Eurographics/ IEEE-VGTC Symposium on Visualization 2008
A. Vilanova, A. Telea, G. Scheuermann, and T. Möller
(Guest Editors)

Volume 27 (2008), Number 3

Particle Level Set Advection for the Interactive Visualization of Unsteady 3D Flow

Nicolas Cuntz[1], Andreas Kolb[1], Robert Strzodka[2], and Daniel Weiskopf[3]

[1]University of Siegen, Germany
[2]Stanford University, Max Planck Center
[3]VISUS, Universität Stuttgart

Abstract

Typically, flow volumes are visualized by defining their boundary as iso-surface of a level set function. Grid-based level sets offer a good global representation but suffer from numerical diffusion of surface detail, whereas particle-based methods preserve details more accurately but introduce the problem of unequal global representation. The particle level set (PLS) method combines the advantages of both approaches by interchanging the information between the grid and the particles. Our work demonstrates that the PLS technique can be adapted to volumetric dye advection via streak volumes, and to the visualization by time surfaces and path volumes. We achieve this with a modified and extended PLS, including a model for dye injection. A new algorithmic interpretation of PLS is introduced to exploit the efficiency of the GPU, leading to interactive visualization. Finally, we demonstrate the high quality and usefulness of PLS flow visualization by providing quantitative results on volume preservation and by discussing typical applications of 3D flow visualization.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surface, solid, and object representations

1. Introduction

Today, large flow data sets are routinely generated by numerical simulation (computational fluid dynamics, CFD) or by experimental techniques such as PIV (particle imaging velocimetry). These data sets often need to be analyzed and explored visually for a good understanding of the data. Typical application areas include the aerospace and automotive industries, other engineering disciplines, and sciences. This paper addresses the challenge of visual mappings for unsteady 3D flow. We follow the strategy of dye advection—a well known and popular metaphor from experimental flow visualization. Dye advection facilitates user-centered exploration in the form of interactive control over seed points, and it can provide information about global flow behavior.

The goal of this paper is to improve the accuracy of interactive 3D dye advection and similar flow visualization techniques. Fig. 1 compares previous dye advection techniques with the technique of this paper. Most previous work on interactive dye advection [JEH02, vW02] is based on semi-Lagrangian advection on regular grids, which is affected by numerical diffusion due to repeated re-sampling via tri-linear reconstruction (Fig. 1a). The extension to level

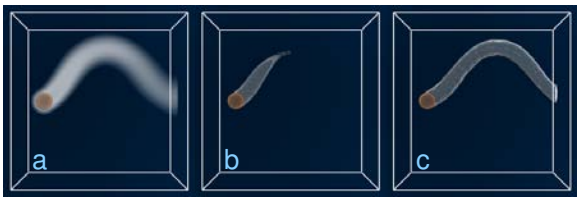

Figure 1: *Dye advection of a spherical volume in a time-dependent sine wave. The injection volume is marked in red. The flow spreads the dye over time and the resulting trace depends on the computation of this process. Diffusive advection (a), level set advection (b), PLS advection (c) – (64^3 grid, 262,144 particles, rendering speed: 78.2, 55.9, 36.6 FPS).*

set based dye advection (Fig. 1b) eliminates the diffusion problem, but leads to inaccuracies in the form of loss of dye volume. To overcome these problems, we propose a particle level set (PLS) method for 3D advection (Fig. 1c). The original PLS method [EMF02] combines a grid-based level set with particle-based tracking to reduce numerical diffusion and volume loss. Marker particles placed near the interface are used to correct the level set representation.

This paper provides the following contributions. First, we adopt the PLS method for the visualization of 3D unsteady flow by extending it to the representation of streak volumes, path volumes (3D analogues to streak lines and path lines), and time surfaces. Streak volumes correspond to the advection of dye. Second, we introduce a fast dye injection mechanism for the hybrid grid/particle representation of PLS. Third, we present an efficient GPU mapping of the PLS approach by utilizing a fine-grained parallelization of the algorithm. Fourth, a new sub-voxel description of the interface can be used to reduce the volume loss of the traditional PLS method. The main benefit of our method is the interactive, accurate visualization of unsteady 3D flow inspired by well-known metaphors like dye advection.

2. Related Work

An early example of stream volumes—the volumetric equivalent to stream lines—is described by Max et al. [MBC93], who use an explicit representation of the volume based on tetrahedra. Unfortunately, explicit representations are difficult for intricate flow because adaptive removal and addition of vertices and changes of topology need to be considered. Even the simpler problem of stream surfaces already requires advanced algorithms to handle these issues [Hul92, GTS*04]; point-based representations of stream surfaces and path surfaces avoid issues of mesh connectivity but still require complicated point generation and removal [STWE07]. In contrast, implicit representations easily allow for topology changes and do not require control of vertex density. Examples include implicit stream surfaces by Van Wijk [vW93], the particle travel time method by Westermann et al. [WJE00], and the application of direct volume rendering to visualizing implicit representations according to Xue et al. [XZC04]. Texture advection is the most popular example of implicit representations for flow visualization due to its high visualization speed, efficient mapping to GPUs, and easy implementation. Texture advection transports dye or similar visual information stored on a regular grid (i.e., the texture) and uses that information for visualization; the information on the texture is typically displayed "as is", without an explicit reconstruction of the implicit surface. Basic texture advection [MB95] can be modified in the form of 2D Image Based Flow Visualization (IBFV) [vW02] and 2D Lagrangian-Eulerian Advection (LEA) [JEH02], which both support texture-based dye advection in order to generate streaklines. Texture advection can be extended to fast 3D GPU algorithms [TvW03, WSE07] for 3D flow visualization. For example, 3D dye visualization can be employed to highlight features [SJM96]. One issue of most texture advection methods is numerical diffusion due to resampling (see discussion in [Wei04]). LEA [JEH02] addresses this problem by frequently restoring the contrast of the transported dye. An alternative approach is the use of distance-field level sets in combination with level set reinitialization, which leads to a non-diffused dye–background interface but is affected by volume loss [Wei04]. For an overview of texture-based flow visualization in general, we refer to Laramee et al. [LHD*04].

Generic level set methods are often applied in the field of visualization and image processing. The first GPU implementation of the level set equation is due to Rumpf and Strzodka [RS01]. Lefohn et al. [LKHW04] additionally incorporate an adaptive memory model for narrow band techniques. An iterative solution to the level set equation is presented by Griesser et al. [GRNG05]. In these applications, numerical dissipation is no problem because, in fact, a smooth boundary of 3D regions is desirable. In contrast, our goal is to minimize numerical diffusion and volume loss in order to achieve high quality visualization of crisp streak volumes, path volumes, and time surfaces. To this end, we include a particle-based correction of the level set according to the PLS idea. The original PLS technique is described by Enright et al. [EMF02]; a method to reduce the order of the advection scheme is presented by Enright et al. [ELF04], which is available in an open source library [MF06]. The main goal of this paper is to extend PLS to allow for dye advection and similar flow visualization methods, which especially requires a dye injection mechanism. In addition, we aim at making PLS interactive by utilizing an efficient GPU mapping.

PLS methods employ a reinitialization of the level set function, which requires the construction of a Euclidean 3D distance field. Distance field computation is a well studied problem (see [Cui99] for an overview). Depending on the initial object representation (using a regular grid or an explicit geometric representation), different approaches have been proposed. Propagation methods for regular grids iteratively propagate the distance information to the neighboring grid points, either by spatial sweeping or by contour propagation. Different parallel approaches for the computation of a distance transform for a set of sites have been proposed, for 2D pixel sites [ST04, RT06] and for 3D polygonal input data [SPG03, SGGM06]. For our PLS framework, we employ a 3D variant of the jump flooding approach [RT06] that relies on hierarchical propagation to balance speed and accuracy [CK07].

3. Particle Level Set Method

In this section, the original particle level set method by Enright et al. [ELF04] is explained. The key idea of the level

set method is the representation of the lower dimensional interface I in a domain D by the iso-contour $I(\phi) := \{x \in D | \phi(x) = 0\}$ of the level set function $\phi : D \to \mathbb{R}$. The interface is moved by evolving ϕ within a velocity field. Typically, ϕ is initialized to be a signed distance field. However, after advection, this property might be lost, and ϕ needs to be reinitialized, i.e. the distance field is recomputed.

Enright et al. [ELF04] use a fast first order accurate semi-Lagrangian method to evolve ϕ in a grid. At first this leads to an accumulation of numerical diffusion, resulting in a considerable volume loss (see Fig. 6, middle). Their PLS method aims to prevent this by Lagrangian tracing of corrective particles placed nearby the interface $I(\phi)$. Positive particles are located in the $\phi > 0$ region and negative particles in the $\phi < 0$ region. Each particle is defined as a sphere around $\mathbf{x}_p$ with radius $r_p \in [r_{\min}, r_{\max}]$ touching the interface, thus $r_p = s_p \phi(\mathbf{x}_p)$, where s_p is the sign of the particle. The correction step involves the definition of a temporary level set function ϕ_p around each particle:

$$\phi_p(\mathbf{x}) = s_p(r_p - \|\mathbf{x} - \mathbf{x}_p\|). \tag{1}$$

After level set and particle advection, *escaped* particles, i.e. those that are further away than their radius on the wrong side of the interface, are used for the level set correction. Each escaped particle p contributes to the eight surrounding grid points through intermediate level set functions ϕ^+ and ϕ^- that are initialized to ϕ and updated according to the formulas:

$$\begin{aligned} \phi^+(\mathbf{x}) &\leftarrow \max(\phi_p(\mathbf{x}),\ \phi^+(\mathbf{x})), \\ \phi^-(\mathbf{x}) &\leftarrow \min(\phi_p(\mathbf{x}),\ \phi^-(\mathbf{x})) \end{aligned} \tag{2}$$

After processing all escaped particles, a new (corrected) ϕ is constructed according to the following operation and then reinitialized in order to restore a signed distance function.

$$\phi(\mathbf{x}) = \begin{cases} \phi^+(\mathbf{x}) & \text{if} \quad \|\phi^+(\mathbf{x})\| \leq \|\phi^-(\mathbf{x})\| \\ \phi^-(\mathbf{x}) & \text{else} \end{cases} \tag{3}$$

The PLS method is known to produce good results even when performing a first order semi-Lagrangian level set advection. The algorithm according to Enright et al. [ELF04] is summarized below. Note that the level set correction is performed twice.

Algorithm 1 (PLS algorithm)
1. Definition of the interface location and velocity field
2. Initialization of the level set based on the interface
3. First order semi-Lagrangian level set advection
4. Second order Runge-Kutta particle advection
5. Correction of the level set function using the particles
6. Level set reinitialization
7. Correction of the level set function using the particles
8. Particle reseeding
9. Go to 3

4. Flow Volume Particle Level Sets

The task of applying PLS to interactive flow visualization poses several challenges:

- Fast parallel algorithms are necessary for level set reinitialization, particle reseeding, and the interchange of data between grids and particles.
- There is a trade-off between speed and accuracy. Accuracy in the context of PLS means no volume loss, a smooth surface, and the preservation of surface features.
- Time surfaces, path, and streak volumes require different handling of the grid and the particle structure and thus have to be considerated separately.
- Streak volumes (i.e. dye advection) require special attention in order to synchronize the grid and the particle structure.

These tasks are addressed in the following sections. As our modifications remain in the spirit of the original PLS, we will refer to the steps in Alg. 1. We assume some familiarity with the use of graphics hardware. For an introduction to GPU programming, see [Buc05, Har05].

4.1. Data Structures

The level set reinitialization algorithm used in our approach (Sec. 4.2) produces a distance transform (DT). On the one hand, the use of a DT is motivated by our propagation-based reinitialization itself, on the other hand, the references stored in the DT can be used for a simple yet efficient particle reseeding (Sec. 4.4) with sub-voxel accurate radii. These advantages compensate the higher memory consumption for storing the reference data.

The DT $\mathbf{dt}(\mathbf{x}) = (\mathbf{dt}_d(\mathbf{x}), \mathbf{dt}_\delta(\mathbf{x}))$ is stored in a grid. For each point $\mathbf{x}$, $\mathbf{dt}_d(\mathbf{x}) = \phi(\mathbf{x})$ is the signed distance and $\mathbf{dt}_\delta(\mathbf{x})$ is a reference to the nearest interface location. Note that in some stages of the algorithm, e.g. after the level set advection, the grid may not represent a precise or complete distance transform.

Internally, we represent $\mathbf{dt}$ as a tiled 4-component 2D float texture. We use frame buffer objects for render-to-texture functionality. Double buffering is used to separate input from output data in the data parallel GPU processing. The velocity field can be given in any form that allows arbitrary sampling, e.g. in a 3D texture. The particle system is held in a 2D float texture (preferably in 32-bit format), storing the position $\mathbf{x}_p$ and the radius r_p of each particle p.

The data flow with the corresponding steps of Alg. 1 is outlined in Fig. 2. In- and outgoing edges represent data input and output, respectively. The error correction (step 5) involves two passes and is split into two separate entities 5a and 5b. We omit the repeated particle correction (step 7) to improve performance with negligible loss of overall accuracy.

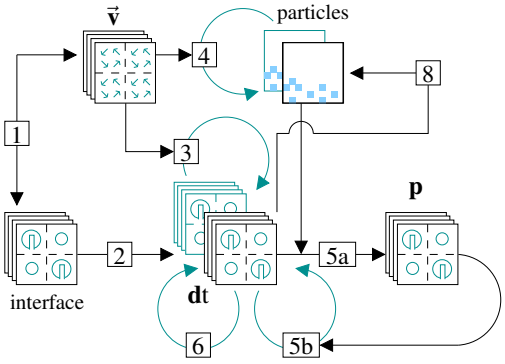

Figure 2: *The data flow in the GPU-based PLS framework. The use of double buffering is marked by cyclic, blue arrows. Numbers indicate steps in Alg. 1. (Step 7 is omitted.)*

4.2. Level Set Reinitialization

Enright et al. [ELF04] solve the problem of level set reinitialization (Alg. 1, steps 2 and 6) via the fast marching method [Set99]. There is no feasible way to do this on the GPU efficiently. Instead, we use a propagation method based on a DT, which is sufficiently fast for per-frame reinitialization.

Before reinitialization, $\mathbf{dt}$ contains only the interface, which is given by means of the grid-based classification (interior/exterior) or by a gray-level function (e.g. the advected $\mathbf{dt}_d$, defining the interface on a sub-voxel level).

First, the interface needs to be identified, i.e. references are assigned to interface grid points. Those points are identified by searching neighbor points $\mathbf{x}$ and $\mathbf{y}$ with a different sign in the distance: $\text{sign}(\mathbf{dt}_d(\mathbf{x})) \neq \text{sign}(\mathbf{dt}_d(\mathbf{y}))$. This check is performed for each point $\mathbf{x}$ in a fragment program, where $\mathbf{y}$ is a point in a 6-neighborhood of $\mathbf{x}$ containing all direct non-diagonal neighbors. If one point $\mathbf{y}$ is found, the reference $\mathbf{dt}_\delta(\mathbf{x})$ is updated by moving along the gradient $\hat{\mathbf{n}}$ at position $\mathbf{x}$:

$$\mathbf{dt}_\delta(\mathbf{x}) \leftarrow \mathbf{x} - \mathbf{dt}_d(\mathbf{x}) \cdot \hat{\mathbf{n}} \tag{4}$$

The gradient is computed by central differences in the 6-neighborhood of $\mathbf{x}$. The level set value $\mathbf{dt}_d(\mathbf{x})$ is set to the distance between $\mathbf{x}$ and $\mathbf{dt}_\delta(\mathbf{x})$, multiplied by the sign previously stored at position $\mathbf{x}$. The sub-voxel references in $\mathbf{dt}_\delta(\mathbf{x})$ can now be used for the reinitialization of $\mathbf{dt}(\mathbf{x})$.

We do the reinitialization using the fast hierarchical algorithm proposed in [CK07], which extends the jump flooding algorithm [RT06]. Reference propagation can be implemented as a fragment program, where $\mathbf{dt}(\mathbf{x})$ is updated by computing distances according to the reference points of neighboring grid points.

4.3. Particle Reseeding

For an efficient PLS correction (Alg. 1, step 5), all particles should be located near the interface (see Sec. 3). This, however, is difficult to achieve in parallel on the GPU, since one cannot iterate over the interface neighborhood and place particles accordingly. Moving the particles to the interface is not only necessary during the initialization in the beginning, but also occasionally during the execution of the algorithm, because more and more particles will drift away from the interface with time evolving. Frequent particle reseeding has the negative side effect that inaccuracies in $\mathbf{dt}_\delta$ are constantly transferred into the new set of particles, thus annihilating the advantage gained by the particle correction. According to [EMF02], a reasonable trade-off is to reposition the particles every 20 time steps on average, e.g. to reposition 5 percent of all particles in each iteration.

Thanks to the level set representation as a DT, the interface location is known at any volume position. First, the initial location $\mathbf{x}_p$ for each particle p is chosen randomly within $[0,1]^3$. Afterward, the particle is pushed towards the interface by following the reference and adding a random offset:

$$\mathbf{x}_p \leftarrow \mathbf{dt}_\delta(\mathbf{x}_p) + \varepsilon \cdot \hat{\mathbf{v}}_{\text{rand}}, \tag{5}$$

where $\hat{\mathbf{v}}_{\text{rand}}$ is a normalized random vector pointing in an arbitrary direction and ε is a predefined small constant scalar value. This way, we produce inner and outer particles surrounding the interface. In our implementation, conforming with [ELF04], we choose ε such as to place particles around the interface within a band that is a few grid cells wide. The random vector can be fetched from a texture with precomputed random values.

Pushing the particles towards the interface according to the above scheme can lead to sparsely populated regions, especially in areas with high curvature. In our experiments, this effect could always be sufficiently reduced by using a higher number of particles, as the surface represented by the level set is bound by the number of discrete grid points.

4.4. Particle Radius

During reseeding, the radius r_p of each particle p must be determined. Two methods for radius sampling have been tested. The first one samples the distance stored in $\mathbf{dt}_d(\mathbf{x}_p)$ by doing tri-linear interpolation. The other approach looks for the nearest reference in a neighborhood $\mathcal{N}$ consisting of 8 grid points around $\mathbf{x}_p$:

$$r_p \leftarrow s_p \cdot \min_{\mathbf{x}' \in \mathcal{N}(x_p)} \{ \left\| \mathbf{dt}_\delta(\mathbf{x}') - \mathbf{x}_p \right\| \}, \tag{6}$$

The sign s_p of the particle is determined by the sign $\mathbf{dt}_d$ for the corresponding grid point. Note that there is no obvious way to obtain the sign on a sub-voxel level as it is the case for the radius.

Fig. 3 (left) shows different results for both methods after 8 rotations involving 100 advection and reseeding steps. One can observe a much higher volume loss when interpolating the distance, while the nearest reference (Eq. 6) leads to a more fluctuating, less smooth surface. With no particles at all, the volume completely disappears after 6 rotations due to numerical diffusion. In Fig. 3 (middle and right), the grid points (red points) contain the length of the normals (red lines) to the interface (blue). In Fig. 3 (2a, 2b), the distance at the particle position (black point) is computed with a bi-linear interpolation of the cell distances. In case of a convex interface the resulting distance (green circle) is too small, because distances corresponding to very different normals are averaged. In Fig. 3 (3a, 3b), the nearest of the reference points is taken (yellow points). Here, we also commit an error, as the selected reference point is not exactly the closest point on the interface to the particle (black point), but for highly convex interface parts, this is more accurate than the interpolation. For (almost) flat interface parts the interpolation is better because all normals point in (almost) the same direction.

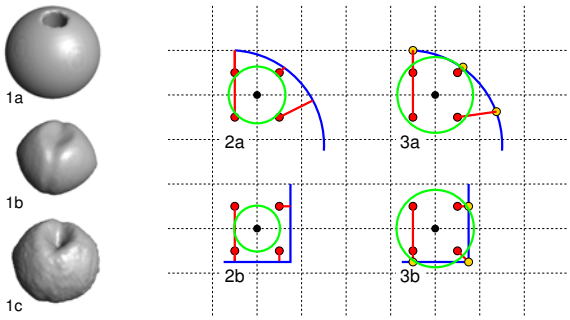

Figure 3: *Comparing particle radii using tri-linear distance interpolation and neighborhood-sampling the nearest reference point. Left: A notched sphere (1a) is rotated 8 times, with reseeding 5 percent of the particles in each step using interpolated radii (1b) and nearest reference (1c). Middle and right: Two (2D) cases where interpolation (2a and 2b) leads to a larger error than nearest reference (3a and 3b).*

In [ELF04], the radii of the particles are reset after level set reinitialization while Mokberi and Faloutsos [MF06] omit this step. The reason for this is similar to the explanation why particle reseeding after each frame should be avoided: as particle radii have to be reset with information stored in the level set, inaccuracies of the level set are propagated into the particle model. Thus, it is reasonable to wait until the next particle reseeding before the radii are updated, and we follow this approach.

4.5. Level Set Advection and Particle Tracing

Level set (LS) advection (Alg. 1, step 3) is easily ported to the GPU due to its parallel nature. New level set values $\mathbf{d}t_d$ are computed according to a semi-Lagrangian approach. For particle tracing (Alg. 1, step 4), the same time step is used in a second order accurate Runge-Kutta integration.

As particle tracing is an operation on all particles, this is a good opportunity for choosing the subset of particles that will correct the level set. Those particles are marked by changing the sign of the first component of the position $\mathbf{x}_p$. As the volume is bounded by $[0,1]^3$, it is then easy to distinguish the position and the marker bit. In contrast to the original PLS approach (Sec. 3), we refine the particle correction by involving more particles in the level set correction: a particle contributes if its radius is greater than its distance to the interface. Following this rule, fewer particles are necessary in order to produce satisfactory results in our examples.

4.6. Level Set Correction

For level set correction (Alg. 1, step 5), we need a way to select particles marked as *escaped* and, based on this selection, to construct the intermediate level set functions ϕ^+, ϕ^-. To reduce bandwidth requirements we do this differently than in the original formulation, postponing the combination with ϕ until the final update.

The construction of ϕ^+, ϕ^- requires particle-to-grid coupling by scattering into the grid data structure. The key idea is to render a marker-geometry at the particle positions to trigger a computation for the 8 grid cells around the particle. In order to process all particles, the particle texture containing $\mathbf{x}_p$ and r_p is copied into a Vertex Buffer Object. Then, all particles are sent through the graphics pipeline and a vertex program detects whether a particle p has escaped by checking the marker bit (see Sec. 4.5). If the particle has not escaped, it does not contribute to the level set correction and is discarded by moving it outside of the volume.

Point scattering approaches are known to be rather time-consuming, e.g. Kolb and Cuntz [KC05] could handle only a few thousand particles at interactive rates, when using relatively large point sprites. After evaluating different types of marker-geometries, i.e. point sprites, quads, and individual points, we conclude that rendering four individual points into two subsequent slices of the grid turns out to be the most effective variant in our situation.

The intermediate level sets ϕ^+ and ϕ^- given in Eq. 1 are stored in a two-component grid $\mathbf{p} = (\phi^+, -\phi^-)$. The accumulation of particle contributions in $\mathbf{p}$ uses min-max blending. Initially, $\mathbf{p}$ is set to $(-\infty, -\infty)$ for all grid cells. Storing $-\phi^-$ instead of ϕ^- allows a single pass update of $\mathbf{p}$ using maximum blending only.

A second pass rasterizes the complete level set, computing the corrected level set $\mathbf{d}t_\delta$ using $\mathbf{p}$ according to Eq. 3, including the postponed combination with ϕ.

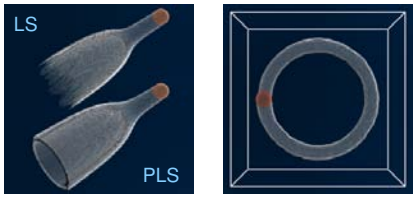

Figure 4: *Left: A path volume is pushed into a tight band around a flow source. Due to numerical diffusion, the LS volume disappears . In contrast, PLS maintains the volume. Right: A difficult case where the streak volume reaches the injection volume. Both: The injection is marked in red.*

4.7. Time Surfaces

So far, the algorithm includes all steps required for the visualization of time surfaces, i.e. a bounded volume moving in a flow while possibly changing its topology.

4.8. Path Volumes

Path volumes can be interpreted as the accumulation of the volume left behind by a time surface. This idea fits well into the presented PLS algorithm. We accumulate the path volume in an additional step after step 6 in Alg. 1 into a separate grid $\mathbf{dt}^{\text{acc}}$. The accumulated grid is the result of a minimum operation on the distance component of the previous $\mathbf{dt}^{\text{acc}}$ and of the current $\mathbf{dt}$, taking the according reference $\mathbf{dt}_\delta$. The result of this operation is a union of the negative distances, thus of the inner regions of both level sets. An example is given in Fig. 4.

4.9. Streak Volumes

Streak volumes are produced by repeatedly injecting a volume $\mathbf{dt}^{\text{inj}} = (\mathbf{dt}_d^{\text{inj}}, \mathbf{dt}_\delta^{\text{inj}})$ to the current level set. The injection $\mathbf{dt}^{\text{inj}}$ is generated analytically using implicit geometries. The volume injection involves an update of both the grid and the particle representation of the level set.

The grid update is similar to path volume accumulation. A minimum of $\mathbf{dt}^{\text{inj}}$ and $\mathbf{dt}$ computes the union of both volumes after step 5 of Alg. 1. After step 6, the particle set is extended in order to cover the new interface added by the injection. Two methods have been tested: 1. explicitly adding new particles at the injection's interface or 2. relying on the standard particle reseeding. In both cases, the scheme presented in Sec. 4.3 can be adapted by binding the appropriate DT texture. The second approach yields nearly as good results as the first one in our examples, despite the fact that the particle density near the injection is lower. The reason is that the injected volume is less susceptible to numerical diffusion because it is re-emitted in each frame.

Both injected particles and old particles can disturb the PLS correction. In Fig. 5, those wrong particles are marked in red. The green circle stands for the injection volume, which partly overlaps the advected volume (blue circle) created in the last level set advection step.

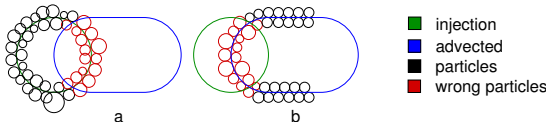

Figure 5: *Two cases (in 2D) where particles must be removed or reseeded when generating streak volumes ((a): injected particles, (b): old particles).*

Fig. 5a shows the situation when injecting new particles. Wrong particles are those with $\mathbf{dt}_d(\mathbf{x}_p) < r_p$. They can be efficiently reseeded during the injection step by using the unified DT. Fig. 5b shows wrong particles coming from the last PLS step. They can be identified by checking $\mathbf{dt}_d^{\text{inj}}(\mathbf{x}_p) < r_p$ in a separate pass over all particles. The identified particles are either removed or reseeded. Removing them is cheaper and can be achieved by moving them out of the volume, However in some examples, too many particles might be lost in this approach.

Fig. 1 and Fig. 8 show examples of streak volumes. The right side of Fig. 4 shows that our reseeding scheme even handles the difficult case where the streak volume reaches the injection area.

4.10. Rendering

The volume renderer is based on a back-to-front slicing technique using view-aligned polygons. Only fragments within a small iso-value range $[-\varepsilon, \varepsilon]$ around the interface are colored in a shader using Phong lighting. Applying alpha blending makes internal structures visible, which is important e.g. for complex flow volumes. This approach is similar to semi-transparent interval volume rendering [FMST96]. When using appropriate semi-transparent transfer functions, the visual results of rendering PLS resemble those of illustrative techniques that highlight 3D flow structure, such as [SJEG05], which supports the spatial perception of flow volume boundaries.

5. Results

In the following, our flow PLS is evaluated by presenting some examples as well as performance and volume preservation results. The hardware used for testing is a PC with an AMD Athlon 64 X2 Dual Core Processor 4200+ (2.21 GHz), 4 GB RAM with a GeForce 8800 GTS graphics chip. For evaluation, we took Zalesak's sphere (Fig. 6), which is a 3D version of the disc used in [ELF04]. Our evaluation involves grids up to 128^3 and max. 4 million particles due to limited GPU memory.

Figure 6: *360° rotation of Zalesak's sphere (100 advections): initial, LS, and PLS. 524,288 particles, 128^3 grid, 14.74 FPS.*

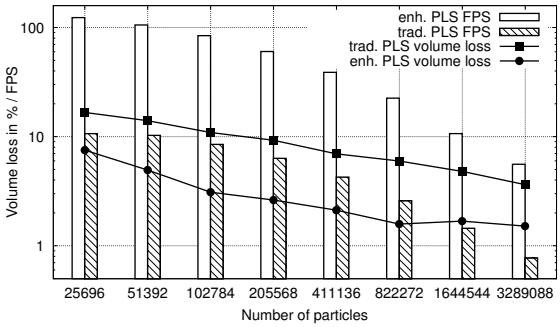

Figure 7: *Comparison of traditional PLS (trad.) and our method (enh.) for Zalesak's sphere (same logarithmic scale for both) – 64^3 grid, no particle reseeding*

We compare our GPU method with the PLS library [MF06] using exactly the same Zalesak's sphere (see Fig. 6) and with equal parameters for both implementations. The test consists of 100 evolution steps in a vortex flow field to a total of 360°. Fig. 7 shows the resulting frame-rate and relative volume loss as function of the number of particles. Following prior work (e.g. Enright et al. [ELF04]), the resulting volume loss is measured by counting the number of interior grid points of the object.

Table 1 lists the time consumption for all steps of our GPU-based algorithm separately for Zalesak's sphere. One can see that, depending on the resolution, the level set reinitialization and correction are the most time-consuming steps of the application. Due to the large size of the kernel used in the propagation method, the level set reinitialization is texture-fetch-bound. The additional reseeding for streak volumes is in the same range as general reseeding. Dye injection takes about 5 percent, dye reseeding about 2 percent of the time for the overall algorithm in the example shown in Fig. 1c.

Fig. 8 shows the results of a time surface and a streak volume in an unsteady flow representing a typhoon. The data set (courtesy of DKRZ Hamburg) is stored in 32 time steps as $106 \times 53 \times 39$ textures. Both rows compare the pure LS and our PLS method, showing the advantage of PLS when using the same grid resolution.

The typhoon data set exhibits prominent swirling features,

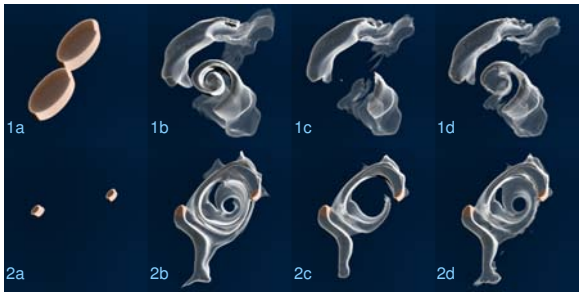

Figure 8: *Top row: Time surface after 190 evolutions. Starting geometry (1a), ground-truth using 128^3 LS (1b), 64^3 LS (1c), our enhanced 64^3 PLS (1d), 262,144 particles (FPS for 1b–1d: 10.6, 19.1, 16.5, time without rendering (in ms): 51.6, 11.1, 22.1). Second row: Streak volume after 275 advections, same parameters (FPS for 2b–2d: 12.5, 28.2, 22.1, time without rendering (in ms): 52.1, 10.1, 19.7). Both: The starting/injection shape is marked in red.*

Table 1: *Run-time of the steps involved in the PLS algorithm. Object: Zalesak's sphere (see Fig. 6), 262,144 particles. In each frame, 5 percent of the particles are reinitialized.*

step	grid / time (ms)	grid / time (ms)
framework	64^3 / 1.3	128^3 / 1.3
LS advection	64^3 / 0.15	128^3 / 1.15
particle tracing	64^3 / 0.05	128^3 / 0.25
LS reinitialization	64^3 / 5.83	128^3 / 40.65
LS correction	64^3 / 10.8	128^3 / 12.88
particle reseeding	64^3 / 0.65	128^3 / 0.475

which can be depicted by both time surfaces and streak volumes. The latter are particularly useful because they resemble the well known dye advection metaphor from experimental flow visualization and they show the temporal evolution of the flow. Figure 8 demonstrates that the high quality and volume preservation of the PLS approach is critical for showing all details of the swirling features, whereas LS fails to depict those details.

6. Conclusions

We have presented a modified and enhanced GPU-based PLS method for flow volumes. The presented method shows that surface evolution can be performed efficiently and accurately on the GPU. We achieve convincing performance and superior quality of results over both, CPU-based PLS methods and grid-only GPU-methods. Examples of accurate and interactive flow visualizations, including time surfaces, path volumes, and streak volumes, have been presented. Our evaluation involves both analytical flows as well as the typhoon data set as an example of a realistic unsteady flow.

References

[Buc05] BUCK I.: Taking the plunge into GPU computing. In *GPU Gems 2*, Pharr M., (Ed.). Addison Wesley, 2005, ch. 32, pp. 509–519.

[CK07] CUNTZ N., KOLB A.: Fast hierarchical 3D distance transforms on the GPU. In *Proc. Eurographics (short paper)* (2007), pp. 93–96.

[Cui99] CUISENAIRE O.: Distance transformations: Fast algorithms and applications to medical image processing. *Ph.D. Thesis, UCL, Louvain-la-Neuve, Belgium* (1999).

[ELF04] ENRIGHT D., LOSASSO F., FEDKIW R.: A fast and accurate semi-Lagrangian particle level set method. *Computers & Structures, 6-7* (2004), 479–490.

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Trans. on Graphics 21*, 3 (2002), 736–744.

[FMST96] FUJISHIRO I., MAEDA Y., SATO H., TAKESHIMA Y.: Volumetric data exploration using interval volume. *IEEE Trans. on Visualization and Computer Graphics 2*, 2 (1996), 144–155.

[GRNG05] GRIESSER A., ROECK S. D., NEUBECK A., GOOL L. V.: GPU-based foreground-background segmentation using an extended colinearity criterion. In *Proc. Vision, Modeling and Visualization* (2005), pp. 319–326.

[GTS*04] GARTH C., TRICOCHE X., SALZBRUNN T., BOBACH T., SCHEUERMANN G.: Surface techniques for vortex visualization. In *Proc. EG/IEEE VGTC Symp. on Visualization (VisSym 2004)* (2004), pp. 155–164.

[Har05] HARRIS M.: Mapping computational concepts to GPUs. In *GPU Gems 2*, Pharr M., (Ed.). Addison Wesley, Mar. 2005, ch. 31, pp. 493–508.

[Hul92] HULTQUIST J. P. M.: Constructing stream surfaces in steady 3D vector fields. In *Proc. IEEE Conf. on Visualization* (1992), pp. 171–178.

[JEH02] JOBARD B., ERLEBACHER G., HUSSAINI M. Y.: Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *IEEE Trans. on Visualization and Computer Graphics 8*, 3 (2002), 211–222.

[KC05] KOLB A., CUNTZ N.: Dynamic particle coupling for GPU-based fluid simulation. In *Proc. 18th Symp. on Simulation Technique* (2005), pp. 722–727.

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., VROLIJK B., POST F. H., WEISKOPF D.: The state of the art in flow visualization: Dense and texture-based techniques. *Computer Graphics Forum 23* (2004), 203–222.

[LKHW04] LEFOHN A., KNISS J., HANSEN C., WHITAKER R.: A streaming narrow-band algorithm: Interactive computation and visualization of level-set surfaces. *IEEE Trans. on Visualization and Computer Graphics 10*, 4 (2004), 422–433.

[MB95] MAX N., BECKER B.: Flow visualization using moving textures. In *Proc. ICASW/LaRC Symp. on Visualizing Time-Varying Data* (1995), pp. 77–87.

[MBC93] MAX N., BECKER B., CRAWFIS R.: Flow volumes for interactive vector field visualization. In *Proc. IEEE Conf. on Visualization* (1993), pp. 19–24.

[MF06] MOKBERI E., FALOUTSOS P.: A particle level set library. *Technical Report* (2006). http://www.magix.ucla.edu/software/levelSetLibrary/.

[RS01] RUMPF M., STRZODKA R.: Level set segmentation in graphics hardware. In *Proc. of IEEE International Conference on Image Processing (ICIP'01)* (2001), vol. 3, pp. 1103–1106.

[RT06] RONG G., TAN T.-S.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proc. ACM Symp. on Interactive 3D Graphics and Games* (2006), pp. 109–116.

[Set99] SETHIAN J. A.: *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1999.

[SGGM06] SUD A., GOVINDARAJU N., GAYLE R., MANOCHA D.: Interactive 3D distance field computation using linear factorization. In *Proc. Symp. on Interactive 3D Graphics and Games* (2006), pp. 117–124.

[SJEG05] SVAKHINE N. A., JANG Y., EBERT D., GAITHER K.: Illustration and photography inspired visualization of flows and volumes. In *Proc. IEEE Conf. on Visualization* (2005), pp. 687–694.

[SJM96] SHEN H.-W., JOHNSON C. R., MA K.-L.: Visualizing vector fields using line integral convolution and dye advection. In *Proc. 1996 Volume Visualization Symp.* (1996), pp. 63–70.

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proc. IEEE Conf. on Visualization* (2003), pp. 12–19.

[ST04] STRZODKA R., TELEA A.: Generalized distance transforms and skeletons in graphics hardware. In *Proc. EG/IEEE VCTC Symp. on Visualization (VisSym)* (2004), pp. 221–230.

[STWE07] SCHAFHITZEL T., TEJADA E., WEISKOPF D., ERTL T.: Point-based stream surfaces and path surfaces. In *Proc. Graphics Interface* (2007), pp. 289–296.

[TvW03] TELEA A., VAN WIJK J. J.: 3D IBFV: Hardware-accelerated 3D flow visualization. In *Proc. IEEE Conf. on Visualization* (2003), pp. 233–240.

[vW93] VAN WIJK J.: Implicit stream surfaces. In *Proc. IEEE Conf. on Visualization* (1993), pp. 245–252.

[vW02] VAN WIJK J. J.: Image based flow visualization. *ACM Trans. on Graphics 21*, 3 (2002), 745–754.

[Wei04] WEISKOPF D.: Dye advection without the blur: A level-set approach for texture-based visualization of unsteady flow. *Computer Graphics Forum (Eurographics 2004) 23*, 3 (2004), 479–488.

[WJE00] WESTERMANN R., JOHNSON C., ERTL T.: A level-set method for flow visualization. In *Proc. IEEE Conf. on Visualization* (2000), pp. 147–154.

[WSE07] WEISKOPF D., SCHAFHITZEL T., ERTL T.: Texture-based visualization of unsteady 3D flow by real-time advection and volumetric illumination. *IEEE Trans. on Visualization and Computer Graphics 13*, 3 (2007), 569–582.

[XZC04] XUE D., ZHANG C., CRAWFIS R.: Rendering implicit flow volumes. In *Proc. IEEE Conf. on Visualization* (2004), pp. 99–106.