Virtual Rheoscopic Fluids for Dense, Large-Scale Fluid Flow Visualizations

Paul A. Navrátil†  William L. Barth†  Hank Childs†
Texas Advanced Computing Center  Lawrence Berkeley National Laboratory

ABSTRACT

Fluid flow visualizations typically employ a combination of isosurfaces and particle tracing, such as streamlines. These techniques capture surface flow well, but they can obscure deeper features from view. Further, these techniques present visualizations that are far removed from physical methods, which complicates their analysis. However, virtual rheoscopic fluids (VRF) provide a physically-based visualization that is similar to, and can be similarly analyzed as, physical rheoscopic fluid. However, we need a scalable VRF algorithm to apply the technique to large-scale data.

In this paper, we present a scalable VRF algorithm based on ray casting implemented in a general visualization toolkit. We demonstrate the application of these techniques to large-scale flow problems beyond what would be physically feasible in the laboratory, and we show that our approach scales through 256 cores.

Keywords: Virtual rheoscopic fluid, Fluid flow

1 INTRODUCTION

Visualization of fluid flow typically uses non-realistic methods, such as streamlines or vector glyphs. While researchers have become used to interpreting such images, they are removed from our everyday experience. Physically-based methods, however, mimic lab methods and can be analyzed in the same way. Simulated dye or rheoscopic fluid reveal both complex flow structures and surface flow more readily than non-realistic methods.

Rheoscopic fluid in particular is useful for visualization since the suspended particles do not dissipate through a fluid like a dye. The particles align with fluid flow, becoming more opaque as alignment increases. Because the structures are typically semi-opaque, it is possible to see into the fluid to identify deeper structures. However, physical rheoscopic fluid is only used for lab experiments, as it is impractical to use in larger fluid experiments. Virtual rheoscopic fluid (VRF) can be used to simulate large-scale flows that are impractical or impossible to study physically. Unfortunately, previous virtual rheoscopic fluid efforts [6, 14] are custom single-node, GPU-based implementations, which were not built for generality or scalability.

In this paper, we adapt the two virtual rheoscopic fluid algorithms in the literature for use in a general-purpose visualization pipeline. By this transformation, we are able to leverage a parallel visualization pipeline to study large-scale flows. Our algorithms demonstrate up to 185\% scaling efficiency through 256 cores, and yield visualizations that more clearly identify structures within flows than non-realistic methods.

The contributions of this paper are:

• a scalable VRF algorithm usable within a ray casting framework; and

The remainder of the paper is as follows: we discuss related work in Section 2; we present our new algorithm in Section 3; we give our methodology in Section 4; we show our results in Section 5; and in Section 6 we offer future work and conclude.

2 RELATED WORK

In this section, we place our approach in the context of prior work, describe previous virtual rheoscopic fluid approaches, and survey previous work in parallel ray-casting, which is computationally similar to our virtual rheoscopic fluid generation.

2.1 Virtual Rheoscopic Fluid

Recent papers by Barth and Burns [6] and Hecht et al. [14] describe the implementation of two different methods for computer simulation of VRFs using volume rendering with ray casting and tracing methods on a GPU. Both methods employ a local orientation derived from the flow velocity field rather than the traditional gradient of a scalar field.

Barth and Burns choose their orientation vector from the resultant shear force in the velocity direction by removing the component parallel to the velocity. This orientation is given by

\[ n = 2Du\hat{u} - 2 \left( \hat{u}^T Du\hat{u} \right) \hat{u} \] (1)

where \( \hat{u} \) is the velocity and \( Du \) is the symmetric part of the velocity gradient tensor. Their method gives a local orientation vector everywhere in the flow field and requires the computation of the velocity gradient tensor. This orientation choice is ad hoc and justified through an appeal to idealized microscopic platelets seeded homogeneously throughout the flow field.

Hecht et al. select their orientation based on a physical model for the long-time behavior of axisymmetric triaxial particles seeded in the flow field. This model presents a time-dependent ODE representation of the particle major and minor axes based on the gradient of the velocity field. The long-time behavior of these particles is then governed by the eigenvalues and eigenvectors of the velocity gradient tensor.

There are three relevant cases to consider in the eigenanalysis: three real, unequal eigenvalues; a positive-real-part complex conjugate pair; and a negative-real-part complex conjugate pair. In the first two cases, one of the eigenvectors yields the desired orientation, however, the negative-real-part complex conjugate case represents a constantly tumbling particle without a well-defined orientation. This latter case requires special treatment in their implementation, and leads to problems in implementation in software volume rendering pipelines as described below.

2.2 Ray-Casting

From a computational perspective, virtual rheoscopic fluid rendering is highly similar to ray-casted volume rendering [19]. Rays are cast for every pixel on the screen, with samples calculated along each ray. Rheoscopic and volume rendering differ only in the types of data evaluated at the samples and how color and transparency
is decided for each sample. To tackle efficient parallelization of this computational workload, Ma and Crockett introduced a parallel sort-last cell-projection algorithm [20]. Their concept was later extended by Childs, Duchaineau and Ma [10] to guarantee fixed upper bounds on performance regardless of camera position. This latter system, implemented in VisIt [9], was adapted for the study presented in this paper.

Multiple studies have shown that parallel ray-casting techniques can be successfully scaled up to high levels of concurrency and work on very large data sets. Peterka et al. [21, 22] demonstrated scaling characteristics up to 32,768 cores. Howison et al. [16, 15] performed a separate study, designed to measure the benefits of hybrid parallelism, and considering scalability up to 216,000 cores. Finally, the VisIt system, including its volume renderer, was shown to perform well with data sets consisting of trillions of cells per time slice using tens of thousands of cores [11]. All of these findings provide evidence that the computations required for virtual rheoscopic fluid rendering on large data sets can be addressed through parallelism.

3 Algorithm Overview

This section describes the specific challenges for integration of VRF algorithms into general visualization pipelines and our new algorithms to work within those constraints.

3.1 Visualization Pipelines

We first describe our approach in a serial environment and then describe how it extends to a parallel setting.

Our approach fits well within a data flow network design [23, 24, 17, 2], and we visualize virtual rheoscopic fluids by constructing a pipeline of filters. One of these filters constructs a gradient field that is used for surface lighting when we do ray-casting. Arbitrary filters can be inserted in the pipeline, of course: users can opt to limit the volume based on spatial clips, thresholding out regions that contain uninteresting values, etc. Once the pipeline is constructed, it is executed, creating the final volume for virtual rheoscopic rendering. Each rendering step operates on this resulting data set. Ray-casting work occurs once per render, but the pipeline is executed only one time overall.

The rendering is done with a sampling-based ray-caster. Hundreds to thousands of samples are evaluated along each ray, calculating the value of one or more fields, transforming each sample to a color and opacity, and then compositing along a ray to get a final pixel color. When doing a standard volume rendering, a scalar field is sampled and a transfer function is applied to find color and opacity. For virtual rheoscopic fluids, a velocity field is sampled and a different transformation is used to determine color and opacity (see sections 3.2 and 3.3 to learn more about this transformation).

Our parallelization strategy depends on data parallelism. Each processing elements loads, processes, and renders a piece of the larger data set. The processing elements instantiate identical data flow networks; the only thing differentiating the data flow networks is the piece they operate on. The data parallel approach requires parallel coordination when ray-casting, however. After each processing element samples its portion of the larger data set, the samples for any ray are likely distributed over many processing elements, preventing compositing. To account for this, the pixels are partitioned over the processing elements and the samples are redistributed so that each processing element has all of the samples for its pixels. Again, our approach evolved from the volume rendering ray-caster described by Childs et al. [10], which means that it can deal with unstructured meshes and makes no assumptions about the ordering of pieces or that the pieces can be ordered from front-to-back for a given camera location.

3.2 Tensor-Based VRF

To implement tensor-based VRF in the pipeline described above, we modified Barth and Burns’s algorithm [6] to function within VisIt’s volume rendering pipeline. We first compute a normal at each data point using Equation 1; we then compute an opacity at each sample by taking the dot product of the generated normal and the view vector. This opacity is used to compute each sample in the direct volume renderer, described above, and the sample is shaded by any direct lighting. The samples are then composited front-to-back, with the later samples attenuated by the opacity of the ones before them.

3.3 Eigen-Based VRF

To implement eigen-based VRF, we followed the same model as for the tensor-based VRF, but with additional modifications to the rendering pipeline to accommodate the three cases Hecht et al. describe [14]. We use the GNU Scientific Library (GSL) v1.15 [13] to perform the eigenalysis, the result of which can fall into three categories. The first two cases (where the particle is stable and where the particle is rotating around the minor axis) generate a normal that we use to compute the opacity in a way similar to the tensor-based method. The third case (where the rheoscopic particle is spinning along the major axis), however, has no constant normal and instead generates an intensity. This return value type conflict causes two complications when being implemented in a general visualization pipeline:

- Normals and intensities are generated at different parts of the pipeline, so the intermediate value from the eigenalysis (the major axis from the third case, computed at normal generation) must be stored until the sampling phase, where intensity is calculated.
- Normal generation occurs before sampling, so the normal at a particular sample point is interpolated from the surrounding generated normals; since the third case does not return a normal, the values for those cases are represented as zeros and their interpolation causes a visual artifact between data in the first two cases and data in the third case (see Figure 1).

Hecht et al. pre-processed their data and customized their visualization pipeline to avoid these artifacts, but doing so in an established, general visualization pipeline would put special cases inside the basic ray-casting mechanism and affecting all visualizations that use the ray casting code.

In the ray casting sampler, we calculate the opacity using the dot product of the light direction and the generated normal, if there is one. Otherwise, we use the stored major axis to generate an intensity using the uniform rotation equation in [14]:

$$I_{rot} = A \cdot \frac{\sin \alpha}{2n} \int_0^{2\pi} | \sin \theta | d\theta = A \cdot \frac{2 \sin \alpha}{\pi}$$

We use this intensity as the opacity for the sample.

4 Methodology

This section describes our experimental methodology, including the hardware platform, the datasets, and the rendering methods that we used to evaluate our scheduling strategy.

4.1 System Configuration

All experiments are run on Longhorn, a 2048 core, 256 node cluster hosted at the Texas Advanced Computing Center. Each node contains two four-core Intel Xeon E5540 “Gainesville” processors and 48 GB of local RAM. All nodes are connected by a Mellanox QDR InfiniBand switch, and we use the MVAPICH2 v1.4
Figure 1: The tensor- (left) and eigen-based (right) VRF methods on a toy dataset designed to emphasize the discontinuities in the eigen method.

Figure 2: The wavy Taylor vortex flow created by Barth’s and Burns’s texture-based method (left) and our direct volume rendering method (right).

MPI implementation. The Breaking Waves 1B scaling runs were run on Lonestar large memory nodes, each with four six-core Intel Xeon E7540 “Westmere” processors and 1 TB of local RAM. These nodes are connected by a Mellanox QDR InfiniBand switch, and we use MVAPICH2 v1.6 MPI implementation. Our algorithms are implemented within VisIt [9], a visualization toolkit designed to operate in parallel on large-scale data. We use the VisIt infrastructure to load data and to render the final images; we implemented all code related to the generation of VRF normals and final opacities.

4.2 Datasets
We use four datasets in our experiments: Taylor-Couette flow, internal splash, thermal hydraulics and breaking waves. A chart with the details for each dataset can be found in Table 1.

Table 1: Dataset Details

<table>
<thead>
<tr>
<th>Dataset</th>
<th>timesteps</th>
<th>nodes / step</th>
<th>size / step</th>
<th>total size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taylor Couette</td>
<td>3224</td>
<td>494 K</td>
<td>167 MB</td>
<td>524 GB</td>
</tr>
<tr>
<td>Internal Splash</td>
<td>1</td>
<td>432 K</td>
<td>14 MB</td>
<td>14 MB</td>
</tr>
<tr>
<td>Thermal Hydraulics</td>
<td>100</td>
<td>32 M</td>
<td>673 MB</td>
<td>67 GB</td>
</tr>
<tr>
<td>Breaking Waves</td>
<td>600</td>
<td>16 M</td>
<td>129 MB</td>
<td>76 GB</td>
</tr>
<tr>
<td>Breaking Waves 1B</td>
<td>10</td>
<td>1 B</td>
<td>10.7 GB</td>
<td>102 GB</td>
</tr>
</tbody>
</table>

4.2.1 Taylor-Couette Flow
Flow between concentric rotating cylinders is a classic problem in fluid mechanics (see, e.g. [4, 18] for review of the early literature). Much experimental work has been devoted to discovering and classifying the flow states and the instabilities that drive them (see, e.g. [4] where Kalliroscope was used to visualize flow features and, thereby, classify the flow regimes). The simulations of Taylor-Couette flow used in this work are generated with MGF—a 3D, parallel, finite element simulation code developed for the study of fluid flow and heat transfer problems and now extended to many other classes of partial differential equation systems [7, 5, 8].

4.2.2 Internal Splash
The second dataset we utilize is the same “internal splash” phenomenon originally described by Abaid et al. [1] and used by Hecht et al. [14] in their virtual rheoscopic fluid work. One timestep of their dataset was provided directly to us for comparison purposes. This dataset models the flow of a heavier-than-fluid ball that passes through the interface between a light and heavy fluid.

4.2.3 Thermal Hydraulics
In the thermal hydraulics data set, twin inlets pump air into a box, with a temperature difference between the inlets. The air mixes in the box and exits through an outlet. Mixing of “hot” and “cold” air, residence time in the box, and identification of both stagnant and highly turbulent regions are areas of active study. The simulation was performed using the NEK5000 code [12] with an unstructured grid comprised of twenty-three million hexahedral elements.

4.2.4 Breaking Waves
The breaking waves dataset simulates the waves generated by a ship’s hull passing through water. As the hull accelerates, the waves “break” and spray sheets from off the hull. The complex interactions among the hull, water and air create turbulence that can interfere with a ship’s smooth travel through water. These simulations allow careful study of these phenomena, since such studies are difficult to perform in situ. This particular dataset shows a simulated water flow around a NACA 0024 airfoil, previously used by Adams and Dommermuth [3]. We also use a one billion point supersampled version of this dataset.

5 RESULTS
This section evaluates our virtual rheoscopic fluid algorithm on the datasets described in Section 4. First, we discuss the visual differences between the tensor and eigen methods. We then focus on the tensor method’s utility on larger datasets, where use of physical rheoscopic fluid is impractical. Finally, we demonstrate the strong scaling of our technique on these larger sets.

5.1 Rendering Comparisons
In Section 3, we discussed the differences between the tensor- and eigen-based approaches. Figure 1 demonstrates this difference visually with a toy dataset designed to do so. The tensor version is continuous, whereas the eigen version has discontinuities between the cases that return a normal and the case that returns an intensity (see Section 3.3). These artifacts cannot be repaired unless a conditional is put deep within the rendering pipeline, which defeats the purpose of implementing these techniques in a general visualization package. For the remainder of the results, we concentrate on tensor-based VRF.

5.1.1 Verification Datasets
We verify our tensor VRF implementation against the Taylor-Couette images presented by Barth and Burns [6]. Figure 2 compares our direct volume VRF visualization of wavy Taylor vortex
flow to Barth’s and Burns’s texture-based visualization. These images are quite similar despite the differences in implementation.

We also verify our tensor and eigen VRF implementations against the internal splash dataset used by Hecht et al. [14]. The images in Figure 3 compare both our methods against the versions presented by Hecht et al. The tensor-based visualizations look similar, though the eigen-based images are visually different. The features are similar, though the shading is clearly different. The difference is likely caused by different light positions (as the eigen technique is chiefly sensitive to the light location) and different values for the albedo of the VRF particles.

5.1.2 Large Physical Datasets
To demonstrate our ability to visualize simulations of large physical phenomena, we use a simulation of air flow in a room (thermal hydrodynamics) and a simulation of a ship hull passing through water (breaking waves). Both of these datasets model situations that would be difficult to recreate with traditional rheoscopic fluid. We compare our visualizations against traditional vector visualization techniques.

Figure 4 shows our visualization of the thermal hydrodynamics dataset compared to a streamline representation. The VRF shows fine-grained details of the flow, and the inherent transparency reveals internal structures not visible in the streamline image. In addition, the animation of the VRF representation more clearly captures the flow evolution than streamlines.

We show the results for the breaking waves dataset in Figure 5. While the isosurface captures the waves well, which was the original purpose of Adams’s and Domermuth’s study [3], the VRF version better represents the complex surface and sub-surface turbulence throughout the dataset. If enough streamlines were included in the image, it would obscure inner features, as in the thermal hydrodynamics dataset.

5.2 Scaling
We performed strong scaling runs for both the thermal hydrodynamics dataset and the up-scaled, 1 billion node breaking waves dataset. We present the scaling efficiency and the raw times of each set in

Figure 4: The thermal hydrodynamics dataset with tensor-based VRF (top) and streamlines (bottom). The VRF image shows finer details and internal flow not visible in the streamline image.

Figure 6: Strong scaling efficiency for the thermal hydrodynamics and up-scaled breaking waves data and the corresponding timings. Scaling for these datasets tails at 256 cores and 128 cores, respectively, when available parallelism has been exhausted.
Figure 5: The breaking waves dataset with tensor-based VRF (left) and an isosurface with streamlines (right). The VRF image shows complex flows and turbulence throughout the data that neither the isosurface nor streamlines capture.

Figure 6. The hydraulics dataset shows super-linear scaling through 256 cores, where the parallelism available in the dataset is exhausted. Whereas rendering a frame on one core takes an average of 341 seconds, rendering on 256 cores takes one second per frame. In contrast, rendering the breaking waves dataset on one core takes 28 seconds on average. At 32 cores, rendering takes 4.9 seconds on average, and much of the available parallelism has been claimed. Increasing the core count eight-fold to 256 only yields a $3 \times$ speed-up in runtime.

6 Discussion

In this paper, we have presented a scalable algorithm for virtual rheoscopic fluids that leverages the capabilities of a general visualization toolkit. Our technique captures more details than traditional vector flow visualization methods, and because we have implemented it within a scalable framework, VRFs can be used for simulations of large physical phenomena where traditional rheoscopic fluids cannot be used. We demonstrate that our algorithm scales to over 256 cores and is able to exploit the available parallelism in our test datasets.

We hope to further improve the performance of our algorithm with improvements to the underlying volume renderer. We also plan to package the VRF code to be included in VTK-based visualization toolkits.

Acknowledgements

We thank Nipun Kwatra, Chris Wojtan and Mark Carlson for their work on the internal splash dataset. We also thank Karla Vega for the breaking waves comparison image. This work was funded in part by U.S. National Science Foundation grant OCI-0906379; by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract DE-AC02-05CH11231; and by the Scientific Discovery through Advanced Computing (SciDAC) Institute of Scalable Data Management, Analysis and Visualization.

References


