Eurographics Symposium on Parallel Graphics and Visualization (2019)
H. Childs, S. Frey (Editors)

Fast Mesh Validation in Combustion Simulations through In-Situ Visualization

Sergei Shudler†[1], Nicola Ferrier[1], Joseph Insley[1,2], Michael E. Papka[1,2], Saumil Patel[1], and Silvio Rizzi[1]
[1]Argonne National Laboratory, USA
[2]Northern Illinois University, USA

Abstract

In-situ visualization and analysis is a powerful concept that aims to give users the ability to process data while it is still resident in memory, thereby vastly reducing the amount of data left for post-hoc analysis. The problem of having too much data for post-hoc analysis is exacerbated in large-scale high-performance computing applications such as Nek5000, a massively-parallel CFD (Computational Fluid Dynamics) code used primarily for thermal hydraulics problems. Specifically, one problem users of Nek5000 often face is validating the mesh, that is identifying the exact location of problematic mesh elements within the whole mesh. Employing the standard post-hoc approach to address this problem is both time consuming and requires vast storage space. In this paper, we demonstrate how in-situ visualization, produced with SENSEI, a generic in-situ platform, helps users quickly validate the mesh. We also provide a bridge between Nek5000 and SENSEI that enables users to use any existing and future analysis routines in SENSEI. The approach is evaluated on a number of realistic datasets.

CCS Concepts
• ***Human-centered computing*** → ***Visualization systems and tools;*** • ***Computing methodologies*** → ***Massively parallel and high-performance simulations;*** • ***General and reference*** → *Performance;*

1. Introduction

As the high-performance computing (HPC) world moves towards extreme-scale systems, the ratio between the amount of data scientific applications produce and the available IO bandwidth grows quickly. On some supercomputers that are already in use now, such as the Titan machine [Tit] in Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (ORNL), the disparity between the computational bandwidth and storage bandwidth is as much as five orders of magnitude [BAA*16]. On Summit [Sum], the next generation machine in OLCF, the disparity is even higher, since the computation ability has increased but no improvements were introduced in the storage system. This means that extreme-scale applications will not be able to write all the data they produce to a persistent storage. Even if applications manage to store as much as possible, the amounts of data make traditional post-hoc analysis time and resource consuming. Post-hoc analysis is a term that refers to the traditional model for analyzing and visualizing data; that is, once the application has finished running, the data is loaded back into the memory for analysis or visualization. This often requires allocating new computational resources on supercomputers, as analysis and rendering routines are computationally heavy and thus can benefit from parallel execution. It is easy to see that this process can quickly become the bottleneck in the users' effort to speed up the rate at which scientific insights are obtained.

In-situ analysis is an alternative approach to post-hoc processing, whereby analysis and visualization occur without first writing the data to persistent storage [YWG*10, KKP*15, BAA*16]. Not only does this eliminate the need to allocate new resources and reload data back into the memory, but it also allows analysis routines to see all the data the application produces. The latter offers the possibility of a meaningful data reduction, instead of just filtering the data according to a timestep number, as scientific applications often do. Another advantage of in-situ techniques is computational steering, which is a process of altering the course of the computation in response to some insight obtained by the user during the computation.

One prominent example of an efficient large-scale code that has been traditionally analyzed in a post-hoc fashion is the Nek5000 code [Nek]. Nek5000 is a high-order spectral element computational fluid dynamics (CFD) solver that has been in use for more than 30 years [OMS*16]. It is used to obtain accurate simulations for a wide range of scientific domains, including fluid flow, thermal convection, combustion, magnetohydrodynamics, and electromagnetics. Over the course of the calculation spectral (mesh) elements undergo a process of deformation. Some deformations are so severe that they causes the element's Jacobian to become negative, a situation sometimes described as a vanishing Jacobian. In such cases,

† E-mail: sshudler@anl.gov

DOI: 10.2312/pgv.20191105

users get a textual output from Nek5000 with indices of the problematic elements and it is a very time consuming process to link these indices back to the mesh elements in the mesh generation application. Seeing where vanishing Jacobians occurred visually can direct users to the regions that need to be fixed more quickly. It saves a significant overhead in the users' work.

In this study, we address the challenge of finding mesh elements with vanishing Jacobians quickly. For this purpose, we use efficient in-situ visualization that highlights the elements with very small and vanishing Jacobians. This allows users to pinpoint and fix the culprit elements faster than they would be able otherwise. Most importantly, however, this approach helps users avoid burning a huge number of core hours in failed big runs. With this, they can verify their meshes on smaller (and cheaper to operate) computing resources. The in-situ visualization is achieved through SENSEI [AWW*16, Sen], a generic in-situ platform that supports a growing number of analysis backends, and the ParaView/Catalyst [PVC] backend. Specifically, our contributions are as follows:

- Novel approach to validate meshes for combustion simulations in situ.
- A study of strong scaling performance of in-situ visualization based on SENSEI and ParaView/Catalyst.
- A bridge between Nek5000 and SENSEI that enables users to use any existing and future analysis backends in SENSEI.

The rest of the paper is organized as follows. We start by reviewing related work in Section 2 and then continue in Section 3 with a brief description of the Nek5000 code and the specifics of identifying vanishing Jacobians. In Section 4 we cover the SENSEI platform before discussing the evaluation and results in Section 5. Finally, we present conclusions and potential future work in Section 6

2. Related Work

Previous work [ABD*16] examined the scalability, overhead, and performance aspects related to instrumenting mini applications (i.e., smaller proxy codes for simulations) with SENSEI. The authors found that SENSEI is a flexible and scalable approach for in-situ analysis and visualization. In a later work [URW*18], SENSEI was used to instrument LAMMPS, a large-scale molecular dynamics code, and visualize the molecules in transit. The difference between in-transit and in-situ analysis is that the former uses separate computing nodes to run the analysis and visualization routines, thereby allowing overlapping simulation and analysis steps. In our case, there is no benefit in performing the visualization in transit since the fluid motion calculations are disabled so the visualization part gets to use all the available computation resources.

Ascent is another in-situ analysis framework [LAA*17] analogous to SENSEI. However, the data model in Ascent is based on Conduit [Con] and not on the VTK data model used in SENSEI. Conduit is a model to describe scientific data to make data exchange in the form of serialization or I/O easier. On the one hand, it offers more flexibility in terms of analysis algorithms available for users. On the other hand, however, it makes integration with ParaView/Catalyst harder. The ability to use the power of ParaView through Catalyst is a key element of our approach. Furthermore, since SENSEI is a generic platform, users can easily switch to using VisIt if it offers better capabilities for some visualization tasks.

Whitlock et al. [WFM11] developed an analogous library to ParaView/Catalyst called VisIt/Libsim [WFM11]. This library allows users to render the simulation data in situ using the VisIt visualization tool. Similar to ParaView/Catalyst, SENSEI provides an analysis backend for VisIt/Libsim. One advantage of using ParaView is that it offers an easy way to produce Cinema databases [AJO*14], a capability that might be very useful for inspecting problematic mesh elements within a larger mesh. At the time of our study, VisIt did not provide support for Cinema.

Bauer et al. [BAA*16] first explain the motivation for in-situ analysis, and then survey methods, infrastructures, and a range of applications that use in-situ techniques for analysis and visualization. One of the presented applications uses ParaView/Catalyst for data analysis and another application combines Cinema with ParaView/Catalyst. Our approach is based on these earlier experiences and is applied to identifying vanishing Jacobians. In earlier work, Yu et al. [YWG*10] and Ribes et al. [RLJF15] demonstrate that in-situ visualization of combustion and CFD simulations, respectively, is useful and can be performed at scale.

3. Nek5000

Nek5000 is a massively parallel, open-source CFD solver based on the spectral element method (SEM) which combines the geometric flexibility of finite element methods (FEM) with the rapid convergence and tensor-product efficiencies of global spectral methods. It uses Eulerian operations that are statically partitioned with the aim of optimized load balance and communication overhead prior to the start of every production run. It is written in Fortran 77 and C with support for post-hoc data analysis and visualization using either VisIt [Vis] or ParaView [Par].

3.1. Combustion simulation

Several studies [SFT*14, SFW*16, FST17, GFB*17] have shown Nek5000 to be a promising tool to investigate the complex thermal-hydraulic phenomena within an internal combustion engine. For this kind of simulations, Nek5000 solves the low-Mach compressible Navier-Stokes equations in the arbitrary Lagrangian-Eulerian framework. In this case, a Lagrangian description of the boundary is adopted where the mesh is updated at every timestep to conform to the new boundary location while the flow-field is solved using an Eulerian approach. As the boundary moves, each spectral element in the domain is deformed due to the inherent body-conforming property of the SEM. Deformations can lead to dramatic changes in the shape of the elements causing high-aspect ratios and potentially vanishing Jacobians. This tends to occur in the vicinity of a complex boundary like the engine valves or the runner-valve-stem connection.

Figure 1 presents an example mesh generated in CUBIT [Cub], a toolkit for generating two- and three-dimensional finite element meshes. This mesh is one of the datasets we evaluate in Section 5. To generate the mesh, the volume of an engine is partitioned in non-overlapping sub-volumes that are simpler to mesh. This means that

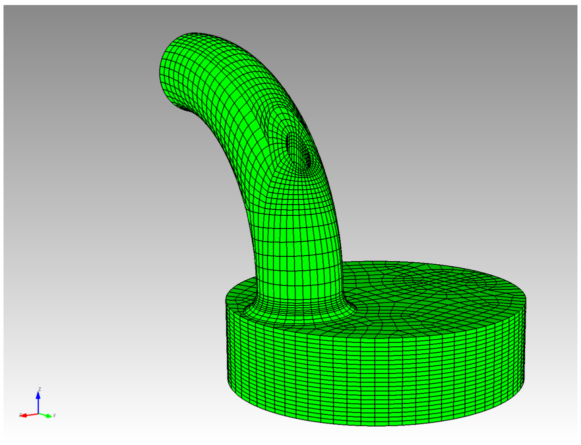


Figure 1: *Example input dataset [SRYK14] in CUBIT.*

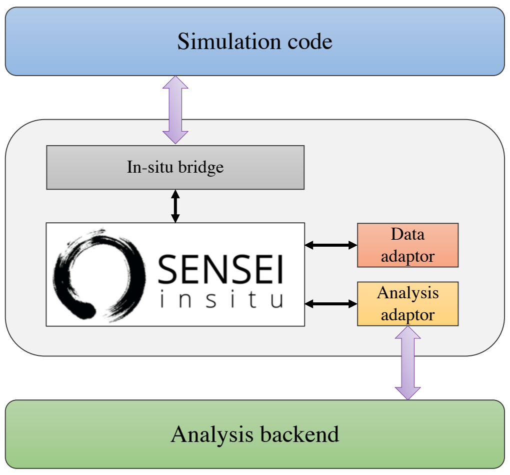


Figure 2: *Overview of the SENSEI platform.*

we can create a mesh on a particular surface (i.e., in 2D) of the volume and sweep or extend the mesh throughout the sub-volume. In most cases, the order in which the volumes are meshed is crucial for creating a continuous and smooth final mesh with relatively homogeneous sizes for the hexahedral elements. When the order is not optimal, the 2D surface mesh is not ideal, or the sub-volumes themselves are too skewed, hexahedral elements may become highly skewed or distorted. In this case, the elements may still be valid (i.e., non-vanishing Jacobians) at the start of a production run, but at the same time, may be prone to vanishing Jacobians much sooner during the mesh motion. As a result, mesh generation requires careful treatment and users invest a significant amount of time and resources to make sure that the mesh is "suitable," that is has a low probability for the occurrence of vanishing Jacobians.

3.2. Identifying vanishing Jacobians

The goal of our workflow is to determine when and where vanishing Jacobians, if any, occur. This allows users to validate the mesh before running any real, production-level simulations. For motored engine operations, the motion of the piston and valve are known a-priori. This means that the hydrodynamic calculation can be disabled leaving only mesh motion active, thereby allowing a faster computation. Once users have identified any potential vanishing Jacobians and updated the mesh accordingly, the fluid motion calculation can be switched back on. Faster computation leads to faster turnaround time for the workflow, which means it can be repeated multiple times. This ensures that after initial vanishing Jacobians are eliminated no new ones occur later in the simulation. Once vanishing Jacobians are identified the process of fixing or regenerating the mesh usually involves re-creation of sub-volumes or refinement of the mesh in a sub-volume to improve skewness and aspect ratio. Usually the re-creation of the element distribution takes place in the region where vanishing Jacobians are found.

Visualizing vanishing Jacobians allows users to locate the problematic regions of the mesh quicker. Without visualization, users see just a textual output from Nek5000 informing them that vanishing Jacobians occurred and what are the indices of the problematic elements. These indices are not directly linked to the IDs of the elements in CUBIT, and users have no easy way to translate these indices to IDs.

4. The SENSEI Platform

In this section, we describe the SENSEI platform and its ParaView/Catalyst [PVC] backend that we use to render the mesh in situ. We also cover the approach to instrument the Fortran-based Nek5000 code with SENSEI, which is a C++ code.

4.1. Overview

SENSEI (or Scalable Environment for Scientific Explorations In Situ) is a platform that aims to improve code portability and reusability by decoupling simulations from the analysis routines. The design philosophy of SENSEI is *write once, use everywhere* [AWW*16]. This means that from the simulation perspective the code can be instrumented once and then use any of the already available or future analysis backends offered by SENSEI.

Figure 2 presents a schematic overview of how SENSEI links simulation code to analysis backends. To support a common instrumentation interface, SENSEI is built with a common data model in mind. This data model is the VTK data model, which is widely used in applications such as ParaView and VisIt. The figure identifies three components of the SENSEI interface: In-situ bridge, Data adaptor, and Analysis adaptor. First, the purpose of the bridge is to expose an API that simulations can use as entry points into SENSEI. Essentially, these entry points are used to pass data to SENSEI and trigger the in-situ analysis. Second, the data adaptor role is to map simulation data to the VTK data model. Lastly, the analysis adaptor serves as a driver for the analysis backends.

When users want to add support for a new simulation, they need to write a new data adaptor and a new in-situ bridge module. Depending on how the data is represented in the simulation, the amount of effort involved is usually small. In some cases the data conversion requires no memory copying at all, a situation known as *zero-copy*. Most importantly, however, the coding of the data adaptor and the in-situ bridge module is a one-time effort. Once these components exist they will, in most cases, support without any changes new analysis backends added to SENSEI. Existing analysis backends include in-situ infrastructures, such as VisIt/Libsim [WFM11] and ParaView/Catalyst, the ADIOS IO framework [LLT*14], and Python. To support a new backend, one has to implement a new analysis adaptor that takes the data from the data adaptor and performs any necessary transformation to allow the backend to use this data.

4.2. Nek5000-SENSEI bridge

The interaction between Nek5000 and SENSEI is implemented in a data adaptor and a bridge. The bridge implements three functions that correspond to three phases of the simulation. Listing 1 shows how these functions fit into the Nek5000 structure. The first function, namely *sensei_bridge_inititalize*, is called before the main iteration loop. The goal is to allow SENSEI to initialize any analysis backends that the user specified for this specific run. It is also used to create the data adaptor and pass it along to analysis adaptors. In this case, the data adaptor has to convert the Nek5000 mesh to the VTK unstructured grid. One of the important steps in the conversion involves generating mesh connectivity information, which is explicit in VTK data model but implicit in Nek5000. The function *sensei_bridge_update* is called at the end of each iteration in Nek5000 to allow SENSEI to execute the analysis backends for this specific iteration. The last function, which is *sensei_bridge_finalize*, is called after the main iteration loop ends. At this point, SENSEI can finalize all the analysis backends, which means the backends can perform last analysis steps, such as render final images or close any open files. Note that *sensei_bridge_update* receives only one argument, namely the timestep number. SENSEI already received the pointers to the mesh data and Jacobians array in *sensei_bridge_inititalize*, and since no data is copied into new memory, any updates of these arrays in Nek5000 will be automatically reflected in SENSEI.

Listing 1 Pseudocode for Nek5000 instrumentation

```
init_simulation()
sensei_bridge_inititalize( mesh, jacobian_arr )
timestep = first
while timestep < last do
    simulation_step()
    sensei_bridge_update( timestep )
    timestep++
end while
sensei_bridge_finalize()
finalize_simulation()
```

Since Nek5000 is a Fortran code and SENSEI is written in C++, the bridge code is compiled as a static library and the interface—with only three functions—is exported as plain C code. This means that the Fortran compiler can link Nek5000 with this library, while the library itself is compiled with a C++ compiler and linked with SENSEI. The result is that all of SENSEI functionality is easily accessible. This approach can serve as a guideline for other codes that want to use SENSEI but are written in Fortran or other languages different from C++.

4.3. ParaView/Catalyst backend

ParaView is a data analysis and visualization application used extensively for scientific visualization. As with many traditional visualization tools, ParaView is usually used post-hoc, that is after the simulation finished running. In a typical workflow, users will import the data (possibly in different formats) into ParaView, perform various filtering or analysis, and then produce the necessary visualization. To speed up rendering, ParaView supports parallel rendering and efficient compositing through the IceT library [MKPH11]. Recognizing the importance of in-situ processing, ParaView includes an infrastructure called Catalyst for in-situ analysis and visualization. It offers both C++ and Python interfaces, and allows codes to use ParaView capabilities (both data analysis and rendering) programmatically from within the code itself and while it still executes [PVC].

Catalyst is one of the analysis backends supported by SENSEI, and there is a clear advantage in using it through SENSEI. First, the Catalyst analysis adaptor includes boilerplate code that makes it easier to use Catalyst. Second, users can easily switch to a different analysis backend without wasting time rewriting the boilerplate code or learning the intricacies of a new interface. To use Catalyst from SENSEI, users provide a Python script file that instructs the ParaView engine how to produce the desired visualization. This script file can be easily exported from a ParaView session. For this to work, however, users have to have the data they need to visualize at hand. They can then load it in a ParaView client and perform any needed analysis and/or visualization steps. Once the desired visualization is achieved, ParaView can produce a Python script that recreates this exact visualization. When running in situ, the data will be passed to Catalyst directly in memory and will not be read from the disk.

To help users find vanishing Jacobians faster, we have to highlight those areas of the mesh that have the smallest Jacobians. For this purpose, we use two *Clip* filters in ParaView. The first one filters out all the mesh elements with Jacobians above a certain threshold, and colors the remaining ones according to the value of the Jacobian. The second one filters out all the mesh elements with positive Jacobians such that all the remaining elements have vanishing Jacobians. To better see these elements within the surrounding mesh, we render them as bright green points. The original mesh is left as a semi-transparent background to provide context for the highlighted elements. Figure 3c, in Section 5, shows an example of the visualization that results from this pipeline.

This workflow does not require full mesh for all the iterations of the simulation. It is enough to have the full mesh for just one timestep, which requires minimal overhead in terms of computing resources and storage. For the purpose of writing the mesh data, SENSEI provides an analysis adaptor called *PosthocIO* that writes the data from the data adaptor directly to the disk. This data is already in VTK format—in case of Nek5000, these are *.vtu and *.pvd files—and can be readily loaded into ParaView.

Table 1: *The datasets for the evaluation. Duration specifies the number of time steps.*

Dataset	Type	No. elements	Duration [ts]
A	3D	6,784	4
B	2D	1,060	10,000
C	3D	19,216	1,700

5. Evaluation

In this section, we evaluate our approach based on in-situ visualization that allows users to quickly validate meshes for combustion simulations. We start with a description of the evaluation environment, including the different datasets we used, and then continue with a detailed discussion of results and performance aspects.

5.1. Experimental setup

We ran the experiments on the Cooley cluster [Coo] at Argonne Leadership Computing Facility (ALCF), a division within Argonne National Laboratory. Cooley comprises 126 nodes with two Intel Haswell CPUs (12 cores in total) and an NVIDIA Tesla K80 GPU for each node. All the nodes are connected via Infiniband. The purpose of Cooley is to serve as a cluster for analyzing and visualizing data obtained from running simulations on other machines at ALCF. In other words, Cooley shares the file system with other machines, so that when simulations finish running and writing data to storage, analysis routines on Cooley could access the data. A separate machine for analysis and visualization fits into the traditional paradigm of post-hoc processing. In our case, however, using Cooley shows that the approach does not require bigger and more expensive, in terms of operational costs, resources.

For the evaluation, we chose three datasets that focus on simulating combustion in an automobile engine. Table 1 presents the parameters of each dataset in terms of mesh type, the number of elements, and the simulation duration. The three datasets represent three possible situations with respect to vanishing Jacobians. For the first dataset, we already knew in advance that vanishing Jacobians will occur at timestep 4 and the simulation will not be able to continue. For the second one, we knew in advance that vanishing Jacobians will not occur, so the simulation was able to run for the whole duration of 10,000 timesteps. Finally, for the third dataset, we did not know in advance whether vanishing Jacobians will occur or not, a situation most close to reality. In the course of the evaluation, we discovered that vanishing Jacobians in this case do occur at timestep 1,700, and this is the duration for dataset C.

5.2. Results analysis

In this subsection, we present the results of our experiments on datasets A, B, and C. Furthermore, we describe how to combine the Cinema framework in our workflow such that users get images from different viewpoints. Finally, we discuss the performance aspects of our approach.

5.2.1. Dataset A

This dataset has a short duration and a fairly small number of elements. We ran it, therefore, on just one node with 12 MPI ranks. Figure 3 depicts the mesh of dataset A in three different situations. Figure 3a shows the mesh without any filters that highlight very small Jacobians. The color range is based on a log scale and represents the range of all Jacobians. This particular image depicts the mesh at the first timestep and it was created in a separate run to show how difficult it is to spot mesh elements with the smallest Jacobians. Figure 3b shows the image, rendered in situ, of a filtered mesh after the first timestep. All the Jacobians greater than 1×10^{-7} are clipped and the remaining ones are located on the top and bottom rims of the mesh. The original mesh is rendered as well, but semi-transparently, so that it provides context without obstructing the areas with potential vanishing Jacobians. Figure 3c shows the image of a filtered mesh rendered after the last timestep, namely the timestep at which vanishing Jacobians occur. In this image, green points depict the mesh elements with these Jacobians. This information provides time-saving guidance in the effort to fix the problematic elements in the mesh generation tool.

5.2.2. Dataset B

Being a 2D dataset rather than a 3D one makes mesh deformation much less severe. Although the chance of vanishing Jacobians occurring in this case is lower, it is still above zero. Figure 4 depicts the mesh of dataset B in three different situations. Figure 4a is analogous to the unfiltered mesh image in the previous dataset. Figure 4b shows the image (of a filtered mesh) rendered in situ after half of the timesteps. At this point, the valve has moved down, which results in less deformation of the mesh elements and, consequently, higher Jacobians. Figure 4c shows the image of a filtered mesh rendered in situ after the last timestep (i.e., timestep 10,000). The valve moved back up and closed the opening, which results in more deformation and lower Jacobians. However, no Jacobians became negative and, as a result, there are no green points in the image.

5.2.3. Dataset C

We ran this dataset on 60 MPI ranks such that there were approximately 320 mesh elements per rank. Figure 5 depicts the mesh of this dataset in four different situations. Figure 5a is analogous to the unfiltered image in the previous datasets. Figure 5b shows the image of a filtered mesh rendered after the first timestep. There is an area of small Jacobians on the pipe leading to the chamber, but since this part is not moving in the simulation, the mesh elements—and the respective Jacobians—do not change. Figure 5c shows the image of a filtered mesh after 85% of timesteps. In this image, the valve is on its way up to close the opening into the chamber. Compared to the first timestep, there are more elements with small Jacobians on the valve's boundary. Figure 5d shows the image rendered in situ after the last timestep, namely the timestep at which vanishing Jacobians occur and the simulation stops. Compared to dataset A, there are much fewer elements with vanishing Jacobians and without rendering the image in situ, it would have been very challenging for users to find these elements and fix the mesh.

Figure 5d also demonstrates that the bright green dots that mark

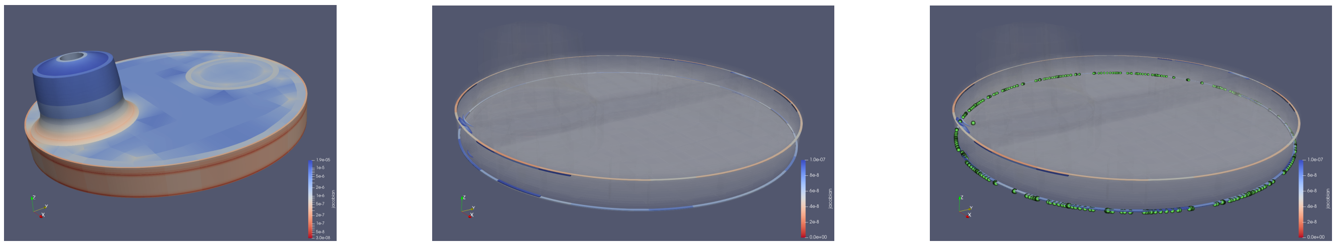

(a) Unfiltered mesh (b) Filtered mesh (first timestep) (c) Filtered mesh (last timestep)

Figure 3: *In-situ visualization of dataset A. Images with a filtered mesh highlight very small and vanishing Jacobians. All the images are RGB with 1400 × 800 pixels and 8-bits per channel.*

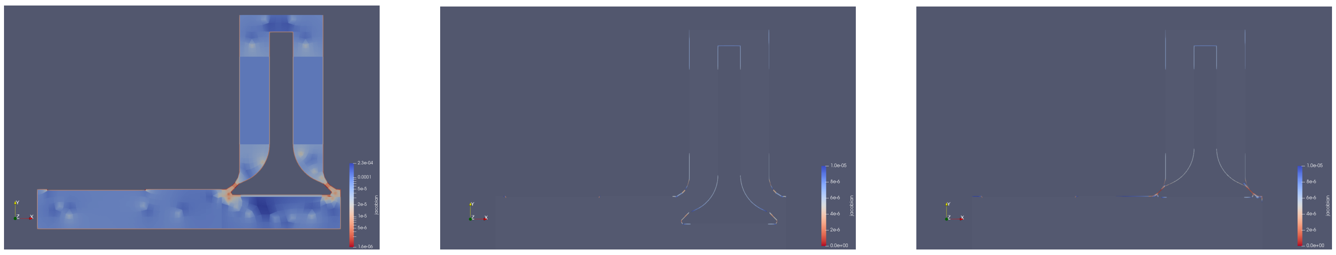

(a) Unfiltered mesh (b) Filtered mesh (50% of time steps) (c) Filtered mesh (last timestep)

Figure 4: *In-situ visualization of dataset B. Images with a filtered mesh highlight very small and vanishing Jacobians. All the images are RGB with 1400 × 800 pixels and 8-bits per channel.*

vanishing Jacobians can be hard to spot in some cases. In particular, when the dataset is bigger and the image resolution is higher. The solution in such cases is to increase the size of the dots. Although this will make it harder to pin-point the exact element with a vanishing Jacobian, identifying a small enough vicinity around such an element is already good enough. This is because the process of fixing the mesh involves re-creating a whole sub-volume around the problematic element.

5.2.4. Cinema databases

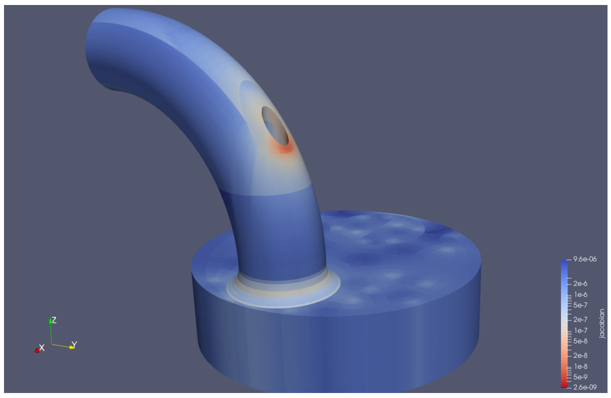
Cinema [AJO*14] is an approach to specify a database of parameters mapped to a set of data artifacts. Most commonly, the data artifacts are images. Essentially, Cinema allows simulations to reduce the size of the data they write to storage, such that post-hoc analysis is less expensive in terms of time and effort. One common use case is to combine Cinema with in-situ visualization and produce images rendered at different camera angles. Applied to our case, such ability gives users the chance to see highlighted mesh elements (i.e., elements with vanishing Jacobians) from different viewpoints. This can help users find the problematic elements within the full mesh faster. One has to remember, however, that rendering multiple images at each timestep increases both the analysis time and stored data size.

The ParaView/Catalyst module has an option to write a Cinema database. When exporting the Catalyst script from ParaView, users can specify the number of ϕ angles and θ angles for the camera relative to the mesh. Upon execution of the script at each timestep, Catalyst moves the camera to each combination of ϕ and θ angles and renders an image. Once the execution is finished, the Cinema database can be examined with one of the available Cinema viewers [Cin].

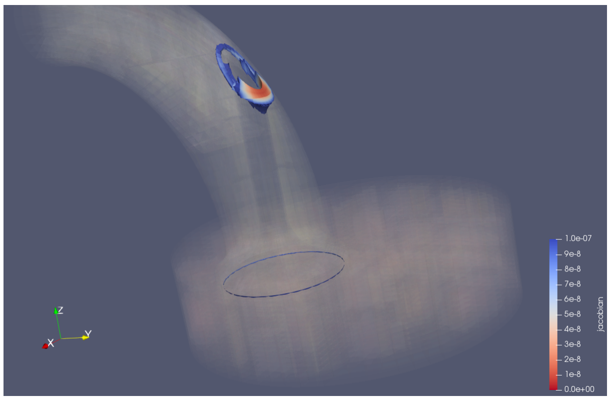
Table 2: *Sizes of data written to persistent storage for different techniques.*

Dataset	Checkpoints	In-situ (Cinema)	In-situ (Single image)
A	1.4 GB	95 MB	1.2 MB
B	15.2 GB	—	248 MB
C	500 GB	35.4 GB	708 MB

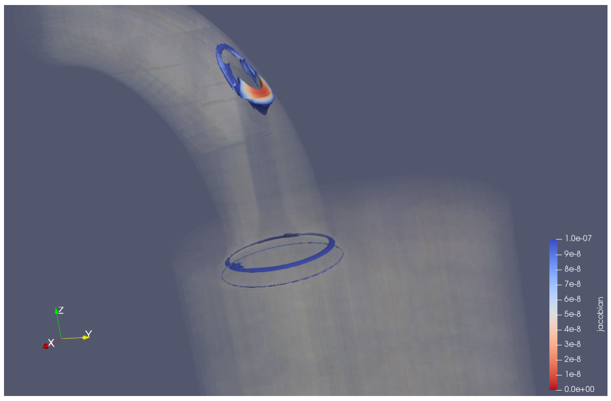
Figure 6 presents three different images of dataset A (at the last timestep) rendered with different camera positions around the mesh. These images are part of a Cinema database produced in situ. In this particular example, there are 12 different ϕ angles and 5 different θ angles, which means that Catalyst renders 60 images for each timestep. This has implications in terms of additional storage space. In the next subsection, we discuss these implications in more detail. Nevertheless, these images show that Cinema provides users with an additional insight and the relative ease of use is another advantage for the combination of SENSEI with ParaView/Catalyst. Note that we ran experiments with Cinema only for datasets A and C, since dataset B is two-dimensional and rotating the camera around it has no added value.



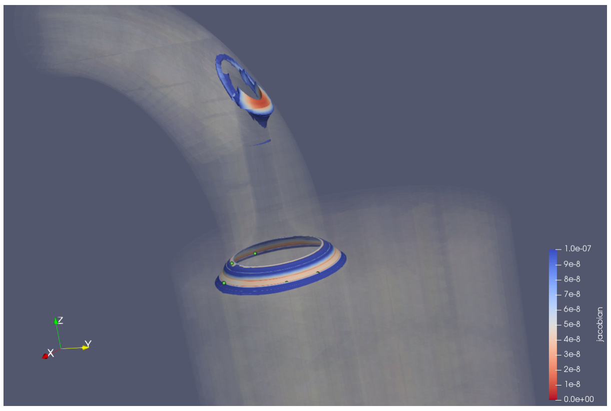
(a) Unfiltered mesh



(b) Filtered mesh (first timestep)

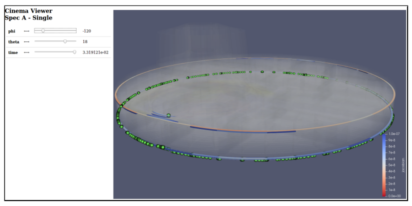


(c) Filtered mesh (85% of time steps)



(d) Filtered mesh (last timestep)

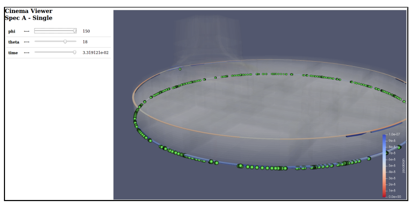
Figure 5: *In-situ visualization of dataset C. Images with a filtered mesh highlight very small and vanishing Jacobians. All the images are RGB with 1200 × 900 pixels and 8-bits per channel.*



Cinema Viewer
Spec A - Single

phi -120
theta 18
time 3.319121e-02

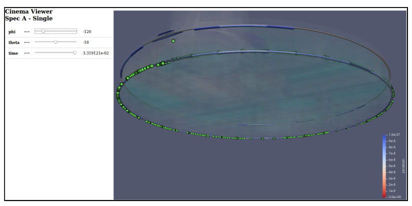
(a) Camera: $\phi = -120°$, $\theta = 18°$



Cinema Viewer
Spec A - Single

phi -120
theta -18
time 3.319121e-02

(b) Camera: $\phi = -120°$, $\theta = -18°$



Cinema Viewer
Spec A - Single

phi 150
theta 18
time 3.319121e-02

(c) Camera: $\phi = 150°$, $\theta = 18°$

Figure 6: *Different viewpoints of dataset A at the last timestep. The images are part of a Cinema database produced in situ.*

5.2.5. Storage space

Table 2 presents the total sizes of the data written to persistent storage. Note that these sizes do not depend on the number of MPI ranks. The Checkpoints column shows the size of the data assuming the simulation was performing a checkpoint for each iteration. The Cinema column shows the size of the Cinema database, and the Single image column shows the size of all the rendered images assuming one image per iteration. Although checkpointing is rarely performed each iteration, it is required to achieve comparable fidelity to the in-situ cases. Since we do not know in advance in which iteration, if any, vanishing Jacobians occur, we have to either checkpoint or render an image (or images) each iteration. The table shows that the single image in-situ approach requires approximately two to three orders of magnitude less space, which is a clear advantage compared to checkpointing. Cinema falls somewhere in between full checkpointing and single image. Users obtain images from different viewpoints, but at the cost of increased storage and longer visualization times. Whether these images help users and justify the increased costs depends on the complexity of the dataset.

It is important to note that although dataset B has more timesptes than dataset C, the sizes of the images produced in situ for one timestep are approximately 25 KB and 426 KB, respectively. This is explained by the increased size and complexity of dataset C. The total size of the data, therefore, for dataset C is bigger.

5.2.6. Performance

Datasets A and B are relatively small either in terms of timesteps or the number of mesh elements. Therefore, we ran the experiments involving these datasets on just a single node with 12 MPI processes. Dataset C, however, is much larger and was a good candidate for investigating the scaling aspects of our approach. Figures 7 and 8 show the results of running the experiments with dataset C on increasing number of processes. In these runs we used 12 MPI processes per node, which means the first run in the figure was with 5 nodes and the last one with 35 nodes. We instrumented the three calls in the SENSEI bridge and also measured the wallclock time of the full execution. In Figure 7, the measured times for the analysis step in the bridge (i.e., *sensei_bridge_update*) were reduced by first calculating the maximum time for each timestep across all the processes, and then calculating the median across all the timesteps. Figure 8 shows the wallclock time for the full execution in seconds. There is a clear correlation between the two plots, which is hardly surprising since we switched off the fluid motion calculation. In other words, scaling in this case depends on the in-situ visualization efficiency. Although the performance of SENSEI is comparable to previous findings [ABD*16] and the analysis time (monotonically) decreases with the number of processes it is clear that there is a scalability obstacle. A number of factors can contribute to this phenomenon. One is the compositing part of the visualization, the cost of which directly depends on the number of processes. Another factor relates to the overhead of running the ParaView/Catalyst module driven by a Python script. Finally, there is a latency overhead of writing the rendered image to the storage. Investigating these factors in depth, however, is beyond the scope of this work. Nevertheless, users we worked with confirmed that the wallclock time of the execution is short enough to provide tangible benefits. In other words, locating the problematic mesh elements following a failed run would have taken much more than one hour—the time it takes to run dataset C with 180 MPI processes (15 nodes).

6. Conclusion and Future Work

In this work, we propose an innovative approach to validate meshes for combustion simulations. Validating that no vanishing Jacobians occur in the mesh saves computation time that might be otherwise lost if a vanishing Jacobian occurs in the middle of a simulation run. If vanishing Jacobians do occur, users obtain an image of the problematic elements highlighted within the full mesh. This kind of visual data helps users fix the mesh and rerun the validation workflow. Since the validation does not require expensive fluid motion calculations, the workflow can be executed multiple times until no more vanishing Jacobians occur. Furthermore, our evaluation confirms that the approach does not require expensive computational

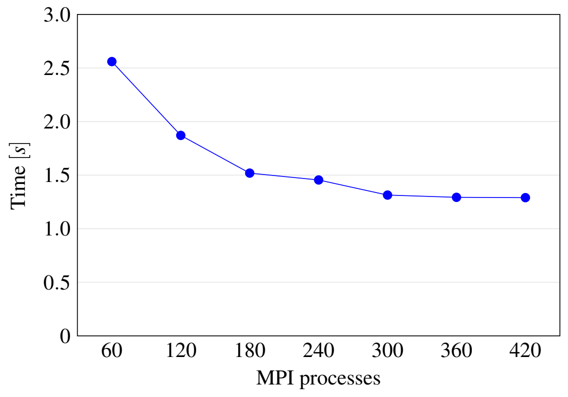


Figure 7: *Analysis step times of dataset C. Each value is the maximum across all processes and a median across all timesteps.*

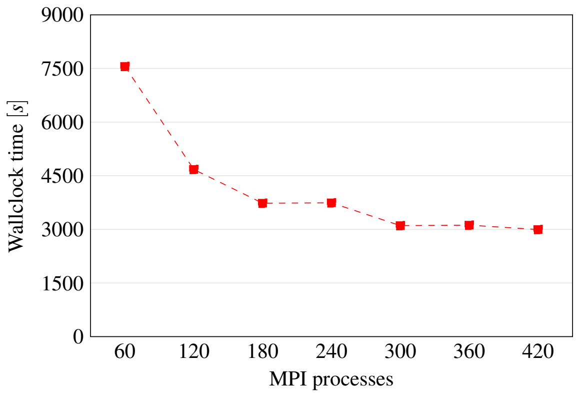


Figure 8: *Wallclock execution times (in seconds) of dataset C.*

resources and the workflow can be executed on a relatively small cluster.

At the heart of our approach is the SENSEI platform and the ParaView/Catalyst module. Producing images that highlight very small and vanishing Jacobians in situ offers advantages in terms of execution time and storage space. It allows us to reach the necessary fidelity level such that the results become useful to the users. Furthermore, by using Cinema databases users can easily obtain even more fidelity.

We performed a strong scaling study that showed us how the approach scales to a higher number of processes. Although execution times monotonically decrease the study revealed a scalability bottleneck. In the future, we intend to experiment with much larger meshes and larger supercomputers. As part of this experimentation, we also plan to perform an in-depth study of potential scalability obstacles in the analysis and visualization part of SENSEI. To make the approach even more helpful to users, we want to investigate better ways to associate between mesh elements rendered in situ to the corresponding elements in the mesh generation tool (e.g., CUBIT [Cub]). One can consider even automating the validation and repair steps, such that the mesh is repaired in situ.

Acknowledgments

This work was supported by and used resources of the Argonne Leadership Computing Facility, which is a U.S. Department of Energy, Office of Science User Facility supported under Contract DE-AC02-06CH11357. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357, through the project SENSEI: Scalable Analysis Methods and In Situ Infrastructure for Extreme Scale Knowledge Discovery.

References

[ABD*16] AYACHIT U., BAUER A., DUQUE E. P. N., EISENHAUER G., FERRIER N., GU J., JANSEN K. E., LORING B., LUKIĆ Z., MENON S., MOROZOV D., O'LEARY P., RANJAN R., RASQUIN M., STONE C. P., VISHWANATH V., WEBER G. H., WHITLOCK B., WOLF M., WU K. J., BETHEL E. W.: Performance Analysis, Design Considerations, and Applications of Extreme-scale in Situ Infrastructures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2016), SC '16, IEEE Press, pp. 79:1–79:12. 2, 8

[AJO*14] AHRENS J., JOURDAIN S., O'LEARY P., PATCHETT J., ROGERS D. H., PETERSEN M.: An Image-based Approach to Extreme Scale in Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Piscataway, NJ, USA, 2014), SC '14, IEEE Press, pp. 424–434. doi:10.1109/SC.2014.40. 2, 6

[AWW*16] AYACHIT U., WHITLOCK B., WOLF M., LORING B., GEVECI B., LONIE D., BETHEL E. W.: The SENSEI Generic in Situ Interface. In *Proceedings of the 2nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization* (Piscataway, NJ, USA, Nov 2016), ISAV'16, IEEE Press, pp. 40–44. doi:10.1109/ISAV.2016.13. 2, 3

[BAA*16] BAUER A. C., ABBASI H., AHRENS J., CHILDS H., GEVECI B., KLASKY S., MORELAND K., O'LEARY P., VISHWANATH V., WHITLOCK B., BETHEL E. W.: In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Computer Graphics Forum 35*, 3 (2016), 577–597. doi:10.1111/cgf.12930. 1, 2

[Cin] Cinema – A Database Approach to Extreme Scale Visualization and Analysis. https://cinemascience.github.io. Accessed: 2019-03-04. 6

[Con] Conduit – Simplified Data Exchange for HPC Simulations. https://software.llnl.gov/conduit. Accessed: 2019-03-4. 2

[Coo] Cooley – Argonne Leadership Computing Facility. https://www.alcf.anl.gov/user-guides/cooley. Accessed: 2019-02-25. 5

[Cub] CUBIT – Geometry and Mesh Generation Toolkit. https://cubit.sandia.gov. Accessed: 2019-03-04. 2, 8

[FST17] FISCHER P., SCHMITT M., TOMBOULIDES A.: Recent developments in spectral element simulations of moving-domain problems. In *Recent Progress and Modern Challenges in Applied Mathematics, Modeling and Computational Science*. Springer, 2017, pp. 213–244. 2

[GFB*17] GIANNAKOPOULOS G., FROUZAKIS C. E., BOULOUCHOS K., FISCHER P. F., TOMBOULIDES A.: Direct numerical simulation of the flow in the intake pipe of an internal combustion engine. *International Journal of Heat and Fluid Flow 68* (2017), 257–268. 2

[KKP*15] KRESS J., KLASKY S., PODHORSZKI N., CHOI J., CHILDS H., PUGMIRE D.: Loosely Coupled In Situ Visualization: A Perspective on Why It's Here to Stay. In *Proceedings of the 1st Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2015), ISAV'15, ACM, pp. 1–6. doi:10.1145/2828612.2828623. 1

[LAA*17] LARSEN M., AHRENS J., AYACHIT U., BRUGGER E., CHILDS H., GEVECI B., HARRISON C.: The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the 3rd Workshop In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2017), ISAV'17, ACM, pp. 42–46. doi:10.1145/3144769.3144778. 2

[LLT*14] LIU Q., LOGAN J., TIAN Y., ABBASI H., PODHORSZKI N., CHOI J. Y., KLASKY S., TCHOUA R., LOFSTEAD J., OLDFIELD R., PARASHAR M., SAMATOVA N., SCHWAN K., SHOSHANI A., WOLF M., WU K., YU W.: Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience 26*, 7 (2014), 1453–1473. doi:10.1002/cpe.3125. 4

[MKPH11] MORELAND K., KENDALL W., PETERKA T., HUANG J.: An Image Compositing Solution at Scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 25:1–25:10. doi:10.1145/2063384.2063417. 4

[Nek] NEK5000 Version 17.0. https://nek5000.mcs.anl.gov. Argonne National Laboratory, Illinois. Release date 2017-12-18. Accessed: 2019-02-18. 1

[OMS*16] OFFERMANS N., MARIN O., SCHANEN M., GONG J., FISCHER P., SCHLATTER P., OBABKO A., PEPLINSKI A., HUTCHINSON M., MERZARI E.: On the Strong Scaling of the Spectral Element Solver Nek5000 on Petascale Systems. In *Proceedings of the Exascale Applications and Software Conference 2016* (New York, NY, USA, 2016), EASC '16, ACM, pp. 5:1–5:10. doi:10.1145/2938615.2938617. 1

[Par] ParaView. https://www.paraview.org. Accessed: 2019-02-20. 2

[PVC] ParaView/Catalyst. https://www.paraview.org/in-situ. Accessed: 2019-02-20. 2, 3, 4

[RLJF15] RIBÉS A., LORENDEAU B., JOMIER J., FOURNIER Y.: In-Situ Visualization in Computational Fluid Dynamics Using Open-Source tools: Integration of Catalyst into Code_Saturne. In *Topological and Statistical Methods for Complex Data* (Berlin, Heidelberg, 2015), Bennett J., Vivodtzev F., Pascucci V., (Eds.), Springer Berlin Heidelberg, pp. 21–37. 2

[Sen] SENSEI – Scalable in situ analysis and visualization. https://sensei-insitu.org. Accessed: 2019-02-18. 2

[SFT*14] SCHMITT M., FROUZAKIS C. E., TOMBOULIDES A. G., WRIGHT Y. M., BOULOUCHOS K.: Direct numerical simulation of multiple cycles in a valve/piston assembly. *Physics of Fluids 26*, 3 (2014), 035105. 2

[SFW*16] SCHMITT M., FROUZAKIS C. E., WRIGHT Y. M., TOMBOULIDES A. G., BOULOUCHOS K.: Investigation of wall heat transfer and thermal stratification under engine-relevant conditions using dns. *International Journal of Engine Research 17*, 1 (2016), 63–75. 2

[SRYK14] SICK V., REUSS D., YANG X., KUO T.-W.: Tcc-iii cfd input dataset. 3

[Sum] Summit – Oak Ridge Leadership Computing Facility. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit. Accessed: 2019-02-19. 1

[Tit] Titan – Oak Ridge Leadership Computing Facility. https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan. Accessed: 2019-02-19. 1

[URW*18] USHER W., RIZZI S., WALD I., AMSTUTZ J., INSLEY J., VISHWANATH V., FERRIER N., PAPKA M. E., PASCUCCI V.: libIS: A Lightweight Library for Flexible in Transit Visualization. In *Proceedings of the 4th Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization* (New York, NY, USA, 2018), ISAV'18, ACM, pp. 33–38. doi:10.1145/3281464.3281466. 2

[Vis] VisIt. https://visit.llnl.gov. Accessed: 2019-02-20. 2

[WFM11] Whitlock B., Favre J. M., Meredith J. S.: Parallel in Situ Coupling of Simulation with a Fully Featured Visualization System. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2011), EGPGV '11, Eurographics Association, pp. 101–109. doi:10.2312/EGPGV/EGPGV11/101-109. 2, 4

[YWG*10] Yu H., Wang C., Grout R. W., Chen J. H., Ma K.: In Situ Visualization for Large-Scale Combustion Simulations. *IEEE Computer Graphics and Applications 30*, 3 (May 2010), 45–57. doi:10.1109/MCG.2010.55. 1, 2