In Situ Volumetric Data Visualization with Feature Tracking using Explorable Images

Category: Research

ABSTRACT

Parallel numerical simulation is a powerful tool used by scientists to study complex problems. It has been a common practice to save the simulation output to disk and then conduct in-depth analyses of the saved data after the simulation is over. System I/O capabilities have not kept pace as simulations have scaled up over time, so a common solution has been to output only subsets of the data to reduce I/O. However, as we are entering peta- and exa-scale computing, the sub-sampling approach is becoming an unreasonable solution to the I/O bottleneck. In situ visualization/analysis is a promising way to address this issue by reducing storage requirements. We present a novel approach using explorable images to provide volumetric data visualization. We extend previous explorable images to allow feature extraction and tracking in addition to standard exploration in the data space. Additionally, we make use of knowledge of features-of-interest specified by user interaction with feature tracking to refine explorable images later in the simulation.

Index Terms: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture;

1 INTRODUCTION

As simulations enter the exa-scale, researchers can no longer depend on the traditional method of simply saving raw data occasionally, then performing visualization as a post-process. The portion of the raw simulation data that can be feasibly stored for post-processing has simply become too small as computational capabilities on large parallel systems have outstripped the available I/O. Maximally efficient use of this I/O has therefore become critical. Ultimately, it has become necessary to find ways to process the raw data before output. The goal is to capture as many interesting aspects of the raw data as possible, while still minimizing the final output size.

One technique to improve I/O efficiency has been to perform visualization in situ with the simulation. This method reduces the size of raw outputs to visualization outputs, which is often simply a set of images or videos. A disadvantage of this technique has been the loss of the capability for exploration of the output data: Because good visualization parameters are often not known a priori, a 'best guess' must be used. Poor parameter choices may result in wasted simulation time, so researchers must manually search for good visualization parameters via repetitive small-scale simulations, then hope that these parameters will still be appropriate at full-scale and full-length simulations. Alternatively, researchers may opt to simply generate many in situ images with a variety of settings in the hope that any relevant data is captured. Even so, the generated images are static, so after simulation completion, researchers are limited to these specific views of their results.

Visualization is not the only kind of data analysis that is possible to conduct in situ. In addition to obvious techniques like statistical analysis, more complicated tasks such as feature extraction and tracking are possible. This is an interesting case, as the output features being tracked are not especially relevant unless the raw simulation output containing these features is output as well, thus limiting the utility of such a technique.

Previous work with Explorable Images has combined the advantages of in situ visualization with some of the exploration capabilities that were possible with visualization as a post-process. In the case of volume visualization, explorable images allow exploration of transfer function modification and relighting. It has been demonstrated that the cost of explorable image generation scales well and has an insignificant cost in computing time as compared with the running simulation. However, thus far it has not been possible to adapt explorable image generation based on user interactivity. Specifically, there are cases where a user may know which portions of an explorable image are interesting during an early stage of the simulation.

Existing explorable image implementations do not provide a method for selecting these specific areas of interest, so the user must generate the entire image at the maximal sampling rate that may be necessary to fully capture any feature. Because of this, the majority of the size of an explorable image may be used to represent portions of the simulation that are known to be uninteresting, but are still necessarily captured at high detail. By providing maximal exploration only in the areas known to be interesting to the user, we may provide significant improvements in explorability without unnecessary increases in the size of the output.

We present an extension of Explorable Images that allows for feature extraction and tracking in volumetric data, thus allowing for feature exploration without incurring the storage cost of saving out raw simulation data. We further allow users to select features of interest in explorable images, then use these user interactions to feed information to the in situ generator to provide additional detail for these features of interest in the explorable images generated in later timesteps. To summarize, we make the following contributions:

- A non-intrusive library that can be easily incorporated into pre-existing simulation code bases.
- A posteriori feature extraction and tracking via explorable images without storage of raw simulation data.
- A method to use user input to determine features of interest, then feed this information back to the simulation to allow generation of additional detail in each area of interest.

2 RELATED WORK

Ahern el al. [1] provides a decent overview of the major challenges that the transition to exa-scale computing has brought. It describes the exa-scale from 3 perspectives: the available hardware and software support, the needs of scientists in different scientific areas, and some different approaches being researched to address the issues introduced by exa-scale computing. In chapter 6.1.1 of [1], the pros and cons of in situ processing are described. [10] describes the challenges and opportunities of in situ visualization in further detail.

The idea of in place visualization was first introduced in [9], in which a volume visualization is generated while a computational fluid dynamics simulation is run. The phrase 'in situ visualization' was first used in [11] to describe the in place relationship between visualization and simulations.

There are two types of in situ processing depending on where the data is being processed, as described in [11]

- Tightly-coupled synchronous: the data is processed using the same computing node as the simulation. The processing stage shares resources with the simulation, both processor time and memory, which means no data movement is required at all. However, it occupies some of the precious resources of simulations. As a result, tightly-coupled in situ processing requires the processing stage to be small. [24] is an example of tightly-coupled synchronous.

- Loosely-coupled asynchronous: the data is first transferred to another location over the network without touching the disk, then analysis/visualization is performed using different computing nodes. Moving the data is expensive, but this approach does not pause the simulation in order to perform analysis/visualization. In transit is usually used to describe loosely-coupled asynchronous, [13] gives a few examples of in transit visualization.

Both types of in situ processing have their advantages and disadvantages. In our case, our overhead is quite small when compared to the simulation, so we are using the tightly-coupled synchronous approach.

In situ generation of static images is a well-known technique. There are multiple libraries that enable in situ visualization, including ParaView Catalyst[2], VisIt and Damaris[4]. Static images represent the smallest possible data size output from in situ visualization, but staic images can present only a very small fraction of the original volume to the scientists. To address this limitation, a common solution is to generate multiple sets of static images with different settings. [8] generates images from many viewpoints. This apporach allows users to explore many perspectives around the original volume, but requires the storage of many static images, and can become prohibitive as a result. Explorable images provide an alternative to the storage of many discrete images for such purposes.

The concept of the explorable image was first introduced by Tikhonova in [19]. In this work, the development of the Ray Attenuation Function allows users to explore the transfer function domain in an image without resimulation. In [18, 20], Tikhonova further extends explorable images by introducing proxy images and a camera modification technique, which enables relighting and small view-angle manipulation.

In [21], Tikhonova presents an efficient technique for scalable parallel generation of explorable images. In this paper, Tikhonova notes that a common way to parallelize simulation is to have each process handle a sub-volume of the simulation. Therefore, sort-last parallel rendering [5] is a natural approach.

Explorable image were also extended to flow field visualization. [23] uses an image based technique to in situ generate pathtubes of flow field simulation.

After rendering the images locally, a final composition is required to generate a global image. The two most common composition techniques are the 2-3 swap[25] and direct send [5]. The 2-3 swap requires multiple rounds of communication between processes but in each round only a few messages are received by each process. On the other hand, the direct send approach has only 1 communication round but each process must send an image to each other process, requiring n^2 messages. We implemented both the direct send and the 2-3 swap methods.

Feature-based data exploration is considered a key approach for the study of large volumetric data sets. Conventional approaches extract features from individual timesteps and then associate them between consecutive timesteps. More recent work utilizes either higher-dimensional iso-surfacing [7] or non-scalar representations of the data [17] for feature tracking, which requires no correspondence analysis over time. Prediction-adjustment based approaches [16, 3, 14, 22], on the other hand, first predict candidate regions based on the feature information (e.g. centroid location) extracted from previous timesteps, and then match or adjust the predicted region for correct feature tracking. Volumetric features can also be clustered based on similarity measures and track features of similar behavior in groups [15].

The prediction-based approach are appealing for their computing efficiency and the reliability in an interactive system. We generalize the prediction-correction based tracking approach used in [14] and [22].

3 METHODOLOGY

In this section, we first present the expected workflow for scientists that use our system. We will then describe in detail how to generate our extended explorable images, and how to use these extensions to enable feature extraction/tracking. Finally, we will explain how we can provide feedback to the simulation based on user interaction to enable adaptive detail in explorable images.

3.1 Workflow Overview

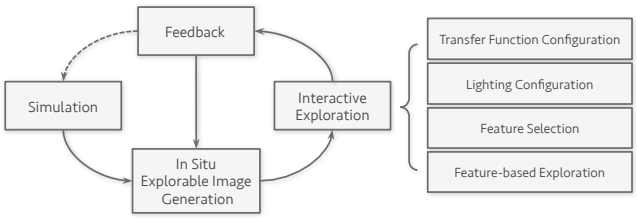


Figure 1: Workflow for data analysis of active simulation with explorable images. When a simulation is begun, we generate explorable images according to default settings. As users interact with these images, we feedback information about these interactions to the simulation, thus enabling higher detail on regions of user interest in future explorable images.

A primary goal in creating our in situ explorable image library was to allow for easy integration into simulations. Scientists may integrate our library by adding 3 small code snippets. Specifically, the library must be initialized, the volume data must be occasionally made available to the library, and the library must be occasionally commanded to output images. We make use of the popular CMake build system to help ensure that building our library in new environments is as simple as possible.

After our in situ library has been incorporated, explorable images will be generated as the simulation runs. Scientists can then use our special viewer to open and interact with the explorable images. Our explorable images enable scientists to change transfer function settings and lighting parameters. Furthermore, scientists may select features of interest by using our viewer. After each feature is selected, it will be tracked both forward and backward through time when the scientists explore the data temporally. Sometimes, the original visualization parameters might not reveal what scientists want to see, so our viewer allows the scientists to upload new settings to the simulation on the fly to ensure that future explorable images generated from the simulation will be using better settings.

3.2 In Situ Explorable Image Generation

In this section, we will briefly explain the explorable image tehcnique for Tikhonova's explorable images[19, 21, 20], then describe the extensions we have made to this technique.

3.2.1 IAF and Depth Proxy

In [19], Tikhonova introduced the Ray Attenuation Function. This is a technique to transform and discretize the volume rendering integral into bins on the intensity domain. To be specific, the volume

rendering integral is given by [12]:

$$C = \int_0^D C(t)\tau(t)e^{-\int_0^t \tau(s)ds}dt, \quad (1)$$

where $C(t)$ is the radiance or color and $\tau(t)$ is the attenuation of a sample t along the view direction. As each ray is cast, intensities are attenuated at each sample position along these rays. Instead of attenuating to a single value, the ray attenuation function attenuates into a finite number of bins in the intensity domain. The ray attenuation function is given by [19]:

$$F(k) = \sum_{\{i|i=0,1,\ldots,M\} \; AND \; bin(i)=k} \alpha(i) \prod_{j=0}^{i} (1-\alpha(j)), \quad (2)$$

where $bin(i)$ is a function that assigns an intensity value of a sample i to a bin k and $\alpha(j)$ is the opacity of a given sample j.

In [20], Tikhonova extends the ray attenuation function and incorporates it with other proxy images to create explorable images. The specific type of proxy image of interest is the depth image. This allows us to reconstruct the normals by analysis of the gradient of local variations in depth. Availability of these normals allows exploration of lighting parameters in the resultant images. Furthermore, such normals allow us to perform more accurate feature extraction/tracking. Tikhonova also presents a multi-view perspective proxy image, which allows users to rotate the camera slightly while exploring, but our current work does not make use of these multi-perspective proxy images.

Explorable images are a natural fit for in situ visualization because the key benefit of explorable images is deferred interaction. As a result, Tikhonova extends the ray attenuation function to interval attenuation function in [21]. The basic idea is to subdivide a ray in volume rendering to intervals. The attenuation of each interval can be computed separately, then later these interval attenuations can be composited into a complete interval attenuation function. The interval attenuation function allows the ray attenuation function to be computed in parallel, which is necessary for efficient in situ visualization. The interval attenuation function for an interval $[L,M]$ is given by:

$$F_{[L,M]}(k) = \sum_{i=L,bin(i)=k}^{M} \alpha(i) \prod_{j=L}^{i-1} (1-\alpha(j)), \quad (3)$$

where $bin(i)$ is a function that assigns an intensity value of a sample i to a bin k and $\alpha(j)$ is the opacity of a given sample j. The attenuation of the i^{th} sub-interval is computed using:

$$A(i) = \sum_{j=0}^{N_i-1} (1-\alpha(j)), \quad (4)$$

where N_i is the number of samples in a sub-interval and $\alpha(j)$ is the opacity of a given sample j. Finally, the total interval attenuation function can be composited as:

$$F(k) = \sum_{i=0}^{P} F(i,k) \prod_{j=0}^{i-1} A(j), \quad (5)$$

where $F(i,k)$ is the interval attenuation function of the i^{th} sub-interval for a bin k.

3.2.2 Integrated Ray Casting

The interval attenuation function is computed by ray casting over the sub-volumes. High sampling rates are necessary to perform high accuracy ray casting, which directly leads to a high rendering cost and is thus not preferable in an in situ visualization environment.

In traditional direct volume rendering, high sampling rates are avoided by using a pre-integration table [6]. The idea of pre-integration is to attenuate not only at the sampling points along a ray, but to integrate over the segments on the ray between these sampling points. Since this integration is done in the transfer function domain, pre-integration can be performed by pre-computing all possible combinations in the transfer function, which is then stored in a 2D array. It is possible to apply pre-integration when computing interval attenuation functions by extending the 2D array to a 3D array, where the third dimension stores the attenuation in each bin of the interval attenuation function. However, this approach uses significantly more memory, especially when as we increase the resolution of the transfer function or the number of bins in the attenuation function. Also, note that the transfer function is replicated in each node in the parallel rendering process.

We introduce a hybrid method that minimizes both the memory and time costs of preintegration of interval attenuation function generation. Whenever a ray segment is obtained while we are casting a ray, there are 2 possible scenarios:

1. Both end points of the ray segment are in the same bin of the interval attenuation function. In this case, the end points must be close to each other in the transfer function domain. In this situation, it is simple and efficient to perform the integration in place.

2. The end points of the ray segment are in different bins of the interval attenuation function. Specifically, the first end point belongs to some initial bin, the second end point belongs to some final bin, and there may be other bins in between these initial and final bins. Figure 2,demonstrates how we perform in place integration for both the initial bin and the final bin. For the bins in between, we must integrate over the entirety of each bin. To do so efficiently, we pre-integrate the attenuation for each bin and query them as needed during the ray casting.

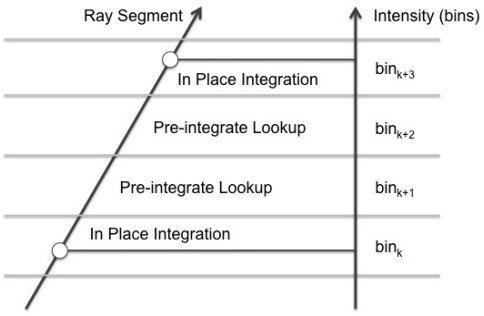
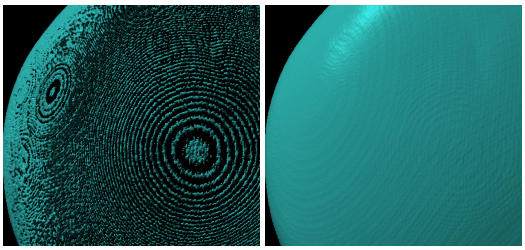


Figure 2: Demonstration of integration for ray casting with our hybrid approach combining pre-integration with in place integration. On the left, a ray segment is ready for integration. Between the two end points, the ray segment crosses two complete bins (bin_{k+1}, bin_{k+2}) and two partial bins (bin_k, bin_{k+3}) in the intensity domain. The attenuation values of the complete bins are pre-integrated and can be queried in constant time during ray casting. The two partial bins require in place integration.

To represent the full integration over each bin for our hybrid method, only n extra floating point numbers are required, where n is the number of bins in the interval attenuation function. We are thus able to eliminate low-sampling-rate artifacts for our generated explorable images as shown in Figure 3.



(a) Without Hybrid Integration (b) With Hybrid Integration

Figure 3: A comparison of ray casting with integration over the intervals between ray samples, and without such integration. Both images show the same isosurface of a supernova dataset, and are obtained at the same sampling rate along each ray. At left, we demonstrate that without integration over the ray segments we may miss important isosurfaces, leading to the ring artifacts shown. On the right, we demonstrate that our hybrid integration technique minimizes such artifacts as well as previous preintegration techniques.

3.2.3 Simplified Default Transfer Function

In observations of how researchers interact with explorable images, we notice that people generally want to visualize semi-transparent isosurfaces in direct volume rendering rather than using smooth transfer functions that generate less detail in the output image. As a result, we provide a simplified default transfer function. With our simplified default transfer function, we can generate good images while further reducing the rendering cost. Note that if a more sophisticated transfer function is needed then users can always supply their own transfer function for the rendering of explorable images in place of this default.

The simplified transfer function generates semi transparent isosurfaces for each bin in the interval attenuation function. Each isosurface represents the middle intensity value of a bin, and the attenuation of the isosurfaces are set to be $\frac{1}{n}$. We are able to use a parametric representation of this default transfer function, which means less memory is needed for storage.

When we combine the default transfer function with our hybrid integration, we are able to further simplify the ray casting procedure. Given a ray segment in ray casting, we test which isovalues are in between the two end points. Then we can composite the attenuation values of the isosurfaces.

3.3 Feature Extraction/Tracking

An explorable image consists of a sequence of proxy images, each corresponding to an intensity value. although the attenuated proxy images are sufficient for approximate rendering the original volume, the depth information of each voxel is lost. That is, the 3-dimensional geometry cannot be reconstructed using only the proxy images. Assume a proxy image corresponding to one bin as depicted as the top-left image of Figure 4, the two features colored in light blue and orange will be considered as a single feature since they have the same isovalue. To distinguish these overlapped features and enable more precise feature extraction, we add a proxy image to store the depth information of each pixel in the explorable image. Note that naively storing depth values of all voxels on isosurfaces will simply result in layers of a point cloud, which would be even less optimal than simply storing the raw data. Instead, we store only the depth information of the front face of each feature. In general, this is all that is required for detection of overlapping features.

For each bin in the explorable image, we store the depth value of the closest occurrence of an isosurface in that bin. This depth map can be regarded as a normal 2-dimensional image, where the intensity value of each pixel is encoded as the depth value. Features of different depth values can be separated, as colored in the left of Figure 4. As the simulation continues, a sequence of 2-dimensional images are generated. We extend our previous work to 2-dimensional explorable image space in order to support feature extraction and tracking on such sequences of explorable images. At a start time point T_0, the features are extracted using the region growing algorithm. As time evolves, we first predict, for each feature, the candidate position based on their history movement. Once the candidate region is obtained, we shrink the predicted region based on the current depth map of T_1 until the common area of the prediction and the actual region is obtained. We then expand the common area using region growing until the actual feature boundary is reached. The prediction-correction tracking schema is illustrated in Figure 4 on the right.

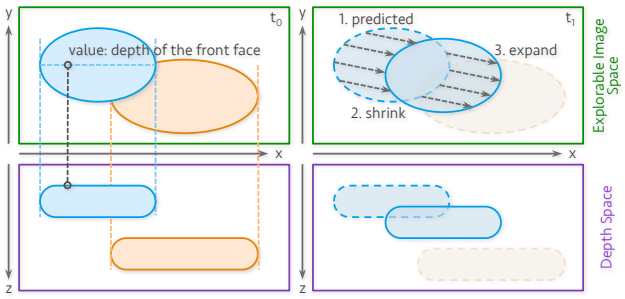


Figure 4: The illustration of the feature extraction and tracking process. The explorable image is enhanced with a layer of depth information (as shown on left of the figure). This depth information allows us to separate overlapping features within the same proxy image. On the right hand side, the prediction-correction schema of feature tracking is illustrated. A candidate region of feature is first predicted based on previous time steps. The candidate feature is then adjusted by shrinking and then expanding to match the actual shape of the feature.

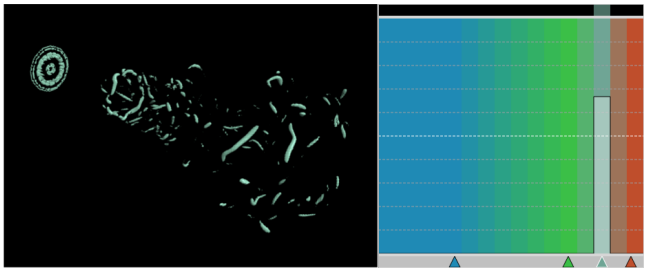
3.4 Feedback to In Situ

As users interact with the explorable images, they sometimes find that additional detail would be helpful in specific regions of the intensity domain. Such 'adaptive binning' is helpful beause it allows users to tweak parameters to ensure useful images are created even while the simulation is still being run.

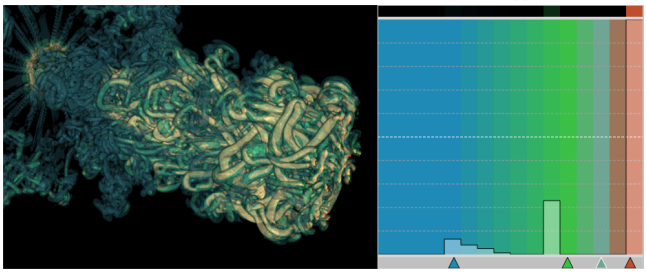
Figure 5 shows a synthesized example for adaptive binning. It is normal to calculate the range between the minimum and maximum intensity of the volume per timestep, which is intended to scale the data to fit the transfer function. However, sometimes these data values are skewed enough that almost all of the data values end up within one bin of the interval attenuation function, such as Figure 5a. In this situation, the visualization would only able to show a very limited portion of the data. By using adaptive binning, when scientists notice that the existing bins do not reflect the ideal distribution for a dataset, they can immediately supply a new bin distribution array so that more bins are available in especially dense or interesting regions of the intensity domain. The simulation will then be able to pick up the new setting in the next timestep, and when new explorable images are generated, they will make use of the new bin distribution to efficiently sample intensity values, as demonstrated in Figure 5b, where many previously undetected features are revealed after adaptive binning.

4 Results

In this section, we begin by presenting performance information for in situ explorable image generation, providing a breakdown of the



(a) This explorable image demonstrates a problem that can occur when the distribution of the input volume is skewed such that almost all of the data ends up being placed into a single bin, as shown in the transfer function on the right. As a result, only a tiny fraction of the original data can be shown to the users for exploration. To address this situation, scientists can immediately supply a new bin distribution so that when the next explorable image is generated, a better bin distribution will be applied.



(b) Adaptive binning allows the user to separate intensity values of interest into different bins. In this case, we reduce the number of bins that had been previously applied to empty intensity regions, and instead allocate more bins in the intensity regions of interest. Although the total number of bins remains identical, as shown in the transfer function at right, adaptive binning allows us to present many more of the interesting details of the original data.

Figure 5: Comparison of a simple but poor default binning with the adaptive binning technique supported by our library.

different variables that can affect performance, and how this performance affects the simulation using our library. We will then show the resultant images that our library can generate, and demonstrate exploration and feature tracking.

4.1 Performance

To be practical, we must demonstrate that our explorable images technique will not incur significant performance or storage costs when compared to the simulation being run. I think if we integrate that and then maybe label the axis on Fig 7 he might leave us alone we can legitimately say that we didn't have time to do the re-binning results with Dr. Ono's data.

4.1.1 Environment

We use the Hopper machine at NERSC as our performance test environment. Hopper consists of 6,384 nodes, 24 cores per node, and 32 GB memory per node. As suggested in NERSC's documentation, we make use of OpenMP by spawning 4 MPI tasks and 6 OpenMP threads per node.

The computational fluid dynamics simulation we use is provided by Ono from RIKEN, Japan. Ono has been using this simulation on the K-computer to generate results with very large volumes. For reference, this simulation generates data such as the one showin in Figure 6. The variable we use in this paper is the 2nd invariant of velocity gradient tensor (i2vgt). The benefit of using i2vgt is to identify potential vortex features.

4.1.2 In Situ Explorable Images

To demonstrate the feasibility of our technique in an in situ environment, we conduct several runs of the simulation provided by Ono.

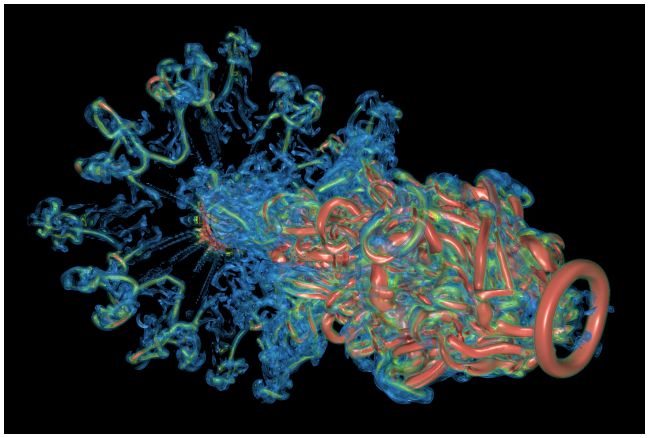


Figure 6: Sample direct volume rendering image generated using the data from Ono's simulation. The image is showing a relatively early stage of the simulation, where the "ring" feature hasn't collapsed into the boundary.

We compare the time taken to generate explorable images of different resolutions in Figure 7a, which demonstrates that both the ray casting time and the image composition time are linearly related to the number of pixels in the output image. We conducted 4 runs of the simulation, where each run simulates a 864^3 volume with 8^3 processes. This translates to a 108^3 subvolume in each process. We generate explorable images at intervals of 20 simulation timesteps. With 1024^2 image resolution, we require 2.5679 seconds to generate an explorable image, which consists of ray casting and composition time. For comparison, the simulation takes an average of 122.42 seconds to compute the 20 timesteps between each generated explorable image.

To determine how ray casting time is affected by simulation volume size, we conducted 3 runs of the simulation. The same explorable image resolution is used for all 3 runs, which is 1024^2. The subvolumes we used for comparison were sized at 36^3, 72^3, and 108^3 voxels per process. During raycasting, we use a sampling rate of one sample per voxel-length along the ray. The result of this comparison is shown in Figure 7b, which demonstrates that ray casting time is linearly correlated to the number of voxels in each subvolume.

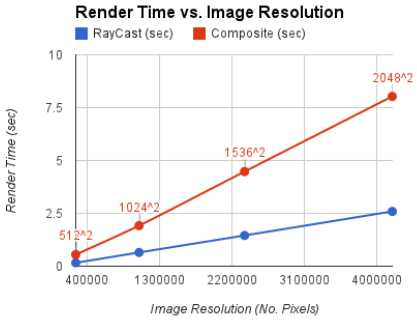
The extensions we have added do not interfere with the known scalability of explorable image generation as demonstrated by Tikhonova [21].

4.1.3 Feature Extraction/Tracking on Explorable Images

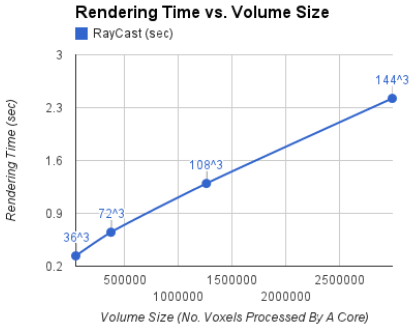
Because feature extraction and tracking are performed based on the generated explorable images rather than the original volume, this operation can be performed locally with no interference with in situ processing. The use of region growing and boundary adjustment for feature extraction and tracking give a time complexity proportional to the number of pixels within each feature, and how many such pixels have changed between timesteps. Because it is efficient to generate many explorable images in situ, we expect that the pixel difference between consecutive explorable images will in general be very small. In our tests, the feature extraction and tracking take an average of 50 milliseconds for a 1024^2 explorable image, which is negligible compared to cost of generation of explorable images, which is itself negligible in comparison to the simulation.

4.2 Visualization

By using our viewer, users can interactively explore an explorable image sequence. In addition to the previously supported kinds of



(a) The ray casting time in explorable image generation is linearly correlated to the image resolution. This is to be expected because we perform a single ray cast for each pixel of the output image. Image composition time is also linearly correlated to the image resolution because this process consists of a single loop through each pixel to composite the interval attenuation function. Simulation time is not affected by the image resolution and is thus not shown in the chart: for all 4 runs, the simulation time is consistent with its average of 6.1213 seconds per timestep.



(b) Our sampling rate along each ray in a raycast is 1 per voxel-length. As a results, our ray casting time is linearly correlated to the subvolume size (the number of voxels processed by a core).

Figure 7: In situ timing results

exploration such as transfer function modification and relighting, users can select/extract features and track such features both forward and backward through time.

4.2.1 Explorable Image

The transfer function in our viewer is represented as a bar chart Figure 8, where the number of bars displayed is equal to the number of bins in the interval attenuation function. Color and opacity for each bin can be modified with the transfer function editor. Figure 9 shows the typical process in which users interact with the transfer function. In general, users have the easiest time selecting features from left to right, skipping any uninteresting bins along the way, until all features of interest are visible.

Explorable images are generated at many timesteps throughout a simulation, so we provide the ability to explore the time domain as well. To do so, we provide a simple handle which the user may drag to change the time step currently displayed. Users can also press the play button and enjoy an animation from the beginning of the simulation to the end.

4.2.2 Feature Extraction/Tracking

Often, features of interest will not show up in early stages of simulations. The capability for storing potential feature information in the form of explorable images provides scientists with the flexibility of studying the simulation with feature extraction and tracking. Figure 10 demonstrates an example of backward tracking of vol-

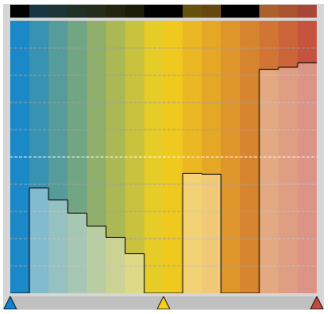


Figure 8: Transfer function editor. The number of bars is equal to the number of bins in the interval attenuation function (in this case, 16). The semi transparent bars are the opacity map, which we can drag to modify the opacity of each bin. The triangle handles at the bottom allow modification of the colors of the handle. The background gradient of the main window changes according to these color handles. The top bar shows the final result of the transfer function on a black background.

umetric features. As the simulation runs, we generate a sequence of explorable images. The domain scientist explores the simulation by adjusting the transfer function and lighting settings, then waits until desired features show up in later explorable images. Once the identified features are selected and highlighted (the four features in orange as shown in Figure 10a), they may be tracked backward (or forward) based purely on the information embedded in the explorable images.

When compared to conventional approaches that attempt to store the entire raw volume, feature tracking from explorable images enables a much longer history of simulation such that the origin of feature-of-interest are traceable. While comparing to 2-dimensional image based feature extraction and tracking, the depth information ensures relatively higher fidelity, so that overlapping features can be identified and tracked separately.

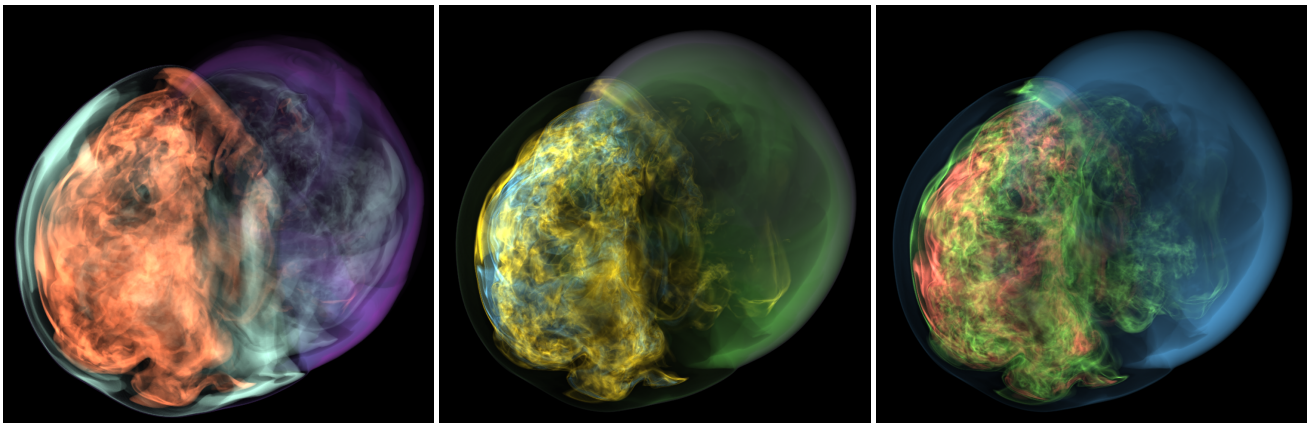
5 Discussion

In the results section, we have shown that our rendering cost is only a small fraction of the simulation cost, and our rendering time scales very well in the parallel environment. Performance is affected by several parameters, including output image resolution, sampling rate along each ray, the size of the subvolume being processed on each core, and the number of bins in the interval attenuation function.

5.1 Parameter Settings

Image resolution linearly affects the ray casting and image composition time, because a single ray cast is performed for each pixel of the output image. Increasing the sampling rate along each ray will increase raycasting time linearly, but will not alter composition time. In our library, the sampling rate is controlled by the volume size and the ray step size. Generally, we default the ray step size to 1 voxel width. As a result, as the volume size increases, the sampling rate increases, which increases the ray casting time linearly. Although we have implemented integration over ray segments, large ray step sizes still generate many 'stairstep' surface artifacts. In our experiments, a single sample per voxel-length is generally sufficient to mitigate such artifacts.

Increasing the number of bins in the interval attenuation function allows each explorable image to better approximate the original direct volume rendering result. However, as the number of bins increases, we also linearly increase the memory and storage requirements. Because we are compositing the interval attenuation functions, we must send them across the network to other processes.



(a) (b) (c)

Figure 9: (a) shows an image with the default binning distribution, where the orange region occupies a whole bin. Obviously there are more details within the orange region. (b) shows the same timestep of supernova with adaptive binning, where there are more bins in the core region of the supernova. (c) shows that we can change the transfer function to generate better color scheme.

Therefore, increasing the number of bins also increases the image composition time. In our experience, 16 bins are often sufficient to generate a good approximation of the volume. By introducing the adaptive binning, this problem is partially sovled because scientists often focus on a small range in the intensity domain.

5.2 Limitations

Each explorable image is an approximation to the original direct volume rendering. The interval attenuation function is a discretization of the direct volume rendering integral into a finite number of bins, usually a small number. As a result, an explorable image is unable to reconstruct a completely accurate direct volume rendering image, but the result is usually close enough for user analysis. [21] shows multiple comparisons between explorable images and ground truth direct volume rendering images.

Depth proxy images can only capture the front face of an isosurface, while the interval attenuation function attenuates both the front and back face of the isosurface. If a slab is being rendered within a bin, a depth proxy image is unable to accurately represent the normal of such a slab. As a result, the lighting computed using the depth map is only an approximation. Because of the same reason, clipping planes is currently not a supported method of exploration for these images.

The interval attenuation functions are computed for a given view, so user modification of the camera angle is prohibited. [20] provides multi-view proxy image that allows limited interaction to the viewing direction, but this would introduce a higher level of uncertainty in the output. However, due to the compactness of the explorable image, users can choose to generate multiple explorable images from different view angles.

Currently, features can only be tracked individually within each bin layer. The reason is different layers represent different isovalues, and with different isovalues, the features should be different.

6 Conclusion

We present a complete workflow to allow scientists to perform in situ visualization. With our library, scientists may generate explorable images in situ. Then, while the simulation is still running, the scientists may interact with the explorable images in both the transfer function domain and time domain. Additionally, scientists may extract and track features of interest both forward and backward through time. When features of specific interest are found, scientists may use our library to apply new parameters to future explorable images being generated, thus steering the generation of better explorable images at later timesteps.

7 Future Work

Although the multi-view proxy image introduces additional uncertainty, it still provides the significant advantage of view-angle modification. We will add the multi-view proxy image to our library, and add methods to approximate the uncertainty introduced by such explorations.

An explorable image is a middle ground between the storing raw simulation data or static images. We will continue to implement additional methods of exploration while retaining low storage cost.

Currently, our library only supports regular grids, but more and more simulations have moved to unstructured grids to provide higher detail in regions of higher importance. We will extend explorable images to support these different types of grid representation.

Features are currently tracked individually within each bin layer. By examining the position and depth information of these features, we should be able to determine many relationships between features across bin layers.

References

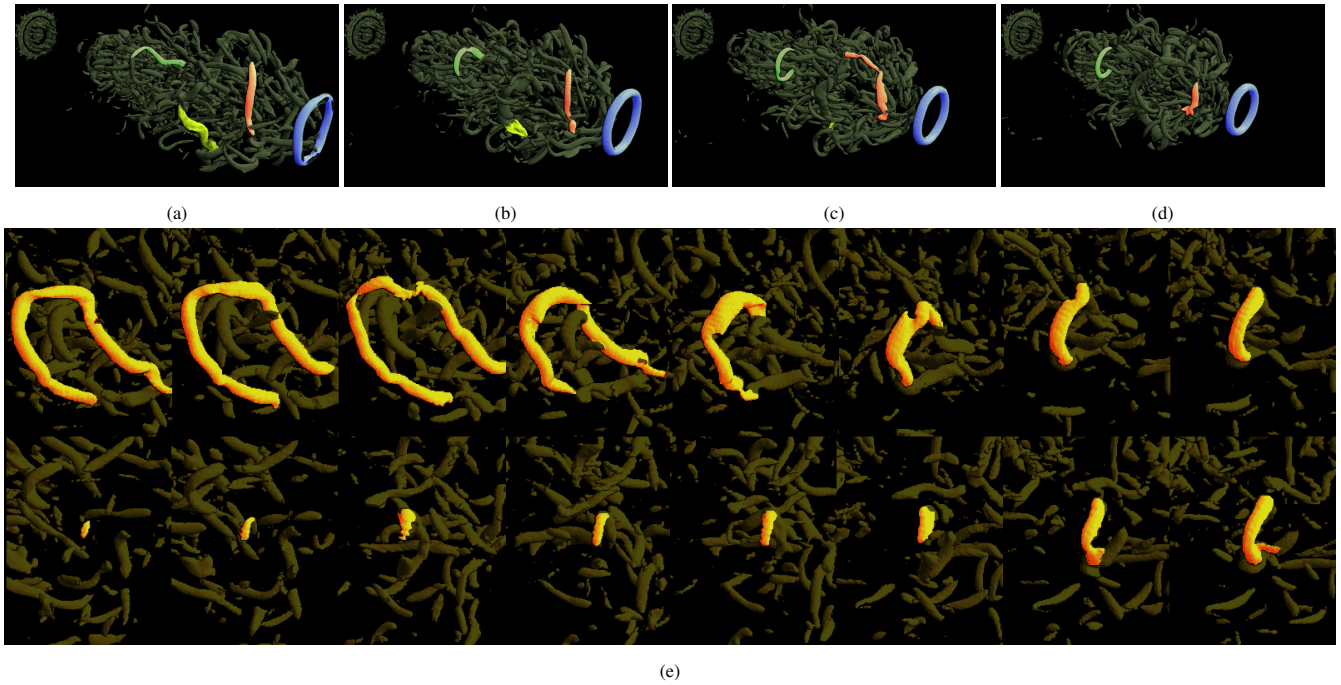
[1] S. Ahern, A. Shoshani, K.-L. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, E. W. Bethel, H. Childs, J. Huang, K. Joy, Q. Koziol, G. Lofstead, J. S. Meredith, K. Moreland, G. Ostrouchov, M. Papka, V. Vishwanath, M. Wolf, N. Wright, and K. Wu. *Scientific Discovery at the Exascale, a Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis, and Visualization*. 2011.

[2] A. BAUER, B. Geveci, and W. Schroeder. The paraview catalyst users guide, 2013.

[3] J. Caban, A. Joshi, and P. Rheingans. Texture-based feature tracking for effective time-varying data visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1472 –1479, 2007.

[4] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, pages 67–75, Oct 2013.

[5] S. Eilemann and R. Pajarola. Direct send compositing for parallel sort-last rendering. In *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, EG PGV'07, pages 29–36, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.



(a) (b) (c) (d)

(e)

Figure 10: Two examples showing the user selected features tracking backward overtime. Figure (a) to (d) show a set four feature are track backward in simulation with the relative position emphasized. Figure (e) shows in detail how a selected feature-of-interest was born from the simulation.

[6] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, HWWS '01, pages 9–16. ACM, 2001.

[7] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In *Visualization, 2003. VIS 2003. IEEE*, pages 209–216, Oct 2003.

[8] A. Kageyama and T. Yamada. An Approach to Exascale Visualization: Interactive Viewing of In-Situ Visualization. Technical Report arXiv:1301.4546, Jan 2013. Comments: 10 pages, 8 figures.

[9] K.-L. Ma. Runtime volume visualization for parallel cfd. Technical report, 1995.

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

[11] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova. In-situ processing and visualization for ultrascale simulations. In *Journal of Physics: Conference Series*, volume 78, page 012043. IOP Publishing, 2007.

[12] N. Max. Optical models for direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 1(2):99–108, Jun 1995.

[13] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, M. E. Papka, and S. Klasky. Examples of in transit visualization. In *Proceedings of the 2Nd International Workshop on Petascal Data Analytics: Challenges and Opportunities*, PDAC '11, pages 1–6, New York, NY, USA, 2011. ACM.

[14] C. Muelder and K.-L. Ma. Interactive feature extraction and tracking by utilizing region coherency. In *Visualization Symposium, 2009. PacificVis '09. IEEE Pacific*, pages 17–24, April 2009.

[15] S. Ozer, J. Wei, D. Silver, K.-L. Ma, and P. Martin. Group dynamics in scientific visualization. In *Large Data Analysis and Visualization (LDAV), 2012 IEEE Symposium on*, pages 97–104, Oct 2012.

[16] F. Reinders, F. H. Post, H. J. W. Spoelder, and K. V. Time-dependent. Visualization of time-dependent data using feature tracking and event detection. *The Visual Computer*, 17:55–71, 2001.

[17] H. Theisel and H.-P. Seidel. Feature flow fields. In *Proceedings of the Symposium on Data Visualisation 2003*, VISSYM '03, pages 141–148, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[18] A. Tikhonova, C. D. Correa, and K.-L. Ma. Explorable images for visualizing volume data. In *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, pages 177–184, March 2010.

[19] A. Tikhonova, C. D. Correa, and K.-L. Ma. An exploratory technique for coherent visualization of time-varying volume data. In *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'10, pages 783–792, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[20] A. Tikhonova, C. D. Correa, and K.-L. Ma. Visualization by proxy: A novel framework for deferred interaction with volume data. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1551–1559, Nov 2010.

[21] A. Tikhonova, H. Yu, C. D. Correa, J. H. Chen, and K.-L. Ma. A preview and exploratory technique for large-scale scientific simulations. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization*, EG PGV'11, pages 111–120, Aire-la-Ville, Switzerland, Switzerland, 2011. Eurographics Association.

[22] Y. Wang, H. Yu, and K.-L. Ma. Scalable parallel feature extraction and tracking for large time-varying 3d volume data. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, EGPGV '13, pages 17–24, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[23] Y. Ye, R. Miller, and K.-L. Ma. In situ pathtube visualization with explorable images. In *Proceedings of the 13th Eurographics Symposium on Parallel Graphics and Visualization*, EGPGV '13, pages 9–16, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[24] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *Computer Graphics and Applications, IEEE*, 30(3):45–57, May 2010.

[25] H. Yu, C. Wang, and K.-L. Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 48:1–48:11, Piscataway, NJ, USA, 2008. IEEE Press.