Fast uncertainty-driven large-scale volume feature extraction on desktop PCs

Conference Paper · October 2015

DOI: 10.1109/LDAV.2015.7348067

3 authors, including:

Jinrong Xie
University of California, Davis
10 PUBLICATIONS **53** CITATIONS

SEE PROFILE

Fast Uncertainty-driven Large-scale Volume Feature Extraction on Desktop PCs

Jinrong Xie* Franz Sauer† Kwan-Liu Ma‡

University of California, Davis

ABSTRACT

The ability to efficiently and accurately extract features of interest is an extremely important tool in the field of scientific visualization as it allows researchers to isolate regions based on their domain knowledge. However, the increasing size of large-scale datasets often forces users to rely on distributed computing environments which have many drawbacks in terms of interaction and convenience. Many of the current feature extraction techniques are designed around these distributed environments. The ability to overcome the memory and bandwidth limitations of desktop PCs can broaden their usability towards large-scale applications. In this work, we present a new hybrid feature extraction technique which combines GPU-accelerated clustering with the multi-resolution advantages of supervoxels in order to handle large-scale datasets on standard desktop PCs. Furthermore, this is paired with a user-driven uncertainty-based refinement approach to enhance extraction results into a desired level of detail. We demonstrate the effectiveness and interactivity of this technique using a number of application specific examples utilizing large-scale volumetric datasets.

Keywords: Segmentation, multi-resolution, hierarchical clustering, volume data, large-scale data.

1 INTRODUCTION

Feature extraction in volumetric datasets is an extremely popular tool with many different uses. Firstly, it allows researchers to isolate regions based on a desired set of qualities. The location, shape, and evolution of these regions allows users to gain scientific insight into the system they are studying. Next, it is a powerful data reduction tool. With the growing size of datasets, the ability to extract and operate on subsets from data has become a necessity. Lastly, feature extraction allows a system to distinguish objects from a "background" and even from one another. Such a distinction becomes important for techniques such as transfer function generation and even has numerous applications in computer vision.

The increasing size of large-scale datasets often forces researchers to rely on the increased computational resources of a distributed computing environment. As a result, many of the current feature extraction techniques which operate on large-scale data are designed for a distributed setting. However, these environments do have a number of drawbacks in terms of interactivity and convenience. For example, there is the need to compete with other users over computational resources resulting in long queues and wait times before executing a job. In addition, the computing environment and available toolkits/libraries are often limited. In the field of visualization, interactivity in data exploration becomes key. Working in a remote environment limits any feedback and interactivity according to available network bandwidth. As a result, many scientists tend to prefer analyzing their datasets locally on a desktop PC where they have more control over their system. The ability to overcome the memory and computational limitations of desktop PCs can make them more applicable towards large-scale applications.

Traditional feature extraction and data clustering techniques can generally be classified into two main approaches. The first approach evaluates the volume as a whole and groups discrete components based on a similar set of properties. This can provide a meaningful approximation of the underlying segments, but may lose track of precise data discontinuities. Using a voxel-based K-means clustering would be an example of such an approach. This normally requires the number of clusters to be known a priori and has a complexity that depends on the number of initial clusters, the data domain size, and the number of iterations it takes to converge. The other approach works on a local level and identifies coherent or connected components. It can accurately detect discontinuities but is sensitive to noise perturbation and may segment a region into too many components. An example of this approach would be seeded region growing and has a complexity based heavily on the size and resolution of the extracted feature.

Each method alone has a number of advantages and disadvantages which become increasingly evident in large-scale datasets. In this work, we present a new hybrid feature extraction technique which combines a GPU-accelerated version of the SLIC [1] clustering technique with the multi-resolution advantages of "supervoxels" in order to handle large-scale datasets on standard desktop PCs. Rough clustering results are then further refined using an uncertainty-driven approach to enhance extraction results into a desired level of detail. A hybrid approach like the one presented here allows one to retain many of the advantages of various feature tracking techniques. Furthermore, our GPU-accelerated method allows users to process and explore their large-scale datasets in real time on a local desktop PC without the drawbacks of a remote distributed system.

In this paper, we present our new hybrid extraction technique and make the following contributions:

- We present a new GPU-accelerated hybrid feature extraction technique for large-scale data.
- We pair this technique with the multi-resolution advantages of supervoxels in order to handle large datasets on a desktop PC.
- We provide an uncertainty-driven feature refinement method to enhance extraction results.

We demonstrate the effectiveness of our algorithm using several real world large-scale datasets, including large combustion, ocean, and flow simulation data. Using each of these datasets we can achieve real time extraction and exploration capabilities on a local desktop computer.

2 RELATED WORK

Feature extraction is a well studied technique for analyzing spatial-temporal data in many scientific simulations both in scalar and

*e-mail: jrxie@ucdavis.edu

†e-mail: fasauer@ucdavis.edu

‡e-mail: ma@cs.ucdavis.edu

vector field study [13, 15]. There are plenty of papers covering the extraction and evolution of individual features, such as [3, 4, 10, 16, 17, 23]. Based on the data granularity and type of features that need to be extracted, different segmentation schemes are used. Many approaches begin with low level feature attributes such as the intensity, color, and gradient of a pixel/voxel and/or its neighborhood. These tend to iteratively group the similar components into larger clusters.

Among many techniques for feature extraction, region growing can be considered as an effective bottom-up approach to detect and isolate features of interest that have spatial or temporal coherency. It is widely studied in the image processing and 3D volumetric data analysis to assist classification and assessment steps [9]. Muelder et al. [14] further improved the region growing performance by utilizing region coherency over consecutive time steps.

On the other hand, other approaches favor multi-resolution schemes that can analyze volume data at various scales. Huang et al. [8] employed morphological operations to capture structural information at various scales. The contour tree [5, 19, 20] and Reeb graph [23] are also used to define the features of interest.

Many of the aforementioned approaches work well when the data set is a manageable size. However, as the simulation resolution keeps increasing, it has become necessary to accompany the data generation phase with a scalable solution for visualizing and analyzing the simulation results to extract valuable insight. Work has also been done to parallelize and accelerate these techniques [18, 22]. These approaches directly work on individual voxels whose attributes (scalar value, gradient, etc.) are used in determining a similarity metric. In contrast, Bremer al et. [3] uses a topological technique to construct a hierarchical merge tree as indexing for feature representation that is orders of magnitude smaller than the original simulation data. However, their method targets feature extraction as a post-processing step, while our approach is suitable for in-situ settings where dumping all time steps in full resolution is not feasible given the increasing gap between data generation speed and limited I/O bandwidth.

While the aforementioned techniques have been proven to be successful in their respective domains, they do not directly provide enough flexibility for users to balance the quality of segmentation results with the computational complexity. Many of these techniques work on a raw pixel/voxel level making them computationally inefficient in large-scale applications since many 3D datasets can contain several billions of grid points, each with multiple variables. In the large-scale data visualization setting, it is often ideal to provide a fast response to a user's query on the data in question at a reasonable level of accuracy. This can then be further refined on an as needed basis depending on a desired level of accuracy. The hybrid technique presented in this paper focuses on providing such capabilities.

3 METHODS

Our approach is motivated by the idea of giving users fast response time when exploring and extracting features of large volume datasets on a local desktop machine. As a result, many of the steps involved in our algorithm are designed to provide a (potentially) user-controlled balance between the performance and accuracy of each step. Different datasets have a very large range in underlying data patterns and in many cases must be treated differently to achieve a quick accurate extraction result.

3.1 Overview

An overview of our approach can be seen in Figure 1. Our approach starts with a preprocessing step that partitions the volumetric data into a set of equally sized blocks called supervoxels. Each supervoxel can represent a 2^3, 3^3, etc. space of voxels and is similar to a

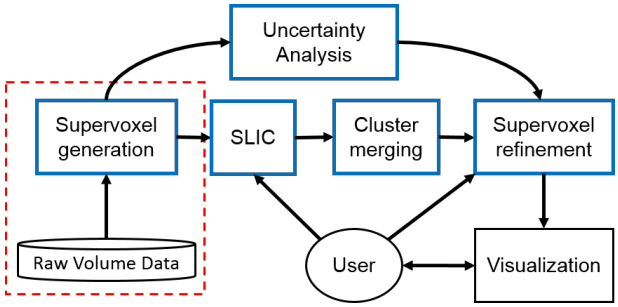


Figure 1: An overview of our system pipeline, blue components represent the technique introduced in this paper. The section outlined in red can be done either as a preprocessing step or during a simulation run (in situ).

down-sampling of the full volumetric dataset. This ensures interactive response times when generating rough initial extraction results. While this is often a fast preprocessing step, it can alternatively be implemented in situ and saved along with the rest of the simulation data. This is to alleviate the burden of disk I/O when working with massive datasets. Supervoxels are generated using the GPU where additional statistical metrics are computed, such as the average intensity value and standard deviation of internal raw voxel values. These metrics are later used to determine which supervoxels need to be refined to generate a more accurate extraction result.

Next, we apply a GPU-accelerated version of the Simple Linear Iterative Clustering (SLIC) algorithm [1] to partition similar supervoxel blocks into a set of clusters. Note that the term supervoxel in [1] is used differently than in this paper. Since the number of clusters is predetermined by this algorithm, we include a merge step which combines similar clusters in a hierarchical fashion. Lastly we use uncertainty metrics based on the standard deviation of values within a supervoxel to guide users in selecting which parts of the extraction result must be refined. This refinement step will fetch the high resolution raw voxel data only for portions of the extraction result that display a high uncertainty in accuracy. The final result will therefore contain data values at both the lower supervoxel resolution and the higher raw data resolution depending on the underlying nature of the dataset.

3.2 Supervoxel Generation

The supervoxel generation step is similar to many approaches used to down-sample large scale data. In our implementation, each supervoxel represents an equally sized block of the original volume data. The resolution of the supervoxel depends on the number of raw data voxels contained inside. For example, we say that a supervoxel containing 3x3x3 = 27 raw data voxels has a resolution of 3. We can choose the resolution of supervoxels to use based on the size of the original raw dataset with larger resolutions required as the data becomes larger.

This conversion step is implemented using GPU-acceleration. The raw volume data is loaded into the GPU as a 3D texture, where it resides in global memory. We then define one GPU thread for every supervoxel which accesses the raw data values of the volume texture. The average intensity and standard deviation are computed for each supervoxel in parallel. This intensity value is used as input into clustering in the next step while the standard deviation is used later during the uncertainty-based refinement step. Since this step can be executed very quickly, we perform it at run time and give users control over the desired supervoxel resolution. Note that if the volume does not fit entirely in GPU memory we process it in chunks. Performance results for the supervoxel generation step are discussed in more detail in the results section of this paper.

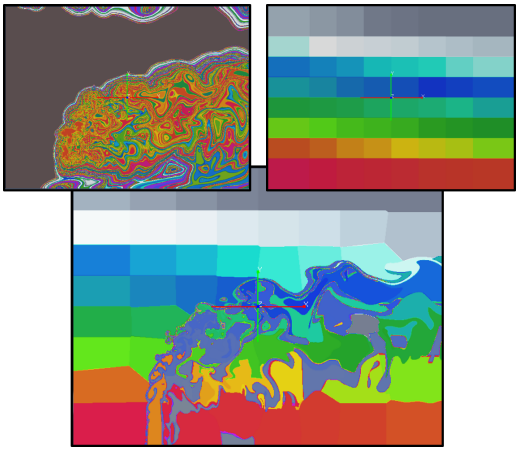


Figure 2: SLIC clustering results on a combustion simulation dataset shown on a 2D slice. This particular example shows a balance between the proximity and intensity weights used. The top left and top right images show the two extreme cases where the proximity and intensity weights are set to zero respectively.

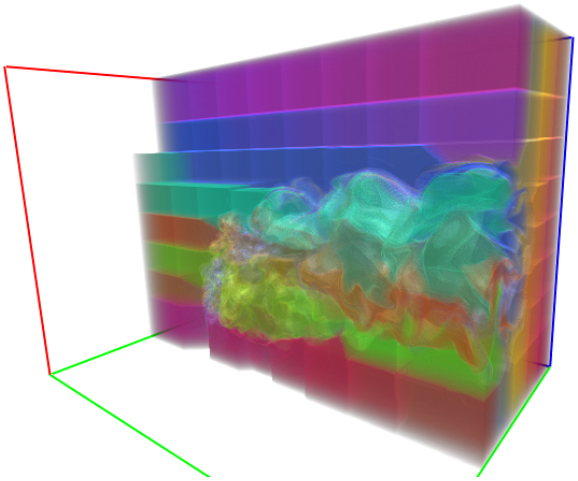


Figure 3: The result of the first iteration after our 3D SLIC implementation on a combustion simulation data set. The color encodes different cluster IDs. In this case, 8x8x8 cluster centers in a grid layout are initialized. Even after one iteration, the salient structure and similar features are captured by the SLIC approach.

3.3 3D Supervoxel Clustering via SLIC

To efficiently identify features from the supervoxel volume, we transform a simple but efficient image clustering technique called Simple Linear Iterative Clustering (SLIC) [1] to segment the supervoxel volume into a number of different clusters. In our implementation, this algorithm is treated in a 3D sense and accelerated using the GPU to ensure fast clustering times. Since we are operating on the supervoxel volume rather than raw large-scale data, this clustering result can also be done on the fly in real time.

The SLIC approach is very similar to k-means clustering, but varies in two main aspects. First, instead of computing the distance from each cluster center to all voxels in the entire domain, the search space is limited to within a region proportional to the cluster size. Second, the distance metric used combines intensity and spatial proximity which provides control over the size and compactness of the clusters. The initial cluster centers are selected on a regular grid in 3D volume with interval length of $S_i (i \in \{0,1,2\})$ for each dimension. The length S_i is determined by the number of initial clusters k_i and the extent of the volume N_i in each corresponding dimension; $S_i = N_i/k_i$. For each iteration, the algorithm assigns each voxel to the most similar cluster, updates the cluster centers, and repeats until a certain stopping criteria is met.

SLIC is computationally less expensive than the conventional k-means algorithm because its distance metric measuring similarity is computed within a $2S_0 \times 2S_1 \times 2S_2$ local region centered at each cluster center. We define the distance metric as:

$$d_{k,i} = w_1 ||c_k - v_i||_2 + w_2 |I_k - I_i|$$

where I_k and I_i are the scalar values at the kth cluster center and voxel i respectively; c_k is the position vector of cluster center k, and v_i is that of voxel i. w_1 and w_2 are weights. Increasing w_1 would result in a clustering biased towards spatial proximity and as a result preserves more compactness. Increasing w_2 would cause the clustering to adhere more tightly to the boundaries between varying scalar values. Figure 2 demonstrates a series of different clustering results on a 2D slice of a volume data with different combinations of w_1 and w_2. Figure 3 shows an example of a clustering result in 3D after just one iteration.

By localizing the search within a $2S_0 \times 2S_1 \times 2S_2$ volume during clustering, the computational complexity of the SLIC algorithm is independent of k and scales with $O(N)$. In contrast, the classical k-means has $O(kN)$ complexity for each iteration. This distinction is important when dealing with large-scale 3D volume data rich in features where k must also be large. There are also other popular algorithms whose clustering results are similar to that of SLIC such as Turbopixels [12] and Quickshift [21]. However, SLIC is superior to many of these approaches in terms of segmentation quality and is faster with a smaller memory requirement. Lastly, we accelerate this step using the GPU. The supervoxel volume is loaded as a 3D texture into global memory. The embarrassingly parallel nature of this technique allows us to compute the distance metric between each cluster and supervoxel simultaneously via a separate GPU thread.

3.4 Hierarchical Merging

The aforementioned SLIC method often over-segments the supervoxel volume into many small clusters, and within each cluster the supervoxels share similar intensities and are within a close spatial proximity. After applying SLIC, small clusters of supervoxels are generated. We use these clusters to generate a hierarchical clustering result by merging the clusters of supervoxels. If one imagines a hierarchical clustering as tree-like structure, the output from the SLIC step would generate nodes found towards the bottom of the tree and merging clusters will build up this structure towards a root node.

One disadvantage of employing SLIC as a clustering step is that it produces many redundant clusters in homogeneous regions such as the background. As seen in Figure 3, while capturing basic flame structure, SLIC also creates many similar cube-like clusters in the background region. Ideally, one cluster suffices to represent the background even though it has subtle variation in its scalar value. Therefore, it is necessary to combine similar clusters into one larger cluster. We would like to first consider those clusters that are both spatially adjacent and are close in terms of the user-specified distance metric. Therefore, a reduce operation is needed to coalesce groups of clusters identified by the SLIC step and reconstruct the connected structure.

Unfortunately, SLIC does not compute the topological connectivity of the output clusters (similar to the marching cubes algorithm that produces a 'soup' of disconnected triangle elements [13]). As the initial step of cluster merging, we work on building up the topo-

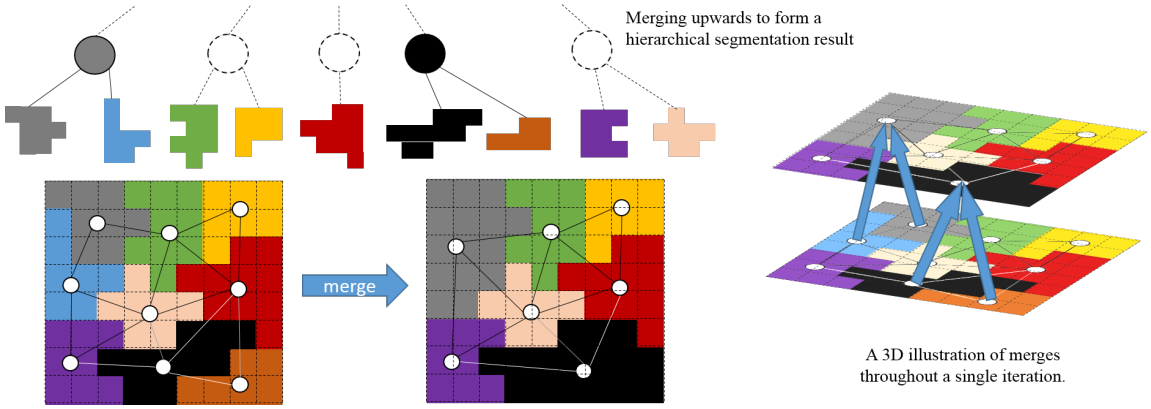


Figure 4: An example of merging sets of clusters to construct a hierarchical partitioning. The blue and gray clusters are merged. This continues until there is a desired level of variation within each cluster. For example, the green and gold clusters may merge in the next iteration.

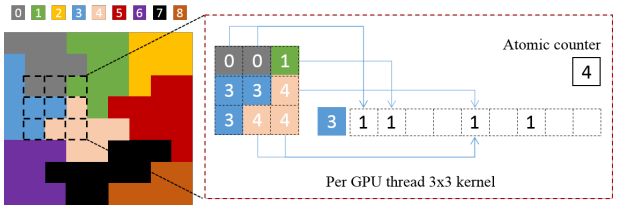


Figure 5: A 2D illustration of running template scheme to detect neighboring clusters for every cluster. Each thread is mapped to the center of a 3×3 template and iterates each pixel within the template. The neighboring clusters i are marked on the ith column of row j corresponding to the cluster j at the template center.

logical connectivity of clusters on GPU. The key idea of our topology reconstruction step is to build an undirected graph G such that each vertex $v_i \in V$ represents cluster i. There exists an edge $e_{ij} \in E$ if and only if cluster i and j are spatially connected.

To efficiently build-up the adjacency matrix M where $M[i,j] = 1$ indicates an edge $e_{ij} \in E$ in graph G, we utilize a running template over each supervoxel in the cluster and check the neighboring cluster information. At each stop of the template, a $3 \times 3 \times 3$ supervoxel-wise spatial region is searched. Let i denote the cluster ID of the supervoxel at template center and j denote the cluster ID of one of 26 neighboring supervoxels in the template. $M[i,j]$ is tagged with 1 if $i \neq j$, otherwise $M[i,j] = 0$. The running template scheme is computationally intensive since each supervoxel will check its 26 neighborhoods. However, each supervoxel is independent from the others, and we can parallelize the process by assigning a GPU thread to a supervoxel. Each thread checks its neighboring supervoxels in parallel and tags the corresponding entry on the adjacency matrix. A 2D example case, is shown in Figure 5. A thread is assigned to work within a 3×3 template. The center of the template belongs to cluster 3 and it has neighboring clusters 0, 1, and 4 shown in different colors. Hence column 0, 1, and 4 of row 3 in the adjacency matrix are tagged with 1.

Once the graph is constructed we can merge similar clusters by combining the respective nodes in the graph and removing their connecting edge. This step can be performed on the CPU since the number of clusters is much smaller than the number of individual voxels. An example of a merge can be seen in Figure 4.

3.5 Uncertainty-based Refinement

The aforementioned steps produce an extraction result on the supervoxel level. However, studying the full resolution data is often necessary in identifying subtle patterns. This final step focuses on refining the supervoxel based clustering in regions of high uncertainty. The uncertainty metric we use is based on the standard deviation of the scalar data values of the raw voxels contained within a supervoxel. This has already been computed during the supervoxel generation step. Instead of refining every supervoxel in our cluster of interest (a very computationally intensive task), users can choose a desired level of uncertainty criteria through which the final result must meet. Any supervoxels whose uncertainty is higher than the user-defined threshold are therefore selected for refinement.

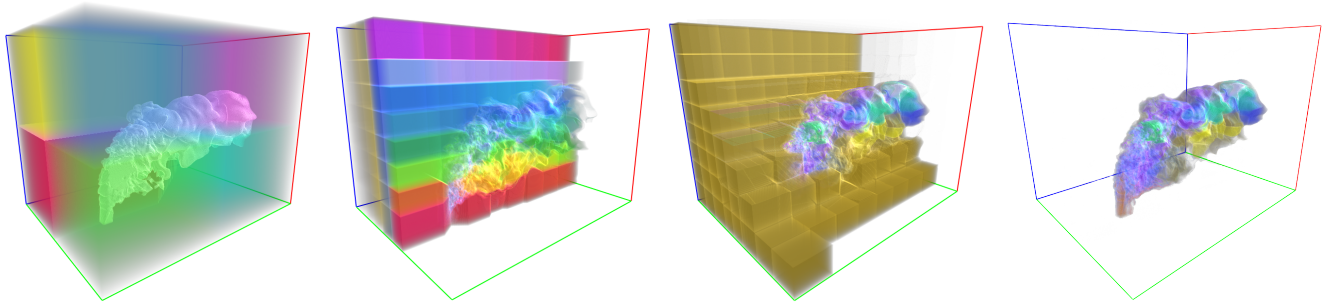
This refinement step requires retrieving the original raw data values associated with the supervoxel from disk. As they are loaded into memory, each raw voxel value is compared against the clustering attributes used to generate the particular supervoxel cluster. If this distance criteria is met, then the raw voxel is kept, otherwise it is checked against any neighboring clusters and potentially added to them. Once a supervoxel has been refined, its uncertainty value drops to zero since we are now operating on the raw data values. This results in an extraction result with multi-resolution components and is represented as an additional level in our hierarchical extraction result.

3.6 Implementation

The implementation is essentially a two-level technique with low resolution clustering blocks and high resolution refinement bricks. To generate supervoxels, we use OpenGL compute shader on Desktop PC or CUDA plus MPI on parallel supercomputer like Titan. SLIC algorithm can also be implemented via either compute shader or CUDA depending on their availability. Visualization of clustered data like the one in Figure 8 is achieved by GPU ray-casting through the coarse level SLIC cluster volume. The cluster volume is stored in GPU 3D texture while the refinement blocks are dynamically generated based on the uncertainty metric and are therefore stored and fetched separately through 3D bindless textures since they dramatically increase the number of unique textures available to shaders at run-time. We utilize the hardware trilinear interpolation within the same level, however, cross level interpolation is not considered in our current implementation, thus discontinuity is visible across the boundary.

4 Results

We have conducted tests using three datasets from different application domains. The first is a dataset generated from the turbulent combustion simulations performed at Sandia National Laboratories [6]. The second is a tropical oceanic data simulated by the National Oceanic and Atmospheric Administration Geophysical Fluid Dynamics Laboratory using Community Climate Model (CCSM): Parallel Ocean Program 2 (POP2) [7]. The third dataset comes from



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

Figure 6: (a) The SLIC clustering results using $2 \times 2 \times 2$ initial cluster centers. (b) The clustering results after one iteration using $8 \times 8 \times 8$ cluster centers. (c) The result after applying merge. Many redundant cluster centers in the background are coalesced into one supercluster colored in dark yellow, while the clusters corresponding to the flame structure remain unchanged with a slight color variation due to the reassignement of cluster id's. (d) Removing the background cluster and emphasizing the entire extracted flame features using intensity value and Euclidean distance as distance metric. The supervoxels in each extracted cluster share both similar intensity and spatial proximity.

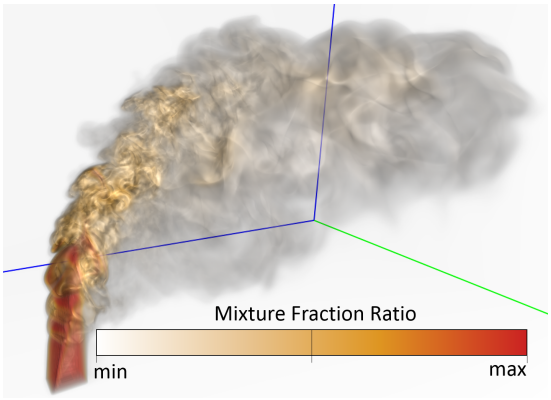


Figure 7: A 3D volume rendering of the combustion simulation data using a mixture ratio variable. Fuel is injected from the lower left part of the image into a region containing oxidizer. As it travels to the top right, it mixes and burns with the surrounding medium.

large-scale flow studies conducted by Argonne National Laboratory [11].

4.1 Combustion Data

The phenomenon of combustion in 3D highly turbulent conditions and mixed-modes has many new physical and chemical properties. To better understand the complicated interactions, direct numerical simulation is required to record and delineate the key turbulence-chemistry interactions. Scientists at Sandia National Laboratories have developed S3D [6] to tackle this challenge at large scale using massive parallel supercomputers. The data set in our study has a spatial resolution of $704 \times 540 \times 550$ and each grid point contains several variables. During the experiment, we consider the mixture fraction field since it is one of the two key combustion parameters associated with turbulent mixing and autoignition. It is a mixing measure of fuel and oxidizer. The highest value represents pure fuel and zero represents pure oxidizer. Figure 7 shows a 3D volume visualization of the mixture ratio.

Figure 6 shows examples of applying the clustering scheme to the combustion dataset. After the merge step, we can remove the background and focus on exploring clusters that make up the flame itself. Since we are clustering based on the mixture fraction, such an extraction result is effective at highlighting the complex structures that form in such a turbulent environment.

We also illustrate the usefulness of the refinement step. While clustering and visualizing the supervoxel volume greatly improves

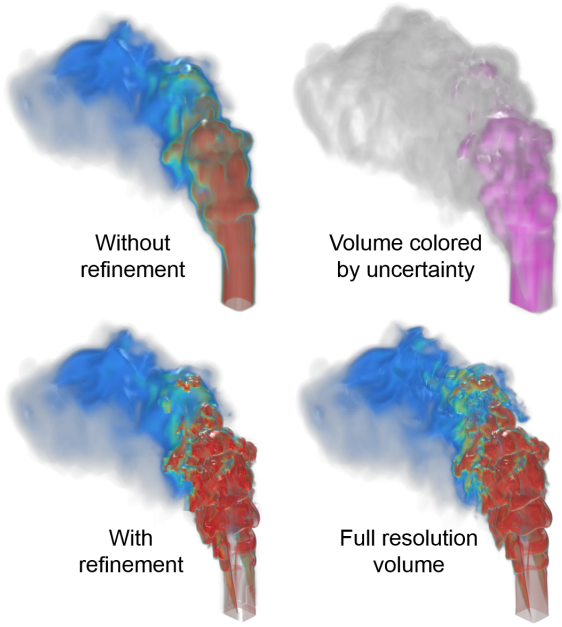


Figure 8: Top-left) Not performing refinement hides many of the details present in the flame structure. Top-right) Coloring supervoxels according to their level of uncertainty. Pink regions represent supervoxels which need to be refined to produce a more accurate extraction result. Bottom-left) Supervoxels with high levels of uncertainty are refined revealing additional details in the data. Bottom-right) The full resolution volume for comparison.

performance in large scale data, many important details can become lost. Figure 8 shows supervoxels colored according to their uncertainty level. We can see that the level of uncertainty varies throughout the volume and highlights which supervoxels needs to be refined to obtain a more accurate extraction result. Refining those supervoxels by sampling the raw high resolution volume reveals the finer structure of the combustion flame and results in an image very similar to the full resolution volume.

4.2 Ocean Data

The POP simulation [7] is an ocean circulation model that is integral to many climate based research endeavors. Its complex 3D nature allows for a detailed study of turbulent mixing processes between surface currents and the deep ocean. The ocean simulation dataset has a resolution of $3600 \times 2400 \times 42$. We use the magnitude

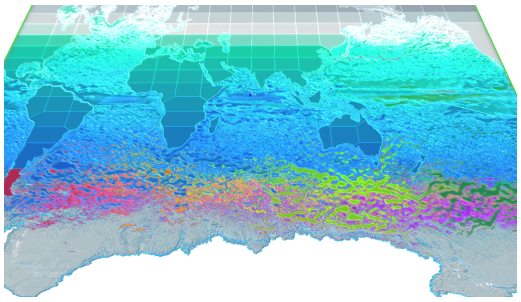


Figure 9: A 3D volume rendering of the clustering results on the ocean simulation dataset based on velocity magnitude. The presence of many interconnected features near the south pole show an increased variation in velocity when compared to regions near the equator.

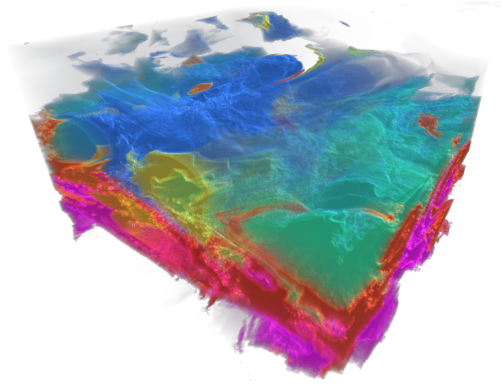


Figure 10: A 3D volume rendering of the clustering results on the flow simulation dataset. The layer-like structures in the stratified flow become clear once extracted.

of the ocean velocity field as the intensity value. In addition, since the depth of the ocean is small compared to its surface area, we use an asymmetric $20 \times 20 \times 1$ layout of initial SLIC cluster centers.

Figure 9 shows the output of our extraction scheme on this dataset. It is immediately evident that there are several interesting structures that form in the southern hemisphere, whereas the regions near the equator are more homogeneously distributed. Since we are clustering based on the velocity field, the large numbers of intertwined clusters, each containing a similar velocity magnitude, suggest an increased amount of turbulence in those regions. This is to be expected since regions near the poles are more strongly affected by the Earth's rotation and deep sea mixing. Real time 3D feature extraction in such a dataset can allow users to explore the intricate structures and driving forces behind this system.

4.3 Flow Data

The complex interaction between waves and turbulence in flows drives many systems in a multitude of engineering applications and physical phenomena. This particular simulation and study "Parameter Studies of Boussinesq Flows" [11] focuses on understanding the coupling between waves and slow motion by studying layer-like structures in stratified flows as well as columnar structures in rotating flows. With a size of $4096 \times 4096 \times 4096$, it is the largest of all of our test cases. Figure 10 shows an example output using our extraction scheme. As expected, the resulting features represent the layer-like structures found in the stratified flow and can aid scientists in studying upscale and downscale energy transfers in these systems.

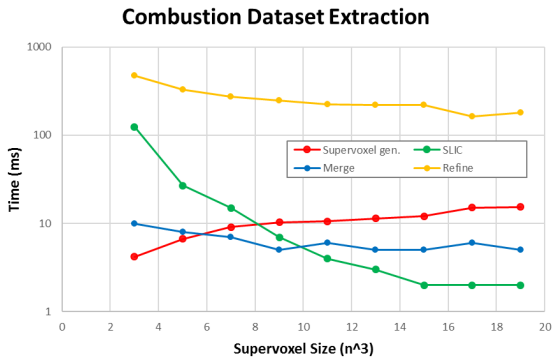


Figure 11: A breakdown of the performance results for each of the extraction steps vs. supervoxel size for the combustion dataset.

4.4 Performance Results

The timing was measured on a desktop PC with two 2.4 GHz Intel Xeon CPUs and 24 GB DDR3 RAM with one GTX Titan X graphics card with 12 GB of GPU memory.

We begin with the combustion dataset and investigate how different supervoxel sizes affect the performance of each of the steps involved in our feature extraction technique. We focus on this particular parameter since it has the largest impact on the overall extraction time. In this example, 512 SLIC clusters were used for each test. Graph 11 shows the timing results for each step of the algorithm for a number of supervoxel sizes. We can see that the supervoxel generation time increases slightly as the supervoxel size increases. This is due to the fact that a larger number of raw voxels must be sampled and used to compute the necessary statistical information (mean and standard deviation). On the other hand, the time for the SLIC step decreases as supervoxel size increases because there are fewer supervoxels that need to be clustered. In addition, the merge step time (which also includes the time to generate connectivity information between clusters) also decreases as the supervoxel size increases. Once again, this is due to the ease of working with lower resolution volume. The time to refine the volume into a desired level of detail has the largest cost and depends heavily on the underlying structure of dataset. It will be shown soon that the other test datasets exhibit different trends when it comes to the cost of the refine step. For this particular configuration the optimum supervoxel size is 17^3 with an overall extraction time of 187 ms showing that this type of analysis can be done interactively in real time.

The ocean dataset shows similar trends. In this example, 400 SLIC clusters were used for each test. Graph 12 shows the timing results for each step of the algorithm. These times represent very similar trends to those produced by the combustion dataset. However, the overall times are higher since the ocean dataset is larger in scale. Nevertheless we can achieve decently interactive speeds when extracting features in the ocean data. The optimum supervoxel size is 19^3 with an overall extraction time of 2299 ms. Since this is the last data point, it could be possible that a larger supervoxel size would result in an even lower extraction time.

The combustion and ocean datasets are manageable enough in size so that disk read times are small enough to ensure interactive speeds when performing the supervoxel generation step on a local machine on the fly. However, the flow dataset, which consists of hundreds of GB of information per time step, can incur a very large I/O overhead. As a result, it is often desirable to perform the supervoxel generation step in situ during the corresponding simulation. In this implementation, the simulation will not only produce the large-scale high resolution volume, but the much smaller supervoxel representation as well.

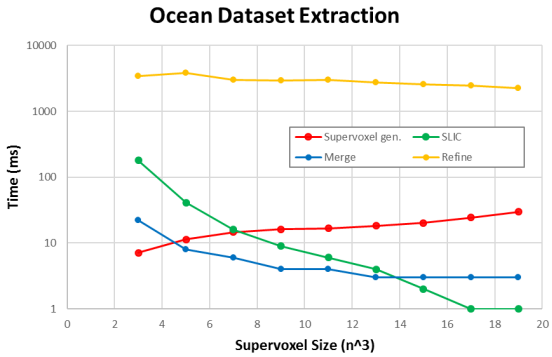


Figure 12: A breakdown of the performance results for each of the extraction steps vs. supervoxel size for the ocean dataset.

Table 1: Strong scaling. In situ supervoxel generation timing for the flow dataset with a constant 8x8x8 supervoxel size.

#Nodes	CPU to GPU (ms)	Supervoxel gen. (ms)	Write (ms)
128	1133	74	527
256	755	37	630
512	618	19	694

We estimate the amount of computation necessary to construct the supervoxel representation if it were done in situ by conducting performance tests on the Titan supercomputer at Oak Ridge National Laboratory. Table 1 shows the timing results for constructing an 8x8x8 supervoxel representation using a varying number of compute nodes. We assume that the data for a particular timestep is already locally available in memory and report the time to transfer information into GPU memory, construct the supervoxel representation on the GPU, and then write the supervoxel data out to disk. We can see that the GPU transfer time decreases as the number of compute nodes increases since the overall volume can be split into increasingly smaller components. Similarly, the supervoxel generation step also decreases. Lastly, we see that the disk write times for the supervoxel volume increases slightly with more nodes since there is a larger communication overhead when many nodes need to write to a single file. These times are on the order of a few seconds or less and show that the extra computation necessary for this in situ step is small in comparison to the normal simulation computation time (which tends to be much larger than a few seconds).

In addition, we can keep the number of compute nodes constant (256 in this case) and look at how different supervoxel sizes affect the time of each step. This is shown in Graph 13. The GPU transfer time remains relatively constant since the entire volume subset must be sent to the GPU regardless of the end resolution of the supervoxel volume. Like the earlier results, the supervoxel generation time increases slightly for larger supervoxel sizes since a larger number of raw data values must be sampled in the GPU. However, the time to output the new supervoxel volume to file decreases dramatically since a larger supervoxel size results in less data that needs to be saved. Once again these times are on the order of a few seconds or less per timestep and will likely reflect only a small portion of the normal computation time of the simulation.

Continuing with the other steps of the algorithm, which are now performed in real time on a local desktop machine, we can see the timing results shown in Graph 14. In this example, 1000 SLIC clusters were used for each test. Once again we see the same trends for each of the remaining steps as observed in the previous two datasets. However, the refinement step exhibits an entirely different trend because it depends heavily on the underlying structure of this dataset. The overall times are larger than the previous examples since this dataset is much larger. In this case, the optimum

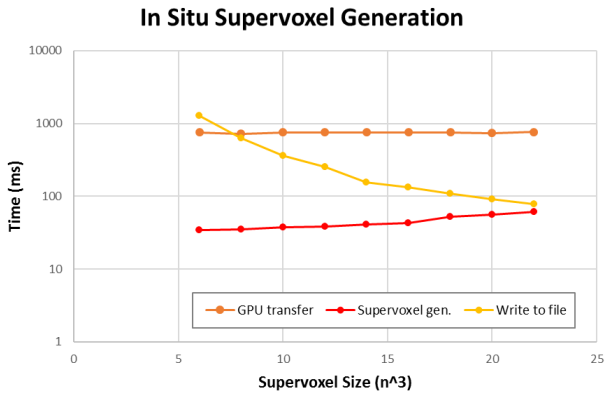


Figure 13: In situ timing results for parallel supervoxel generation vs. supervoxel size.

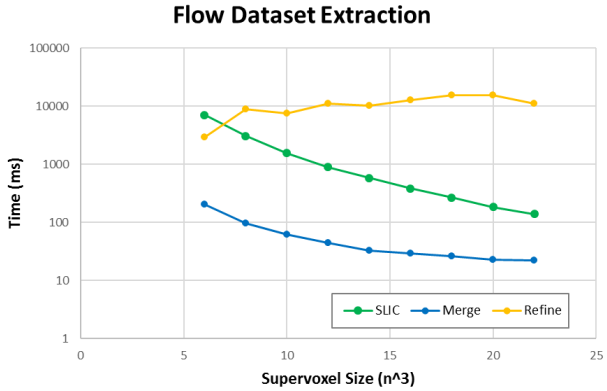


Figure 14: A breakdown of the performance results for each of the extraction steps vs. supervoxel size for the flow dataset.

supervoxel size is 10^3 with an overall extraction time of 9159 ms showing that we can simultaneously extract many coherent features in such a large dataset on a desktop PC with very little overhead.

5 Discussion

The above results demonstrate the ability of this technique to handle datasets of various scales. While certain datasets are easily managed using this scheme, other extreme-scale cases can be handled through some additional in situ processing.

5.1 Data Size Limits on the Desktop PC

Using the datasets shown in this paper as an example, we can see that the combustion ($704 \times 540 \times 550$) and ocean ($3600 \times 2400 \times 42$) datasets are easily manageable on a desktop PC using our extraction scheme. However, the much larger flow dataset ($4096 \times 4096 \times 4096$) incurs a disk I/O time that is too large for real time supervoxel generation. This is alleviated either through a potentially lengthy one time preprocessing step into the supervoxel volume or can be computed directly during the simulation itself. Such a representation can be saved in conjunction with the full resolution version. The technique presented in this paper only fetches the full resolution voxels in regions of the domain with a high uncertainty and thus can extract high resolution features very quickly.

We were able to generate an extraction result for the flow dataset in about 9 seconds, the majority of which consisted of fetching high resolution voxels from disk during the refine step. Moreover, this scheme was able to extract $\sim 10^2$ features simultaneously in such a large volume on a desktop PC. This extraction cost only needs to be

repeated if input parameters, such as the number of clusters, SLIC weights, refinement threshold, etc., need to change.

5.2 User-driven vs. Automatic Extraction

Our feature extraction technique is designed to be run in either a user-driven or automatic manner. In the user-driven mode, input is provided into each step in the algorithm to ensure an extraction result that closely matches the interest of the user. While many of these decisions are based on choosing an appropriate balance between accuracy and performance, others can fundamentally alter which features are extracted. Note that the final uncertainty-driven supervoxel refinement step attempts to minimize the differences observed in the final extraction result based on initial supervoxel sizes.

Choosing an appropriate distance metric for the SLIC cluster generation will have the largest impact on the final result. This is where the user can choose which data variables and spatial properties to emphasize. Knowledge over this matter is often domain specific, making a user-driven approach desirable in many cases. Lastly, the uncertainty-driven refinements step allows users to choose what level of accuracy the final extraction result must meet. In certain cases a fast rough extraction is sufficient, while in other cases a very detailed feature extraction is necessary to study subtle patterns in the data.

In the automatic approach, the algorithm will generate its best estimate of a desirable feature extraction result without any user input. First, the system will automatically choose an appropriate supervoxel resolution based on the scale of the raw dataset, namely one that allows the supervoxel volume to fit entirely into GPU memory for fast access. Next, it will perform the SLIC utilizing a single data variable with equal weights on intensity and spatial proximity. Lastly, the system will choose to refine a default fraction of supervoxels. While the automatic approach uses a very simple set of inputs, users may want to generate a fast general decomposition of the data in order to become better acquainted with its structure before attempting a more detailed exploration.

6 Conclusion

Overall, we develop a new hybrid feature extraction technique which combines the multi-resolution advantages of supervoxels with a GPU accelerated version of SLIC to efficiently manage large-scale datasets on a desktop PC. Using an uncertainty-based refinement method, users can explore extraction results quickly while enhancing detail in certain regions only when necessary. We are able to show the applicability of this technique to a number of large-scale real world datasets and justify its efficiency through performance results.

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..

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2274–2282, 2012.

[2] P. Bremer, E. Bringa, M. Duchaineau, A. Gyulassy, D. Laney, A. Mascarenhas, and V. Pascucci. Topological feature extraction and tracking. In *Journal of Physics: Conference Series*, volume 78, page 012007. IOP Publishing, 2007.

[3] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. *Visualization and Computer Graphics, IEEE Transactions on*, 17(9):1307–1324, 2011.

[4] P.-T. Bremer, G. H. Weber, V. Pascucci, M. Day, and J. B. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):248–260, 2010.

[5] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, 2003.

[6] J. H. Chen, A. Choudhary, B. De Supinski, M. DeVries, E. Hawkes, S. Klasky, W. Liao, K. Ma, J. Mellor-Crummey, N. Podhorszki, et al. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery*, 2(1):015001, 2009.

[7] P. R. Gent, G. Danabasoglu, L. J. Donner, M. M. Holland, E. C. Hunke, S. R. Jayne, D. M. Lawrence, R. B. Neale, P. J. Rasch, M. Vertenstein, P. H. Worley, Z.-L. Yang, , and M. Zhang. The community climate system model version 4. *Journal of Climate*, 24(19):4973–4991, 2011.

[8] R. Huang, E. Lum, and K.-L. Ma. Multi-scale morphological volume segmentation and visualization. In *Visualization, 2007. APVIS'07. 2007 6th International Asia-Pacific Symposium on*, pages 121–128. IEEE, 2007.

[9] R. Huang and K.-L. Ma. Rgvis: Region growing based techniques for volume visualization. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 355–363. IEEE, 2003.

[10] G. Ji, H.-W. Shen, and R. Wenger. Volume tracking using higher dimensional isosurfacing. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 28. IEEE Computer Society, 2003.

[11] S. Kurien and H. Aluie. Joint downscale fluxes of energy and potential enstrophy in rotating stratified boussinesq flows. *American Physical Society*, 55(16), 2010.

[12] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(12):2290–2297, 2009.

[13] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[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. IEEE, 2009.

[15] F. H. Post, B. Vrolijk, H. Hauser, R. S. Laramee, and H. Doleisch. The state of the art in flow visualisation: Feature extraction and tracking. In *Computer Graphics Forum*, volume 22, pages 775–792. Wiley Online Library, 2003.

[16] R. Samtaney, D. Silver, N. Zabusky, and J. Cao. Visualizing features and tracking their evolution. *Computer*, 27(7):20–27, 1994.

[17] F. Sauer, H. Yu, and K.-L. Ma. Trajectory-based flow feature tracking in joint particle/volume datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2565–2574, 2014.

[18] A. Sherbondy, M. Houston, and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Visualization, 2003. VIS 2003. IEEE*, pages 171–176. IEEE, 2003.

[19] A. Szymczak. Subdomain aware contour trees and contour evolution in time-dependent scalar fields. In *Shape Modeling and Applications, 2005 International Conference*, pages 136–144. IEEE, 2005.

[20] M. Van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 212–220. ACM, 1997.

[21] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *Computer Vision–ECCV 2008*, pages 705–718. Springer, 2008.

[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*, pages 17–24. Eurographics Association, 2013.

[23] G. Weber, P.-T. Bremer, M. Day, J. Bell, and V. Pascucci. Feature tracking using reeb graphs. In *Topological Methods in Data Analysis and Visualization*, pages 241–253. Springer, 2011.