Eurographics Symposium on Parallel Graphics and Visualization (2009)
J. Comba, K. Debattista, and D. Weiskopf (Editors)

Hybrid Parallelization for Multi-View Visualization of Time-Dependent Simulation Data

Bernd Hentschel[1], Marc Wolter[1], Peter Renze[2], Wolfgang Schröder[2], Christian Bischof[3], and Torsten Kuhlen[1]

[1]Virtual Reality Group, RWTH Aachen University, Germany
[2]Institute of Aerodynamics, RWTH Aachen University, Germany
[3]Institute for Scientific Computing, RWTH Aachen University, Germany

Abstract

Interactive analysis using multiple linked views has been successfully applied to time-dependent simulation data. In this paper we extend previous work by embedding multiple views in a virtual environment. Here, we combine 3D scatterplots with direct interaction and natural stereoscopic viewing. In order to deal with today's simulation data effectively, we propose a hybrid parallelization scheme based on distributing the workload between a powerful compute back-end and a rendering client. It minimizes the amount of latency introduced by the distributed setup, which is vital in order to facilitate highly interactive operations such as brushing. We illustrate the effectiveness of our approach in a case study from the field of flow visualization.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.6]: Interaction Techniques—Computer Graphics [I.3.7]: Virtual Reality—Simulation and Modelling [I.6.6]: Simulation Output Analysis—

1. Introduction

Interactive data exploration has been found highly beneficial for the analysis of complex, multi-dimensional data. This is particularly true, if the phenomenon to be found is not known a priori or if the researcher wants to gain insight into a complex process which has not yet been sufficiently explained by an analytical model. According to Spence, interactive exploration helps to build a mental model of the data in these cases [Spe06]. The concept of brushing multiple linked views has been established as a method to quickly identify multi-dimensional relationships for both quantitative data and simulation data [BC87,DGH03]. However, two problems arise when using this approach for the analysis of simulation data in a desktop-based environment: First, workstations only provide limited computational resources, restricting the amount of data that can be analyzed. Second, it is often hard to understand the shape of intricate three-dimensional structures. In this paper, we extend previous work by a) distributing the workload between a parallel compute server and a visualization client and b) by embedding the exploration process into an immersive virtual environment and adding three-dimensional scatterplot views.

In order to enable an interactive exploration in this setting, two main requirements have to be fulfilled: First, we need to provide users with an efficient way to interact with their data in 3D. This is done by allowing direct interaction on each of the scatterplots as described in Section 3.1. Second, the system has to be capable of handling large simulation data sets. This is facilitated by exploiting the underlying structure of the update computations which readily lends itself to parallelization. In particular, a remote compute server is used to handle large data (cf. Section 3.2) and the data is reduced to a manageable and displayable size in the process (cf. Section 3.3). A key point is the minimization of the latency penalty resulting from the remote computation scheme, which is vital in order to enable a highly interactive operation such as brushing. This is achieved by employing parallelization on different levels and limiting the amount of data that has to be transmitted for each update.

Please note that we do not target high end, tera-scale data sets here, which cannot be handled interactively in their raw form. We rather focus on mid-sized problems, that regularly arise in daily research work but cannot be dealt with in an interactive way using standard desktop-based methods. In

DOI: 10.2312/EGPGV/EGPGV09/079-086

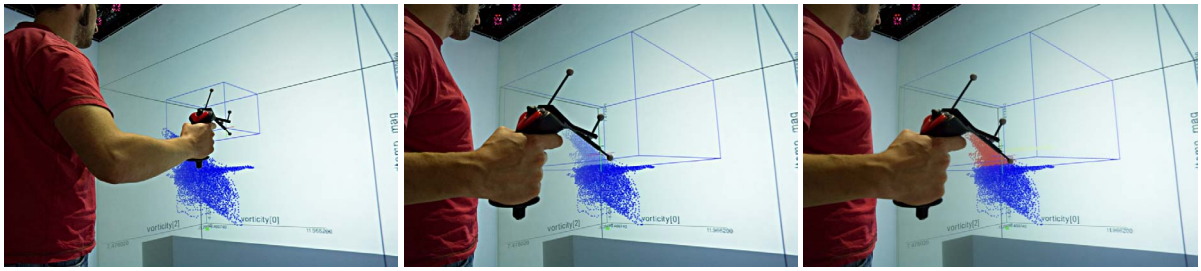

Figure 1: *A user is interactively brushing values of high streamwise vorticity in a 3D scatterplot inside a CAVE-like virtual environment. Starting with an empty selection box, he drags the the input device to create a new selection (left). Points influenced by this selection are locally highlighted in real-time (center). After the selection has been fixed by releasing the device's button, the update is remotely computed and selected points are depicted in red for all linked views (right).*

Section 4, we present a set of runtime measurements and a case study for such a data set. The study has been carried out in collaboration with fluid mechanics engineers. It therefore reflects the findings of domain experts on real-world data.

In summary, we present the following contributions: First, we describe a scalable visualization approach which combines multiple linked views and a VR-based user interface. Second, we describe a distribution and parallelization scheme which facilitates interactive brushing updates. Third, we validate the overall method in a case study.

2. Related Work

Parallel computation has been widely used in visualization, e.g., by Ahrens et al. in order to handle large simulation data [ABM+01]. Bryson et al. employed a distributed, parallel approach in their Windtunnel system in order to speed-up costly visualization computations [BGY92]. Later Schirski et al. presented the ViSTA FlowLib system, which follows a similar approach but specifically aims at the VR-based analysis of *time-dependent* flows [SGvR+03]. More recently, Schirski et al. discuss the problem of resource allocation in distributed/parallel visualization scenarios [SBK07]. They particularly mention the latency penalty, which results from offloading visualization computations to a remote system. While this penalty is tolerable for a wide range of standard visualization techniques, direct user feedback is mandatory for highly interactive operations such as the remote brushing updates described in this paper. Moreover, the usability of a VR-based visualization system strongly depends on well-designed interaction techniques, as pointed out by De Haan et al. and Kreylos et al. [dHKP02, KBB+06].

Linked views are a key concept of information visualization [Spe06]. Combined with *interactive brushing* multiview visualizations provide a very powerful tool to discover and assess multi-dimensional relationships in the data [BC87]. Recently, the use of information visualization methods for the analysis of scientific data sets has been an active area of research [DGH03, BW08, JBS08]. Here, we specifically build on concepts introduced by Doleisch et al. who use brushing of multiple linked views to analyze unsteady simulation data [DGH03]. We extend this work by incorporating immersive visualization, direct interaction with the data in three dimensions and a parallel compute server for large data processing. Conceptually, this work is similar to query-driven visualization [SSWB05]. However, brushing linked views provides a graphical interface for query specification.

Various renderings of three dimensional scatterplots have been presented, e.g., by Piringer et al., but these are still limited to 2D renderings of 3D data [PKH04]. The use of three-dimensional scatterplots embedded into a virtual environment has been evaluated by Arns et al., as well as Raja et al. [ACN99, RBLN04]. Both studies conclude that immersion can be beneficial for information visualization. However, the techniques were only evaluated for rather small data sets and the authors conclude that further investigations into VR-based information visualization are warranted.

3. Multiview Visualization in Virtual Reality

In general, our visualization approach relies on multiple linked three-dimensional scatterplots which are presented and interacted with in a virtual environment, as shown in Figure 1. Each plot shows an arbitrary combination of three data attributes which are recorded per grid point. Time-dependent data is handled by animating the plots.

The overall method's fundamental idea is to quickly generate and check hypotheses about the data by iteratively marking interesting data ranges and then cross-reference to the spatial plots in order to find out, where exactly the marked values occur in the data set. Interactive *brushing* allows one to highlight a set of points by marking them with a three-dimensional selection box. In effect, this box is a directly manipulable visual representation of three one-dimensional range queries. Based on the range queries, a selection status is computed for each point using a logical operation over all active queries. As stated above, *linking* en-

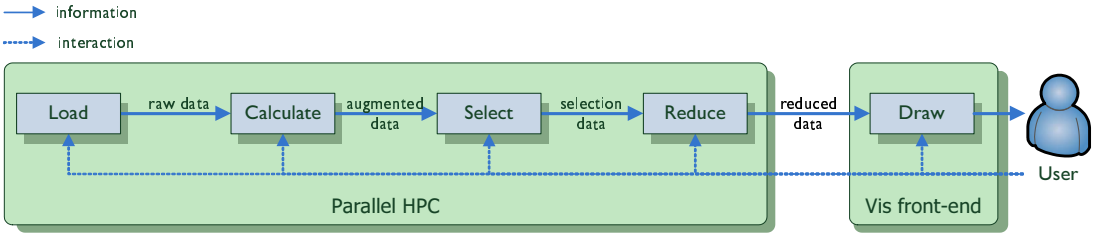

Figure 2: *Pipeline model for the data flow during the analysis process. It facilitates the distribution of compute intensive tasks to a parallel high performance computer (HPC) which in turn enables an interactive work process on large data. Moreover, an explicit reduction step makes data reduction user-controllable.*

sures that a change to the selection in either plot is directly reflected in all other plots.

The underlying computations are organized along the pipeline model depicted in Figure 2. It consists of five consecutive stages, each of which may be influenced by user interaction. First the data has to be *loaded* from secondary storage. This stage contains a data management facility, which loads the data with respect to the data requests from downstream pipeline stages. For transient data, caching and prefetching strategies help to reduce overall runtimes by overlapping storage access with computations. Based on the raw data, the user may choose to perform a *calculation of derived attributes*. These may range from simple magnitude operations for vector valued attributes to complex feature indicators. In the following *selection step*, we do not filter out unselected points but rather compute a continuous degree of interest (DOI) value as proposed by Doleisch et al. [DGH03]. The current selection computation tests all points in a linear pass over the data, i.e., it does not rely on any kind of indexing. This strategy is readily parallelizable. The selection phase is followed by a *data reduction stage*, where the data is brought down to a manageable and notably a displayable resolution. The rationale behind this lies in the fact, that today's data sets have a resolution that by far outreaches the resolution of even state-of-the-art displays. Therefore, displaying this data always implies a resolution reduction. We incorporate this step explicitly in order to make it user controllable. Moreover, this stage will sort out all attributes which are not needed for the current display setup. After this step the data is finally prepared for *drawing*.

In our system, the pipeline is split up into two parts. In order to process data sets of meaningful size, all compute intensive operations are out-sourced to a parallel machine which provides enough computing resources. Only the final display in a virtual environment and the interaction therewith are done on a visualization workstation. This separation facilitates scalability in two ways: The parallel machine may be scaled from a small cluster to a super-computer depending on data set sizes and user needs. Much in the same way, the visualization front-end may range from a desktop-based fishtank VR solution to a CAVE-like display.

In the next three subsections we will first show how direct 3D-interaction with the data is implemented, then discuss the details of the parallel data processing and finally describe the data reduction scheme.

3.1. Brushing 3D-Scatterplots via Direct Interaction

As outlined above, a rectangular brush may be created on each scatterplot. This brush is represented by a box-shaped widget, which we call *drag box*. The widget is defined via a 6-DOF tracked wand with – at least – one button. Three different forms of manipulation are implemented, as depicted in Figure 3. First, an initial box is defined by pressing the device's button, then dragging the device along the desired box's body diagonal and releasing the button again. The process is illustrated in Figure 1. Whenever the user positions the 3D-cursor outside of this box and presses the button, a new box is created. Throughout this process the previously defined box remains visible in addition for reference. Second, each of the range queries represented by the box may be exclusively edited. This is done by placing the cursor next to either of the box's faces and pressing the button. Third, when the user positions the device inside the box and presses the button, the entire box can be moved around freely.

During interaction, a small crosshair cursor, which is slightly offset from the device's center, serves as reference in the data. Each of the three actions is indicated by a small tooltip which pops up next to the cursor. Moreover, the entire box is highlighted whenever the cursor is inside, whereas a face will be highlighted, whenever the user is close enough to move it. This style of interaction is very direct because it does not rely on any form of picking ray and thus all selections are made within arms reach.

Regardless of the data's size, immediate visual feedback is key to interactive brushing. Although we will show below how remote computation helps to minimize system latency, we additionally guarantee immediate user feedback by using the GPU to highlight all points in a plot which are affected

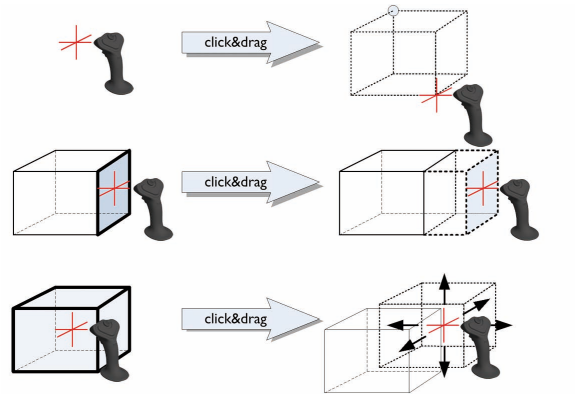

Figure 3: *Illustration of the drag box's possible interactions. A selection box is defined by dragging its body diagonal (top). Once a box is defined, either a single face can be changed (center) or the entire box may be moved (bottom).*

by the current operation. However, this feedback only affects the active plot, i.e, the GPU-based highlighting cannot be linked to the other plots. For these, a small traffic light metaphor indicates the update status by turning red or green depending on whether the plot's information is current.

3.2. Task Distribution and Parallelization

In order to provide near real-time updates even in the face of multi-gigabyte data sets, we use a client-server setup comprising two systems. The server typically is a parallel computer with enough main memory to load the raw data. As outlined above, all computations that directly involve the raw data are done on this system. The client system is responsible for managing user input and image generation. User interactions trigger update requests which are sent to the server, where the corresponding result is computed and sent back for immediate display. A streaming protocol is used in order to help keep the client-side memory footprint small. At any given time, the client only stores the displayable data for a few time steps.

In order to obtain good scaling, we exploit a natural decomposition of the underlying computations at three different levels. First, each time step can be handled independently. Thus, we distribute updates for different time steps to different processes using MPI [GSL99]. A dynamic scheduling is employed in order to minimize waiting times for visible results and maintain a good load balancing. Whenever possible, an update request is assigned to a free process that has already loaded the necessary raw data. This may either be a process which has already completed a previous update for the requested time step or one which has just prefetched the raw data using the underlying data management system. A message passing approach was chosen in order to be able to use distributed memory systems. The pipeline instance in each process is executed for the assigned time steps one at a time. Second, we use threads to decouple tasks such as data loading, communication, and pipeline execution from each other. This ensures a pipelined execution of independent tasks for successive time steps. Third, each filters' execution is internally parallelized. Most filters' computations require at least one pass over the entire point set for a single time step. Because the computations for each data point are independent of each other, this can be efficiently parallelized using shared memory parallelization with OpenMP [CJP07]. This results in a faster update without requiring explicit data distribution or extra inter-process communication.

Data transfer between the server and the client machines is a major bottleneck, both in terms of total time and latency. In order to minimize the effect, we carefully track the data which is present on the client side at any given time. Based on this information we use minimal re-transmissions of data. There are two different transmission formats: First, when the user creates a new plot, meta information and full data attributes for the entire time-dependent data set have to be sent. Meta information includes, for instance, the total ranges for each attribute over the entire time range. Second, for each change of a query, we only transmit the updated selection information. This is by far the most frequent form of transmission, because the point positions in every plot remain constant once the data attributes have been fixed. Selection values are conveniently encoded using 8 bit per point, which provides a good trade-off between continuous DOI values and space requirements. The visualization client handles incoming data with a set of dedicated threads. This helps to provide constant frame rates in the immersive environment even during data reception.

3.3. Data Reduction

In order to reduce the data we employ a straightforward quantization scheme which bins the input point cloud to a Cartesian grid of limited resolution. Each input point is mapped to its corresponding cell in this grid. The number of hits per cell is counted and later serves as density field over the entire quantization grid. After the mapping, a single output point is created for each grid cell which contains at least one input point. In order to speed up the process, we only execute the binning process when the attribute data has changed. For the more frequent case where only the selection status for points changes, we keep an index which directly maps input points to output points. It is computed on-the-fly during the initial binning pass. Based on the index, the selection status of output points can be determined with only a single pass over the input data. This results in a significant speed-up of selection updates.

4. Results

The system's evaluation includes the discussion of a set of runtime measurements and an application case study con-

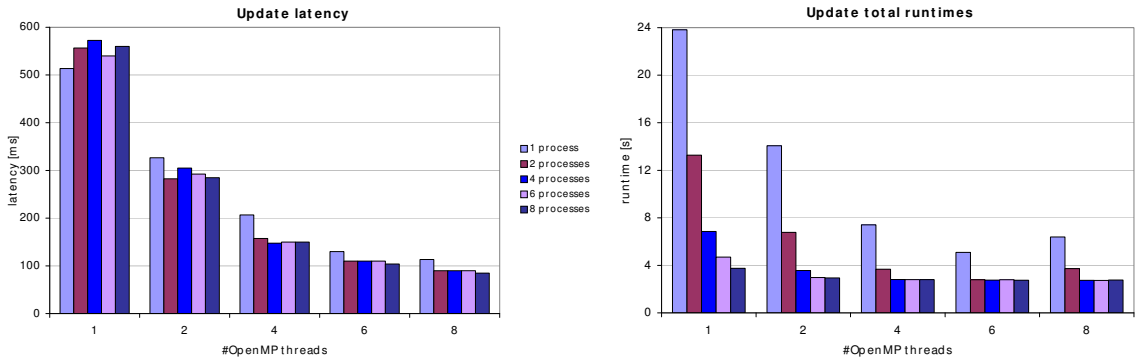

Figure 4: *Overview of update latencies (left) and total runtimes (right) for the crossflow data set.*

ducted in collaboration with domain scientists. All tests were based on a 3D unsteady simulation data set. An overview of the simulated phenomenon, a so-called *jet-in-a-crossflow*, follows in Section 4.2. The data set resulted from a large-eddy simulation (LES). It comprises 100 time steps, each of which is given on an unstructured grid with 2.9 million points. The result contains 10 scalar and 5 vector attributes and consumes a total of 54 GB of disk space.

For the runtime measurements we used a compute cluster of eight nodes, each equipped with 2 Intel Xeon E5450 CPUs (4 cores, 3.0 GHz) and 32 GBytes RAM. The client application ran on a workstation equipped with an Intel Core 2 Q6600 (4 cores, 2.4 GHz), 8 GB RAM, and an NVIDIA Quadro FX 4600 graphics board. It has a 1 GBit non-dedicated Ethernet connection to the server. The case study has been performed in a five-sided CAVE-like environment driven by an 11-node LINUX cluster. Each node features two AMD Opteron 2218 CPUs (2 cores, 2 GHz), 8 GB RAM, and an NVIDIA Quadro FX 5600 graphics board. GBit Ethernet is used for both, the internal cluster network, and the link to the server. The frame rate continuously exceeded 40 fps during all visualization sessions.

4.1. Performance

In order to get comparable results for all different configurations of MPI processes and OpenMP threads, we restricted the runtime measurements to 48 equally spaced time steps. These fit into the 32 GB of memory provided by a single node and therefore this setup allows in-core measurements using a single system. The performance measurements account for selection update times only, i.e., they include neither the time to load the data from secondary storage nor the initial full update. Both are carried out during system initialization. We conducted in-core measurements, as the efficiency of the data management is not evaluated here and otherwise would have affected the runtime measurements. For the measured setup, we executed a worst case query which switched the selection status of all data points from unselected to selected. Three plots had to be updated, which were configured in the same way as for the case study. The selection update for all three plots totalled 624 KB per time step. Compared to 10,612 KB for a full attribute update per time step, the selection-only transmission reduced communication to 5.8% of the original size.

During exploration, two different update times are relevant: On the one hand we have to minimize *total computation times* and on the other hand we have to reduce *update latencies*, i.e., the time to arrival for the first results as much as possible. While reducing overall run-times is generally desirable, cutting latency is particularly important in an interactive setup in order to provide immediate feedback to the user. Both times are evaluated in an end-to-end fashion, i.e., we measure the time between the user interaction, which triggers the update, and the point where the updated data is first rendered. Notably, the restriction to 48 time steps only influences total runtime but not latency, because this only accounts for the first time step update and all time steps share the same size.

The measurements were conducted to analyze whether the system is suitable for interactive use and if so, which configurations of MPI processes and OpenMP threads are particularly effective. The basic hypothesis is, that a larger number of MPI processes reduces overall runtimes whereas a higher number of OpenMP threads leads to shorter latency times.

Figure 4 (left) shows the latency times for various configurations of threads and processes. It is evident, that a larger number of threads in fact leads to a significant reduction of latency, up to the point where interactive updates with reaction times below 100 ms become possible; for the eight processes/eight threads configuration the average latency is 85 ms. As hypothesized, no significant effect of the number of processes on the reaction time can be seen, although the results for a single process are up to 38% slower than those

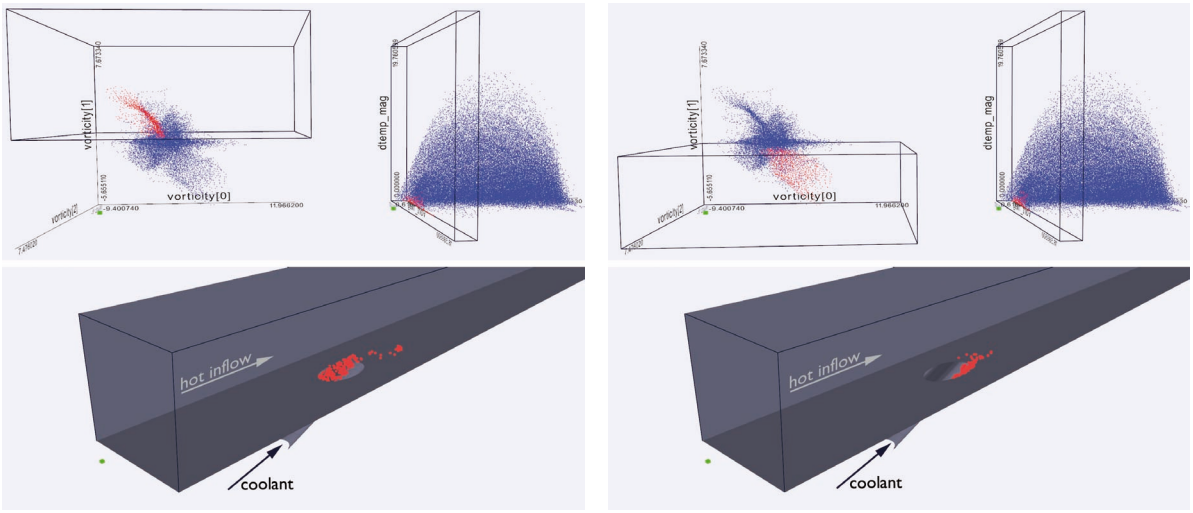

Figure 5: *Analyzing the formation of the counter-rotating vortex pair on both sides of the injection hole. Each of the scatterplots in the top row shows the current selection box. Selected points, i.e., points which lie inside both boxes, are highlighted in red. Unselected points (blue) are not shown in the spatial plots. Points with low temperatures are selected in the right plot of both top images. The top left image shows points which additionally feature positive streamwise vorticity, the top right one those showing negative streamwise vorticity. The lower images show the resulting point sets inside the simulation's domain boundary.*

of the other setups for a larger number of threads. The exact reason for this behavior has yet to be determined.

In Figure 4 (right) the overall runtimes are compared to each other. Again, the predicted behavior is immediately apparent. A larger number of MPI processes leads to a reduction in overall runtimes, as more time steps are processed in parallel. Because there is only very little communication between processes, the scaling is almost ideal when switching from one to two processes. Overall update times reach 2.7 s for all configurations with eight processes and more than two threads. However, as the overall number of utilized CPUs increases, the scaling significantly degrades. This is due to the fact, that the visualization client is no longer able to process even the reduced amount of incoming data. Moreover, communication becomes a limiting factor.

Overall runtimes even increase for one and two MPI processes, when increasing the number of OpenMP threads per process from six to eight. This again results from a communication bottleneck: Each process has one additional thread for sending results off to the client machine. In the aforementioned cases, this thread is overloaded resulting in an increase of transmission times. Note, that the first package to be sent off is not affected by this and thus latency times still do decrease.

In order to prove that scalability to a larger number of time steps, we conducted another measurement with 16 MPI processes and four OpenMP threads and used 96 out of the 100 time steps for this. In this measurement, latency was 157 ms while the overall update took 5.5 s. Assuming a data animation which shows ten simulation time steps per second user time, each animation cycle will take approximately 10 s of real-time. Therefore, the system is able to update time steps within one animation cycle and thus the user will hardly notice the computation.

In conclusion, we can say that the proposed combination of MPI and thread-level parallelization is well suited for the given visualization problem. While being quite heavy weight, the MPI paradigm enables the use of distributed memory systems, e.g., compute clusters. Handling time steps independently in such a setup significantly reduces memory demands on the individual systems. For example when using eight processes in the given example, each node had to store six time-steps resulting in a footprint of only 3.3 GBytes. Hence, an increase of processes is favorable for data containing more time steps. On the other hand, shared memory parallelization with OpenMP brings a good speed-up for low-level tasks at virtually no cost for the developer. It therefore helps to utilize the individual cores of each sub-system in a cluster. This results in a faster computation of results for individual time steps, which is favorable if these time steps are rather large. In our measurements, configurations with eight OpenMP threads minimized latency. However, scaling for both strategies is limited by communication: in the former case because the front-end is eventually saturated with incoming data, in the latter because the data cannot be sent off in a timely fashion. It is therefore important to note, that a system like the proposed one has to be carefully optimized in an end to end fashion.

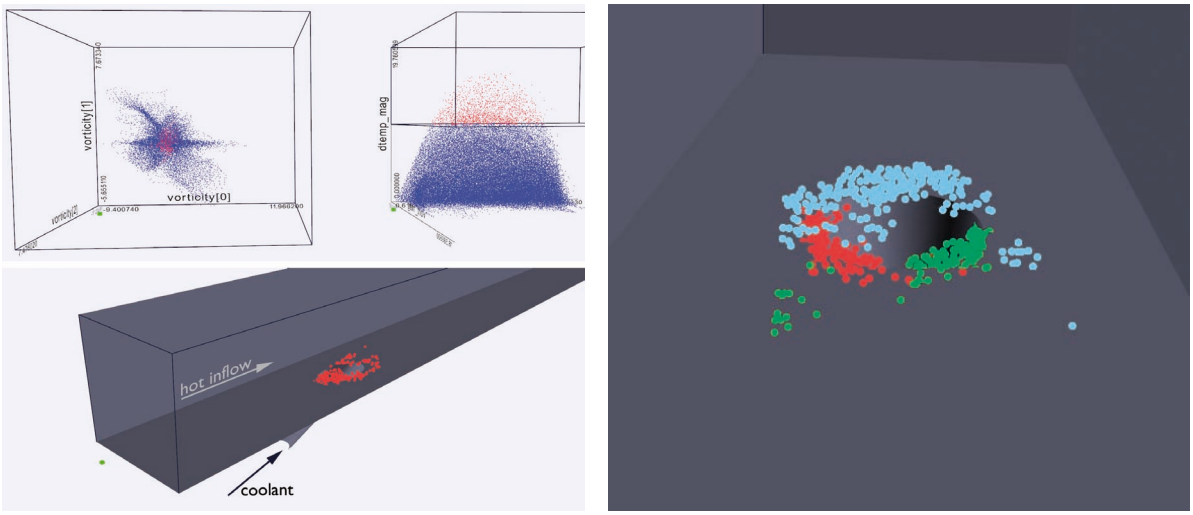

Figure 6: *Left: Marking high temperature gradients (top) reveals the shear layer in front of the injection hole (bottom). Right: An artificially created close-up overlaying the three different selections: CVP (red, green) and shear layer (light blue).*

4.2. Case Study

We have shown in the last section, that the parallelization facilitates interactive working on the example data set. We now discuss observations from an exploration session, which has been carried out by researchers from the field of fluid mechanics. The jet-in-a-crossflow simulation underlying this study describes the injection of a coolant into a turbulent flow of hot fluid. It is motivated by the fact that in gas turbine engines a high thermal efficiency is generally reached by high inlet gas temperatures. Film cooling techniques are investigated to protect the turbine components from the induced thermal stresses. These techniques generate a thin film layer between the surface and the hot combustion gases. Since the technical design process of film cooling systems depends on the exact knowledge of the generated flow field, a detailed understanding of the flow physics is crucial in order to improve existing cooling techniques.

In the presented case, the cooling film is generated by an injection of a cooling fluid through a row of staggered holes. Under the assumption of symmetry the simulation is restricted to a single jet hole. The flow field resulting from the interaction of the inclined cooling jet and the turbulent boundary layer is governed by complex vortex dynamics, which is hard to predict in advance. The averaged flow field downstream of the jet hole is dominated by a *counter-rotating vortex pair* (CVP), which is the leading mechanism in the mixing process between the hot gas and the coolant. A detailed description of the numerical method can be found in Renze et al. [RSM08].

Here, we focus on the analysis of the CVP in the instantaneous flow field in order to understand the coherent structures in the averaged flow field. It is of particular interest, where exactly the vortices form and how they influence the mixing process right after leaving the injection hole. This is one example for an analysis task where the exact outcome is not known in advance and therefore cannot be described by an analytical model suitable for automatic extraction. To analyze the CVP's behavior, we set up three different plots, as shown in Figure 5: The first one shows the three components of vorticity ω. Here the streamwise and spanwise components, i.e., ω_x and ω_y, are of central interest. The second plot contains the temperature T, the magnitude of the temperature gradient $\|\nabla T\|$ and the vorticity magnitude $\|\omega\|$. The third plot shows the three coordinate axes. For additional spatial context a polygonal model of the simulation domain is provided. Generally, selected points are highlighted in red, whereas unselected points are shown in blue. While the first two plots show all the data points regardless of their selection status, the third plot is restricted to the selected points, to prevent a cluttered visualization in the spatial domain.

It was known in advance, that the CVP forms right at the edges of the injection hole. Points in this region are characterized by low temperature, which resulted in a first selection criterion. The vorticity distribution exhibited an interesting shape in the (ω_x, ω_y)-plane, which resembles a mirror-inverted integral sign. Marking the upper leg of this structure, i.e., points where $\omega_y > 0$ shows a vortex on the right hand side of the injection hole (cf. Figure 5, left). Setting the selection to the lower leg shows the matching counter-rotating vortex on the other side of the hole (cf. Figure 5, right). From the images it is evident, that the strong vorticity is generated at the edges of the jet hole, which has a strong impact on the flow field's instantaneous turbulent structures. It can be recognized as the CVP in the averaged flow field.

Since these vortices are the driving factor of the mixing process further downstream, it is interesting to see them in correlation to the shear layer, where the turbulent hot inflow first hits the coolant. This layer is characterized by high temperature gradients. Marking these in the right plot, it becomes obvious that the layer passes above the formation region of the CVP (cf. Figure 6, left). Interestingly, all of these points lie in the central part of the vorticity plot, i.e., they share low vorticity in all of the components. An artificially created overlay of the three different selections clarifies the relative positions of all three structures (cf. Figure 6, right).

5. Summary and Future Work

In this paper we have described an interactive data analysis method, which combines the expressiveness of a VR user interface with the processing power of parallel computers. Both main requirements stated in the introduction have been fulfilled. First, the drag box widget's design allows the user to intuitively interact with the data. In the case study, the use of Virtual Reality has not only been proven to work but was reported to help domain experts to correlate different flow variables in a readily accessible way. Second, the shift of all major compute tasks to a parallel system enables interactive work on data which would not be manageable on a standard workstation. The explicit inclusion of data reduction before transmission is an important step in order to bring the data down to a displayable resolution. Moreover, data reduction and minimal retransmission help to reduce system latency which is mandatory to facilitate interactive brushing.

Regarding future work, we see two main issues: First, we have to acknowledge the fact that we are dealing with continuous data rather than isolated points. A straightforward way of displaying data points may lead to overdrawing problems and even misinterpretations, as was recently pointed out by Bachtaler and Weiskopf [BW08]. Second, a more formal evaluation would be beneficial. However, here we face the problem of weighing a reasonably complex exploration task against a meaningful number of qualified participants.

Acknowledgements

This work has been partially funded by the German Research Foundation (DFG) under grant WE 2186/5. The crossflow-data set has been simulated as part of the Collaborative Research Centre (SFB) 561.

References

[ABM+01] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka. Large-Scale Data Visualization using Parallel Data Streaming. *IEEE Computer Graphics and Applications*, 21(4):34–41, 2001.

[ACN99] L. Arns and C. Cruz-Neira. The Benefits of Statistical Visualization in an Immersive Environment. In *Proceedings of IEEE VR*, pages 88–95, 1999.

[BC87] R. A. Becker and W. S. Cleveland. Brushing Scatterplots. *Technometrics*, 29(2):127–142, May 1987.

[BGY92] S. Bryson and M. Gerald-Yamasaki. The Distributed Virtual Windtunnel. In *Proceedings of of the IEEE Supercomputing '92*, pages 275–284, 1992.

[BW08] S. Bachthaler and D. Weiskopf. Continuous Scatterplots. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1428–1435, 2008.

[CJP07] B. Chapman, G. Jost, and R. Van Der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, December 2007.

[DGH03] H. Doleisch, M. Gasser, and H. Hauser. Interactive Feature Specification for Focus+Context Visualization of Complex Simulation Data. In *Proceedings of the Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, pages 239–248, 2003.

[dHKP02] G. de Haan, M. Koutek, and F. H. Post. Towards Intuitive Exploration Tools for Data Visualization in VR. In H. Sun and Q. Peng, editors, *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST)*, pages 105–112, 2002.

[GSL99] W. Gropp, A. Skjellum, and E. Lusk. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 2nd edition, November 1999.

[JBS08] H. Jänicke, M. Böttinger, and G. Scheuermann. Brushing of Attribute Clouds for the Visualization of Multivariate Data. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1459–1466, 2008.

[KBB+06] O. Kreylos, T. Bernardin, M. I. Billen, E. S. Cowgill, R. D. Gold, B. Hamann, M. Jadamec, L. Kellogg, O. G. Staadt, and D. Y. Sumner. Enabling Scientic Worksflows in Virtual Reality. In *Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications (VRCIA2006)*, 2006.

[PKH04] H. Piringer, R. Kosara, and H. Hauser. Interactive Foxus+Context Visualization with Linked 2D/3D Scatterplots. In *Proceedings of the 2nd Int. Conference on Coordinated & Multiple Views in Exploratory Visualization*, pages 49–60, 2004.

[RBLN04] D. Raja, D. A. Bowman, J. Lucas, and C. North. Exploring the Benefits of Immersion in Abstract Information Visualization. In *Proceedings of the 8th IPT Workshop*, pages 61–69, 2004.

[RSM08] P. Renze, W. Schröder, and M. Meinke. Large-Eddy Simulation of Film Cooling at Density Gradients. *Int. J. Heat Fluid Flow*, 29:18–34, 2008.

[SBK07] M. Schirski, C. Bischof, and T. Kuhlen. Interactive Exploration of Large Data in Hybrid Visualization Environments. In *Proceedings of the 13th EGVE and 10th IPT Workshop*, pages 69–76, 2007.

[SGvR+03] 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 9th EGVE and 7th IPT Workshop*, pages 77–85, May 2003.

[Spe06] R. Spence. *Information Visualization: Design for Interaction*. Prentice Hall, 2nd edition, December 2006.

[SSWB05] Kurt Stockinger, John Shalf, Kesheng Wu, and E. Wes Bethel. QueryDriven Visualization of Large Data Sets. In *Proceedings of IEEE Visualization*, pages 167–174, 2005.