Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Time Step Prioritising in Parallel Feature Extraction on Unsteady Simulation Data

M. Wolter[1], B. Hentschel[1], M. Schirski[1], A. Gerndt[1], T. Kuhlen[1]

[1]Virtual Reality Group, RWTH Aachen University

Abstract

Explorative analysis of unsteady computational fluid dynamics (CFD) simulations requires a fast extraction of flow features. For time-varying data, the extraction algorithm has to be executed for each time step in the period under observation. Even when parallelised on a remote high performance computer, the user's waiting time still exceeds interactivity criteria for large data sets. Moreover, computations are generally performed in a fixed order, not taking into account the importance of partial results for the user's investigation.

In this paper we propose a general method to guide parallel feature extraction on unsteady data sets in order to assist the user during the explorative analysis even though interactive response times might not be available. By re-ordering of single time step computations, the order in which features are provided is arranged according to the user's exploration process. We describe three different concepts based on typical user behaviours. Using this approach, parallel extraction of unsteady features is enhanced for arbitrary extraction methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Parallel Processing I.3.2 [Computer Graphics]: Distributed/Network Graphics I.3.7 [Computer Graphics]: Virtual Reality I.6.6 [Simulation and Modeling]: Simulation Output Analysis

1. Introduction

Modern simulations of natural or technical phenomena, in particular computational fluid dynamics (CFD), exhibit an increasing temporal and spatial resolution. As raw data is too large in size and not directly displayable, renderable *extracts* have to be computed. Extracts are derived objects which provide insight into the data, e.g. isosurfaces, cutplanes or streamlines. Which kind of extracts are selected depends on the problem under investigation.

In this context, van Dam et al. [vDFL*00] proposed the employment of Immersive Virtual Reality (IVR) which combines interactive visualisation with immersive sensation (see figure 1). Interactivity provided by virtual environments supports an exploration of the raw data, which resembles a discovery rather than a mere presentation. This approach, also called explorative analysis, works directly on the simulation data set with the aim to extract characteristic structures as fast as possible, to set up hypotheses, or to gain further insights. The fundamental procedure is a trial and error approach. The user iteratively determines a set of visualisation parameters until a comprehension of the flow characteristics is attained.

Explorative analysis heavily relies on interactivity. We refer to [BJ96], where interactivity is defined as the ability to provide a response to the user's input in less than 100 ms and to maintain a minimal framerate of the application of 10 frames per second. [BJ96] additionally state that computational results should be presented in less than 500 ms. Providing interactivity with large data sets is a challenging task, especially if the data set consists of a large number of discrete time steps. While in some cases a precomputation of features or feature related meta-data is possible, this meta-data is mostly constricting and the amount of possible precomputations may cause secondary storage problems.

One common approach to speed up the extraction process is to use parallel computation on high performance computers, which provide adequate resources in the form of CPUs, main memory and high secondary storage bandwidth. The Viracocha toolkit [GHW*04] separates post-processing on a parallel computer from visualisation on a graphics worksta-

tion. In this work we apply Viracocha for the parallelisation of transient feature extraction. Time steps are dynamically assigned to processes and management tasks are overlapped with computation. The integrated data management system controls loading and caching of large data sets. Parallelising time steps, we achieve good scalability for different extraction methods and unsteady data sets.

While this approach reduces the overall computation time especially for large data sets, the user's waiting time mainly depends on the complexity of the extraction algorithm as well as the applied data set. For most common extraction algorithms, optimised computation methods and data structures for single time step computations exist. When executed on several time steps, the order in which single computations are processed influences the user's perception of the unsteady feature. If the order of visualisation and computation diverge, additional waiting time is induced. This is due to the fact that results for a given time step might be available long before they are displayed in the animation loop.

In this paper, we propose a method to enhance explorative analysis of unsteady data sets by responding to the temporal characteristics of the visualization. Therefore, waiting times become less noticeable for the user. Depending on the user's investigation behaviour, priorities are assigned to subtasks of the overall computation. These priority values change the arrival order and therefore the arrival time of computed results. This method only affects the scheduling of single time step computations and is independent of the underlying extraction algorithm. Based on the general concept, we introduce solutions for three typical use cases:

- The user focusses his search by specifying a certain time region of interest.
- The user stops the animation to investigate a single time step and its temporal neighbourhood.
- The user demands a continuous visualisation of a feature in order to investigate a feature's temporal evolution.

The remainder of this paper is structured as follows: In section 2 we briefly review previous and related work. In section 3 we describe the framework used to extract features from unsteady data sets in parallel. The results we present for different extraction algorithms motivate the need for more sophisticated methods to facilitate exploration. In section 4 we introduce our method of prioritising different tasks according to the user's interaction. This includes concepts for handling the three use cases outlined above. A conclusion and outlook is given in section 5.

2. Previous work

One of the first available systems for VR-based flow visualisation was the Virtual Wind Tunnel and its follow-up Distributed Virtual Wind Tunnel [BGY92]. The latter introduced a connection to a vectorized post-processing backend, which then was responsible for post-processing computations. Recently, Allard et al. introduced FlowVR [AGL*04], a middleware which can be used to flexibly connect various so-called modules to form a distributed VR-application. One goal of their work is to alleviate software engineering problems when writing VR-applications for cluster platforms. By distributing several identical modules to different nodes in a cluster, data parallel execution of modules can be obtained. According to the authors, the architecture can be applied to a wide variety of problems.

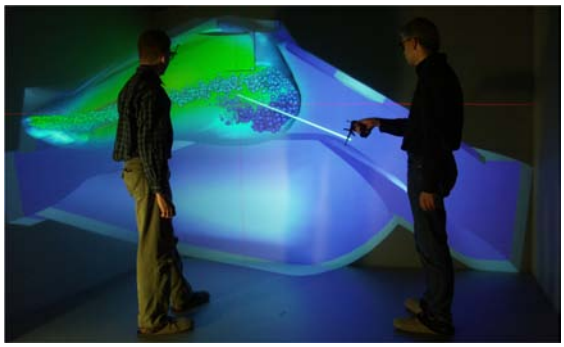

Figure 1: *Explorative analysis of CFD simulations in virtual environments using ViSTA FlowLib.*

Another distributed software environment is COVISE (COllaborative VIsualization and Simulation Environment) [RFL*98], which focuses on cooperative work. One or more users participate in a session controlled by a master user. Modules containing processing steps like I/O, filtering or rendering can be distributed across different workstations. COVISE also integrates a module for Virtual Reality called COVER, supporting tracking systems and some multi-screen display devices.

Ma and Camp [MC00] introduced a system for parallel remote rendering of unsteady data sets. Simulation data is volume rendered on a parallel computer and the resulting images are efficiently transmitted to some viewer application. This work uses compression mechanisms to provide fast image transmission using custom networks. Parallelisation is done for spatial and temporal extents of the data set.

The ViSTA FlowLib [SGvR*03] toolkit is developed for scientific visualisation in virtual environments. It combines the capabilities of the VR toolkit ViSTA [vRKG*00] and research activities in the area of CFD post-processing, which are based on functionalities of the Visualisation Toolkit (VTK).

3. Parallel Post-Processing of Unsteady Data Sets

This section describes the architecture used to parallelise feature extraction on unsteady data sets for explorative analysis in virtual environments. We evaluate the parallel

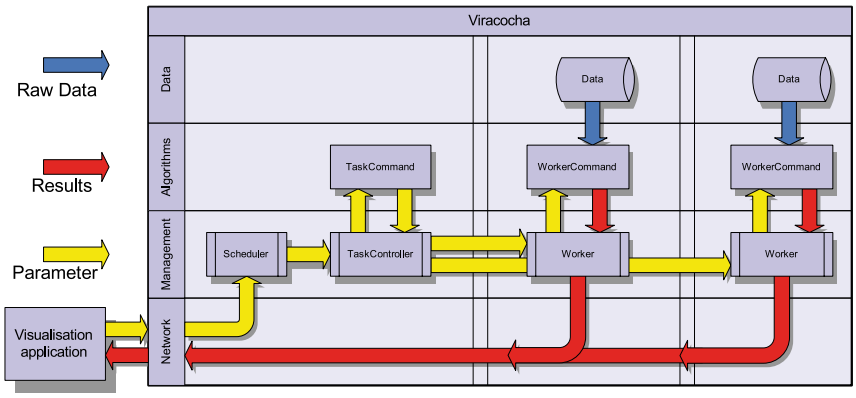

Figure 2: *Viracocha setup and workflow of a typical command. The double lines depict process borders. The management, algorithm and data layers (denoted left) use separate threads to overlap different subtasks.*

post-processor with extracts of different complexity on two time-varying data sets.

3.1. Parallel Architecture

The Viracocha software provides parallel post-processing for connected visualisation applications in a client-server setup, with Viracocha as server component. It makes use of a layered abstraction design that is made up of four layers (see figure 2). The layers hide implementation details and are exchangeable. For example, on the network layer, processes of the parallelisation framework communicate via MPI [GLS99], while requests and data are sent via TCP/IP between visualisation client and Viracocha. The other three layers attend to manage parallel resources, algorithms or data.

Viracocha combines different forms of parallelisation. The unsteady extraction task is parallelised using MPI, therefore it may be distributed even on heterogenous distributed memory machines. Subtasks of each layer, such as loading data, computing the extraction algorithm or management are parallelised using different threads. A Viracocha application running on a parallel computer called *workhost* consists of one *scheduler* process as central organising unit and multiple *worker* processes. The user of the visualisation application may issue *commands* together with a set of parameters to the workhost. Each new command is assigned to a *taskcontroller*, which assigns needed resources for the execution of the command and gathers a group of *worker* nodes. While this workflow is part of the framework, a user may integrate own methods into the algorithmic layer. These methods are called *taskcommand* on a taskcontroller and *workercommand* on a worker node accordingly. While the workercommand contains the concrete feature extraction algorithm, the taskcommand contains a control algorithm used to coordinate these extractions.

Figure 2 depicts the workflow of an issued command. The command and its associated parameters (yellow) are initially assigned to a taskcontroller and a connected group of workers. The taskcommand may alter the parameters prior to distribution. Based upon the parameters, the workercommand requests required raw data (blue) and produces some sort of result (red). The results are transmitted using the transport layer to the requesting visualisation application.

For extraction of unsteady features, we apply parallelisation on distinct time steps. Each worker computes a number of time steps iteratively. That is, after an initial setup phase including communication and thread setup, the following computations use the already existing objects, furthermore reducing overhead.

Load balancing is applied with a self scheduling strategy on time steps. As we do not balance single time step computations, load imbalancing may occur when execution times for different time steps vary significantly. This imbalance is adjusted when enough time steps are distributed. All time steps to be processed are managed by the taskcommand, which assigns them to free workers. Whenever the user issues a parameter update of a running command, e.g. changing the isovalue, the same workflow occurs.

3.2. Visualisation in Virtual Environments

Virtual reality applications have special requirements including stereoscopy, tracking and 3D input devices. For analysis of simulation data in virtual environments, we apply the ViSTA FlowLib toolkit as visualisation application. Viracocha handles all computationally expensive tasks requested by the user in a virtual environment. This decoupling enables ViSTA FlowLib to focus on visualisation with high frame rates. Simple tasks like head tracking, transformation or navigation are interactively processed. All tasks that would lower the framerate are sourced out to the remote parallelisation service.

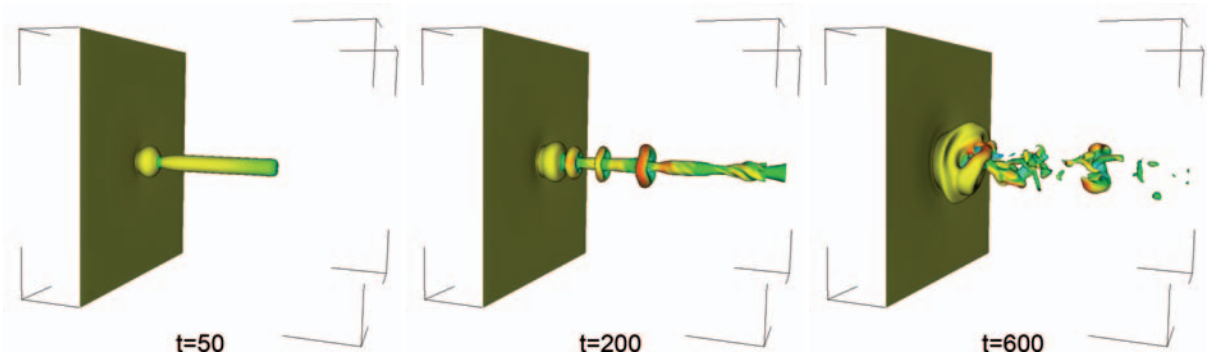

Figure 3: *Dynamic evolution of a pressure isosurface in the shock data set over several time steps. The isosurface is coloured by mach number.*

In this work we do not regard overhead on the visualisation computer. This includes sending a command, receiving and preprocessing resulting data for visualisation. These tasks are processed by a multi-threaded approach as well, to avoid delaying the computing worker.

But, rendering of large results is not a trivial task. Resulting data may be too large to be rendered effectively. Several different approaches for these problems exist, which often depend on the type of extract. In the scope of this work, we assume that produced results are small enough to be rendered immediately. The extracts we use in section 3.3 are only a few MB in size.

3.3. Evaluation Setup

Two large data sets were used to evaluate the parallelisation. The first data set called *shock* consists of 919 time steps of a rectilinear grid. Each time step contains approximately two million grid points, added up to a total filesize of 70 GB. The simulation data describes an ultrasonic shock induction. Goal of the simulation is to investigate the vortex structures depending on the induction angle. Extracts of several time steps are depicted in figure 3. The second data set called *propfan* is made up of only 50 time steps, each with 2.5 million points in an unstructured grid. It simulates a counter-rotating propulsion turbine. With every time step the blades are rotated a few degrees, therefore the data set has a moving grid. The total file size is 9.5 GB.

To show the efficiency of our framework we choose two exemplary extracts. As we apply parallelisation on distinct time steps, time-dependend extracts like pathlines are not suitable. The computation of isosurfaces for a given isovalue is a quite simple and fast algorithm. The computationally more expensive extracts are vortex regions identified by a helicity treshold method. On each grid point the normalised helicity is computed. This value is compared to a treshold to decide if this point belongs to a vortex region. This is formulated in equation 1 (where v is the velocity, h_{tresh} is the helicity treshold):

$$\frac{\nabla \times v}{|\nabla \times v|} \cdot \frac{v}{|v|} \geq h_{tresh} \tag{1}$$

While helicity may be precomputed, we compute it online as an example algorithm for high computational load, to demonstrate a worst case extraction method. Both extraction algorithms produce isosurfaces as results, which have the same order of magnitude in terms of polygon count.

The system we used is a Sun Fire E25k computer with 72 UltraSparc IV 1.05 GHz dual core processors. This amounts to a total of 144 processors, which have access to 288 GB main memory. We utilised up to 64 worker nodes for 128 time steps of the shock data set and 32 worker nodes for 50 time steps of the propfan. The visualisation computer is a workstation (3.2 GHz dual processor, 2 GB main memory, NVIDIA GeForce 6800 GT) connected via a non-dedicated 100 MBit/s network.

3.4. Evaluation Results

Two isosurfaces of energy scalar values are extracted in a time region of 128 time steps of the shock data set. A single worker node needs more than 17 minutes for this task on a total of 256 million grid points, which is only 14 % of the whole data set. We choose only a time region of the complete data set, otherwise computation would take too much time for an explorative analysis. As the propfan data set consists of only 50 time steps, we compute the whole data set. The results for both data sets and extraction methods are depicted in figure 4. Vortex computation takes approximately 10-15 times longer than isosurface extraction. Using 32 worker nodes, computation times for unsteady isosurfaces are reduced to 39 s (shock) and 23 s (propfan). Vortex computation takes still too long with 386 s (shock) and 329 s (propfan).

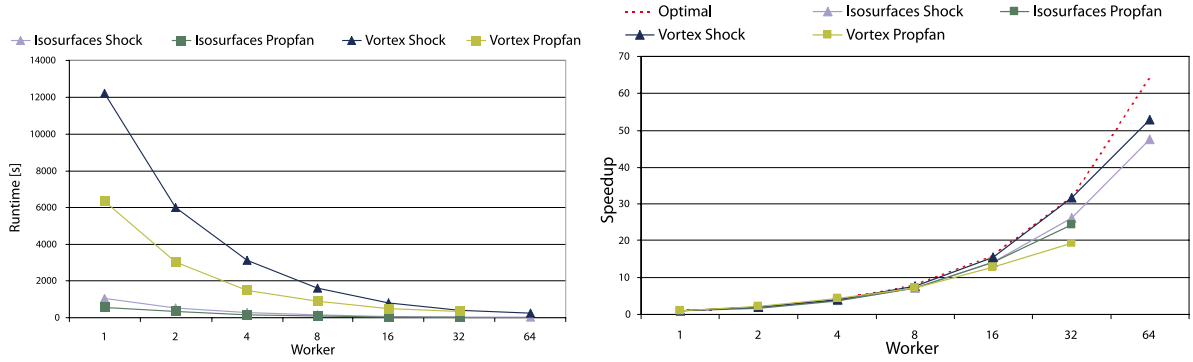

Figure 4: *Measurement results for parallel computation of isosurfaces and vortices on the shock and propfan data sets. Left: Overall runtimes. Right: Speedup values. The optimal speedup is dashed red.*

Speedup values are near to the optimal speedup for up to 16 worker nodes (approximately 90 % efficiency). For a larger number of processes efficiency decreases as only a few time steps are processed per worker node. This results in imbalanced computations, as the applied extraction algorithms are not balanced.

The user's total waiting time is reduced with high efficiency, but the underlying extraction algorithm and the data set's size are too complex to provide fast results. The question of the execution order of single time step computations remains. The naive approach to compute a time region from the lowest to the highest time step does not consider the user's exploration process. Additional waiting time is caused if the time span currently under investigation is computed last. Therefore we now propose methods to adapt the overall computation time to the user's investigation behaviour.

4. Interactive Exploration

Concerning direct manipulation, the separated visualisation application continuously renders already computed extracts in an animation loop. For most extraction results, a frequency of more than 10 frames per second is achieved, which accomplishes the formerly described interactivity criteria. Whether one is able to achieve a fast result response (in less than 500 ms [BJ96]) for unsteady extraction of flow phenomena depends on the optimisation of the algorithm, the size of the unsteady data set and the performance of available hardware. Since we cannot guarantee interactive response times, all the more important are interactivity issues supporting the exploration process. If results are not delivered fast enough, the user should at least be able to influence the longer computation time interactively. Especially in an explorative analysis, where the user frequently changes parameters to investigate the data set, a computation has to adapt to the user's behaviour. This includes obvious features like the ability to cancel a running computation if the presented results are unsatisfying or the ability to change parameters of a running extraction. The introduced framework for parallelising unsteady computations supports both abilities.

To further support the user's exploration process, we propose a concept of interactively influencing the order of computation by assigning priorities to time steps of unsteady feature extractions. Time steps with higher priority are computed first, that is their results are sooner available to the user. While this does not fulfil a direct response time of less than 500 ms independent of the extraction algorithm, occuring computation times are arranged to overlap with the user's exploration process.

4.1. Time Step Priorities

This section explains the general setup of the time step priority system and its implementation. As the user may update computation parameters of a running feature extraction, this infrastructure is used to implicitly transmit visualisation and animation parameters to the workhost. Whenever the user changes animation speed, pauses, continues the animation or starts an extraction, current animation parameters are transmitted. The taskcommand, which assigns time steps to workers dynamically, holds a priority queue which contains all time steps to compute. Whenever a worker requests a new time step, the queue's element with highest priority is assigned. Using this infrastructural concept, the remaining task is to define the priority function which assigns a priority value to each time step. Based on previously collected runtime information about different tasks, heuristic methods are used to assign priorities to time steps independent of the currently executed kind of extraction algorithm. Therefore, these heuristics support exploration of unsteady data sets as a general method.

These methods exploit the fact that by using Viracocha, time steps are processed in parallel. While the time for a single time step computation stays the same, in the same time

approximately n time steps are processed, if n workers compute the whole task.

4.2. Time Scales

In the visualisation of simulation data, several time scales occur. We distinguish between simulation time, animation time and computation time, which have all different durations for one time step.

Simulation time t^{sim} is the time scale of the simulated data, e.g. months or weeks for meteorological data. Each time step of a data set describes an instant of this simulation.

Animation time $t^{animation}$ is the time scale in which time steps are displayed. The animation time gives each distinct time step a concrete display length and may be arbitrarily adapted by the user. Typically one animation loop of all time steps spans several seconds up to a couple of minutes.

Computation time t^{comp} is the time scale of task computation runtime. The runtime for each time step is determined by its size and its complexity regarding the applied extraction algorithm.

To adapt to the user's temporal behaviour, the taskcommand needs to know the current animation time, animation loop time and the mapping from animation time to time steps. This data is transmitted with the computation parameters. From there on the taskcommand keeps a clock which is synchronised with the animation loop, evading communication overhead for regular synchronisation. The clock is resynchronised on every user input concerning the computation.

While a data set may consist of several hundreds or thousands of time steps, the user is mostly interested in a subset of all available time steps. In the following, we will refer to this subset as *time region*. This may be a fixed interval $[t_1, t_2]$ as well as a subsampling of the original time steps, e.g. every tenth time step. To counteract imbalanced computation for our heuristics, we assume t_{ts}^{comp} to be the worst case runtime for a time step computation. Accordingly, $t_{ts}^{animation}$ is the constant time span one time step is displayed in the animation.

4.3. Prioritisation Methods

Based on the conceptual framework, several heuristics for reordering tasks according to different assumptions on user behaviour are implemented. As the reordering process is done on the algorithmic layer, the implementations can be easily expanded or variated. We present three different methods we consider useful for general data sets. These methods cover three typical user actions: the user may be interested in a coarse overview, in a detailed time step or in the dynamics of a discovered feature.

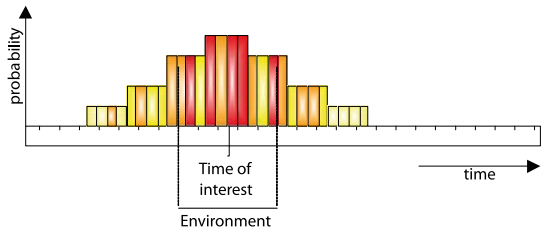

Figure 5: *Scheme of gaussian distributed priority for a time region. The time of interest is the mean value of this distribution, the environment is its root mean square deviation. High priorities are depicted red, lower priorities yellow.*

4.3.1. Time of Interest

Region of interest is a common concept for spatial relationship. This implementation helps to quickly discover dynamic features in a temporal region. The user interactively specifies a time step t he is interested in, as well as a coarse environment σ where the dynamic feature is expected. This information is used as input for a normal distribution $N(t,\sigma)$, which randomly distributes the priority on the time region, as depicted in figure 5. Time steps near to the mean value t will be computed prioritised, while time steps beyond the deviation σ are assigned lower priorities. Due to the random priorities, this results in disjoint time steps in the animation loop. Nonetheless, this method may give a good impression of the feature's dynamics even without a complete computation of the time region. Waiting time is shortened if the user quickly discards the computation, otherwise the complete time region is computed.

4.3.2. Pausing

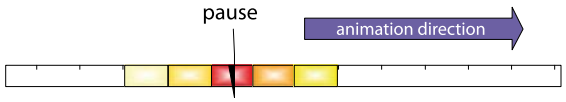

Figure 6: *Priority of time steps when a user pauses the animation. The paused time step obtains highest priority, neighbouring time steps are assigned lower priorities. Time steps in line with the animation direction receive higher priorities.*

The user may stop the visualisation animation every time. We assume the user stops at a given time step t to analyse a single timelevel of a transient feature in more detail. If the user is also interested in the dynamics of that feature, he will examine the temporal environment around t, stepping forward or backward in time. Whenever the animation is stopped, the taskcontroller is informed. The time step at this animation time is assigned the highest priority. All other time steps are marked with a priority according to their distance to t (see figure 6). Time step neighbours in the same

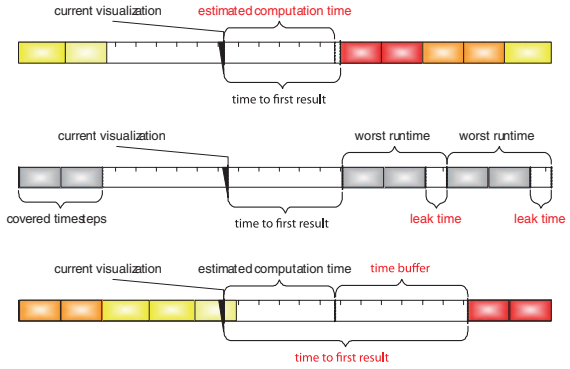

Figure 7: *Occuring scenarios of the continuous visualisation heuristics (top to bottom): the first estimation, leak times and time buffering.*

direction as the animation get a higher priority than neighbours in the opposite direction. This makes sure that more time steps are available in the direction of the animation, as the animation may be started again. In short, the time a user examines a certain time step is used to compute the temporal environment around that time step. When the user decides to step forward or backward in time, the appropriate data can be available depending on the time he spent analysing the paused time step.

4.3.3. Continuous Visualisation

The most common display mode is the continuous visualisation of the animation loop. The complete simulation or a time region are displayed in an endless loop to investigate feature dynamics. The goal of this heuristic algorithm is to provide a continuous visualisation of requested extracts after an initial waiting time. First, the shortest waiting time is estimated. Then the algorithm tries to maintain a continuous visualisation. Continuous visualisation helps to understand the temporal evolution of a feature. Gaps in the visualisation should not occur, as they disturb the correlation of dynamic features. The connection between successive time steps which form a dynamic feature can easily be lost.

We define $t_{first}^{animation}$ to be the time between the user's input and the availability of the first result. $t_{first}^{animation}$ may be enhanced by optimisation of the single time step computation. As the user moves forward in animation time while waiting for results, the algorithm estimates the time step the user reaches in the animation loop after the worst case runtime. This estimation may not lie within the requested time region. Therefore, we choose the time step inside the time region that lies closest to the estimated time for the first result. Starting with this time step, priorities are assigned with decreasing values (see figure 7 top).

As we use the worst case runtime of all n workers, after every computation time of t_{ts}^{comp} we cover an animation time of $n \cdot t_{ts}^{animation}$. After the inital gap of $t_{first}^{animation}$, a continuous visualisation is therefore guaranteed if

$$t_{ts}^{comp} < n \cdot t_{ts}^{animation} \tag{2}$$

As we cannot make guarantees for an unknown extraction algorithm, the system has three possibilities to fulfil equation 2. First, the number of workers n for this task may be increased. Second, if not enough free resources are available, the visualisation application may reduce the animation speed. This may not be desired by the user, as it changes the perception of the flow dynamics. The third possibility is to delay the first result until enough computational lead is collected to cover all gaps in the continuous visualisation. The naive approach is to delay a whole animation loop, which gives the most computational lead, but extends the waiting time to its worst case. Therefore, we compute the required lead as follows.

The time span of the gap between computational time and animation time is called leak time (see figure 7 center). If $n \cdot t_{ts}^{anim} < t_{ts}^{comp}$, the worst leak time is the time span between the worst computation runtime and the amount of thereby covered animation time:

$$t_{leak}^{comp} = t_{worst}^{comp} - n \cdot t_{ts}^{animation} \tag{3}$$

Similar to video streaming approaches, a time buffer is introduced to counteract the effect of leak times. To cover t_{leak}^{comp}, the first result is moved the time buffer ahead. This additional waiting time allows the workhost to compute enough results to provide a continuous animation (see figure 7 bottom). The time buffer covers the leak time a number of times equal to the worker's iterations k needed to compute the whole command:

$$t_{buffer} = t_{leak}^{comp} \cdot k \tag{4}$$

While this increases the initial waiting time to $t_{first}^{animation}$ + t_{buffer}, the user gains continuous visualisation even for large data sets. Problems occur if the time buffer is greater than the time for one animation loop. That is, first results may be finished prior to the worst runtime and are therefore displayed one animation loop too early. In these cases either the animation time must be slowed down or the workhost keeps results until the valid animation loop starts.

We applied the continuous visualisation algorithm on the scenarios introduced in section 3.3. The overall computation time for vortex extraction exceeds the animation loop time considerably, which results in $t_{ts}^{animation}$ of several seconds. For continuous visualisation of vortex regions we suggest a large number of processes or a smaller time region. Table 1 shows required time buffer lengths for computing isosurfaces with 16 processors for different animation times per time step. With only 16 processes, a continuous visualisation

Table 1: *Continuous visualisation of isosurfaces with 16 workers.*

data set	$t_{ts}^{animation}$ [s]	$t_{loop}^{animation}$ [s]	t_{buffer} [s]
shock	1	128	0
	0.75	96	3
	0.5	64	38
propfan	1	50	0
	0.5	25	11
	0.33	16.7	18

is possible without time buffer if one time step is displayed per second. For small display times and many time steps, the time buffer grows too large. Within these restrictions the user is able to balance between waiting time and animation speed.

5. Conclusions and Future Work

To support explorative analysis in virtual environments even with large unsteady data sets, we utilised the Viracocha framework for parallelising transient feature extraction, resulting in near optimal speedup. But even when computing simple extracts, interactive response times are not achieved. So we focus on the interaction between the user's behaviour regarding the visualisation animation and the computation of tasks. Three different concepts for supporting a faster and more convenient explorative analysis are described, including a way to provide a continuous presentation of results.

Our future research will focus on the following topics. First, we will analyse further concepts of prioritising time steps, e.g. a strided approach for the generation of a fast overview. Second, we will develop more sophisticated methods of runtime estimations in order to improve the heuristics outlined in section 4.3.3. Third, we will evaluate the presented methods by means of user studies.

Acknowlegement

The authors would like to thank the Institute of Aerodynamics at Aachen University, the German Aerospace Centre (DLR), Institute of Propulsion Technology at Cologne and the FEV Engineering Services for the simulation data sets kindly made available.

References

[AGL*04] ALLARD J., GOURANTON V., LECOINTRE L., LIMET S., MELIN E., RAFFIN B., ROBERT S.: FlowVR: a Middleware for Large Scale Virtual Reality Applications. In *Proceedings of Euro-Par* (2004), pp. 497–505.

[BGY92] BRYSON S., GERALD-YAMASAKI M. J.: The Distributed Virtual Windtunnel. In *Proceedings of the IEEE Supercomputing* (1992), pp. 275–284.

[BJ96] BRYSON S., JOHAN S.: Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments. In *Proceedings of the IEEE Visualization* (San Francisco, CA, 1996), pp. 255–261.

[GHW*04] GERNDT A., HENTSCHEL B., WOLTER M., KUHLEN T., BISCHOF C.: VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *Proceedings of the IEEE Supercomputing* (November 2004).

[GLS99] GROPP W., LUSK E., SKJELLUM A.: *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd ed. MIT Press, 1999.

[MC00] MA K.-L., CAMP D. M.: High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network Status. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)* (2000).

[RFL*98] RANTZAU D., FRANK K., LANG U., RAINER D., WÖSSNER U.: COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data. In *2nd Workshop on Immersive Projection Technology (IPT '98)* (Ames, Iowa, May 1998).

[SGvR*03] SCHIRSKI M., GERNDT A., VAN REIMERSDAHL T., KUHLEN T., ADOMEIT P., LANG O., PISCHINGER S., BISCHOF C.: ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments. In *Proceedings of the 7th International Immersive Projection Technology Workshop and 9th Eurographics Workshop on Virtual Environment* (May 2003), pp. 77–85.

[vDFL*00] VAN DAM A., FORSBERG A. S., LAIDLAW D. H., LAVIOLA J. J., SIMPSON R. M.: Immersive VR for Scientific Visualization: A Progress Report. *IEEE Computer Graphics and Applications 20*, 6 (2000).

[vRKG*00] VAN REIMERSDAHL T., KUHLEN T., GERNDT A., HENRICHS J., BISCHOF C.: ViSTA: A multimodal, platform-independent VR-toolkit based on VTK, WTK and MPI. In *Fourth International Immersive Projection Technology Workshop (IPT 2000)* (Ames, Iowa, 2000).