Eurographics Symposium on Parallel Graphics and Visualization (2013)
F. Marton and K. Moreland (Editors)

# In Situ Pathtube Visualization with Explorable Images

Yucong Ye, Robert Miller, and Kwan-Liu Ma

University of California, Davis

**Abstract**

*In situ processing is considered to be the most plausible data analysis and visualization solution for extreme-scale simulations. Explorable images were introduced as an in situ visualization method to enable interactive exploration of scalar field data without need for access to the massive original data and a powerful computer. We present a technique for in situ generation of explorable images for the visualization of vector field data without incurring additional inter-processor communication during simulation. We demonstrate this technique for pathtube generation on a variety of large datasets. The resulting pathtube visualization succinctly captures the flow structure over the full time span of the simulation. Users may explore the vector field structure through the generated images by changing the view angle, generating block cutaways, adjusting lighting, or changing transfer functions to recolor pathtubes or provide partial transparency.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Viewing algorithms Computer Graphics [I.3.1]: Hardware Architecture—Parallel Processing

## 1. Introduction

In recent years, the computational power of supercomputers has not been matched by the other resources required by the simulations running on those systems. As a result, simulations running on these systems are often not limited by processor speed, but instead are limited by available communication, I/O, or memory requirements. One strategy is thus to trade small computational time for reduced communication, I/O, or storage space. The scientists who use these supercomputers often need to visualize the results of their simulations, so in situ visualization and data reduction has become a popular way to make use of some of the extra computational power [Ma09]. There are, however, some difficulties in using in situ visualization effectively.

Most of the early in situ visualization works render static images of the simulation output from some prespecified angle, using prespecified lighting and transfer function parameters. The inherent problem here is that it is difficult to know a priori what parameters will provide useful visualizations before the simulation has been run. As a result, some users first run a subset of the simulation to determine useful visualization parameters, then repeat the simulation with these new parameters for a final rendering. For exploration of the simulation space, either multiple simulation runs, direct storage of geometry for rendering, or often heavyweight run-time visualization and interaction techniques may be necessary. While each of these solutions has its place, each may be expensive with respect to I/O or computational resources.

An alternative approach is to generate several static images on each simulation run, using different visualization parameters for each. It is common, for instance, to render several different views of a simulation from different angles. This often addresses some of the aforementioned problems, but in addition to the excessive cost there are still cases where none of the generated images are valuable, especially when searching for useful transfer function parameters. To resolve these problems, we extend a technique based upon *explorable images* [TCM10a, TCM10b], which was shown effective for visualizing scalar field data, to defer specification of relevant visualization parameters for visualizing vector field data until after the simulation run is complete. Specifically, we compute pathtube visualization for the vector field and make it explorable. We would like to let the users adjust view angle, transfer function, and lighting conditions after simulation completion, without reference to the full simulation data or re-simulation. We would also like to provide spatial cutaways to reveal internal structure.

Inter-processor communication and I/O are often the critical resource bottlenecks for modern simulations, so in situ visualizations should require as little communication and I/O

DOI: 10.2312/EGPGV/EGPGV13/009-016

as possible. Therefore, the ideal case is for each node to compute visualization of its data individually, and the visualization should request minimal information from other nodes. Compression should be used whenever possible to reduce cost of data transfer. Combination of these results into a single visualization should be deferred until after the simulation completes and communication and I/O are no longer scarce.

Parallel particle pathlines and pathtubes are examples of common visualizations that can have high requirements for storing output geometry as compared with static images. We show that such pathtubes can be rendered in parallel with no additional inter-processor communication requirements given an existing particle tracer, under the assumption that the simulation subdivides space as outlined previously. Pathtubes are also an example case where determination of good visualization parameters can be difficult due to high occlusion.

To summarize, we contribute the following:

- A method for in situ visualization that renders pathtubes to generate explorable images with low communication, storage, memory and computational requirements.
- An exploratory technique to analyze pre-rendered vector field data from different viewpoints, recolor by different properties, vary lighting conditions, and enable partial transparency and cutaways.

## 2. Related Work

Our explorable images can be generated in a classic postprocessing style, but we believe they are most useful if generated in situ with simulations. In situ visualization can often provide great benefits at little cost [ASM*11]. In situ visualizations can be roughly divided into the following two types:

- Tightly-coupled synchronous: In this method, the visualization and simulation share CPUs and main memory. This method generates the visualization as the simulation is running, and can thus be used for simulation monitoring. Additionally, it can reduce data output size, leading to lower I/O and storage requirements. [YWM08]
- Loosely-coupled asynchronous: In this method, the visualization and simulation run on separate compute and storage resources. This has the advantage of specialized hardware for the different techniques, but data must be transferred to the separate storage space first. [LZKS09]

Whitlock, et al. provide a good overview of the the problem and a library to ease in situ integration with VisIt [WFM11]. Lofstead, et al. present a system called ADIOS which allows user-selection of efficient I/O methods while not restricting output file formats or identification of data for analysis, and which may be used as the basis for loosely-coupled asynchronous techniques [LZKS09]. Bennett, et al. use ADIOS for efficient data movement for in situ processing [BAB*12]. Kim, et al. also use ADIOS to demonstrate an in situ indexing technique for high-performance data access [KAC*11]. A variety of other frameworks useful for in situ visualization have also been recently developed, including EAVL, Dax, DIY and PISTON. [SMM*12]

Kageyama, et al. provide similar exploratory capabilities via generation of myriad in situ movies, which are then exported to a dataset on the order of 10TB. Our benefit over this method is greatly reduced storage and I/O cost [KY13].

Particle pathtubes can provide an effective visualization of time-varying flow fields [Gra85]. Efficient visualization of time varying flow fields remains an active area of research [YWM07]. One of the underlying assumptions of our technique is that many simulations subdivide space and assign subregions to nodes of the cluster. This approach is common enough that it has been used as an example of a parallel visualization pattern [KHP*11] [PRN*11].

Several approaches for efficient parallel rendering have been developed. Eilemann introduces an efficient sort-last composition algorithm [EP07]. Many researchers address reduction of I/O requirements for parallel rendering of volume data, including Yu [YMW04], Kim [KAC*11], Kendall [KHP*11], and Vishwanath [VHI*10]. Takeuchi provides improvements of the binary swap composition technique [TIH03], while Yu introduces a 2-3 swap technique [YWM08]. Peterka confirms that in situ composition techniques are scalable on distributed memory architectures [PYR*09]. For individual compute nodes, Wang demonstrates a method that composes of parallel renderings implicitly using modern graphics hardware [WLL*11].

To achieve high-speed rendering with modern graphics hardware on desktops, we make use of per-pixel displacement maps [KS01]. Levoy et al present lightfields which use dense sampling to obtain new viewpoints of models [LH96]. Others such as McMillan present methods to combine multiple images from the same viewpoint to produce environment maps [MB95]. Our chosen technique for view synthesis is similar to Shade's layered depth images [SGHS98]. Todt presents a light field technique using per-pixel depth to allow view angle changes [TRSK07]. Chan describes a technique for generating multiple viewpoints for animations [CSN07].

Curless and Levoy present a method for determining object surfaces given range information [CL96]. Chang, et al. describe a tree-based method to improve sampling to generate new viewpoints [CBL99]. Yamazaki, et al. describe an approach using an inverse volume rendering technique. [YMK06]. We perform mesh reconstruction of the point cloud in hardware using a triangle filtering technique in the geometry shader via a modification of a method proposed by Ha, et al [HRK12]. IBR methods exist to mitigate disocclusion errors, such as Mei's Occlusion Camera [MPS05]. Use of depth-map discontinuity is also possible [PA06].

Tikhonova, et al. present a method for visualization by proxy [TCM10b], which stores extra information into a proxy image which in turn can be processed into output images with different lighting and transfer functions, and which can slightly alter the view angle. Ma introduces the concept of explorable images [MTC10]. Tikhonova later demonstrated explorable images for volume visualization [TCM10a], where multiple images of a volume from the same viewpoint with different transfer function values are layered into a single image. In other work, Tikhonova demonstrates the use of ray attenuation functions to store compact representations of large volume datasets to allow for later alteration of transfer functions [TCM10b].

## 3. Explorable Image Generation

To generate pathtube images, we perform each of the following substeps per node:

- The particle tracer sends the data for two timesteps of each of this node's particles.
- We construct tube segments between these particle positions, capped by spheres to prevent artifacts.
- These segments are rendered into framebuffers stored locally on the node, with several framebuffers being necessary for each camera position.
- When time to render the next frame, the particle tracer sends the data for each particle at the next timestep. The above steps are then repeated, and in aggregate the segments form cohesive pathtubes (See Figure 3)
- When the final timestep completes, the framebuffers are written to disk and archived as a single explorable image.

### 3.1. Explorable Image Format

Each node on the server renders to several different framebuffers as shown in Figure 1, which we will denote as the scattered images. These scattered images are then composited into a single explorable image in a customized PNG format, which we will denote as a multilayered PNG format. The multilayered PNG format is designed to achieve two main goals. First, when viewed in a standard PNG viewer, it should be viewable as a standard image. Second, when viewed in our specialized client, full exploratory capabilities should be exposed. While other formats such as TIFF would be suitable for this kind of extension, we choose PNG for its portability, ease of extension, and compression performance.

#### 3.1.1. Scattered Images

In the traditional sort-last parallel rendering composition technique, after all nodes have completed rendering the image there comes a final composition step where the images from all nodes are composited in front-to-back order so that in the end a correct view is established from the camera viewpoint. Due to occlusion from regions closer to the camera, this will discard information about the regions processed by more distant nodes, especially for dense images with high occlusion. Instead, we defer this composition step until analysis takes place on a standard desktop. In this way, we are able to preserve some of the occluded information, but this comes at the cost of higher storage. Each set of scattered images contains camera configurations and a set of image buffers. Each buffer enables certain exploration options.

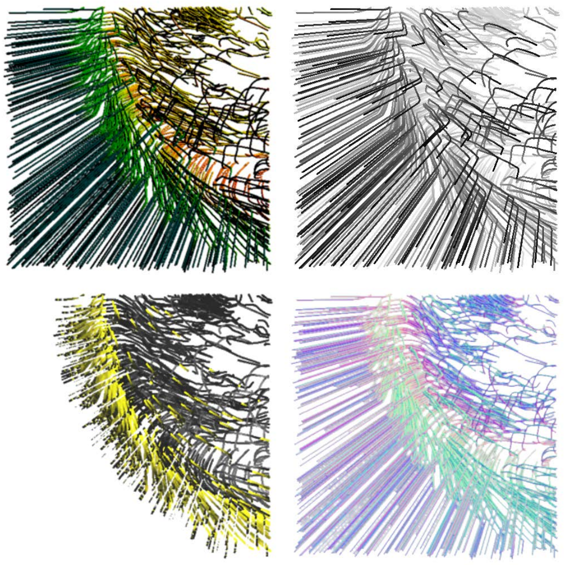
Figure 1: *Scattered Images: The color buffer (top left), depth buffer (top right), property buffers (bottom left), and normal buffers (bottom right) each provide exploratory capabilities.*

- Color buffer: RGB information of each pixel. This is the view given in standard PNG viewers.
- Normal buffer: Compressed representation of the normal vector of the rendered surfaces at each pixel. When present, these normals, combined with the depth map, allow for relighting of pre-rendered surfaces.
- Depth buffer: Depth from the camera to the surface at each pixel. Depth is local to each node's domain for greater accuracy. This allows view-angle changes, multiple camera position integration and arbitrary cutaways.
- Property buffer: This buffer may store any other per-pixel property and is used as input to a transfer function that the user may alter interactively during analysis.

Each framebuffer is generated independently per node. After completion of the final timestep, these are written as independent PNG files. In total, these images are much smaller than storage of dense geometry from particle tracing data, but it is difficult to manage them in this form because of their scattered nature, so we perform a global composition to organize them into a single explorable image. This allows improved compression, choice of maximum sample depth, and choice of desired exploration options for the end user.

We perform a global composition and depth-reordering before storing the final result, as shown in Figure 2. The global composition combines each scattered image into a global viewport, then depth reordering is performed by constructing a list of samples per-pixel in the global viewport, sorted by depth.

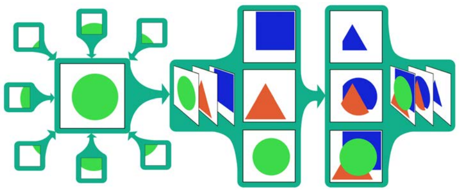

Figure 2: *Global composition and reordering: Global composition, shown left, composites the scattered images into cohesive layers. These layers are then partitioned into new layers based on visibility for compact storage, shown right.*

### 3.1.2. Multilayered PNG Format

After global composition and depth-reordering, the resulting layers for each framebuffer stored in the multilayered PNG format, which extends the standard PNG format. The color buffer of the front layer is stored as normal in the PNG format and is thus visible in standard viewers. The remaining layers are encoded into custom chunks as supported by the PNG specification. Each custom chunk contains relevant metadata such as camera orientation and buffer type, along with the pixel data for the associated buffer. When viewed in our specialized client, we detect these buffers and enable the associated exploratory capabilities for the image.

In addition to distribution in a format usable in standard viewers, we benefit from existing image compression techniques in the PNG format, and allow the user to choose whether explorability or storage size is more important via activation or deactivation of the different framebuffers.

If deactivation of a buffer is not desirable, the bit depth of an image buffer may be reduced. Combination of some buffers is also possible, such as encoding a property buffer into the color buffer with a reversible transfer function.

## 3.2. Pathtube Generation

Many vector field simulations operate by performing a spatial subdivision and assigning regions of space to each compute node. In this approach, the particle tracer for each node is assigned some initial population. The tracer then updates the position and other properties of these particles between timesteps. When particles leave the spatial region assigned to the node, they are passed to the appropriate neighbor nodes for further processing.

The standard approach for generating pathlines is to periodically store particle positions and any necessary attributes. This generates a representation of geometry with a size that is linear in the number of particles and in the number of timesteps. By comparison, image-based approaches have a constant maximum size. For sparse sets, our explorable images are closer to the former case, whereas for denser sets our approach converges to the latter.

To render pathtubes, we require the particle tracer to periodically pass two sets of particles to our pathtube generator, which are the particles from the previous timestep

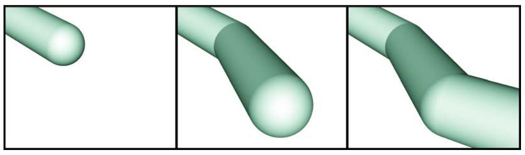

Figure 3: *At each timestep, a capped tube segment from the previous timestep to the current timestep is rendered. In aggregate, these segments form the complete pathtube.*

and the particles from the current timestep. Each particle should include its position, radius and any properties for the output property buffers. A spherically capped cylinder between these two locations is then rendered into our framebuffer. We preserve this framebuffer and the associated depth map for use when rendering subsequent timesteps. Once all timesteps have been rendered, this produces an image of the pathtubes generated by each node from all requested viewpoints. Currently software rendering is used, but this approach should also work well with hardware rendering.

## 4. Explorable Image Interaction (client-side)

When explorable images are loaded with our specialized viewer, we support two visualization techniques and several exploratory capabilities.

## 4.1. Visualization Methods

The two image-based rendering visualization methods we support are point-sprite cloud generation and mesh regeneration via triangle filtering. Each offers significant exploratory capability and good image quality, but each also comes with some associated artifacts. Both methods begin by first projecting the pixels back into their 3D positions using the depth buffers and the stored camera position.

### 4.1.1. Point-sprite Cloud

The point-sprite cloud is a standard IBR method, in which we directly render the projected pixels as points, or sprites. At the original zoom level, the point-sprite cloud rendering approach works well, because gaps between points are not visible at this level. However, as the user zooms in, these gaps become apparent. A standard IBR solution for this problem is to simply scale up the point-sprites based on the zoom level and distance to cover these gaps and give the appearance of a cohesive surface. This tends to give the surface a somewhat blocky appearance. Changes to the view angle make the disconnects especially apparent, but this can be largely resolved via the storage of multiple camera angles.

### 4.1.2. Triangle Filtering

In our other visualization method, we attempt to regenerate the mesh by connecting pixels from connected surfaces to form triangles. To determine which pixels should be connected, we can store the ID of each tube into a property map. Then we can generate triangles based on the IDs of the corners of a pixel quad, as shown in Figure 4. In order to save GPU memory and increase rendering speed, we perform the triangle filtering in the geometry shader. As input to the shader, we need all the pixel properties and a base mesh, which is a rectangular grid of triangles.

To perform the triangle filtering, we use the ID map to break the original connectivities in the bash mesh and then reconnect the vertices according to the IDs, as shown in Figure 4. Finally we compute the vertex positions and apply the transfer function in the fragment shader.

Reconstructing a mesh eliminates the gap artifacts in the point-sprite cloud version with a cost of higher computational cost. In addition, triangle filtering introduces other types of artifacts such as discontinuities between discrete pathtubes and incorrect connectivities with spiral pathtubes.

### 4.2. Exploration Techniques

The exploration techniques we support are transfer function modification, view angle changes, relighting, and cutaways.

#### 4.2.1. Transfer Function Modification

By including a transfer function editor and making use of the property buffers in the explorable image, we allow user-defined recoloring. Although transparency is not supported in the initial in situ rendering, it is supported during user analysis. Transfer functions may be applied independently for each property buffer, but each additional property buffer linearly increases the size of the explorable image.

#### 4.2.2. Variant Viewing Angle and Illumination Angle

We use the quaternion camera model to allow users to observe the volume from any arbitrary angle, but the quality of the altered view is limited to the information available from the original viewpoints within the image. In addition to view angle modification, the normal buffers within the image allow relighting of rendered surfaces. We accomplish this local relighting in the fragment shader via the phong model.

#### 4.2.3. Cutaways

Users may define arbitrary cutting planes, as shown in Figure 5. Quality is best when cutting planes are near node region boundaries due to fewer disocclusion errors.

## 5. Results

### 5.1. In Situ Performance

Our visualization code is not capable of adding inter-node communication. No I/O is required except for the storage of the completed image upon completion of the simulation.

Our memory requirements are linearly related to the resolution of the output images. In our technique, the only memory used is the memory necessary to store the image buffers in each node. If there are 6 image buffers (a standard case

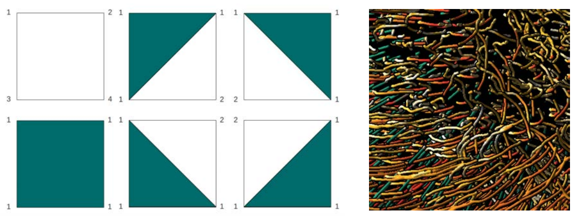

Figure 4: *Left: The 6 cases of the triangle filtering technique, where the corner numbers are the IDs. Right: Regenerated mesh with triangle filtering and the associated halo effect.*

for full exploration), the memory requirement is 6 times the size of a single image buffer. In the supernova case, 512x512 resolution was used for each node's image buffer with nodes in a 4x4x4 grid, leading to a global resolution of 2048x2048. As a result, $6*512*512*4 = 6,291,456$ memory bytes are required above the original simulation requirements. Note that this value remains constant throughout the simulation.

The required storage I/O is determined by the size of the scattered images from all the computing nodes. Without compression, required I/O would be identical to memory usage. However, compression via the PNG format greatly reduces necessary I/O. In the supernova case with 64 processes, 6 images per process, image resolution of 512x512 per node, and 32 particles per node, the total size of all output images collected after compression is 26.7 MB. Without the PNG compression, it would have become 384 MB.

We have measured the time performance of our visualization code on the XSEDE supercomputer (see Table 2) and JAGUAR (see Table 1). We can observe the same conclusions from both tables: The performance per frame linearly correlates to the number of particles. The resolution of the output image also definitely affects the performance, but this is not as strong a connection as the number of particles rendered. There is significant variance in node timings because the workload of this simulation is not well balanced.

### 5.2. Composition Results

All generated images are viewable in static form in standard PNG viewers. If we choose to combine the depth and normal buffers, we can reduce the size of the scattered images in the supernova case from 26.7 MB, to 16.0 MB.

### 5.3. Exploration Technique Results

Our explorable images provide several distinct kinds of exploration: Transfer function editing, cutaways, relighting, and view angle changes. We will now present our images' efficacy for each kind of exploration.

As shown in the leftmost section of Figure 5, careless selection of the initial transfer function can render some in situ results useless. In this case, the structure of the supernova

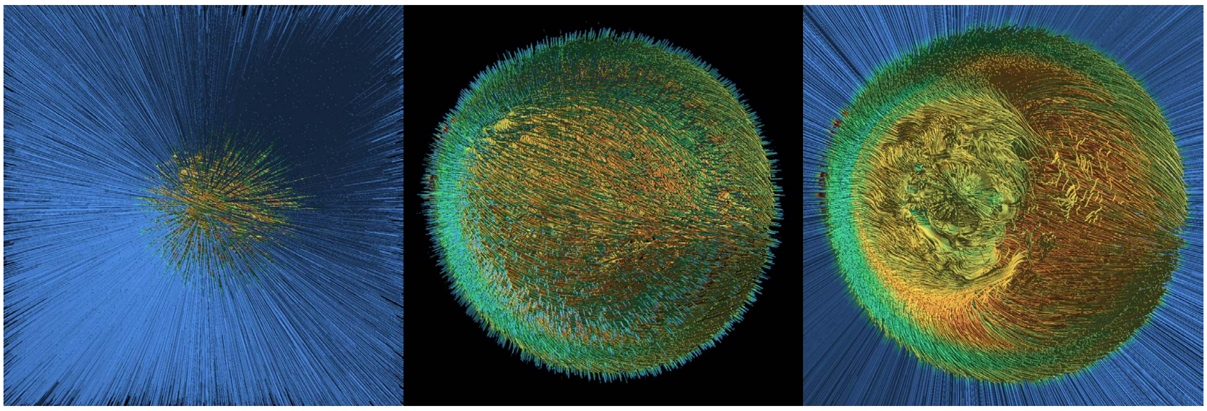

Figure 5: *No exploration (left), transfer function modification (middle), and cutaways (right). Images contain 25600 pathtubes.*

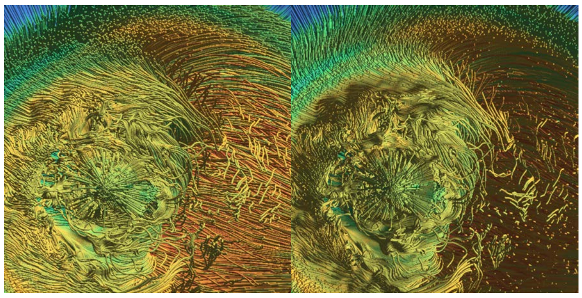

Figure 6: *Zoomed in view of the supernova core: In the left image, we have the light source at the camera position. All surfaces are well lit, but this creates a cluttered image. Moving the light (right) helps highlight spatial relationships.*

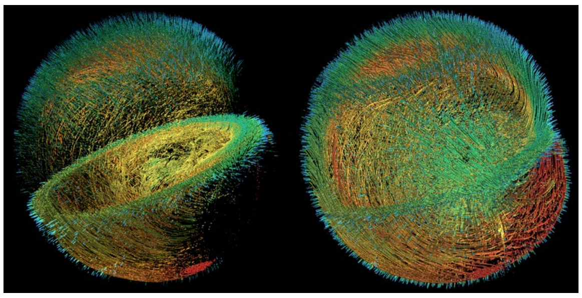

Figure 7: *The left image shows how dense datasets are not adequately represented by use of a single camera. On the right, we added another camera angle to the explorable image which reveals features not captured by the first camera.*

has been mostly obscured. through modification of the transfer function, we may edit the transparency of the external pathtubes as shown in the middle section of Figure 5.

Another important tool for dealing with occlusion in images are cut-planes, or cutaways. In the rightmost section of Figure 5, we cut directly into the supernova core to reveal additional detail. The quality of such cutaways depends on the density of the rendered data and on how finely the nodes are assigned regions for tracing, but in practice images can become quite dense before significant artifacts are apparent.

Lighting in an image can play a critical role in understanding the spatial relationships between objects. Consider Figure 6: A standard approach of placing the light at the camera position is employed in the left image, equally illuminating the entire image. By modifying the light position in the right image, we emphasize structures of interest.

View-angle modification is difficult with image-based techniques. The left part of Figure 7 demonstrates the artifacts visible with a large angle rotation for an explorable image with only one camera angle. As the image on the right shows, adding camera angles partially alleviates this issue.

Sparse images, such as the combustion dataset shown in Figure 8, are far better suited for large angle rotations. This image contains two diametrically opposed camera positions. The right image shows the same view rotated 90 degrees without the need for additional camera angles.

## 6. Discussion

Visualization on supercomputers is sometimes divided into the two categories of in situ visualization and a posteriori visualization. The former has the advantages of low I/O and storage requirements, but makes data exploration without rerunning the simulation difficult. The latter provides full exploration of simulation results, but imposes very high storage requirements which may then require significant I/O, which is already often the bottleneck for parallel simulations. Explorable images provide a middle-ground with the advantages of each, but with highly mitigated downsides.

Our explorable images allow for many common data exploration techniques, but require more storage space than a standard image. We have found that the additional storage required remains low enough to be negligible for supercomputer storage systems. Consider Figure 9: Static images have

| Total Particles | Node Count | Particle per Node | Total Resolution | Resolution per Node | Ttl. Mean (ms/frame) | Total Std. (ms/frame) | Total Min (ms/frame) | Total Max (ms/frame) |
|---|---|---|---|---|---|---|---|---|
| 4096 | 4096 | 1 | 1024x1024 | 64x64 | 22.7517 | 10.5985 | 0 | 71.6556 |
| 4096 | 4096 | 1 | 2048x2048 | 128x128 | 30.5011 | 14.1962 | 0 | 96.4901 |
| 65536 | 4096 | 16 | 1024x1024 | 64x64 | 384.601 | 78.1439 | 67.8146 | 578.543 |
| 65536 | 4096 | 16 | 2048x2048 | 128x128 | 411.506 | 84.0779 | 69.2053 | 659.073 |
| 65536 | 4096 | 16 | 4096x4096 | 256x256 | 460.257 | 94.9946 | 83.7748 | 747.815 |
| 65536 | 4096 | 16 | 8192x8192 | 512x512 | 528.287 | 104.228 | 97.4172 | 796.954 |
| 131072 | 4096 | 32 | 2048x2048 | 128x128 | 830.407 | 158.797 | 161.258 | 1191.66 |
| 131072 | 4096 | 32 | 4096x4096 | 256x256 | 936.168 | 178.931 | 176.755 | 1397.28 |

Table 1: *Time performance measured on Jaguar, Oak Ridge with the combustion data set.*

| Total Particles | Node Count | Particle per Node | Total Resolution | Resolution per Node | Ttl. Mean (ms/frame) | Total Std. (ms/frame) | Total Min (ms/frame) | Total Max (ms/frame) |
|---|---|---|---|---|---|---|---|---|
| 4096 | 4096 | 1 | 1024x1024 | 64x64 | 3.0772 | 1.7338 | 0 | 12.2 |
| 4096 | 4096 | 1 | 2048x2048 | 128x128 | 3.4423 | 1.9338 | 0 | 13 |
| 65536 | 4096 | 16 | 1024x1024 | 64x64 | 47.7833 | 16.9476 | 8.4 | 141.5 |
| 65536 | 4096 | 16 | 2048x2048 | 128x128 | 53.6628 | 19.0581 | 9.7 | 161.3 |
| 65536 | 4096 | 16 | 4096x4096 | 256x256 | 62.9508 | 22.1502 | 12.2 | 193.5 |
| 65536 | 4096 | 16 | 8192x8192 | 512x512 | 80.1056 | 27.5718 | 14.6 | 218.3 |
| 131072 | 4096 | 32 | 2048x2048 | 128x128 | 106.8254 | 37.0618 | 22.7 | 300.8 |
| 131072 | 4096 | 32 | 4096x4096 | 256x256 | 124.3466 | 42.6937 | 25.4 | 358.7 |
| 262144 | 4096 | 64 | 8192x8192 | 512x512 | 230.5014 | 12.0475 | 170.3 | 278.3 |

Table 2: Time performance measured on Stampede, XSEDE with the supernova data set.

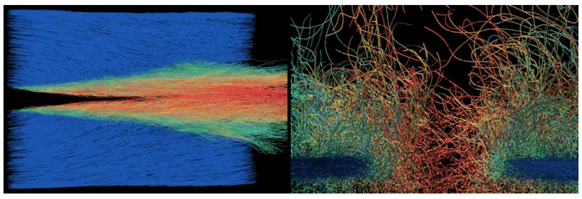

Figure 8: *Large-angle rotation on sparse dataset. Far fewer artifacts are visible for rotation of sparser images.*

a constant maximum size for their resolution which depends on the complexity and density of the data being rendered; this property is retained by our explorable images. Storage space for raw (or compressed) geometry, on the other hand, tends to scale linearly with the number of elements rendered.

Our approach works well for cases of low occlusion, such as pathlines and pathtubes. The approach is less suitable for other surfaces due to the higher likelihood of occlusion.

## 7. Conclusions and Future Work

We present a new technique for explorable image generation of pathtubes, which is suitable for in situ visualization. Our technique provides a better overview of the output data than static images while incurring no additional I/O cost other than final storage and minimal additional storage cost. Additionally, our technique requires very little memory or computational overhead, and scales well to large node counts.

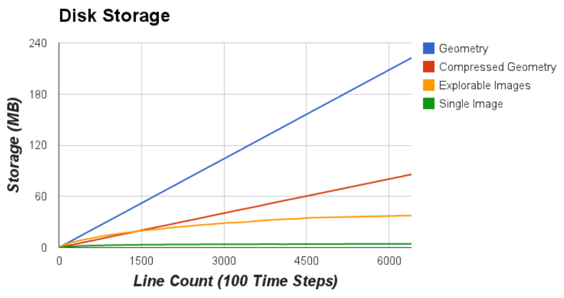

Figure 9: *Storage comparison among regular geometry, compressed geometry, explorable image, and single image. As images become more dense, the size of explorable images converges to a maximum size.*

We also present a modification of the PNG image format to allow distribution of the explorable images in a format viewable in a standard PNG viewer. Combined with our viewer, this allows for view angle changes, relighting, user-defined transfer functions, zooming, and cutaways.

For future work, we will integrate our technique with a better quality, externally developed simulation. Then we will try alternative mesh generation techniques for better integration of multiple cameras, and develop methods to extend the technique to integrate both scalar and vector fields. Finally, we will provide methods to measure uncertainty of the output during exploration.

## 8. Acknowledgements

This work has been supported in part by the U.S. National Science Foundation through grants OCI-0749227, CCF-0811422, OCI-0850566, and OCI-0905008, and also by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FC02-12ER26072, program manager Lucy Nowell.

## References

[ASM*11] AHERN S., SHOSHANI A., MA K.-L., CHOUDHARY A., CRITCHLOW T., KLASKY S., PASCUCCI V., AHRENS J., BETHEL E., CHILDS H., ET AL.: Scientific discovery at the exascale. report from the doe ascr 2011 workshop on exascale data management. *Analysis, and Visualization* (2011). 2

[BAB*12] BENNETT J. C., ABBASI H., BREMER P.-T., GROUT R., GYULASSY A., JIN T., KLASKY S., KOLLA H., PARASHAR M., PASCUCCI V., PEBAY P., THOMPSON D., YU H., ZHANG F., CHEN J.: Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *SC '12* (2012), IEEE Computer Society Press, pp. 49:1–49:9. 2

[CBL99] CHANG C.-F., BISHOP G., LASTRA A.: Ldi tree: a hierarchical representation for image-based rendering. In *SIGGRAPH '99* (1999), ACM, pp. 291–298. 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *SIGGRAPH '96* (1996), ACM, pp. 303–312. 2

[CSN07] CHAN S., SHUM H.-Y., NG K.-T.: Image-based rendering and synthesis. *Signal Processing Magazine, IEEE 24*, 6 (Nov. 2007), 22 –33. 2

[EP07] EILEMANN S., PAJAROLA R.: Direct send compositing for parallel sort-last rendering. Favre J. M., Santos L. P., Reiners D., (Eds.), EGPGV '07, Eurographics Association, pp. 29–36. 2

[Gra85] GRANGER R.: *Fluid Mechanics*. Dover Classics of Science and Mathematics. Dover Publications, 1985. 2

[HRK12] HA I., RHEE T., KIM J.: Smooth mesh generation from noisy depth image. In *GCCE '12* (Oct. 2012), pp. 495 –497. 2

[KAC*11] KIM J., ABBASI H., CHACON L., DOCAN C., KLASKY S., LIU Q., PODHORSZKI N., SHOSHANI A., WU K.: Parallel in situ indexing for data-intensive computing. In *LDAV '11* (Oct. 2011), pp. 65 –72. 2

[KHP*11] KENDALL W., HUANG J., PETERKA T., LATHAM R., ROSS R.: Toward a general i/o layer for parallel-visualization applications. *Computer Graphics and Applications, IEEE 31*, 6 (Nov.-Dec. 2011), 6 –10. 2

[KS01] KAUTZ J., SEIDEL H.-P.: Hardware accelerated displacement mapping for image based rendering. In *GRIN'01* (2001), CIPS, pp. 61–70. 2

[KY13] KAGEYAMA A., YAMADA T.: An approach to exascale visualization: Interactive viewing of in-situ visualization. *arXiv preprint arXiv:1301.4546* (2013). 2

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH '96* (1996), ACM, pp. 31–42. 2

[LZKS09] LOFSTEAD J., ZHENG F., KLASKY S., SCHWAN K.: Adaptable, metadata rich io methods for portable high performance io. In *IPDPS '09* (May 2009), pp. 1 –10. 2

[Ma09] MA K.-L.: In situ visualization at extreme scale: Challenges and opportunities. *Computer Graphics and Applications, IEEE 29*, 6 (Nov.-Dec. 2009), 14 –19. 1

[MB95] MCMILLAN L., BISHOP G.: Plenoptic modeling: an image-based rendering system. SIGGRAPH '95, ACM. 2

[MPS05] MEI C., POPESCU V., SACKS E.: The occlusion camera. *Computer Graphics Forum 24*, 3 (2005), 335–342. 2

[MTC10] MA K.-L., TIKHONOVA A., CORREA C. D.: Distance visualization of ultrascale data with explorable images. In *ACM SIGGRAPH 2010 Talks* (2010), SIGGRAPH '10, ACM, p. 9. 3

[PA06] POPESCU V., ALIAGA D.: The depth discontinuity occlusion camera. In *I3D '06* (2006), ACM, pp. 139–143. 2

[PRN*11] PETERKA T., ROSS R., NOUANESENGSY B., LEE T.-Y., SHEN H.-W., KENDALL W., HUANG J.: A study of parallel particle tracing for steady-state and time-varying flow fields. In *IPDPS '11* (May 2011), pp. 580 –591. 2

[PYR*09] PETERKA T., YU H., ROSS R., MA K.-L., LATHAM R.: End-to-end study of parallel volume rendering on the ibm blue gene/p. In *ICPP '09* (Sept. 2009), pp. 566 –573. 2

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *SIGGRAPH '98* (1998), ACM. 2

[SMM*12] SEWELL C., MEREDITH J., MORELAND K., PETERKA T., DEMARLE D., LO L.-T., AHRENS J., MAYNARD R., GEVECI B.: *The SDAV Software Frameworks for Visualization and Analysis on Next-Generation Multi-Core and Many-Core Architectures*. Tech. rep., LANL, 2012. 2

[TCM10a] TIKHONOVA A., CORREA C., MA K.-L.: Explorable images for visualizing volume data. In *PacificVis '10* (Mar. 2010), pp. 177 –184. 1, 3

[TCM10b] TIKHONOVA A., CORREA C., MA K.-L.: Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (Nov.-Dec. 2010), 1551 –1559. 1, 2, 3

[TIH03] TAKEUCHI A., INO F., HAGIHARA K.: An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. *Parallel Computing 29*, 11-12 (2003), 1745 – 1762. 2

[TRSK07] TODT S., REZK-SALAMA C., KOLB A.: *Fast (spherical) light field rendering with per-pixel depth*. Tech. rep., Technical report, University of Siegen, Germany, 2007. 5, 2007. 2

[VHI*10] VISHWANATH V., HERELD M., ISKRA K., KIMPE D., MOROZOV V., PAPKA M., ROSS R., YOSHII K.: Accelerating i/o forwarding in ibm blue gene/p systems. SC '10. 2

[WFM11] WHITLOCK B., FAVRE J. M., MEREDITH J. S.: Parallel in situ coupling of simulation with a fully featured visualization system. EGPGV '11, Eurographics Association. 2

[WLL*11] WANG P., LIU H., LI S., ZENG L., CAI X.: Cad/graphics '11. pp. 103 –107. 2

[YMK06] YAMAZAKI S., MOCHIMARU M., KANADE T.: Inverse volume rendering approach to 3d reconstruction from multiple images. In *Computer Vision - ACCV 2006*, Narayanan P., Nayar S., Shum H.-Y., (Eds.), vol. 3851 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 409–418. 2

[YMW04] YU H., MA K.-L., WELLING J.: I/o strategies for parallel rendering of large time-varying volume data. In *EGPGV '04* (2004), vol. 4, pp. 31–40. 2

[YWM07] YU H., WANG C., MA K.-L.: Parallel hierarchical visualization of large time-varying 3d vector fields. In *SC '07* (Nov. 2007), pp. 1 –12. 2

[YWM08] YU H., WANG C., MA K.-L.: Massively parallel volume rendering using 2–3 swap image compositing. In *SC '08* (Nov. 2008), pp. 1 –11. 2