Volume Renderings of Sheared Thermal Convection

Jean M. Favre[a], Alexander Blass[b]

[a]Swiss National Supercomputing Center (CSCS), Via Trevano 131, CH-6900 Lugano, Switzerland
[b]Physics of Fluids Group, Max Planck Center for Complex Fluid Dynamics, J. M. Burgers Center for Fluid Dynamics and MESA+ Research Institute, Department of Science and Technology, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

We present visualizations of sheared thermal convection in large scale wall-bounded turbulent flow simulations. The visualization is based on volume renderings of the temperature field. To address the needs of supercomputer users with different hardware and software resources, we evaluate different implementations supported in the ParaView[1] environment: two GPU-based solutions with Kitware's native volume mapper or NVIDIA's IndeX library, and a software-only OSPRay-based implementation.

Keywords:
scientific visualization, high performance computing, Naviers-Stokes solver

1. Introduction

Oceans play a bit role in the nature of our planet. About 70% of our earth is covered by water. Strong currents are transporting warm water around the world and therefore don't only make life possible, but also allow us to harvest its power producing energy. But humanity tends to easily forget that oceans also yield a much more deadly side. Floods and tsunamis can easily annihilate whole cities and destroy life in seconds. The earth's climate system is also very much linked to the circulation in the ocean due to its large coverage of the earth's surface. One has to treat the ocean with respect and carefulness, but it is unambiguously clear that humanity and its science is just attempting to reach the first step on an oceanic staircase to understanding nature.

2. Numerical Simulations

A more fundamental setup of this natural mechanism is sheared thermal convection. Many processes in nature can be based on the interaction of heat and momentum transfer and therefore the correlation of buoyancy and shear in a flow. Direct numerical simulations were performed with our second-order finite-difference code [2] using the following three-dimensional non-dimensional Navier-Stokes equations with the Boussinesq approximation on a staggered grid:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u} = -\boldsymbol{\nabla}P + \left(\frac{Pr}{Ra}\right)^{1/2}\nabla^2\boldsymbol{u} + \theta\hat{z}, \tag{1}$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0, \tag{2}$$

$$\frac{\partial \theta}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\theta = \frac{1}{(PrRa)^{1/2}}\nabla^2\theta. \tag{3}$$

We use $\boldsymbol{u} = u(\boldsymbol{x}, t)$ as the velocity vector with streamwise, spanwise and wallnormal components. θ is the non-dimensional temperature ranging from $0 \leq \theta \leq 1$. The simulations are performed in a computational box with periodic boundary conditions in streamwise and spanwise directions and confined by a heated plate below and a cooled plate on top of the flow. The shearing of the flow is implemented by a Couette flow setting where both top and bottom plates of the flow are moved in opposite directions with the speed u_w keeping the average bulk velocity at zero and therefore minimizing dissipation errors. The size of the flow field is $(L_x \times L_y \times L_z) = (9\pi h \times 4\pi h \times h)$ using a grid of $(n_x \times n_y \times n_z) = (6912 \times 3456 \times 384)$ which is homogeneously distributed in the streamwise and spanwise directions and clustered with an error-function $\eta(\xi) = erf(2\xi)/erf(2)$ in wallnormal direction.

Our open sourced finite-difference Navier-Stokes solver *AFiD* [2] was written in Fortran 90 to study large-scale wall bounded turbulent flow simulations. In collaboration with NVIDIA, USA, the code was ported in its newest version to a GPU setting using a MPI, OpenMP and CUDA Fortran hybrid implementation optimized to run and solve large flow fields.

For this visualization showcase, data from Blass et al. was used [3], where a parameter study over different input parameters was conducted to study the influence of such to the flow field. Control parameters were the temperature difference between the top and bottom plates as the strength of the thermal forcing, non-dimensionalized as the Rayleigh number (Eqn. 4), and the wall velocity as the strength of the shear forcing, non-dimensionalized as the wall shear Reynolds number (Eqn. 5), while keeping the Prandtl number (Eqn. 6), which is the ratio of thermal and viscous viscosity of a fluid, at unity. The equations read:

$$Ra = \frac{\beta g h^3 \Delta T}{\kappa\nu}, \tag{4}$$

$$Re_w = \frac{h u_w}{\nu}, \tag{5}$$

$$Pr = \frac{\nu}{\kappa}, \tag{6}$$

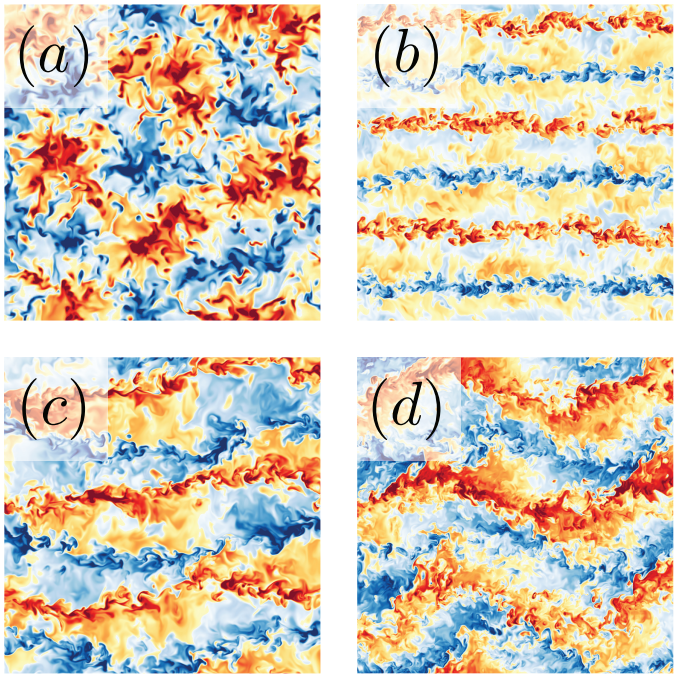

Figure 1: Instantaneous snapshots of temperature fields of a sheared and thermally forced flow transitioning through all flow regimes for $Ra = 2.2 \times 10^6$ and (a) $Re = 0$, (b) $Re = 2000$, (c) $Re = 3000$, (d) $Re = 6000$

where β is the thermal expansion coefficient of the fluid, g the gravitational acceleration, κ the thermal diffusivity, ν the kinematic viscosity of the fluid, h the distance between the plates and u_w the wall velocity.

In Fig. 1 we present instantaneous snapshots of temperature fields in different flow regimes. It can be observed that the flow passes from a thermally dominated regime with large plumes (Fig. 1a) into a regime where the mechanical forcing is dominant. Here, large-scale meandering structures can be observed (Fig. 1d). To transition between the regimes, the flow has to pass a transitional stage, in which the thermal plumes get stretched into large streaks (Fig. 1b). If the shearing is further increased, these streaks become instable and start meandering into the final flow state (Fig. 1c).

In turbulent flows it is very important to research how certain characteristic parameters are influenced by the flow. Changing flow structures might have a supporting effect or may disrupt a previously transport-favorable flow situation. In thermal convection, such parameter is the heat transfer, non-dimensionally defined through the Nusselt number:

$$Nu = \frac{Qh}{\kappa \Delta \theta} \tag{7}$$

Here, Q is the total vertical heat flux. In pure thermal convection, also called Rayleigh-Bénard convection, the scaling between Ra and Nu has been a vital base for many studies on their pursuit towards the so called ultimate regime, where the boundary layers become turbulent and the $Nu - Ra$ scaling changes. In the present study from Blass et al. [3] this was continued by implementing a wall shearing to the previously pure thermally driven flow to research if the expected ultimate $Nu - Ra$ scaling can be achieved this way. As previously observed in Fig. 1, the amount of imposed mechanical forcing is a key factor for the emergence of various large-scale structures. These structures are then a vital tool to study the behaviour of the heat in the system.

The mesh used in our simulations is made of $6912 \times 3456 \times 384$ grid points. The temperature scalar field stored as *float32* takes 36Gb of memory, an overwhelming size to handle on a normal desktop. Using different parallel programming paradigms has enabled us to provide an engaging environment to promote interactive tuning of visualization options and high productivity for movie generation. We use ParaView v5.5.2, a world-class, open-source, multi-platform data analysis and visualization application installed on Piz Daint. Piz Daint, a hybrid Cray XC40/XC50 system, is the flagship supercomputer of the Swiss National HPC service. We have deployed and tested several solutions within ParaView where parallelism is expressed at different degrees: data-parallel visualization pipelines with GPU-based renderings, or multi-threaded parallelism for software renderings.

3. Volume rendering libraries and setup

Visualization of 3D scalar fields is a very mature field. Many techniques are available to get some sense of the 3D nature of the data, and its variations through the volume. Surface-based renderings with isosurface thresholds or slicing planes haved a great appeal in that they are easy to use, but volumetric renderings, first made popular in medical applications, are also a great fit for scalar visualizations, especially in the realm of time-dependent outputs.

ParaView's "Smart" VolumeMapper is an image mapper that will delegate to a specific volume mapper based on rendering parameters and available hardware. It is a convenient wrapper to test multiple options. The largest partition of the Piz Daint supercomputer nodes is equipped with one Intel Xeon E5-2690 (12 cores, 64 Gb RAM) and one NVIDIA Tesla P100 GPU (16 Gb RAM), so our preferred rendering mode will be the GPU ray casting method; Alternatively, the vtkFixedPointVolumeRayCastMapper can be used, but it is of less interest in the use case of movie production. We aim to test ParaView's GPU ray casting implementation, against an NVIDIA-developped custom library called IndeX, and against a CPU-only solution, to give a valid option to users of supercomputers not equipped with GPUs. Our performance evaluation is based on ParaView's benchmarking Python source code[1].

For all methods used, we have ignored disk-based I/O costs. There is often quite a bit of variability when running on a large distributed file system shared by hundreds of users. Our motivations are rendering-centric, and two-fold: evaluate the memory cost and resources (CPU, GPU) required to get a first image on the screen, and see if color/opacity transfer function editing, as well as other image tuning can be done in real time, using any

[1] source code found in ./Wrapping/Python/paraview/benchmark/

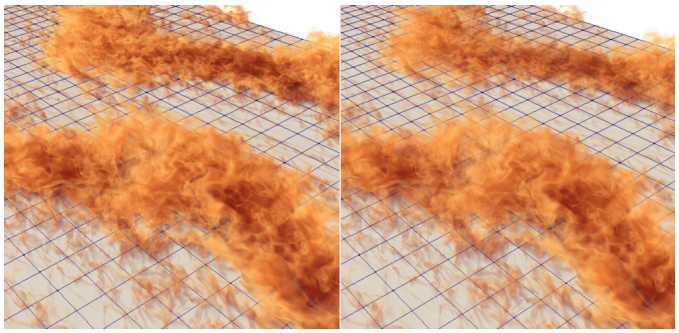

Figure 2: Volume renderings of temperature with ParaView's Smart Shader (left), and with NVIDIA IndeX (right)

of the three methods proposed. In the evaluation of rendering costs, ParaView's benchmark code does a careful management of double buffering, saving images from the back buffer to bypass driver optimizations.

In the two GPU-based methods evaluated, we use an EGL-based rendering layer, relaxing the need to have a server-side X-Windows server running on the compute node. This enables offscreen rendering with GPU acceleration. We note however that although the GPUs provide phenomenal compute power, they are limited by the available memory, 16Gb in our case. For our simulations outputs, we are forced to use data-parallel pipelines on multiple nodes to use the aggregate memory of the different GPUs.

Our third option, however, uses OSPRay and offscreen rendering but does not suffer from a memory hard limit. Compute nodes are easily and often found with large memory banks. We use Piz Daint's high memory nodes with 128 Gb of RAM, where our mesh of over 9 billion voxels can be rendered easily on a single node.

3.1. ParaView's GPU Ray Casting

ParaView's most efficient mapper - if the GPU hardware supports it - is a volume mapper that performs ray casting on the GPU using fragment programs. Care must be taken when evaluating and comparing the different approaches. We highlight the fact that in this preferred mode, Paraview converts the 32-bit float data to 16-bit integer. In Fig. 2(left), we note very small differences of illumination with the NVIDIA IndeX-rendered image at full 32-bit resolution, but no degradation due to down-converted data.

3.2. NVIDIA IndeX

NVIDIA IndeX[4] is a commercial 3D volumetric visualization SDK developped to enable the visualization of massive datasets. NVIDIA has worked in tandem with Kitware to bring an implementation of IndeX to ParaView, and we have enjoyed the benefits of a close partnership between the Swiss National Supercomputing Center (CSCS), and NVIDIA, to be able to use IndeX in a multi-GPU setting. We use the ParaView plugin v2.1 with the core library NVIDIA IndeX 1.5 (build 299900.2384). In Fig. 2(right), an NVIDIA IndeX rendering is done with identical color and opacity transfer functions as used in 3.1. Small differences can be accounted for a non uniform interpretation of the sampling rates for the scalar opacity unit distance as coded in the three proposed implementations.

3.3. Intel OSPRay

OSPRay[5] is a ray tracing framework for CPU-based rendering. It supports advanced shading effects, large models and non-polygonal primitives. OSPRay can distributes "bricks" of data as well as "tiles" of the framebuffer, although in our current implementation, we only use brick subdivisions. The Texas Advanced Computing Center has developed a Paraview plugin which enables us to test the possibility of using a ray-tracing based rendering engine for volumetric rendering. This is the best solution for clusters where no GPU hardware is available.

In the case of OSPRay, we rely on another parallel computing paradigm. The emphasis is no more on data parallism, but rather on multi-threaded execution. A complete *software-only* ParaView installation was deployed with an LLVM-based and OpenGL Mesa layer. We used Mesa v17.2.8, compiled with LLVM v5.0.0, and the OSPRay v1.6.1 library to provide a very efficient multi-threaded execution path taking advantage of Piz Daint's alternate partition of compute nodes. These nodes are built with two Intel Broadwell CPUs (2x18 cores and 64/128 Gb RAM). We run ParaView with SLURM's option "–cpus-per-task=72 –ntasks-per-core=2", effectively taking full advantage of the multi-threading exposed by the LLVM and OSPRay libraries.

3.4. Parallel Image Compositing

ParaView's default mode of parallel computing is to use data-parallel distribution, whereby sub-pieces of a data grid are processed through identical visualization pipelines. To combine the individual framebuffers of each computing nodes, Paraview uses Sandia National Laboratory's IceT[6] compositing library. We use it in its default mode of operation doing sort-last compositing for desktop image delivery. We note here that NVIDIA's IndeX uses a propriatary compositing library, so for the IndeX tests only, we disable ParaView's default image compositor.

4. Volume rendering of the thermal convection

In visualizing the temperature field, we seek to highlight the turbulence which is best shown by clearly differentiating between cold and hot regions to see how they interact with each other. Our movie animation shows an initial phase where region of blue tint is superposed on top of the hotter region. Plumes emerging from the bottom and mixing into the cold regions highlight this phenomenom.

4.1. Visual effects

When presented with multiple visualizations including different illumination and shading, the scientists actually preferred the renderings which emphasize the amorphous nature of the field data. As can be seen in Fig. 3, shading based on gradient estimation offers little improvement because our data does

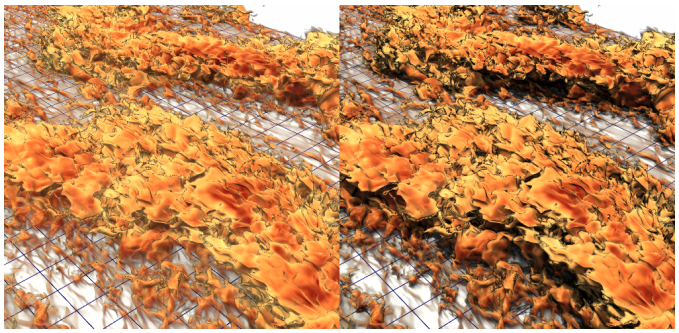

Figure 3: Volume rendering with shading based on gradient estimation (left), and with OSPRay-enabled shadows (right)

not have strong gradients, and the use of shadows which at first might seem more appealing, produces images with a strong *surface-like* look, which we discarded upon further analysis.

4.2. Overall costs

Volumetric rendering at this scale of grid has a non significant cost which we briefly document here. Creating the first frame after data has been read in memory, i.e. the startup cost has a great impact in having users adopt a particular implementation. In a *post-hoc* visualization, data would be read from disk; in an *in-situ* scenario, data might have to be converted to VTK data structures. Thus, we count time from the moment ParaView has collected all the data, and we toggle the volumetric rendering options which trigger the build of internal structures for ray-tracing. If memory costs are substantial, more nodes, and/or more GPUs will be required, increasing the runtime cost of the visualization. Our results are of great interest; we were pleasantly surprised to find out that the OSPRay based implementation, using a single node and 72 execution threads makes the first image at a cost of roughly 7-8 seconds. After the first frame, interactive renderings are done in sub-second times, color and opacity transfer functions editing is also interactive and very intuitive. We ran all tests on different combinations of nodes, at HD (1920x1080 pixels) and 4K UDH (3840x2160 pixels) resolutions.

4.3. Benchmarking tests

We have done this, and this, to evaluate the performance.

Table 1 summarizes our experiences. In the first column, we show the startup cost computed from the time all the data has been read to disk into a partitioned VTKImageData grid, until the first frame is seen in volumetric rendering mode. In the next column, we show average rendering times for all subsequent rendering for 300 frames. The memory cost per node is then shown in the last column.

5. Summary and conclusion

Based on the benchmark results and visual preferences we adopted the implementation featuring the XYZ library, because justify, justify...

Table 1: Startup time for the first frame, all other frames, and memory costs per node.

Method	Startup	Subsequent frames	memory
”Smart” Mapper 2GPUs	0.0	0.0	10.0
”Smart” Mapper 4GPUs	0.5	3.8	5.1
NVIDIA IndeX 4GPUs	1.0	5.1	0.2
OSPRay 1 node	1.5	4.0	-4.7
OSPRay 2 nodes	2.0	0.4	-9.6
OSPRay 4 nodes	2.5	-5.6	-14.5

Acknowledgments

Alexander Blass conducted his simulation runs at the Swiss National Supercomputing Center, under compute allocations s713 and s802. The authors thank the ParaView development team at Kitware, USA, for fruitfull discussions and motivational material. Dave Demarle has been particularly helpful in discussion related to the OSPRay plugin. Mahendra Roopa at NVIDIA has also been extremely receptive to our feedback and instrumental in helping us get the best of the IndeX library in a multi-GPU setting.

References

[1] J. Ahrens, B. Geveci, C. Law, Paraview: An end-user tool for large data visualization.

[2] E. P. van der Poel, R. Ostilla-Mónico, J. Donners, R. Verzicco, A pencil distributed finite difference code for strongly turbulent wall-bounded flows, Computers & Fluids 116 (2015) 10–16.

[3] A. Blass, X. Zhu, R. Verzicco, D. Lohse, R. J. A. M. Stevens, Direct numerical simulations of sheared thermal convection, in prep. for J. Fluid Mech.

[4] Nvidia index, `https://developer.nvidia.com/index`.

[5] Ospray: a ray tracing based rendering engine for high-fidelity visualization, `http://www.ospray.org/index.html`.

[6] K. Moreland, W. Kendall, T. Peterka, J. Huang, An image compositing solution at scale.