction frame. Below is the obsolete data deletion algorithm:

```
for every second, search each leaf node of the octree
{
    Assume current time is t and the node's latest access time is nt
    if (t - nt) > ϵ
    {
        Calculate the current node's ID number;
        Delete this node's point cluster storage;
    }
}
Send all deleted node IDs to the server for synchronization.
```

The threshold ϵ indicates the no-access period beyond which the points will be considered to be obsolete.

Obsolete data deletion is especially important when dealing with visualization with data animation or deformation, similar to the particle system implementations. [Reeves83]

5.4. View reconstruction (Local splatting)

The client side visualization is achieved by splatting the locally cached point model for interactive view reconstruction. The quality of view reconstruction depends on the point splatting algorithms. High quality point splatting techniques such as the perspective accurate splatting [Zwicher04] use EWA resampling filter to achieve perspective correct splat shapes, avoiding artifacts such as holes caused by the affine approximation of the perspective projection. Instead of using slower high-quality point splatting, the fastest and simplest point splatting by rendering GL_POINTS with different point width is currently used in the PPVE framework due to the critical requirement of visualization frame rate in VR applications. Our experiments have shown that the viewing quality is good enough.

The octree-based space partition of the point geometry enables fast frustum culling, by which the client-end view reconstruction can efficiently splat only those point clusters inside the current viewing frustum onto the screen. Back face culling can be used to further cull invisible point patches inside a chosen point cluster if each point patch has a normal cone[Shirman 93] attribute.

After a point cluster passes the frustum culling, the ***desired pixel range*** is computed based on its bounding box position and the current viewpoint and viewing matrices. This desired pixel range indicates the desired sample resolution for this portion of the geometry as if it is directly sampled from the original triangle mesh. Based on the desired sample resolution, one point patch of the closest resolution will be chosen for each available point sampling inside the point cluster. After that, a view-dependent splat-size calculation algorithm is applied to splat the chosen point samples onto screen with proper point width.

Below is the two-step point splatting algorithm for one point cluster:

Step 1. For each sampling frame, choose a proper resolution point patch whose pixel range is the closest to the desired pixel range.

Step 2. Calculate the splat size of the chosen point patch by equation (1):

$$\textbf{Splat size} = \frac{\textit{scale factor} * \textbf{point patch pixel range}}{\textbf{desired pixel range}} \qquad \textbf{(1)}$$

Here the *scale factor* is used to ensure a 2D view reconstruction is as seamless as possible by point splatting.

Figure 8 illustrates the view reconstruction algorithm for a point cluster which contains point samples from four view-sampling frames. Each view-sampling frame has three point patches of different resolutions, which are signified by their pixel ranges. The whole structure is illustrated by a pyramid filled with colored lines, where each line indicates a point patch and all lines with the same color are from the same view-sampling frame. The length of each line indicates the corresponding point patch's pixel range. For example, the longest light green line at the bottom of the pyramid is the originally sampled point patch, and the other two lines with the same color is the decimated version of the original point patch. The dashed line simulates the current view-reconstruction with a desired pixel range. The colored arrow indicates that, from each view-sampling frame, one point patch which has the closest pixel range to the desired pixel range value is chosen and then all the point samples in the chosen point patch are either minified or magnified to form a new view reconstruction.

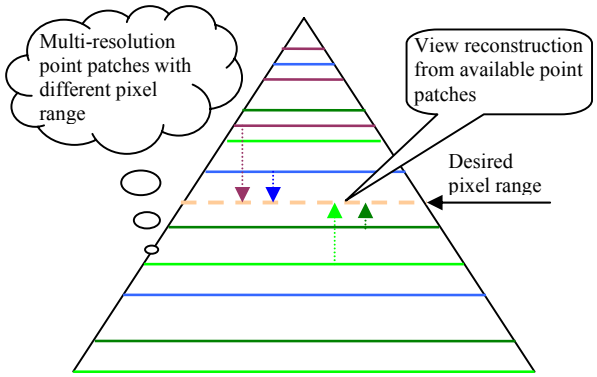


Figure 8: view reconstruction from multi-resolution point patches for one point cluster. Each colored line in the pyramid indicates a point patch with a certain resolution. Longer line indicates larger pixel range or higher resolution. Point patches from different sampling frames are signified by different colors.

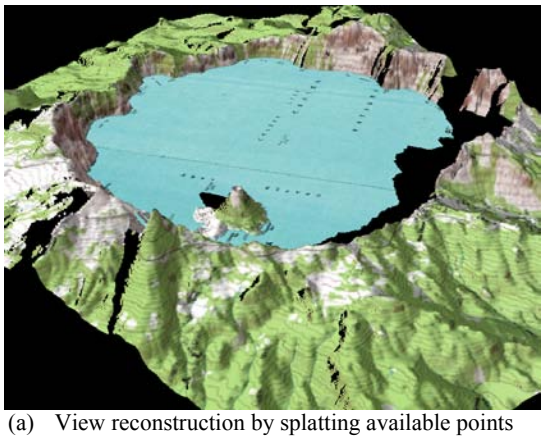
There is one more issue in the view reconstruction process. In Section 2, the drawback of the redundancy elimination algorithm was discussed. Using the splat size calculated by equation (1), sometimes the new view reconstruction will be blurred even if there are high-resolution data available. This is because low-resolution sampling may attempt to get point samples with depth value smaller than or equal to high-resolution sampling, so high-resolution data will actually be blocked by the depth test stage of graphics processing. To remedy this problem, the splat size determination algorithm is revised as following:

Let sp_size = splat size of the point patch with closest pixel range to the desired pixel range value ;

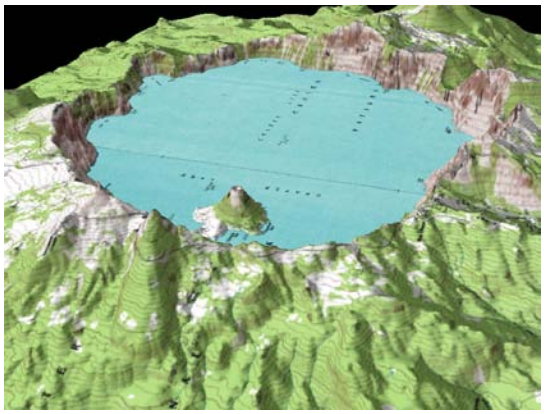
```
for each chosen point patch
{
   compute its splat size cu_size;
   if(cu_size > ratio * sp_size) cu_size = sp_size + ε * cu_size;
}
```

Here *ratio* > 1, and 0< ε <1. Basically this algorithm means that when a low-resolution point patch may block out the higher-resolution data, its splat size should be set as a smaller value.

Furthermore, the viewpoint's moving velocity can also be considered in the LOD control algorithm. In this case, the desired pixel range in equation (1) will be divided by the moving velocity first for the splat size calculation. The viewpoint's moving velocity is computed by the client every frame as the head tracking position change divided by the passing time.



(a) View reconstruction by splatting available points



(b) View reconstruction after new view update

Figure 9: 2D view reconstruction by point splatting before and after new view update from remote server.

The view reconstruction by splatting locally cached 3D points with constant head movement will introduce holes and gaps because of data incompleteness, and splatting artifacts because of low data resolution. By replenishing new points every view updating frame, a better view will be constructed in exchange for a short waiting period. The interactive VR visualization experience is expected to be quite smooth if the waiting time for the new view updating is no more than 2-3 seconds since the user can still move their head and see the existing splats at about 15fps.

Figure 9 shows the 2D view reconstruction by point splatting before and after a new view update from remote server. In image (a), the view reconstruction contains holes, gaps and fat splats, but they will disappear in image (b) after new view update without any noticeable artifacts.

6. Experiments and results

The experimental server computer is a Linux machine with Intel Xeon 1.8 GHz CPU, Quadro4 900 XGL graphics card and 2 GB main memory. The client computer is also a Linux machine with dual 64 bit AMD Opteron 246 2 GHz processors, Quadro FX 4400 graphics card and 4 GB main memory.

The client computer drives a Varrier™ autostereo VR display. To achieve the autostereo effect, a virtual linescreen and the visualization scene need to be drawn twice (once for each eye) in every frame. [Sandin05]

The server sampling is at 1280*800 resolution, and the client visualization is at 2560*1600 resolution. The client and server resolutions are independent; they do not need to be integral multiples of each other nor have the same aspect ratio.

Table 1 lists the description of the experimental datasets and results. All datasets are non-transparent surface triangle meshes.

Figure 1 at the start of this paper shows the client-side point-based view reconstruction of those datasets.

Dataset	Size (triangles)	Avg. server sampling rate	Avg. client visualization rate
Crater lake	5M	1 fps	17 fps
Skull	0.45M	12 fps	20 fps
Bone, skin, head together	1.2 M	4 fps	18 fps

Table 1: dataset description and experimental results

The overall performance of the client-end view reconstruction is more than 15 fps, independent of the original dataset complexity. Although the above experiments only show the visualization results of triangle mesh surfaces, the server computation is not limited to the traditional rendering of triangle meshes. Ray tracing computation and volume sampling can also be accommodated in the PPVE framework.

7. Conclusion and future work

The combination of the server-sampling and client-splatting techniques makes smooth interactive VR visualization of large datasets possible. Transparency handling is still a challenge, since correct point sampling of a transparent dataset is not a straightforward task.

The average server computation time per frame is expected to be no more than 2-3 seconds for a reasonable view update rate in client visualization. So, for scientific dataset sizes on the order of terabits, the server computation itself requires accelerations which are not covered in current PPVE framework.

In the next step, the framework is expected to be extended to support computer cluster-based multi-processor server computation and client-side tiled display.

GPU-accelerated splatting techniques [Botsch03] will be implemented later for faster and higher-quality point splatting.
A normal cone attribute will be added to every point patch in a point cluster, making efficient backface culling and client side lighting possible.

Finally, a more sophisticated algorithm is under design to deal with dataset transparency and deformation.

8. Acknowledgement

The Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago specializes in the design and development of high-resolution visualization and virtual-reality display systems, collaboration software for use on multi-gigabit networks, and advanced networking infrastructure. These projects are made possible by major funding from the National Science Foundation (NSF), awards CNS-0115809, CNS-0224306, CNS-0420477, SCI-9980480, SCI-0229642, SCI-9730202, SCI-0123399, ANI 0129527 and EAR-0218918, as well as the NSF Information Technology Research (ITR) cooperative agreement (SCI-0225642) to the University of California San Diego (UCSD) for "The OptIPuter" and the NSF Partnerships for Advanced Computational Infrastructure (PACI) cooperative agreement (SCI 9619019) to the National Computational Science Alliance. EVL also receives funding from the State of Illinois, General Motors Research, the Office of Naval Research on behalf of the Technology Research, Education, and Commercialization Center (TRECC), and Pacific Interface Inc. on behalf of NTT Optical Network Systems Laboratory in Japan. Varrier and CAVELib are trademarks of the Board of Trustees of the University of Illinois.

9. References

[Borgeat05] Louis Borgeat, Guy Godin, Francis Blais, Philippe Massicotte, Christian Lahanier: GoLD: Interactive Display of Huge Colored and Textured Models, In *Proceedings SIGGRAPH 2005*, pages 869-877. ACM SIGGRAPH 2005.

[Botsch03] Mario Botsch and Leif Kobbelt. High-quality point-based rendering on modern GPUs. In *Proceedings Pacific Graphics2003*, pages 335–343. IEEE, Computer Society Press, 2003.

[Bowman04] Ian Bowman, John Shalf, Kwan-Liu Ma, and Wes Bethel, "Performance Modeling for 3D Visualization in a Heterogeneous Computing Environment" (June 30, 2004). *Lawrence Berkeley National Laboratory.* Paper LBNL-56977.

[CAVEwave] CAVEwave™ End-to-End 10 Gbps Wavelength Inaugurates National LambdaRail, www.evl.uic.edu

[Dachsbacher03] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In Proceedings ACM SIGGRAPH 03, pages 657–662. ACM Press, 2003.

[Engel00a] K.Engel, Ove Sommer, T.Ertl: A Framework for Interactive Hardware-Accelerated Remote 3D-Visualization, Data Visualization 2000, Springer Computer Science, Pages 67-177, 291

[Engel00b] Engel, K., Hastreiter, P., Tomandl, B., Eberhardt, K., Ertl, T. : Combining local and remote visualization techniques for interactive volume rendering in medical applications. In Proc. of IEEE Visualization 2000, 449–452.

[Funkhouser93] FUNKHOUSER, T.A., SEQUIN, C.H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of ACM SIGGRAPH 93*, pages 247-254. ACM SIGGRAPH 1993.

[Gross01] Markus H. Gross. Are points the better graphics primitives. In *Computer Graphics Forum 20(3), 2001*. Plenary Talk Eurographics 2001.

[Grossman98] J.P. Grossman and William J. Dally. Point sample rendering. In *Proceedings Eurographics Rendering Workshop 98*, pages 181–192. Eurographics, 1998.

[Ignatenko03] Alexey Ignatenko, Anton Konushin. A Framework for Depth Image-Based Modeling and Rendering, *13-th International Conference on Computer Graphics and Vision GraphiCon-2003* Moscow, September 5 -10, 2003. Conference Proceedings, pp.169-172.

[Nelson89] Mark Nelson. LZW Data Compression. *Dr. Dobb's Journal*. October, 1989

[Reeves83] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactionson Graphics*, 2(2):91–108, Apr. 1983.

[Rusinkiewicz00] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings SIGGRAPH 2000*, pages 343–352. ACM SIGGRAPH, 2000.

[Sandin05] D. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, T. DeFanti. "The Varrier™ Autostereoscopic Virtual Reality Display", In *Proceedings SIGGRAPH 2005*, pages 894-903, SIGGRAPH 2005, July 2005

[Shalf03] John Shalf, E. Wes Bethel: The Grid and Future Visualization System Architectures. IEEE Computer Graphics and Applications 23(2): 6-9 (2003)

[Shirman 93] Shirman, L. and Abi-Ezzi, S. "The Cone of Normals Technique for Fast Processing of Curved Patches," *Proc. Eurographics*, 1993.

[Westover90] L. Westover. Footprint Evaluation for Volume Rendering. In *Computer Graphics*, Proceedings of SIGGRAPH 90, pages 367–376. August 1990.

[Xiong05] C.Xiong, J.Leigh, E.He, V.Vishwanath, T.Murata, L.Renambot, T.DeFanti: LambdaStream – a Data Transport Protocol for Streaming Network-intensive Applications over Photonic Networks, Proceedings of The Third International Workshop on Protocols for Fast Long-Distance Networks, Lyon, France. February, 2005

[Zwicher01] M. Zwicker, H. Pfister, J.v.Baar, M.Gross, Ewa volume splatting, IEEE Visualization 2001, pp 29-36, 2001.

[Zwicher04] ZWICKER, M., REN, J., BOTSCH, M., DACHSBACHER, C., AND PAULY, M. 2004. Perspective accurate splatting. In *Proceedings of Graphics Interface 2004*. 247–254.