

Parallel Calculation of Accurate Path Lines in Virtual Environments through Exploitation of Multi-Block CFD Data Set Topology

ANDREAS GERNDT, MARC SCHIRSKI, TORSTEN KUHLEN AND CHRISTIAN BISCHOF
Center for Computing and Communication, Aachen University (RWTH), Germany *

(Submitted: October 10, 2003)

Abstract. The use of Virtual Reality (VR) techniques for the investigation of complex flow phenomena offers distinct advantages in comparison to conventional visualization techniques. Especially for unsteady flows, VR methodology provides an intuitive approach for the exploration of simulated fluid flows. However, the visualization of Computational Fluid Dynamics (CFD) data is often too time-consuming to be carried out in real-time, and thus violates essential constraints concerning real-time interaction and visualization. To overcome this obstacle, we make use of the fact that typically a multi-block approach is employed for domain decomposition, and we use the corresponding data structures for the computation of path lines and for parallelization. In this paper, we present the synthesis of fragmented multi-block data sets and our implementation of an accurate path line integration scheme in order to speed up path line computations. We report on the results of our efforts and describe a combination of this algorithm with a highly efficient visualization approach of large amounts of particle traces, thus considerably improving interactivity when exploring large scale CFD data sets.

Keywords: Virtual Reality, Computational Fluid Dynamics, Multi-Block, Initial Value Problems, Interpolation, Parallel Algorithms, High-Performance Computing

Mathematics Subject Classifications(2000): 76Mxx, 76M27, 76M28, 65M55, 65L05, 65L06, 65D05, 65Y05, 68U05

1. Introduction

Visualization techniques are applied in scientific computation to achieve easily understandable representations of numerical data. The use of Virtual Reality (VR) methods allows fast and intuitive comprehension of complex spatial relationships but requires real-time interaction capabilities. In order to minimize computational load on the visualization host,

* email: {gerndt,schirski,kuhlen,bischof}@rz.rwth-aachen.de

the handling and computationally expensive post-processing of large scale simulation data is usually shifted to a dedicated high-performance computer system.

Computational Fluid Dynamics (CFD) can be divided into three distinct steps [13]. First, the domain decomposition process converts the region that contains the flow field into a discrete geometry. Enriched with initial and boundary conditions, the time-consuming simulation computes the flow solution by applying discretization methods like finite differences or finite elements. The concluding post-processing step deals with the analysis and visualization of the results and is forced to work with data structures created by the first step and filled up by the second step. This includes the computation of objects like streamlines, isosurfaces, and cut planes, which can subsequently be converted into displayable objects.

Scientific projects at Aachen University that allow for a VR-based exploration of the corresponding data mainly make use of our VR toolkit *ViSTA* [17]. For CFD post-processing, *ViSTA* is additionally enhanced by the CFD framework *ViSTA FlowLib* [19], which is optimized for unsteady flows. It is based on the open-source Visualization Toolkit (VTK) [20], which offers hundreds of classes for scientific visualization. However, VTK supports steady flows only. For the more complex post-processing of unsteady flows, we have developed additional functionality.

This paper deals with the parallel computation of time-dependent path lines. First, we consider the advantages and exploitation of multi-block topologies. For rapid prototyping, we use a rather small multi-block data set defining the flow field of a combustion engine as test case.

Next, we outline VTK's streamline integration scheme in order to illustrate performance and accuracy issues as well as shortcomings. Afterwards, our own path line algorithm is described, which makes use of different approaches for more sophisticated adaptive step-size control and time-position interpolation.

Our parallelization framework then combines the developed multi-block management and the improved particle integrators to accelerate path line computations. It also makes use of multi-block topologies in order to reduce the I/O access to the stored block data files, which is one of the most important points in order to accelerate these computations.

Finally, we demonstrate the usage of parallelly computed path lines by *ViSTA FlowLib* for a highly efficient visualization of particle traces in unsteady flows.

2. Data Structures

The type of grids generated is not only important for the quality of the simulation but also for post-processing algorithms. Grid generation is exclusively geared towards an optimally converging simulation result. The extraction and filtering methods are forced to make use of these grids.

To solve the Partial Differential Equations (PDEs) governing complex flow fields, the computational domain is typically subdivided into multiple structured grids. Structured grids consist of hexahedral cells and can be classified into cartesian, regular, and curvilinear grids. However, in order to simplify the development of a PDE solver, the solver transforms the curvilinear grid from physical space ($\mathcal{P}$-space) to computational space ($\mathcal{C}$-space), where all cells are uniform cubes.

Another approach to domain decomposition uses unstructured grids. Unstructured grids consist of arbitrary polyhedral elements. The most commonly used elements are tetrahedrons, prisms, and hexahedrons. Normally, only one cell type is used, but different cell types can be mixed as well. Sometimes the flow region is not only decomposed by a unique grid type but by structured and unstructured grid blocks, which results in a so-called hybrid grid. Overset Grids (also referred to as Chimera grids) are multi-block grids containing partially overlapping sub-grids. Another decomposition type is the hierarchical grid, whose cells are subdivided and organized by octrees. [13]

2.1. Topology

Multi-block solvers based on finite volume methods are efficient and widely used. This approach solves Navier-Stokes equations with a large number of unknowns and makes use of a multi-block grid, which often consists of more than one hundred blocks. [10]

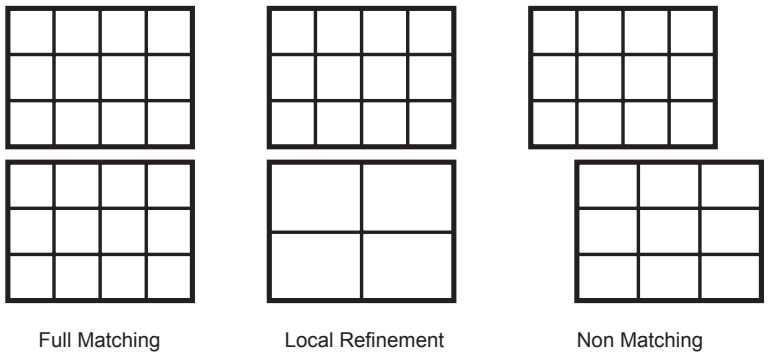


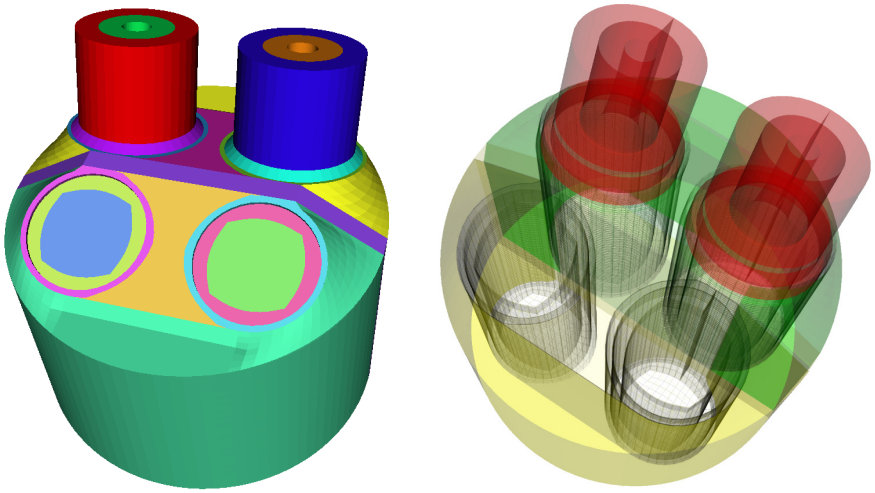
Figure 1. Different types of multi-block connectivity.

Multi-block grids can be classified by considering the boundary connectivity (Figure 1). The most widely used type is the matching type. Each interface node has a corresponding node of the connected neighboring block. This guarantees the continuity of the physical domain at the block interfaces. In case of local refinement, a discontinuity of the solution might appear. In the most general case, the boundaries of neighboring blocks are displaced and the partitioning is locally refined; i.e. in general, there are no common points anymore.

2.2. The Engine Multi-Block Data Set

To deal with restricted memory size, file server access time, and small network bandwidth, it is essential to make use of topology information to get data just required for current computations on time. For the development of our ideas, we use a relatively simple full-matching multi-block data set. However, it possesses all properties needed to develop a software framework that manages arbitrary multi-block data sets.

The data set describes the inflow of fluid in a spark ignition engine with 4 valves [1]. The simulation includes the filling and the compression phase but neither ignition nor combustion. The simulation domain is decomposed in 23 single blocks (Figure 2), and the simulation process is divided into 63 discrete time levels.



a. Single colored blocks b. Transparent and wireframe blocks

Figure 2. Used blocks for domain decomposition.

The multi-block grid geometry is warped to fit into the domain. During the compression simulation (from time level 35 onward), the inlet valves are closed and not considered anymore. Therefore, blocks

18 to 23 are removed. In addition, for each time level, blocks potentially change their resolution to model piston and valve movement. Thus, the whole grid consists of 114.998 nodes in time level 1 and grows to 245.588 nodes in time level 34. This leads to a total data set size of 365 MByte. Compared to data set sizes of current CFD simulations, which can reach some hundred Giga-Bytes [3], this data set is relatively small.

2.3. Cell Search

During a particle integration step (as described in Section 3), the respective particle might leave its current cell. Therefore, each new determined position must be validated if it is still located within the same cell. This cannot directly be decided in $\mathcal{P}$-space because of non-linearity between position and grid structure; however, this linearity is given in $\mathcal{C}$-space. Iterative algorithms like the *stencil walk* method or the Newton-Raphson iteration are applied to find the cell containing the point [18]. For each iteration step, these methods have to compute a variety of Jacobians by means of finite differences in order to convert current $\mathcal{P}$-space positions into $\mathcal{C}$-space. If the algorithm detects that the assumed cell does not contain the current position, the iterative algorithm restarts with a neighboring cell. Therefore, determination of point locations is rather expensive.

When the point location algorithm terminates successfully, not only the searched cell but also the point distance to the cell vertices in $\mathcal{C}$-space as well as in $\mathcal{P}$-space is determined. The three distances in $\mathcal{C}$-space (the range is [0,1]) can be designated as cell offsets or parametric cell coordinates. In $\mathcal{P}$-space we have eight distance values called weights. These weights can now be used to obtain the required values (e.g. the actual velocity vector) at a given position. This interpolation method is called inverse distance weighting [18].

Kenwright and Lane depict numerical problems of hexahedral-based search strategies and recommend a more efficient approach to find the cell and the corresponding weights. In their approach, the hexahedral cells of the grid are decomposed into tetrahedrons. Fortunately, this grid decomposition does not have to be carried out in advance but can be performed locally whenever a hexahedral cell is entered [11, 15].

If a particle leaves the current cell but stays within the current block, the internal block topology allows direct selection of the neighboring cells. However, a particle might also leave a block of a multi-block grid. In that case, the newly entered block and cell must be located. In order to facilitate this search, we developed a multi-block topology data structure (Figure 3), which can be considered as a substitution of the inherent cell topology of a single block for this inter-block connectivity.

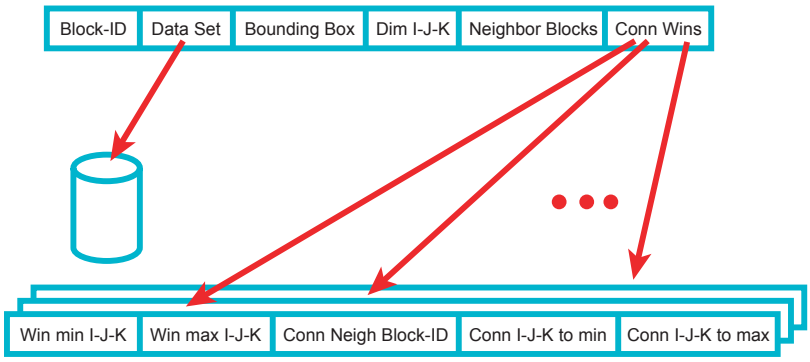


Figure 3. Data structure for multi-block topology information.

Not each single cell connection between neighbored blocks is stored. In general, a group of cell sides of one block borders to another group of cell sides of a second block. A *connection window* is stored by means of *i*-*j*-*k*-indices of two grid nodes that are lying diagonally at corners of that window. This specifies a well-defined window, because a window is restricted to one side of a hexahedral block. However, start and end corners can be selected differently. Therefore, the start corner is the one with the lowest index. The corresponding window of the neighboring block must use the matching start and end corners. An example is depicted in Figure 4.

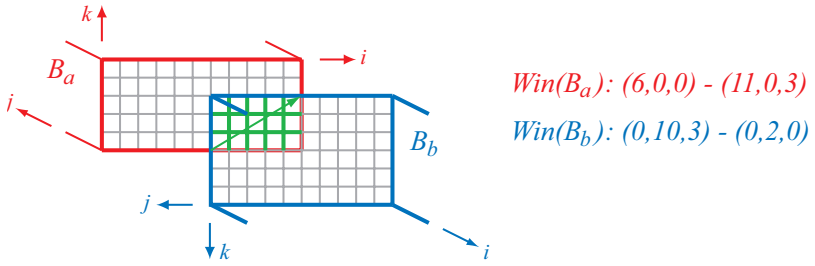
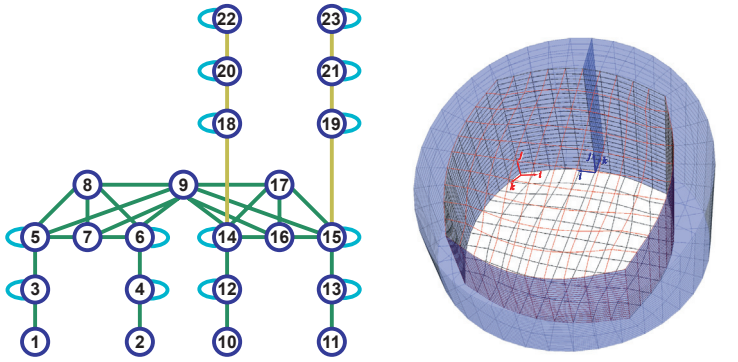


Figure 4. Example of a connection window between block a and block b defined in $\mathcal{C}$-space.

Connection windows can now be used to determine the neighboring block and cell directly. Only blocks with connected cell sides are considered as neighbors. If only a cell edge or cell vertex is shared, this connection is not considered, although a particle might leave a current block and enter an edge-connected or vertex-connected block. Therefore, not only connection windows but a hierarchy of search strategies is used to facilitate locating of the current block and cell. If the connection window search fails, a more expensive cell search within this particular neighboring block is started.

If the given position can not be located in any cell of this block, the algorithm falls back to neighbor block information. These neighbors are explored next. The neighborhood relationship of the sample engine data set is shown in Figure 5.a. If these searches also fail, a search pass uses blocks not yet tested and whose bounding boxes contain the current particle position.



a. Block topology tree b. Connection of block 1 and 3

Figure 5. Topology of the engine multi-block data set.

This search strategy is used to reduce search time and data loading time. It also reduces memory demands, because only reasonable block candidates will be loaded into memory and rarely used ones can be unloaded. Furthermore, if topology information is not available on one hierarchy level the algorithm can fall back to the next level. If no topology exists at all, a brute-force and expensive search over all blocks must be applied.

A special aspect of connection windows is depicted in Figure 5.b. Block 1 located in the center is surrounded by block 3. Block 1 looks like a box. In contrast, block 3 was stretched and thereafter clung around block 1 like a bangle. Therefore, only one side of block 3 is connected to four sides of block 1. Furthermore, two sides of block 3 touch each other, which is called a self-connection and is also stored as a connection window. In Figure 5.a, a self-connection is depicted as a cyclic link.

An additional feature offered by the multi-block topology information structure is the possibility to store boundary information for each single connection window. Therefore, it is possible to determine whether a window has a corresponding window of a second block or describes the boundary of the whole domain. A particle that has completely left the multi-block data set can immediately be recognized as such. For domain boundaries, viscosity boundary conditions (slip and no-slip), inflow or outflow boundary conditions, and other conditions can be assigned. In our test case, the walls of the combustion chamber consist of no-

slip boundaries (zero velocities are assigned to all except the bottom surface, which inherits the velocity of the connected piston) and two inflow boundaries (at the valves). Therefore, particles can enter but never leave the flow domain.

3. Particle Advection

Particle advection is based on the Lagrangian formulation of flow fields. It depicts the time-dependent movement of material particles and results in a path line, which is a numerical solution of an ordinary differential equation (ODE) described as an initial value problem. The initial value is the seed point. The motion of particles through a flow field is computed with the following equation [18, 21]:

$$\frac{dx}{dt} = v(x,t) \tag{1}$$

3.1. The Visualization Toolkit

The Visualization Toolkit (VTK)[1] does not support unsteady CFD post-processing and manages single data sets only. For these data sets, a variety of steady flow visualization algorithms, like the calculation of cut planes, isosurfaces, and streamlines are implemented. The computation of particle advection in steady flows results in streamlines. Therefore, we first examine the streamline implementation of VTK.

VTK offers the class `vtkStreamLine` as an interface to compute streamlines. After determination of all start cells of the selected seed points and additional control parameters, the super-class `vtkStreamer` calls `ThreadedIntegrate()`, which is the main integration function to compute the corresponding streamlines. The integrator, as well as the `vtkStreamer` object, makes use of `vtkInterpolatedVelocityField`, which controls the access to the data set. It is utilized whenever a velocity vector or scalar value at a certain position of the flow field is needed; i.e., it hides the discrete nature of the respective data set and allows continuous access by interpolation between cell nodes.

Furthermore, `vtkInterpolatedVelocityField` addresses the problem that searching a cell within an arbitrary data set is a time-consuming task, and that during integration a particle often stays in one cell for several time steps. Therefore, it caches the current cell and assumes that this cell will be valid for the next integration step. This speeds up

[1] This work bases on VTK version 4.0.

streamline calculations considerably. The current cell can also be set by the application, and cell caching can be enabled and disabled.

VTK offers Runge-Kutta second order and fourth order integration schemes to solve initial value problems. Fourth-order Runge-Kutta (Equation 2) is more accurate and shows already good results with coarse step sizes [7]. The most important method offered by the class `vtkRungeKutta4` is `ComputeNextStep()`, which expects the current particle position x_i and the corresponding velocity $v(x_i)$. Furthermore, it needs the step size h and the elapsed time t_i the particle has traveled.

$$\begin{aligned} k_1 &= v(x_i, t_i) \\ k_2 &= v(x_i + \tfrac{h}{2} \cdot k_1, t_i + \tfrac{h}{2}) \\ k_3 &= v(x_i + \tfrac{h}{2} \cdot k_2, t_i + \tfrac{h}{2}) \\ k_4 &= v(x_i + h \cdot k_3, t_i + h) = v(x_i + h \cdot k_3, t_{i+1}) \\ x_{i+1} &= x_i + h \cdot \left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \right) \end{aligned} \tag{2}$$

3.2. Adaptive Step Size Control

Generally, reduction of the step size improves the quality of the solution by decreasing the truncation error; however, more iterations are needed and the accumulated roundoff error increases. To strike a balance, an ODE integration approach should employ an adaptive integration step size control [22].

Numerical sensitivity in VTK solutions is already pointed out by the use of different compiler flags. Our test case depicts slightly different solutions between a version that includes debug information and a compiler optimized version. In these circumstances, it is important to note that VTK uses only single-precision, i.e. 32-bit, floating point values for computations, even on 64-bit platforms.

VTK applies an integration scheme similar to one described by David Lane [12], making use of the current speed of the considered particle and the diameter of the cell where the particle is just located. In order to measure the diameter of the cell, the bounding box diagonal is computed. The step size is additionally modified by a user-defined constant, which defines the fraction of a cell size to be considered. The resulting computation is shown in Equation 3. David Lane further proposes that smaller steps should be used for grid regions where the flow velocity varies rapidly.

$$stepSize = cellStepLength \cdot cellDiag / particleSpeed \tag{3}$$

However, VTK neither uses global information nor does it consider truncation errors. Other approaches estimate this error in order to

determine the next step size. A rather straightforward approach is step doubling introduced by Press et al. as a general ODE solution technique [16]. The current step size is used to compute a first estimation of the next particle position. In addition, for the same step, only half the size is used and applied twice. The distance between these two positions can be used as an indication for the truncation error. If this difference is too large or too small, the step size must be adjusted.

Darmofal and Haimes measure the angle between velocity vectors at successive particle positions [5]. If the change in direction is larger than 15 degrees, the step size is halved, whereas the step size is doubled if it is less than 2 degrees. Kenwright and Lane [11] implemented a similar scheme, which measures the angle between successive line segments on a path line (Equation 4). Through experiments they found their algorithm more accurate than the step doubling scheme with an upper limit of 12 degrees and a lower limit of 3 degrees.

$$\cos\theta_i = \frac{(x_{i-1} - x_i) \bullet (x_i - x_{i+1})}{|x_{i-1} - x_i|\,|x_i - x_{i+1}|} \tag{4}$$

Friedrich Weller [22], who describes general numerical approaches for ODEs, suggests calculating a value q by using the already computed weights k_1 to k_3 of the Runga-Kutta formula:

$$q = \left| \frac{k_3 - k_2}{k_2 - k_1} \right| \tag{5}$$

Furthermore, he suggests to double the step size if $q < 0.005$ and halve the step size if $q > 0.04$. The current step is then recalculated with this new step size. For $0.005 < q < 0.04$, the step size remains unchanged and the next step can be computed. In some cases, k_1 and k_2 are equal, thus, $|k_2 - k_1|$ is directly set to 0.01.

Weller's general approach works with scalars. It considers the fact that Runge-Kutta's intermediate steps already provide a measure for the growth of the gradient. Therefore, too large changes can be estimated by the weights k_i. Because we consider vectorial data, we applied his estimation for each cartesian direction; i.e., each component of k_i is treated separately and, for the 3 dimensional case, produces three q components. We increase the step size if all q components are less than 0.005 and reduce it if only one q component exceeds 0.04.

Press prefers the Runge-Kutta-Fehlberg method [16] to step doubling in order to solve ODEs. This robust embedded Runge-Kutta algorithm is more efficient than algorithms based on step doubling roughly by a factor of two. Press uses Runge-Kutta fifth order (Equation 6) and compares the result with the embedded fourth-order formula (Equation 7). The used constants c_i and c_i^* are given by Cash and Karp [16] and

are slightly more efficient than Fehlberg's original ones. The difference between both results is used as an error estimate and controls the modification of step sizes.

$$
\begin{aligned}
k_1 &= h \cdot v(x_i, t_i) \\
k_2 &= h \cdot v(x_i + b_{21}k_1,\ \ t_i + a_2 h) \\
k_3 &= h \cdot v(x_i + b_{31}k_1 + b_{32}k_2,\ \ t_i + a_3 h) \\
k_4 &= h \cdot v(x_i + b_{41}k_1 + b_{42}k_2 + b_{43}k_3,\ \ t_i + a_4 h) \\
k_5 &= h \cdot v(x_i + b_{51}k_1 + b_{52}k_2 + b_{53}k_3 + b_{54}k_4,\ \ t_i + a_5 h) \\
k_6 &= h \cdot v(x_i + b_{61}k_1 + b_{62}k_2 + b_{63}k_3 + b_{64}k_4 + b_{65}k_5,\ \ t_i + a_5 h)
\end{aligned}
$$

$$x_{i+1} = x_i + c_1k_1 + c_2k_2 + c_3k_3 + c_4k_4 + c_5k_5 + c_6k_6 \tag{6}$$

$$x^*_{i+1} = x_i + c^*_1k_1 + c^*_2k_2 + c^*_3k_3 + c^*_4k_4 + c^*_5k_5 + c^*_6k_6 \tag{7}$$

We implemented the scheme by Kenwright and Lane, and adapted the approaches by Weller and by Fehlberg for path line purposes. The results were compared with the original VTK results. The angle approach breaks off after some iterations because of the highly dynamical nature of our test data sets. Illegal deviations are recognized too late and dividing the step size only confirms the deviations. The other two approaches work well. The great advantage is that they merely need a few additional computations in order to verify the truncation error. Weller's approach is considerably faster since it needs less correction steps. Using the same start step size, it is only somewhat slower than the original VTK scheme.

All schemes used are so-called multi-stage schemes and use only the last known position for a single integration step. Multi-step methods [22, 16] are also available. The best-known schemes are Adam-Bashford, Adams-Moulton, and backwards differentiation. Frühauf [7] and Darmofal and Haimes [6] describe different aspects of these approaches and compare them with multi-stage schemes. The latter paper also addresses the start-up problem. The start-up error heavily influences the accuracy of the whole scheme. If the start-up error is already below a certain order, the scheme of higher order cannot achieve higher computation accuracy than the start-up order.

3.3. Particle Tracing in Unsteady Flows

In order to enable particle tracing for unsteady flows, `vtkStreamline` and `vtkStreamer` must be replaced. In addition, not only position interpolation is needed but time-dependent interpolation as well. A straightforward approach just utilizes the surrounding two time levels of the current particle and is shown as pseudo-code in Listing 1.

```
Compute seedPoints;
FOR (ALL seedPoints)
{
  curTime  = curTimeLevel + 0.0;
  curPos   = seedPoint;
  curSpeed = Speed ( Velocity(seedPoint) );
  WHILE (in DataSet)
  {
    // (1) search next pos
    firstProbePos  = RungeKutta (curPos, curTimeLevel);
    secondProbePos = RungeKutta (curPos, curTimeLevel+1);
    deltaTime = curTime - curTimeLevel;
    nextPos = (1 - deltaTime) * firstProbePos +
                deltaTime    * secondProbePos;

    // (2) calculate remaining params
    nextSpeed = Speed (Velocity[nextPos]);
    nextTime = curTime + (nextPos - curPos) * 2.0 /
                         (curSpeed + nextSpeed);

    // (3) store new particle pos
    AddPathlinePos (nextPos);

    // (4) prepare next loop
    curTime  = nextTime;
    curPos   = nextPos;
    curSpeed = nextSpeed;
  }
}
```

Listing 1. Path line algorithm involving two time levels.

This algorithm ignores the continuous time information and maps all Runge-Kutta computation steps to the last discrete time level (Figure 6). Additionally, the same approach is applied for a second probe using the next time level. Afterwards, the final particle position is interpolated between the obtained two probes considering the particle's new total integration time. However, one must take into account that both probes typically have slightly different step sizes. Therefore, to determine the new integration time, these values are also interpolated according to the last particle integration time.

A second and more accurate approach is the conservation of time information during the integration process. The Runge-Kutta classes already handle time information correctly. Therefore, our modified `vtkStreamer` now feeds Runge-Kutta with appropriate time data. However, `vtkInterpolatedVelocityField`, the object that provides us with flow

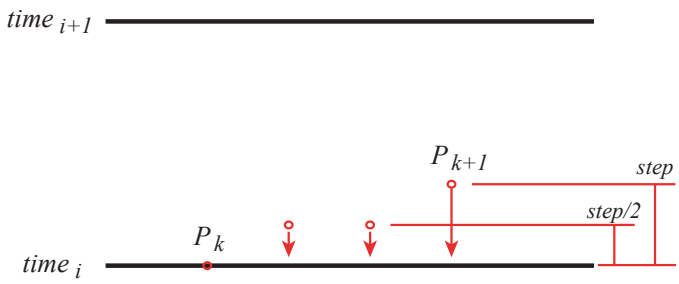


Figure 6. Time information is ignored and mapped to current time level.

field data at arbitrary positions, again ignores time information and interpolates in one data set only. Thus, our own interpolator substitution for the original one was also extended in order to enable path line computation. Currently, it needs data sets of different time levels at once to be able to interpolate on the time axis as well. Figure 7 shows a case in which more than the generally used two time levels are needed.

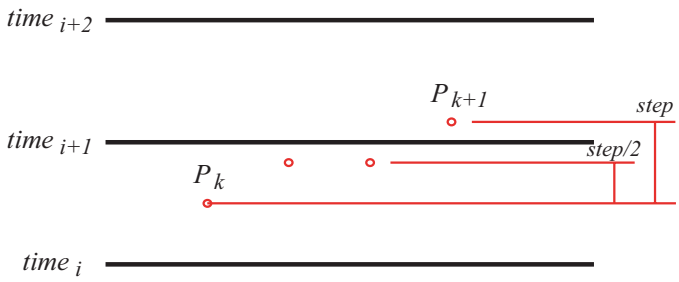


Figure 7. In this case, fourth-order Runge-Kutta needs three time levels at once.

4. Parallelization

Since particle integration makes use of previous particle positions, one particle trace cannot be split up into several independent pieces to be computed in parallel by different processes. Data parallelism is only possible if more than one path line is computed. In this case, each process is responsible for computing a given amount of path lines one by one.

VTK already offers multi-threading capabilities for streamline calculation. For a more generalized approach, which is applicable in shared memory systems as well as in distributed memory systems, we integrated the path line extraction into the parallelization framework of

our Virtual Reality toolkit *ViSTA* [8]. This framework is based on the Message Passing Interface MPI [9]. In order to relieve the visualization system, the CFD extraction computation is completely shifted to a high-performance multi-processor system (herein after referred to as work host). Now, the visualization host can send a request to compute path lines to this work host. Here, each process, called work node, assigned to this task computes one or more path lines. The results are sent back to the visualization host, where they can directly be displayed as polylines, moving particles, and *Virtual Tubelets* (see below).

Our test case starts with 8 particles on a seed line underneath one of the outlet valves. The start time is level 14 and all path lines are integrated to level 22. The path line integration that uses the first described interpolation approach and the different adaptive step-size control schemes listed above are applied. The results are matched visually and by distance comparison of the end positions of each path line. Furthermore, the complete computation time and the needed time steps are measured. One result is depicted in Figure 8.

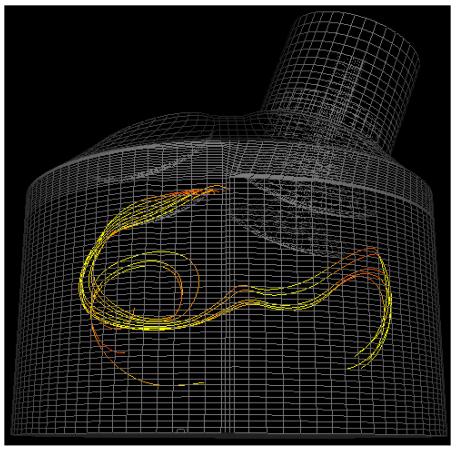


Figure 8. Path lines computed by Runge-Kutta-Fehlberg scheme, integration time level 14 to 22, start integration step size: 0.0482706.

Tests have been run on a high-performance Sun Fire 6800 SMP cluster from SUN Microsystems. It consists of 24 UltraSPARC-III 900 MHz processors connected to a shared memory. Each process of the Sun Fire 6800 can access up to 24 GByte shared main memory. The MPI implementation on the Sun Fire 6800 is optimized for SMPs, i.e. data packages are passed via shared memory.

Loading multi-block data files from a file server is one of the most time-consuming tasks. Each work node proceeds independently and reads its own data when needed. Therefore, blocks are read multiple times rather than being read once and being shared afterwards. For

SMP clusters, a shared usage is preferred; nevertheless, repeated access to stored files is the unavoidable drawback of distributed systems such as Linux clusters. The number of multiple loaded blocks depends on the seed points. In our measurements, the amount is rather high.

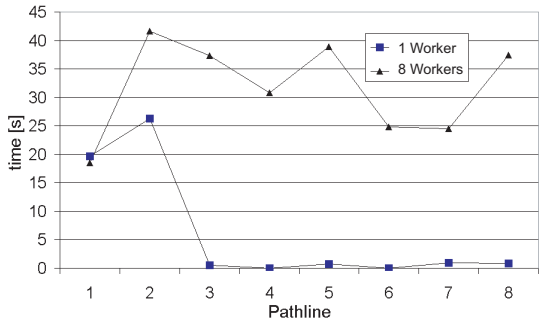


Figure 9. Time needed to load blocks using 1 processor (sequential load) and 8 processors (parallel load) to compute 8 path lines applying the VTK scheme.

Figure 9 illustrates the benefit of shared access to already loaded data files. If only one worker computes all path lines, most files have to be read in for the first two path lines. Thereafter, the remaining path lines can fall back on the already loaded files. The second, upper curve describes the time needed to load blocks when each worker merely computes one path line. The accumulated time of this curve would be identical to the accumulated time of the first curve if already loaded data blocks were shared!

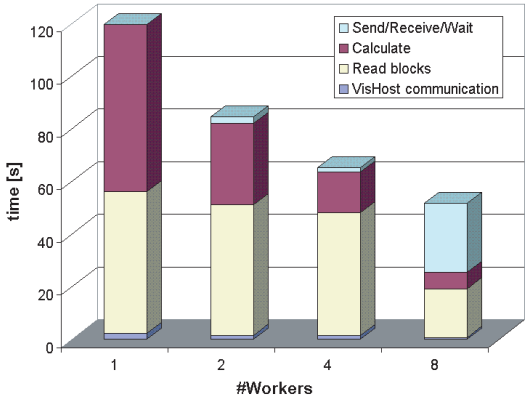


Figure 10. Needed computing time subdivided in essential parts.

Figure 10 shows the parallelization result. The speed-up is not entirely satisfying. The reason is, once again, the time needed to read

single data blocks from file servers. This affects load balancing as well. Aside from the irregularities, which occurred in the column for 8 workers, time needed to send data from one worker to another can nearly be ignored.

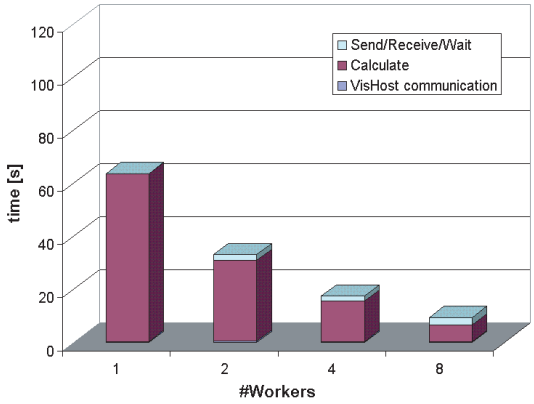


Figure 11. Needed computing time with prefetched data.

In order to verify this result, we load all blocks of all time levels into main memory, which is feasible only for small data sets. With this prefetched data sets, the result shows the expected speed-up (Figure 11). Nevertheless, communication time slightly reduces the optimum efficiency (Figure 12).

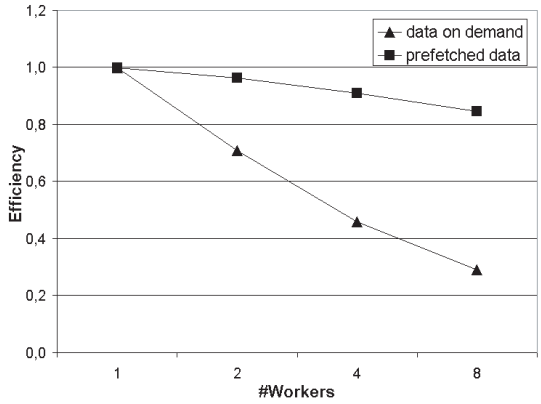


Figure 12. Efficiency.

5. Virtual Tubelets

Common drawbacks of rendering streamlines and path lines as simple line primitives include issues concerning depth perception and depth order of several crossing lines (see Figure 13). These pitfalls can be avoided by drawing path lines as solid tubes, which results in a significantly higher polygon count and reduces the maximum achievable frame rate. This, in turn, reduces the interactivity of a given visualization considerably.

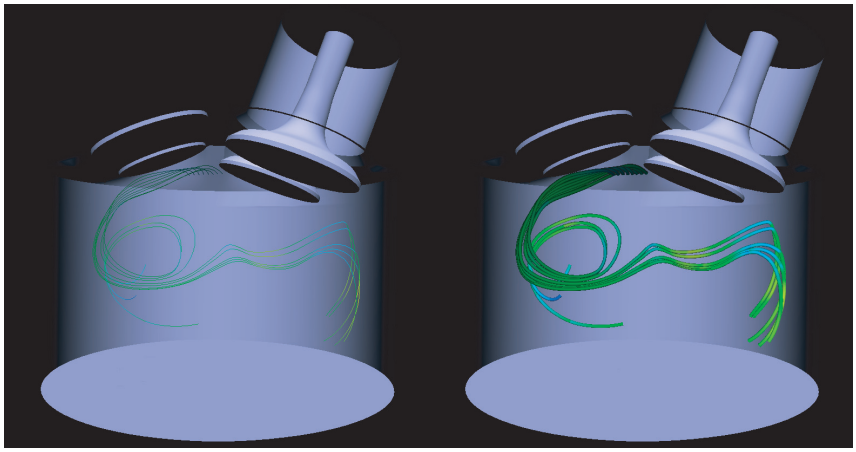


Figure 13. Rendering particle traces as lines (left) and *Virtual Tubelets* (right).

We avoid these problems by using and extending billboarding techniques [2] for an efficient depiction of the particles' trajectories, which is visually similar to stream tubes. These *Virtual Tubelets* consist of cylindrical, world-aligned billboards, which are rotated to face the viewer. The illusion of rounded, partly self-illuminated geometry is created via an appropriate texture map. A problem of this approach is a perception error at the beginning and at the end of every single tubelet. Due to their flat nature, the tubelets' heads do not have the rounded edges a viewer would expect, which results in the impression of the tubelet pointing away from the viewer (see Figure 14). We solved this problem by capping the tubelet with an additional quadrilateral, which is orthogonal to its first section, and which, again, is appropriately textured to maintain the illusion of rounded geometry.

To avoid a cluttering of the view space, we limit the tubelets' extents to a certain time window, which can be controlled interactively. Further hints about the particles' behavior in time are given by reducing the tubelets' luminance gradually to zero from their heads to their back

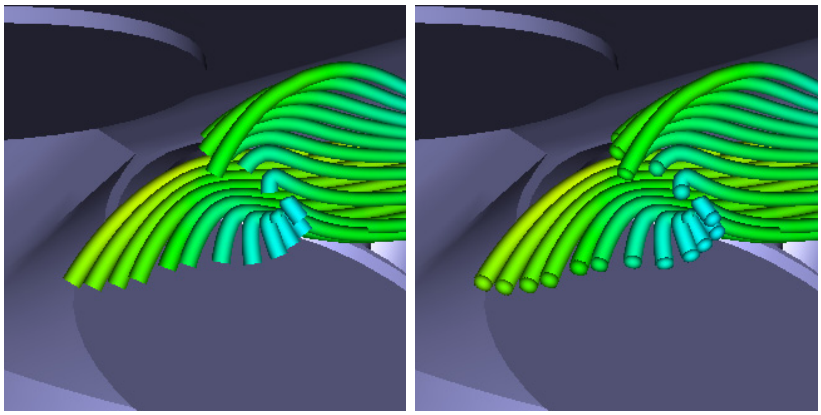


Figure 14. Perception errors (left) are avoided by capping the textured quadrilaterals (right).

ends. Furthermore, we use the tubelets' color to encode an additional scalar value, in this case the velocity of the traced particle.

Virtual Tubelets allow us to display the results of the parallelized computation of path lines described before in an intuitive way. By offering interactive control over certain aspects of the visualization, like the time window for the particle traces, we present a flexible method for the display of even large amounts of particle traces while still maintaining interactive frame rates.

6. Conclusions and Future Work

The main goals of our efforts were to relieve the visualization host and to speed up the computation of path lines. Both goals were achieved by shifting the computational load of visualization computations to dedicated HPC clusters where the multi-block structure of the given data was exploited through a suitable parallelization approach. However, our approach suffers from concurrent and repeated access to hard disk in order to read the required data. In our test case, the total speed up is reduced considerably. A possible solution for this problem is transfering the responsibility for hard disk access from the single worker nodes to a centralized data manager, which loads required data blocks on demand from disk, caches them, and offers them to the worker nodes. Such centralized data management is the obvious approach for SMP clusters like the Sun Fire cluster we used but poses non-trivial problems for distributed clusters. Nevertheless, due to increasing data set sizes, we will incorporate an integrated data management into our parallelization framework. This includes technologies like parallel I/O

and MPI-IO, which promise advantages even for distributed systems like Linux clusters.

Considering topology information is our main strategy to accomplish a considerable speed-up even in sequential path line computations. In our test setup, real-time interaction is achieved and the time to wait for CFD extraction results is reduced to acceptable amounts. In addition, our adaptive step-size control helps avoiding unnecessary attempts to find usable solutions and reduces the number of needed integration steps, thus resulting in an additional speed-up and less rounding errors. We can certainly achieve an even more accurate and faster algorithm by means of decomposing hexahedrons into tetrahedrons for the cell search. Currently, we are implementing this approach.

In the future, we will implement higher order interpolation kernels as well [14]. Although linear interpolation is a frequently used approach, e.g. cubic interpolation kernels can offer higher order continuities at the borders of neighboring cells, blocks, and time levels. At the moment, we parallelize the computation of whole path lines only. Higher order interpolation kernels could potentially be used for finer grained parallelization strategies; i.e. each probe could be calculated in parallel for each separate time level. Along with optimized strategies for the Sun Fire cluster, which make use of integrated cache programming and fast data access between worker nodes via shared memory, this could be achieved by employing OpenMP [4] in our parallelization framework.

Acknowledgements

The authors thank the Institute of Aerodynamics, Aachen University, for the combustion engine data set kindly made available.

References

1. Aschaf Abdelfattah. *Numerische Simulation von Strömungen in 2- und 4-Ventil-Motoren*. PhD thesis, RWTH Aachen, 1998.
2. Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A.K. Peters Ltd., 2nd edition, 2002.
3. Steve Bryson, David Kenwright, Michael Cox, David Ellsworth, and Robert Haimes. "Visual Exploring Gigabyte Data Sets in Real Time". *Communications of the ACM*, 42(8), August 1999.
4. Rohit Chandra, Leo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon. *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers, 2000.
5. Dave L. Darmofal and Robert Haimes. "Visualization of 3-D Vector Fields: Variation on a Stream". In *Proceedings, AIAA 92-0074*, 30. Aerospace Sciences Meeting and Exhibit, Reno, Nevada, January 1992.

6. Dave L. Darmofal and Robert Haimes. "An Analysis of 3-D Particle Path Integration Algorithms". In *Proceedings*, 1995 AIAA CFD Meeting, 1995.
7. Thomas Frühauf. *Graphisch-Interaktive Strömungsvisualisierung*. PhD thesis, TH Darmstadt, 1997.
8. Andreas Gerndt, Thomas van Reimersdahl, Torsten Kuhlen, and Christian Bischof. "A Parallel Approach for VR-based Visualization of CFD Data with PC Cluster". IMACS 20000, Lausanne, Swiss, 2000.
9. William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI - Portable Parallel Programming with the Message Passing Interface*. MIT Press, 2nd edition, 1999.
10. Roberto Grosso, Martin Schulz, Jan Kraheberger, and Thomas Ertl. "Flow Visualization for Multiblock Multigrid Simulations". In *Virtual Environments and Scientific Visualisation '96*, pages 296–307. Springer-Verlag, 1996.
11. David N. Kenwright and David A. Lane. "Optimization of Time-Dependent Particle Tracing Using Tetrahedral Decomposition". In *Proceedings of Visualization '95*, Los Alamitos, CA, 1995.
12. David A. Lane. "UFAT - A Particle Tracer for Time-Dependent Flow Fields". In D. Bergeron and A. Kaufman, editors, *Proceedings of Visualization '94*, Washington, D.C., 1994.
13. David A. Lane. "Scientific Visualization of Large-Scale Unsteady Fluid Flow". In G. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization - Overviews, Methodologies, Techniques*, pp. 59–101. IEEE Computer Society Press, 1997.
14. Barthold Lichtenbelt, Randy Crane, and Shaz Naqvi. *Introduction to Volume Rendering*. Hewlett-Packard Professional Books. Prentice Hall, 1998.
15. Gregory M. Nielson and Il-Hong Jung. "Tools for Computing Tangent Curves for Linearly Varying Vector Fields over Tetrahedral Domains". *IEEE Transactions on Visualization and Computer Graphics*, 5(4), 1999.
16. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++*. Cambridge University Press, 2nd edition, 2002.
17. Thomas van Reimersdahl, Torsten Kuhlen, Andreas Gerndt, Jörg Henrichs, and Christian Bischof. "ViSTA: a multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI". Proceedings of Fourth International Immersive Projection Technology Workshop (IPT2000), Ames, Iowa, 2000.
18. I. Ari Sadarjoen, Theo van Walsum, Frits H. Post, and Andrea J. S. Hin. "Particle Tracing Algorithms for 3D Curvilinear Grids". 5th EuroGraphics Workshop on Visualization in Scientific Computing, Rostock, Germany, 1994.
19. Marc Schirski, Andreas Gerndt, Thomas van Reimersdahl, Torsten Kuhlen, Philipp Adomeit, Oliver Lang, Stefan Pischinger, and Christian Bischof. "ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments". Seventh International Immersive Projection Technology Workshop (IPT2003), Zurich, Swiss, 2003.
20. William J. Schroeder, editor. *The VTK User's Guide*. Kitware Inc., 2001.
21. Christian Teitzel, Roberto Grosso, and Thomas Ertl. "Efficient and Reliable Integration Methods for Particle Tracing in Unsteady Flows on Discrete Meshes". In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97*, pp. 31–41, Wien, April 1997.
22. Friedrich Weller. *Numerische Mathematik für Ingenieure und Naturwissenschaftler - Eine Einführung für Studium und Praxis*. Vieweg, 1996.