In Situ Depth Maps Based Feature Extraction and Tracking

Yucong (Chris) Ye* UC Davis
Yang Wang† UC Davis
Robert Miller‡ UC Davis
Kwan-Liu Ma§ UC Davis
Kenji Ono¶ RIKEN, Japan

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 post-hoc in-depth analyses of the saved data. System I/O capabilities have not kept pace as simulations have scaled up over time, so a common approach has been to output only subsets of the data to reduce I/O. However, as we are entering the era of peta- and exa-scale computing, this sub-sampling approach is no longer acceptable because too much valuable information is lost. In situ visualization has been shown a promising approach to the data problem at extreme-scale. We present a novel in situ solution using depth maps to enable post-hoc image-based visualization and feature extraction and tracking. An interactive interface is provided to allow for fine-tuning the generation of depth maps during the course of a simulation run to better capture the features of interest. We use several applications including one actual simulation run on a Cray XE6 supercomputer to demonstrate the effectiveness of our approach.

1 INTRODUCTION

As simulations reach exa-scale, researchers can no longer depend on the traditional method of simply saving raw data occasionally, then performing visualization in a post-process. The portion of the raw simulation data that can be feasibly stored for post-hoc visual analysis has simply become too small to be useful 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 minimizing the 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 for full-scale, 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 researchers are limited to these specific views of their results. Even though it is possible to alter the settings of the in situ images for the future time steps of the simulation, with no exploration on the static images, there is no effective way for researchers to come up with new image settings other than guessing.

Different techniques are proposed to enhance the explorability of in situ visualization, such as volumetric depth images [11], image databases [2], and explorable images [30]. In this paper, we present a new in situ approach based on depth maps, which enables a new level of post-hoc visual exploration. First, depth maps are generated in situ by rendering volumetric data into multiple layers of isosurfaces in image space. Isosurface exploration is then made possible with a novel algorithm that we have introduced to reconstruct from the depth maps for visualization of any isovalue. Second, the same depth maps allow post-hoc 3D feature extraction and tracking (FET) in image space without requiring the original volumetric data. This concept of performing 3D FET in image space is novel; especially, its cost in term of both computational and storage requirements is orders of magnitude lower than that of conventional methods. An interactive interface is created for depth maps based visualization and FET. In addition, through this interface, the user is allowed to tune the parameter setting for generating depth maps over the remaining simulation run or a new run. That is, by previewing an early part of the simulation using depth maps, it is possible for the scientists to adjust the setting to better capture flow features of interest in the simulation.

To summarize, our work makes the following contributions:

- A novel algorithm to reconstruct isosurface by interpolating nearby layers of isosurfaces.
- A posteriori feature extraction and tracking via depth maps without requiring the raw simulation data.
- A feedback mechanism allowing users to fine-tune the depth maps rendering for future simulation time steps.
- A workflow allowing users to monitor simulations and modify visualization results in situ.

2 RELATED WORK

Ma [18] and Johnson et al. [14] first demonstrated coupling visualization with simulation back in the 1990s. Ma [19] presents the challenges and opportunities of in situ visualization. Later, Ahern et al. [1] also discusses the pros and cons of in situ processing. There are different types of in situ processing, as described by Dreher et al. [10]. Essentially in situ processing can be divided depending on where the data is being processed:

- Tightly-coupled synchronous: Data processing and the simulation use the same set of compute nodes [16, 25, 32]. As a result, it is important that in situ data processing does not compete computing time and storage space with the simulation. The general rule of thumb is to keep the in situ processing time under 5% of the total simulation time [19]. Damaris [9] extends this category by utilizing a dedicated core per compute node for asynchronous I/O operations, which hides the I/O cost from the main simulation.
- Loosely-coupled asynchronous: The data is first transferred to another set of compute nodes or another computer over the network without touching the hard disk, and then analysis/visualization is performed there with little impact to the

*e-mail:chrisyeshi@gmail.com
†e-mail:ywang@ucdavis.edu
‡e-mail:bobmiller@ucdavis.edu
§e-mail:ma@cs.ucdavis.edu
¶e-mail:keno@riken.jp

simulation. 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. Moreland et al. [21] gives a few examples of in transit visualization.

The desired scientists' workflow determines which type of in situ processing to be adopted. Bennett et al. [4] combine in situ and in transit processing to take advantage of both techniques. The approach is to perform highly efficient parallel data processing in situ and the reduced data is asynchronously processed (i.e., in transit) to possibly use more computationally demanding calculations. The cost of using our technique is small when compared to the simulation, so we use the tightly-coupled synchronous approach.

In situ data compression algorithms are introduced to reduce I/O and storage requirements. Schendel et al. [26] present an asynchronous method exploiting data access patterns for more efficient data compression and I/O. Lakshminarasimhan et al. [17] introduce an error bounded in situ compression algorithm for scientific data, which offers up to 85% compression ratio by taking advantage of both spatial and temporal coherence of the data.

In situ rendering to generate static images is commonly used. There are multiple toolkits that enable in situ visualization, including ParaView Catalyst [3, 25], VisIt [7, 25, 31] and Damaris [9]. Static images represent the smallest possible data size output from in situ visualization, but static images can present only a very small fraction of the original volumetric data to the scientists. To address this limitation, a common solution is to generate multiple sets of static images with different settings. Kageyama et al. [15] generate images from many viewpoints, allowing post-hoc exploration in different perspectives of the original data. However, this approach could require too much storage space both online and offline.

Explorable Images, introduced by Tikhonova et al. [29], is a volume data exploration method without requiring the original volume data and a powerful computer. The basic concept is to quantize the intensity levels into a small number of bins and each bin is used to store the attenuation values from ray casting. With ray attenuation functions, post-hoc interactive volume visualization and exploration is enabled. The parallelized version of explorable images can be generated in situ with the simulations [30]. Our technique can be coupled with ray attenuation function calculations because both use ray casting as the rendering algorithm.

Ahern et al. [2] extends on the idea of generating images from multiple viewpoints to enable camera space exploration. Instead of only storing the final rendering results, multiple objects per viewpoint is stored and a post-hoc composition of objects is rendered as final image. As meta-data for each object is stored, object querying is also enabled. Our approach focuses on the exploration of a single object in a single viewpoint, which not only allows opaque isosurface rendering but also allows rendering of semitransparent multiple layers of isosurfaces.

Volumetric depth images, introduced by Frey et al. [12], is another approach to creating post-hoc view-dependent volume visualization. Instead of subdividing the intensity domain like ray attenuation functions, volumetric depth images breaks each view ray into depth ranges with composited color and opacity. Final visualization result is then the composition of the stored depth ranges. Volumetric depth images are later parallelized into an in situ technique by Fernandes et al. [11]. Since depth ranges are used, generating isosurfaces with volumetric depth images can be uncertain. The depth value of isosurface in our technique is generated during ray casting, which is the ground truth value.

Volumetric depth images are based on layered depth images (LDI), which is introduced by Shade et al. [27]. In LDI, multiple pixels at different depths are stored per line of sight. Each pixel contains multiple attributes, including color. LDI is mainly developed to enable viewpoint changes. Our depth maps only store the depth information of isosurfaces, and is mainly used for isosurfaces visualization.

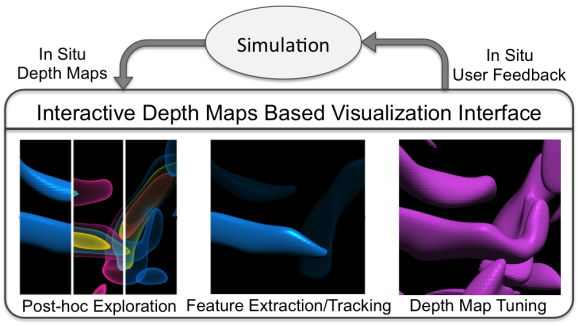


Figure 1: Workflow for data analysis of active simulation with depth maps. Depth maps are outputted in situ while the simulation is running. With depth maps, multiple layers of semitransparent isosurfaces can be visualized. Furthermore, feature extraction and tracking can be performed on top of depth maps. Finally, users can provide feedback to the ongoing simulation in order to enhance the quality of depth maps.

Biedert et al. [5] construct depth images from ray casting segmented volume data based on the contour tree of the data. The contour tree defines the relationship among the different isosurfaces, with which ray casting can be optimized accordingly. However, with their design, isosurfaces might not be generated at the exact isovalues that the user desires. Their design is potentially usable for in situ visualization, but it was not evaluated nor demonstrated in a massively parallel setting. Ours can also utilize the contour tree of the data for handling features across multiple layers within the same depth maps.

In a parallel processing setting, after rendering the images locally, a final composition is required to generate a global image. Moreland et al. evaluates multiple image composition techniques [20]. In our work, we adopt an implementation of the 2-3 swap algorithm [33]. The 2-3 swap requires multiple rounds of communication among processes but in each round only a few messages are received by each process, meeting our efficiency requirements.

Feature-based data exploration is considered a key approach to the study of large time-varying flow field data. Conventional approaches extract features from individual time steps and then associate them between consecutive time steps. More recent work utilizes either higher-dimensional iso-surfacing [13] or non-scalar representations of the data [28] for feature tracking. These techniques require users to manually track features in different time steps. On the other hand, prediction-correction based approaches [6, 22, 24] first predict candidate regions according to the feature descriptors (such as boundary or centroid location) extracted from previous time steps, and then adjust the predicted region to match the corresponding region in the next time step for correct feature tracking. The prediction-correction based approaches are appealing for their computing efficiency and reliability in an interactive system.

3 Methodology

This section introduces our depth maps based design and corresponding operations.

3.1 Workflow

The expected workflow with our system is shown in Figure 1. The simulation periodically generates depth maps according to a configuration file, which is set by the user. In situ exploration of the simulation results is done by interacting with depth maps on a desktop computer or even a mobile device. With depth maps, isosurfaces can be rendered over the entire range of isovalues. Furthermore,

feature extraction and tracking is performed on top of the isosurfaces. Sometimes, the original configuration file can not reveal the features of interest. Scientists can update the configuration file according to what are seen with the current depth maps. Then the updated configuration file is uploaded to the simulation in situ, which impacts the depth maps generation for future time steps.

3.2 Depth maps

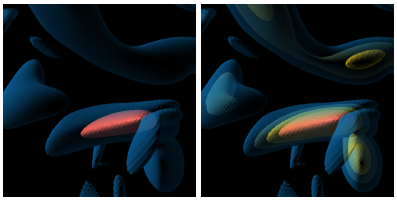
Depth maps are the fundamental elements of our workflow. There are two main advantages of depth maps. First, the data size of depth maps is orders of magnitude smaller than raw simulation data, therefore outputting only depth maps effectively increases the I/O efficiency. Secondly, unlike static images, depth maps enable multiple visualization operations: 1) Normal estimation, 2) lighting based on the estimated normals, and 3) image-based feature extraction and tracking. Depth maps consist of multiple depth layers. Each layer is a depth map of an isosurface from the camera point of view, as shown in Figure 3a. Only the front most surface of an isosurface is stored. Users can choose how many isovalues and which isovalues to generate depth layers.

Marching cubes and ray casting are both common ways to generate isosurfaces. Ray casting is used in our workflow because ray casting is strictly view dependent, which avoids the extra computation outside of the view frustum that occurs in marching cubes. Biedert et al. [5] also use ray casting to generate depth images. Because they perform volume segmentation before ray casting, they are able to skip voxels if the eight corners of a voxel are within the same branch in the contour tree. In our case, we have no volume segmentation information so we have to step through each voxel along the rays to identify the isosurfaces. Even though a full ray casting algorihtm is slower, we save time on not performing volume segmentation.

3.3 Adaptive Isovalue Selections

Making depth maps with uniform value intervals is simple and generally produces acceptable results. However, when the data is skewed, which means a lot of the interesting features are within a small interval of the scalar value range, there is a high chance that this small interval is in between two default isovalues. Such an example is shown in Figure 2.

An array of isovalues is used to instruct the rendering of isosurfaces. During the ray casting process, each ray segment is tested against this isovalues array. When the ray segment crosses an isosurface, the depth is compared and recorded to the depth maps. Users can modify the array in order to generate more isosurfaces in the interesting intervals of the scalar value range.

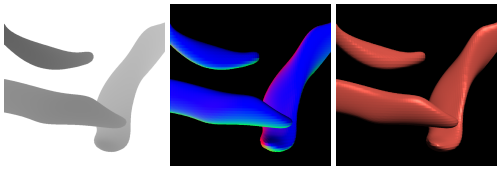


(a) Default Isovalues (b) Adaptive Isovalues

Figure 2: The comparison of default constant interval among isovalues and adaptive selected isovalues is shown using the vorticity dataset. (a) shows two nested isosurfaces in blue and red. The internal structures of certain features become indiscernible. In (b), those structures are revealed by adding two more isosurfaces between the original two isosurfaces.

3.4 In Situ Rendering of Depth maps

In this work, we focus on supporting parallel 3D flow simulations that distribute the modeled 3D spatial domain among all the processing nodes and output volume data.



(a) Depth Layer (b) Normal Map (c) Isosurface

Figure 3: The process of visualizing an isosurface consists of 1) obtaining a depth layer from depth maps, as shown in (a), 2) extracting a normal map from the depth layer, as shown in (b), and finally 3) applying phong shading, as shown in (c).

To generate depth maps in situ, each processing node independently performs ray casting rendering to generate regional depth maps. These local depth maps are then composited by taking the minimum depth value of each isosurface to form the overall depth maps. Our current implementation uses 2-3 swap [33], a state of the art algorithm.

The simulation runs independently of what the user has to do with the depth maps. Whenever the simulation is ready to generate a new set of depth maps, it computes new depth maps with the most updated configuration file.

3.5 Isosurfaces Visualization

The direct rendering of depth maps conveys very limited information, as shown in 3a. Instead, isosurfaces can be visualized with the depth maps, as shown in Figure 3c. In order to apply lighting to the isosurfaces, normal maps are first estimated from the depth layers, as shown in Figure 3b. Shading can then be applied according to the normal maps.

A normal map is computed from each depth layer. The simple way is $Normal = \Delta x \times \Delta y$, where (x,y) is the current pixel position. The Δx vector is $< x+1, y, d_{x+1,y} > - < x-1, y, d_{x-1,y} >$ and Δy vector is $< x, y+1, d_{x,y+1} > - < x, y-1, d_{x,y-1} >$, where $d_{x,y}$ is the depth of pixel (x,y). The 4 samples may not land on the same feature, thus there appears to be a thin border at feature boundaries with incorrect normals, as shown in Figure 5a.

The improved normal estimation procedure is applied to each pixel in the depth layer. From the current pixel to its 8 neighboring pixels, 8 triangles are formed according to Figure 4. The normal of each triangle is then computed. If a normal is pointing toward the side instead of toward the screen, the corresponding triangle is at the isosurface edge and the normal is discarded. The rest of the triangle normals are averaged to produce the final normal for the current pixel. The improved isosurface rendering is shown in Figure 5b where features have no borders.

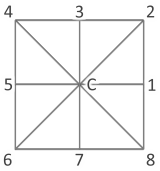


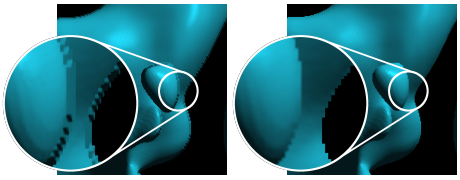
Figure 4: Current pixel and its 8 triangles

Once normals are available, we can apply lighting to the isosurfaces. A simple phong shading is applied. A resulting isosurface rendering image is shown in Figure 3c.

3.6 Depth Layer Interpolation

Isosurface rendering and feature extraction and tracking (FET) are performed on top of a depth layer. Since a depth layer corresponds to one isovalue, isosurface rendering and FET are limited by the available layers. With depth layer interpolation, we are able to estimate isosurface and FET with any isovalue, allowing users to make highly educated modifications to the isovalues array.

Consider two depth layers, A and B. A is in front of B in image space. The general idea is to identify corresponding points on the two depth layers then interpolate between them. A naive approach would be to treat the same pixel locations on the two depth layers



(a) Edge Artifacts (b) Improved Visualization

Figure 5: Using a more sophisticated normal estimation algorithm effectively reduces the edge artifacts. A thin border of incorrect normals appears in (a). Our normal estimation algorithm successfully eliminates the border, as shown in (b).

as corresponding points. In this case, only the overlapping areas in image space would be interpolated, which is not ideal. Instead, we trace a ray from each pixel on depth layer A according to its normal vector. This normal vector closely approximates the gradient of the field, and thus will more likely intersect interior layers. The origin point on the outer surface and the intersection point on the interior surface are considered as corresponding points, even though they do not overlap in image space. Given an interpolation ratio, we linearly interpolate to find a point along each segment between corresponding points, effectively forming a point cloud. A new interpolated depth layer is then reconstructed from this point cloud. More specifically, the following procedure is applied:

1. For each pixel P in image space, a 3D point P_A is constructed as $(P.x, P.y, depth(P,A))$, where $depth(P,A)$ denotes the depth value of P on depth layer A.
2. A 3D ray $ray3d(P_A)$ is constructed with origin P_A and direction $normal(P_A)$, where $normal(P_A)$ denotes the normal vector of P_A computed according to section 3.5. Also, a 2D ray $ray2d(P)$ is extracted from $ray3d(P_A)$ by taking the (x,y) components of $ray3d(P_A)$'s origin and direction.
3. The super cover of $ray2d(P)$ is traversed according to the algorithm introduced by Dedu [8]. A super cover is all the pixels a 2D ray crosses in an image. In our case, the super cover represents all the possible intersecting locations of $ray3d(P_A)$ on depth layer B.
4. For each pixel in the super cover on depth layer B, an intersection test is performed against $ray3d(P_A)$. Since a pixel is represented as a patch with 4 corners, we use a ray-patch intersection algorithm introduced by Ramsey [23].
5. If $ray3d(P_A)$ intersects pixel Q_B, an interpolation segment is constructed. An interpolation segment consists with P_A and Q_B. The interpolation segments are then stored in a pool. Figure 6 shows how interpolation segments are obtained.
6. Given a ratio in the range $[0,1]$, a 3D point can be interpolated from each interpolation segment. All the interpolated points form a point cloud which is used to reconstruct the new interpolated depth layer.
7. To effectively transform the point cloud into a surface, the points are first stored into a 2D uniform grid.
8. For each pixel in the new depth layer, points within a certain radius are queried from the 2D uniform grid, then a weighted average function is applied to calculate the interpolated depth value for the pixel. An example of depth layer interpolation is shown in Figure 7.

Since only the rays that intersect with both depth layer A and B are stored, some surface on the new depth layer can be missing, especially when the new depth layer is close to depth layer A. The resulting depth layer is provided as an estimation for users to make adjustments to the isovalues array.

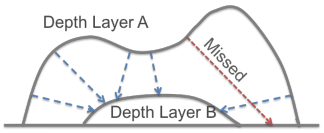


Figure 6: Depth interpolation begins by tracing rays from depth layer A according to the normals. The blue dashed arrows show the rays that hit depth layer B, while the red arrow shows an example of missed rays. Interpolation segments are constructed from the blue arrows and later on used to reconstruct the interpolated depth layer.

Figure 7: Depth layer interpolation example: the left and right images are two isosurfaces rendered with existing depth layers. These two depth layers are used to generate an interpolated depth layer with ratio 0.5, and the resulting isosurface is shown in the center image. Artifacts appear at the joint of the two features due to high amount of discarded interpolation segments.

3.7 Feature Extraction and Tracking

Other than isosurfaces rendering, feature extraction and tracking (FET) is also enabled by depth maps. Our FET algorithm operates on a single depth layer in depth maps. A single depth layer can be regarded as an image with pixels representing the depth values of an isosurface. The key idea is to utilize depth discontinuity to distinguish features in the same depth layer, as shown in the left column of Figure 8.

The feature extraction process is illustrated in Figure 9. For each depth layer, we first create a mask layer with all pixel values initiated to 0. Then we seed from the top-left corner of the depth layer and assign corresponding pixel in the mask layer with an integer feature ID. The feature boundary expands to neighboring pixels if the depth difference with the neighboring pixel is within range (threshold defined by user). All the pixels within the same feature are labeled using the same integer feature ID as the seed point. The region growing process continues until there is no more pixels suffice the threshold of the depth constraint. Then, we increase the integer feature ID by one and search for the next non-background ($depth < 1.0$) pixel, using it as the next seed point and repeat the region growing process until all connected pixels are found to be marked as the same feature in the mask layer. It takes a total of $O(numPixel)$ time to finish the feature extraction process. The output is a 2D mask layer with each pixel containing the integer feature ID it belongs to. An example of multiple features being extracted is shown in Figure 10.

Our feature tracking on depth layer is a prediction-correction based approach. To track a feature in depth layer from T_n to T_{n+1}, we first compute the movement vector by subtracting the centroids (the unweighted average of the all pixel positions of the features) from T_{n-1} to T_n. Since depth maps are generated in situ as the simulation runs, the feature movement between two consecutive calculation steps is small and thus predictable based on previous time steps. Based on the movement vector, the mask of the feature in T_{n+1} is predicted. We then shrink the predicted mask by the depth layer in T_{n+1} to the common region. The shrinking process also utilizes the depth discontinuity as feature identification. Finally, region growing is applied to obtain the actual feature in T_{n+1}. The prediction-correction tracking schema is illustrated in the right column of Figure 8.

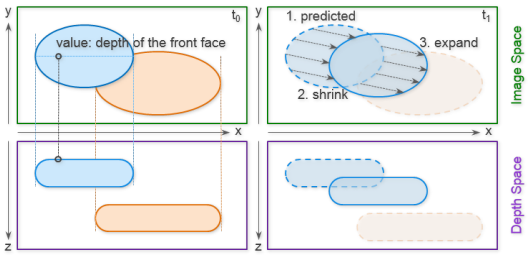


Figure 8: The illustration of the feature tracking process. The right column illustrates the prediction-correction schema of feature tracking. A candidate region of feature is first predicted based on previous time steps. The candidate feature is then adjusted by shrinking and then expanding to match the actual feature.

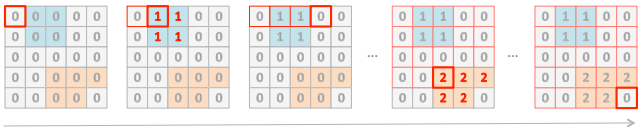


Figure 9: The illustration of the feature extraction process. The process starts by initializing a feature ID mask to all 0, and then iterates through each pixel in the depth layer. If the current pixel is in the background ($depth >= 1.0$) or its feature ID is already set, the process skips to the next pixel. If the current pixel is in a feature ($depth < 1.0$) and the feature ID is not set, region growing is deployed to set a feature ID to all the pixels within the same feature.

During feature tracking, two types of feature events need to be taken care of. When two separated features merge together, the correction process of one feature will capture the pixels of the other feature and the feature IDs of the latter feature on the mask layer will be replaced. When a feature splits into two or more features, the same feature ID will be used until users manually re-extract the feature of interest.

4 Test Results

We conducted tests to measure the cost of depth maps generation and to demonstrate the in situ process using a flow simulation on NERSC's Hopper, which is a Cray XE6 with peak performance of 1.28 Petaflops/sec. There are 6,384 compute nodes, and each node consists of 24 compute cores and 32 GB memory. The simulation of choice is FFV-C (FrontFlow / violet Cartesian), provided by RIKEN (http://www.riken.jp). FFV-C is a three-dimensional unsteady incompressible thermal flow simulator on a Cartesian grid system. We performed a jet engine simulation using FFV-C, and the scientists used Q-criterion to identify interesting vortices.

4.1 In Situ Computing Costs

The extra memory requirement for our method is the memory space to store the depth maps, which is an array per pixel of the image. The array size is the same as the isovalues array.

To assess the feasibility of our method, we conducted 3 sets of tests against varying 1) subvolume voxel count, 2) image resolution, and 3) subvolume count. We isolate the impact of the variable in focus by keeping the other two variables constant for each test. Note the simulation time is measured as the average simulation time per time step.

The time with respect to subvolume voxel count was measured with 16 runs of the FFV-C simulation. Image resolution was set to 512x512 and 8 MPI processes were used. Subvolume size varies from 4x4x4 to 64x64x64. We use a small number of MPI processes because it only affects the image composition time, which is not the concern of this test. The simulation time, local render time, and image composition time are shown in Figure 11a. The local render time and subvolume voxel count appear to be linearly related. The simulation time is also linearly proportional to subvolume voxel count, but we can see the rendering time is significantly lower than the simulation time. Since a fixed number of subvolumes are being composited and the image resolution is also fixed, the image composition time appears to be mostly constant across the multiple FFV-C runs.

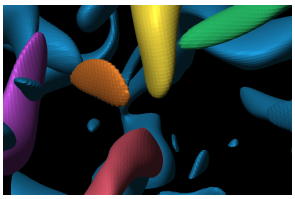


Figure 10: Image showing multiple features being extracted using the depth maps generated from the vorticity dataset. Depth discontinuity is used to distinguish the selected vortices from other ones. The purple vortex on the left is correctly extracted with 2 other vortices in front of it.

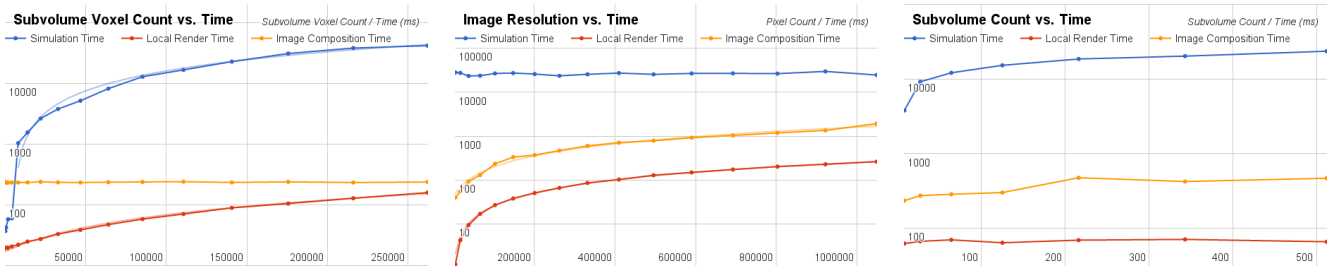
The study of time with respect to image resolutions was conducted with 16 runs of the FFV-C simulation. Subvolume size was set to 32x32x32 and total volume size was set to 256x256x256. Therefore 512 MPI processes are used to ensure good evaluation of image composition. The image resolution varies from 64x64 to 1024x1024. Figure 11b shows the resulting simulation time, local render time, and image composition time. The simulation time remains mostly constant because the subvolume size and the total volume size are fixed. The local render time and image composition time increases linearly according to the pixel count, as expected. The same reasoning applies to the image composition algorithm (2-3 swap), plus sending a larger image buffer across the network also increases the image composition time.

The study on the cost with respect to the subvolume count is used to show the scalability of our technique. The results of 7 runs of the FFV-C simulation are used. Subvolume size was set to 32x32x32 and image resolution was set to 512x512. Total volume size varies from 64x64x64 to 256x256x256. The timing results are shown in Figure 11c. The local render time remains mostly constant because the subvolume size and the image resolution are fixed. The simulation time and image composition time increases accordingly because more network communication is required. The image composition algorithm 2-3 swap has been shown scalable [33]. Again, the time required for depth maps rendering is significantly lower than the simulation time, making our in situ method acceptable.

4.2 Feature Extraction and Tracking

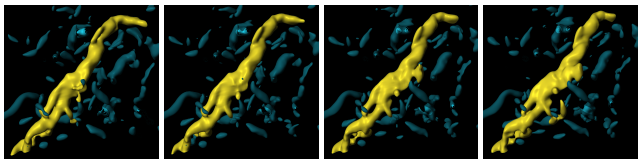
We demonstrate FET using the forced isotropic turbulence dataset from the John Hopkins Turbulence databases. The "coarse" version of the dataset has 1024 time steps, ranging from 0.000 to 2.048 with 0.002 interval. Each time step consists of a 1024^3 velocity field. The following results are generated using time steps 1.000 to 1.120 and the central portion 512^3 of the volumetric data. The velocity field was first processed by a Gaussian filter with $\sigma = 10$ and then Q-criterion was calculated from the velocity field. Depth maps were then generated using the 80 Q-criterion fields 512^3.

A large vortex tube is being extracted and tracked in multiple depth layers of depth maps. We show the extracted features corresponding to isovalues 6.875 and 18.125 in Figure 12 and Figure 13. The tracked vortex tube with isovalue 6.875 remains as a single feature throughout the time span. However, with isovalue 18.125, the vortex tube splits at time step 1.080. Our algorithm is able to detect the split and track both features. By tracking the same fea-



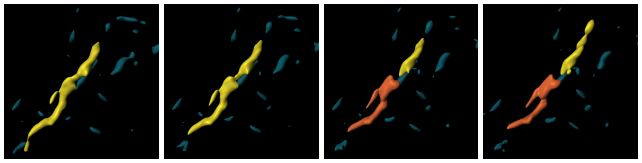
(a) Varying Subvolume Sizes (b) Varying Image Resolutions (c) Varying Subvolume Counts

Figure 11: Simulation time, local render time, and image composition time are measured for different settings, indicating the cost of computing depth maps. Y axis is time in milliseconds. With varying subvolume sizes, shown in (a), the simulation time and local render time appear to be linearly related to the subvolume voxel count, while the local render time is significantly lower than the simulation time. The image composition time appears to be constant over the multiple FFV-C runs. With varying image resolutions, shown in (b), the simulation time stays unchanged. The local render time and image composition time linearly increase according to image pixel count. With varying number of subvolumes, shown in (c), the local render time appears to be constant across the multiple FFV-C runs. The simulation time and image composition time increases accordingly. The rendering time again is significantly lower than the simulation time.



(a) 1.000 (b) 1.040 (c) 1.080 (d) 1.120

Figure 12: The central vortex tube of the isotropic data set from John Hopkins turbulence databases is extracted and tracked at isovalue 6.875. The entire vortex tube is being extracted at time step 1.000 and tracked through time step 1.120.



(a) 1.000 (b) 1.040 (c) 1.080 (d) 1.120

Figure 13: Layer with isovalue 18.125 in the depth maps are shown with FET. The feature of interest is the central vortex tube of the isotropic dataset from John Hopkins turbulence databases. The entire vortex tube is extracted at time step 1.000 and 1.040. The vortex tube later on splits into two features from time step 1.080.

ture with multiple isovalues, more behavioral insights of the feature are revealed.

4.3 Tuning In Situ Depth Maps Generation

Here, we show from generating depth maps in situ to tuning the isovalues array when running FFV-C on Hopper.

The volume dimension is 512^3, which is divided up by 16 in each dimension, resulting 4096 subvolumes with the size of each subvolume to be 32^3. Each MPI process is assigned one subvolume, while 6 OpenMP threads are enabled per MPI process. As a result, each compute node is responsible for 4 subvolumes with 6 cores are used per subvolume. Thus the entire 24 compute cores of each compute node are utilized. Total 24,576 compute cores (1024 compute nodes) are utilized for the simulation.

To demonstrate that we do not require depth maps at every time step, we choose to generate a set of depth maps every 80th simulation step. The camera configuration is preselected and uploaded as model view projection matrices. 16 layers are generated per set of depth maps and the resolution of each layer is [1375, 1007]. The simulation time, local render time, and image composition time follows closely to the trends described in section 4.1. An actual performance number is not shown here because the performance can be affected differently depending on which changes are made to the configuration file.

From the beginning of the simulation, the layers are generated using a uniformly distributed isovalues array. Because the flow in the early time steps of the simulation is not fully developed yet, we wait until time step 8040 to inspect the depth maps.

The depth maps generated at time step 8040 has a scalar value range of [-20.63, 11.245]. Isovalues are uniformly distributed within this range. Six selected layers are shown in the top row of Figure 14. Not many interesting features are revealed by the depth maps because most of the layers are mostly empty. As a result, refinement of the isovalues array is necessary.

By inspecting the isosurfaces of time step 8040, we can tell the isosurfaces generated in the range [-20.63, -1.505] and [4.87, 11.245] are mostly empty. As a result, we want to focus on the range [-1.505, 4.87]. Since we still don't know the distribution of isovalues within the focused range, a uniformly distributed isovalues array within the focus range is uploaded to generate the next set of depth maps. The resulting isosurfaces from time step 8120 are shown in the middle row of Figure 14. More layers are filled with features and we get a better capture of the simulation from the updated depth maps. However, the isosurfaces near isovalue 0.0 are too cluttered. As a result, we want to avoid isovalue 0.0.

To avoid isovalue 0.0, we need to modify the isovalues array to focus on the range [-0.485, -0.23] and range [1.045, 1.895]. The resulting isosurfaces from time step 8200 are shown in the bottom row of Figure 14. All the isosurfaces are filled with interesting features while some layers focus on the negative values and the other layers focus on the positive values.

We are able to refine the depth maps rendering as above while the simulation is running. We started from the default isovalues array and finally arrived at the optimal isovalues array with focused intervals of interest.

5 Limitations

Our technique allows scientists to save a fraction of the simulation data and be able to explore the isosurfaces using the depth maps. We expect in most cases scientists will continue to save out all simulation data due to the fact that depth maps are view dependent. However, we hope that after incorporating depth maps, scientists will be confident enough to reduce the amount of saved simulation data. Following are immediate limitations of our technique.

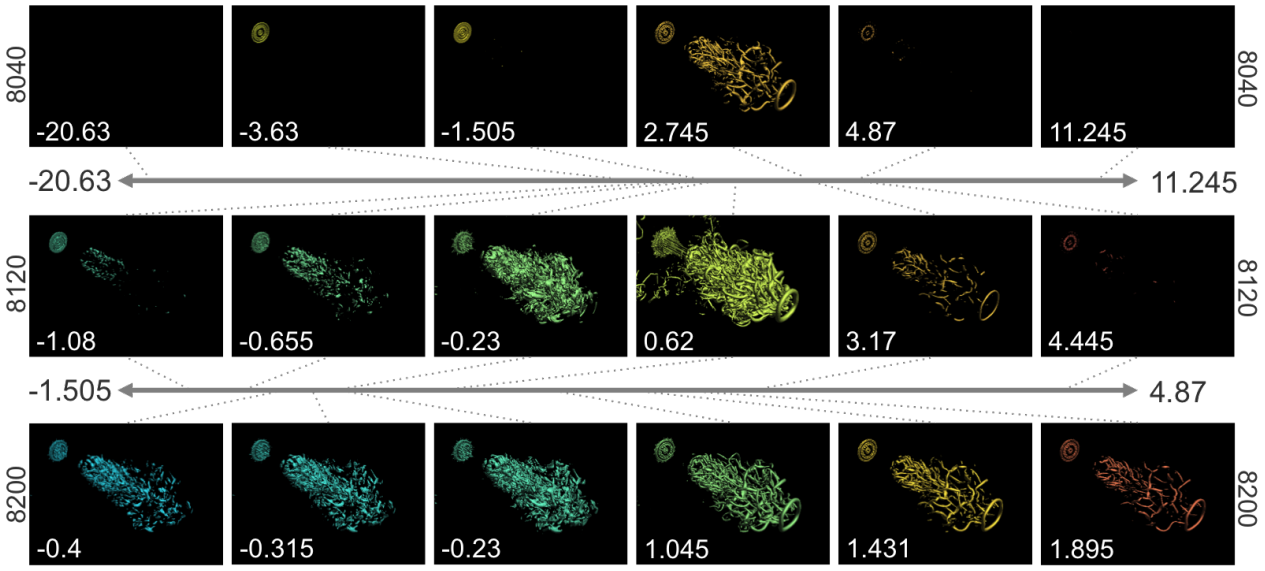
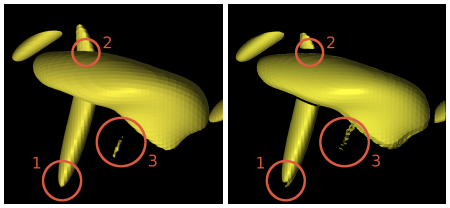


Figure 14: Images showing the in situ process of fine tuning the isovalues array. The top row of images are generated at simulation time step 8040 with 16 uniformly spaced isovalues across the scalar value range. Only selected images are shown. The middle row of images are generated with 16 isosurfaces within the range [-1.505, 4.87]. The resulting isosurfaces are more meaningful yet some of them are still unwanted. The bottom row of the isosurfaces are generated by concentrating at two separate ranges: [-0.485, -0.23] and [1.045, 1.895]. Thus, generating isosurfaces that match our expectations.



(a) Ground Truth (b) Interpolated Depth Layer

Figure 15: Limitations of depth layer interpolation (red circles): 1) Artifact due to rapid change of the normal vectors, 2) missing interpolation segments at the overlapping areas, and 3) low density interpolation segments due to small features.

5.1 Isosurface Rendering

Since depth maps capture the smallest depth values of isosurfaces, only the front face of an isosurface can be visualized. If the isosurface is a manifold or multiple layers of isosurfaces are the same isovalue, our current system is unable to visualize the other layers. Multiple layers of the same isosurfaces can be stored to solve this problem, but this drastically increases the storage requirement.

An isosurface can be rendered using an interpolated depth layer. The quality of the rendered isosurface greatly depends on the reconstructed depth layer, which in turn greatly depends on the interpolation segments found between the two adjacent depth layers. The low density areas of interpolation segments are typically prone to artifacts, and the low density areas are usually found at the overlapping areas, sharp features, and small features. Figure 15 shows a comparison of interpolated depth layer and the ground truth, where various artifacts are shown in the interpolated depth layer.

5.2 Feature Extraction

There are two major limitations of our image-space isosurface based feature extraction method. 1) We can only extract features that are defined by isosurfaces. 2) Only post-hoc visual tracking is supported, which precludes feature-based statistics such as the tracking of a quantity for a given feature over time. For quantities that can be calculated in image space, it may be possible to pregenerate desired values and encode them into maps in much the same way that depth maps are generated currently. An example of such a quantity might be surface temperature or curvature. However, quantities relying on full 3D shape of the feature, such as volume, are not directly supported with this technique and would require additional in situ precalculation.

We compare our image-based feature extraction method with 3D feature extraction. The comparison results are shown in Figure 16. Feature extraction on depth maps greatly depends on the view configuration. When features are obstructed by other features, as in Figure 16b, image-based feature extraction considers the visible segments of the obstructed features to be individual features, while 3D feature extraction is able to correctly connect the obstructed features. There are also cases where the depth difference between different features are within the user defined threshold. As a result, image-based feature extraction identifies them as a single feature while in 3D they are separate.

6 Future Work

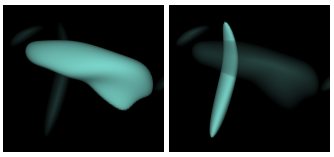
Depth layer interpolation serves as a user feedback technique for generating future depth maps. Presently, there is no quantitative measure for conveying the quality of interpolated depth layers. We plan to develop a model for uncertainty evaluation of the interpolated depth layer to better guide the users.

Currently, our system cannot correlate the features with different isovalues. Contour trees are a good candidate for connecting the features within the same depth maps.

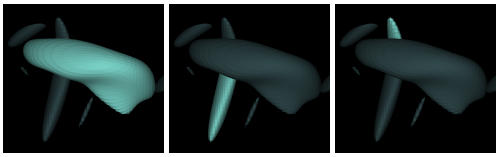
Our technique suffers from the locked view frustum. Limited view exploration can be enabled using an incremental warping algorithm described by Shade et al. in [27].

7 Conclusion

We have introduced an in situ visualization method using depth maps. Depth maps enable post-hoc visualization of multiple semi-transparent isosurfaces with any isovalues, providing a way to explore the flow field without requiring the original 3D volumetric data. Furthermore, feature extraction and tracking can be done on



(a) Ground truth feature extraction result on volumetric data



(b) Feature extraction result on depth maps

Figure 16: Comparing FET on volumetric data and FET on depth maps. The extracted features are in bright green color while the other features are rendered as semitransparent or darkened. The leftmost image in both (a) and (b) show that the unobstructed features can be correctly extracted. However, the feature shown in the right image in (a) is partially hidden. As a result, feature extraction with depth maps recognize the two pieces as two separate features, as shown in the images on the right in (b).

top of depth maps, allowing features to be tracked both forward and backward in time. It is also possible to tune the in situ depth maps generation for future time steps to better capture modeled features of interest. The concept of extracting and tracking flow features in image space in terms of depth maps is a sound and powerful one, suggesting extensions to support other visualization operations.

Acknowledgements

This research is sponsored in part by the U.S. National Science Foundation via grants NSF DRL-1323214 and NSF IIS-1320229, and also by the U.S. Department of Energy through grants DE-SC0005334, DE-FC02-12ER26072, and DE-SC0012610. The forced isotropic turbulence dataset is obtained from the John Hopkins Turbulence Databases.

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] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proc. SC '14*, pages 424–434, 2014.

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

[4] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *Proccedings of SC '12*, pages 1–9, Nov 2012.

[5] T. Biedert and C. Garth. Contour Tree Depth Images For Large Data Visualization. In *Proceedings of EGPGV '15*, 2015.

[6] 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.

[7] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. In *Proceedings of IEEE VIS '05*, pages 191–198, Oct 2005.

[8] E. Dedu. Bresenham-based supercover line algorithm, 2001.

[9] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro. Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework. In *Proc. LDAV '13*, pages 67–75, Oct 2013.

[10] M. Dreher and B. Raffin. A flexible framework for asynchronous in situ and in transit analytics for scientific simulations. In *Proceedings of CCGrid '14*, May 2014.

[11] O. Fernandes, S. Frey, F. Sadlo, and T. Ertl. Space-time volumetric depth images for in-situ visualization. In *Proceedings of LDAV '14*, pages 59–65, Nov 2014.

[12] S. Frey, F. Sadlo, and T. Ertl. Explorable volumetric depth images from raycasting. In *Proc. SIBGRAPI '13*, pages 123–130, Aug 2013.

[13] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In *Proceedings of IEEE VIS '03*, pages 209–216, Oct 2003.

[14] C. Johnson, S. Parker, C. Hansen, G. Kindlmann, and Y. Livnat. Interactive simulation and visualization. *Computer*, 32(12):59–65, 1999.

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

[16] O. Kreylos, A. M. Tesdall, B. Hamann, J. K. Hunter, and K. I. Joy. Interactive visualization and steering of CFD simulations. In *Proceedings of VISSYM '02*, pages 25–34, 2002.

[17] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C. S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

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

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

[20] K. Moreland, W. Kendall, T. Peterka, and J. Huang. An image compositing solution at scale. In *Proc. SC '11*, pages 25:1–25:10, 2011.

[21] 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 PDAC '11*, pages 1–6, 2011.

[22] C. Muelder and K.-L. Ma. Interactive feature extraction and tracking by utilizing region coherency. In *Proc. of PacificVis '09*, April 2009.

[23] S. D. Ramsey, K. Potter, and C. Hansen. Ray bilinear patch intersections. *Journal of Graphics Tools*, 9(3):41–47, 2004.

[24] 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.

[25] M. Rivi, L. Calori, G. Muscianisi, and V. Slavnic. In-situ visualization: State-of-the-art and some use cases. *PRACE White Paper*, 2012.

[26] E. R. Schendel, S. V. Pendse, J. Jenkins, D. A. Boyuka, II, Z. Gong, S. Lakshminarasimhan, Q. Liu, H. Kolla, J. Chen, S. Klasky, R. Ross, and N. F. Samatova. Isobar hybrid compression-I/O interleaving for large-scale parallel I/O optimization. In *Proc. of HPDC '12*, 2012.

[27] J. Shade, S. Gortler, L.-w. He, and R. Szeliski. Layered depth images. In *Proceedings of SIGGRAPH '98*, pages 231–242, 1998.

[28] H. Theisel and H.-P. Seidel. Feature flow fields. In *Proceedings of VISSYM '03*, pages 141–148, 2003.

[29] 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.

[30] 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 EGPGV '11*, pages 111–120, 2011.

[31] B. Whitlock, J. M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of EGPGV '11*, pages 101–109, 2011.

[32] H. Yi, M. Rasquin, J. Fang, and I. Bolotnov. In-situ visualization and computational steering for large-scale simulation of turbulent flows in complex geometries. In *Proc. of IEEE Big Data '14*, Oct 2014.

[33] H. Yu, C. Wang, and K.-L. Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *Proc. of SC '08*, 2008.