Eurographics Symposium on Parallel Graphics and Visualization (2007)
Jean M. Favre, Luis Paulo dos Santos, and Dirk Reiners (Editors)

Dynamic Regions of Interest for Interactive Flow Exploration

M. Wolter[1], C. Bischof[2], T. Kuhlen[1]

[1]Virtual Reality Group, RWTH Aachen University
[2]Institute for Scientific Computing, RWTH Aachen University

Abstract

Virtual Reality (VR) provides a useful tool for understanding complex, unsteady flow phenomena. The user can directly interact with the data and therefore benefits from a spatial coherence of action and result. However, visualization in virtual environments imposes very high demands on interactivity in order to maintain this coherence. Exploration of large, unsteady datasets in VR requires efficient visualization or data reduction algorithms to produce results within acceptable waiting times. We propose a technique for reducing required data especially suited for direct interaction in virtual environments. We use a distributed system to parallely extract a dynamic region of interest (DROI) from the simulation data. This DROI is adapted according to the user's interaction behavior and allows for the analysis of local flow features. With this reduction we provide interactive extraction of local features from large, time-varying datasets.

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

1. Introduction

In the explorative analysis of complex, unsteady simulation data, Virtual Reality (VR) has shown to be a useful instrument. Time-varying 3D structures are perceived in a natural way, which delivers a deeper insight into complex flow phenomena. Moreover, direct and natural manipulation of objects is a central benefit gained from virtual environments. The common example is a data probe or a particle seeder moved by a hand-attached manipulation interface through 3D space. This possibility of interaction creates a spatial coherence of the user's action and the perceived results.

However, this coherence is easily destroyed, as Virtual Reality imposes high demands on interactivity. High latency or a low graphical update rate immediately have a significant negative effect on the user's acceptance of a virtual environment, worse than in a desktop environment. To maintain this coherence, Bryson [Bry04] suggested the following timing restrictions:

- The graphical update rate must be greater than 10 frames per second.
- Interaction responsiveness (of direct interaction) must be less than 100 milliseconds.
- Data updates should be available in less than one third of a second.

The enormous increase in computing power available today has resulted in ever increasing dataset sizes generated by computer simulations. For large datasets, these interactivity requirements are not easily met. Especially the visualization of time-varying data requires the periodical reading of large files from slow secondary storage. Additionally, a time-varying data domain changes continuously, immediately invalidating previously computed results at each data time change. Even for very simple visualization algorithms like a point probe, the memory bottleneck prevents fast data updates. A lot of research has been done in the field of efficient data access and out-of-core methods, but most solutions are optimized for specific visualization methods.

One approach to accelerate the computation is the usage of parallel computers and distributed systems. By parallelization of the feature extraction and the rendering process, a high performance gain can be achieved. However, distribution means communication, and communication results in additional latencies, which have to be compensated.

We introduce a distribution concept which takes advantage of the knowledge about the user's exploration behavior. This concept aims at interactive visualization of large, time-varying data where parameter changes originate from a direct interaction interface inside the virtual environment. We use the fact that, when using a direct manipulation metaphor for extracting features from time-varying data, this manipulation primarily concerns spatial parameters. Temporal parameters change continuously as the user watches the animation of successive time steps. Thus, it is easier to predict which time step is going to be visualized next than to predict the exact movement of the user's interaction device.

We exploit this difference in requirements for a specially suited distribution method: instead of computing a feature directly from the complete dataset, we confine the required data to a region around the user's interaction device. All spatial parameter changes are computed locally on this limited region. As the region is connected with the user's direct interaction behavior, we define it as a region of interest (ROI).

To provide the required data for temporal parameter changes, a high performance computer loads the dataset and continuously extracts a spatially connected ROI from each time step. These regions are transmitted to the visualization system just in time before the corresponding time step is displayed. Whenever the data time changes, the analyzed regions become invalid and are immediately removed from memory. We define a set of ROIs for consecutive time steps as a dynamic region of interest (DROI).

Dynamic regions of interest make it possible to analyze time-varying datasets with a large set of visualization methods while satisfying interactivity criteria required for immersive virtual environments. Each DROI contains only a small part of the spatial and temporal extent of the whole dataset. Since they are computed on a parallel HPC machine, a large number of regions can be extracted concurrently. With the usage of spatially reduced data and the avoidance of slow secondary storage or network bottlenecks, interactive update rates can be maintained at the visualization system.

The rest of this paper is structured as follows: section 2 briefly summarizes related work regarding interactive visualization especially in virtual environments. Section 3 describes the setup which motivated our work and which we used to evaluate our method. The method itself is explained for time-invariant data in section 4, and expanded to the time-varying case in section 5. We measured different update rates using several datasets and report the achieved results in section 6. Section 7 resumes benefits and drawbacks and gives an outlook for future work.

2. Related Work

One of the first available systems for VR-based flow visualization was the Virtual Wind Tunnel and its follow-up, the Distributed Virtual Wind Tunnel [BGY92]. The latter introduced a connection to a vectorized post-processing back-end, which then was responsible for post-processing computations. In the Virtual Windtunnel, any scalar or vector field can be attached to a data probe to obtain quantitative data at any point. Recently, Allard et al. introduced FlowVR [AGL*04], a middleware which can be used to flexibly connect various modules to form a distributed VR-application. Data parallel execution of modules can be obtained by distributing several identical modules to different nodes in a cluster. Another distributed software environment is COVISE [RFL*98], which focuses on cooperative work. In addition, it integrates a module for Virtual Reality called COVER. Modules containing processing steps like I/O, filtering or rendering can be distributed across different workstations.

All of these systems provide VR capability and use distributed approaches to achieve the required flexibility and performance, but they do not consider the interacting user for the distributed visualization process.

Parallel data streaming for large datasets using the Visualization ToolKit (VTK) was presented by Ahrens et. al. [ABM*01]. Their approach achieves a nearly linear speedup for data parallelization with a large number of processors. They also divide data into regions to process data piecewise. This data streaming aims at reducing the memory footprint, therefore acting as an out-of-core strategy.

Regions of interest are a well-known technique in computer graphics. One of the first applications in 3D was the MagicSphere [CMS94] used for multiresolution rendering. One of the latest applications of this class of techniques for volume rendering was presented by Krüger [KSW06].

This work is methodically based on [WHS*06], where priority scheduling for parallelization of time-varying data was proposed. Priorities were assigned to time steps according to their relevance for the user's exploration. A weak point of this method is that it is not really interactive, as parameter changes only have effect after some time. In this work we therefore introduce a system suitable for interactive parameter changes.

3. Distributed System

Several strategies for parallel and distributed computation and rendering of large datasets are possible. To classify our approach, we will briefly describe the framework we apply for exploration in virtual environments.

We use parallel machines to compute extraction algorithms on large datasets. This has the advantage that algorithms are "close to the data". Several different distribution strategies for the visualization process exist. Our approach uses remote data handling and extraction in combination with local rendering. The typical workflow is as follows: the user in a virtual environment issues a visualization command, which is handled by a parallel machine and the results are transmitted to a visualization system, where they

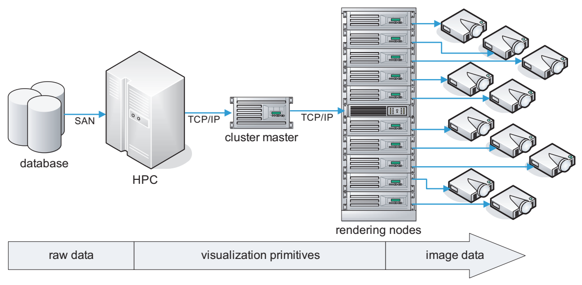


Figure 1: *A distributed hardware architecture for flow field exploration in virtual environments. Each request from a user is executed by the high performance computer (HPC) and results are transmitted to the cluster of rendering nodes, which render the images for several displays.*

are rendered. One advantage of this method when used with VR is that the effect of head-tracking is computed locally at the visualization system. The disadvantages are that geometrical primitives of arbitrary size must be transmitted, the local computer must be capable of rendering the extracted data, and each user command requires a round-trip over the network to produce extracts. The complete system setup is depicted in figure 1.

The raw simulation data resides on a storage system attached to a high performance computer (HPC). Flow features are extracted using the Viracocha software [GHW*04], which builds on VTK [SML06]. Visualization primitives are sent to the front-end node (cluster master) of a PC cluster. We use a cluster made up of several off-the-shelf personal computers with commodity graphics cards to drive a five-sided CAVE-like display system. With this setup, no special-purpose hardware is needed to operate this room-mounted VR display.

The received primitives have to be distributed to each rendering node using a dedicated network. All nodes run a mirrored application with synchronized application states. They only differ in their viewport into the virtual world, which changes with the user's head movement. The user's head and interaction devices are tracked with an optical tracking system which provides a high accuracy at 60 Hz update rate (which is therefore the upper limit for input device updates).

4. Time-Invariant Regions of Interest

First, we will describe how to extract a region of interest for interactive exploration from a single time step. This is the case when the user stops the animation and starts investigating a specific region of the data domain. The user utilizes a tracked input device for direct manipulation of some point of interest. We will refer to the position of this point of interest briefly as the input position p_{in}.

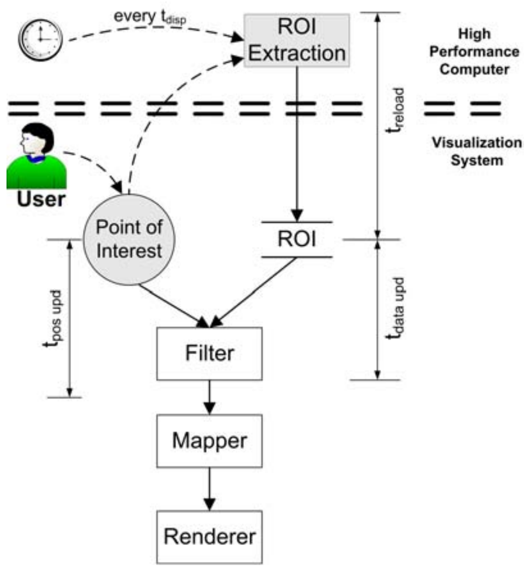


Figure 2: *The visualization pipeline based on interactive ROI extraction. Each reload including transmission takes t_{reload}. Two types of updates occur: a position update by user movement taking t_{posupd} and a data update of the current ROI taking $t_{dataupd}$.*

As the user moves the input device through the dataset, the point of interest moves accordingly. Associated with the input position is a visualization pipeline (see figure 2) specifying the feature the user is interested in. Instead of computing this feature on the complete dataset, we create a region of interest (ROI) around the input position. In this context, we define a region of interest as a spatial region of user-defined size corresponding to a selected subset of the original dataset.

The point of interest is an input parameter of the visualization filter and also affects the ROI, which is the data the filter works on. To be useful as a data source, the ROI must keep the following condition:

> At every time, the data needed by the filter component for an arbitrary position p_{in} is provided by the region of interest $ROI(p_{in})$.

This points out that the extraction of global features or integrated features which could cover the global data environment may not be combined with this method, as data is only available inside $ROI(p_{in})$. Hence, the minimal bound for the ROI is given by the filter's input request. The larger the feature extracted by the pipeline in the data domain, the larger the user must define the size of the ROI.

As the user may freely move the point of interest through the data domain, the ROI must be computed such that this condition is fulfilled. In a time-invariant exploration, the ROI

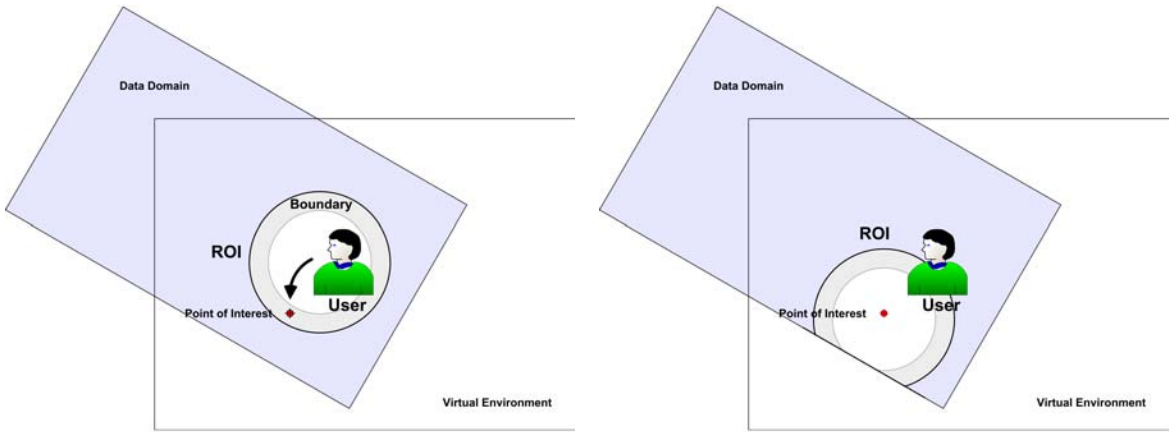


Figure 3: *Scheme of the update mechanism for time-invariant data. ROI are depicted as spheres, but various shapes are possible.* Left*: The input position moves into the boundary region due to the user's direct manipulation.* Right*: The recomputed ROI has its center near the new point of interest.*

may only be left by a spatial movement, not by a data time change (see figure 3). Therefore, the system has to guarantee that a new ROI is extracted and transmitted before the user leaves the current ROI.

The reload time t_{reload} of a ROI depends on the size and topology of a single time step, the algorithm used to extract the ROI, and the performance of the extracting parallel system as well as the available network bandwidth. As these factors may differ, we propose two methods for extracting the ROI from a single time step.

1. The most accurate method is to extract a subgrid, that is cutting a part out of the original grid. Depending on the type of the original grid, this may produce a large subgrid with a varying number of cells. In regular grids, this method produces accurate results with a predictable number of cells. In unstructured grids, interesting regions most likely to be analyzed by the user tend to have a high spatial resolution, resulting in a complex ROI.
2. Subsampling of the specified region produces equal sized regular grids every time, but introduces interpolation errors. Depending on the visualization method applied interactively to the ROI, this typically speeds up the computation and rendering time at the visualization system. For some visualization techniques a subsampled regular grid is favorable, e.g., for volume rendering in the region of interest.

Assuming a maximal velocity v_{max} of the tracked input device and an estimated time t_{reload} for recomputation and transmission, a new region is requested if the extracted feature (in the data domain, not its geometric size) enters a boundary of width r_{bound} from the border of the current ROI:

$$\frac{r_{bound}}{v_{max}} > t_{reload} \Leftrightarrow r_{bound} > v_{max} \cdot t_{reload} \qquad (1)$$

The distance r_{bound} is the maximum distance the user can move in the time span t_{reload}. The assumption of a maximum velocity is adequate, as a user exploring a dataset will lose the context of his analysis if he moves too fast. The control over the structure and the size of the ROI is left to the user. There is a lower limit for the size of a useful region, as the region must contain the size of the extracted feature and the boundary. If t_{reload} is long, the boundary must be chosen respectively large. If the boundary fills out the complete ROI, each user movement results in a reload.

At the visualization system, the visualization pipeline (figure 2) is now updated in two cases. First, the point of interest changes due to user interaction. Second, the ROI changes due to a reload, which occurs frequently because of data time changes (see section 5). Let t_{upd} be the computation time for each update of the applied pipeline. This time may depend on the type of update, therefore we have to distinguish t_{posupd} and $t_{dataupd}$. As the point of interest updates may occur with a high frequency (up to the update rate of the input device), t_{posupd} should be smaller than $t_{dataupd}$.

As an example, exchange of the underlying dataset can result in the construction of auxiliary data structures (e.g., a k-d-tree for point location), to speed up later requests. To restrict the interactivity criteria only to the position updates without additional initialization cost, the initialization of the pipeline with new data is done before the ROI becomes valid. This changes the estimation time for the distribution algorithm to $t_{reload} + t_{dataupd}$.

While methods for time-critical computations exist which cancel running computations or adapt the input data size in order to provide interactive response times, we focus on the concept of providing the user with useful data parameterized by the user himself. Therefore, the user may choose if he

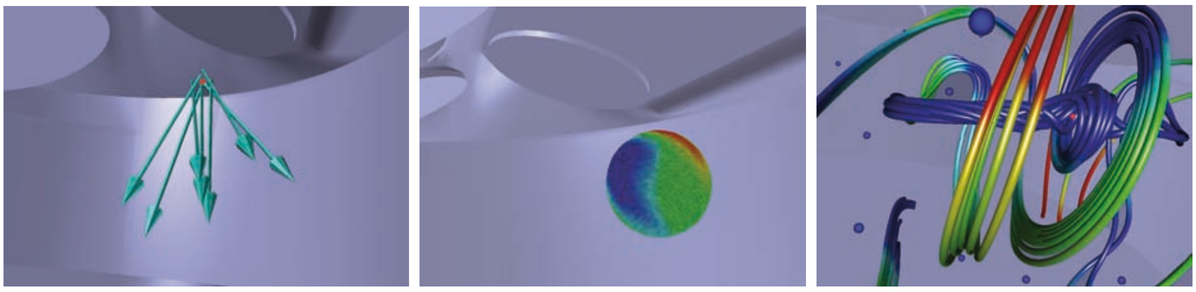


Figure 4: *Three different local visualization methods in the vicinity of the point of interest (see also color insert).* Left*: Velocity vectors depicted as glyphs around the point of interest.* Middle*: A cutsphere centered at the point of interest using Image Space Advection.* Right*: The point of interest is a seeder for streamlines, which are rendered as Virtual Tubelets (depicted together with critical points, which are not part of the ROI).*

wants to use subsampled or exact data, and how large the region of interest is.

As mentioned in section 3, for some immersive virtual environments the display system itself consists of multiple nodes. All rendering nodes are synchronized and provided with data by a single master node, which is the rendering cluster's front-end receiving the ROIs from the high performance computer. The master distributes all ROIs to the rendering nodes, which execute the pipeline locally. Even though this approach wastes resources by concurrently executing the same task, it induces less latency for highly frequent position updates. Therefore, any additional cost introduced by the display cluster is contained in the reload time t_{reload} and not in the more frequently occuring update time of the visualization pipeline.

4.1. Local Features Inside the ROI

Different feature extraction algorithms may be applied, as long as the computation and rendering of these features do not violate the interactivity criteria. Maintaining an interactive visualization is the user's responsibility, as he may freely define visualization pipelines executed on the region of interest.

We propose three visualization pipelines of different size and complexity, which we consider useful to analyze local flow phenomena (see figure 4).

- A simple point probe visualized using arrow glyphs. It requires the least size, as only a number of points have to be evaluated in the data domain (see figure 4, left).
- An implicit function (e.g., a sphere) cutting the ROI. The resulting cut object is colored by a scalar value and overlayed with an Image Space Advection texture [LJH03] displaying information about the vectorfield on the objects surface (see figure 4, middle).
- The point of interest serves as particle seeder for streamline computation. Here, the region is not restricted by the seeder, but the resulting streamlines are restricted by the region. Therefore, we propose to display additional information to the user to determine if streams were aborted because they left the ROI. To provide a better depth impression in the virtual environment, the streamlines are rendered as tubes using Virtual Tubelets [SKH*05] (see figure 4, right).

It should be noted that particularly streamlines are only meaningful for steady data. We regard the computation of pathlines as future work, since ROIs from at least two time steps must be available for a local computation.

4.2. Prediction

If t_{reload} spans several seconds, the user may have moved a considerable distance towards the current ROI's boundary. To avoid frequent reloads, the recomputed ROI should be centered at the new point of interest by the time it replaces the current ROI. As this position is not known at the time the new ROI is requested, the new position has to be estimated.

We decided to use a simple dead reckoning algorithm to extrapolate the user's movement. We apply a first order predictor using the following equation:

$$p_{new} = p_{current} + t_{reload} \cdot v \qquad (2)$$

where v is the input device's velocity. While there are better algorithms predicting user movements in virtual environments (e.g., Kalman filters), we consider the simple predictor satisfactory, as a high accuracy is not fundamental. As the extrapolated position is only the center point of a larger region, the prediction error is negligible as long as it is small compared to the size of the ROI. In the case of long reload times, more accurate predictors should be employed, as these simple predictors are too inaccurate for long prediction times. This issue is a topic for future studies.

5. Dynamic Regions of Interest

Expanding the time-invariant method to a time-varying analysis raises new problems. We refer to an animated visualization setup, where data time continuously changes and all time steps are running in an endless loop. The user may influence the display time t_{disp} each time step is shown, start and stop the animation at any time, or move to selected time steps. Data results which are not available at the time the corresponding time step is displayed lead to a gap of the continuous visualization.

As we focus on methods for a continuous animation of time steps, this implies that new ROIs for single time steps must be available with the change rate of data time. The visualization system holds a buffer of size n for ROIs. As the entries in this buffer correspond to consecutive time steps, we call this set of ROIs a dynamic region of interest (DROI). This expands the condition in section 4 to:

> At every time, the data needed by the filter component for an arbitrary input position p_{in} at data time t is provided by the dynamic region of interest $DROI(p_{in}, t)$.

In order to satisfy this condition, the system must provide an updated DROI whenever the point of interest leaves the spatial or temporal extents of the current DROI (see figure 5).

First, we will consider only the temporal condition. The user leaves the current ROI after every display time span t_{disp} as the region for this time step becomes invalid. This is a more frequent event than a reload due to a position update, but can be predicted very well. To deliver the ROIs just in time, we use the system for continuous visualization explained in more detail in [WHS*06]. In short, the discrete time steps of the dataset are distributed by a dynamic load-balancing algorithm to processes running on a parallel machine. The distribution algorithm optimizes the assignment not for best speed-up, but for a just-in-time arrival of computed results with respect to the continuous visualization. For a more detailed description of the applied parallel techniques, especially the parallel architecture and scalability, please refer to [WHS*06].

The number of ROI extraction processes n used of the parallel machine is chosen according to the following equation:

$$n \cdot t_{disp} > t_{reload} \Leftrightarrow n > \frac{t_{reload}}{t_{disp}} \quad (3)$$

After an initial waiting time, a continuous transmission of ROIs is provided. After each reload time t_{reload}, a time span of $n \cdot t_{disp}$ is covered, that is data for n time steps is available for local computation. If a gap occurs in this stream due to an inaccurate estimation of t_{reload} (as t_{disp} stays constant), the point of interest will be invalid for a time span of t_{disp}. To counteract these cases, we use the worst computation time as the estimation value.

Second, the DROI must fulfill the spatial condition. The position predictor must make an accurate estimation for a prediction time up to $n \cdot t_{disp}$, which is the time covered by

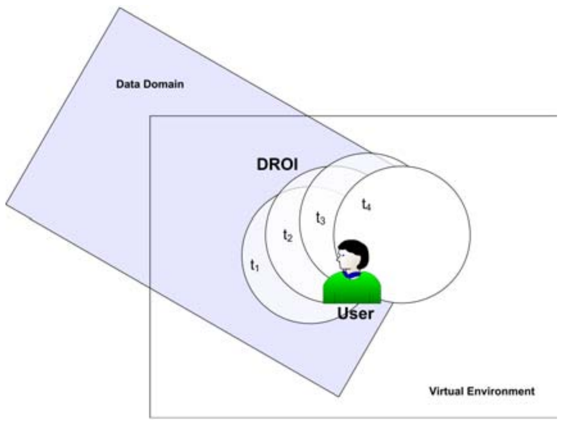


Figure 5: *With changing data time (t_1 to t_4) and input position, a changing DROI is computed. The center positions for the time-dependent ROIs are estimated.*

one iteration of the parallel computation taking t_{reload} in total. Each process computes a single time step i out of n and predicts the position of the point of interest after $i \cdot t_{disp}$.

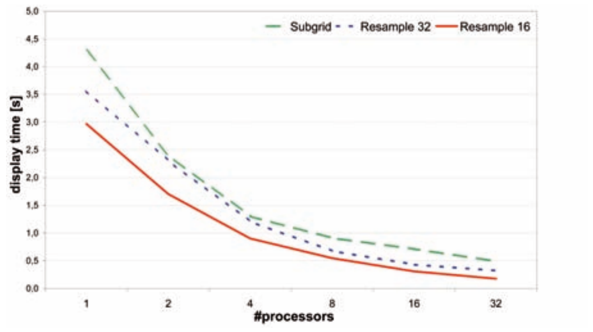
Even with a currently stopped animation in an analysis of a single time step as described in section 4, a DROI of n consecutive ROIs is computed. While this again wastes resources as it is unnecessary for exploring the time-invariant data, the user may at any time start the animation again. Without a buffer of n ROIs, he would have to wait for t_{reload} until a large enough DROI is computed.

6. Results

We measured update times and display times achieved with our implementation of DROIs using two different datasets. The dataset designated as *shock* consists of 919 time steps of a rectilinear grid. Each time step contains 1.95 million grid points, adding up to a total file size of 70 GB. The simulation data describes an ultrasonic shock induction with the goal of investigating the vortex structures depending on the induction angle. A second dataset denoted as *propfan* is made up of 50 time steps, each with 2.5 million points in an unstructured moving grid. It simulates a propulsion turbine with moving fans. The total dataset size is 9.5 GB.

The high performance system we used is a Sun Fire E6900 computer with 24 UltraSparc IV 1.2 GHz dual core processors. Visualization nodes are equipped with a 3.2 GHz dual Xeon, 2 GB main memory, and an NVIDIA GeForce 6800 GT. The systems are connected via a non-dedicated 100 MBit/s network.

For both datasets, the size of the ROI in the data domain is approximately 20% of the axis length for all three spatial dimensions. Three different resolutions of the ROI are examined: a subgrid, a resampled grid with resultion 16^3, and



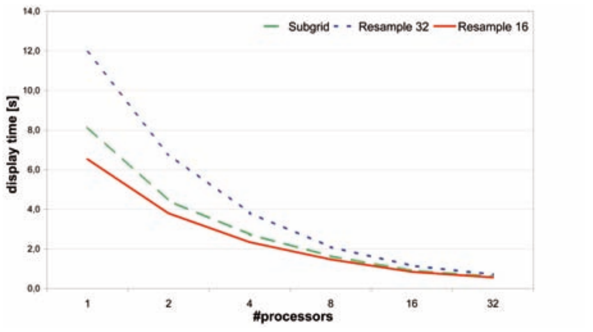


Figure 6: *Achievable display times per time step using parallel computation of the DROI.* Left: *Display times for the shock dataset.* Right: *Display times for the propfan dataset.*

a resampled grid with resolution 32^3. For these three types of ROI reload times and update times for the proposed local features are measured.

Table 1: *Reload times for time-invariant ROIs*

ROI extract	worst case of t_{reload} [s]	standard deviation of $t_{reload}[s]$	avg. number of cells
shock dataset			
subgrid	3.08	0.11	27114
resample 32^3	2.391	0.192	29791
resample 16^3	0.641	0.056	3375
propfan dataset			
subgrid	4.08	0.124	26767
resample 32^3	4.593	0.84	29791
resample 16^3	0.985	0.057	3375

The worst case reload times for steady ROIs are depicted in table 1. In the rectilinear grid, the reload time for a subgrid-ROI is considerably larger than the subsampled version, however in the unstructured grid this effect decreases, as point location takes more time. The measured reload times are relatively stable, which makes sure that most reload times fall below the worst time estimation t_{reload}. The reload times are dominated by the subgrid algorithm and the resampling algorithm, respectively. Required data is well predictable, which results in low I/O waiting times. The algorithms used are non-optimized VTK algorithms. In addition, the applied UltraSparc processors have a limited single processor serial performance. First tests with an optimized and parallelized resampling algorithm showed promising results for the reduction of reload times.

Figure 6 shows achievable display times per time step t_{disp} for the parallel extraction of the DROI. With up to 32 processors, a maximum data time change rate of approximately 3 time steps per second can be achieved with subsampled regions in the shock dataset and up to 2 time steps per second for the more complex propfan dataset. The frame rate stayed above 20 fps, thus allowing for interactive manipulation.

Table 2: *Update times for several local features*

ROI extract	glyph probe [ms]	cutobject [ms]	streamlines [ms]
shock dataset			
original data	33	1200	700
subgrid	3	16	26
resample 32^3	<1	5	16
resample 16^3	<1	1	11
propfan dataset			
original data	20	1100	220
subgrid	2	36	24
resample 32^3	<1	23	8
resample 16^3	<1	8	7

Table 2 shows update times for the three different local visualization methods. All three pipelines are constructed of VTK components (with own renderers). For the glyph probe, a simple arrow shaped glyph at one probe point is used. The implicit function of the cut object is a sphere with a radius of 20% of the ROI's size. The particle seeder injected eight particles integrated forward and backwards.

The data labeled "original" is a measurement of the update times on the original, full-size dataset, with the dataset already available in main memory. Using the restricted data domain of the ROI, all three methods update data noticeably faster than the required 333 ms, even faster than the demanded direct interaction response time of 100 ms. These results point out that more complex visualization methods can be used when working with DROIs.

On the original dataset, even when no file I/O is necessary, only simple point probing is possible with acceptable update times. That is, even if enough main memory was available on the visualization system, without online reduction of the data interactivity could not be achieved.

7. Conclusions

We presented the concept of dynamic regions of interest to enable interactive exploration for large, time-varying CFD datasets. By taking the user's direct interaction into consideration, a spatially and/or topologically reduced region of interest is computed. As this region surrounds the direct interaction point, local features can be extracted with high efficiency.

The user keeps full control over this region's parameters while the system adapts itself to changing user behavior. DROIs represent a very flexible approach, as they may be used with a large set of visualization methods and are controllable by the user during the exploration process. When the parameters for a parallel just-in-time extraction of regions are adjusted, DROIs enable a continuous exploration independent of the number of time steps.

Choosing reasonable parameters for the input device's maximum velocity, ROI size and resolution, and an appropriate prediction technique remains an open question. To obtain meaningful parameters, user studies have to be carried out. The definition of non-biased tasks for several probing techniques is a non-trivial task. We are currently discussing such a study with engineers and psychologists, but the execution and analysis remains future work.

The weak point of this method are inaccurate estimations (for reload time and position). Bottlenecks for resources (network and parallel computer) result in inaccurate worst reload time estimations. When reload times grow large due to complex single time steps this results in inaccurate position predictions, as the prediction time gets too long. In these cases, the computed DROI may not satisfy the required conditions and the user's exploration is disturbed. We suggest to provide feedback about the current state and layout of the DROI to the user on demand, so that gaps are perceived as such.

The only solution is to optimize the computation of the DROIs by applying hybrid parallelization, faster I/O and more efficient data structures. This will be the focus of our future research, as well as the examination of more accurate movement prediction algorithms.

Acknowledgement

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

References

[ABM*01] AHRENS J., BRISLAWN K., MARTIN K., GEVECI B., LAW C., PAPKA M.: Large-Scale Data Visualization using Parallel Data Streaming. *IEEE Computer Graphics and Applications 21*, 4 (2001), 34–41.

[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 '04* (2004), pp. 497–505.

[BGY92] BRYSON S., GERALD-YAMASAKI M. J.: The Distributed Virtual Windtunnel. In *Proceedings of the IEEE Supercomputing '92* (1992), pp. 275–284.

[Bry04] BRYSON S.: Direct Manipulation in Virtual Reality. In *Visualization Handbook* (2004), Hansen C., Johnson C., (Eds.), Elsevier, pp. 413–430.

[CMS94] CIGNONI P., MONTANI C., SCOPIGNO R.: MagicSphere: An Insight Tool for 3D Data Visualization. *Computer Graphics Forum 13*, 3 (1994), 317–328.

[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 '04* (2004).

[KSW06] KRÜGER J., SCHNEIDER J., WESTERMANN R.: ClearView: An Interactive Context Preserving Hotspot Visualization Technique. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 941–948.

[LJH03] LARAMEE R., JOBARD B., HAUSER H.: Image Space Based Visualization of Unsteady Flow on Surfaces. In *Proceedings of the IEEE Visualization '03* (2003), IEEE Computer Society, pp. 131–138.

[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)* (1998).

[SKH*05] SCHIRSKI M., KUHLEN T., HOPP M., ADOMEIT P., PISCHINGER S., BISCHOF C.: Virtual Tubelets - Efficiently Visualizing Large Amounts of Particle Trajectories. *Computers and Graphics 29*, 1 (2005), 17–27.

[SML06] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit User's Guide*, 5th ed. Kitware, Inc., 2006.

[WHS*06] WOLTER M., HENTSCHEL B., SCHIRSKI M., GERNDT A., KUHLEN T.: Time Step Prioritising in Parallel Feature Extraction on Unsteady Simulation Data. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '06)* (2006), Eurographics/ACM SIGGRAPH, pp. 91–98.