Interactive Particle Tracing in Time-Varying Tetrahedral Grids

M. Bußler[†], T. Rick[‡], A. Kelle-Emden[§], B. Hentschel, and T. Kuhlen

Virtual Reality Group, RWTH Aachen University

Abstract

Particle tracing methods are a fundamental class of techniques for vector field visualization. Specifically, interactive particle advection allows the user to rapidly gain an intuitive understanding of flow structures. Yet, it poses challenges in terms of computational cost and memory bandwidth. This is particularly true if the underlying data is time-dependent and represented by a series of unstructured meshes. In this paper, we propose a novel approach which maps the aforementioned computations to modern many-core compute devices in order to achieve parallel, interactive particle advection. The problem of cell location on unstructured tetrahedral meshes is addressed by a two-phase search scheme which is performed entirely on the compute device. In order to cope with limited device memory, the use of data reduction techniques is proposed. A CUDA implementation of the proposed algorithm is evaluated on the basis of one synthetic and two real-world data sets. This particularly includes an assessment of the effects of data reduction on the advection process' accuracy and its performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors I.6.6 [Computer Graphics]: Simulation and Modeling—Simulation Output Analysis J.2 [Computer Applications]: Physical Science and Engineering—Engineering

1. Introduction

Particle tracing methods, i.e. the computation of the trajectories of massless particles, form one of the major classes of vector field visualization techniques. While the general idea and the mathematical formulation of particle tracing are straightforward, a realization on reasonably large, unstructured, and time-varying data is rather challenging. On the one hand, one can exploit the parallel nature of trajectory computation for large populations of particles. On the other hand, an efficient solution has to address the problems of efficient cell location and limited data bandwidth. In this paper, we describe a parallel, many-core approach to interactive particle tracing on time-dependent, unstructured, tetrahedral meshes and investigate the impact of data reduction on particle tracing computations.

When dealing with unstructured data, two main prob-lems arise. First, the connectivity information for unstructured grids has to be stored explicitly, driving up the overall memory consumption. In addition to this, cell location – a frequent operation in particle advection – becomes a major issue. Similar to others, we use a two-stage search scheme, consisting of a global and local search. Global search schemes based on hierarchical data structures, e.g. kd-trees, are known to reach device limitations in terms of data structures (stacks) and compute kernel complexity. In this paper, we specifically address the impact – with respect to accuracy and complexity – of the global search on the local search task to construct lightweight compute kernels for an efficient device occupation.

Second, the handling of time-varying data greatly increases memory as well as bandwidth requirements. While single time steps of a simulation may fit into a well-furnished workstation's memory, a series of 100 or more time steps can easily congest even very high-end visualization systems. Moreover, the data does not only need to be stored but also has to be transferred between the different stages in the computation process. If accelerator hardware is used to speed up

---

[†] michael.bussler@rwth-aachen.de

[‡] corresponding author: rick@vr.rwth-aachen.de

[§] {kelle-emden,hentschel,kuhlen}@vr.rwth-aachen.de

the overall computation, this particularly impacts the interface between host and device memory. To this end, we investigate the use of state-of-the-art data reduction techniques in order to reduce overall data set size while maintaining reasonable precision.

We summarize the contributions of this work as follows:

- After discussing the performance and precision impact of several implementation choices, we identify an efficient solution to the cell location problem which runs entirely on the compute device.
- We investigate the effect of data reduction techniques for particle tracing to address device memory limitations.

The remainder of this paper is structured as follows. We briefly review related work in the next section. We will then discuss the general algorithm for many-core particle tracing in Section 3. In Section 4, we will present the data reduction technique that is used to reduce the amount of raw data. In Section 5 we will present a comprehensive analysis with regard to performance and precision. Finally, we will summarize and discuss our results in Section 6.

2. Related Work

Particle advection is one of the fundamental classes of vector field visualization methods [WE04]. They provide the basis for a variety of derived visualization techniques, all of which rely on efficient particle tracing methods, e.g. stream and streak surfaces [BFTW09, GKT*08, KGJ09], dense vector field visualization methods [LHD*04], and the computation of the finite time Lyapunov exponent [FBTW10, GGTH07]. While the general computation scheme is straightforward to implement, the advection of large amounts of particles poses several challenges, particularly with regard to the size of the underlying data, bandwidth limitations, and computational complexity. The need for interactive methods further aggravates these issues.

Most contemporary particle advection techniques make use of parallelization. Ellsworth et al. pre-compute a large, dense population of particles on a parallel compute cluster [EGM04]. The resulting trajectories may subsequently be explored in an interactive session. Pugmire et al. describe a distributed-memory, parallel approach to streamline tracing on massive data [PCG*09].

Due to its embarrassingly parallel nature, particle tracing maps well to modern many-core compute devices. A focus has been on the adaption of particle advection to increasingly powerful programmable graphics hardware. Krüger et al. were among the first to introduce a GPU-based particle tracing on Cartesian, steady-state data [KKKW05]. Schirksi et al. later extended the general idea to unstructured grids and specifically addressed the problem of efficient point location [SBK06]. Bürger et al. in turn proposed a particle tracing system which handles time-dependent Cartesian grids on the GPU [BSK*07]. As of this writing, we are not aware of a method that copes with both, time-dependent, as well as unstructured data.

Cell location is a major challenge that seriously impacts the performance of particle tracing algorithms on unstructured grids. Many approaches use a spatial index structure, e.g. a kd-tree [LW77], in order to locate a mesh vertex close to the desired position. This initial guess is then refined, typically by means of a *walking scheme* as introduced by Kenwright and Lane [KL96]. Langbein et al. describe a two-stage scheme that follows this idea [LST03]. The location algorithm used by Schirski et al. executes a global point search on the host computer and moves the walking scheme to the compute device [SBK06]. Marmitt and Slussallek use an optimized ray-triangle intersection test based on Plücker coordinates in order to speed up the walking procedure for direct volume rendering via ray tracing [MS06]. Andrysco and Tricoche proposed *Matrix Trees*, a tree-based approach to point location, in which the tree's connectivity is efficiently encoded using a matrix representation [AT10]. Garth and Joy presented a data structure which is based on bounding interval hierarchies in order to further improve on previous results [GJ10]. Wald et al. advocate the use of bounding volume hierarchies which are adaptively refitted to consecutive time steps, in order to speed up isosurface raytracing in time-varying tetrahedral grids [WFKH07].

Mesh decimation techniques help to reduce the overall footprint of a data set. A variety of different decimation operators have been developed. Most of them share the property that an incremental operation, e.g. an edge collapse, is iteratively applied to a number of mesh primitives. Approaches mostly differ in this operator and in the quality metric based on which the next primitive is determined. Garland and Heckbert originally proposed the use of *quadric error metrics* for triangle mesh simplification [GH97]. Garland and Zhou later generalized this approach to arbitrary dimensions [GZ05]. Hoppe [Hop09] and Cignoni et al. [CCM*00] use various error metrics based on mass and density analogies. Regarding the simplification operator, Renze and Oliver propose the deletion of individual points along with a subsequent re-tessellation of the resulting area or volume [RO96]. Both aforementioned algorithms by Garland based on iterative edge collapses, as are the algorithms by Gelder et al. [GVW99] and Hoppe [Hop09]. Chopra and Meyer chose to collapse entire tetrahedra [CM02].

3. Time-Dependent Particle Tracing on the GPU

3.1. Particle Tracing

Interactive particle tracing relies on the advection of massive amounts of particles to convey flow structures. Let $\mathbf{x}(t)$ denote the particle position at time t and $\mathbf{v}(\mathbf{x}(t),t)$ the corresponding flow field velocity. Particle tracing in time-varying flow fields comprises the steps of

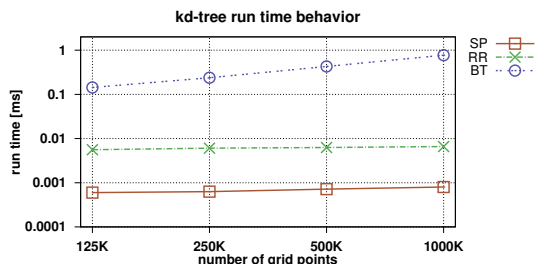1. Specify initial particle position $\mathbf{x}(t_0) = (x,y,z)$ at time t_0.

Figure 1: *Run time behaviour of the kd-tree implementations for the SP, RR and BT strategy. Note the logarithmic scale on the Y axis.*

Figure 2: *Average length of the tetrahedral walk after an initial search with the SP, RR and BT variation of the kd-tree traversal.*

2. Perform cell locations, e.g. for a linear interpolation in time find the grid cells c_i, c_j that contain the particle position in step t and $t + \Delta t$.
3. Spatially interpolate the velocities in each cell c_i, c_j at position $\mathbf{x}(t)$ to yield two interpolated velocities, $\mathbf{v}_i, \mathbf{v}_j$.
4. Perform a temporal interpolation between $\mathbf{v}_i, \mathbf{v}_j$ to determine the local velocity at time t.
5. Integrate along the local velocity field to determine the next particle position at time $t + \Delta t$ as $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \int_t^{t+\Delta t} \mathbf{v}(\mathbf{x}(s), s) \, ds$. If integration schemes of higher order are used the evaluation of the integral can involve additional spatial and temporal interpolations, and in particular, additional cell locations if cell boundaries are crossed.

3.2. Cell Location on Many Core Architectures

Kenwright and Lane [KL96] subdivided the cell location into a global search to first find a cell nearby the query point and a subsequent local search to find the cell that contains the query. We refer to their local search scheme as *tetrahedral walk*. The physical coordinates of a query point are transformed into the natural (barycentric) coordinate system of a tetrahedral cell. If neither component of its natural coordinates is negative and their sum is less or equal to one, the point lies inside the cell and the natural coordinates can be used to compute the interpolated velocities. If one or more of these conditions are invalid the query point lies across cell boundaries. The violated conditions reveal on which side of the cell the search should continue and a new candidate can be chosen from the adjoint tetrahedra. Schirski et al. [SBK06] implemented the tetrahedral walk on the *Graphics Processing Unit* (GPU) and used a kd-tree search to come up with a starting cells near the query points. They applied their method only to tetrahedral grids with a fixed cell structure. Their tree search was implemented on the CPU and was only executed once upon seeding new particles.

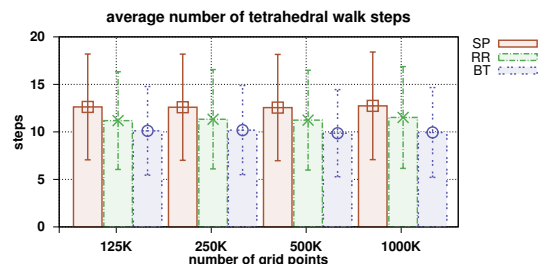The implementation of hierarchical search algorithms is known to be non-trivial on many-core compute platforms

(GPUs) due to architectural limitations. In particular, algorithms that rely on recursion or dynamic data structures (e.g. stacks) require additional efforts to realize. Due to further limitations on thread-level (e.g. number of registers, local memory) lightweight compute kernels are essential for an efficient device occupation.

Depending on the influence of the global search (kd-tree traversal) on the local search (tetrahedral walk) implementation strategies may differ. We investigated how the length of the tetrahedral walks is influenced by the choice of starting cell from the global search. More specifically, are more accurate (but more complex) tree traversals worth the implementation effort? We consider three different strategies for kd-tree traversal with increasing accuracy and complexity:

The *single pass* (SP) method performs exactly one run through the kd-tree and find *a* nearest cell vertex which is not guaranteed to be *the* nearest cell vertex to the query point. The advantages of this strategy are its light-weight implementation and fixed run-time. In particular, no divergent branching can occur.

In the *random restart* (RR) strategy [Pan08], the SP method is executes N times, where each time a random offset is added to the query point and only the result with the smaller distance to the query point is kept. Again, this method is not guaranteed to find nearest grid node to query point but performs well in practice. Like the SP divergent branching can not occur and the fixed run-time is N times that of SP.

The *back tracking* (BT) method [Ben90] traverses the kd-tree with backtracking and is guaranteed to find the nearest cell vertex to the query point. The backtracking is usually implemented recursively or iteratively with stacks or queues. Both data structures are not trivially to use on the GPU and can significantly increase the memory consumption and register use of compute kernels. Furthermore, varying run-time behavior can lead to divergent branches in compute kernels and is exponential in the worst-case [LW77].

For an evaluation, all three traversal algorithms were im-

plemented on the CPU to avoid device limitations. For reference, we used an Intel Xeon CPU E5540 at 2.53 GHz with 12GB RAM. We varied the type of kd-tree implementation (SP, RR, BT), measured run duration of the tree search and observed the length of the subsequent tetrahedral walk. We averaged the results over 100K random query points and repeated each test scenario four times. The number of random restarts in the RR method was ten. We evaluated the different strategies on the synthetic data set as described in Section 5. The results are summarized in Figures 1 and 2, respectively. As expected, we observed a fairly constant run time with both SP and RR and an exponential increase with BT (cf. Figure 1). To find the cell containing the query point the BT method required the least number of tetrahedral walk steps (approximately ten steps) whereas the SP method required two more tetrahedral steps on average (cf. Figure 2). RR performed slightly worse than BT but better than SP. Based on these results, we decided to use the SP method for point location on the many-core device for its light-weight kernel and constant run-time (to avoid divergent branches).

3.3. Implementation

We implemented the tracing of particles through time-varying tetrahedral grids completely within the CUDA framework [KH10]. The movement of each particle can be computed independently from one another which gives a straightforward parallelization: each particle is assigned to a thread and updated independently.

Particle data is stored in global device memory. Node position, attributes and cell neighborhood are streamed (synchronously) into global memory (bound as texture units) depending on memory availability and requirements. Beside particle positions, at least three time steps of the data has to fit into device memory. Whenever the adaptive DOPRI5 integration scheme is used, an additional block of global device memory is used to store the size of the last sub step for each particle which is required for the next advection step. Section 4 presents a decimation approach for data sets that exceeds device capabilities.

We set the maximum number of particles beforehand since dynamic memory allocation directly from within compute kernels has only recently become available and may reduce overall performance. Unused particle memory is marked and is assigned when new particles are seeded interactively. Memory of particles that have left the flow field or have expired for other reasons is marked as unused again and allows the filling of orphaned regions to maintain a dense distribution of particle positions.

Since the flow field evaluation requires random read access, we hold the node and cell data in texture memory to benefit from its cached read access. Each tetrahedral cell is assigned a unique ID such that neighborhood can be described by an integer vector with four components which can be accessed by a single texture fetch. We observed a significant increase in performance due to cache hits when, upon seeding, particles are first sorted according to the memory offset of the initial cell given by the global search step.

The tracing procedure is subdivided into three compute kernels: kd-tree search, tetrahedral walk and flow field integration. The kd-tree is built off-line in a preprocessing step that was neither implemented in parallel nor optimized for performance. The additional memory requirements for the search structure are detailed in Table 1. The kd-tree traversal is implemented as a SP method, and hence requires no recursive data structures. The implementation of the tetrahedral walk draws heavily from the GPGPU version of Schirski et al. [SBK06]. For reference, we implemented commonly used integration schemes like the Euler scheme, *third order Runge-Kutta* (RK3) and the DOPRI5 [DP80] with adaptive step size control.

4. Data Reduction

Data size is a critical problem in GPU-based particle tracing. Large, time-varying data typically does not fit into device memory. This makes it necessary to dynamically reload the next time steps. As both memory size and bandwidth are limited, we need a method to reduce overall data size. The algorithm we use here is based on [GZ05]. The goal is to reduce the data size down to a given size while preserving the information content. In the following, we will shortly outline the approach of Garland and Zhou, which preserves both, the geometry and the data content as much as possible.

The algorithm is based on iterative edge collapses. It starts by assigning an error value to each edge in the tetrahedral grid. It then selects the edge with the lowest error value and merge its two points. Cells adjacent to exactly one point are updated, cells adjacent to both points degenerate to triangles and are therefore deleted. The error values in the local environment are updated. This process is repeated until the target reduction is reached or no further edge can be collapsed.

The error metric used here is a special case of the general quadric-based error metric proposed by Garland and Zhou [GZ05]. We interpret our grid as a 3-dimensional complex in a k-dimensional space. A grid node has 3 geometric dimensions and $k-3$ data dimensions. For a given node vector $\mathbf{n} = (n_1, \cdots, n_k)$ and a given cell $(\mathbf{n}, \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3))$, we define the quadric $Q_{\mathbf{n}}$ as

$$Q_{\mathbf{n}}(\mathbf{x}) = \mathbf{x}^T A \mathbf{x} + 2b^T \mathbf{x} + c$$

where

$$A = I - \mathbf{e}_1 \mathbf{e}_1^T - \mathbf{e}_2 \mathbf{e}_2^T - \mathbf{e}_3 \mathbf{e}_3^T, \quad \mathbf{b} = -A\mathbf{p}, \quad c = \mathbf{p}^T A \mathbf{p}$$

and $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ is an orthonormal basis of the subspace spanned by $\mathbf{m}_1 - \mathbf{n}$, $\mathbf{m}_2 - \mathbf{n}$ and $\mathbf{m}_3 - \mathbf{n}$.

To preserve the boundary, we use an additional boundary

surface quadric. This is basically equal to the quadric defined above except the value of A, which is defined as

$$A = I - \mathbf{e}_1 \mathbf{e}_1^T - \mathbf{e}_2 \mathbf{e}_2^T$$

with an orthonormal basis $(\mathbf{e}_1, \mathbf{e}_2)$, spanning the two-dimensional subspace containing the boundary triangle.

For every cell c we compute the corresponding quadric Q_c. We assign to each point $\mathbf{n}$ the quadric $Q_\mathbf{n} = \sum_c Q_c$, the sum over the quadrics of the adjacent cells. If a point is on the boundary, we also add the boundary surface quadric, multiplied with a given penalty factor, which for our measurement is set to 10^{10}. The error metric for a given edge $(\mathbf{n}_1, \mathbf{n}_2)$ and the corresponding new point $\mathbf{n}_{new}$ is thus defined as:

$$err(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_{new}) = Q_{\mathbf{n}_1}(\mathbf{n}_{new}) + Q_{\mathbf{n}_2}(\mathbf{n}_{new})$$

We used two additions to the algorithm proposed by Garland and Zhou: Calculation of the optimal and point and flipping prevention. The optimal new point has to be a linear combination of $\mathbf{n}_1$ and $\mathbf{n}_2$, so we have $\mathbf{n}_{new} = \mathbf{n}_1 + c(\mathbf{n}_2 - \mathbf{n}_1)$. To compute c, we need to minimize the error value defined above, which leads to

$$c = -\frac{\left(\mathbf{p}_1^T A + \frac{1}{2}\left(\mathbf{b}_1^T + \mathbf{b}_2^T\right)\right)(\mathbf{n}_2 - \mathbf{n}_1)}{(\mathbf{n}_2 - \mathbf{n}_1)^T A (\mathbf{n}_2 - \mathbf{n}_1)}$$

To avoid spikes in the output data, we limit c to $[0, 1]$. To every edge $(\mathbf{n}_1, \mathbf{n}_2)$, we assign the resulting error value:

$$err(\mathbf{n}_1, \mathbf{n}_2) = Q_{\mathbf{n}_1}(\mathbf{n}_1 + c(\mathbf{n}_2 - \mathbf{n}_1)) + Q_{\mathbf{n}_2}(\mathbf{n}_1 + c(\mathbf{n}_2 - \mathbf{n}_1))$$

At each step, we select the edge $(\mathbf{n}_1, \mathbf{n}_2)$ with the lowest error value. For all cells adjacent to exactly one of the endpoints the point information is updated. The cell $(\mathbf{n}_1, \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3)$ will be transformed to $(\mathbf{n}_{new}, \mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3)$. Cells adjacent to both endpoints will degenerate to a triangle. These cells are deleted. We assign a new quadric $Q_{\mathbf{n}_{new}}$ to the new point, which is calculated as $Q_{\mathbf{n}_{new}} = Q_{\mathbf{n}_1} + Q_{\mathbf{n}_2}$. The costs are updated for all edges adjacent to the new point.

Edge collapsing comes at the risk of flipping cells, i.e. changing their orientation. We use an explicit orientation check to avoid cell flips. For every tetrahedron $(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3, \mathbf{n}_4)$ must hold, that the triangle $(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ is oriented counter-clockwise if seen from $\mathbf{n}_4$. If this constraint would not be fulfilled for any tetrahedron any more after an edge collapse, we ignore this collapse.

5. Results

5.1. Setup

Platform — All measurements have been carried out on a visualization workstation equipped with a quad core Intel XEON E5420 (host) rated at 2.53 GHz, 64 GB of RAM, and an NVidia GForce GTX 480 (device) featuring 1.5 GB of
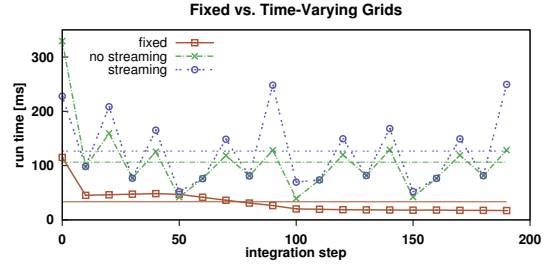

Figure 3: *Run time of the synthetic data, as fixed time step, as time-varying grid that is completely resident in device memory, and as time-varying grid that is streamed into device memory on demand.*
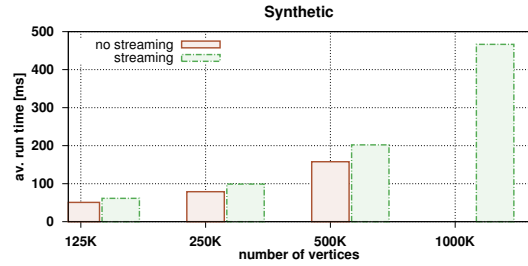

Figure 4: *Performance of streamed vs. non-streamed data for the synthetic data set. Additionally, the scalability of our implementation with respect to increasing size of the tetrahedral can be seen. No performance data for 1000K is available for the no streaming method due to limited device memory.*

video memory. This particular graphics cards has 15 Multiprocessors, each with 32 cores, hence 480 cores in total at a clock rate of 1.50 GHz. The measured average bandwidth for paged memory from host to device was about 4000 MB/s. The host memory was sufficiently large to keep all the data sets in host memory, eliminating the need for online swapping or reloading from disk.

Datasets — For our evaluation we used the following three data sets. Table 1 contains a summary of relevant data set statistics.

In the *synthetic dataset* nodes are randomly distributed over a rectangular domain of (-0.5, -0.5, -1.0) × (0.5, 0.5, 1.0) The synthetic flow is described at each position (x, y, z) as $\mathbf{v}(x, y, z) = (y, x, c_t)$ with time dependent parameters c_t that takes the value of two for the second, minus two for the fourth and zero for all other time steps. This essentially moves particles along a helix back and forth in the domain. A tetrahedral grid is generated using a constraint Delaunay algorithm [Si06]. We generated four instances of the data set with varying resolution, i.e. we used 125K, 250K, 500K, and

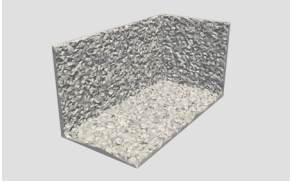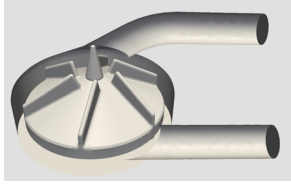|	Synthetic				Gyro				DeBakey			
	125K	250K	500K	1M	original	10%	20%	50%	original	10%	20%	50%
time steps	5	5	5	5	101	101	101	101	200	200	200	200
vertices per step	125K	250K	500K	1000K	197K	179K	161K	107K	631K	571K	511K	335K
total vertices	0.6M	1.3M	2.5M	5.0M	19.9M	18.0M	16.2M	10.8M	126.1M	114.1M	102.2M	67.1M
cells per step	0.8M	1.7M	3.4M	6.7M	1.1M	1.0M	0.9M	0.6M	3.7M	3.3M	3.0M	1.9M
total cells	4.2M	8.4M	16.9M	33.7M	115.0M	103.5M	92.0M	57.5M	742.9M	668.6M	594.3M	371.5M
memory per step [MB]	16.6	33.3	66.7	133.5	23.4	21.1	18.8	11.9	75.9	68.4	60.9	38.6
total memory [MB]	83	167	333	667	2,361	2,129	1,898	1,206	15,186	13,686	12,189	7,715
mem. overhead per step [MB]	14.7	29.6	59.3	118.8	21.1	19.3	17.5	10.5	71.4	65.6	53.1	35.6
total mem. overhead [MB]	74	148	297	594	2,131	1,949	1,768	1,061	14,280	13,120	10,620	7,120

Table 1: *Summary of data set statistics.*

1M vertices for mesh generation, respectively. Each data set consisted of five discrete time steps.

The two real world data sets, which we used for further evaluation, both resulted from numerical simulations of ventricular assist devices. The *GYRO* is an example of a centrifugal blood pump design. Here, the blood is transported by an impeller mounted at the bottom of the device. The axis of rotation is orthogonal to the main direction of flow. In contrast, the *DeBakey* pump resembles an axial design. Here, the impeller's axis of rotation is aligned with the main direction of flow. As a side note, it should be mentioned that the two simulations have been performed with respect to different length and time units and therefore time steps and distances are not directly comparable. A detailed assessment and discussion of the GYRO device has been published in [ABP06] whereas [HTP*08] contains a discussion of blood damage visualization techniques evaluated on the DeBakey device.

The impeller's motion has been incorporated in the simulations by way of a moving tetrahedral grid, which discretizes the volume around the impeller's surface geometry. The solver internally stitches this grid to the static part of the mesh for every simulation time step. This setup poses a challenge for subsequent visualization, because it results in an overall changing grid topology for every simulation time step. Hence, the data for the entire mesh connectivity including search data structures has to be updated in graphics memory once for each time step.

As an additional challenge, particles need more than one impeller revolution to pass the device. It will typically take 10 to 15 revolution for a particle to travel from inlet to outlet for the GYRO. For the DeBakey it takes about 20 to 30 revolutions. This further increases bandwidth demands because

each of the two data sets does not entirely fit into graphics memory. Hence, the data has to be re-transmitted for every revolution.

In order to investigate the effects of data reduction, we applied the algorithm described in the previous section to both data sets. The different resolutions are defined by their respective reduction fraction, which is given in terms of a fraction of the original number of tetrahedra. For example, the 10% version of the GYRO consists of *10% less tetrahedra* than the original data set. For the GYRO data set we created decimated versions of 1%, 2%, 5%, 10%, 20%, and 50%. The levels below 10% have been created for validation purposes only and are thus not included in Table 1 for brevity's sake. For the DeBakey we investigated target reductions of 1%, 2%, 5%, 10%, 20%, and 50%.

Integration Settings — For the synthetic data set, we comparatively evaluated three different integration schemes, namely standard forward Euler, a three step Runge-Kutta method (RK3), and an adaptive Runge-Kutta method introduced by Dormand and Prince (DOPRI5, [DP80]). All tracing experiments used a step size of 0.02 for the Euler and RK3 integration scheme. That of the adaptive DOPRI5 scheme was limited to the interval $[0.02, 0.2]$. Particles were advected for a total of 200 time steps.

For both blood pumps, we only used the RK3 integration scheme and set the step length to one fifth of the simulation time step, resulting in a step size of 0.00006 and 0.0008. We traced particles for 15 and 26 full impeller revolutions, respectively.

Generally, all performance measurements were conducted with a particle population of 1M particles. In case of the syn-
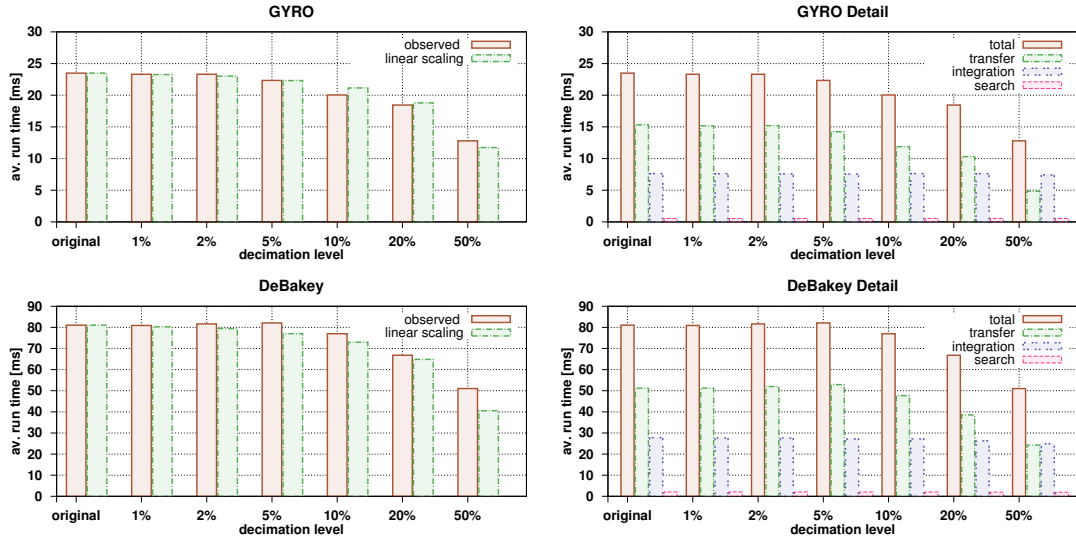
Figure 5: *Performance results for the two blood pump simulation data sets. Top row: Performance figures for the GYRO VAD. Bottom row: DeBakey axial VAD.*

thetic data set, these have been seeded in a box-shaped region centered in the data domain. For the two blood pumps, we used a spherical seeder region at each device's inlet.

5.2. Performance

Synthetic — We first analyzed how much performance gets lost when switching from fixed tetrahedral grids towards time-varying grids. In general, all cell locations have to performed twice as often in the time-varying case whereas the total number of cell location depend on the order of integration scheme. We distinguished between three conditions, (1) one *fixed* time step, the first time step of the synthetic data was initially copied to the device, (2) *no streaming*, all time steps were initially uploaded to to the device with no further memory transfer between host and device, and (3) *streaming*, only two consecutive time steps were kept in device memory, and data was synchronously transfered from host to device whenever a new time step was encountered. For the benchmark we used the synthetic data set as described in Section 5.1 and averaged the results over all data sizes (except 1000K since it did not fit completely into device memory) and integration schemes to yield one performance measure per test condition. The results are detailed in Figure 3. The run time of the fixed method gradually decreases as more and more particle leave the domain. The average run time of the fixed method is effectively doubled in the non streaming case where minor peaks in performance are probably due to cache misses upon new time steps. When taking transfer times into account in the streaming case, the average run time is slightly worse that of the non stream-

ing method. Major peaks are visible when new data is synchronously copied to the device.

The impact on the memory transfer between the streaming and non streaming method with respect to the individual data set sizes is detailed in Figure 4 as well the scalability of our implementation with increasing data size.

Real-World Performance — The performance of our method on the real-world data sets GYRO and DeBakey is summarized in Figure 5. The figures on the left depict the scaling with respect to various decimation levels of decreasing data size and linear scaling for reference. The figures on the right show how much computation time is distributed among data transfer, integration and cell relocation search in between successive time steps.

We observe an almost linear scaling with respect to data size for the increasing decimation levels of the GYRO. When comparing the results in detail we see that the transfer time is responsible for that, since the integration time is more or less constant. In the DeBakey data set, the linear scaling is no longer present at 50% decimation. In both cases we notices that the transfer time is the major performance bottleneck for large data sizes and integration becomes dominant as soon as data size is significantly reduced.

5.3. Precision

In order to assess the error that is introduced by data reduction, we performed the same tracing computations on the reduced data sets. Table 2 summarizes the error introduced to the velocity field by the data reduction step. Figure 6 shows

	GYRO				DeBakey			
	1%	10%	20%	50%	1%	10%	20%	50%
RMSE velocity magnitude	0.00044	0.00084	0.00137	0.00500	0.00013	0.00013	0.00014	0.00021
RMSE velocity direction	0.04988	0.22482	0.55205	2.02165	0.02097	0.07542	0.21763	1.51743

Table 2: *Summary of the error introduced by data decimation, given as root mean square error (RMSE) per grid point. Lengths are given in the respective length unit of the simulation, the direction errors are given in degrees.*
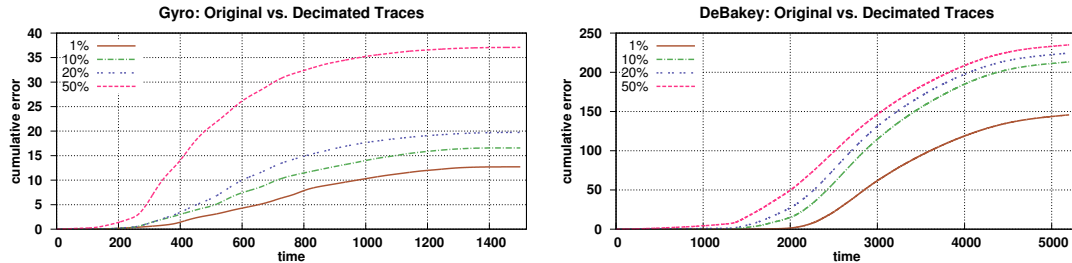

Figure 6: *Impact of decimation of particles traces. We show the cumulative error between traces from the original grid and each decimated level for GYRO (left) and DeBakey (right).*

plots of the cumulative distance error for the GYRO and De-Bakey data sets and the different reduction stages, respectively. These results were obtained by a point-wise comparison of every two traces starting at the same seed point and being advected through the original and the reduced version of the data respectively. The resulting error is added for all previous point pairs up to the given point in time.

A surprising result is that already the 1% version introduces decisive errors in the tracing computation for both data sets. This is due to the fact that the underlying decimation algorithm starts to eliminate big tetrahedra in the largely homogeneous regions of the inlet first. While the local data error introduced in these regions is relatively small, it leads to a slight deviation in the vector magnitude. Hence, particles in the reduced version travel at a different speed than their cousins on the original data. Eventually, these particles reach the entry to the impeller at a different point in time and consequently at a different rotation state. This leads to a significant dispersion of the particle traces and therefore results in the steep ascent in the error curves. The point in time, i.e. after approximately 300 and 2,000 time steps for the GYRO and the DeBakey respectively, matches with this observation, i.e. this is the time where the majority of the particles enter the impeller in the respective devices.

Beyond the 1% version, the error level increases only lightly for the GYRO, when moving to the 10% and 20% decimation levels. However, a big jump is evident when going to the 50% level. Meeting the 50% decimation requirement, i.e. effectively eliminating half of the tetrahedra, requires reduction of previously untouched regions. This is evident from Figure 7, which gives an overview of the error in the velocity magnitude. While the 20% version still shows

relatively little, isolated error regions, the 50% data exhibits severe problems. These are concentrated in two main regions: first, a significant lengthening of the original vectors in the inlet region leads to the aforementioned problems; second, major errors are evident in the region around the impeller's mount, which is characterized by a large number of very small tetrahedra that withstood decimation so far. These observations directly follow the error metrics given in Table 2. The underlying simulation grid, too, shows a significant jump in average point-wise error for both, the velocity direction and the velocity magnitude, between the 20% and the 50% versions.

Decimation of the DeBakey data does not have as drastic an effect on the particle traces as it had for the GYRO simulation. Similarly to the GYRO case, the 1% decimation already shows a significant error. However, the 10% and 20% versions are very close in terms of the cumulative integration error. Even the 50% decimation shows results that are significantly closer to the 10 and 20% versions, albeit the drastic increase in data error shown in Table 2. Figure 8 shows the length deviation for three different decimation levels. While the 10 and 20% levels show only limited error, the 50% version shows significant deviations throughout the entire domain. It can be seen that even in the lower first errors are evident in the inflow region before the flow straightener. Analogous to the GYRO device, these errors result from the decimation of rather big tetrahedra with largely uniform velocity distribution. These lead to a small error in the integration process early on. As explained above, this small deviation has significant effects further downstream. However, the 50% decimation does not have the harsh effect on particle tracing compared to the 50% version of the GYRO. The reason for this is that the main source of error, namely the point
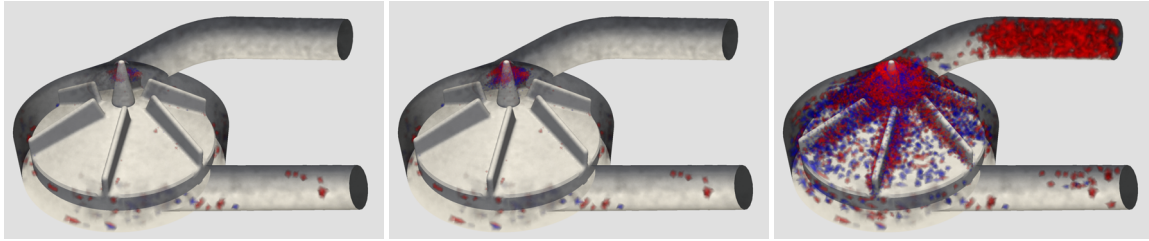
Figure 7: *Volumetric representation of vector length differences in the decimated Gyro dataset (left: 10%, mid: 20%, right: 50% decimation). Vectors in full red regions are at least one tenth of the diameter of the inlet longer than the original vector, vectors in blue regions are one tenth of the diameter shorter.*
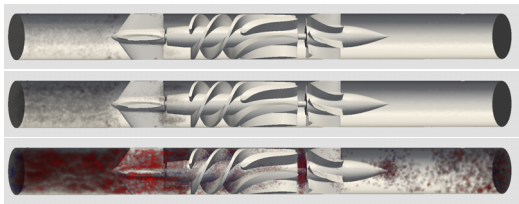

Figure 8: *Volumetric representation of vector length differences in the decimated DeBakey dataset (top: 10%, mid: 20%, bottom: 50% decimation). Vectors in full red regions are at least one tenth of the diameter longer than the original vector, vectors in blue regions are one tenth of the diameter shorter.*

at which the traces enter the impeller, is also the only major variation in this data set. In contrast, traces in the GYRO also take paths at different radii around the impeller. Due to the axial layout this is less critical in the DeBakey, because traces will not diverge as quickly as in the centrifugal design.

In summary, the particle traces which are traced on the decimated grids still reveal the significant flow structures, i.e. they are suitable for an initial qualitative overview of the data. Nonetheless, we have to conclude that the decimation introduces significant errors for both data sets, which makes the reduced data largely unusable for subsequent quantitative analysis, e.g. hemolysis prediction [ABP06].

6. Summary and Future Work

In this paper we have presented a many-core algorithm for particle advection on time-dependent, unstructured simulation data. After outlining the general algorithm, we have discussed several options for the crucial problem of cell location. A straightforward two phase search scheme which uses a search tree for initial point location in combination with a short tetrahedral walk was used for cell location. We found the best trade-off between performance and implementation effort by a single pass kd-tree traversal which avoids the use of recursive data structures on the GPU. A major bottleneck

when dealing with time varying data in general and time-dependent unstructured grids in particular is device memory and memory bandwidth. To this end, we have discussed the use of data reduction techniques. We evaluated the overall approach, both in terms of performance and precision. The performance evaluation indicates that the data transfer time is the major performance bottleneck for large data sizes and integration becomes only dominant as soon as data size is significantly reduced.

Regarding future work, we would like to investigate user-driven data reduction techniques, i.e. decimation algorithms which provide high detail in a user-defined region of interest whereas the outside context zones are only approximated with significantly less detail. The major challenge here will be to provide an interactive algorithm which is able to handle changing user input with short reaction times. In addition, we would like to analyze the effects of temporal subsampling for data reduction.

References

[ABP06] ARORA D., BEHR M., PASQUALI M.: Hemolysis Estimation in a Centrifugal Blood Pump Using a Tensor-Based Measure. *Artificial Organs 30*, 7 (2006), 539–547.

[AT10] ANDRYSCO N., TRICOCHE X.: Matrix Trees. *Computer Graphics Forum 29*, 3 (2010), 963–972.

[Ben90] BENTLEY J. L.: K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry* (New York, NY, USA, 1990), ACM, pp. 187–197.

[BFTW09] BÜRGER K., FERSTL F., THEISEL H., WESTERMANN R.: Interactive Streak Surface Visualization on the GPU.

IEEE Transactions on Visualization and Computer Graphics 15, 6 (2009), 1259–1266.

[BSK*07] BÜRGER K., SCHNEIDER J., KONDRATIEVA P., KRÜGER J., WESTERMANN R.: Interactive Visual Exploration of Unsteady 3D Flows. In *Proceedings of the Joint EG/IEEE VGTC Symposium on Visualization* (2007), pp. 251–258.

[CCM*00] CIGNONI P., COSTANZA D., MONTANI C., ROC-CHINI C., SCOPIGNO R.: Simplification of Tetrahedral Meshes with Accurate Error Evaluation. In *Proceedings of IEEE Visualization* (2000), pp. 85–92.

[CM02] CHOPRA P., MEYER J.: TetFusion: An Algorithm For Rapid Tetrahedral Mesh Simplification. In *Proceedings of IEEE Visualization* (2002), pp. 133–140.

[DP80] DORMAND J., PRINCE P.: A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics 6*, 1 (1980), 19 – 26.

[EGM04] ELLSWORTH D., GREEN B., MORAN P.: Interactive Terascale Particle Visualization. In *Proceedings of IEEE Visualization 2004* (2004), pp. 353–360.

[FBTW10] FERSTL F., BÜRGER K., THEISEL H., WESTER-MANN R.: Interactive Separating Streak Surfaces. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1569–1577.

[GGTH07] GARTH C., GERHARDT F., TRICOCHE X., HAGEN H.: Efficient Computation and Visualization of Coherent Structures in Fluid Flow Applications. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1464–1471.

[GH97] GARLAND M., HECKBERT P. S.: Surface Simplification Using Quadric Error Metrics. In *the ACM Siggraph* (1997), pp. 209–216.

[GJ10] GARTH C., JOY K. I.: Fast, Memory-Efficient Cell Location in Unstructured Grids for Visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1541–1550.

[GKT*08] GARTH C., KRISHNAN H., TRICOCHE X., BOBACH T., JOY K. I.: Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1404–1411.

[GVW99] GELDER A. V., VERMA V., WILHELMS J.: Volume Decimation of Irregular Tetrahedral Grids. In *Computer Graphics International* (1999), pp. 222–230.

[GZ05] GARLAND M., ZHOU Y.: Quadric-Based Simplification in Any Dimension. *ACM Transactions on Graphics 24*, 2 (April 2005), 209–239.

[Hop09] HOPPE H.: New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *Proceedings of IEEE Visualization* (2009), pp. 59–66.

[HTP*08] HENTSCHEL B., TEDJO I., PROBST M., WOLTER M., BEHR M., BISCHOF C., KUHLEN T.: Interactive Blood Damage Analysis for Ventricular Assist Devices. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1515–1522.

[KGJ09] KRISHNAN H., GARTH C., JOY K. I.: Time and Streak Surfaces for Flow Visualization in Large Time-Varying Data Sets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1267–1274.

[KH10] KIRK D. B., HWU W. W.: *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann., 2010.

[KKKW05] KRÜGER J., KIPFER P., KONDRATIEVA P., WEST-ERMANN R.: A Particle System for Interactive Visualization of

3D Flows. *IEEE Transactions on Visualization and Computer Graphics 11*, 6 (2005), 744–756.

[KL96] KENWRIGHT D. N., LANE D. A.: Interactive Time-Dependent Particle Tracing Using Tetrahedral Decomposition. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (1996), 120–129.

[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*, 2 (2004), 203–221.

[LST03] LANGBEIN M., SCHEUERMANN G., TRICOCHE X.: An Efficient Point Location Method for Visualization in Large Unstructured Grids. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2003* (2003), pp. 27–35.

[LW77] LEE D., WONG C. K.: Worst-Case Analysis for Region and Partial Region Searches in Multidimensional Binary Search Trees and Balanced Quad Trees. *Acta Informatica 9*, 1 (1977), 23–29.

[MS06] MARMITT G., SLUSALLEK P.: Fast Ray Traversal of Tetrahedral and Hexahedral Meshes for Direct Volume Rendering. In *Proceedings of the Joint Eurographics – IEEE TCVG Symposium on Visualization* (2006), pp. 235–242.

[Pan08] PANIGRAHY R.: An improved algorithm finding nearest neighbor using Kd-trees. In *LATIN'08: Proceedings of the 8th Latin American conference on Theoretical informatics* (Berlin, Heidelberg, 2008), Springer-Verlag, pp. 387–398.

[PCG*09] PUGMIRE D., CHILDS H., GARTH C., AHERN S., WEBER G.: Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of of the IEEE/ACM Conference for High Performance Computing, Networking, Storage, and Analysis, published on CD-ROM* (2009).

[RO96] RENZE K. J., OLIVER J. H.: Generalized Unstructured Decimation. In *IEEE Computer Graphics and Applications* (1996), vol. 26, pp. 24–32.

[SBK06] SCHIRSKI M., BISCHOF C., KUHLEN T.: Interactive Particle Tracing on Tetrahedral Grids Using the GPU. In *Proceedings of Vision, Modeling, and Visualization (VMV) 2006* (2006), pp. 153–160.

[Si06] SI H.: On Refinement of Constrained Delaunay Tetrahedralizations. In *Proceedings of the 15th International Meshing Roundtable*, Pébay P. P., (Ed.). Springer Berlin Heidelberg, 2006, pp. 509–528.

[WE04] WEISKOPF D., ERLEBACHER G.: Overview of Flow Visualization. In *The Visualization Handbook*, Hansen C. D., Johnson C. R., (Eds.). Elsevier Academic Press, 2004, pp. 261–278.

[WFKH07] WALD I., FRIEDRICH H., KNOLL A., HANSEN C. D.: Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (2007), 1727–1734.