Vision, Modeling, and Visualization (2023)
T. Grosch and M. Guthe (Eds.)

Out-of-Core Particle Tracing for Monte Carlo Rendering of Finite-Time Lyapunov Exponents

Nicholas Grätz and Tobias Günther

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

The motion in time-dependent fluid flows is governed by Lagrangian coherent structures (LCS). One common approach to visualize hyperbolic LCS is to extract and visualize the finite-time Lyapunov exponent. Its visualization on large time-dependent fluid flow is challenging for two reasons. First, the time steps needed for particle tracing do not necessarily fit at once into memory. And second, conventional ray marching exhibits artifacts when the FTLE ridges are sharp, which instead requires Monte Carlo volume rendering techniques to produce unbiased results. So far, these two problems have only been looked at in isolation. In this paper, we implement the first out-of-core Monte Carlo FTLE tracer, which is able to visualize the finite-time Lyapunov exponent field of time-dependent fluid flows that do not fit into main memory at once. To achieve this, we designed a data processing pipeline that alternates between two phases: a photon tracing phase and a particle tracing phase. We demonstrate and evaluate the approach on several large time-dependent vector fields.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Computing methodologies*** → ***Ray tracing;***

1. Introduction

The transport analysis of time-dependent fluid flows is among the greatest open challenges in flow visualization [BYH*20]. One approach to address this problem is to extract and visualize hyperbolic Lagrangian Coherent Structures (LCS) [Hal15], which govern the temporal evolution of the fluid flow, since they act as attracting and repelling transport barriers. In flow visualization, the finite-time Lyapunov exponent (FTLE) is a widely-used scalar measure that exhibits ridges along hyperbolic transport barriers. In the light of ever-increasing data resolutions, the extraction of FTLE is challenging for two reasons. First, the FTLE measure is fundamentally based on the observation of particles over time, which has an unbounded memory requirement when all time steps need to fit into main memory at once, making it necessary to design distributed or out-of-core particle tracing systems [CS13]. Second, the FTLE measure can exhibit fine ridge structures that are much smaller than the flow data resolution, which requires Monte Carlo volume visualization in order to reach groundtruth level quality without grid discretization artifacts [GKT16,BRGG20]. The two aforementioned problems have been addressed in isolation. In this paper, we implement the first out-of-core Monte Carlo FTLE tracer, making it for the first time possible to calculate Monte Carlo volume visualizations for time-dependent flows that do not fit into memory. For this, we design a data processing pipeline that alternates between two phases: a photon tracing phase and a particle tracing phase. The photon tracing phase either uses Delta tracking [WMHL65] (on view rays) or a transmittance estimator [Spa66,NSJ14] (on light rays) to advance the photons one step forward. The decision on how to proceed with a photon depends on the FTLE value, which has to be calculated for every single sample point during photon tracing. Thus, the second phase traces particles in a fluid flow, which is streamed to the GPU via a ring buffer. To hide the memory IO latency of the fluid flow streaming, we launch more photons to occupy the GPU, minimizing unnecessary idle time. We investigate the impact of several design decisions, including the utilization of pre-fetching and the utilization of NVMe direct storage to lower the loading times. In summary, we contribute:

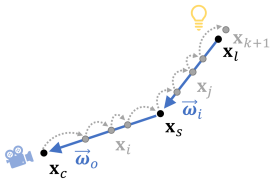
- An interleaved photon and particle tracing scheme that parallelizes over pixels and particles, and operates on streamed fluid flow data with bounded memory consumption.
- An evaluation of the benefits of pre-fetching, NVMe direct storage, and ray batching in the context of Monte Carlo FTLE, which obtains a speed-up of up to factor 23.

We evaluate the GPU utilization of the approach on three time-dependent vector fields, and present visual results and timing measurements for all test scenes.

2. Background

In the following, we introduce the fundamental basics for the two main ingredients of our out-of-core Monte Carlo FTLE renderer. The subsequent section covers related work in these two areas.

DOI: 10.2312/vmv.20231241



$\mathbf{x}_c$	camera position
$\mathbf{x}_i$	virtual event on view ray
$\mathbf{x}_s$	scattering location
$\mathbf{x}_j$	virtual event on light ray
$\mathbf{x}_{k+1}$	real event on light ray
$\mathbf{x}_l$	light position
$\vec{\omega}_o$	view ray direction
$\vec{\omega}_i$	light ray direction

Figure 1: *Overview of notations during single-scattering.*

2.1. Particle Transport in Fluid Flow

In this work, we are concerned with the visualization of a three-dimensional unsteady vector field, formally denoted as $\mathbf{v}(\mathbf{x},t) : \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}^3$, where $\mathbf{x}$ is a spatial position, t is a point in time, and $\mathbf{v}(\mathbf{x},t)$ is the velocity at position $\mathbf{x}$ at time t, which describes the fluid flow of air or liquid at a point in the space-time domain.

Particle Tracing. In a vector field $\mathbf{v}(\mathbf{x},t)$, the trajectory of a massless particle is governed by the first-order differential equation:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(\mathbf{x}(t),t). \tag{1}$$

For an initial position $\mathbf{x}(t_0) = \mathbf{x}_0$ and start time t_0, the location reached after integration duration τ is given by the flow map $\phi_{t_0}^{\tau}(\mathbf{x})$:

$$\phi_{t_0}^{\tau}(\mathbf{x}) = \mathbf{x}_0 + \int_{t_0}^{t_0+\tau} \mathbf{v}(\mathbf{x}(t),t)\,\mathrm{d}t \tag{2}$$

Note that the tracing process requires access to all vector field time steps in the considered time range $[t_0, t_0+\tau]$.

Finite-Time Lyapunov Exponents. The finite-time Lyapunov exponent (FTLE) [Hal02] is a scalar field that measures the maximum rate of separation between particles released close to each other from a specific point $\mathbf{x}$ at time t after integration duration τ. It can be computed in forward and backward time, where ridges in the forward FTLE field represent repelling LCS, while backward FTLE reveals attracting LCS. FTLE estimates the separation by computing the largest eigenvalue λ_{max} of the right Cauchy-Green deformation tensor, which is mapped logarithmically, cf. [Hal15]:

$$\mathrm{FTLE}(\mathbf{x},t,\tau) = \frac{1}{|\tau|} \ln\left(\sqrt{\lambda_{max}(\nabla\phi_{t_0}^{\tau}(\mathbf{x})^{\mathrm{T}}\nabla\phi_{t_0}^{\tau}(\mathbf{x}))}\right) \tag{3}$$

For notational convenience, we later drop the dependence on start time t and duration τ and abbreviate it as $f(\mathbf{x}) := \mathrm{FTLE}(\mathbf{x},t,\tau)$.

2.2. Monte Carlo Volume Rendering

Volumetric Radiance Integral. In volume rendering, the radiance at a point $\mathbf{x}_c$, arriving from direction $\vec{\omega}_o$ is calculated by moving the scattering point $\mathbf{x}_s$ along a ray $\mathbf{x}_s = \mathbf{x}_c - s\cdot\vec{\omega}_o$ for $s \in [0,d]$ and gathering the incoming radiance $L_i(\mathbf{x}_s \leftarrow \vec{\omega}_o)$ reflected towards $\vec{\omega}_o$.

$$L(\mathbf{x}_c \leftarrow \vec{\omega}_o) = \int_0^d \mu_s(\mathbf{x}_s)\cdot T(\mathbf{x}_c \leftarrow \mathbf{x}_s)\cdot L_i(\mathbf{x}_s \leftarrow \vec{\omega}_o)\,\mathrm{d}s \tag{4}$$

where μ_s denotes the probability of a scattering event occurring per unit distance and the transmittance $T(\mathbf{a} \leftarrow \mathbf{b})$ measures the fraction of light that went extinct along a ray segment from point $\mathbf{b}$ to point $\mathbf{a}$, i.e., $T(\mathbf{a} \leftarrow \mathbf{b}) = L(\mathbf{a})/L(\mathbf{b})$, which can be computed by integrating the extinction coefficient μ_t [GMH*19]:

$$T(\mathbf{a} \leftarrow \mathbf{b}) = \exp\left(-\int_0^1 \mu_t(s\cdot\mathbf{a} + (1-s)\cdot\mathbf{b})\,\mathrm{d}s\right) \tag{5}$$

In a single-scattering participating medium, the incoming radiance L_i from a light source with radiance L_e being placed at position $\mathbf{x}_l$, coming in from direction $\vec{\omega}_i$ at $\mathbf{x}_s$ is [KF12]:

$$L_i(\mathbf{x}_s \leftarrow \vec{\omega}_o) = \rho(\mathbf{x}_s, \vec{\omega}_o \leftarrow \vec{\omega}_i)\cdot T(\mathbf{x}_s \leftarrow \mathbf{x}_l)\cdot L_e \tag{6}$$

where $\rho(\mathbf{x}_s, \vec{\omega}_o \leftarrow \vec{\omega}_i)$ is the phase function, which models the fraction of light at $\mathbf{x}_s$ being reflected from direction $\vec{\omega}_i$ in direction $\vec{\omega}_o$.

The integral in Eq. (4) can be solved by Monte Carlo integration by sampling the scattering location $\mathbf{x}_s$ on the view ray N times according to the probability distribution $p(\mathbf{x}_s)$ as follows [GKT16]:

$$L(\mathbf{x}_c \leftarrow \vec{\omega}_o) \approx \frac{1}{N}\sum_{n=1}^{N} \frac{\mu_s(\mathbf{x}_s)\cdot T(\mathbf{x}_c \leftarrow \mathbf{x}_s)\cdot \rho(\mathbf{x}_s, \vec{\omega}_o \leftarrow \vec{\omega}_i)\cdot T(\mathbf{x}_s \leftarrow \mathbf{x}_l)\cdot L_e}{p(\mathbf{x}_s)} \tag{7}$$

Evaluating this sum requires two ingredients: the sampling of scattering points $\mathbf{x}_s$ and the estimation of transmittance $T(\mathbf{x}_s \leftarrow \mathbf{x}_l)$, which are explained in the following. Fig. 1 provides an overview of positional and directional variables used throughout the paper.

Free-flight Sampling. The process of placing scattering locations $\mathbf{x}_s$ according to the probability distribution $p(\mathbf{x}_s) = \mu_t(\mathbf{x}_s)\cdot T(\mathbf{x}_c \leftarrow \mathbf{x}_s)$ is called free-flight sampling. One approach is Delta tracking [WMHL65]. The algorithm adds a fictitious spatially-varying medium $\mu_n(\mathbf{x})$ with perfect forward scattering to the spatially-varying extinction $\mu_t(\mathbf{x})$ to reach a spatially-constant majorant extinction $\bar{\mu}_t(\mathbf{x}) = \mu_t(\mathbf{x}) + \mu_n(\mathbf{x})$. In the spatially-constant medium, the free-flight distance $d_i - d_{i-1}$ is sampled analytically

$$d_i = d_{i-1} - \frac{\ln(1-\eta_i)}{\bar{\mu}_t}, \qquad \mathbf{x}_i = \mathbf{x}_c - d_i\cdot\vec{\omega}_o \tag{8}$$

which is repeated until a real scattering event occurred, which has the probability $\mu_t(\mathbf{x}_i)/\bar{\mu}_t(\mathbf{x}_i)$. Thus, the scattering location $\mathbf{x}_s$ is:

$$\mathbf{x}_s = \mathbf{x}_c - d_{\hat{i}}\cdot\vec{\omega}_o \quad \text{with} \quad \hat{i} = \arg\min_i \frac{\mu_t(\mathbf{x}_i)}{\bar{\mu}_t} \geq \zeta_i \tag{9}$$

Thereby, both η_i and ζ_i draw uniformly distributed random numbers in $[0,1)$ and the initial distance is $d_0 = 0$.

Transmittance Estimation. Many transmittance estimators

$$T(\mathbf{x}_s \leftarrow \mathbf{x}_l) \approx \frac{1}{N}\sum_{n=1}^{N} \widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) \tag{10}$$

have been explored by the graphics community [GMH*19]. The simplest unbiased estimator is the *track length* estimator [Spa66, SG08], which sends a photon from $\mathbf{x}_s$ to $\mathbf{x}_l$:

$$\widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) = \begin{cases} 1 & \text{if } \|\mathbf{x}_l - \mathbf{x}_s\| \leq \|\mathbf{x}_{k+1} - \mathbf{x}_s\| \\ 0 & \text{else} \end{cases} \tag{11}$$

where $\mathbf{x}_{k+1}$ is the first real event after k virtual scattering events. The track length estimator assigns a score of 1 when a particle reaches $\mathbf{x}_l$ and assigns a score of 0 otherwise, which estimates the fraction of photons that travel from $\mathbf{x}_s$ to $\mathbf{x}_l$ without a real event.

Novák et al. [NSJ14] presented ratio tracking, which is unbiased and estimates the joint probability that all particle interactions on the path towards a light source (from $\mathbf{x}_s$ to $\mathbf{x}_l$) have been virtual:

$$\widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) = \prod_{j=1}^{k} \left(1 - \frac{\mu_t(\mathbf{x}_j)}{\overline{\mu_t}}\right) \tag{12}$$

where k is the index of the last event before reaching the light source, i.e., $d_k \leq \|\mathbf{x}_s - \mathbf{x}_l\| \leq d_{k+1}$. Ratio tracking has a higher per-iteration cost than the track length estimator, since rays are fully traversed, but still outperforms it in most cases. Novák et al. [NSJ14] suggested a combination of the two methods, by using ratio tracking along the ray until the transmittance $\widehat{T}$ falls under a certain threshold (0.1%). The transmittance along the remainder of the ray is then estimated using the track length estimator. In our work, we use Delta tracking on view rays and ratio tracking with track length estimation for negligible transmittance on the light ray.

3. Related Work

3.1. Particle Tracing

Different strategies have been investigated to reduce the memory I/O overhead and to maximize parallelism during particle advection [ZY18]. For example, Chen and Shen [CS13] dynamically loaded data bricks during flow map advection. Nouanesengsy et al. [NLL*12] parallelized the particle advection not only in space but also in time, when tracing FTLE for multiple time spans. For unstructured grids, efficient cell locators have been proposed to interpolate velocities [GJ10, BRKE*11]. Childs et al. [CPA*10] employed and evaluated data pre-fetching to hide the data loading latency. Lu et al. [LSP14] utilized caching and Camp et al. [CCC*11] suggested to use memory hierarchies. To improve scheduling, work stealing [MCHG13] and dynamic load balancing strategies have been investigated [ZGH*18]. Lifeline based scheduling has recently been demonstrated to be a successful option for distributed sytems [BPNC19, BPC*21]. A commonly cited application of parallel particle tracing is the computation of FTLE. Unlike the related work above, we do not discretize FTLE onto a grid, but perform the parallel particle tracing in a Monte Carlo rendering loop.

3.2. FTLE Computation

The finite-time Lyapunov exponent in Eq. (3) can be calculated numerically in different ways. The most common approach is using finite differences [Hal02]. Rather than releasing nearby particles from regular grid points, local linearization of the flow map via matrix exponentiation of the Jacobian leads to localized FTLE [KPH*09]. The different approaches have been compared by Kuhn et al. [KRWT12] in a benchmark. Several approximation strategies have been proposed, including Catmull-Rom interpolation [GGTH07], adaptive mesh refinement [SP07], reconstruction from sparse samples [BT13], interpolation of flow maps [COJ15], space-time construction of FTLE ridges [WRT18], neural upsampling [JGG21, SLB22], and the neural representation of flow maps [XLT23]. For interactive FTLE visualization, Barakat et al. [BGT12] interleaved the direct volume rendering with view-dependent particle tracing. Conti et al. [CRK12] investigated FTLE computation on many-core architectures. FTLE computation on discrete grids nevertheless entails discretizations and a direct volume rendering based on ray marching is biased [GKT16].

3.3. Monte Carlo FTLE

Our approach is based on the unbiased Monte Carlo FTLE rendering approach by Günther et al. [GKT16]. They estimated the FTLE value by releasing particles from a numerical epsilon neighborhood and mapped the FTLE scalar field via transfer functions to the extinction $\mu_t(\mathbf{x}_s) := \mu_t(f(\mathbf{x}_s))$ and to color $\mathbf{c}(\mathbf{x}_s) := \mathbf{c}(f(\mathbf{x}_s))$, where the color is the ratio between scattering and extinction coefficient, i.e., $\mathbf{c}(\mathbf{x}_s) = \mu_s(\mathbf{x}_s)/\mu_t(\mathbf{x}_s)$. Inserting the free-flight sampling probability into Eq. (7) leads to the simple expression:

$$L(\mathbf{x}_c \leftarrow \vec{\omega}_o) \approx \frac{1}{N} \sum_{n=1}^{N} \mathbf{c}(\mathbf{x}_s) \cdot \rho \cdot \widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) \cdot L_e \tag{13}$$

They used an isotropic phase function $\rho = \frac{1}{4\pi}$ and the light transmittance $\widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l)$ was estimated with the track length estimator in Eq. (11). The transmittance estimation in Eq. (10) is combined with the outer Monte Carlo sum, i.e., one transmittance sample is taken per iteration. Baeza Rojo et al. [BRGG20] replaced the track length estimator with the ratio tracking estimator in Eq. (12) and introduced approximations and gradient domain denoising. Note that both approaches assume that the time-dependent vector field fits into GPU memory, i.e., one iteration of Eq. (13) was evaluated in a single compute shader. Each thread of the compute shader iterates the full view ray, the light rays, and traces particles in the fluid flow for each extinction sample along the way. When lifting the assumption that all time steps must fit into memory, the evaluation of Eq. (13) has to be split into multiple kernels. The demonstration of how this can be achieved is the key contribution of our work.

4. Method

The goal of this paper is to extend the unbiased Monte Carlo FTLE approach by Günther et al. [GKT16] to time-dependent vector fields that do not fit at once into GPU memory. With our approach, Monte Carlo FTLE visualizations can be computed for arbitrary temporal domains within a bounded memory consumption.

4.1. Problem Statement

For each pixel, our method calculates the incoming radiance $L(\mathbf{x}_c \leftarrow \vec{\omega}_o)$ at $\mathbf{x}_c$ from direction ω_o in Eq. (13) using Delta tracking on the view ray as in Eq. (9). Further, we follow Baeza Rojo et al. [BRGG20] and use ratio tracking on the light ray as in Eq. (12):

$$L(\mathbf{x}_c \leftarrow \vec{\omega}_o) \approx \frac{1}{N} \sum_{n=1}^{N} \mathbf{c}(\mathbf{x}_s) \cdot \rho \cdot \underbrace{\prod_{j=1}^{k} \left(1 - \frac{\mu_t(\mathbf{x}_j)}{\overline{\mu_t}}\right)}_{\widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l)} \cdot L_e \tag{14}$$

What makes Monte Carlo FTLE challenging is the dependence of both the color $\mathbf{c}(\mathbf{x}) := \mathbf{c}(f(\mathbf{x}))$ and the extinction $\mu_t(\mathbf{x}) := \mu_t(f(\mathbf{x}))$ on the FTLE value $f(\mathbf{x})$. Thus, the evaluation of every $\mu_t(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ requires the numerical integration of six particle trajectories using Eq. (2) through the entire time-dependent vector field in order to calculate FTLE using Eq. (3). If the vector field does not fit into memory, a streaming approach is needed, as described next.

Algorithm 1: LOOP

Data: iterations M, time steps T
Result: Radiance estimate for pixel L

```
photon, L ← INITIALIZE() ;
for m ← 1 to M do
    particles ← SEED(photon) ;
    for t ← 1 to T do
        particles ← ADVECT(particles) ;
    end
    photon, L ← UPDATE(L, photon, particles) ;
end
```

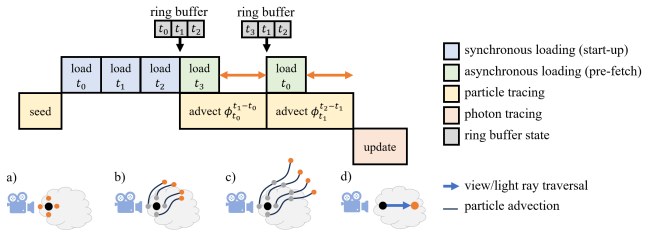


Figure 2: *Pipeline illustration. In the* particle tracing*, particles are seeded around the photons (a), all particles are iteratively advected in parallel (b and c), FTLE is estimated based on the separation and is mapped to extinction and color for the* photon update *(d).*

4.2. Overview

Update Loop. Our approach alternates between two phases, namely *particle tracing* and *photon tracing*, and persists the states of photons and particles in memory when switching between the phases. The outer-most rendering loop is listed in Alg. 1 and an overview is given in Fig. 2. When launching the program, the **photon** state is initialized by generating view rays that originate from the camera sensor, i.e., from the pixel (INITIALIZE). The rendering loop then proceeds M times by first placing **particles** at the current positions of photons (SEED in Fig. 2 a), iteratively tracing those particles out-of-core by loading the vector field data into a ring buffer (ADVECT in Fig. 2 b and c), and eventually calculating FTLE, mapping it to color and extinction before updating the state of photons (UPDATE in Fig. 2 d). During the update phase, we distinguish between photons that are currently traveling on the view ray from those currently traveling on light rays. In the background, the flow data is asynchronously streamed into a GPU ring buffer.

Design Decisions. Each photon may require a different number of free-flight steps before it scatters or reaches the light source. Thus, when parallelizing across the pixels, photons may be in different stages of the ray tracing loop, as they individually generate a new ray when they are finished. During particle tracing, however, all particles exist in the same time step to reduce the number of vector field time steps that need to be in memory at once. For each pixel on the screen, the final output radiance L, the photon state, and six particles are stored for computing the finite differences of the flow map, which is listed in Alg. 2. The photon state contains the ray origin and direction, the current free-flight distance, the joint probability of the light ray transmittance estimation, the FTLE value at the scattering location, and the ray type (view ray or light ray). In the following, we elaborate on the individual steps in Alg. 1.

Algorithm 2: State of a photon and a particle.

```
struct photon {
    float origin[3] ;          // ray origin x_c or x_s
    float direction[3] ;       // ray dir ω_o or ω_i
    float dist ;               // distance d_i or d_j
    float joint_prob ;         // product T̂(x_s ← x_l)
    float scatter_ftle ;       // FTLE f(x_s) at x_s
    int ray_type ;             // 0=VIEW, 1=LIGHT
};
struct particle {
    float position[3] ;        // spatial position x
};
```

4.3. Initialization

For initialization in Alg. 3, a new photon is randomly placed on the sensor pixel at $\mathbf{x}_c$ and a view ray direction $\vec{\omega}_o$ from $\mathbf{x}_c$ to the camera's eye position is calculated. The view ray is oriented towards the camera's eye, since it models the transport of radiance towards the viewer. All other variables are reset to zero, apart from the joint probability, which later decays along the light ray from 1. The ray type is a VIEW ray. The initial radiance is set to $L = 0$.

Algorithm 3: INITIALIZE

Result: photon state, radiance L

```
photon.origin ← x_c ;
photon.direction ← ω_o ;
photon.dist ← 0 ;
photon.joint_prob ← 1 ;
photon.scatter_ftle ← 0 ;
photon.ray_type ← VIEW ;
L ← 0 ;
```

4.4. Seeding

In order to advance a view ray or light ray, the exinction coefficient $\mu_t(f(\mathbf{x}))$ is required, which requires the FTLE value. Thus, in a first step, six particles are seeded in the immediate vicinity of the photon position $\mathbf{x}$ with separation distance $\varepsilon = 10^{-6}$, see Alg. 4.

$$\mathbf{x}^{(0)} = \mathbf{x} + (\varepsilon,0,0)^T, \quad \mathbf{x}^{(2)} = \mathbf{x} + (0,\varepsilon,0)^T, \quad \mathbf{x}^{(4)} = \mathbf{x} + (0,0,\varepsilon)^T,$$
$$\mathbf{x}^{(1)} = \mathbf{x} - (\varepsilon,0,0)^T, \quad \mathbf{x}^{(3)} = \mathbf{x} - (0,\varepsilon,0)^T, \quad \mathbf{x}^{(5)} = \mathbf{x} - (0,0,\varepsilon)^T$$

Algorithm 4: SEED

Data: photon state, sepration distance ε
Result: particles particle[6]

```
x ← photon.position + photon.dist · photon.direction ;
particle[0].position ← x + (ε,0,0);
...
particle[5].position ← x − (0,0,ε);
```

4.5. Advection

Next, all particles are advected from time t_0 to $t_0 + \tau$. Since not all time steps fit into GPU memory, we instead maintain a ring buffer. We load three consecutive time steps into memory $(\mathbf{v}_i, t_i)$, $(\mathbf{v}_{i+1}, t_{i+1})$, $(\mathbf{v}_{i+2}, t_{i+2})$ and advect all particles using fourth-order

Runge-Kutta integration (RK4) from time step t_i to t_{i+1} using Eq. (2), which is listed in Alg. 5. The third time step is required, since the RK4 integrator may access the time range $[t_{i+1}, t_{i+2}]$. Once the time step t_{i+1} is reached, time step t_i is freed and the next time step t_{i+3} is copied into the ring buffer into its position. The advection continues until the end time $t_0 + \tau$ is reached for all particles. The advection is parallelized over particles.

The subsequent TRACING process has to wait while data is loading from disc, which may cause a stall. Since the order of time steps is deterministic, we pre-fetch the time step that will be requested next into an intermediate GPU buffer while the tracing process is still under way. The data loading was implemented using the Microsoft Direct Storage API (version 1.0.2 with a 4 MB block request size). This API allows to directly load files from disc into GPU memory and is optimized for reading data from NVMe SSD drives. In Section 5, we compare Direct Storage with the conventional loading process of first reading a file into CPU memory and subsequently uploading it to the GPU.

Algorithm 5: ADVECT

Data: particle, time steps $(\mathbf{v}_i, t_i)$, $(\mathbf{v}_{i+1}, t_{i+1})$, $(\mathbf{v}_{i+2}, t_{i+2})$
Result: particle

1 particle.position $\leftarrow \phi_{t_i}^{t_{i+1}-t_i}$(particle.position) ;

4.6. Photon Update

Once the particle advection finished, photons can be updated. The update procedure depends on whether the ray is a view ray or light ray, and on whether the particle exited the domain or not. A listing is provided in Alg. 6 and the cases are explained in the following.

4.6.1. Ray has not Terminated

If the ray has not terminated yet, i.e., it has neither left the domain nor has the light source been reached on a light ray, we can first load the six advected particles, estimate the flow map gradient via finite differences and compute the finite-time Lyapunov exponent $f(\mathbf{x})$ using Eq. (3). The FTLE value is then mapped via transfer function to an extinction value $\mu_t(\mathbf{x})$. Russian roulette then decides whether a virtual or real scattering event was found.

View Ray. If a real scattering event was found on the view ray, then the photon remembers the FTLE value at the scattering point, since this information will be needed later after the light ray has finished in order to map the FTLE value at the scattering location to a color. In addition, a light ray is initialized to follow towards the light source, which means the ray type is set to LIGHT. The last line of the kernel advances the photon forward towards the light source using a Delta tracking step.

Light Ray. If a real scattering event was found on the light ray, then the joint probability of the virtual interactions is inspected. If the joint probability falls below a threshold, we follow Novák et al. [NSJ14] and switch to a track length estimator to terminate the ray early. Thus, a photon either stops its path when it sees a real interaction or it continues its path in case of a virtual interaction. If it stops, a new view ray is generated by sampling a new location on the sensor pixel and thus, the ray type switches to VIEW. Further, the counter of Monte Carlo radiance estimates is incremented,

Algorithm 6: UPDATE

Data: L, photon, particle[6], L_e, ρ, $\overline{\mu}_t$
Result: L, photon

1 **if** *ray not terminated* **then**
2 $\quad\mathbf{x} \leftarrow$ photon.position + photon.dist $\cdot$ photon.direction ;
3 $\quad\nabla\phi \leftarrow$ finite_difference(particle) ;
4 $\quad f(\mathbf{x}) \leftarrow \frac{1}{|\tau|} \ln\left(\sqrt{\lambda_{max}(\nabla\phi^{\mathrm{T}}\nabla\phi)}\right)$;
5 $\quad\mu_t(\mathbf{x}) \leftarrow \mu_t(f(\mathbf{x}))$;
6 $\quad$is_real $\leftarrow \mu_t(\mathbf{x})/\overline{\mu}_t \geq \zeta$;
7 $\quad$**if** *photon.ray_type = VIEW* $\wedge$ *is_real* **then**
8 $\quad\quad$photon.ray_type $\leftarrow$ LIGHT ;
9 $\quad\quad$photon.origin $\leftarrow \mathbf{x}$;
10 $\quad\quad$photon.direction $\leftarrow \vec{\omega}_i$;
11 $\quad\quad$photon.dist $\leftarrow 0$;
12 $\quad\quad$photon.scatter_ftle $\leftarrow f(\mathbf{x})$;
13 $\quad$**else if** *photon.ray_type = LIGHT* **then**
14 $\quad\quad$**if** *photon.joint_prob* $< 0.1\%$ **then**
15 $\quad\quad\quad$**if** *is_real* **then**
16 $\quad\quad\quad\quad$count $\leftarrow$ count $+1$;
17 $\quad\quad\quad\quad$photon.ray_type $\leftarrow$ VIEW ;
18 $\quad\quad\quad\quad$photon.origin $\leftarrow \mathbf{x}_c$;
19 $\quad\quad\quad\quad$photon.direction $\leftarrow \vec{\omega}_o$;
20 $\quad\quad\quad\quad$photon.dist $\leftarrow 0$;
21 $\quad\quad\quad$**end**
22 $\quad\quad$**else**
23 $\quad\quad\quad$photon.joint_prob $\leftarrow$ photon.joint_prob $\cdot(1-\mu_t(\mathbf{x})/\overline{\mu}_t)$;
24 $\quad\quad$**end**
25 $\quad$**end**
26 **else if** *ray terminated* **then**
27 $\quad$**if** *photon.ray_type = VIEW* **then**
28 $\quad\quad L \leftarrow L + L_{bg}$;
29 $\quad$**else if** *photon.ray_type = LIGHT* **then**
30 $\quad\quad\widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) \leftarrow$ photon.joint_prob ;
31 $\quad\quad\mathbf{c}(\mathbf{x}_s) \leftarrow \mathbf{c}$(photon.scatter_ftle) ;
32 $\quad\quad L \leftarrow L + \mathbf{c}(\mathbf{x}_s) \cdot \rho \cdot \widehat{T}(\mathbf{x}_s \leftarrow \mathbf{x}_l) \cdot L_e$;
33 $\quad$**end**
34 $\quad$count $\leftarrow$ count $+1$;
35 $\quad$photon.ray_type $\leftarrow$ VIEW ;
36 $\quad$photon.origin $\leftarrow \mathbf{x}_c$;
37 $\quad$photon.direction $\leftarrow \vec{\omega}_o$;
38 $\quad$photon.dist $\leftarrow 0$;
39 **end**
40 photon.dist $\leftarrow$ photon.dist $- \frac{\ln(1-\eta_i)}{\overline{\mu}_t}$;

which is stored in the alpha channel of radiance L. If the joint probability is not below the threshold, then the probability of a virtual interaction is multiplied to the joint probability, following Eq. (12).

4.6.2. Ray has Terminated

View Ray. If the ray terminated on the view ray, i.e., when the photon left the domain of the vector field, then the background color L_{bg} is added to the radiance estimate. Note that the Monte Carlo iteration counter is incremented, as well. Afterwards, a new view ray is generated and the first Delta tracking step is executed.

Light Ray. Ray termination on the light ray means that the photon reached the light source, meaning that we can calculate the light path contribution using Eq. (14). Again, a new view ray is generated and the first Delta tracking step follows.

4.7. Ray Batching

The most time-consuming part of our Monte Carlo rendering pipeline is reading new files from disk, even with pre-fetching. By tracing multiple rays per pixel at once, we can mitigate this issue.

5. Results

Next, we evaluate the approach in three time-dependent data sets. The evaluation focuses on the performance improvements that arise from the improved concurrent data loading and batching. The visual results that are displayed next to the performance plots are denoised with Intel Open Image Denoise without auxiliary buffers.

Asteroid. The ASTEROID data set in Fig. 3 contains a numerical simulation of an asteroid with a diameter of 250 m that hits a deep water ocean at an impact angle of 45° with no air burst. The data set was subject of the IEEE Scientific Visualization Contest 2018 and was simulated using the multi-physics hydrodynamics solver xRage [G*08]. The considered time range spans 21 time steps with 1.43 GB each. Since only 3 time steps need to be read at once, the memory requirement is reduced to 14% of the full data.

Exajet. The EXAJET data set (exajet-d12) in Fig. 4 is a numerically-simulated fluid flow around a civilian transport aircraft. The flow was simulated by Dassault Systèmes and is hosted publicly on the Nasa portal. For a detailed description of the data set, we refer to Wald et al. [WZU*21]. The data set uses adaptive mesh refinement (AMR), which follows an octree-type of subdivision. For our system, we resampled the data to a regular grid on its finest resolution. In the considered time range, the flow consists of 93 time steps with 541 MB each. In this flow, the storage requirement reduces to 3.2% of the full data.

Half Cylinder The HALF CYLINDER flow in Fig. 5 contains a numerical simulation of a viscous 3D fluid flow around a half cylinder at a Reynolds number of $Re = 320$ in dimensionless units. The fluid flow was simulated with Gerris [Pop03] and was provided by Baeza Rojo et al. [BRG20]. The vector field contains a classic von-Kármán vortex street, in which FTLE separates the vortices. We traced particles over a time period with 21 time steps with 140 MB each. With this flow, we deliberately chose a smaller data set to see how the batching and DirectStorage impact a data set that is less bound by IO traffic. We can see that in combination with batching, DirectStorage was less helpful than pre-fetching. Although the data set would fit into memory in its entirety, the ring buffer reduced the storage requirements to 14% of the full time series.

To assess the impact of pre-fetching (P), DirectStorage (D), and ray batching (B), we performed an ablation study on all three data sets. The measurements were performed on a desktop machine with an AMD Ryzen 5 1600 Six-Core Processor (3.2GHz), with 16 GB RAM, 931 GB NVMe storage, and an NVIDIA GeForce RTX 2070 SUPER graphics card, and the measurements are averaged over multiple iterations (ASTEROID: 1,000, EXAJET: 200, HALF CYLINDER: 1,000). The scenes were rendered at a screen resolution of 704×840 (ASTEROID), 655×1022 (EXAJET), and 1450×705 (HALF CYLINDER), respectively. Of interest is the concurrency of the particle advection (PT) and the data loading (DL). The remaining processing steps, such as the photon update (UPDATE), take milliseconds and are negligible in comparison to the other two, which are measured in seconds.

In Figures 3, 4, 5, the performance measurements are listed for all cases, revealing the idle times that the optimizations aim to minimize. Since batching traces multiple iterations concurrently, we let the variants without batching trace the equivalent number of iterations sequentially in order to make the results comparable. The number of iterations to batch was chosen empirically by incrementally increasing the number of concurrent iterations until the particle tracing time exceeds the data loading time. The best throughput is obtained when both are equal. The performance plots show a doubled loading time when a new particle tracing loop begins. This is because the time-dependent vector fields are not periodic in time, and thus two time steps are read to refill the ring buffer for the next iteration. From the available 8 GB of video memory, 6 GB, 2.27 GB, and 0.59 GB, were used on our three data sets (ASTEROID, EXAJET, HALF CYLINDER), respectively. The photon buffer allocates 136 bytes per pixel (including random seeds, accumulated radiance, etc.), which results in 80.42 MB, 91.04 MB, and 139.03 MB, for the resolutions given above, respectively.

In all data sets, we observed that the Direct Storage API significantly helped to reduce loading times by 73.67% on the ASTEROID, 81.09% on the EXAJET, and 40.95% on the HALF CYLINDER. Additionally, we saw that the bottleneck of our rendering loop can generally be shifted to particle tracing by making use of pre-fetching and batching. Pre-fetching reduces the data loading by 7.14% on the ASTEROID, 25.26% on the EXAJET, and 12.69% on the HALF CYLINDER. Batching hides the IO latency, giving an overall improvement of 41.56% on the ASTEROID, 84.89% on the EXAJET, and 69.89% on the HALF CYLINDER. Throughout all examples, the best performance was reached when enabling all optimizations.

6. Conclusions

In this paper, we extended the unbiased Monte Carlo rendering of finite-time Lyapunov exponent fields [GKT16] to operate within fixed memory bounds by interleaving a photon and particle tracing phase. With the approach it becomes for the first time possible to calculate Monte Carlo FTLE visualizations for time-dependent data sets that do not fit into memory at once, such as the ASTEROID sequence and the EXAJET simulation. In addition, we evaluated the impact of pre-fetching, NVMe storage, and batching on the overall performance. An orthogonal direction for further improvements is the utilization of more recent tracking methods and transmittance estimators that reduce the variance of the Monte Carlo integration faster [GMH*19]. Further, it would be imaginable to extend the approach to support irregular grids. This would not only be useful to reduce the memory footprint of adaptively-refined data sets, it also opens the door to only streaming velocity fields for the spatial regions of the data set that are currently populated with particles.

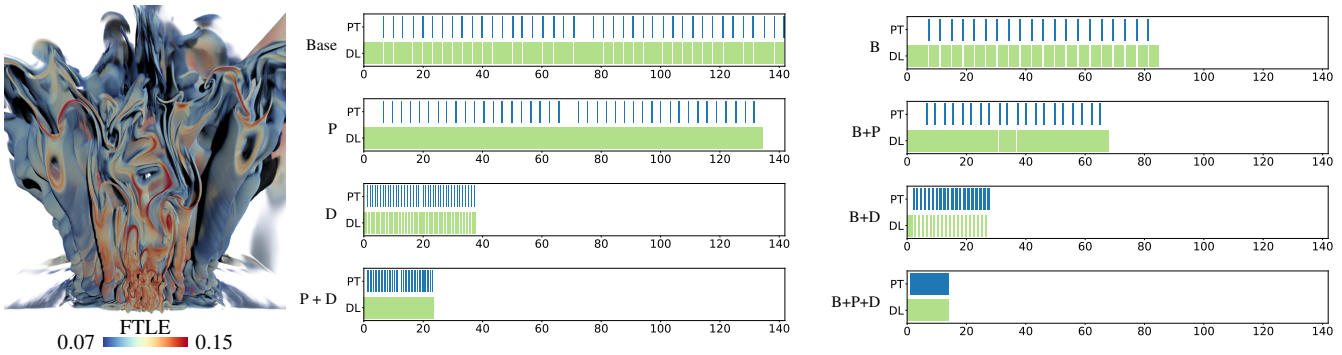


Figure 3: *Backward FTLE in the* ASTEROID *data set for start time $t_0 = 2.474$, integration duration $\tau = 1.484$ and the linear extinction transfer function bounds $[0.07, 0.09]$ with majorant extinction $\bar{\mu}_t = 10$ after 2,400* UPDATE *iterations (400 iterations with batch size 6). The plots show the detailed timeline (in seconds) of particle advection (PT) and data loading (DL) for the baseline method (Base), enabling pre-fetching (P), enabling DirectStorage (D), and enabling batching (B). For comparability with batching, all plots show the completion of two* UPDATE *iterations.*

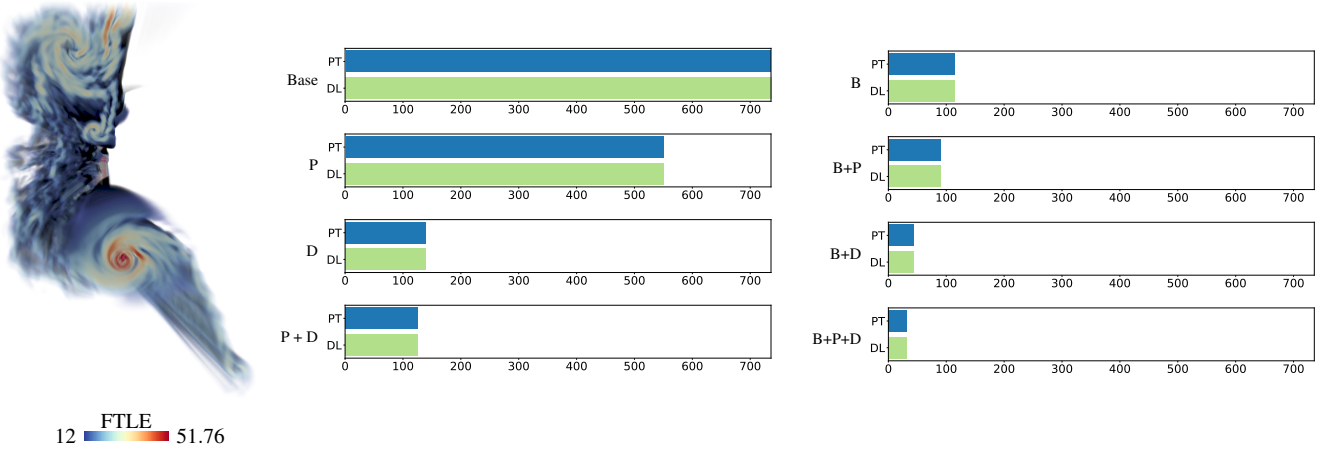


Figure 4: *Backward FTLE in the* EXAJET *data set for start time $t_0 = 0.211$, integration duration $\tau = 0.046$ and the linear extinction transfer function bounds $[10, 20]$ with majorant extinction $\bar{\mu}_t = 4$ after 18,000* UPDATE *iterations (3,000 iterations with batch size 6). The plots show the detailed timeline (in seconds) of particle advection (PT) and data loading (DL) for the baseline method (Base), enabling pre-fetching (P), enabling DirectStorage (D), and enabling batching (B). For comparability with batching, all plots show the completion of six* UPDATE *iterations.*

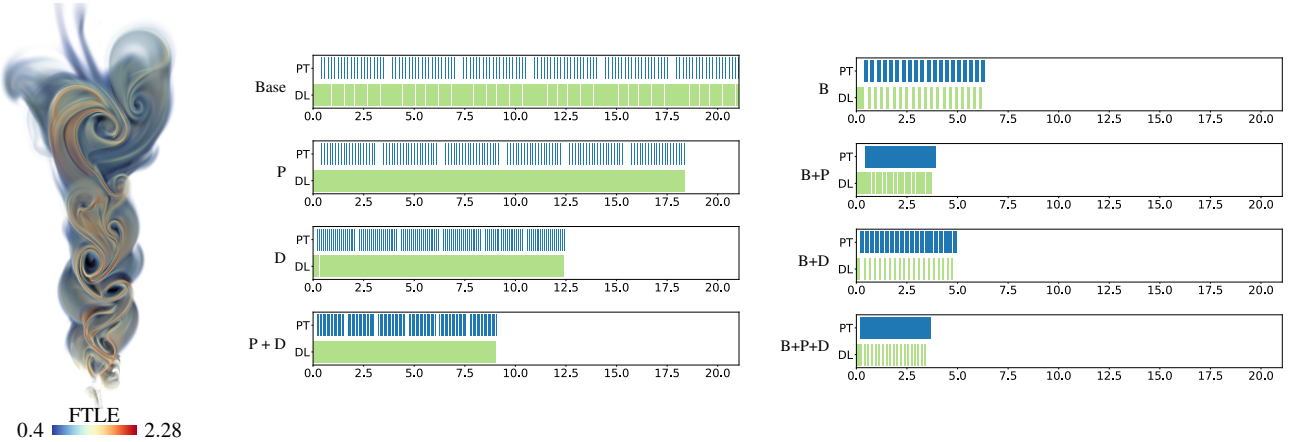


Figure 5: *Backward FTLE in the* HALF CYLINDER *data set for start time $t_0 = 10$, integration duration $\tau = 2$ and the linear extinction transfer function bounds $[0.36, 1.4]$ with majorant extinction $\bar{\mu}_t = 16$ after 12,000* UPDATE *iterations (2,000 iterations with batch size 6). The plots show the detailed timeline (in seconds) of particle advection (PT) and data loading (DL) for the baseline method (Base), enabling pre-fetching (P), enabling DirectStorage (D), and enabling batching (B). For comparability with batching, all plots show the completion of six* UPDATE *iterations.*

References

[BGT12] BARAKAT S., GARTH C., TRICOCHE X.: Interactive computation and rendering of finite-time Lyapunov exponent fields. *IEEE Trans. on Visualization and Computer Graphics 18*, 8 (2012), 1368–1380. 3

[BPC*21] BINYAHIB R., PUGMIRE D., CHILDS H., LARSEN M., SADLO F.: Hylipod: Parallel particle advection via a hybrid of lifeline scheduling and parallelization-over-data. In *Eurographics Symposium on Parallel Graphics and Visualization.* (2021). 3

[BPNC19] BINYAHIB R., PUGMIRE D., NORRIS B., CHILDS H.: A lifeline-based approach for work requesting and parallel particle advection. In *2019 IEEE 9th Symposium on Large Data Analysis and Visualization (LDAV)* (2019), IEEE, pp. 52–61. 3

[BRG20] BAEZA ROJO I., GÜNTHER T.: Vector field topology of time-dependent flows in a steady reference frame. *IEEE Transactions on Visualization and Computer Graphics 26*, 1 (2020), 280–290. 6

[BRGG20] BAEZA ROJO I., GROSS M., GÜNTHER T.: Accelerated Monte Carlo rendering of finite-time Lyapunov exponents. *IEEE Trans. on Visualization and Computer Graphics 26*, 1 (2020), 708–718. 1, 3

[BRKE*11] BUSSLER M., RICK T., KELLE-EMDEN A., HENTSCHEL B., KUHLEN T.: Interactive Particle Tracing in Time-Varying Tetrahedral Grids. In *EG Symposium on Parallel Graphics and Vis.* (2011). 3

[BT13] BARAKAT S. S., TRICOCHE X.: Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2753–2762. 3

[BYH*20] BUJACK R., YAN L., HOTZ I., GARTH C., WANG B.: State of the art in time-dependent flow topology: Interpreting physical meaningfulness through mathematical properties. *Computer Graphics Forum 39*, 3 (2020), 811–835. 1

[CCC*11] CAMP D., CHILDS H., CHOURASIA A., GARTH C., JOY K. I.: Evaluating the benefits of an extended memory hierarchy for parallel streamline algorithms. In *2011 IEEE Symposium on Large Data Analysis and Visualization* (2011), pp. 57–64. 3

[COJ15] CHANDLER J., OBERMAIER H., JOY K. I.: Interpolation-based pathline tracing in particle-based flow visualization. *IEEE Transactions on Visualization and Computer Graphics 21*, 1 (2015), 68–80. 3

[CPA*10] CHILDS H., PUGMIRE D., AHERN S., WHITLOCK B., HOWISON M., PRABHAT, WEBER G. H., BETHEL E. W.: Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications 30*, 3 (2010), 22–31. 3

[CRK12] CONTI C., ROSSINELLI D., KOUMOUTSAKOS P.: GPU and APU computations of finite time Lyapunov exponent fields. *Journal of Computational Physics 231*, 5 (2012), 2229–2244. 3

[CS13] CHEN C.-M., SHEN H.-W.: Graph-based seed scheduling for out-of-core FTLE and pathline computation. In *IEEE Symposium on Large-Scale Data Analysis and Visualization* (2013), pp. 15–23. 1, 3

[G*08] GITTINGS M., ET AL.: The RAGE radiation-hydrodynamic code. *Computational Science & Discovery 1*, 1 (2008), 015005. 6

[GGTH07] GARTH C., GERHARDT F., TRICOCHE X., HANS H.: Efficient computation and visualization of coherent structures in fluid flow applications. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1464–1471. 3

[GJ10] GARTH C., JOY K. I.: Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1541–1550. 3

[GKT16] GÜNTHER T., KUHN A., THEISEL H.: MCFTLE: Monte Carlo rendering of finite-time Lyapunov exponent fields. *Computer Graphics Forum 35*, 3 (2016), 381–390. 1, 2, 3, 6

[GMH*19] GEORGIEV I., MISSO Z., HACHISUKA T., NOWROUZEZAHRAI D., KŘIVÁNEK J., JAROSZ W.: Integral formulations of volumetric transmittance. *ACM Transactions on Graphics (TOG) 38*, 6 (2019), 1–17. 2, 6

[Hal02] HALLER G.: Lagrangian coherent structures from approximate velocity data. *Physics of fluids 14*, 6 (2002), 1851–1861. 2, 3

[Hal15] HALLER G.: Lagrangian coherent structures. *Annual Review of Fluid Mechanics 47*, 1 (2015), 137–162. 1, 2

[JGG21] JAKOB J., GROSS M., GÜNTHER T.: A fluid flow data set for machine learning and its application to neural flow map interpolation. *IEEE Trans. on Vis. and Comp. Graph. 27*, 2 (2021), 1279–1289. 3

[KF12] KULLA C., FAJARDO M.: Importance sampling techniques for path tracing in participating media. *Computer Graphics Forum 31*, 4 (2012), 1519–1528. 2

[KPH*09] KASTEN J., PETZ C., HOTZ I., NOACK B. R., HEGE H.-C.: Localized finite-time Lyapunov exponent for unsteady flow analysis. In *Vision Modeling and Visualization* (2009), pp. 265 – 274. 3

[KRWT12] KUHN A., RÖSSL C., WEINKAUF T., THEISEL H.: A benchmark for evaluating FTLE computations. In *2012 IEEE Pacific visualization symposium* (2012), IEEE, pp. 121–128. 3

[LSP14] LU K., SHEN H.-W., PETERKA T.: Scalable computation of stream surfaces on large scale vector fields. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2014), pp. 1008–1019. 3

[MCHG13] MÜLLER C., CAMP D., HENTSCHEL B., GARTH C.: Distributed parallel particle advection using work requesting. In *IEEE Symp. on Large-Scale Data Analysis and Visualization* (2013), pp. 1–6. 3

[NLL*12] NOUANESENGSY B., LEE T.-Y., LU K., SHEN H.-W., PETERKA T.: Parallel particle advection and FTLE computation for time-varying flow fields. In *Proc. Intern. Conference on High Performance Computing, Networking, Storage and Analysis* (2012), pp. 1–11. 3

[NSJ14] NOVÁK J., SELLE A., JAROSZ W.: Residual ratio tracking for estimating attenuation in participating media. *ACM Trans. Graph. 33*, 6 (2014). 1, 3, 5

[Pop03] POPINET S.: Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. *Journal of computational physics 190*, 2 (2003), 572–600. 6

[SG08] SPANIER J., GELBARD E. M.: *Monte Carlo principles and neutron transport problems*. Courier Corporation, 2008. 2

[SLB22] SAHOO S., LU Y., BERGER M.: Neural flow map reconstruction. *Computer Graphics Forum 41*, 3 (2022), 391–402. 3

[SP07] SADLO F., PEIKERT R.: Efficient visualization of Lagrangian coherent structures by filtered AMR ridge extraction. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1456–1463. 3

[Spa66] SPANIER J.: Two pairs of families of estimators for transport problems. *SIAM Journal on Applied Mathematics 14*, 4 (1966), 702–713. 1, 2

[WMHL65] WOODCOCK E., MURPHY T., HEMMINGS P., LONGWORTH S.: Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proc. Conf. Applications of Computing Methods to Reactor Problems* (1965). 1, 2

[WRT18] WILDE T., ROSSL C., THEISEL H.: FTLE ridge lines for long integration times. In *IEEE Scientific Visualization Conference (SciVis)* (Los Alamitos, CA, USA, 2018), IEEE Computer Society, pp. 57–61. 3

[WZU*21] WALD I., ZELLMANN S., USHER W., MORRICAL N., LANG U., PASCUCCI V.: Ray tracing structured AMR data using ExaBricks. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (2021), 625–634. 6

[XLT23] XI Y., LUAN W., TAO J.: Neural Monte Carlo rendering of finite-time Lyapunov exponent fields. *Visual Intelligence 1*, 1 (2023), 10. 3

[ZGH*18] ZHANG J., GUO H., HONG F., YUAN X., PETERKA T.: Dynamic load balancing based on constrained k-d tree decomposition for parallel particle tracing. *IEEE Transactions on Visualization and Computer Graphics 24*, 01 (2018), 954–963. 3

[ZY18] ZHANG J., YUAN X.: A survey of parallel particle tracing algorithms in flow visualization. *Journal of Visualization 21* (2018), 351–368. 3