Autonomous Particles for Interactive Flow Visualization

Wito Engelke, Kai Lawonn, Bernhard Preim and Ingrid Hotz

The self-archived postprint version of this journal article is available at Linköping University Institutional Repository (DiVA):

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-151724>

N.B.: When citing this work, cite the original publication.

Engelke, W., Lawonn, K., Preim, B., Hotz, I., (2018), Autonomous Particles for Interactive Flow Visualization, *Computer graphics forum (Print)*, , 1-12. <https://doi.org/10.1111/cgf.13528>

Original publication available at:

<https://doi.org/10.1111/cgf.13528>

Copyright: Wiley (12 months)

<http://eu.wiley.com/WileyCDA/>

Autonomous Particles for Interactive Flow Visualization

Wito Engelke¹, Kai Lawonn^{2,3}, Bernhard Preim² and Ingrid Hotz¹

¹Department of Science and Technology, Linköping University, Sweden
{wito.engelke, ingrid.hotz}@liu.se

²Department of Simulation and Graphics, University of Magdeburg, Germany
{lawonn, bernhard}@isg.cs.uni-magdeburg.de

³Institute for Computational Visualistics, University of Koblenz-Landau, Germany

Abstract

We present an interactive approach to analyse flow fields using a new type of particle system, which is composed of autonomous particles exploring the flow. While particles provide a very intuitive way to visualize flows, it is a challenge to capture the important features with such systems. Particles tend to cluster in regions of low velocity and regions of interest are often sparsely populated. To overcome these disadvantages, we propose an automatic adaption of the particle density with respect to local importance measures. These measures are user defined and the systems sensitivity to them can be adjusted interactively. Together with the particle history, these measures define a probability for particles to multiply or die, respectively. There is no communication between the particles and no neighbourhood information has to be maintained. Thus, the particles can be handled in parallel and support a real-time investigation of flow fields. To enhance the visualization, the particles' properties and selected field measures are also used to specify the systems rendering parameters, such as colour and size. We demonstrate the effectiveness of our approach on different simulated vector fields from technical and medical applications.

Keywords: flow visualization, scientific visualization, visualization, real-time rendering

ACM CCS: Visualization [Human-centered computing]: Visualization application domains–Scientific Visualization

1. Introduction

The increasing complexity and size of data in computational fluid dynamics make suitable analysis tools for domain experts inevitable. Thereby, particle-based techniques generally provide easy-to-understand methods for flow visualization complementing static images as, for example, streamlines. However, the number of used particles and their seeding positions are crucial for the effectiveness and efficiency of the visualization. There is a general trade-off between *sparse seeding* reducing clutter with a high chance of missing features and *dense seeding*, which increases clutter but is less likely to miss features. However, dense seeding also does not guarantee that no features are missed since particles tend to cluster in regions of low velocity or convergent behaviour, while regions with high velocity or divergent behaviour are under-sampled. By introducing a system of *autonomous particles* we target both aspects, decreasing the particle density in regions of less interest and increasing the sampling rate in regions of interest automatically during run-time. The resulting set of particles can be considered as a smart sampling of the flow. Compared to classic particle systems,

it highlights features without explicitly controlling particle density, which would corrupt the highly parallel nature of the underlying algorithms. Instead, the density is controlled by particle-events like birth, death and spawning new particles based on *local field information*, for example, divergence or rotation and few *particle properties* like its energy level. The approach supports interactive data exploration without requiring pre-computation or complex field analysis. It is a compromise between efficient interactive direct visualization and a feature-based exploration. Our main contributions are as follows:

- A new particle-based method, which adapts its density automatically to the complexity of the underlying flow highlighting important features while still providing context information.
- Non-trivial GPU-based implementation for all algorithmic parts, which enable interactive frame rates even for large particle sets.
- An interactive framework for analysis and exploration of vector fields in real-time. Few intuitive parameters provide direct control over the behaviour of the particle set during run-time.

We demonstrate our approach on four different data sets obtained by fluid simulation from technical and medical applications. The remainder of this paper is organized as follows. After related work in Section 2, we give an overview of our method in Section 3 followed by a detailed description of the individual building blocks. Afterwards, we present several use cases and results, achieved with our method, on the different data sets. Subsequently, we analyse timing and memory consumption in Section 5 and finally conclude our work with a discussion in Section 6, also naming possible future developments.

2. Related Work

Glyph-based techniques are widely used for scientific visualization to encode information. They are employed, for example, in molecular dynamics [TCM06], medical applications [KWL*04], astrophysics [WKM05] and flow simulation [BAH13]. The shape of a glyph may be specifically designed for certain domains (e.g. glyphs on a map or for individual data points, arrow glyphs representing blood flow) or of simple shape like points. For visualizations it is common to animate glyphs over consecutive frames. The resulting motion and development of structure lead to a better understanding of the underlying data.

For flow visualization, glyph-based and particle-based methods, in particular, are well known. Important contributions were presented by Krüger *et al.* [KKKW05]. They rely on a GPU-based approach for particle integration and rendering by Kipfer *et al.* [KSW04]. Later they introduced an importance driven scale-space approach to target visual clutter and information overload in the work of Bürger *et al.* [BKKW08]. As a different application, Van Pelt *et al.* used particle-level sets to enhance accuracy of interactive dye advection [VBB*11].

2.1. Streamline selection and seeding strategies

Such techniques also target the problem of avoiding clutter highlighting important features in flow data. They have been well studied over the past years [MLP*10]. The specific methods vary a lot with respect to targeted features and constraints. Most methods for 3D vector fields are based on some intelligent selection of a large amount of pre-computed lines. Salzbrunn *et al.* [SS06] introduced streamline predicates for line selection. This approach was later also extended to pathlines [SGSM08]. McLoughlin *et al.* [MJL*13] presented a similarity measure for stream and path lines. Furthermore, Tao *et al.* [TMWS13] contributed a unified approach for the selection of streamlines and viewpoints. Similarly, Xu *et al.* [XLS10] discussed streamline selection and placement in the context of information theory. Besides these selection methods, which consider the streamline as a whole, feature-based streamline seeding is used placing the start point of integration in some region of interest. With this, such methods are related to our approach but they are less flexible and can only simulate parts of the results obtained with autonomous particles.

2.2. Adaptive refinement

Our method is also related to adaptive refinement methods which have been used in many contexts. Thereby, refinement is mostly

applied based on some local error measure related to the current sampling. This is especially useful, when high accuracy is required as, for example, for the computation of FTLE ridges [BT13]. In contrast, our method is not targeted to ensure accuracy but to use a given contingent of particles in an efficient way without overhead.

Feature-based methods defining criteria for interesting regions in vector-fields are various. Here, we summarize the most commonly used methods. Besides, direct measures (e.g. magnitude) we focus on criteria for vortices and vortical behaviour. Techniques, for the explicit extraction of vortex cores, were presented by Sujudi and Haines [SH95] as well as by Kasten *et al.* [KWP*10]. Other methods use scalar identifiers for vortex extraction. They are mostly based on the Jacobian $J(\mathbf{x})$ which is approximated by calculating the central differences of the vectors in $\mathbf{V}$ in x, y, z -direction for every sample point $\mathbf{x} \in \mathbf{D}$. Such measures are more suitable for our purpose, since they do not require complex data preprocessing and can be computed locally. Exemplary, the method proposed by Jeong and Hussain [JH95] uses the second largest eigenvalue λ_2 of $P := \frac{1}{4}((J + J^T)^2 + (J - J^T)^2)$ as scalar identifier.

Besides measures related to rotation, those related to divergence are of interest. The finite-time Lyapunov exponent (FTLE) introduced by Haller *et al.* [HP98] was frequently used over the past years to identify areas of flow separation. Especially ridges in the resulting scalar fields were often subject of investigation in the literature. Work was undertaken in the direction of tracking ridges, as presented by Lipinski *et al.* [LM10] or in the efficient computation of FTLE on both, CPUs and GPUs [CRK12] as well as higher accuracy in computation [ÜSK*12]. Using FTLE fields for clustering approaches was presented by Englund *et al.* [ERH16].

3. Autonomous Particles

Our visualization method is based on a system of autonomous particles adapting to the flow addressing both previously mentioned challenges, that is, the reduction of visual clutter and increase of sampling rate in regions of interest. The main idea is that instead of a fixed number of particles a flexible budget of *autonomous particles* is used, which is divided into two categories *context particles* 1(a) and *feature particles* 1(c). Furthermore, particles have the ability to appear, vanish or split by following simple rules for birth, death and multiplication, this results in an intelligent sampling of the flow and an efficient use of the budget. With this, flow features are highlighted but context information is still preserved. The adaptation requires only local field evaluations and few particle properties keeping track of their history. Thereby, each particle explores the flow independently without communicating to any other particle. In the following these rules will be explained in more detail.

We use the following notation: We refer to a single particle as p , the particles position as $\mathbf{p}(p) \in \mathbb{R}^3$ and particle property vector as $\omega(p) \in \mathbb{R}^4$. The vector field $\mathbf{V}$ is defined as a map from $\mathbf{D}$ to $\mathbb{R}^3$: $\mathbf{V} : \mathbf{D} \rightarrow \mathbb{R}^3$. Where the vector of the particle p at the position $\mathbf{p}$ is denoted by $\mathbf{v}(\mathbf{p})$.

Figure 1: Particle types: The system distinguishes between context and feature particles. (a) shows context particles only ($\theta_1 = -10.0$). The systems sensitivity to features in the flow can be interactively steered during the run-time of the algorithm. In (b) this sensitivity is increased ($\theta_1 = -1.0$), which results in splitting of context particles when hitting a flow feature. Further increase of the sensitivity ($\theta_1 = -0.1$) can be observed in (c), where rendering of context particles is deactivated. For all images the λ_2 criterion was used.

3.1. Requirements

Our method empowers the user to explore an unknown data set and gain understanding without any pre-computation or data set analysis. With this we derive the following requirements.

REQ. 1—REDUCTION OF VISUAL CLUTTER: Visual clutter as a result of dense particle seeding is reduced.

REQ. 2—HIGHLIGHTING FLOW FEATURES: The behaviour of the underlying flow near features, defined with respect to a user-selected criterion, are highlighted and visualized by an increased sampling rate.

REQ. 3—DIRECT INTERACTION: The user is able to directly control appearance and behaviour of the autonomous particles during its run- and in real time.

REQ. 4—HIGH PERFORMANCE: The algorithm does not perform significantly slower than a fixed-size particle system. Ideally, the GPU is leveraged for computation as well as rendering due to the algorithms parallel nature.

IMPLEMENTATION: An important feature of our particle system is its interactivity, which requires an efficient implementation. In each subsection we address implementation details within the last paragraph. Figure 2 gives an overview of the algorithm's pipeline which each particle undergoes every frame. Our current implementation uses CUDA for computation and OpenGL for rendering. Furthermore, the implementation facilitates direct interoperability between the CUDA and OpenGL contexts. The computation is performed by separate CUDA kernels for different algorithm steps. Specifically, four kernels are used for:

- Particle initialization
- Particle advection/evaluation/split and delete
- Memory defragmentation
- Particle insertion of context particles

Memory defragmentation is an integral part of our algorithm but performed only if necessary or explicitly requested. Similarly, context particles are only inserted if explicitly requested.

3.2. Particle and system properties

A few particle and system properties steer the overall and each particles behaviour. The main component of the system is a collection of autonomous particles. Besides its position each particle carries four additional properties stored in the particles property vector. These

Figure 2: The algorithm shows the steps performed in parallel for each particle in each frame. The highlighted steps are optional and only performed when necessary or explicitly requested.

are local velocity magnitude $|\mathbf{v}|$ with $\mathbf{v} = \mathbf{V}(\mathbf{p})$, particle energy ω_e , particle generation ω_g and the local evaluation of the split criterion for rendering purposes. The energy and the generation property keep track of the history of the particle and, as such, play a fundamental role in controlling the split and delete events and thus the overall behaviour of the system.

The system properties include user-defined split criteria, which correspond to features of interest. They are evaluated by the individual particles in each calculation step. The criteria can either be computed beforehand or on-the-fly, for example, a local evaluation of the properties of the Jacobian or the particle history. Our implementation supports both. Additionally, criteria combination is supported.

A further central property of the system is its total particle budget n_b . During initialization, a small amount $n_p \leq 0.01 \cdot n_b$ of context particles is created. Hence, they are not generated by a split event; these particles can also be understood as parent particles. In fact, we use these terms interchangeably. Then, during progression of the system, split operations create additional particles, these are called

feature particles, which consume memory from the budget. Both, the amount of context and feature particles vary during run-time but their sum never exceeds the total budget $n_p + n_c \leq n_b$. This particle budget n_b is an upper bound and only limited by computing power and available memory.

IMPLEMENTATION: As stated each particle p is represented by a position $\mathbf{p} \in \mathbb{R}^3$ and additional properties collected in a property vector $\omega \in \mathbb{R}^4$. These data are stored in two vertex buffer objects (VBOs) of size n_b and updated during the individual steps. The systems particle budget n_b is defined during initialization and cannot be changed during run-time. During initialization the corresponding amount of GPU memory (VBO) for rendering and computation is allocated. Furthermore, the system stores and updates two indices i_c and i_p . Memory index i_p points the last parent particle, that is, the location where $\omega_g(p_{i_p-1}) = 0$ and $0 < \omega_g(p_{i_p}) \leq g_{max}$. Therefore, this indicates the current amount of parent (or context) particles in the system. Similarly, i_c points to the first unused memory location, that is, the location where $0 \leq \omega_g(p_{i_c-1}) \leq g_{max}$ and $\omega_g(p_{i_c}) = g_{max} + 1$.

As numerical integration method we use the fourth-order Runge–Kutta scheme with a fixed step size, as this is a good trade-off between performance and accuracy. The vector field resides on the GPU as 3D-texture and is interpolated linearly.

3.3. Initialization and particle birth event

Context particles are seeded during the initialization phase of our algorithm. The start positions are either unconstrained, that is, particles are randomly inserted in the domain or constrained, that is, the position is determined by a binary importance volume, for example, domain sub-spaces. Their energy level $\omega_e(p)$ is set to e_{max} and their generation property is set to zero $\omega_g(p) = 0$ flagging them as parent particles. In addition, our particle system can be configured to maintain a constant amount of context particles. For this, the system inserts further parent particles if the count n_p is falling below a desired value.

IMPLEMENTATION: Seeding the initial amount of context particles is realized with a kernel launch. It is also responsible for determining the initial position. Typically the amount of these particles is very low and in the order of less than 1% of the particle budget, for all experimented data sets. However, our method does not restrict the number of initial parent particles in any sense.

3.4. Particle energy level and death event

The particles energy level (see Section 3.2) is reduced in every computation step during run-time. Additionally, the amount of lost energy is determined by a Gaussian distribution function, where the mean value is under the control of the user during run-time (determined by σ_1 , see Section 3.7). If the energy level ω_e falls below 0 the particle is deleted. This can be understood as deleting particles, which did not traverse an interesting region for a specific amount of time. The random fluctuation, determined by the Gaussian distribution function, ensures that not all particles of a certain generation are deleted at once.

IMPLEMENTATION: The deletion of a particle is realized by flagging its memory position as available. For this, the generation property is set to $\omega_g(p) = g_{max} + 1$ which is outside of the interval $[0 \dots g_{max}]$ used for valid particles. During rendering these memory positions are ignored and no colour or depth information is written into the corresponding buffers.

3.5. Particle split event

A split event occurs if the local evaluation of the selected criterion with respect to the particles sensitivity is reached. The criteria for splitting of particles can be classified into three categories, geometric, for example, distance fields, flow related, for example, λ_2 criterion, vorticity and particle history, for example, local curvature along the particles path. A particle split creates n_{split} new child particles. Furthermore, the individual particle positions are determined by adding an individual random offset to the parent particle location. The magnitude of this displacement is configurable (see Section 3.7). Subsequently the parent particle dies. For all examples in this paper a split number of $n_{split} = 4$ is used, which is a good trade-off between exploration of a particles surrounding and preventing a strong local growth in particle count. The child particles, generated by a split event, are initialized with a generation property increased by one compared to their parents and a maximum energy level.

To avoid exponential increase of particles in regions where the split criterion is fulfilled, the split sensitivity also depends on the particle generation. Therefore, a flow feature has to be more pronounced to cause splitting of higher generation particles. Furthermore, no splitting operation is performed if the budget n_b is exhausted or the maximum allowed generation is reached (determined by θ_3 , see Section 3.7).

The process is schematically visualized for two particles in Figure 3. These are advected with the vector field (Figure 3b). In Figure 3(c) one of the particles (represented in gray) dies due to a low energy level. In Figure 3(d) the splitting criteria for the second particle is fulfilled and the particle spawns n_{split} child particles. For each child a random vector is added to the parent particles location. The parent particle itself dies (Figure 3e). The newly spawned

Figure 3: Illustration of our algorithm. (a) initial situation with two context particles, (b) context particles advected in the field, (c) deletion of one context particle, (d) split of one context particle into four feature particles. (e) deletion of parent and (f) deletion of two particles and splitting of one into four children.

particles are added to the particle system and are treated according to the same rules as the parent particles.

IMPLEMENTATION: Particle split events are handled as follows. If the split criterion for a particle is fulfilled the system checks for available memory. Specifically, the current index i_c of the next available memory position is compared to the budget such that $n_p + n_c + n_{split} < n_b$ since a split can only be performed if enough memory is available. Next, the particle generation $\omega_g(p)$ of the split candidate is compared to the maximal allowed generation g_{max} . If both conditions are fulfilled the index i_c is updated by the executed kernel facilitating an atomic operation, which ensures that every other thread operates on that updated index. The positions of the n_{split} new children are determined by $\mathbf{p}(c_i) = \mathbf{p}(p) + \theta_4 \cdot \mathbf{r}_i$ with a unit random vector $\mathbf{r}_i$ scaled with θ_4 for $1 \leq i \leq n_{split}$. This scaling can be interactively controlled (see Section 3.7). Generation and energy levels are updated as described above. Specifically, a particle p of generation $\omega_g(p)$ splits into $c_1, \dots, c_{n_{split}}$ children with $\omega_g(c_i) = \omega_g(p) + 1$ for $1 \leq i \leq n_{split}$. The energy level of the child particles is initialized with $\omega_e(c_i) = e_{max}$ for $1 \leq i \leq n_{split}$. Lastly, the memory position of the parent particle is flagged as available by setting $\omega_g(p) = g_{max} + 1$.

3.6. Defragmentation

The described particle death and split events lead to fragmentation of the linear memory and therefore to an inefficient memory usage. This situation is illustrated in more detail in Figure 4. After the initialization the memory is densely packed and the index i_c is pointing to the first unused memory location, for example, the first location where $\omega_g = g_{max} + 1$, see Figure 4(a). Furthermore, given the dense packing this location is minimal. During a split operation new particles are generated and i_c is advanced accordingly. At the same time the parent particle is deleted, Figure 4(b). Further split and delete operations during the systems development lead to increasing fragmentation as shown in Figure 4(c). To guarantee that the algorithm does not overwrite occupied memory locations, the

index i_c has to point to the *last* position where $\omega_g(p_{i_c}) = g_{max} + 1$ and $0 \leq \omega_g(p_{i_c-1}) \leq g_{max}$. At this state i_c is not minimal anymore. Figure 4(c) illustrates this situation wherein the system could only perform one additional split, for example, consuming four additional memory locations although, in total, 10 memory locations are available, which would be enough for two splits.

The fragmentation does not influence rendering or integration performance, however, leads to less efficient usage of the allocated memory. Consequently, the system may not perform additional splits even if the necessary amount of memory locations are unoccupied but scattered within the memory segment (see Figure 4c) and therefore unusable. To resolve this issue a defragmentation mechanism is integrated. This mechanism is realized with a GPU-based sorting of the linear memory based on particle generation. This algorithm has to ensure optimal usage of the available resources and must not incorporate recursive calls. Possible candidates, which fulfill the requirements are Parallel Radix Sort by Zagha *et al.* [ZB91] and Bitonic Merge Sort presented by Pharr *et al.* [PF05]. The latter methods' foundation is a sorting network contributed by Batcher *et al.* [Bat68] and is used in our implementation. For this, three operations are performed:

- Sorting elements with respect to particle generation property;
- Index i_c update, such that $\omega_g(p_{i_c}) = g_{max} + 1$ and $0 \leq \omega_g(p_{i_c-1}) \leq g_{max}$ is fulfilled and i_c is minimal;
- Index i_p update, such that $\omega_g(p_{i_p}) = 0$ and $\omega_g(p_{i_p-1}) > 0$ is fulfilled and i_c is minimal.

DEFRAGMENTATION: The particle data reside entirely on the GPU, throughout the run-time of the algorithm. Thus, copying it between device and host memory is no feasible option and would cause a significant overhead. Therefore, an in-place GPU-based sorting algorithm is used, which also exploits the computing power of the GPU for the sorting step. Specifically, Bitonic Merge Sort [PF05], proposed for the use with programmable graphics hardware, is integrated in our system. In contrast to other implementations our variant operates in-place, meaning that no additional memory is allocated for the sorted sequence. Our algorithm sorts particles' positions and properties according to the ω_g key in the property vector. The algorithm sorts the elements in an ascending order, for example, an exchange is performed if $\omega_g(p_i) > \omega_g(p_j)$ for $0 < i < j < n_b$. This method fully utilizes the GPU resources and runs with $\frac{n_b}{2}$ parallel threads at all steps.

INDEX UPDATE: After sorting the memory an update step to the index i_c and i_p is performed. Instead of using a linear search pattern ($O(n)$) a kernel executions with n_b threads (parallel complexity of $O(1)$) and an atomic max operation is used to update i_c and i_p , respectively.

The resulting configuration after sort and index update steps is illustrated in Figure 4(d). Since sorting is the most computational expensive part of the algorithm (see Section 5) it is only performed in so-called *sorting frames*. Specifically, the *adaptive sort* mode performs a sort and index update if $i_c \geq 0.75n_b$ and $i_c \leq 0.99n_b$. This metric prevents the system of performing costly sorting steps when this is not necessary ($i_c \geq 0.75n_b$) or not beneficial ($i_c \geq 0.99n_b$). As example, the results shown in Figure 7 are achieved with only one sorting frame out of 45 frames in total. Note that the

Figure 4: Memory fragmentation and sort: Illustrative visualization of an exemplary set of autonomous particles with $n_p = 2$ and $n_b = 16$. The gray boxes are empty memory locations, whereas the coloured are occupied: (a) shows initial configuration with $i_c = 2$, (b) after first split event ($i_c = 6$), (c) after further split and delete events ($i_c = 10$) and (d) after sorting ($i_c = 6$).

set of autonomous particles is configurable to *never sort*, *adaptive sort* and *always sort*. Furthermore, the before-mentioned insertion of context particles (see Section 3.3) requires a configuration of *always sort*.

3.7. Interactive control

Besides the algorithms configuration (e.g. particle budget n_b , sorting scheme, count n_{split} of spawned children during split events) there are few parameters to interactively control the behaviour of the system. They influence the algorithm itself and the particle sensitivity for splitting. In our implementation two vectors $\theta, \sigma \in \mathbb{R}^4$ are passed to the CUDA kernel as launch parameters, in every frame:

- θ_1 : General split sensitivity for p with $\omega_g(p) = 0$
- θ_2 : Additional split sensitivity for p with $0 < \omega_g(p) \leq g_{max}$
- θ_3 : Maximal allowed particle generation
- θ_4 : Magnitude of children displacement vector
- σ_1 : Amount of particle energy loss

Currently three components of σ are unused and available for future development.

Parameters related to particle split events are as follows:

PARAMETER θ_1 controls the threshold of splitting. No splitting operation occurs, if the local value of the selected flow field measure, is below (e.g. λ_2) or above (e.g. velocity magnitude) this threshold. It influences the general sensitivity of the system. PARAMETER θ_2 controls the additional sensitivity against the selected flow field measure(s) for higher generations. Given this, a particle with a higher generation needs a stronger measure to perform a split. This prevents rapidly firing split events, when particles reach a critical area. With this the evaluation of the current flow criteria follows $S(\mathbf{p}) < \theta_1 - \theta_2 \cdot \omega_g(p)$ for a negative scalar measure like the λ_2 criterion.

PARAMETER θ_3 determines the upper limit of maximal allowed particle generations. Specifically, a particle p is only split if it has a generation property with $\omega_g(p) \leq \theta_3$. This parameter prevents an early exhaustion of the particle budget n_b and determines g_{max} .

PARAMETER θ_4 controls the distance of the displacement of the children relative to their parent particle. Thus, all n_{split} children lie on a sphere with radius θ_4 centered around the generating particle. Higher values can lead to faster exploration of the surrounding of a specific location, whereas smaller values keep particles closer together and increase local density.

Parameters related to particle delete events are:

Parameter σ_1 defines the energy loss of the particles in every frame. It thus determines the lifetime of the particles which die if the energy level falls below zero. The effect is that particles disappear in uninteresting regions. To prevent the simultaneous death of many particles and achieve a higher frame coherence the energy loss follows a Gaussian distribution with mean value $\mu = \sigma_1$. The standard deviation is configurable and set to $\sigma = 0.2 \cdot \mu$ in our current implementation.

3.8. Rendering

Our particle rendering uses sphere impostors supporting two different rendering modes exemplarily:

- A deferred shading pipeline
- A forward rendering mode.

The deferred shading pipeline uses post processing for the calculation of screen space ambient occlusion (SSAO) and physically based shading (PBS). Whereas, the forward rendering mode is configured to disable depth testing and used for density projection. In general more rendering styles, like transparent particles, can be supported by our prototype. For this a depth-based sorting on fragment level for correct blending calculation is necessary. However, a discussion on particle rendering techniques is out of the scope of this paper.

SPHERE IMPOSTORS: To ensure low memory usage and high rendering performance each particle is represented as a position in 3D space and a 4D property vector (see Section 3.2). During rendering a screen-aligned quad is positioned at each particles location. This is realized by using *instanced drawing* instead of a relatively slow geometry shader. Each quad is coloured by a user-selected attribute, where the user can choose from.

- Particle attributes, for example, particle generation, particle energy
- Local field measures, for example, velocity magnitude
- Used split criterion, for example, λ_2 , curvature, geometric distance.

These attributes are mapped onto a user defined colour scale. Each quad is then clipped, which results in screen-aligned circles. Texture coordinates are used to generate normals. After performing the depth test, all necessary information is written to the G Buffer for lighting calculation in a post-processing step (deferred shading [PFG05, Lau10, SBF15]).

DEFERRED SHADING AND SSAO: Local lighting information makes it possible to distinguish single particles by visually separating them from each other. Whereas, overlaps or cavities are not easy perceptible with this approach. Eichelbaum *et al.* [EHS13] demonstrated how ambient occlusion enhances the spatial relationship of lines. The visualization of complex molecule structures also benefit from additional ambient occlusion as demonstrated by Tarini *et al.* [TCM06]. Gribble *et al.* [GP06] used a combination of ambient occlusion and diffuse inter-reflections to create more realistic particle visualizations, which was confirmed in a user study. However, this technique relies on an expensive pre-computation step.

We use a occlusion-based metric for the calculation of ambient light. Methods in this direction are widely known as SSAO and first introduced by CryTec [Mit07, Kaj09]. Bavoi *et al.* [BSD08] proposed an alternative approach by using Horizon Mapping [Max88] on the depth buffer for the occlusion approximation. With this, overlaps and cavities can be differentiated from other clusters. The user can freely decide, during run-time, how much contribution of each individual lighting term he wants to see in the resulting visualization. Furthermore, the trade-off between quality and performance, of the ambient occlusion approximation, can be tweaked during

run-time. With this interactive frame-rates are achieved without compromising visual quality of the visualization. Note that all our images, except Figure 3, 2 and 4, in this paper are based on actual screenshots of our framework.

DENSITY PROJECTION: Besides deferred shading, our framework allows visualization of particles via density projection as second rendering technique. This method is based on the approach of Kuhn *et al.* [KLG*13] for rendering integral lines and adapted to point rendering. Unlike enhancing the perception of spatial relation between objects or primitives (SSAO), this technique unveils the local density by deactivating the depth test and using a blend operation on the framebuffer. With this, we can directly visualize particle density in a frame-coherent manner and therefore unveil inner structures of point clusters. Figure 7(c) and 5(d) have been rendered by utilizing this approach.

4. Application Cases

For the demonstration of the effectiveness of the proposed method data sets related to technical and medical flow simulation have been chosen. For the first group the well-known, as benchmark data set, and the delta wing were used. The second group is formed by two aneurysm simulations from the CFD Rupture Challenge 2013 [BRB*15].

KÁRMÁN VORTEX STREET: The data set represents a simulation of a 3D flow around an obstacle, in our case a squared cylinder. Furthermore, the left boundary plane represents the inflow and the right boundary plane the outflow. We use this data set to illustrate the

variability of the system in terms of splitting parameters and criteria. We use the data set in two different configurations the original form and with subtracted mean flow. All results have been achieved by randomly inserting parent particles within the domain. Furthermore, the system has been configured to constantly insert parent particles, providing context information.

Different split criteria are compared in Figure 6. Figure 6(a) uses the curvature of the corresponding streamline, approximated as angle between the current and last integration step, as splitting criterion and a relatively long particle live time, which is realized by a low energy loss per computation step. The resulting particle flow highlights the vortex behind the cylinder, which is the only region with high line curvature. The particle stream nicely summarizes the particle movement in the data. The second image 6(b) facilitates λ_2 . The result is a high splitting rate also in the vortices further downstream in the flow. Since these vortices are hidden by the strong mean flow they emphasized features that are not directly visible in the particle movement.

The distance to the obstacle has been used in the second row. Due to the proximity of the dominant vortex in the flow also this configuration emphasizes the first vortex. These images display the impact of changing the particle displacement θ_4 during split events. Decreasing this value can be used to generate sharper line like structure.

In Figure 5, the mean flow has been subtracted. In this setting the particle flow is coherent with the λ_2 criterion which has been used for all examples. A high particle concentration in all vortices can be observed. In image 5(a) the rendering of parent particles has

Figure 5: Kármán Vortex Street: Images show the Kármán Vortex Street after subtraction of the laminar flow field. Colour encodes: (a) velocity magnitude, (b) particle energy level, (c) particle generation, (d) particle density. Figure (a) shows calculation step 48, whereas (b) and (c) show the system in a more developed state. Used parameters are: $n_b = 4 \cdot 10^6$, $\theta_1 = -0.1$, $\theta_2 = 0.1$, $\theta_3 = 0.6$, $\theta_4 = 0.25$, $\sigma_1 = 0.25$.

Figure 6: Kármán Vortex Street: Different split criteria for the Kármán Vortex Street. Colour encodes velocity magnitude in all images. In Figure (a) the curvature of the streamline is used for splitting, in (b) λ_2 is used as split criterion. In the second row the geometric distance field to the obstacle is used. For (c) and (d) the magnitude of the displacement θ_4 has been decreased which results in more line like structures.

Figure 7: Delta wing: Overview of data set after computing 45 steps. Seeding is constrained to a binary importance volume, which is rendered as bounding box. For (a) colour encodes local velocity magnitude, whereas for (b) colour encodes particle energy level. Particle density projection is used for revealing inner structures in (c), therefore transparency depicts particle density. Used parameters are: $n_b = 1 \cdot 10^6$, $\theta_1 = -0.1$, $\theta_2 = 0.1$, $\theta_3 = 0.6$, $\theta_4 = 0.15$, $\sigma_1 = 0.15$.

been deactivated and colour encodes velocity magnitude. A further developed state of the system is shown by Figures 5(b) and 5(c). Here, colour encodes particle energy level and particle generation, respectively. Lastly, Figure 5(d) shows a density projection of the particles. This rendering style is especially useful for still images.

DELTA WING: This simulation data set describes the flow around a simplified aircraft geometry often referred to as ‘delta wing’. For this type of aircraft geometry the uplift is generated by vortices forming at both sides of the wing.

For this data set particle seeding has been constrained to a binary importance volume. Again, our system has been configured to constantly insert particles within this volume. These few parent particles act as a source for continuous splitting near the wings tip. Primary and secondary vortex structures are found and automatically highlighted by our method. With the help of little user interaction our method revealed all vortices clearly. Besides that the inner structures of the two big vortices were observable (Figure 7c) by utilizing density projection.

FLOW IN AN ANEURYSM: CFD Challenge Case 1 and 2: Simulations depicting the blood flow within two aneurysm data sets presented for the CFD Challenge in 2013. In both data sets, complex flow structures are formed within the aneurysm sack.

Snapshots of our particle system are shown in Figure 8 and Figure 9. First experiments using the λ_2 criterion were not as successful and did not extract the internal vortex structure as expected. The λ_2 value close to the boundary is much stronger than in the vortex of interest and dominates the adaptive splitting (see Figure 8c). Instead the autonomous particle system has been configured to utilize line curvature based on the particle history as split criterion. With this, the normalized angle between the last and current integration step is evaluated and a particle is split if this measure falls under the current θ_1 value. This criterion emphasizes the internal structures of the flow clearly. The few particles near the wall provide necessary context information to instantly gain understanding of the flow behaviour within the aneurysm sack (see Figure 8a). During run-time the sensitivity against the criterion has been decreased and with this the vortex core structure isolated. This area of

Figure 8: Aneurysm Case 1: Results for the ‘Computational Fluid Dynamics Rupture Challenge 2013’ data set. As split criterion the local particle history, i.e. angle between current and last integration step was used. Figure (a) shows the inner part of the complex vortex within the aneurysm sack. For (b) the sensitivity is reduced, this leads to a harder condition and therefore a smaller split region. Particles near the inner wall provide context information. Iso-surfaces of the λ_2 are shown in (c) for an iso-value of -0.05 and -0.15 , respectively. When used, this criterion results in a less beneficial particle distribution leading to occlusion and under-sampling, since it does not cover the vortex structure clearly. Colour encodes velocity magnitude. Used parameters are: $n_b = 1 \cdot 10^6$, $\theta_1 = 0.8$, $\theta_2 = 0.1$, $\theta_3 = 0.8$, $\theta_4 = 0.15$, $\sigma_1 = 0.15$.

Figure 9: Aneurysm Case 2: Results for the ‘Computational Fluid Dynamics Rupture Challenge 2013’ data sets. As split criterion λ_2 was used. This data set is especially challenging. Compared to Figure 8 the inner vortex structures are not as pronounced but the data sets complex flow behaviour can still be well understood. Colour encodes velocity magnitude. Used parameters are: $n_b = 2 \cdot 10^6$, $\theta_1 = -0.1$, $\theta_2 = 1.0$, $\theta_3 = 0.6$, $\theta_4 = 0.15$, $\sigma_1 = 0.05$.

relatively low velocity magnitude, encoded as colour in both images, can be clearly observed.

In contrast, utilization of the λ_2 criterion lead to the visualizations shown in Figure 9. The vortex is not as pronounced when compared to the former shown Case 1 but the complex flow behaviour can still be understood. Here, particles where seeded in a spherical region located in the inflow vessel of the aneurysm.

The shown results demonstrate the flexibility of our method, which performed excellent on all data sets. Important flow features were highlighted easily and context information still preserved.

5. Performance Evaluation

All conducted tests were performed on a workstation with a one Xeon E5-1650v4 running at 3.6 GHz, 64 GB system memory and an nVidia Geforce 1080TI graphics card driving two QHD (2560 × 1440) displays.

ALGORITHM TIMINGS: The first observation one can make is that particle advection, including split and delete operations, are the least and the defragmentation the most time-consuming part of our algorithm. Table 1 provides detailed timing information for different particle system configurations. Furthermore, we compare these

Table 1: Algorithm timings of our method. Measurements are conducted multiple times and the average time is listed. Memory defragmentation is the most expensive step in our algorithm but not performed for every frame. Our system has a penalty of 30–50% in the advection step compared to a non adaptive (classic) system.

Budget $n_b \cdot 10^6$	0.26	1.04	4.19	8.38	–
Advection (classic)	0.4	0.6	1.6	2.9	ms
Advection (adaptive)	0.6	0.8	2.8	4.1	ms
Defragmentation	9.8	19.1	57.2	116.1	ms
Total	10.4	19.9	60.0	120.2	ms
Overhead	50.0	33.3	33.3	41.3	%

Table 2: Memory consumption for the individual data sets. For the total value $n_b = 8 \cdot 10^6$ as an upper bound (224 MB), and the G-Buffer (225 MB) were considered.

Data set	Kármán	Delta wing	Cases 1 & 2	–
Vector field	8.8	47.6	512.0	MB
Scalar field	2.2	11.9	128.0	MB
Total	458.0	506.5	1089.0	MB

timings to a non-adaptive version running under the same conditions. A direct comparison is difficult, since our proposed method provides a smarter sampling of the domain and therefore uses less particles than a traditional system.

MEMORY CONSUMPTION: Since our algorithm is running entirely on the GPU, we focus on the GPU memory only. We consider the three main parts, where GPU memory is consumed. First, data set storage. Second, the set of autonomous particles. Third, rendering, as well as post-processing (e.g. deferred shading and SSAO).

The vector fields are uploaded once as a four-channel texture (see Section 3.2) and therefore consume four 32-bit floating-point values for each element. If an additional pre-computed scalar field is used, we store an additional one-channel 32-bit texture.

The second part is the particle system itself. For this the GPU memory is allocated once during initialization. The amount of consumed memory is proportional to n_b and the amount of attributes per particle. Currently, two vertex attributes for each particle bound to the particle systems vertex array object (VAO) are stored. Specifically, three 32-bit floating-point values encoding position and four 32-bit values reserved for properties per particle are used. With this, the memory consumption of the particle system is $7 \cdot n_b$ bytes, which results in 28 MB, 112 MB and 224 MB for sizes of $n_b = 1 \cdot 10^6$, $n_b = 4 \cdot 10^6$, $n_b = 8 \cdot 10^6$, respectively.

Lastly, additional graphics memory is consumed for our deferred shading and post processing approach. Our G-Buffer implementation uses four 32-bit floating point RGBA textures with a resolution of 2560×1440 pixels which results in a memory consumption of 225 MB. The resulting memory consumption is listed in Table 2. Note that the memory for the particle system as well as deferred shading and post-processing do not depend on the used data set.

6. Discussion and Conclusion

Observing moving particles is a powerful way to experience a flow [BKKW08]. To facilitate this experience for a large class of different flows has been the main motivation for our work. This means building a system capable of handling large sets of particles, reducing visual clutter while highlighting the flow in regions of interest. A system that automatically handles the trade-off between *sparse* and *dense seeding*. Thereby the notion of an ‘interesting region’ is treated as a flexible concept. These goals have been implemented by the introduction of autonomous particles, which explore the flow by themselves, only communicating with the local field properties at their current position. The results given in Section 4 clearly show that the set of autonomous particles fulfill these goals expressed in the targeted requirements Req.1 and Req.2 described in Section 3.

Through few intuitive parameters, our approach is easily controllable and flexible to target different usage scenarios, as described in Req.3. The setting of the few control parameters and the used split criterion provide a simple and meaningful way to steer the system. The resulting effect on the visualization when changing individual parameters can be observed in the accompanying video. The presented method is purely GPU-based. It is interactive also for large sets of particles. The performance evaluation in Section 5 proves that the proposed method does not perform significantly slower than a traditional particle system and therefore fulfills Req.4. Since the concept of autonomous particles is parallel in nature, it can also serve as a basis for distributed and cluster-based methods targeting the ever-growing size of flow data sets. Additionally, our method does not rely on pre-computation or field analysis, which is highly beneficial in the context of streaming and time-dependent data.

In summary, the system serves as smart sampling of the underlying flow providing insight in flow features within necessary context information. The results from the automatic adaptation of the particle density are not achievable by simple adjustment of the seeding strategy with classic particle systems. Observing moving particles supports an intuitive way of understanding flow phenomena in unknown data or known data for more detailed inspection. As such, it is a powerful and easy to control method for explorative flow visualization or as complement for other visualization techniques.

Results of streamline selection and placement methods can, depending on the criteria, deliver similar results but are in general targeting different goals. Our method can certainly not replace such approaches since the quality of a particle path is never evaluated as a whole; it is also not possible to consider view dependence with autonomous particles. On the other hand, our method avoids the expensive pre-computation of streamlines which will never be rendered, it can directly be applied to an unknown data set using a fixed budget of particles in an efficient way to render the data at hand. Methods, which rely on comparison of streamlines and clustering directly counteract with the idea of autonomous particles that do not know anything of each other.

During our experiments we found that the system reaches a steady state frequently, meaning that the particle count in total stays almost constant and deletion as well as split and insertion operations level each other out with respect to particle count. This is a desirable behaviour and is intended, meaning that a smart sampling of the

data set (i.e. high density in areas of interest, low density but context information in other areas) is found in a consistent manner.

In the current setting the method targets static data sets and cannot directly deal with unsteady flows. Principally the concept is transferable to time-dependent data sets, however it would require a reformulation of the split and delete criteria as well as seeding and some of the rendering strategies to be able to handle path, streak and time line and surface like structures. Given this, we consider the handling of time-dependent data sets as future work.

Acknowledgements

We would like to thank Carsten Neumann and Jochen Jankowai for various discussions. We also thank Markus Rütten (DLR Göttingen) for providing the delta wing data set. This work was supported through grants from the Stiftelsen för Strategisk Forskning Sweden (SSF project BD15-0082), from the Excellence Center at Linköping and Lund in Information Technology (ELLIIT), and the Swedish e-Science Research Centre (SeRC).

References

- [BAH13] BRAMBILLA A., ANDREASSEN O., HAUSER H.: Integrated multi-aspect visualization of 3D fluid flows. *Proceedings of VMV: Vision, Modeling & Visualization* (Lugano, Switzerland, 2013), The Eurographics Association, pp. 1–9.
- [Bat68] BATCHER K. E.: Sorting networks and their applications. *Proceedings of AFIPS* (1968), 307.
- [BKKW08] BÜRGER K., KONDRATIEVA P., KRUGER J., WESTERMANN R.: Importance-driven particle techniques for flow visualization. In *Proceedings of IEEE PacificVIS* (Kyoto, Japan, 2008), pp. 71–78.
- [BRB*15] BERG P., ROLOFF C., BEUING O., VOSS S., SUGIYAMA S.-I., ARISTOKLEOUS N., ANAYIOTOS A. S., ASHTON N., REVELL A., BRESSLOFF N. W., BROWN A. G., CHUNG B. J., CEBRAL J. R., COPELLI G., FU W., QIAO A., GEERS A. J., HODIS S., DRAGOMIR-DAESCU D., NORDAHL E., SUZEN Y. B., KHAN M. O., VALEN-SENDSTAD K., KONO K., MENON P. G., ALBAL P. G., MIERKA O., MÜNSTER R., MORALES H. G., BONNEFOUS O., OSMAN J., GOUBERGRIIS L., PALLARES J., CITO S., PASSALACQUA A., PISKIN S., PEKKAN K., RAMALHO S., MARQUES N., SANCHI S., SCHUMACHER K. R., STURGEON J., ŠVIHOVÁ H., HRON J., USERA G., MENDINA M., XIANG J., MENG H., STEINMAN D. A. and JANIGA G.: The computational fluid dynamics rupture challenge 2013 phase II: Variability of hemodynamic simulations in two intracranial aneurysms. *Journal of Biomechanical Engineering* 137, 12 (2015), <https://doi.org/10.1115/1.4031794>.
- [BSD08] BAVOIL L., SAINZ M., DIMITROV R.: Image-space horizon-based ambient occlusion. *ACM SIGGRAPH Talks*, 3 (2008), 1.
- [BT13] BARAKAT S. S., TRICOCHÉ X.: Adaptive refinement of the flow map using sparse samples. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 12 (2013), 2753–2762.
- [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.
- [EHS13] EICHELBAUM S., HLAWITSCHKA M., SCHEUERMANN G.: Lineao-improved three-dimensional line rendering. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 3 (2013), 433–445.
- [ERH16] ENGLUND R., ROPINSKI T., HOTZ I.: Coherence maps for blood flow exploration. In *Eurographics Workshop on Visual Computing for Biology and Medicine* (Bergen, Norway, 2016), S. Bruckner, B. Preim, and A. Vilanova (Eds.), Eurographics Association.
- [GP06] GRIBBLE C. P., PARKER S. G.: Enhancing interactive particle visualization with advanced shading models. In *Proceedings of the 3rd Symposium on Applied Perception in Graphics and Visualization* (2006), ACM, pp. 111–118.
- [HP98] HALLER G., POJE A.: Finite time transport in aperiodic flows. *Physica D: Nonlinear Phenomena* 119, 3 (1998), 352–380.
- [JH95] JEONG J., HUSSAIN F.: On the identification of a vortex. *Journal of Fluid Mechanics* 285 (1995), 69–94.
- [Kaj09] KAJALIN V.: Screen space ambient occlusion. *ShaderX7* (2009), 413–424.
- [KKKW05] KRÜGER J., KIPFER P., KONDRATIEVA P., WESTERMANN R.: A particle system for interactive visualization of 3d flows. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 11, 6 (2005), 1–13.
- [KLG*13] KUHN A., LINDOW N., GÜNTHER T., WIEBEL A., THEISEL H., HEGE H. -C.: Trajectory density projection for vector field visualization. In *EuroVis: Short Papers 2013* (Leipzig, Germany, 2013), M. Hlawitschka and T. Weinkauff (Eds.), pp. 31–35.
- [KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: a gpu-based particle engine. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware* (Los Angeles, CA, USA, 2004), T. Akenine-Möller and M. McCool (Eds.), ACM, pp. 115–122.
- [KWL*04] KINDLMANN G. L., WEINSTEIN D. M., LEE A. D., TOGA A. W., THOMPSON P. M.: Visualization of anatomic covariance tensor fields. In *Conference of the IEEE Engineering in Medicine and Biology Society* (San Francisco, California, USA, 2004), vol. 1, pp. 1842–1845. <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1403548&tag=1>
- [KWP*10] KASTEN J., WEINKAUF T., PETZ C., HOTZ I., NOACK B. R., HEGE H. -C.: Extraction of coherent structures from natural and actuated flows. *Active Flow Control II* 108 (2010), 373–387.
- [Lau10] LAURITZEN A.: Deferred rendering for current and future rendering pipelines. *SIGGRAPH Course: Beyond Programmable Shading* (2010), 1–34.

- [LM10] LIPINSKI D., MOHSENI K.: A ridge tracking algorithm and error estimate for efficient computation of lagrangian coherent structures. *Chaos (Woodbury, N.Y.)* 20, 1 (2010), 017504.
- [Max88] MAX N. L.: Horizon mapping: Shadows for bump-mapped surfaces. *The Visual Computer* 4, 2 (1988), 109–117.
- [Mit07] MITTRING M.: Finding next gen: Cryengine 2. *ACM SIG-GRAPH Courses* (2007), 97–121.
- [MJL*13] MCLOUGHLIN T., JONES M., LARAMEE R., MALKI R., MASTERS I., HANSEN C.: Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 8 (2013), 1342–1353.
- [MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., CHEN M., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum: State of the Art Reports* 29 (2010), 1807–1829. <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2010.01650.x>
- [PF05] PHARR M., FERNANDO R.: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, MA, USA, 2005.
- [PFG05] POLICARPO F., FONSECA F., GAMES C.: Deferred Shading Tutorial. *Pontifical Catholic University of Rio de Janeiro* 31, (2005), 32.
- [SBF15] SCHOLLMAYER A., BABANIN A., FRÖHLICH B.: Order-independent transparency for programmable deferred shading pipelines. *Computer Graphics Forum* 34, 7 (2015), 67–76.
- [SGSM08] SALZBRUNN T., GARTH C., SCHEUERMANN G., MEYER J.: Pathline predicates and unsteady flow structures. *The Visual Computer* 24, 12 (2008), 1039–1051.
- [SH95] SUJUDI D., HAIMES R.: *Identification of swirling flow in 3D vector fields*. Tech. rep., Technical report, Department of Aeronautics and Astronautics, MIT, 1995. AIAA Paper 95-1715, Boston, USA, 1995.
- [SS06] SALZBRUNN T., SCHEUERMANN G.: Streamline predicates. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 12, 6 (2006), 1601–1612.
- [TCM06] TARINI M., CIGNONI P., MONTANI C.: Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 12, 5 (2006), 1237–1244.
- [TMWS13] TAO J., MA J., WANG C., SHENE C.: A unified approach to streamline selection and viewpoint selection for 3d flow visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 19, 3 (2013), 393–406.
- [ÜSK*12] ÜFFINGER M., SADLO F., KIRBY M., HANSEN C., ERTL T.: FTLE Computation beyond first-order Approximation. In *Eurographics (Short Papers)* (Cagliari, Sardinia, Italy, 2012), C. Andujar and E. Puppo (Eds.) pp. 61–64.
- [VBB*11] VAN PELT R., BESCOS J. O., BREEUWER M., CLOUGH R. E., GRÖLLER M. E., TER HAAR ROMENIJ B., VILANOVA A.: Interactive virtual probing of 4D MRI blood-flow. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 17, 12 (2011), 2153–2162.
- [WKM05] WALKER R., KENNY P., MIAO J.: Visualization of smoothed particle hydrodynamics for astrophysics. In *EG UK Theory and Practice of Computer Graphics* (2005), L. Lewer (Ed.), pp. 133–138.
- [XLS10] XU L., LEE T. -Y., SHEN H. -W.: An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 16, 6 (2010), 1216–1224.
- [ZB91] ZAGHA M., BLELLOCH G. E.: Radix sort for vector multi-processors. *Proceedings of the ACM/IEEE conference on Supercomputing* (1991), 712–721.