Markov Prefetching in Multi-Block Particle Tracing

Marc Wolter*[1], Andreas Gerndt[1], Torsten Kuhlen[1] and Christian Bischof[2]

[1]Virtual Reality Group, RWTH Aachen University
[2]Center for Computing and Communications, RWTH Aachen University
Seffenterweg 23, 52064 Aachen, Germany
*E-mail: wolter@rz.rwth-aachen.de - Web page: http://www.rz.rwth-aachen.de/vr

ABSTRACT

Visualization of flow phenomena in large, time varying data sets demands extensive graphical, computational and memory resources. Recent research in the field of parallel post-processing algorithms and out-of-core data management shows that a good utilization of given resources together with a considerable speed-up may be achieved. One of the main problems is the I/O bottleneck commonly alleviated by data caching and overlapping I/O with computation. Standard prefetching techniques suffice for predictable data demands but fail for more unpredictable requests, e.g. in particle tracing. Therefore, we introduce a Markov prefetcher on application level for predicting data requests in multi-block data sets. Our prefetcher uses former request behavior as well as various information about the flow field and the data set's topology graph to provide required data almost without delay. We achieve a reduction of data access and time lags when applying Markov prefetching with tracing particles in different data sets.

1. INTRODUCTION

In general, post-processing of large-scale data sets is a time-consuming task. In order to enable explorative analysis of Computational Fluid Dynamics (CFD) data in interactive environments, a common approach for a considerable speed-up is parallelization. However, quite often the size of main memory is also a strong constraint, which led to the development of out-of-core strategies. The general idea behind this approach is that only that data portion is loaded into main memory which is really needed for current computation tasks. A rather straightforward out-of-core algorithm is based on multi-block (MB) data sets. In such data structures, the flow field is already spatially clustered into disjunctive regions. Therefore, only the block that contains the currently needed data cells must be loaded from file system. But loading data on demand can be one of the most fundamental bottlenecks in a post-processing system as the algorithm must stop the computation and wait for new data. In a parallel environment, these interruptions typically result in scaling and balancing problems.

In order to alleviate this problem for complex stream- or pathline computations, this paper presents a new prefetching approach for particle tracing that uses Markov processes to predict the data needed next. This prediction is initiated by probabilities determined either by random seed samples applied offline or by topological heuristics. During the online integration, the actually computed trajectory of the particle is continuously evaluated in order to modify the probability graph. Using this kind of meta-information, in an optimum overlapping case, particle integration can now work without delay, as selected data blocks are prefetched in time.

The following section presents previous work in the area of particle tracing and prefetching strategies. Thereafter, in section 3, multi-block data sets and the use of their topology for parallel particle integration is briefly explained. Section 4 introduces the Markov prefetcher for multi-block CFD data sets, which has been implemented in our framework to overlap particle

tracing computation with loading of predicted blocks. Finally, the achieved results are presented and evaluated in section 5.

2. PREVIOUS WORK

Visualization algorithms for large disk-resident data sets have been studied by many research groups. In particular, out-of-core solutions for various feature extraction methods like isosurface or particle tracing have been developed.

Prefetching techniques are primarily designed and analyzed for computer or file systems but may also be adapted to most application level caching systems. A Markov prefetching for computer systems that can be easily integrated in existing systems and prefetches multiple predictions is described in [6]. Doshi et al. present adaptive prefetching techniques for visual data exploration [2]. Different prefetching schemes are implemented and the appropriate combination of strategies is dynamically selected according to the user's exploration pattern.

Visualization of time-varying fluid flows using large amounts of particles is facilitated by the UFAT (Unsteady Flow Analysis Toolkit) software [7]. UFAT provides instantaneous stream-, path- and streaklines for unsteady single and multi-block data sets. Depending on the underlying disk system, the I/O time accounts up to 25% of the execution time. Application controlled demand paging was investigated with UFAT by Cox and Ellsworth [1]. They point out trashing problems when relying on virtual memory and describe a paged segment out-of-core technique to take advantage of existing main memory.

Interactive out-of-core computation of streamlines is done by Ueng et al. [9]. Unstructured grids are restructured by an octree, and required data in the main memory is extremely reduced by on-demand loading of new octant blocks. Block extents are bound by a fixed set of different sizes, so-called block size levels.

3. MULTI-BLOCK TOPOLOGY

The type of grids generated for CFD is not only important for the quality of the simulation but also for the efficient design of post-processing algorithms. In this paper, we focus on multi-block data sets consisting of several structured grids, each stored separately in a file. For inter-block connectivity, we furthermore apply topology information in order to speed-up the determination of neighbors. This topology is also stored as metadata in a separate file [4].

Additionally to simple block-to-block connectivity, the knowledge which cells are actually adjacent is considered. In general, a group of cell sides of one block borders to another group of cell sides of a second block. Therefore, not each cell link but a so-called *connection window* is stored. It is defined by means of *i*-*j*-*k*-indices of two grid nodes lying diagonally at corners of that window, which must be restricted to one side of a hexahedral block (see figure 1).

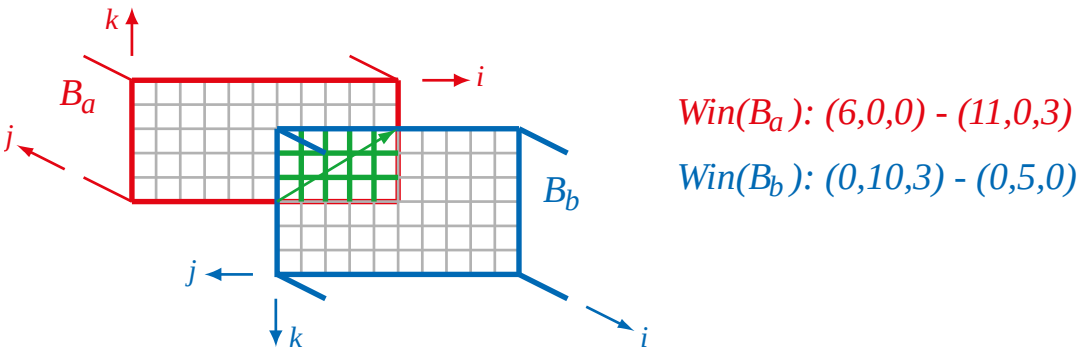

Figure 1: Connection window between block B_a and B_b.

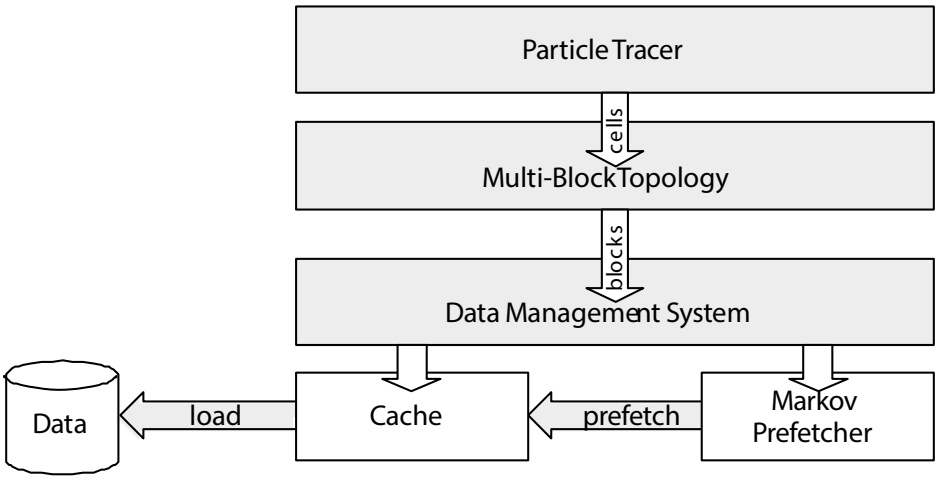

Figure 2: Software scheme of the post-processing toolkit.

As the Markov prefetcher is a general approach, it is embedded into an existent data management system. For interactive exploration of unsteady CFD data sets in virtual environments, the software framework ViSTA FlowLib [8] is applied. The extraction of flow features from large CFD data sets is done on a remote post-processor called Viracocha [3], connected with ViSTA FlowLib via a standard TCP/IP network. This decoupling allows the use of specialized hardware for different tasks: high-end graphics cards on the visualization workstation, and high-performance, large memory computer systems for feature extraction. The software scheme of the particle tracer on each parallel node is depicted in figure 2. All block requests from multi-block topology are answered by Viracocha's data management system, which prefetches files into the cache according to the markov prefetcher's predictions.

4. MARKOV PREFETCHING

With predictable access patterns, Viracocha achieves quite good results by the use of simple OBL prefetching [3]. But when algorithms with unpredictable access patterns are applied, the sequential prefetcher has almost no effect on the reduction of cache misses. To provide useful prefetching for unpredictable patterns, we now introduce the Markov prefetcher for multi-block data sets.

A Markov predictor of i^{th} order uses the i last queried blocks to predict the next request, i.e., it utilizes the probability $P(X_t = j | X_{t-i} = i_{t-i}, \ldots, X_{t-1} = i_{t-1})$. The simplest variant of first order chooses the next prefetch only on basis of the last block requested.

The prefetcher may be an independent unit of the software framework. As input, it works on the stream of block requests from an algorithm. Using this stream, the prefetcher builds a probability graph for the succession of blocks. The output of the prefetching unit is at every time a set of predicted blocks which are possible successors for the current block. Since we are dealing with multi-block data sets, in the current implementation, only the most likely successor is used as prefetch prediction. Other Markov prefetchers e.g. in [6], prefetch several proposed predictions, but they work on small, equal sized blocks. Prefetching a set of blocks increases the I/O load in our system considerably due to the size and imbalance of our blocks.

While the Markov probability graph provides good results after a certain runtime, which is needed to adopt to the overall request behavior, the probability graph is empty after the system starts. In these cases, the implemented Markov prefetcher makes simply no predictions. This results in non-overlapping of computation and I/O. Even sequential prefetching works better in the starting phase, since it makes predictions and some are actually hits. This drawback leads to the introduction of the external initializing for Markov prefetchers.

4.1. External Initializing

Initializing the Markov prefetcher is either done by simulating a request stream by an external resource or by loading a Markov graph filled with topology information. This section deals with the second possibility how to use existing or precomputed topology information to obtain a ”good” Markov graph, i.e. a graph which results in many successful and accurate prefetches.

We introduce two approaches for multi-block CFD data: a connection window based heuristic and precomputed flow statistics. Both methods’ goal is to generate a probability distribution that describes the flow field in an appropriate way.

The **connection window heuristic** uses the already existent topological information. The multi-block structure described in section 3 contains connection windows for every neighboring block. A particle movement between blocks has to be performed via a connection window, which may be accessed using the topology meta-data. Therefore, we use the block topology graph as Markov graph, where each topology edge is divided into two opposed directed edges as connection windows may be passed from both sides. The graph’s edges are labeled with the transition probability proportional to the size of the connecting windows. We define the size of a connection window with window coordinates i_{min}, j_{min}, k_{min} and i_{max}, j_{max}, k_{max} respectively, as product of the two non-zero terms $(i_{max} - i_{min})$, $(j_{max} - j_{min})$, $(k_{max} - k_{min})$. This Markov graph is easy to compute but involves only the topological information of the grid. The behavior of the flow is completely disregarded, which leads to the second approach.

Precomputed flow statistics are generated using an iterative algorithm. In an initialization phase, the probabilities are set according to a uniform distribution, i.e., every neighboring block has the same probability to be chosen. For each block, a number of seed points is generated and traced until they leave the current block. Thereafter the algorithm determines whether the particle left the whole data set or entered a new block. In the latter case, the probability distribution will be updated. The number of seed points depends on two termination conditions: first, a total maximal number of seed points may not be exceeded in order to have some control over the total runtime of the preprocessing algorithm. Second, the algorithm terminates if the probability distribution stays unchanged for several seeding steps. This criterion stops once the distribution has reached some kind of balanced state, being the favored result.

Both offline computed probability distributions describe the flow field of one time level in a heuristic way. For initializing streamline computation, these probabilities are inserted directly in the Markov prefetcher’s graph. When computing pathlines, we consider the time-dependency by inserting weighted edges to the next time step using fixed weights for each data set.

5 RESULTS

In this section, we present and discuss the results of the methods introduced before. For the evaluation of the presented approaches, we currently use two different multi-block data sets. The nose data set is used to compute 50 particle traces in 30 time levels (see figure 3). 8 trajectories in 9 time levels are integrated in the engine data set (see figure 4). All results are computed on a SunFire v880z with 4 GB main memory and four Ultra Sparc III Cu processors. While both data sets are small enough to fit into main memory completely, their different fields are suitable to show the behavior of the Markov prefetcher.

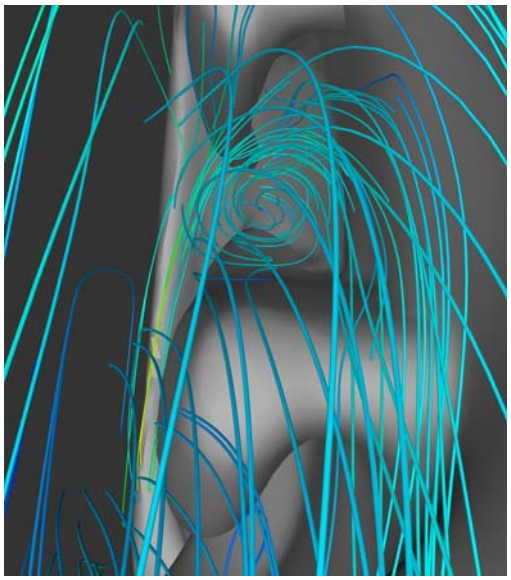

Figure 3: Inside view of streamlines through the nose data set.

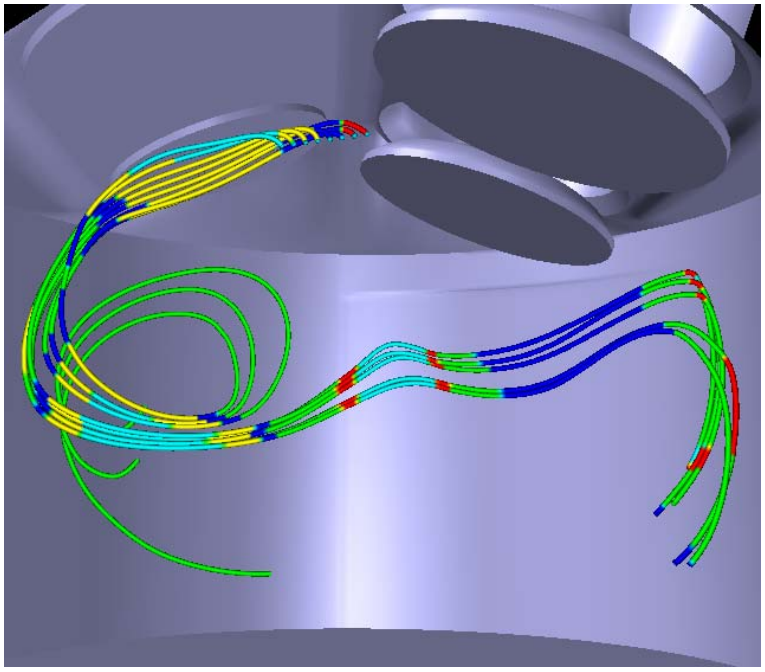

Figure 4: Pathlines in the engine data set, coloring specifies block changes.

The nose data set represents the flow field inside an artificial human nasal cavity [5]. The domain decomposition is time-invariant, i.e., the geometrical definition of used grids does not change over time. Furthermore, during the inspiration and expiration phase, the air flow within a nose shows a dominant direction. Each of the 165 time levels is divided into 34 blocks containing 443,000 points. The engine data set depicts the inflow and compression phase of a four-valve combustion engine. In contrast to the multi-block geometry of the nose, this data set is a moving grid, i.e., the geometry and dimension of defining blocks change now with each successive time level. Besides, an obvious flow direction is hardly determinable. An average of 200,000 points is contained in 23 blocks over 62 time levels.

5.1 Multi-Block

The on-demand multi-block method's main goal is to reduce the number of loaded blocks. The values in figure 5 indicate the percentage of blocks loaded on demand compared to total blocks.

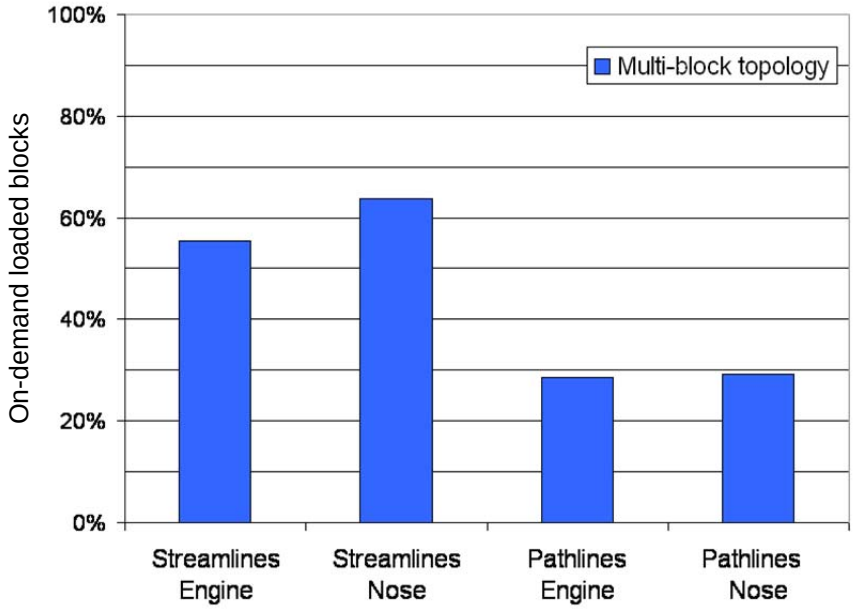

Figure 5: Percentage of on-demand blocks using multi-block topology.

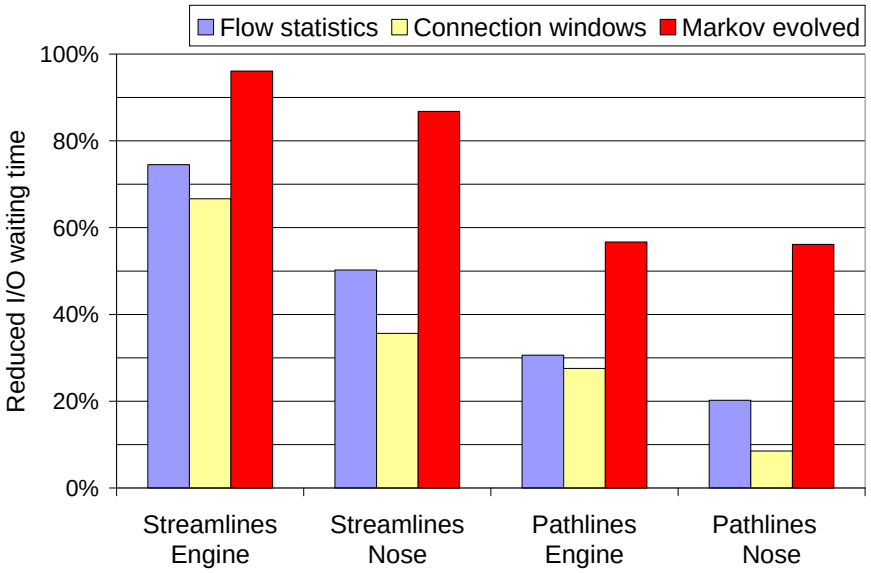

Figure 6: Saved loadtime (LT) for evolved and externally initialized Markov graphs.

When computing streamlines, we saved nearly half of the loading operations. With pathlines, the achieved results are even better as the absolute amount of saved blocks is higher in pathline than in streamline extraction. This is because the total amount of possible blocks is *block * timesteps* and therefore grows with used time levels instead of the fix number of blocks in one time level.

5.2 Prefetching

To judge the efficiency of our implemented prefetching methods, we use several measures:

1. Prefetch coverage = useful prefetches / (useful prefetches + cache misses): indicates the fraction of removed cache-misses due to prefetching.
2. Prefetch efficiency = useful prefetches / total prefetches: describes the prefetching accuracy, i.e. how many prefetches could effectively be used.

3. Loadtime (LT) saving refers to the saved time the computation thread has to wait for I/O. This is a criterion for the quality of prefetch overlapping.

While the first two values are more structural, the latter values measure the real gain using the prefetching method. With reduced loadtime (i.e. computation waiting for I/O), the parallel algorithms are better balanceable and the user's waiting time decreases.

Three test series were performed. The first one labeled *Markov runtime evolved* starts with an empty Markov graph. Then it evolves from consecutive particle computations with similar seed points. This behavior resembles an exploring user during runtime examining a special part of the flow field. The measurement itself results from a single particle trace at the end of the exploring phase. The prefetching measures for the runtime-evolved Markov prefetcher are shown in table 1. Efficient prefetches are made, which reduces most cache misses. Efficiency is highly fluctuating with used seed points.

Table 1: Prefetching measures for all three series.

	Engine Streamlines	Pathlines	Nose Streamlines	Pathlines
Markov runtime evolved				
Coverage	92 %	57 %	79 %	72 %
Efficiency	100 %	74 %	90 %	87 %
Flow statistics series				
Coverage	46 %	23 %	41 %	26 %
Efficiency	60 %	47 %	53 %	24 %
Connection windows				
Coverage	17 %	18 %	27 %	29%
Efficiency	40 %	38 %	55 %	35 %

The *Flow Statistic* and *Connection Windows* series present values from a single particle trace with a respectively initialized Markov graph. The initialization has been executed on an empty graph. Side-effects from previous particle traces are excluded, so that the series depict the pure benefit from an external initialization. As shown in table 1, the efficiencies for both external initialized particle traces are at most 60 %. However, up to 46 % of all cache misses are reduced by the flow statistic method. This is a quite good result regarding the simple heuristic we used. Tracing in the more turbulent engine data set, the flow statistics approach beats the heuristic method. This may be due to the bigger connection windows (more than 72 %) in the engine's topology which do not correspond to the main flow direction.

The savings in loading time for particle tracing using just initialized graphs are depicted in figure 6. Regarding waiting time for I/O, both methods reduce loading time, in particular when applied to streamline calculation. The inferior I/O time reduction with pathlines in contrast to streamlines may be ascribed to the time-dependent nature of pathlines, while the used topology characterizes a single time-level only.

6. CONCLUSION AND FUTURE WORK

We presented a Markov prefetcher as an optional component of the parallel post-processor Viracocha. The prefetcher is used for predicting data requests in multi-block particle tracing. By the exploitation of auxiliary multi-block topology meta-data, the particle integration algorithm already diminishes the amount of blocks to be loaded. The prefetcher takes care of an optimum overlapping of I/O and computation.

The remaining I/O waiting time is reduced considerably when using the runtime evolved Markov prefetcher. To be efficient even when the post-processing framework is started or new data sets are selected, different approaches of external Markov graph initializations are

integrated. This yields a substantial improvement in comparison to uninitialized Markov prefetching or sequential prefetching strategies.

One shortcoming of the applied MB meta-data is that it merely considers topology information on time-levels separately. This can occasionally result in inadequate Markov initializations for time-variant algorithms. Therefore, in the future, we will also regard MB topologies between time levels, which could result in improved predictions especially for moving grids. Currently, we are working on more efficient storage of and access to simulation data. Tree based indexing structures fir fast loading of cells or meta-cells instead of blocks allow for a more granular access. We will evaluate our prefetcher for these structures, too.

ACKNOWLEDGEMENTS

The authors are grateful to the Institute of Aerodynamics, Aachen University, for the combustion engine and nose data set kindly made available. We would also like to thank the German Research Foundation (DFG), who kindly funded some of the methodical work under grand WE 2186/5.

REFERENCES

[1] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings IEEE Visualization '97*, pages 235–244, 1997.

[2] P. R. Doshi, G. E. Rosario, E. A. Rundensteiner, and M. O. Ward. A strategy selection framework for adaptive prefetching in data visualization. In *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pages 107–116, Cambridge,Massachusetts, 2003.

[3] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. Bischof. VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *Proceedingsof the IEEE SuperComputing (SC2004)*, November 2004.

[4] A. Gerndt, M. Schirski, T. Kuhlen, and C. Bischof. Parallel calculation of accurate path lines in virtual environments through exploitation of multi-block CFD data set topology. *Journal of Mathematical Modelling* and Algorithms (JMMA 2005), 1(4):35–52, 2005.

[5] B. Hentschel, T. Kuhlen, and C. Bischof. VRhino II: Flow field visualization inside the human nasal cavity. In Proceedings of the IEEE VR 2005, pages. 233-236, March 2005.

[6] D. Joseph and D. Grunwald. Prefetching using markov predictors. *IEEE Transactions on Computers*, 48(2):121–133, 1999.

[7] D. A. Lane. UFAT: a particle tracer for time-dependent flow fields. In *Proceedings IEEE Visualization '94*, pages 257–264, Washington, D.C., 1994.

[8] M. Schirski, A. Gerndt, T. van Reimersdahl, T. Kuhlen, P. Adomeit, O. Lang, S. Pischinger, and C. Bischof. ViSTA FlowLib - a framework for interactive visualization and exploration of unsteady flows in virtual environments. In *Proceedings of the 7th Internationial Immersive Projection Technology Workshop and 9th Eurographics Workshop on Virtual Environment*, pages 241–246. ACM SIGGRAPH, May 2003.

[9] S.-K. Ueng, C. Sikorski, and K.-L. Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997.