Abstract

We present an exemplary steering system that performs 2D flow simulation and visualization on graphics processing units (GPUs). The topology of a vector field provides the overall structure and therefore lends itself for steering purposes. We build on the concept of Lagrangian coherent structures present as ridges in the finite-time Lyapunov exponent (FTLE). This allows to perform steering with respect to the true time-dependent dynamics in a given time scope. Based on the insights from the FTLE visualization, our CUDA-based implementation allows effective interactive manipulation of boundary conditions such as solid obstacles or velocity profiles.

Keywords: computational steering, flow visualization, time-dependent vector fields, Lagrangian coherent structures, finite-time Lyapunov exponent, vector field topology, GPU.

1 Introduction

Interactive investigation of CFD flow has several advantages. It can provide a means of achieving targeted flow behavior fast and in an intuitive way, since changes of the parametrization of the simulation are directly reflected in the visualization. Interaction with such a system can also provide faster and better understanding of the underlying mechanisms, improving design efficiency, and possibly leading to new approaches and even theories.

Whereas building such a system for steady-state simulation is merely a question of compute time and efficient simulation, it is not straightforward to design such systems for time-dependent flow. Although using traditional visualization concepts such as instantaneous glyphs or streamlines can provide some insight, these typically lack the notion of time-dependency, or in other words, a *time scope*, to account for dynamic

processes inside the flow.

Lagrangian coherent structures (LCS) represent an increasinlgy popular concept that features such a time scope: it reflects the overall organization in flow maps, which take seed points of path lines to their end points over a given time scope. In our context, LCS are present as ridges in the finite-time Lyapunov exponent (FTLE) field [1], which is computed by finite-time advection. This not only provides a large-scale representation of the structure of vector fields similar to vector field topology [2], it furthermore accounts for time-dependency by incorporating true advection processes. We provide an introduction to FTLE and its computation in Section 2.

Besides FTLE and LCS, traditional flow visualization techniques like geometric path lines or streak lines [3] provide additional tools for vector field exploration and are also capable of interactive performance with modern graphics hardware [4, 5]. Advanced systems like VisTrails [6] combine features of workflow and visualization systems for exploratory computational tasks.

Computational steering is the process of reparametrizing an application or a simulation based on the analysis from visualization to improve a solution to a given problem. Van Liere et al. [7] presented a formal description of a computational steering environment (CSE) and its requirements. Mulder et al. [8] classified CSEs such as SCIRun [9] with a taxonomy in a survey paper, which we will adopt for our approach in Section 3.3. However, typical steering environments for CFD applications exhibit large compute clusters to achieve interactive performance, e.g., the Lattice Boltzmann approach by Hardt et al. [10]. In this paper, we employ commodity graphics hardware on a standard workstation to perform interactive flow steering.

2 Background of Finite-Time Lyapunov Exponent

Vector fields exhibit a spectrum of Lyapunov exponents. It is the largest exponent in this spectrum that has become a prominent tool for predictability analysis in time-dependent vector fields: denoted Lyapunov exponent (LE) throughout this paper. The LE can be determined by computing two neighboring trajectories in phase space and measuring their separation rate as time approaches infinity. Thereby, precaution has to be taken to assure that the trajectories do not separate too far, e.g., by renormalization [11].

Since the systems under investigation are often defined on a finite temporal domain only, or because it is often the aim of the researcher to restrict the analysis to a temporal region of interest, a variant has becoming more and more popular: the finite-time Lyapunov exponents (FTLE). Again, there are techniques to assure proximity of the (implicitly) involved trajectories, such as the localized FTLE [12].

Whereas the LE and FTLE have been used for a long time for predictability analysis, there is a recent trend in the visualization community to use FTLE for revealing the topology of time-dependent vector fields. Haller [1] showed that ridges present in the FTLE represent a time-dependent counterpart to separatrices from vector field

topology [2]; they separate regions of different behavior.

In the context of time-dependent vector field topology, the FTLE is typically computed from the *flow map* $\phi_t^{t+T}(\mathbf{x})$, a mapping from seed points $\mathbf{x}$ of trajectories to their end point after advection for finite time T. According to Haller [1], the finite-time Lyapunov exponent $\sigma(\mathbf{x}, t)$ computes from the flow map ϕ as follows:

$$\sigma(\mathbf{x}, t) = \frac{1}{|T|} \ln \sqrt{\lambda_{\max} \left[\left(\frac{d\phi_t^{t+T}(\mathbf{x})}{d\mathbf{x}} \right)^{\top} \frac{d\phi_t^{t+T}(\mathbf{x})}{d\mathbf{x}} \right]}, \tag{1}$$

$\lambda_{\max}$ being the largest eigenvalue.

Beyond separating regions of qualitatively different behavior, LCS represent transport barriers with respect to advection. Hence, they can be used, for example, in the context of pollution: if diffusive mechanisms are neglected, they provide time-dependent barriers for transport of a pollutant [13]. Another application is in the context of mixing processes. FTLE can be computed from both forward ($T > 0$) and backward ($T < 0$) flow maps and if ridges inside the former intersect ridges in the latter, this gives rise to the Lagrangian skeleton of mixing [14]. We utilize this property in Section 5.2. Intersections of forward-time and reverse-time LCS also give rise to the concept of hyperbolic trajectories [15, 16], i.e., path lines passing through these intersections, important in the context of time-dependent vector field topology and mixing.

3 Interactive Flow Steering

CFD simulations are hard to parametrize with respect to a given goal. The complex relationship between flow and boundary conditions requires extensive experience to achieve a certain behavior. Typical simulations exhibit high computational cost and generate vast amounts of multi-attribute data. The analysis of their results usually leads to a redesign of boundary conditions and to a reparametrization of the simulation. Typically, this time-consuming process is iterated until the desired aim is achieved.

In this section, we demonstrate methods for steering 2D flows interactively as shown in Figure 1 with the help of parallel graphics processing units (GPUs). We employ an interactive flow simulation based on the incompressible Euler equations that is capable of manipulating boundary conditions, such as solid walls and velocity profiles, interactively. The spatio-temporal structure of the unsteady velocity field is then visualized with FTLE and advected particles for providing insight into the flow behavior. Depending on the application, an engineer can identify regions of flow separation or attachment, related to ridges in the FTLE field, as well as general barriers of advective transport, separating regions of different behavior. The instant feedback of the simulation lends itself to steer the flow and to improve the overall result interactively by modifying the shape of solid boundaries or by changing prescribed velocity conditions.

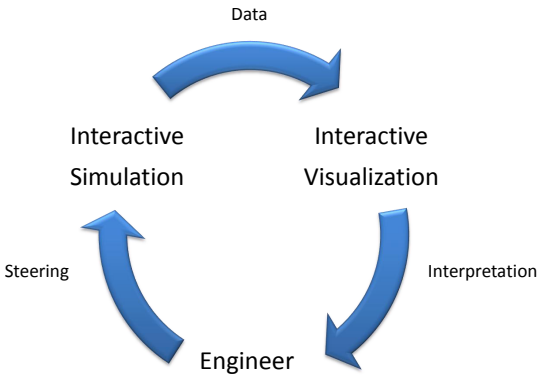


Figure 1: Process of interactive flow steering. The simulation generates data processed with specific visualization techniques, e.g., FTLE or particle trajectories. The engineer interprets the results visually and modifies parameters, such as boundary conditions, to steer the simulation interactively into a desired direction.

3.1 Flow Simulation

The first step of the interactive steering process depicted in Figure 1 is an interactive flow simulation. Typically, full 3D Navier-Stokes solvers exhibit high computational expense and are hence not capable of interactive performance. Therefore, we demonstrate our approach by reverting to the incompressible Euler equations in 2D:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \mathbf{f}, \tag{2}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3}$$

where $\mathbf{u} = (u_x, u_y)^\top$ is the unsteady velocity vector field, ρ is the constant mass density, p the pressure, and $\mathbf{f} = (f_x, f_y)^\top$ the sum of all external forces.

Equations (2) and (3) are solved numerically by splitting the operators to seperate the advection, external force, and pressure/incompressibility parts. Then, we advance the simulation with an explicit Euler step in time. Furthermore, the spatial domain is discretized with a uniform grid in 2D. For background information on CFD and its numerical solutions we refer the reader to Anderson's textbook [17]. Details of our approach are given next and in Section 4.1.

3.1.1 Velocity Advection

After splitting the equations, we can solve the advection step separately, which boils down to the inviscid Burgers equation. For interactive simulations, comparably large time steps are required; hence it should be ensured that the numerical solver does not become unstable. Stam [18] presented the semi-Lagrangian advection technique, which is unconditionally stable and well suited for large time steps. Stam's approach is based on the method of charateristics by tracing particles back in time. The velocity

is then interpolated and is advanced forward in time. The drawback of this method is numerical diffusion. It can be shown that the method actually solves the *viscid* Burgers equation; it introduces artifical viscosity to the simulation. However, this behavior alleviates the missing viscosity term in Euler's equations to some extent. In the following, we use this technique as a building block for a more advanced method and we denote the semi-Lagrangian advection with an operator $A(\mathbf{u})$.

Selle et al. [19] introduced a modified MacCormack method based on back and forth error compensation and correction (BFECC):

$$\hat{\mathbf{u}}^{t+1} = A(\mathbf{u}^t), \tag{4}$$

$$\hat{\mathbf{u}}^{t} = A^R(\hat{\mathbf{u}}^{t+1}), \tag{5}$$

$$\mathbf{u}^* = \hat{\mathbf{u}}^{t+1} + \tfrac{1}{2}\left(\mathbf{u}^t + \hat{\mathbf{u}}^t\right), \tag{6}$$

where A^R is the semi-Lagrangian advection reversed in time and $\mathbf{u}^*$ is the result of the advection step. This method reduces numerical diffusion significantly compared to Stam's approach. For our simulation, we stick to Equations (4)-(6).

3.1.2 Pressure Projection

The second step of the operator splitting is enforcement of incompressibility. The intermediate result $\mathbf{u}^*$ from the advection step is usually not free of divergence, due to advection and interpolation. As in Stam [18], we employ pressure projection to calculate the final velocity $\mathbf{u}^{t+1}$. Helmholtz-Hodge decomposition and the appliance of the divergence operator on both sides lead to the following Poisson equation for the pressure p:

$$\Delta p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}^*, \tag{7}$$

where Δt is the time step. The discretization of the Laplacian operator on a uniform grid with finite differences leads to a large but sparse system of linear equations. We solve the system numerically with a conjugate gradient solver for the unknown pressure. For more details on numerical solutions of linear systems, we refer the reader to Saad's textbook [20]. Once the pressure is calculated, the intermediate result $\mathbf{u}^*$ is projected onto its solenoidal component:

$$\mathbf{u}^{t+1} = \mathbf{u}^* - \nabla p, \tag{8}$$

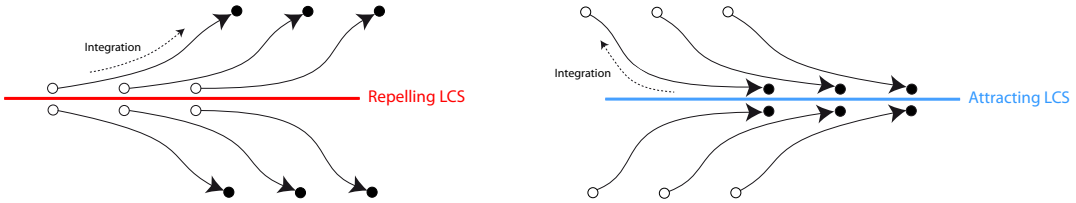
with $\nabla \cdot \mathbf{u}^{t+1} = 0$. At this stage, one simulation step is finished and is ready to be visualized.

3.2 Visualization

In this section, we demonstrate how the data from the previous simulation step is visualized to provide insight into the spatio-temporal structure of the unsteady vector field. Our main technique is the visualization of Lagrangian coherent structures with FTLE. Furthermore, we discuss well-established methods like the visualization of path lines, particle tracing, and vorticity briefly.

3.2.1 Lagrangian Coherent Structures with FTLE

The FTLE in Section 2 can be calculated using positive and negative advection times T. For positive values of T, the FTLE measures separation and yields repelling coherent structures (Figure 2(a)). For negative values of T, the FTLE measures separation in reverse time and hence yields attracting coherent structures, illustrated in Figure 2(b).



(a) (b)

Figure 2: (a) Repelling LCS present as ridge (red) in the forward FTLE; adjacent particles diverge when advected with the flow. (b) Attracting LCS present as ridge (blue) in reverse FTLE; particles converge to the LCS.

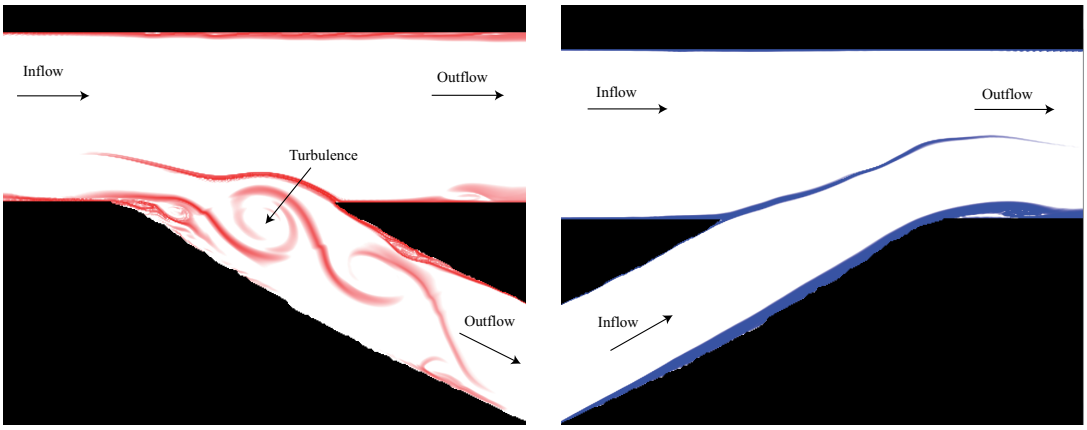
In general, the FTLE is calculated continuously throughout the entire space-time domain of the fluid. In practice, however, the domain is discretized with a finite FTLE grid that is independent from the simulation grid. For each node of the grid, a particle is seeded and is advected forward or backward in time, respectively. When the advection time $\pm T$ has passed, the final position of each particle is used to calculate the flow map and the FTLE is obtained according to Section 2.

As the FTLE is a scalar value, it can be visualized with a color coding. A continuous image of the discrete values is obtained with bilinear interpolation. High values of the FTLE indicate areas of strong Lagrangian separation in the resepective flow direction. In the visualization, these areas typically form colored ridges that represent Lagrangian coherent structures of the flow. Shadden et al. [21] showed that the sharpness of the ridges is a measure for the flux across an LCS, i.e., the quality of an LCS as a flow barrier. Therefore, we do not perform explicit ridge extraction, e.g. of height ridges [22], but leave it to the user to judge the quality and role of the ridges. Figure 3 depicts an example of both FTLE visualizations.

3.2.2 Other Visualization Techniques

In addition to FTLE, we also support typical traditional flow visualization techniques. Together with FTLE, they provide valuable tools to analyze and understand complex unsteady vector fields.

Path lines are trajectories that individual particles traverse when they are advected with time-dependent flow. A path line can be formulated as the solution of an inital value problem of a non-autonomous ordinary differential equation from the vector field. A path line contains the integrated time history of the motion of a single massless particle. It can be visualized experimentally by taking a long-time exposure record of



(a) (b)

Figure 3: Example of FTLE visualizations of two rivers. Fluid is flowing from left to right in both cases. (a) Bifurcation in forward time. The flow is split into two parts at the wedge-shaped obstacle. The Lagrangian dynamics of this splitting is conveyed by the long red ridge. Furthermore, it is visualized how the flow in the bottom arm gets turbulent. (b) Confluence with reverse-time FTLE. The long blue ridge visualizes the Lagrangian process of merging.

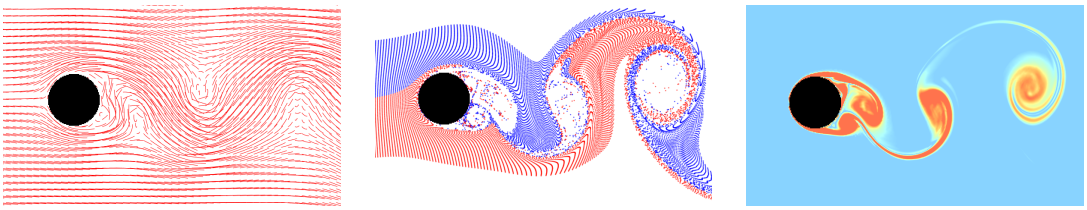
the motion of the particle. We visualize path lines by tracing particles from discrete locations in the fluid domain and by connecting the positions of each particle over time with a line strip. As path lines can be considered as the spatio-temporal record of the flow map, they facilitate the interpretation of the FTLE directly and they provide additional information about flow direction.

Injecting particles into unsteady flow is a common visualization technique. It is the direct analogon to injecting tracers in real-world flow experiments. We support this method by placing particle emitters interactively into the fluid domain and by advecting the particles with the flow. The particles are displayed as colored points. Each emitter is able to color its particles in a different color, hence mixing or clustering processes can be visualized in a straightforward manner.

Furthermore, we visualize vorticity magnitude. This is a simple way to locate eddies in turbulent flow. We calculate the curl of the velocity field and map it to a color value. Figure 4 shows the three different visualization methods for the same setup.

3.3 Computational Steering

With the visualization techniques from the previous section, an engineer can modify parameters of the simulation while it is in progress, leading to computational steering. In general, Mulder et al. [8] developed a taxonomy to classify computational steering environments (CSEs) concerning *scope*, *architecture*, and *user interface*. The diagram of a generic CSE is illustrated in Figure 5.



(a) (b) (c)

Figure 4: Flow from left to right around a sphere. Visualization with (a) path lines, (b) particle injetion and tracing, and (c) colorcoding of vorticity magnitude.

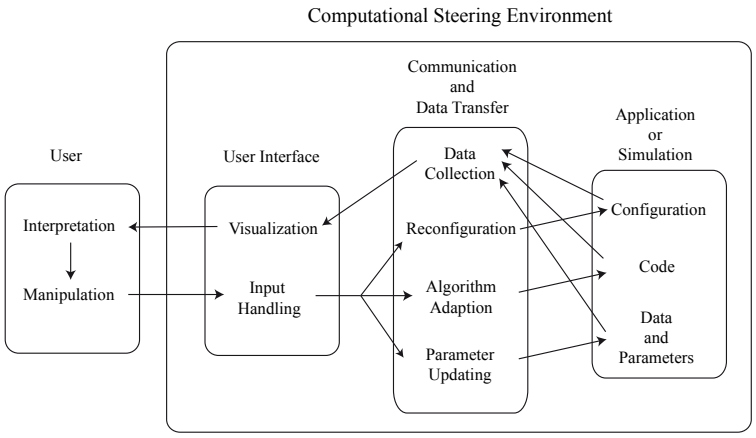


Figure 5: Diagram of a computational steering environment after Mulder et al. [8].

According to Mulder's taxonomy, the *scope* of our CSE provides steering for model exploration of unsteady vector fields from CFD. In general, Mulder's definition of model exploration deals with the application's output data and its input parameters which are in our case the velocity field and its boundary conditions.

The *architecture* of our prototype is a standalone single-threaded CPU process with access to a parallel GPU for accelerated computation. The information provided to the user include results from flow visualization (see Section 3.2). Steerable items that the user can manipulate are the insertion and the removal of solid obstacles to and from the fluid domain, which allows the user to modify the shape and position of geometry interactively. Furthermore, boundary conditions such as velocity profiles can be inserted or removed to modify flow control.

The *user interface* of our application is handled with a graphical user interface (GUI). For monitoring, the user can choose which visualization technique is best for the current task and furthermore, multiple visualizations can be blended together to provide an overall picture of the flow that a single technique cannot reveal. The steering interface lets the user modify solid and velocity boundary conditions directly in the view port window, which is discussed in detail in Section 4.3.

Interactive steering enhances productivity because it reduces the time between changes of parameters and the visualization of the result significantly, allowing for

a tight design cycle. However, steering typically requires high performance computing on large and expensive clusters with low-latency network interconnects to perform simulations and to visualize the vast amount of data at interactive speed. In this paper, we take a first step to perform interactive flow steering of reduced 2D model complexity on a single PC with commodity graphics hardware.

4 Implementation

In this section, we describe our implementation of interactive flow steering on the GPU. We employ NVIDIA's Compute Unified Device Architecture (CUDA) for general purpose programming of graphics hardware. CUDA gives developers access to the instruction set and memory of the GPU. The high floating-point performance and the high bandwidth of GPUs become accessible for computation, by providing an abstraction layer on top of the rendering pipeline. GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads (single intruction, multiple threads). The programming of the GPU with CUDA is realized either with Fortran or C and for our implementation we used the latter. The GPU code is encapsulated in kernels, which are called as functions from the regular CPU code. One of the specifics of CUDA is the possibility to use 2D and 3D textures to store data. Textures are spatially uniform data containers that are usually used to add details on geometric meshes in computer graphics. We make use of textures because the GPU provides very fast bi-/trilinear interpolation in hardware and read access is cached efficiently as long as spatial coherence is ensured.

4.1 Fluid Solver

The 2D Euler solver is an implementation of [19] for the advection and of a preconditioned conjugate gradient algorithm for the pressure projection, described in Section 3.1. The velocity field is stored in a 2D float2 (i.e., with two floating-point components) texture with a resolution according to the grid size. The semi-Lagrangian advection in Equations (4) and (5) benefits from fast interpolation and caching from the texture unit. For the Poisson Equation (7), we employ the preconditioner by Ament et al. [23] to accelerate convergence. The method is suited specifically for the Poisson problem and for parallel GPU processing. To achieve interactive frame rates, we are currently bound to single precision for the numerical solution. With current graphics hardware, we can simulate fluids of interactive performance with a grid resolution of up to 1024×1024 on a single PC. However, the computation of the FTLE in the next section is memory-consuming and we revert to a fluid domain of 512×512 for most cases.

4.2 FTLE Computation

The computation of the FTLE requires particle trajectories forward and backward in time according to $+T_{\text{forward}}$ and $-T_{\text{backward}}$. For unsteady vector fields, the FTLE is also time-dependent, which requires continuous recomputation, i.e., animated time series of the FTLE. In a naïve implementation, each frame requires to simulate the fluid for $n = (T_{\text{forward}} + T_{\text{backward}})/\Delta t$ time steps with interleaved particle tracing for the path lines. For interactive performance, this approach is typically not feasible.

We observe that there are many redundant computations between two frames, i.e., $(n-1)$ time steps are identical to the previous frame. For that reason, we come up with a time sliding window that stores redundant vector fields in a stacked buffer object. On the GPU, this buffer is represented by a 3D float2 texture. In this texture, we keep all vector fields for a period of T_{backward} backward in time and of T_{forward} ahead of current time. Figure 6 shows our approach for two time steps t_0 and t_1.

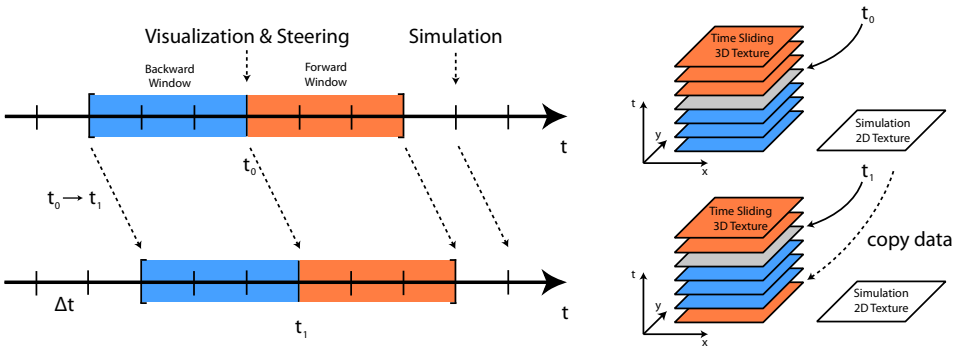


Figure 6: Left: Time line with sliding window. For each frame visualized, we store vector field information for the backward (blue) and forward (red) FTLE. The simulation runs $(T_{\text{forward}} + \Delta t)$ ahead of time. Right: Texture memory layout. Each 2D vector field is represented as a time slice in a 3D texture. At the transition from t_0 to t_1, the "oldest" slice is replaced with new simulation data.

With this approach, we can reduce the computational expense for the simulation to one time step per visualization frame. When time is advanced from t_0 to t_1, the sliding window is moved forward in time. Before the next frame can be visualized, the new simulation data is copied to the 3D texture. To reduce bandwidth we employ a dynamic time coordinate in the 3D texture, i.e., we do not advance all the data forward in time which would induce heavy data transfer, but instead we keep a dynamic pointer to the current time slice (gray shaded in Figure 6) and overwrite the earliest time slice with new simulation data. With this approach the memory requirement is proportional to the grid size and to the advection times T_{forward} and T_{backward}. For example, a grid size of 512×512 and 256 discrete time steps for each FTLE window has a memory consumption of 1GB.

For the visualization of both FTLEs in each frame, we integrate particle trajectories numerically through the spatio-temporal 3D texture as depicted in Figure 7. The

computation is very efficient because the GPU threads traverse the texture slice by slice in parallel, i.e., they traverse time simultaneously. Therefore, random access is rather small within one integration step, yielding high cache coherence and thereby good performance.

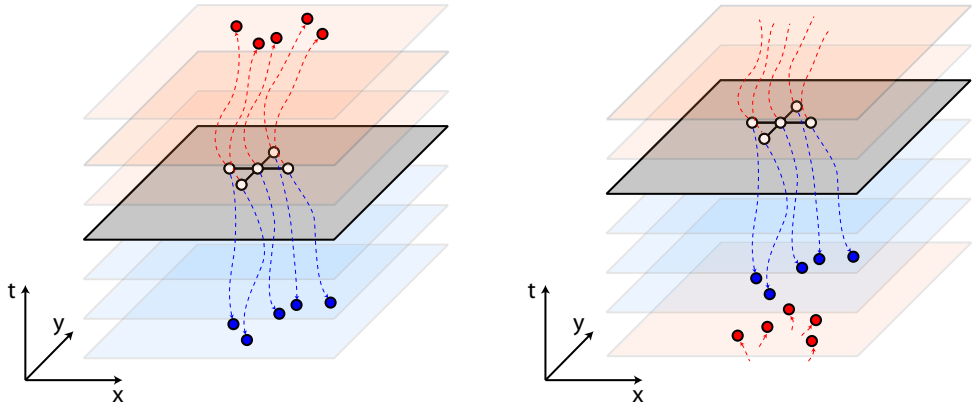


Figure 7: Left: Particle trajectories for time t_0. Right: Trajectories for time t_1. The time line is wrapped around as the sliding window is moved forward.

Once the final positions $\mathbf{x}(t + T_{\text{forward}})$ and $\mathbf{x}(t - T_{\text{backward}})$ of the particles are calculated, we store them in the flow map kept in another 2D float2 texture. This means, for each node of the FTLE grid (white points in Figure 7), the positions of the advected particles (red and blue points in Figure 7) are stored in the texture, depending on which FTLE-type is visualized.

Afterwards, we compute the gradient of the flow map with finite differences. For example, at an arbitrary point (i, j) in our FTLE grid with $T = T_{\text{forward}}$, central differences yield:

$$\left.\frac{d\phi_t^{t+T}(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}_{i,j}} = \begin{pmatrix} \frac{x_{i+1,j}(t+T)-x_{i-1,j}(t+T)}{x_{i+1,j}(t)-x_{i-1,j}(t)} & \frac{x_{i,j+1}(t+T)-x_{i,j-1}(t+T)}{y_{i,j+1}(t)-y_{i,j-1}(t)} \\ \frac{y_{i+1,j}(t+T)-y_{i-1,j}(t+T)}{x_{i+1,j}(t)-x_{i-1,j}(t)} & \frac{y_{i,j+1}(t+T)-y_{i,j-1}(t+T)}{y_{i,j+1}(t)-y_{i,j-1}(t)} \end{pmatrix} \tag{9}$$

The FTLE is then calculated according to Equation (1) and is finally stored in a 2D float texture used for mapping the FTLE value to color. The final image is visualized as a textured quad with OpenGL.

The choice of T is dependent on the particular application. In general, the longer T is, the more refined the LCS become. However, an important indicator for the appropriate choice of T is the time it takes particles to exit the domain. Therefore, if nearly all points in the FTLE grid leave the domain in, for example 2 seconds, then it does not make sense to choose $T > 2s$, since further integration would not change the FTLE field. Therefore, it is difficult to employ a constant advection time for all applications and as a consequence our tool offers the possibility to adjust the advection time for both FTLEs independently.

4.3 Steering of Boundary Conditions

In our implementation of interactive steering, we provide direct manipulation of boundary conditions. For simple editing, our application offers a pencil tool for image-based steering in the visualization view port. The pencil can have one of the following modes: set/erase solid obstacles or set/erase prescribed velocities. Furthermore, the pencil size is variable, similar to well-known tools from image editing. For the velocity, the direction and magnitude is set with a seperate GUI element. The dialog for boundary interaction is shown in Figure 8.

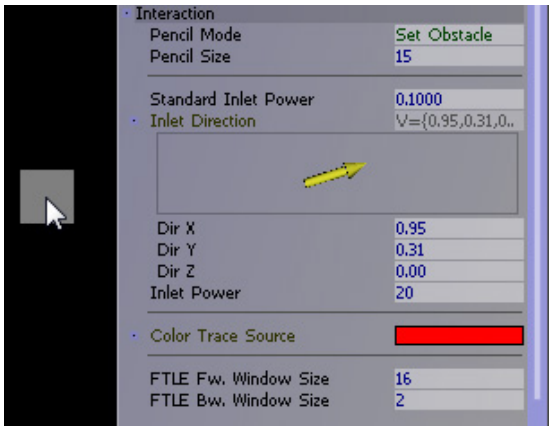


Figure 8: The user can choose the type of editing with the "Pencil Mode" together with the corresponding "Pencil Size" (in grid cells). On the left, the mouse cursor is shown and the white square underneath is a representation of the pencil's footprint. The direction and magnitude of prescribed velocity conditions is set with the yellow arrow and the "Inlet Power" GUI element (velocity magnitude), respectively. In addition to steering parameters, the interaction dialog also includes the placement of particle sources and the control of the FTLE window sizes.

With the selected tool from the GUI, boundary conditions can be modified easily with a mouse by drawing into the window domain. In our implementation, we keep two textures for representing boundary conditions. First, a 2D unsigned char texture is used to tag each grid cell with the boundary type, which can be either `BC_FLUID=0`, `BC_SOLID=1`, or `BC_VELOCITY=2`. Second, a 2D float2 texture is required to store vector information about velocity conditions. These textures are used in our kernels to determine the boundary type of each cell. The appliance of boundary conditions is implemented in the pressure projection step of the simulation. Therefore, we update the matrix of the discretized Laplacian in Equation (7) with the help of these textures whenever the boundary conditions change.

As the simulation runs ahead of the visualized time frame (Figure 6), it takes $(T_{\text{forward}} + T_{\text{backward}} + \Delta t)\,/\Delta t$ time steps until the sliding window is recalculated and the FTLE visualization is up to date. However, as our entire workflow runs interactively, this process takes only a few hundred milliseconds and during that time visualization remains smooth and responsive. An alternative strategy would be to blank

the visualization screen for the duration of the sliding window after each update of boundary conditions.

5 Results and Applications

5.1 Performance

In this section, we provide performance evaluations of our implementation with CUDA. Our test system is a standard PC platform with an Intel Core i7 X980 CPU at 3.33GHz, 12GB RAM, and an NVIDIA GTX-480 graphics card with 1536MB of video memory. Figure 9 illustrates performance in frames per second (fps).

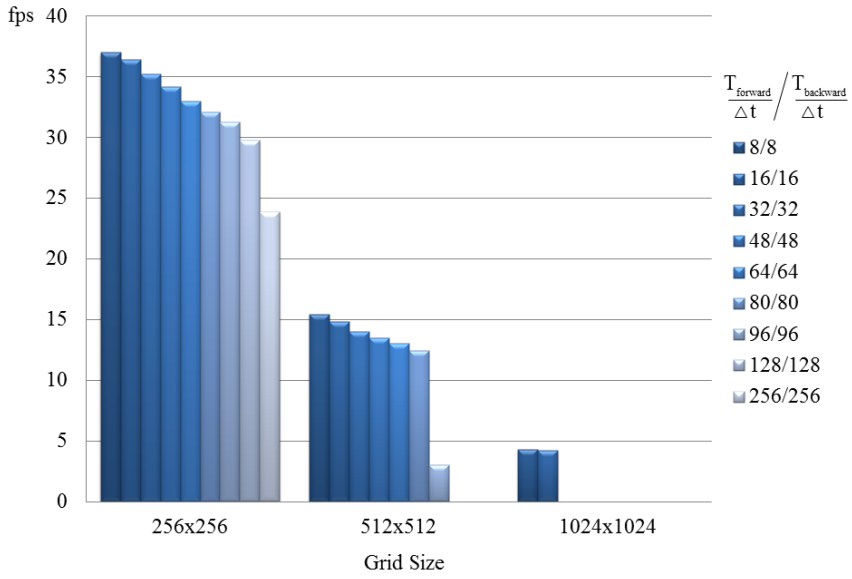


Figure 9: Performance evaluation of our interactive steering workflow. We evaluate 3 different grid sizes for the fluid and FTLE domains. The differently colored bars depict performance for varying integration lengths of the forward and backward FTLE. The drop down in performance for larger advection times results from the increased expense for calculating longer FTLE particle trajectories. For large grid sizes, there is insufficient GPU memory to store the textures for large FTLE sliding windows.

We evaluate performance for resolutions of 256×256, 512×512, and 1024×1024 cells for both grids, fluid and FTLE. In addition, we show the impact of growing advection times for the FTLE computation of Section 4.2. The legend on the right of Figure 9 depicts the sizes of the sliding windows for forward and backward FTLE. The computational expense grows linearly with the integration length due to the increased duration of particle tracing. The strong drop down for large advection times

(see 512×512 between 80/80 and 96/96) results from a bottleneck in data transfer as video memory and texture cache run out. However, for practical purposes, our implementation reaches interactive performance for grid sizes of up to 512×512 and advection times of up to $80 \cdot \Delta t$ for each FTLE.

5.2 Application: Air Conditioning

In the following, we show typical applications for interactive flow steering based on our method. First, we present an example of an air conditioning setup for a computer cluster, shown in Figure 10. The spatial arrangement of the compute nodes with respect to the inflow from the air conditioning system is crucial for a good cooling performance. Insufficient mixing reduces the overall performance of cooling, which can cause overheating and damage of hardware.

In the top row of Figure 10, the nodes are arranged in a more aligned manner compared to the bottom row. While the glyph visualization in Figure 10(a) provides only limited insight into the flow and its mixing properties, the FTLE visualization in Figure 10(b) depicts a rather simple LCS. According to Mathur et al. [14], intersections of repelling and attracting LCS in the Lagrangian skeleton indicate turbulent areas of high mixing; hence the first arrangement of nodes is likely to be cooled insufficiently. In the second arrangment, the nodes are positioned in an interleaved manner. The glyphs in Figure 10(c) show hardly much difference compared to the aligned arrangment but the FTLE visualization in Figure 10(d) depicts plenty of intersecting forward and backward LCS, indicating enhanced mixing. The fluid and FTLE domain sizes are 512×512 and the advection time is $56 \cdot \Delta t$ for both FTLEs.

5.3 Application: Airfoils with Active Flow Control

In our second example, we investigate active flow control [24] for airfoil design. By injecting energy via actuators into key areas, changes in local or global flow can improve performance of aircrafts significantly. Our computational steering supports engineers in positioning actuators and adjusting their power according to the desired flow behavior; Shadden et al. [21] have shown that FTLE can indicate flow separation, one of the main mechanisms for efficiency drops in aircraft operation. With our exemplary implementation, we can model 2D airfoil profiles and insert actuators in the form of velocity boundary conditions. Figure 11 illustrates the impact of an actuator compared to the uncontrolled flow around an airfoil. The fluid and FTLE domain sizes are 512×512 and the advection time is $64 \cdot \Delta t$ for the FTLE.

6 Conclusion

In this paper, we presented interactive steering of a 2D flow simulation on a GPU. We employed an Euler solver to simulate inviscid and incompressible fluids with inter-

active performance. Our visualization with FTLE in forward-time and reverse-time allows insight into the complex behavior of the unsteady velocity field, which is crucical for improving applications. We provide interactive steering by modifying solid and velocity boundary conditions similar to well-known tools from image editing. The instant feedback of our CUDA implementation allows efficient modeling and continuous redesigning towards a certain goal.

Our performance-oriented implementation of the fluid simulation with CUDA is currently bound to single precision, which limits numerical accuracy accordingly. Furthermore, the rather high memory consumption of the FTLE computation becomes a limiting factor for long advection times. However, in contrast to CPUs, performance of graphics processors still grows very fast due to massive parallelism, which can remedy some of these issues in the near future.

Apart from specifics of the implementation, we have demonstrated that FTLE lends itself for interactive flow steering and has several advantages compared to traditional flow visualization techniques. Future systems could investigate what other methods of flow feature extraction, e.g. vortex extraction, are well suited for interactive steering.

Acknowledgements

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart and within the Transregional Collaborative Research Center SFB-TRR 75.

References

[1] G. Haller, "Distinguished Material Surfaces and Coherent Structures in Three-Dimensional Fluid Flows", *Phys. D*, 149: 248–277, March 2001.

[2] J. Helman, L. Hesselink, "Representation and Display of Vector Field Topology in Fluid Flow Data Sets", *Computer*, 22(8): 27–36, 1989.

[3] T. McLouglin, R.S. Laramee, R. Peikert, F.H. Post, M. Chen, "Over Two Decades of Integration-Based Geometric Flow Visualization", *Computer Graphics Forum*, 29(6): 1807–1829, 2010.

[4] K. Bürger, J. Schneider, P. Kondratieva, J. Krüger, R. Westermann, "Interactive Visual Exploration of Unsteady 3D Flows", in *Proc. Eurographics/IEEE VGTC Symposium on Visualization (EuroVis)*, pages 251–258. Eurographics Association, Norrköping, Sweden, 2007.

[5] J. Krüger, P. Kipfer, P. Kondratieva, R. Westermann, "A Particle System for Interactive Visualization of 3D Flows", *IEEE Transactions on Visualization and Computer Graphics*, 11: 744–756, 2005.

[6] URL `http://www.vistrails.org`, VisTrails: A Scientific Workflow Management System, Scientific Computing and Imaging Institute (SCI).

[7] R. van Liere, J.D. Mulder, J.J. van Wijk, "Computational Steering", *Future Gener. Comput. Syst.*, 12: 441–450, April 1997.

[8] J.D. Mulder, J.J. van Wijk, R. van Liere, "A Survey of Computational Steering Environments", *Future Generation Computer Systems*, 15: 119–129, February 1999.

[9] URL `http://www.scirun.org`, SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI).

[10] P. Hardt, S. Kühner, M. Krafczyk, E. Rank, "Computational Steering of a Lattice-Boltzmann based CFD-Solver in Virtual Reality", in *Proc. Conference on Construction Applications of Virtual Reality*. Virginia, USA, 2003.

[11] G. Benettin, L. Galgani, A. Giorgilli, J.M. Strelcyn, "Lyapunov Characteristic Exponent for Smooth Dynamical Systems and Hamiltonian Systems; a Method for Computing All of Them", *Mechanica*, 15: 9–20, 1980.

[12] J. Kasten, C. Petz, I. Hotz, B. Noack, H.C. Hege, "Localized Finite-time Lyapunov Exponent for Unsteady Flow Analysis", in M. Magnor, B. Rosenhahn, H. Theisel (Editors), *Vision Modeling and Visualization*, Volume 1, pages 265–274. Universität Magdeburg, Inst. f. Simulation u. Graph., 2009.

[13] C. Coulliette, F. Lekien, J. Paduano, G. Haller, J.E. Marsden, "Optimal Pollution Mitigation in Monterey Bay Based on Coastal Radar Data and Nonlinear Dynamics", *Enviromental Science and Technology*, 41: 6562–6572, August 2007.

[14] M. Mathur, G. Haller, T. Peacock, J.E. Ruppert-Felsot, H.L. Swinney, "Uncovering the Lagrangian Skeleton of Turbulence", *Phys. Rev. Lett.*, 98(14): 144502, April 2007.

[15] G. Haller, "Finding Finite-Time Invariant Manifolds in Two-Dimensional Velocity Fields", *Chaos*, 10(1): 99–108, 2000.

[16] F. Sadlo, D. Weiskopf, "Time-Dependent 2-D Vector Field Topology: An Approach Inspired by Lagrangian Coherent Structures", *Computer Graphics Forum*, 29(1): 88–100, 2010.

[17] J.D. Anderson, *Computational Fluid Dynamics: The Basics with Applications*, McGraw-Hill, New York, NY, USA, 1995.

[18] J. Stam, "Stable Fluids", in *Proc. 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[19] A. Selle, R. Fedkiw, B. Kim, Y. Liu, J. Rossignac, "An Unconditionally Stable MacCormack Method", *J. Sci. Comput.*, 35: 350–371, June 2008.

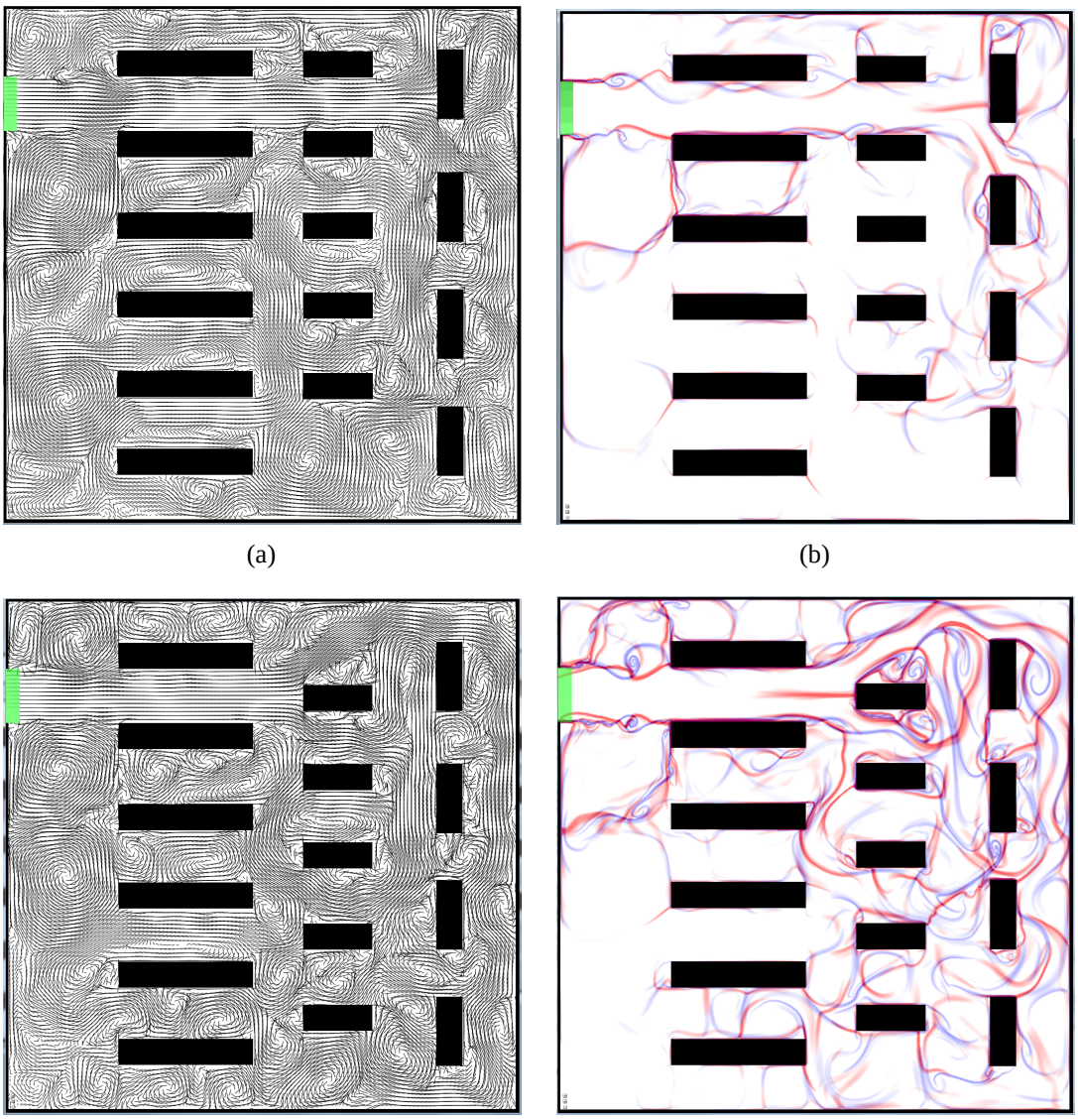
[20] Y. Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[21] S.C. Shadden, F. Lekien, J.E. Marsden, "Definition and Properties of Lagrangian Coherent Structures from Finite-Time Lyapunov Exponents in Two-Dimensional Aperiodic Flows", *Physica D: Nonlinear Phenomena*, 212(3-4): 271–304, 2005.

[22] D. Eberly, *Ridges in Image and Data Analysis*, Computational Imaging and Vision. Kluwer Academic Publishers, 1996.

[23] M. Ament, G. Knittel, D. Weiskopf, W. Strasser, "A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform", in *Proc. 18th Euromicro Conference on Parallel, Distributed, and Network-based Processing*, pages 583–592. IEEE Computer Society, Los Alamitos, CA, USA, 2010.

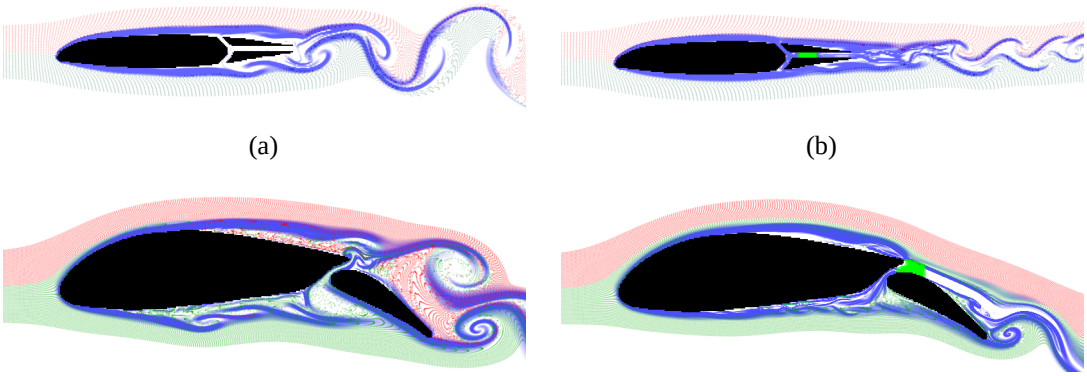
[24] M. Gad-el Hak, *Flow Control: Passive, Active, and Reactive Flow Management*, Cambridge University Press, Cambridge, United Kingdom, 2000.



(a) (b)

(c) (d)

Figure 10: Air conditioning of a compute cluster: aligned arrangement (top row) versus optimized arrangement (bottom row). Air flows in from the left (green) into a closed room. (a) Arrow glyphs provide limited insight into flow. (b) Forward (red) and backward (blue) FTLE visualize comparably simple LCS. (c) Optimized arrangement exhibits slightly more vortices. Overall mixing behavior is still hard to judge. (d) FTLE exhibits complex Lagrangian skeleton of mixing (many intersections of attracting and repelling LCS), indicating increased mixing.



(a) (b)

(c) (d)

Figure 11: Airfoils with active flow control. Visualization with backward FTLE (blue) and two particle tracers (red, dark green) around an airfoil profile. (a),(c) Uncontrolled flow generates high wake turbulence. (b),(d) A velocity actuator (light green) attaches the flow; the resulting wake turbulence is diminished significantly.