Rendering and Compositing Infrastructure Improvements to VisIt for Insitu Rendering

Burlen Loring and Oliver Oliver Rübel

November 3, 2015

Abstract

Compared to posthoc rendering, insitu rendering often generates larger numbers of images, as a result rendering performance and scalability are critical in the insitu setting. In this work we present improvements to VisIt's rendering and compositing infrastructure that deliver increased performance and scalability in both posthoc and insitu settings. We added the capability for alpha blend compositing and use it with ordered compositing when datasets have disjoint block domain decomposition to optimize the rendering of transparent geometry. We also made improvements that increase overall efficiency by reducing communication and data movement and have addressed a number of performance issues. We structured our code to take advantage of SIMD parallelization and use threads to overlap communication and compositing. We tested our improvements on a 20 core workstation using 8 cores to render geometry generated from a 256^3 cosmology dataset and on a Cray XC31 using 512 cores to render geometry generated from a $2000^2 \times 800$ plasma dataset. Our results show that ordered compositing provides a speed up of up to $4\times$ over the current sort first strategy. The other improvements resulted in modest speed up with one notable exception where we achieve up to $40\times$ speed up of rendering and compositing of opaque geometry when both opaque and transparent geometry are rendered together. We also investigated the use of depth peeling, but found that the implementation provided by VTK is substantially slower,both with and without GPU acceleration, than a local camera order sort.

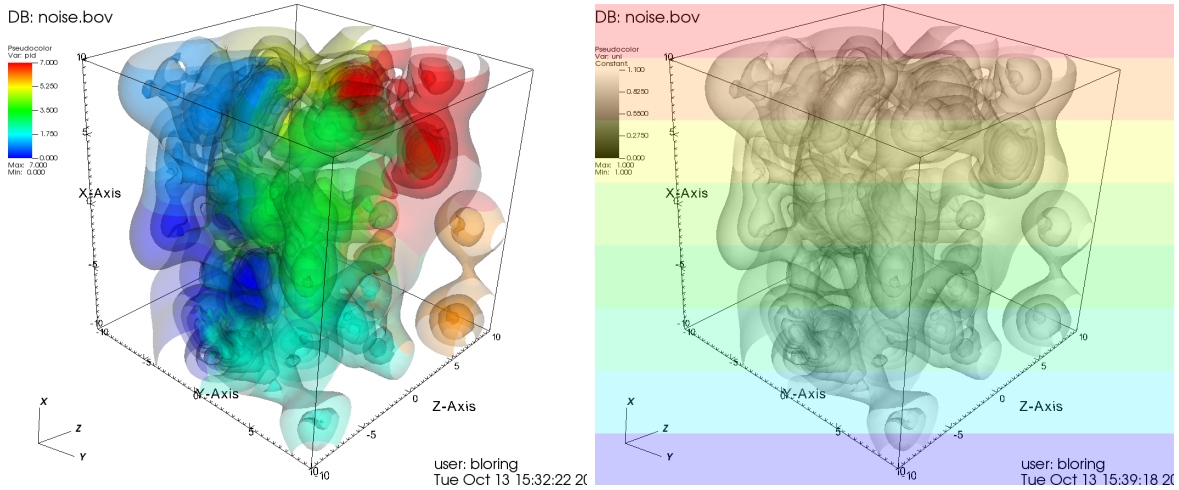


Figure 1: Ordered compositing and image tiled compositing example. The original domain decomposition is shown on the left. Ordered compositing renders the data using this decomposition. The tiled image compositing decomposition is shown on the right. Tiled compositing moves data to that decomposition prior to rendering.

composite order = 4 5 6 0 7 1 2 3	
round 0:	$4 \leftarrow 5, 6 \leftarrow 0, 7 \leftarrow 1, 2 \leftarrow 3$
round 1:	$4 \leftarrow 6, 7 \leftarrow 2$
round 2:	$4 \leftarrow 7$

1 Introduction

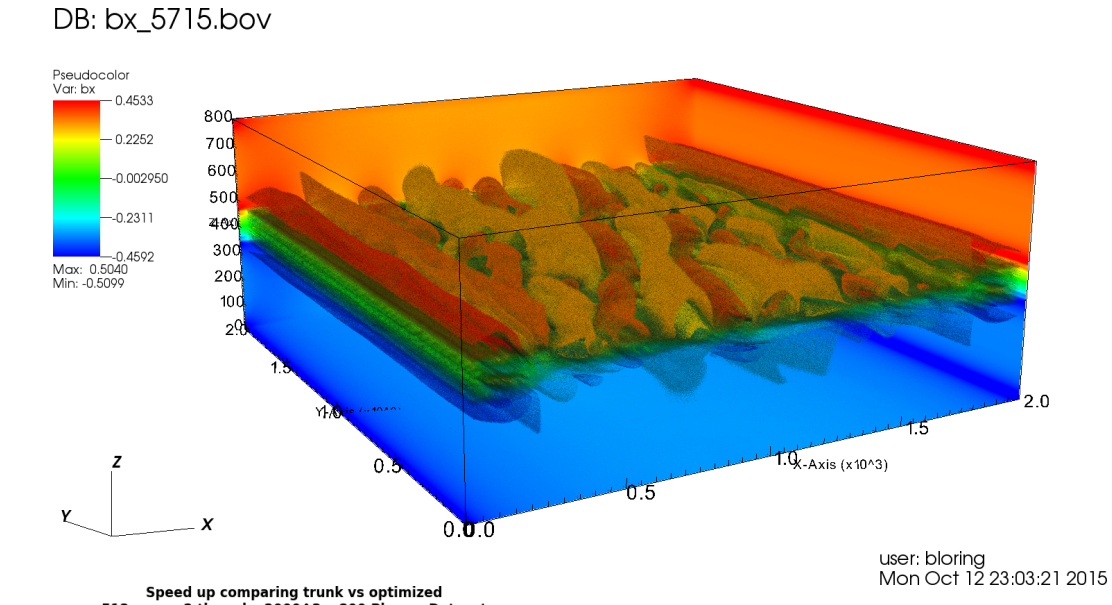
Insitu visualization and analysis seeks to reduce or eliminate data movement to and from disk by linking a simulation and a visualization run time using shared memory for data transfer. Insitu rendering has the potential for huge reductions in I/O costs as problem sized data is reduced to a single image before I/O. One potential drawback to insitu rendering is that one often doesn't know which fields will be interesting and which rendering parameters will produce the best result. For these reasons, compared posthoc rendering, insitu rendering generates a larger number of images, covering more fields, and wider array of rendering parameters. For these reasons rendering performance and scaling becomes a crucial issue in the insitu setting.

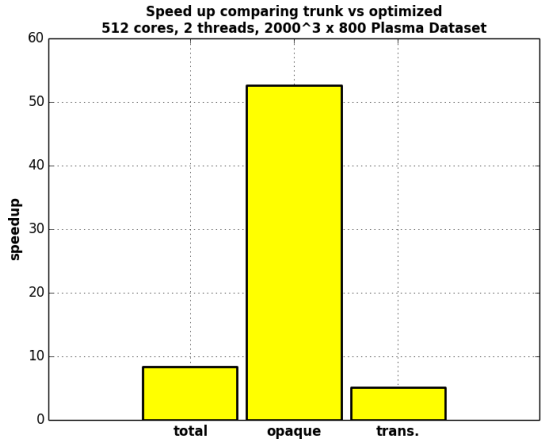
We profiled VisIt during 3D simulations of proton accelerator using WarpIV, an insitu visualization application that couples VisIt and Warp a laser plasma accelerator framework. Our profiling identified the rendering and compositing of translucent geometry as issues.

In this work we present optimizations and improvements to VisIt's rendering and compositing infrastructure. We added the capability for alpha blend compositing and use it with ordered compositing when datasets have disjoint block domain decomposition to increase performance when rendering transparent geometry. We also made improvements that increase overall efficiency and addressed a number of performance issues. We structured our code to take advantage of SIMD parallelization and use threads to overlap communication and compositing. We tested our work on a 20 core workstation using 8 cores to render geometry generated from a 256^3 cosmology dataset and on a Cray XC31 using 512 cores to render geometry generated from a $2000^2 \times 800$ plasma dataset. Our results show that ordered compositing provides a speed up of up to $4\times$ over the current sort first strategy. The other improvements resulted in modest speed up with one notable exception where we achieve up to $40\times$ speed up of rendering and compositing of opaque geometry when both opaque and transparent geometry are rendered together. We also investigated the use of depth peeling, but found that the implementation provided by VTK is substantially slower,both with and without GPU acceleration, than a local camera order sort.

2 Description of the Work

Prior to our work VisIt always used a "sort first" strategy for rendering transparent geometry in parallel. In this strategy prior to rendering data was moved using MPI such that each process had all the geometry covered by a screen space tile. After rendering each process sends its tile to rank 0 for assemblage. In this arrangement one can apply serial rendering algorithms, and alpha blend compositing is not necessary. This is illustrated in figure 1. On the left a dataset with 10 iso-surfaces is shown. The coloring by MPI rank id reveals the block based domain decomposition. In the sort first strategy, prior to rendering, the transparent geometry is moved to the screen space domain decomposition shown on the right. In what





Data rendered, Plasma		
type	**desc.**	**zones**
opaque	3 slice	7190403
trans.	20 iso	464815961

Speedup vs trunk	
total	8.34
opaque	52.65
transparent	5.12

Render and Composite Time				
code	**configuration**	**total**	**trans.**	**opaque**
opt.	global sort	21.0305	20.6666	0.354182
	ordered comp.	4.55346	4.06768	0.326197
trunk	global sort	37.9864	20.8121	17.1737

Figure 2: **Top:** $2000^2 \times 800$ plasma dataset **Bottom Left:** Speedup of rendering and compositing using 512 cores, and 2 threads, on the $2000^2 \times 800$ plasma dataset. Total and per pass time are shown. In the opaque pass z-buffer compositing is used while in the transparent pass ordered compositing and alpha blending are used. **Bottom Right:** Data and measurements.

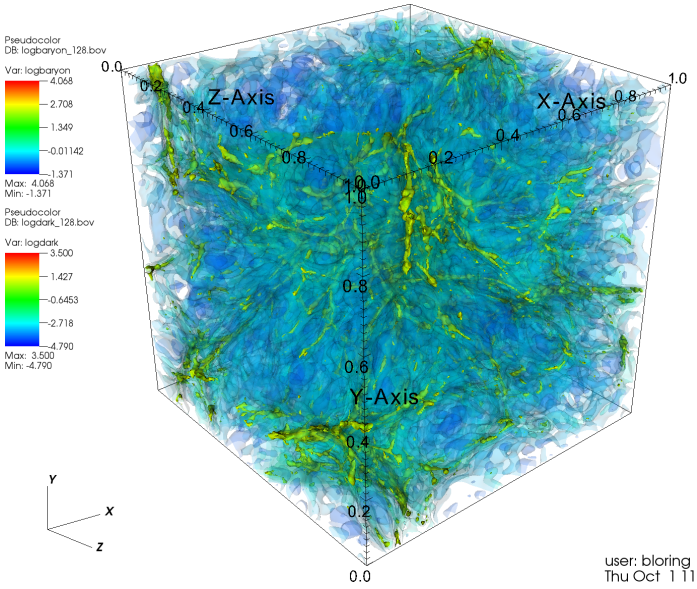
follows we refer to this data movement as a "global sort". The global sort is expensive because all of the data may need to be moved and it doubles the amount of memory used to store transparent geometry.

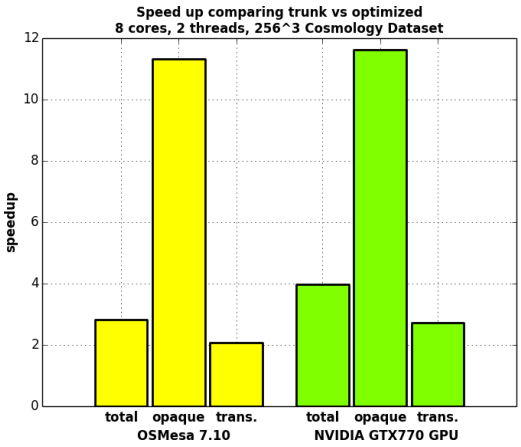
Our work builds on VisIt svn rev 27127(9/9/2015). We used rev 27127 as the baseline to compare our optimizations against. Parallel rendering in VisIt occurs in two logical passes, a pass for opaque geometry, and a pass for transparent geometry. Prior to our work, the opaque pass included a depth based composite where the unsigned char r,g,b color channels and float depth values are packed into an AOS(array of structures) layout and composited in user defined MPI reduction operation; and the transparent pass included a distributed global geometry sort which moves data onto a decomposition such that each rank ends up with geometry for a unique screen space tile, followed by a local camera order sort and render, followed by an MPI gather of each ranks tile to rank 0.

Our work extended the transparent rendering and compositing capabilities of VisIt in four main ways:

- We implemented an alpha blending compositer and a depth based compositer. Our new compositer operates on SOA(structures of arrays) data layout to make use of compiler auto-vectorization. We used threads to overlap communication and compositing. We arranged the communications in a binary tree pattern but also made provision for supporting a variety of alternative communication patterns.

- We implemented the ordered compositing optimization, which for translucent block domain decomposed data eliminates the communication associated with the global sort and cuts memory overhead in half.



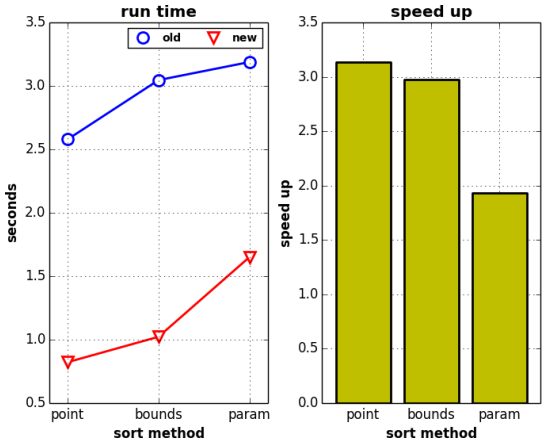


Cosmology data rendered		
type	**desc.**	**num zones**
opaque	1 iso	488520
trans.	10 iso	9884369

Speedup compared to trunk		
GTX770	total	3.98
	opaque	11.64
	transparent	2.73
OSMesa	total	2.83
	opaque	11.34
	transparent	2.08

Render and Composite Time, Cosmology							
		OSMesa 7.10			NVIDIA GTX770 GPU		
code	**configuration**	**total**	**trans.**	**opaque**	**total**	**trans.**	**opaque**
opt.	global sort	4.30541	4.11388	0.187808	2.51254	2.33435	0.174466
	ordered comp.	2.16096	1.93264	0.184575	1.09519	0.860169	0.172628
	depth peel	42.0545	41.8255	0.185432	6.2353	6.02038	0.170824
trunk	global sort	6.11546	4.02224	2.09239	6.11546	4.02224	2.09239

Figure 3: **Top Left:** 256^3 cosmology dataset. **Top Right:** Speedup of total rendering and compositing time using 8 cores, and 2 threads, on the 256^3 cosmology dataset. Total and per pass time are shown. In the opaque pass z-buffer compositing is used while in the transparent pass ordered compositing and alpha blending are used. **Bottom:** Data and measurements.



code	point	bounds	param
old	2.58 sec	3.05 sec	3.19 sec
new	0.82 sec	1.02 sec	1.65 sec
speed up	3.14 x	2.98 x	1.93 x

Figure 4: **Left:**Time and speed up of just the depth sort sorting the iso-surfaces shown in figure 3 using 1 core. **Right:** Recorded times.

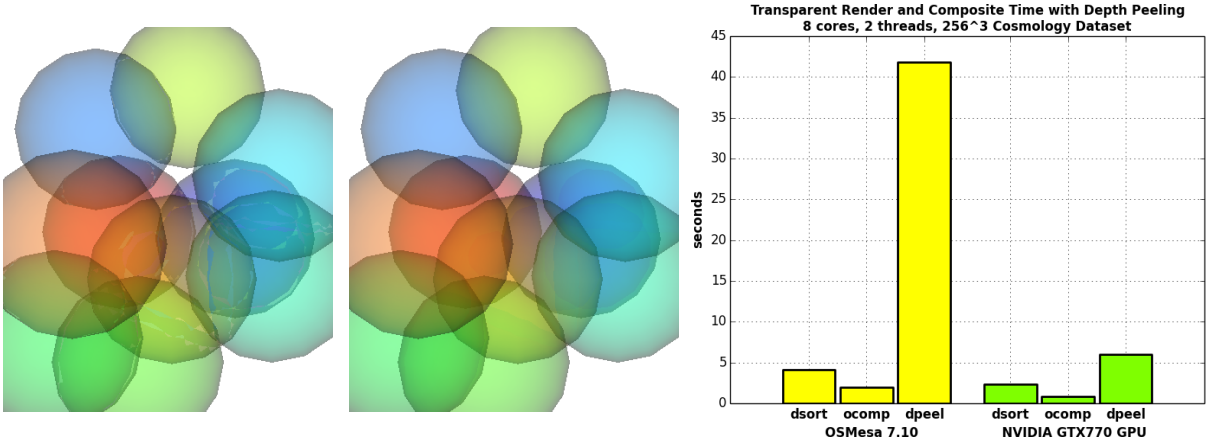


Figure 5: **Left:** Spheres rendered with camera order sort. **Right:** Spheres rendered with depth peeling. **Far Right:** Depth peeling performance.

- We added the capability to enable VTK's depth peeling algorithm from VisIt's GUI.
- We optimized code, cleaned up, and fixed bugs as we went.

3 Optimizations and Improvements

Ordered Compositing

Ordered compositing is an optimization for rendering translucent block decomposed data. When the decomposition's block's bounds are mutually disjoint one can find a compositing order that produces the same result as a global sort without moving any data. Compared to the sort-first strategy rather than re-distributing geometry in a global sort, ordered compositing renders the data in place and uses alpha bend compositing to combing each processes rendering. Ordered compositing is faster when the amount of data to move in the global sort is large. An example is shown in figure 1. The data is rendered using the domain decomposition shown on the left. Our alpha blending compositer makes use of associativity of the alpha blending operation and structures the compositing in a binary tree. The compositing order and subsequent tree is shown in the table in figure 1. When the root of the tree is not rank 0, the final image needs to be transferred to rank 0.

Depth Sort

The primary mechanism for correctly rendering translucent geometry in VisIt is a camera order geometry sort, this is generally called depth sort. VisIt makes use of VTK's depth sort called vtkDepthSortPoly-Data. Our profiling of the VTK class revealed several opportunities which we capitalized on:

- replaced qsort with std::sort to take advantage of compiler in-lining
- replace GetCell with GetCellPoints. GetCell explicitly constructs a VTK cell object containing a number of sub-objects using virtual API. Because we only need the cell's points, constructing a cell object is over kill. The GetCellPoints API copies the point indices into an array, we then access the points in-place without copying them. This change requires templates, the use of which further improves the compiler ability to optimize.
- refactor the bounds computations so the compiler can vectorize it using SIMD instructions.
- Refactor vtkPolyData::BuildCells to avoid the virtual vtkDataSet API
- Move the non-virtual vtkPolyData::GetCellPoints into the header to enable in-lining
- Add a non-virtual vtkPolyData::GetCell to enable fast out of order transfer of cells from one dataset to another. With this method one may use memcpy cell by cell.

Depth Peeling

We investigated the use of depth peeling, a GPU accelerated algorithm that would eliminate the need for local camera order geometry sort during translucent rendering. There are two user modify-able parameters in VTK's implementation, number of peels and occlusion ratio. The occlusion ratio sets a threshold on the the fraction of pixels changed per peel below which the algorithm terminates. The number of peels sets the maximum number of peels to perform. When the occlusion ratio is set to zero all of the peels are performed. We found that for complex scenes 32 peels and a occlusion ratio of 0.01 were reasonable settings, and as expected when rendering simple scenes the non-zero occlusion ratio resulted in early termination and better performance.

Other Improvements

While implementing ordered compositing, writing our compositer, and refactoring existing code, we investigated the following other optimizations:

- SIMD vectorization of the depth and blending compositer kernels
- using threads to overlap communication and compositing
- eliminating memcpy where ever possible, including making better use of shallow copy of VTK objects
- eliminate read-back from OpenGL if not using the channel
- eliminating or reducing communication where ever possible
- only update sorting pipeline when dependent input parameters change
- use templates to avoid memcpy and conversion and improve compiler optimization

Most of these are what could be considered contemporary "best practices" for C++ development.

4 Results

We benchmarked our work on two datasets, a 256^3 cosmology simulation and a $2000^2 \times 800$ plasma simulations, on two systems, a 20 core graphics workstation using 8 cores and a Cray XC31 using 512 cores. The details can be found in the tables in figures 3 and 2. Our primary optimization, ordered compositing, generally produced substantial speed up. The results are dependent on the geometry size and the number of processes used. In our tests we achieved $\approx 2\times$ speed up on the workstation and $\approx 4\times$ speed up on the Cray. In general during the opaque pass we achieved a modest speed-up of on the order of 10%. However, when both rendering passes were active (because there is both opaque and translucent geometry), during the opaque pass we achieved $\approx 11\times$ speed up on the workstation using 8 cores, and $\approx 50\times$ speed-up on the Cray using 512 cores. This is due to a bug we fixed that was causing the translucent geometry to be sorted (global+local) twice. On the workstation we looked at the performance both using OSMesa a software OpenGL implementation and using a hardware accelerated OpenGL using an NVIDIA GTX770 card. The speedup is higher on the GPU because rendering time is a significant portion of the total time, meaning improvements there magnify the impact of the other optimizations.

Depth Peeling

Unfortunately we found VTK's depth peeling algorithm to be substantially slower than the local geometry sort. The results are shown right most panel of figure 5.

Despite its poor performance depth peeling is a useful feature because VTK doesn't split intersecting geometry. The left two panels of figure 5 show a comparison of rendering using local geometry sort on the left and depth peeling on the right. Artifacts from intersecting geometry are present in the locally sorted rendering. Depth peeling is slower but produces the correct result. For this reason it is a useful feature to have available.

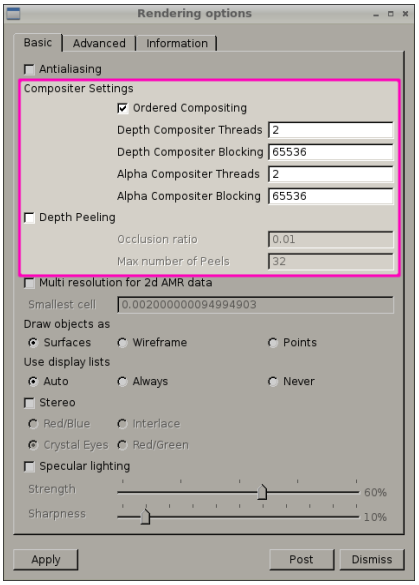


Figure 6: Additions to VisIt's GUI controlling ordered compositing and depth peeling.

5 VisIt integration:

We added controls in VisIt's GUI for the new features. These are shown in figure 6. Ordered compositing is enabled by default but can be optionally disabled. Depth peeling is disabled by default be can be optionally enabled.

6 Conclusion

Compared to posthoc rendering, insitu rendering often generates larger numbers of images, as a result rendering performance and scalability are critical in the insitu setting. We implemented improvements to VisIt's rendering and compositing infrastructure that deliver increased performance and scalability in both posthoc and insitu settings. Specifically we have added alpha bend compositing capability to VisIt and applied the well known ordered compositing optimization for transparent rendering, optimized the camera order sorting algorithm in VTK, and have addressed a number of performance and efficiency issues in VisIt. The impact is that VisIt performs is better prepared for insitu rendering work loads.