Table of Contents

Acknowledgments

The organizers of the 5th High-End Visualization Workshop wish to thank the LSU Center for Computation & Technology, Baton Rouge, Louisiana, U.S.A. for sponsorship and local organization. Thanks also to Mercury Computer Systems Visualization Sciences for sponsorship.

The 5th High End Visualization Workshop took place on the LSU campus March 18-21, 2009. This year's feature was remote and collaborative visualization. The proceedings contain the six original papers submitted for this workshop.

Beyond the Visualization Pipeline: The Visualization Cascade

Werner Benger[1] and Georg Ritter[2] and
Marcel Ritter[1,3] and Wolfram Schoor[3]
[1]Center for Computation and Technology, Louisiana State University, USA
[2]Institute for Astro- and Particle Physics, University of Innsbruck
[3]Institute of Computer Science, University of Innsbruck, Austria
[4]Fraunhofer Institute for Factory Operation and Automation
werner@cct.lsu.edu, georg.ritter@uibk.ac.at,
marcel@cct.lsu.edu, Wolfram.Schoor@iff.fraunhofer.de

Abstract

The concept of a pipeline has become a quite common way of thinking about the process of visualizing data. In this article we discuss the inherent limits of this concept and argue for the need to expand this concept for achieving higher performance and convenience to the end user. While the traditional model of a visualization pipeline describes the execution of some data flow, it is most suitable for a static data-set. However for time-dependent data (e.g.) we intend visualizations to be as fast in time as they are in space. The pipeline model does not recognize similarity and repetition of operations which is essential to achieve the desired performance. The pipeline model thus needs to be extended to efficiently cover multiple traversals and caching of intermediate results, which we call the *Visualization Cascade*. It will be demonstrated in practice within its implementation in the VISH visualization environment.

1 Introduction

The process of visualizing data begins with the source data containing the information to be visualized and ends, finally, with a derived image representing the data. To arrive at the image the data needs processing, like being searched or filtered, depending on the nature of the data and the analysis requirements. It then must be mapped to graphical entities that are subsequently rendered into an image. In [Haber & McNabb, 1990] the authors identify and refine the general operations data undergoes in the process of creating visualization. The data flows in a pipeline through a chain of stages, as depicted in Figure 1, and finally, results in a representing image. The pipelined model they present, is known as the Haber-McNabb model of the visualization pipeline and

has been a widely successful concept for the design of visualization software. Several well known software packages have been built upon this idea, examples include AVS [Upson et al., 1989], VTK [Schroeder et al., 1997], IRIS Explorer [Foulser, 1995], OpenDX [Treinish, 1997] (for a more complete list see [A.A. Ahmed, 2007]).

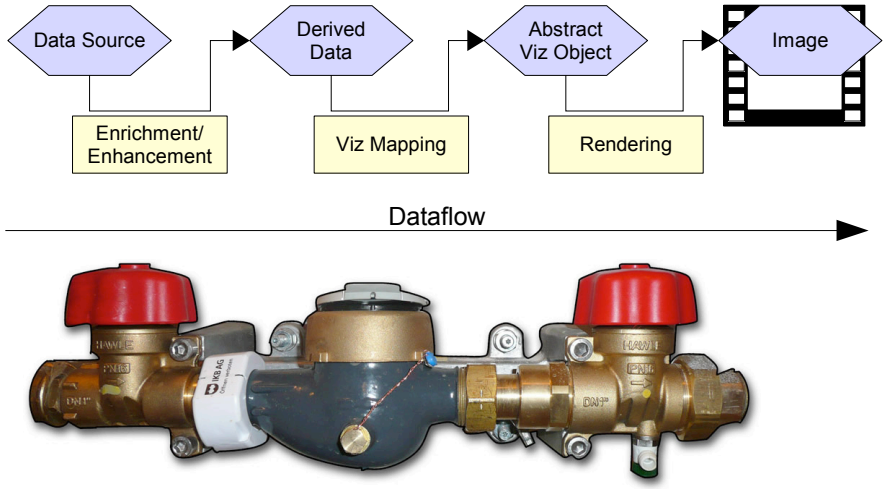


Figure 1: Visualization Pipeline: Data flows through the pipeline while operators modify the stream. They extract, filter, map and render data. Finally the pipeline outputs is an image or image stream.

While trying to understand the data through exploratory visualization, as outlined in [Upson et al., 1989, Card et al., 1999], a flexible and easy to use mechanism to enter the operation on the data into the software is needed. Exploring and trying to understand as much as possible from the data by inspecting them, includes making frequent changes to parameters of the visualization, sometimes even to the parameters or models used to create the data. Being able to easily change and adapt the parameters and, in addition, derive the visualization results fast, is of great importance as it helps to decrease the time needed for one investigative cycle. For some data, features might only become visible if it is possible to navigate interactively inside the parameter space.

The pipeline concept has been found to be well suited and intuitive to understand when used to represent the flow of data in a user interface. The user can graphically construct a visualization pipeline by interconnecting different stages through attaching a pipeline to nodes. An early example of such a graphically programming interface has been implemented in ConMan [Haeberli, 1988], more modern examples include the Spiegel framework [Bischof et al., 2006], or the graphical user interface of LabVIEW [Johnson & Jennings, 2001] in which a data stream, mainly originating from measurements, can be connected to processing nodes or the gstreamer framework [Black et al., 2002] in which multi-

media data is handled this way.

Once the structure of the pipeline has been set up, it needs to be executed. Different schemes have been implemented to derive the final image. When using implicit execution, as applied in VTK [Schroeder et al., 1997], data is time stamped and only "upstream" nodes are executed, if demanded. Explicit execution, as used in the IRIS Explorer [Foulser, 1995], relies on external management of the data processing nodes to decide which needs re-computation.

The flow of data is initiated in two principal ways. In the "pull" case, a downstream receiver requests the data from an "upstream" node. In a "push" case, the "upstream" node would forward the data to the next stage in the pipeline as soon as it is available. Also a mixed version can be implemented, as they are independent.

If we want achieve full interactivity in the visualization of large data-sets we find that the current design of pipelines and their execution models are not well suited to meet the requirements. Response times to changes of parameters are not fast enough, as data travels too slowly through the whole pipeline to reach an interactive frame rate, as described in [Shen, 2006] for the case of time dependent data.

Not only for a change in the time parameter, but for any change made to a parameter of the visualization, the whole visualization pipeline has to be executed for every single frame. As this often exceeds the time required for an interactive frame rate, a common solution is to apply off-line rendering. Frames of an animation sequence are rendered for later viewing, but interactively exploring the data would be more desirable and would also possibly increase the chance of gaining further understanding of for example spatial-temporal features of the data.

Working toward the visualization challenges one ("Make the spatial and temporal resolution of visual displays indistinguishable from physical reality.") and four ("Optimize physical resources used to perform visual interactions."), as described in [Hibbard, 1999] and incorporating the user wishes for interactivity, here we present an extended pipeline model that utilizes the structure of modern graphics hardware to minimize the time needed to derive the final image. We propose that, by introducing a flexible caching mechanism, it is possible to increase re-usage of already processed data, especially in between different pipelines constructed for different parameter sets. In combination with a data storage model and utilizing GPU memory, full interactivity on a data-set of the size of 17 GB, containing 1.6 million points in 200 time steps [Benger, 2008], has been achieved.

2 The Visualization Cascade

The major drawback of the concept of the visualization pipeline is that it does not talk about caching of results. If an operation similar to an earlier is to be repeated, we would not want to have the entire pipeline to be traversed again. Only those sections that have changed shall be re-computed. A typical usage

scenario is running an animation of time-dependent data. The "conventional way" is to load each time step, pump it through the visualization pipeline to create a pixel frame for each time step, and then eventually watch the evolution of the data as a movie. Many features of a dynamic data-set are only appreciated when viewed at interactive speeds of e.g. 30 frames per second, but usually the traversal of the entire visualization pipeline is much slower than 1/30th of second.

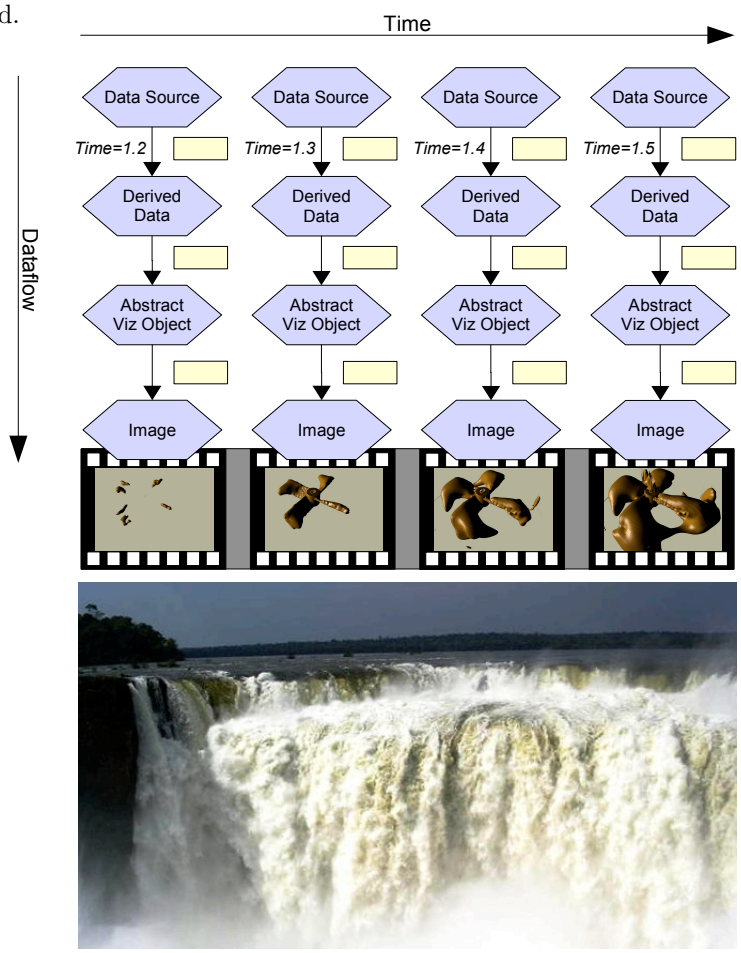


Figure 2: *Exploring a full parameter space of some data-set requires parallel traversal of the data flow - resulting in a cascade rather than a single pipeline traversed repeatedly.*

While speeding up the initial traversal of the pipeline might just be impossible, results of previously computed operations can be cached up to available RAM. We may consider the rendering of an animation as the execution of a sequence of multiple parallel visualization pipelines. The execution nodes of each

pipeline reside on the same level. At each such node we might need to cache some result of a previous operation. We may think of such a system as a cascade of data flowing down a water fall, in many parallel ways and intermediate levels where data reside to be cached.

Furthermore, data may eventually flow from one stream to another one, i.e. the "visualization pipeline" from one time frame may employ parts of a visualization pipeline from another time frame as well. Such may be the case when fusing data-sets given at different time intervals, for instance a data-set that is sampled at T=0.0, T=10.0 and another one at T=0.0, 1.0, 2.0, etc. If interpolation in time is not requested but rather the "most recent" timestep should be displayed, then at T=1.0 the coarse data-set at T=0.0 would be used, which does not require traversal of the viz pipeline for the coarse data-set all they way up to its source. A new computation will only be required when both data streams from the two pipelines will merge.

We may consider "time" as a parameter that is orthogonal to the flow of the visualization pipeline. It rather parameterizes the visualization pipeline (a linear, one-dimensional data flow) and unfolds it into multiple instances, thereby creating the visualization cascade (a two-dimensional flow of data). At each cascade level, there will be the essential decision when to re-execute the computation or to re-use existing data. This depends on additional parameters that have been changed since the last traversal. If the data at each level depend on "time" only, then there is no need for re-computation at all once data exist there already. However, there may be other parameters as well, such as depending on user interaction. For instance, when inspecting some time-dependent 3D data volume, the user-defined threshold level of an isosurface, or the range and color values of a colormap during volume rendering. In both cases, there is no need to reload data from disk when repeating an animation over a previous time range. However, in the case of the isosurface display, the computation of the geometry has to be re-executed. In the case of the dynamic volume rendering, there is not even a need to reload data on the graphics card, but only some texture maps might need to be updated when changing the colors.

While some parameters in the visualization cascade might not require requesting data up from the source, others might. For instance, changing the range of a volume rendering colormap or applying another filter (e.g. a non-linear filter) may require operating directly on the original data, and thus need to reload data and full pipeline traversal. We therefore have to distinguish between two classes of parameters: those that require re-computation, and those that do not. The efficiency of the visualization cascade will depend on proper choices at each node, to avoid unnecessary computation but still perform the essential ones. We will discuss our implementation in the next sections.

2.1 Data Result Caching

Each node within a visualization pipeline is an operation on the data stream. In an object-oriented environment, it is usually an object with data structures and member functions. It is straightforward and common use to store computational

data in here. Using this associated node-local space as cache for intermediate results is an option, but not an optimal one.

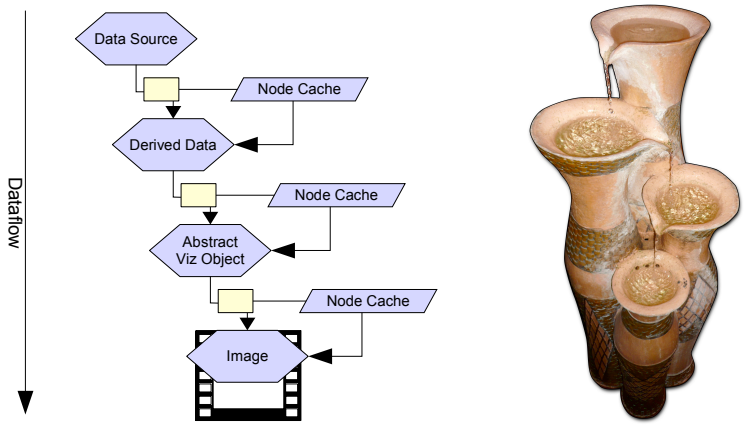


Figure 3: Caching of intermediate results of the data flow through a visualization pipeline allows to avoid repeating previously performed computations. Instead, final results may be retrieved from intermediate levels of the visualization *cascade*.

In our case we utlize Vish [Benger et al., 2007] as visualization environment, which comes with the option to use a so-called fiber bundle data model [Benger, 2004, Benger et al., 2006, Benger, 2008]. This model is a framework to handle a wide class of data for scientific visualization within the same structures. It intrinsically supports time dependency and thereby allows to store intermediate results within this data model (a data structure available at the data source) rather then the computational objects themselves.

This approach to store intermediate results directly at the data source (the so-called "Bundle") as extension to the source comes with various benefits:

1. The computational nodes are kept completely procedural; they never store any data itself, and may thus be utilized for any kind of data operation even stemming from different sources. Data are merely seen as parameters to the procedure, but not actually "transported" into the object.

2. Another instance of the same procedure would automatically recognize existing results, as it would store its results in the same location. In the purely object-oriented approach, objects would not know about the existence of other instances.

3. Since the data source is equipped with I/O methods, all intermediate results can be stored on disk and reloaded at a later instance; there is no requirement to equip each computational node with explicit functionality to store its own data.

4. With additional data added to the source, they are available to be inspected with other procedures or visualization modules. This may well lead to unexpected discovery and insight into the data itself, with no additional cost, but in a natural way. Additional data are just available.

Within VISH, the functionality of an *Operator Cache* is provided to attach any kind of data to a data source with minimal requirements. If the data source is a *fiber bundle*, then a more specific method can be applied.

2.2 Operator Cache

The "Operator Cache" is a C++ template class used to memorize the result of some computational operation as implemented by a node of a visualization pipeline (the "Operator"). This generic approach only fulfills the first property in the aforementioned list. Hereby the data source has to provide the property to be an "Intercube" object, as described in [Benger et al., 2007]. Basically this is an runtime-version of multiple inheritance, which allows to attach additional objects ("interfaces") to some container (the "intercube" holding many "interfaces"), e.g. an object providing data for visualization.

For instance, if we want to memorize a vector of doubles, then we simply instantiate the OperatorCache template over this data type:

```
typedef OperatorCache<std::vector<double> > OC_t;
```

Now given an InterCube object provided to a computational routine, one may retrieve an OperatorCache object that may be stored there. If not, we would need to create one anyway:

```
void VizNode::compute(InterCube &MyData)
{
        OC_t*OC = OC_t::retrieve( MyData, this );
        if (!OC) OC = new OC_t();
}
```

Note that the retrieve function basically has two parameters, the data object "MyData" and the visualization node. Thus, the operator cache can install a copy of the requested data with each data object and each visualization node. It is a unique place where the node may store data outside its own local memory, as illustrated in Figure 4

The Operator Cache is furthermore related to a set of variable values, a "ValueSet". Its purpose is to associate the OperatorCache with such a set of values. If any of these values has changed upon a repeated call of the viz node's compute function on the *same* data-set then the Operator Cache needs to be equipped with data from a new computation. For instance we might consider an operator that computes some isosurface. If the isolevel value or some maximum number of allowed triangles is changed, then the operator would need to execute the numerical routine again, otherwise it could just return the data stored in the Operator Cache. An "OperatorCache::unchanged()" member function checks if

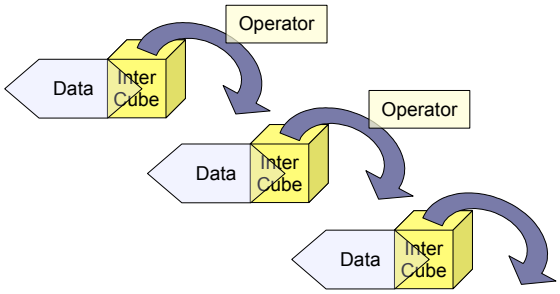


Figure 4: *Data Storage in Intercubes*

there are any such differences among the values stored with the OperatorCache and the current values had occured (it will automatically return "false" if the OperatorCache was newly created):

```
ValueSet*Changeables = new ValueSet();
        Changeables->addValue(IsoValue);
        Changeables->addValue(NumberOfAllowedTriangles);

        if (OC->unchanged( Changeables ) )
        {
                // do something with the data existing in OC
                return;
        }
        // compute new data and put them into the OC
```

If there had been changes, then following code is supposed to compute new data. There may be other parameters that do not require re-computation, such as another coloring of the resulting isosurface, see Figures 5, 6 and 7. These will be part of the visualization node, but not be added to the ValueSet that is used to inspect the Operator Cache. (The actual source code uses a slightly different syntax as it employs operator overloading to provide a more compact coding.)

2.3 Caching in the Fiber Bundle

When data are available in the fiber bundle, and results are storable in the fiber bundle, one would not employ the OperatorCache. Rather, any results will be stored directly in the incoming data structures. To depict how it works, we do not need to know the entire complexity of the full model. It suffices to know that there are objects called `Bundle` and `Grid`. A `Bundle` may be accessed with a floating point value and a string to yield a `Grid` object. Such a `Grid` object may represent a 3D data volume with scalar fields (which is to be identified via some string), or a triangular surface such as the result of an isosurface computation. The actual numerical routine "`Isosurface`" will require a `Grid` object, a field name, and a floating point value specifying the isolevel. The schematic code will look similar to this:

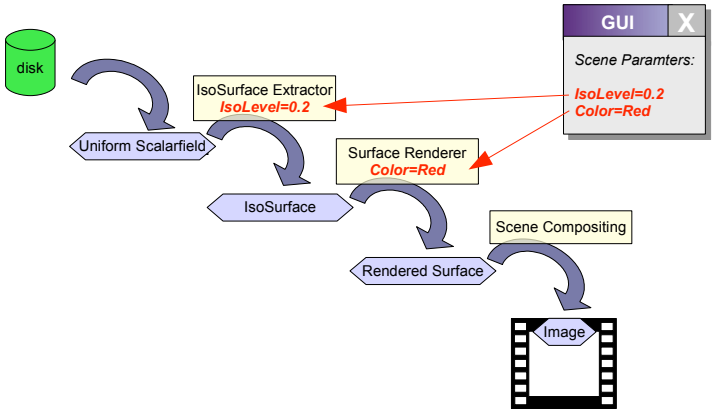


Figure 5: *The execution flow: The first time, the complete cascade has to be executed. The operators read the data-set from disk, compute the isosurface, render the surface and composite it to the final image. The separation into operators is hidden and is not all seen by the user.*

```
Grid VizNode::compute(Bundle&B, double time,
                      string Gridname, string Fieldname,
                      double Isolevel)
{
      // Construct a unique name for the computational result
string  IsosurfaceName = Gridname + Fieldname + Isolevel;
      // Check whether result already exists for the given time
Grid IsoSurface = B[ time ][ IsosurfaceName ];
     if (!IsoSurface)
     {
          // No, thus need to retrieve the data volume
     Grid DataVolume = B[ time ][ Gridname ];
          // and perform the actual computation
       IsoSurface = Compute( DataVolume, Fieldname, Isolevel);
          // finally store the resulting data in the bundle object
          B[ time ][ IsosurfaceName ] =  IsoSurface;
     }
     return IsoSurface;
}
```

Note that in case an IsoSurface Grid already exists, there is no need to request a DataVolume object. The operation of requesting such might be effortsome, including slow disk access (on-demand loading), numerical computation of the source field, network access, etc. Never are any data actually stored in the `VizNode` object itself. In this version, a new geometry is stored for each time step and each isolevel value. Since these are floating point values, this may well need to an immense number of surfaces that are stored when exploring the parameter space of time and isolevel. Therefore, some appropriate memory

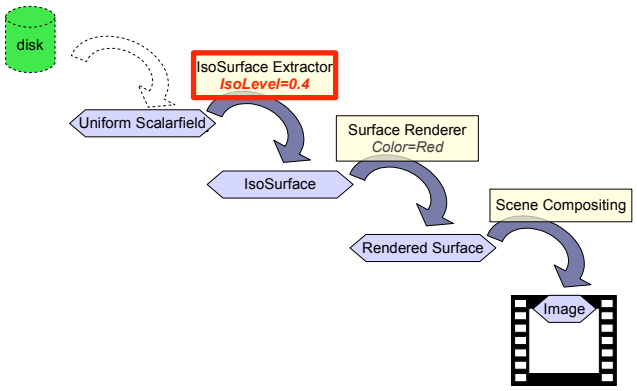


Figure 6: *The execution flow: Resulting data flow when changing the isosurface level parameter. The data-set need not be read from disk again.*

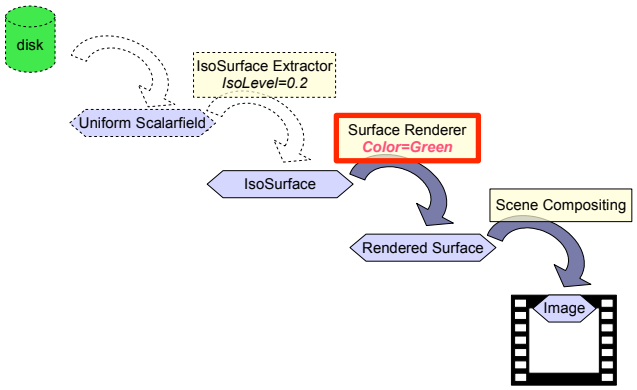


Figure 7: *The execution flow: Resulting data flow when changing the color of the iso surface. The isosurface need not be recomputed again.*

management that discards old objects that have not been accessed for a long time will be mandatory.

2.4 OpenGL Caching

The final stage of producing pixels using modern graphics hardware is loading data onto the memory of the graphics card (GPU). Once all data that are required for rendering are transferred to the GPU, pixel generation will be as fast as possible. Via means of OpenGL, large data at the GPU are modeled as *Display Lists*, *Textures* and *VertexBuffer Objects*. Framebuffer objects might fall into this category as well, but we did not consider them yet. While the graphics memory is limited, it may still provide enough space to also store objects that are not visible in the currently viewed frame but a previous one. Re-utilizing

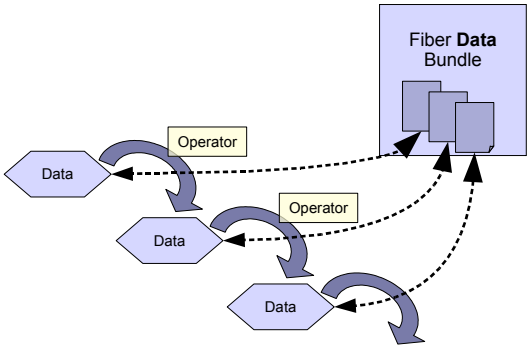


Figure 8: *Data Storage in the Fiber Bundle*

objects already stored in GPU memory is much faster than re-loading objects from RAM. We may expect so even in case the graphics driver is placing some object into RAM itself.

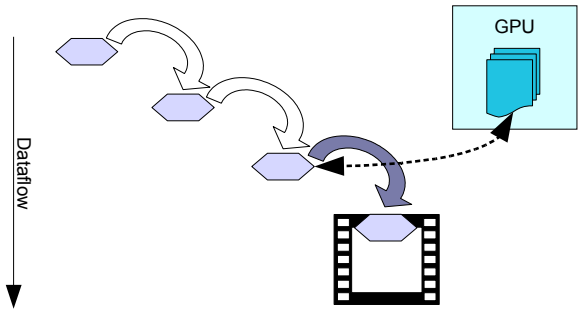


Figure 9: *Caching on the GPU*

In contrast to caching RAM data like those on the fiber bundle, GPU data is only available through some handle within an OpenGL context. It cannot be stored with the data source. We thus utilize a management system for the OpenGL handle identifiers that is associated with a viewer, called the "`GLCache`". The GLCache is a mapping from certain keys to an OpenGL identifier object (internally just an integer). This mapping is three-dimensional and of the structure:

```
GLuint DisplayList = GLCache[ Intercube ] [ typeid ] [ ValueSet ];
```

Hereby, a given `GLCache` object, a DisplayList identifier can be retrieved by specifying

1. an arbitrary Intercube object
2. an intrinsic C++ type ID
3. a set of values

The functionality is similar to the OperatorCache, where an InterCube and a visualization node is utilized to specify a location of the OperatorCache, plus a set of values used to determine whether re-creation of the data is required. Here, the storage location of the cached objects is provided by the `GLCache`, a parameter that is provided to a visualization object's render routine. The InterCube object (which, for instance, is available with each Grid or Field object within a fiber bundle data-set) is used to find a unique storage location within the `GLCache`. The `typeid` will be the type of the rendering object, such that multiple instances of the same rendering functionality will automatically be able to share their OpenGL objects. The set of values will contain all those rendering parameters which require re-creation of the OpenGL object. A typical rendering code will (schematically) look like this:

```
void VizNode::render(GLCache Context, Grid G)
{
ValueSet VS;
    // assign cacheable variables into the value set
InterCube&C = G;
GLuint DisplayList = GLCache[ Intercube ] [ typeid(this) ] [ VS ];
if (!DisplayList)
   {
       DisplayList = glGenLists(1);
       glNewList(DisplayList, COMPILE_AND_EXEC);
       // do actual rendering of grid data G
       glEndList();
       GLCache[ Intercube ] [ typeid(this) ] [ VS ] = DisplayList;
   }
   else
       glCallList(DisplayList);
}
```

A similar synopsis will be applied for OpenGL object types others than display lists. Some automatic discarding mechanism to ditch unused objects will be required here as well. Note that in case an OpenGL object already exists for a given input data-set (here a "`Grid`" object), then there is no need to actually request the internal data of such an object and to traverse the associated visualization pipeline up to its source, such as shown in Figure 10.

3 Conclusion

An universal caching mechanism has been presented. It can be used to extend the visualization pipeline model (as defined by [Haber & McNabb, 1990]) to maximize the reuse of computed data and thereby minimizing the response time of an interactive visualization to parameter changes. The mechanism can relate computed data for all parameters of the visualization, which make fast and easy navigation in the whole parameter space possible. Special emphasis

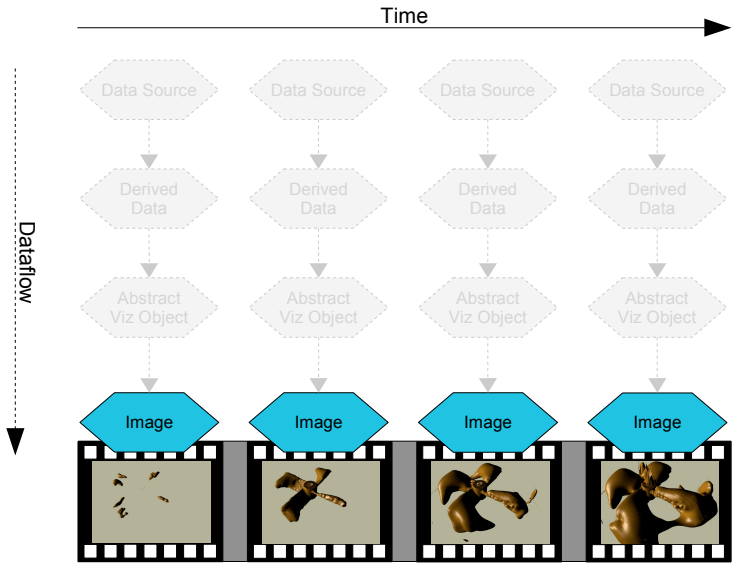


Figure 10: *A GPU cached visualization cascade provides the animation without expensive data flow.*

is given to the time parameter and time depended data. The implementation demonstration utilizes the Vish framework and also the fiber-bundle model. By incorporating the described GPU caching mechanism full interactivity when browsing an astrophysical data set of 17GB - containing 1.6 million points in 200 time steps - has been achieved. This visualization was run on a Linux 64 bit workstation equipped with a eight 1.6GHz cores (only one of which was used by Vish), 8 GB of RAM and a Geforce Quattro FX5600 graphics card with 1.5GB of GPU memory. The caching mechanism accelerated the visualization to a achieve interactive rates of 30 frames per second when traversing in time in addition to arbitrary spatial camera movement. The first access of the data including reading from disk and processing data in contrast required a couple of seconds for each newly accessed time step. It was also possible to maintain the interactive frame rate while changing parameters such as color-maps and density shape-functions used for volume rendering.

4 Acknowledgements

This cooperation research work was supported by the DFG (SCHO 1346/1-1). This research employed resources of the Center for Computation & Technology at Louisiana State University, which is supported by funding from the Louisiana legislature's Information Technology Initiative. Portions of this work were supported by NSF/EPSCoR Award No. EPS-0701491 (CyberTools).

References

[A.A. Ahmed, 2007] A.A. Ahmed, e. a. (2007). Automatic visualization pipeline formation for medical datasets on grid computing environment.

[Benger, 2004] Benger, W. (2004). *Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model*. PhD thesis, FU Berlin.

[Benger, 2008] Benger, W. (2008). Colliding galaxies, rotating neutron stars and merging black holes - visualising high dimensional data sets on arbitrary meshes. *New Journal of Physics*, 10. URL: `http://stacks.iop.org/1367-2630/10/125004`.

[Benger et al., 2007] Benger, W., Ritter, G., & Heinzl, R. (2007). The concepts of vish. In 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007* (pp. 26–39).: Berlin, Lehmanns Media-LOB.de.

[Benger et al., 2009] Benger, W., Ritter, M., Acharya, S., Roy, S., & Jijao, F. (2009). Fiberbundle-based visualization of a stir tank fluid. In *WSCG 2009, Plzen*.

[Benger et al., 2006] Benger, W., Venkataraman, S., Long, A., Allen, G., Beck, S. D., Brodowicz, M., MacLaren, J., & Seidel, E. (2006). Visualizing katrina - merging computer simulations with observations. In *WORKSHOP ON STATE-OF-THE-ART IN SCIENTIFIC AND PARALLEL COMPUTING Umeå, Sweden, June 18-21, 2006* (pp. 340–350).: Lecture Notes in Computer Science (LNCS), Springer Verlag.

[Bischof et al., 2006] Bischof, H.-P., Dale, E., & Peterson, T. (2006). Spiegel - a visualization framework for large and small scale systems. In *MSV* (pp. 199–205).

[Black et al., 2002] Black, A. P., Huang, J., Koster, R., Walpole, J., & Pu, C. (2002). Infopipes: an abstraction for multimedia streaming. *Multimedia Syst.*, 8(5), 406–419.

[Card et al., 1999] Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). Using vision to think. (pp. 579–581).

[Foulser, 1995] Foulser, D. (1995). Iris explorer: a framework for investigation. *SIGGRAPH Comput. Graph.*, 29(2), 13–16.

[Haber & McNabb, 1990] Haber, R. & McNabb, D. A. (1990). Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*.

[Haeberli, 1988] Haeberli, P. E. (1988). Conman: a visual programming language for interactive graphics. *SIGGRAPH Comput. Graph.*, 22(4), 103–111.

[Hibbard, 1999] Hibbard, B. (1999). Top ten visualization problems. *SIGGRAPH Comput. Graph.*, 33(2), 21–22.

[Johnson & Jennings, 2001] Johnson, G. W. & Jennings, R. (2001). *LabVIEW Graphical Programming*. McGraw-Hill Professional.

[Schroeder et al., 1997] Schroeder, W., Martin, K., & Lorensen, B. (1997). *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Prentice Hall.

[Shen, 2006] Shen, H. (2006). Visualization of large scale time-varying scientific data. *J. of Physics: Conf. Series*, 46, 535–544.

[Treinish, 1997] Treinish, L. A. (1997). Data explorer data model. `http://www.research.ibm.com/people/l/lloydt/dm/dx/dx_dm.htm`.

[Upson et al., 1989] Upson, C., Thomas Faulhaber, J., Kamins, D., Laidlaw, D., Schlegel, D., Vroom, J., Gurwitz, R., & van Dam, A. (1989). The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4), 30–42.

Visualizing Quarks

Massimo Di Pierro
School of Computing, Telecommunications and Information Systems
DePaul University, 243 S Wabash, Chicago, Illinois, USA
mdipierro@cs.depaul.edu

ABSTRACT

Quantum Chromo Dynamics (QCD) is the mathematical model that best describes quarks, their interactions, and how they give rise to the complexity of known matter (including the proton and the neutron). QCD is a relativistic four-dimensional model based on Quantum Mechanics. It is characterized by a highly non-linear behavior and its output consists mainly of correlation functions among quantum fields. In this paper we describe a numerical and algorithmic formulation of QCD and we present a visualization toolkit designed to compute those correlation functions, project them into three dimensional fields, and visualize them. The first two steps (the computation and the projection) tend to be very computationally expensive, hence they are usually carried on on a high-end parallel computing infrastructure, typically a cluster of hundreds of nodes. We created a software library that implements the relevant algorithms and which generates VTK files for either real time or off-line visualization. In this paper we present a short introduction to lattice QCD, its challenges and its goals; we provide a description of the workflow of the toolkit and the tools that comprise it; finally, we show some sample images generated with our toolkit. We believe that this visualization approach will help improve our understanding of lattice QCD and help create better numerical algorithms to simulate it.

1. INTRODUCTION

The Standard Model (SM) of Particle Physics[1] represents the *state of the art* in High Energy Physics. It can successfully explain the structure of matter at scales of $10^{-18}m$ and above, where gravity is weak and its effects are negligible. The SM model is a Relativistic Quantum Field Theory and it provides a framework to compute correlation functions between observable quantities. The degrees of freedom of the model are six quarks, six leptons (the electron, the muon, the tau and three neutrinos), one Higgs boson and the, so called, gauge fields. The dynamics of the system is described by a Lagrangian. Those terms in the Lagrangian that describe the quarks and their interactions are referred to as Quantum Chromo Dynamics[2] (QCD).

QCD is the most interesting part of the Standard Model because is it responsible of the formation of a plethora of composite particles (including the proton and the neutron) and most of the complexity that we observe in the physical world. This complexity originates by non-linear interactions among quarks and its effects at large distances are not fully understood. Luckily, QCD can be approximated numerically [3] and the correlation functions can be computed on a lattice, *i.e.* a discretized portion of four dimensional spacetime.

Relativistic Quantum Field Theories constitute a generalizaiton of Quantum Mechanics that include Special Relativity and are compatible with causality principles. This is true for the Standard Model and QCD as well. The Poincare group is broken by the discretization which is done in a specific reference frame. The symmerty group is restored in the continuum limit.

We have developed a general purpose library for implementing parallel lattice QCD computations. It is called FermiQCD[5]. This library includes a visualization toolkit, designed to extract visual information from the computation. In many current lattice QCD computations, most of the spatial information that could be visualized is instead discarded.

A typical lattice QCD computation consists of about 1000 lattices of up to 128×96^3 sites each and each lattice site storing 72 double precision floating point numbers. This requires hundreds of Gigabytes of Ram memory, hundreds of Terabytes of disk space and computing resources in excess of one million hours of an ordinary modern PC. For this reason typical lattice QCD computations run on dedicated parallel machines such as the QCDOC [7] and computer clusters [8].

Every computation can be reduced to a multi-dimensional integral where the dimensionality of the integration domain tends to infinity. This limit is computed numerically. For every finite dimensionality of the integration domain the integral is computed using a Markov Chain Monte Carlo (MCMC) method. The Monte Carlo samples are discretized states of the four-dimensional fields of QCD (of the order of 10^9 floating numbers each) and they evolve according to the MCMC. The output correlation functions are functions of the fields averaged over these Monte Carlo steps.

Some of the correlation functions that we are interested in visualizing have an immediate interpretation in terms of three dimensional objects in the ordinary space (for example the wave function of composite quark-antiquark objects called mesons, fig. 3, and the average energy density in the space around a static quark-antiquark pair, fig. 2), while others do not but, they can teach us about QCD itself and about the algorithms we use to approximate it (for example the evolution of the local topological charge of the gauge field as it evolves under the MCMC evolution, fig. 1.).

For example, one open issue in lattice QCD is that while the individual MCMC steps are local, the topological charge of the fields appears to evolve in a non-local way, i.e. while it is conserved globally, it may appear in one place and disappear in another place without showing a movement from one region to another. This is one of the issues that eventually we hope to understand thanks to our toolkit.

A MCMC step normally also comprises the inversion of very large sparse matrices. This is done using some variation of the the Minimal Residue or of the Conjugate Gradient algorithms. By visualizing the residue as it evolves during the inversion steps, we may be able to understand better and eventually cure pathologies of these algorithms. Perhaps we will be able to derive better and faster inversion algorithms.

In section two we will briefly describe some of the algorithms that we utilize in the computations, the workflow of these computations, and some implementation issues. In section three we present and discuss some results.

Our toolkit generates VTK [4] files and we visualized them offline using Visit [10] and/or MayaVi [11] which is based on the Enthought workbench [12].

The images presented here should be interpreted mainly as proof-of-concepts because, even if they are accurate, they are computed using very small lattice sizes and relatively small computing resources.

2. FERMIQCD AND THE VISUALIZATION TOOLKIT

Our toolkit is designed to interoperate with various standard lattice QCD software packages (Chroma, MILC[9], FermiQCD[5]) but, internally it is based on FermiQCD, which is, despite its name, a library for general purpose distributed computations. The reason for the name is that it is primarily employed in lattice QCD computations. We extended this library to include a toolkit that can generate VTK files in parallel, while the computation is carried on.

The FermiQCD library consists of the following classes:

- `mdp_site`. A site is a vertex of the lattice. In the case of QCD a vertex of a four dimensional lattice. In a parallel architecture, FermiQCD automatically maps each site to a computing node and identifies optimal communication patterns in oder to minimize communication latency.
- `mdp_lattice`. A lattice is a set of sites with their physical topology (notion of distance between the sites). In the case of QCD, a lattice has a four-dimensional torus topology.
- `mdp_field<T>`. A field is a data structure that is comprised of the data structure, the template `T`, replicated for each site of the lattice.

  For example, `mdp_field<float>` consists of one floating point for each lattice site. In the case of QCD there are multiple fields and they belong to the following categories: gauge fields (`gauge_field`) and quark fields (`fermi_field`).
- `fermi_field`. This is a 4×3 matrix of complex numbers for each lattice site. Each `fermi_field` represents a quark flavor (u,d,c,s,b, and t), its 3-values index represents the "color" quantum number, while the 4-values index represents the four spin components.
- `gauge_field`. This is a 4-vector of 3×3 unitary matrices of complex numbers for each lattice site. The matrices belong to the $SU(3)$ group. Given a gauge field `U`, we represent with `U(x,mu)` the matrix at the lattice site `x`, component `mu` of the 4-vector. Each $SU(3)$ matrix represents the phase acquired by a `fermi_field` when shifted by one lattice unit in the direction of the 4-vector `mu`.

In a parallel architecture, FermiQCD distributes the lattice sites over the available computing nodes and handles the replication of the field variables using buffers so that every node has read access to the field variables of the neighbors of the local sites. FermiQCD also handles transparently the required communications to synchronize replicas of the fields variables with non-local sites. FermiQCD requires only GCC and a version of MPI.

FermiQCD stores all copies of site variables received from the same computing node close in memory so that the update of those sites can be done in a single `send`/`recv` for each couple of nodes. Nodes communicate only when fields need to be updated and only if they share some sites (when one of them keeps a copy of the sites that are local to the other). FermiQCD is optimized to work on clusters where each computing node has a single network interface. Hence it organizes all communications so that each node is involved in no more than one `send` and one `recv` at the time. This behaviour minimizes network latency and it is optimal for a switched network (assuming the switch can handle the total network bandwidth).

Also notice that MPI `send` and `recv` functions never need to be called explicitly. FermiQCD exposes an additional abstraction layer on top of MPI that allows explicit point-to-point communication of any C++ object and allows global reduce operations.

FermiQCD also provides a `forallsites(x) {....}` iteration that executes the code in `{...}` in parallel for each lattice site and handles all required communications. For example in the following statements:

```
forallsites(x) phi(x)=0;
```

every computing node loops over the lattice sites that are stored there, `x` is the looping variable, and sets the field variable of field `phi` at site `x` to zero. Effectively all the field variables are set to zero in parallel.

Field objects have the following methods:

- `update`. It performs required communications to synchronize all buffers of the field. It should be called, inside each `forallsites` iteration, by those fields that have been modified in the iteration.
- `save`. It saves the field on disk. All field variables are saved in an order that is specified by the lattice topology and independent of the parallel partitioning on the lattice. The method `save`. performs all required communications to send the file to a single node which, in turn, performs the Input/Output.
- `load`. This method performs the opposite function of `save` It reads a FermiQCD field from disk and scatters its site variables across the parallel architecture.
- `save_vtk`. This method is similar to `save` but it is designed to work specifically for scalar fields and saves the field in the VTK format (ASCII or BINARY). For fields in 4 dimensions or more, it slices the lattice and saves all the 3D slices in one VTK file, in parallel.

As an example here is a minimalist but complete FermiQCD program that solves the Laplace equation:

$$(\partial_0^2+\partial_1^1+\partial_2^2)\phi(\mathbf{x})=\rho(\mathbf{x}) \qquad with \qquad \mathbf{x}=(x_0,x_1,x_2) \quad (1)$$

in 3 dimensions ($10 \times 10 \times 10$), iteratively, and, for each iteration, it generates a VTK file.

```
#include "fermiqcd.h"
int main(int argc, char** argv) {
  mdp.open_wormholes(argc,argv);
  int L[3]={10,10,10};
  mdp_lattice cube(3,L);
  mdp_site x(cube);
  mdp_field<float> rho(cube);
  mdp_field<float> phi(cube);
  rho.load('rho_input.mdp')
  phi=0;
  for(int i=0; i<100; i++) {
    forallsites(x)
      phi(x)=1.0/6.0*(rho(x)+phi(x+0)+phi(x-0)+
         phi(x+1)+phi(x-1)+phi(x-2)+phi(x+2));
    phi.update()
    phi.save_vtk("image.*.vtk");
  }
  mdp.close_wormholes();
  return 0;
}
```

In the code, the statements `open_wormholes` starts communications among the parallel nodes and `close_wormholes` stops them; `cube` is the 3D $10 \times 10 \times 10$ lattice object; `rho` is the input scalar field (for example a distribution of electrical charge); `phi` is the output scalar field (for example the electromagnetic potential). The line `phi(x)=....` is the discretized expression for the Laplace equation.

Any differential equation can be solved using FermiQCD by replacing the expression for the Laplace equation with a different one. Notice that though is a parallel program based on MPI, there is not explicit call to the `send` and `recv` functions. The only communication function that is called is `update`, to notify FermiQCD that the value of a field has changed and buffers have to be synchronized.

Notice that if `x` is a lattice size, `x+0` is the site neighbor up in direction 0, and `x-0` is the site down in direction 0, etc. (notice that `x+0` is in general distinct from `x-0`). If `x` is a lattice site stored locally by the computing node, its close neighbors may or may not be local sites therefore they cannot be modified but, they can always be accessed because FermiQCD keeps a copy of them in a buffer. We refer to the closest neighbors as 1-hop neighbors, next-to-next neighbors as 2-hop, etc. FermiQCD allows to set the size of the buffer and thus set how many hops one can move away from each site before failing to find a copy of the requested site.

Many typical lattice QCD computations, when implemented in FermiQCD, are very similar to the above program except that `cube` is replaced by 4D lattice typically of the order $96 \times 64 \times 64 \times 64$, the `spacetime`; the float fields are replaced by `gauge_field`s and `fermi_field`s; the Laplace equation is replaced by a much more complex algorithm that encodes the dynamics of the system, for example the Wilson Heatbath; the VTK files are generated from a scalar field that is constructed by a contraction of the other fields. The expression for the contraction determines what is being measured on the system.

Below is an example of a program that loops over 100 MCMC steps. It starts with a `gauge_field U` that is half cold (matrices are set to identity) and half hot (matrices are set to random) and evolves under the Wilson Heatbath algorithm (which implements a discretized version of the QCD Lagrangian with the additional simplification that second quantization for quarks is switched off). For each step the `gauge_field` is copied (`V=U`), smoothed out by a local iterative algorithm called `ApeSmearing`, then used to compute the topological charge, and the latter is saved as VTK files.

```
#include "fermiqcd.h"

int main(int argc, char** argv) {
   mdp.open_wormholes(argc,argv);
   int L[]={16,4,4,4};
   mdp_lattice spacetime(4,L);
   coefficients coeff;
   int nc=3;
   coeff["beta"]=5.0;
   gauge_field U(spacetime,nc);
   gauge_field V(spacetime,nc);
   forallsites(x)
       for(mu=0;mu<4;mu++)
           U(x,mu)=(x(0)<8)?identity(nc):
              spacetime.random(x).SU(3);
   U.update()
   for(int i=0; i<100; i++) {
      WilsonGaugeAction::heatbath(U,coeff,10);
      V=U;
      ApeSmearing::smear(V,0.7,1,10);
      topological_charge_vtk(V,"topcharge.*.vtk");
   }
   mdp.close_wormholes();
```

```
   return 0;
}
```

The output of this algorithm for three different iteration steps is shown in fig. 1. The figure represents the topological charge of the vacuum of QCD as it evolves under MCMC. The initial data-set was engineered so that half of the lattice has no charge and the other half has random fluctuations of the charge. Even if the MCMC algorithm is local, we see instantons (lumps of topological charge) appear in the empty region but we do not observe a continuous drift of the topological charge from the random region. This phenomenon is not very well understood.

Note that FermiQCD provides a local random number generator, *i.e.* a random number generator at each point of the lattice, `spacetime.random(x)` and this is necessary to guarantee reproducibility of results even when the random number generator is used in a parallel environment. The random number generator class includes dedicated algorithms like the Cabibbo-Marinari to generate random $SU(N)$ matrices, uniformly within the group.

In this example the definition of topological charge is beyond the scope of this paper but we try to explain it with an analogy. Let's imagine a net made with strings. When two strings cross each other there is a knot. The knots are not all the same and we can characterize the knot at each crossing by its "winding number", *i.e.* by how many times one string winds around the other string. This net looks a lot like a lattice and at each vertex (when two strings cross) there is an integer (the winding number). A gauge `gauge_field` in QCD is very much like a net in 4D but the strings are replaced by matrices and the winding number is replaced by a function of the products of $SU(3)$ matrices, the topological charge.

3. PRELIMINARY RESULTS

Now we immediately see the main challenge of our visualization project. The topological charge can be defined and computed (fig. 1) within the model and it plays a very important role (not yet fully understood) in QCD, but it does not correspond to a physical object that can be observed. Nevertheless it has profund implications on the physical properties of QCD.

The vacuum of QCD is believed to behave as a dual superconductor. In an ordinary super-conducting material, there is no resistance to the flow of electrical current. Therefore if one places a piece of material inside a magnetic field, the material responds by generating currents that screen the magnetic field, and the magnetic field is squeezed into flux tubes that go across the material. This effect is observed in superconducting materials. The vacuum of QCD is believed to behave as the dual of this. It opposes no resistance to the flow of chromo-magnetic currents thus it can screen chromo-electric fields and squeezes it into flux tubes. This effect is a consequence of the topological property of the vacuum of QCD. It can be computed and it is visible in the third image in figure 2 for three different MCMC steps.

The figure shows the average chromo-electro-magnetic-energy of the vacuum in the space around a pair of static quark-antiquark as more and more Markov Chain steps are computed and results are averaged. The quark-antiquark particles are not shown but they are located at the edges of the ellipsoid. We observe that as more MCMC steps are performed the noise due to quantum fluctuations dissipates and the energy density gets localized in an ellipsoid with vertices corresponding to the locations of the quarks-antiquark particles. This ellipsoid is an instance of the flux tube mentioned above. It represents the vacuum's response to the chromo-electric field originating in the quarks.

The practical consequence of this effect is that the potential energy between the quark-antiquark pair grows with distance, the opposite of what happens in electromagnetism. This phenomenon is called confinement because it makes it impossible to isolate individual quarks and leads to the formation of composite particles made of quarks such as the *mesons* and the *baryons*.

From an experimental point of view it is a fact that we have never been able to observe an isolated quark but we have observed many bound states of a quark-antiqurk (the mesons) and bound states of three quarks (the hadrons, for example the proton and the neutron). We have also observed how the breaking of one bound state leads to formation of new bound states in a way consistent with the predictions of QCD.

In fact, even if we cannot observe individual quarks, we know that mesons are made of quarks and we know how they interact. Some mesons are particularly interesting because we can draw analogies with similar and better understood systems. Let's consider a hydrogen atom. It is comprised of a heavy proton at the center, a light electron bound to the proton by the electrical field. The state of the electron can be described by a wave function and the modulus squared of this wave function is normally referred to, in chemistry, as the "orbital" of the atom.

Similarly, we can build a Heavy-Light meson that is comprised by a heavy b quark and a light u antiquark, bound to the quark by the chromo-electrical field, and determine its "orbital".

The different images in figure 3 represent "orbitals" for two possible Heavy-Light mesons, a B and a B^* meson respectively.

While these particles are observed experimentally, their internal structures can only be determined from the model, for example in computations like ours [6].

Although in the physical world quarks do not exist as isolated particles, in our computations we can engineer a state that consists of a single quark in the vacuum and measure its wave function. Two components of this wave function can be seen in fig. 4.

From a computational point of view, a major complexity here is that there are many ways to discretize a quark on the lattice and thus to implement the Dirac equation. While these different methodologies should lead to the same physical results, this is not always the case, because all techniques

that approximate continuum space-time with discrete space-time suffer from one pathology: They break some symmetry of the continuum Lagrangian of QCD. The effects of symmetry breaking can usually be quantified and controlled, and we believe our toolkit can be helpful to study them. Fig.5, for example, shows a solution of the Dirac equation for an isolated quark in the vacuum for a staggered quark, *i.e.* an alternative method to distribute the degrees of freedom of a quark on the lattice.

4. CONCLUSIONS

The goal of our project is to develop a visualization toolkit to help physicists visualize information that is generated in lattice QCD computations. We believe visualization techniques can lead to a better understanding of QCD itself and better simulation algorithms. Least but not last our toolkit can be used as a training tool to give life to some of the equations normally seen in QCD textbooks.

Our toolkit is part of the FermiQCD library, also developed by the author, and is already very powerful. It includes many algorithms normally used in lattice QCD computations and has the ability to project fields into standard VTK files which can be visualized in real-time and off-line. In this paper we have shown some results for the topological charge of QCD, the energy density in presence of a quark-antiquark couple, the wave function of an antiquark in a Heavy-Light meson and a solution of the Dirac equation for a static quark.

5. ACKNOWLEDGMENTS

This project is supported by the Department of Energy under the Scientific Discovery Through Advanced Computing grant DE-FC02-06ER41441. We wish to thank the Fermilab theory group for continuing to share their data and their computing resources with the author, and the USQCD collaboration[13].

6. REFERENCES

[1] W. N. Cottingham, D. A. Greenwood, *An Introduction to the Standard Model of Particle Physics*, Cambridge University Press, 2007

[2] W. Marciano and H. Pagels, *Physics Reports* **36** (1978) pp. 135

[3] M. Di Pierro, *International Journal of Modern Physics A* **21,3** (2006)

[4] W.J. Schroeder *et al.*, *IEEE - Computer Graphics and App lications* **20** (2000) pp. 20-27

[5] M. Di Pierro, *Nuclear Physics B - Proc. Suppl.* **106-107** (2002) pp.1034-1036

[6] A. Kronfeld, *Journal of Physics* **46** (2006) pp. 147-151

[7] S. Gottilieb, *IEEE: Computing in Science and Engineering* **8** (2006) pp.15

[8] D. J. Holmgren, *Nuclear Physics B - Proc. Suppl.* **140** (2005) pp.183-189

[9] S. Gottlieb, *Nuclear Physics B - Proc. Suppl.* **106-107** (2002) pp.1031-1033

[10] https://wci.llnl.gov/codes/visit/

[11] P. Ramachandran, 4th Annual Symposium, Aeronautical Society of India, 2001.

[12] `http://www.enthought.com`

[13] R. Brower, *Journal of Physics - Conf. Series* pp.142

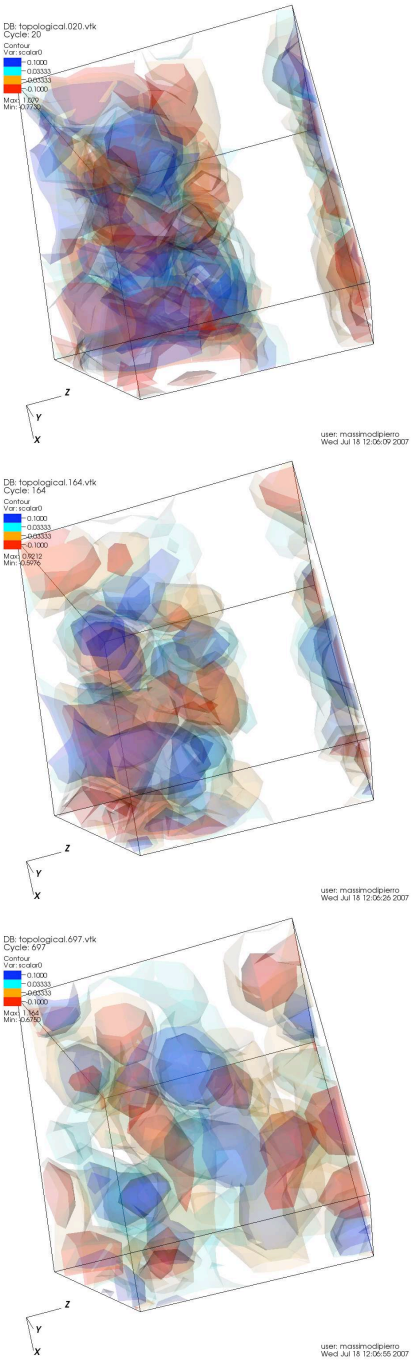


Figure 1: The figure represents the topological charge of the vacuum of QCD as it evolves under MCMC. Is it a sequence of snapshots which from top to bottom has been evolved as more MCMC points have been averaged. The initial data-set was engineered so that half of the lattice would have no charge and the other half would have random fluctuations of the charge. Even if the MCMC algorithm is local, we see instantons (lumps of topological charge) appear in the empty region but we do not observe a continuous drift of the topological charge from the random region. This phenomenon is not very well understood.

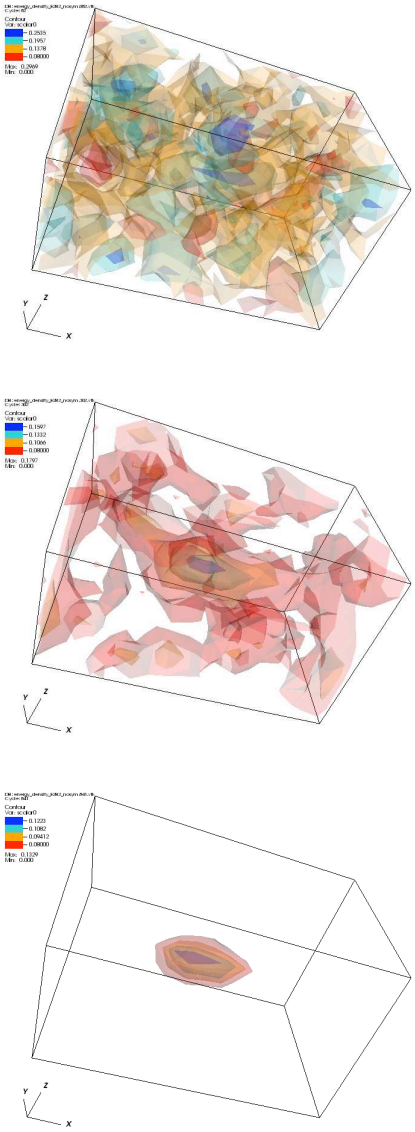


Figure 2: The figure shows the average chromo-electro-magnetic-energy of the vacuum in the space around a pair of static quark-antiquark as more and more Markov Chain steps are computed. The quark-antiquark particles are not shown but they are located at the edges of the ellipsoid. We observe that as more steps are performed the noise due to quantum fluctuations dissipates and the energy density gets localized in an ellipsoid with vertices corresponding to the locations of the quark-antiquark particles. This indicates that the potential energy between the two particles grows with their distance, the opposite of electromagnetism, and this phenomenon, called confinement, leads to the formation of composite particles such as mesons and baryons, including protons and neutrons.

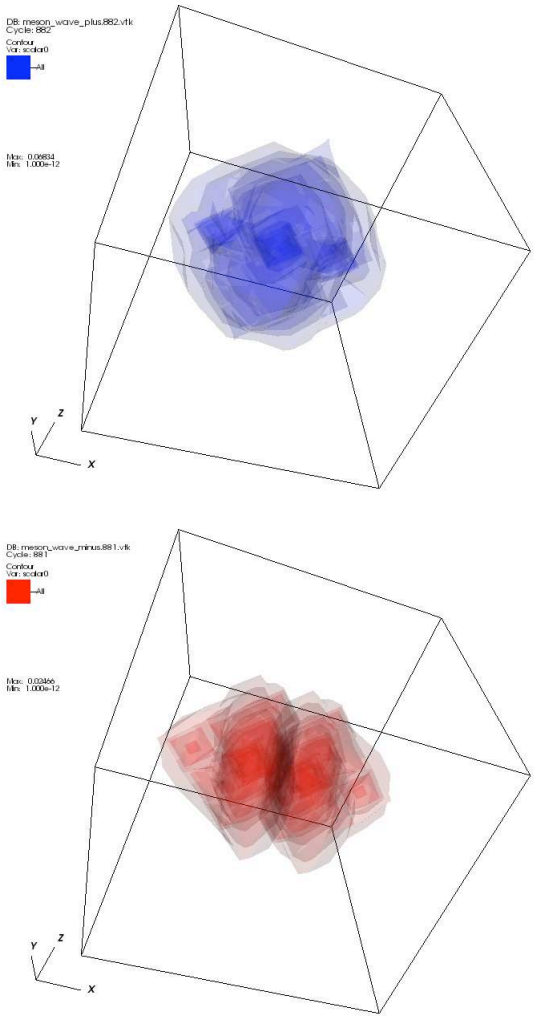


Figure 3: In analogy with the "orbitals" of the Hydrogen atom well known to chemists, we have computed the wave function of an antiquark in a heavy meson (a bound state made of a heavy quark and a light antiquark). The different images represent wave functions for different excitation levels. They correspond to a B and a B^* meson respectively. While these particles are observed experimentally, their internal structures can only be computed numerically, for example in computations like ours.

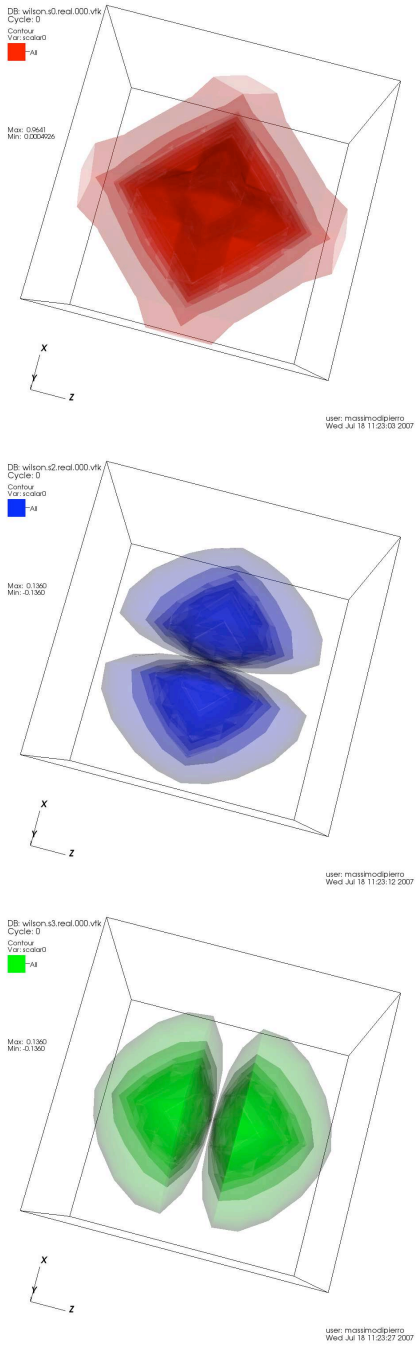


Figure 4: The image shows different field components of the solution of the Dirac equation for a quark in the vacuum. Solving the Dirac equation is an important part of the MCMC step of lattice QCD.

Figure 5: The image shows different field components of the solution of the Dirac equation for a quark in the vacuum for an alternative discretization technique for the quark. Different discretizations techniques correspond to different approximations for the continuum space-time. The discretization process has the effect of breaking some of the symmetries of the QCD Langrangian. We hope that visualization techniques can help understand and eliminate any possible bias introduced by the discretization.

Envisioning a Standard Image Storage Framework

Matthew Dougherty
National Center for Macromolecular Imaging
Baylor College of Medicine, Houston Texas, USA
matthewd@bcm.edu

Abstract

The evolution of images has led to imagery having a digital manifestation. Unlike previous forms of imagery, digital images are ephemeral and inherently numeric; these images are directly accessible only through computers and as such they are capable of representing abstract and physical modalities beyond two-dimensions. As science embraces such tenuous datasets for purposes of acquisition, visualization, and archiving, it is in the general interest to have enduring image standards serving future generations. Proposed are requirements for a standard scientific image storage framework, and an outline of computational and social methods to achieve formal standardization.

1 Introduction

"If I have seen further, it is by standing upon the shoulders of giants." — Isaac Newton

The last half-century has resulted in digital imagery coming to the fore. Inherently coupled to computers, this form of imaging is transformative. And unlike previous forms of imagery it is not tied to conventional thinking as to what images are.

Frequently in the history of science, images have spoken for themselves by providing the conclusive proof that dispels lingering doubt; much as in judicial proceedings, recorded images are given evidentiary preference and have significant archival value.

Science, medicine, and engineering have adopted recorded images as coin of the realm in many observations.

Just as in the invention of money the standardization of currency enabled society to progress in many unpredictable ways. It is suggested that the standardization of scientific digital images, like the standardization of numbers, will have major implications in science and society. [ICSU, 2004] [ICSTI, 2004]

What is needed is the establishment of a formal standard for an n-dimensional image storage framework that is capable of generating archival-ready datasets for the multitude of scientific research communities. [Edwards, 2004] [NCRR, 2002]

Described are **1)** recommendations for the development of such a standard, **2)** a computational solution to achieve transparent interoperability of existing scientific image formats through a high-performance computational image model, and **3)** a method for formal standardization.

2 Brief History of Images

"When it comes to atoms, language can be used only as in poetry. The poet, too, is not nearly so concerned with describing facts as with creating images." — Niels Bohr

Imagery can be described as the first written language of humanity. This is evidenced by the stories presented through ancient cave drawings in Australia and Europe. But are images inherently two-dimensional?

Many of the notes of Leonardo da Vinci go to great detail explaining the transformation of four dimensions (three spatial, one temporal) to two spatial dimensions: the selection, modeling and re-creation of an event that accurately portrays the timelessness and subtlety of the moment in a painting.

The invention of chemical photography in 1825 unleashed a torrent of scientific imaging that has been used to settle bets, penetrate the behavior of subatomic particles, and fathom the cosmos. The invention of motion pictures allowed two-dimensional images, all having realistic three-dimensional perspective, to animate continuously in time.

In the last century, a variety of image standards have been adopted by various industries to form immutable elements in their infrastructure; the effects of such adoption of standards have generally been beneficial. The 35mm film format used by the motion picture industry and the DICOM standard used in medical imaging are but two examples.

The invention of electronic analog imaging has led to the economic dissemination of "live" imagery ubiquitously to the general population. This is demonstrated by the pervasiveness of television, which itself is now undergoing its own conversion to digital imagery. The formative years of television were plagued by problems with competing standards, sometimes for reasons of technical superiority, but more often for reasons of controlling market share; we saw this again with the VCR and DVD media wars. These lessons were not lost in the design and conversion to digital High Definition TV, problems circumvented by the establishment of a formal global standard for digital video.

For practical reasons prior to modern imagery, images were treated as two-dimensional, sculptures as three-dimensional, and in the theater stories were acted out in four dimensions. With digital images, the paradigm changes dramatically because of their scalability and correspondence to mathematical abstraction. For in this paradigm, images are numbers, instantiations to visualizations, instead of being merely terminal visual manifestations.

3 The Standardization of Numbers

"It is India that gave us the ingenious method of expressing all numbers by means of ten symbols, each symbol receiving a value of position as well as an absolute value; a profound and important idea which appears so simple to us now that we ignore its true merit. But its very simplicity and the great ease which it has lent to computations put our arithmetic in the first rank of useful inventions; and we shall appreciate the grandeur of the achievement the more when we remember that it escaped the genius of Archimedes and Apollonius, two of the greatest men produced by antiquity." — Pierre-Simon de Laplace

There was a time when the state of numbers was much like the current state of scientific image formats: varied, incompatible, imperfect, and suitable mainly for parochial applications. The history of numerical notation is a fascinating subject because it is the story of the first precise and unambiguous universal human language. It is an efficient intellectual abstraction based in numerical mathematics.

Through the conjunction of four great ideas (radix symbols, numeric sequence position, fractional notation, and the conceptualization of zero), a scalable and simple method to describe and compute numbers through a standard positional decimal-place notation has profoundly changed the course of science and human history — annotation capable of

revealing fantastic numbers that touch the edge of the physical universe, numbers such as $6.62606896 \times 10^{-34}$.

We take our numbering system for granted because of its simplicity and frequently fail to realize that it did not mature into its current form until the last 400 years, a chaotic global standardization process that took humanity four millennia to achieve. Efficient and scalable mathematical operations are possible because of this system's subtle and parsimonious design; compare this to computation using Inca quipus, Roman numerals, or the Mayan mixed radix system. [Ifrah, 1999]

4 The State of Digital Image Formats

"The images offer a way of translating among disciplines, of bridging the gap" — Rita Colwell

Over the last forty years there have been myriad of digital imaging file formats that roughly fall into two categories. But regardless of the category, digital images can usually be segmented into pixel and metadata subsets. [MacTavish, 1999] [CENDI, 2007]

In the first category are generic formats such as the two-dimensional spatial image formats TIFF, JPEG, and PNG, and the three-dimensional MPEG format. All have the primary advantages in that they are commonly used by many computer software programs and are formally standardized. To varying extents, these generic formats allow users to append their metadata to the image pixels; it is to be noted, however, that extensible dimensions and modalities are inherently constrained. Compression is also a distinguishing feature of these formats; in the case of MPEG, for example, it is a lossy video compression, which is suitable for many but not all scientific applications.

In the second category, image formats are customized file designs controlled by specific communities, and usually driven by specific metadata requirements, higher image dimensions, or complex pixel modality. Frequently the scientific, medical and engineering communities prefer these customized formats because total control of all aspects of the image design can be maintained and changed by the community. Examples of this include the DICOM medical imaging format, the SEGY geophysical image format, the astronomical FITS format, and the MRC electron microscopy format.

Two significant problems in the foundations of existing image formats must be resolved: the management of pixels and the inclusion of metadata from differing communities.

During the last forty years, image sizes have grown dramatically: the Panoramic Survey Telescope and Rapid Response System utilizes 1.4 giga-pixel cameras; the field of clinical microscopy is planning for the routine production scanning of specimens at greater than 64k x 64k pixels; and the Earth Observation System consists of 24 instrument sets generating 1,500 tera-bytes per year. Pixel management is a common problem across scientific imaging communities because most of the formats have inefficient pixel designs that use two-dimensional thinking to grapple with n-dimensional problems, which is barely tolerable when the entire image can be read into the random-access-memory of the computer. Generally, most image formats have this impending weakness, an Achilles' heel when images exceed something on the order of 1-10 gigabytes in size. [Gray, 2005]

Amalgamation and encapsulation of community metadata is another major design problem. Although generic image formats allow for the inclusion of multi-user metadata passed as text strings, none of the image formats allow for the inclusion of substantially larger metadata such as files or multi-community metadata. As would be expected, as an image progresses from acquisition, signal processing, visualization, analysis, animation, and

deposition, there is the need for many different communities to generate their own independent metadata, typically in proprietary file formats. For example, many software programs generate their metadata in the form of XML files. Bundling this metadata in context to the pixels will be crucial for maintaining archival-ready image datasets, because if the context is lost the quality of data is degraded. [Marciano, 2006] [Tanner, S., 2006] Also, while metadata frequently represents less than 1% of the image file size, it is usually the primary reason for image format conversions. If different communities' metadata cannot be integrated to co-exist without conflict, it is inevitable that complete pixel duplications are required in order to move pixels forward in the research pipeline. Maintaining all of these duplications is uneconomical and untenable for researchers and archivists.

5 Requirements for a Standard Scientific Image Storage Framework

"Things should be made as simple as possible, but not simpler." — Albert Einstein

Envisioning a standard scientific image storage framework brings to mind these requirements:

- **The scientific communities must be allowed to continually evolve their own image designs.**

  This is particularly important in the matter of metadata, for it is this metadata that truly distinguishes these communities and defines their growing sophistication in their respective fields of research and tool development. Four excellent examples developed by strong community efforts are DICOM used in clinical imaging, NeXus used by the beamlines, netCDF used in geoscience, and the International Virtual Observatory Alliance.

  Because each research community has different priorities, the metadata and related ontologies of one are frequently incompatible, unintelligible, or irrelevant to another. A single comprehensive metadata construct for all of science is unrealistic and should be avoided in defining a scientific image standard. Imposing by edict that all scientific communities utilize the same metadata adds unnecessary complexity and could result in laughter or insurrection, or worse, flawed research that wastes time and taxpayer's treasure.

- **Efficiently couple pixel management with changing computer system realities in order to continually optimize scalable performance.**

  Pixels in an image are structured similarly as molecules in homogeneous orthorhombic crystal. The organization of the pixels in a mass storage device directly affects the performance of access and related computation. During the last fifty years computer systems have undergone tremendous hardware re-designs, having a direct effect on the buffering, caching and size of data paths. As image sizes increase, efficient performance will be a critical factor in achieving interactive visualization and related computation.

  Fortunately pixels are a scalable problem that can be managed by the open source tool ***Hierarchical Data Format (HDF)***. [Folk, 1999] HDF was developed by the National Center for Supercomputer Applications funded by National Aeronautics and Space Administration, the National Science Foundation, and the Department of Energy over the last twenty years; it should be considered the most significant data storage tool available and is now completing ISO standardization in support of the aircraft engineering community. HDF is utilized by the Earth Observation System as its primary

data container to manage pixels in its peta-byte repositories, and is also used by the neutron and x-ray communities to manage their image pixels and instrument metadata. But this good news is not to say pixel management is a trivial software problem [Becla, 2005]; a miscalculation of *"doing it yourself from scratch"* could plunge a research effort into a labor-intensive abyss, or produce datasets that are corrupted beyond correction. Issues such as subset extraction, multi-scale, extensible dimensions, extensible modality, tiling/chunking, regional compression, and the inclusion of dissimilar communities' metadata all must work together flawlessly. [Folk, 2003]

Provide communities with the ability to transparently interoperate their software and related image formats within a common unified image model.

A new and advanced image framework must be able to simultaneously present images in the old familiar file formats without requiring changes to existing application software. It should present the images through a scalable image model API for software developers, researchers and vendors to utilize when they are ready to make the necessary technical transitions to directly access the advanced high-performance image design.

Transparent interoperation offers a path of least resistance by providing users the continued ability to function uninterrupted and to use their application software without requiring a change. As such, adoption of a transparent interoperable standard should generate less opposition from relevant communities.

Adopt the recommendations of archival community regarding the design of archival-ready datasets.

Many of the scientific image format designs lack direct input from archivists and library scientists, groups who must maintain these images for indefinite periods of time. Expecting the archival community and library scientists to retroactively correct for fundamental design problems in the lifecycle of imaging formats is asking much from a community that is perennially under-staffed, under-funded, and marginalized.

A critical archival characteristic of a sustainable format is that it is a self-describing, that is there must be sufficient infrastructure within the dataset to properly document its content, context, and structure of the image and related community metadata, thereby minimizing reliance on external documentation.

The most recognized recommendation for managing scientific digital archives is the *Reference Model for an Open Archival Information System* (OAIS), developed for the permanent and long-term preservation of digital information obtained from terrestrial and space observations. [CCSDS, 2002] OAIS completed its ISO certification in 2003.

In undertaking such an effort organizations such as the Library of Congress [Arms, 2003], the National Archives and Records Administration (NARA), International Council for Science, The Committee on Data for Science and Technology, World Data Centers, and the Digital Library Federation should be consulted and integrally involved in defining the strategic goals and characteristics of archival-ready image datasets.

Provide a comprehensively tested software API in lockstep with the image standard's specification.

Providing corresponding software with any standard specification is a critical decision because of legal and economic considerations. The advantages of a lockstep software API include economy, comprehensive testing, detailed benchmarking [Traeger, 2008], and the ability to widely offer timely updates through a single trusted authority. [CENDI, 2007] The alternative is to defer the problem to individual researchers to interpret the specification and faithfully implement independent software codes. But there are differing opinions that warrant further discussion; for example, in the development of DICOM [DICOM, 2006] the interface software is the responsibility of individual equipment manufacturers, who must individually justify the development and maintenance costs, including potential liability due to software failure.

The standard's specification and software API should be freely accessible.

Although addressed in the OAIS and NARA archival recommendations, open disclosure is fundamental lifeblood of scientific research and should be further emphasized separately from an archival perspective. Imagine the use of the alphabet, numbers, or the spoken word where a royalty pre-payment or license fee is required. At some level, unencumbered communication allows researchers to be individuals, to freely retest other's research, and to share ideas, all to the general societal benefit. Considering that taxpayers fund the bulk of basic research, reusable image products should, as a matter of equity, be available in a format that is always public. [ICSTI, 2004] [CENDI, 2007] This is not to say that intellectual property rights have no value, because they are frequently the drivers of research and innovation, nor does it imply that all datasets operating within this image format be made public. Copyrights, patents, and national security will invariably have involvement, but at the core there should be a *tabula rasa* that is truly a blank slate.

Be sponsored and maintained by an accredited standards organization.

Realizing a world-class specification will require the guidance of a professional standards organization that sees the development of such an image standard as part of its mission to serve the public and its membership. Several organizations that may be suitable to sponsor a scientific image standard are NEMA, ECMA, IEEE, I3A, and NISO. Sometimes multiple standards organizations are involved in a standards effort, as in the case of CD-ROM when NISO in the United States and ECMA in Europe simultaneously coordinated the final specification through the International Standards Organization.

6 Proposed Computational Methods

"Simplicity means the achievement of maximum effect with minimum means." — Koichi Kawana

Although HDF is technically a data file format, it is more importantly a high-performance scientific framework acting as a data container capable of organizing scientific data, particularly images, in any configuration imaginable. [Folk, 1999]

The mounting of files as read-only virtual file systems is sometimes used during the download of software updates. Software prototypes developed at NCMI have mounted HDF files as writable virtual file systems using *"Filesystem in User SpacE (**FUSE**)"* [Szeredi, 2009], resulting in a computational hyperspace created between (**1**) the operating system process space that manages mass storage, and (**2**) the user process space whose software applications are requesting access to the image files on mass storage. Because of the similarity of hierarchical design of HDF and the hierarchical design of virtual file systems,

there is approximately a one-to-one mapping between the two systems: groups become folders and datasets become files.

At a simplest level the reading and writing of all files into HDF-VFS is performed assuming that the files are one-dimensional arrays of bytes, identical to the way a typical virtual file system would manage them. There are very few assumptions made as to the nature and composition of the data, it may or may not be an image.

But the HDF-VFS hyperspace enables the following intriguing possibilities:

(1) transparently re-organizing pixels to their appropriate dimensionality and modality within HDF, instantly making them optimal for computation,

(2) encapsulating and managing community metadata through community-specific modules operating within the hyperspace, and

(3) systemically intercepting data streams from any user software application and performing adjunct management such as provenance accounting for archival purposes, or enforcing a community's folder and file-naming conventions.

This in turn, allows for two crucial facilities:

(1) the ability to efficiently and comprehensively manage pixels in a variety of image formats through an HDF-coupled generic pixel model, and

(2) the ability to transparently present these images in a community's native image format and in a manner those existing community software applications could immediately utilize without application software changes.

The design of a common image model is an essential component for interoperability and should be developed by wide consensus through a standards process. This model must grapple with the design of a generic pixel model mapped to the HDF dataset design, and the design of a co-existent method to simultaneously encapsulate multiple communities' metadata in association with the pixels.

By integrating four ideas (HDF-VFS, a common image model, encapsulation of community metadata, and moderation through community-specific modules) into a simple image framework, the scientific user communities will have the technical means to efficiently coordinate their images into the foreseeable future using a high-performance cyber-infrastructure.

7 Proposed Social Methods

"If you can't describe it simply, you can't use it simply." — Anonymous

Establishing a voluntary consensus standard for scientific imaging poses some unique inertial problems inasmuch as it is (or perceived as):

(1) a minor technical issue to be resolved independently by anonymous researchers and vendors, and that it will somehow evolve into a standard for the entire scientific community,

(2) a Gordian knot that cannot be solved because the task is too broad,

(3) a resolved matter because existing imaging standards are up to the task (therefore no further action is needed), or

(4) many scientific researchers are disinclined to become involved because it detracts time and resources from their primary research interests.

Unlike other standards activities such as digital video MPEG and clinical imagery DICOM, whose primary constituency mandated that vendors must fund and produce such specifications for reasons of economics and reliability, the overall scientific community is hard-pressed to find vendors who see it as their core business mission to take on the whole problem at the trench level. *Vis-à-vis*, it is difficult for the vendors to identify the need for and lead the development of a scientific image standard for a highly varied and independent user community. The tactical difficulty is that the image pipelines created by the scientific communities are vast, parochial, and loosely organized. As such the scientific imaging communities have difficulty asserting the demand on vendors to form and finance such a wide consensus. [Bethel, 2003]

Setting the stage requires the assembling of consensus by individuals who are motivated by the overriding strategic advantages, examples being CD-ROM and Unicode. As this informal group evolves it strives to be inclusive by being open to representatives from all interested parties and at the same time seeking out the appropriate professional standards organization to guide the formal development.

The formal development of standards requires transparent discussions leading to impartial draft standards that are presented for broad-based public review and comment. Subsequent consideration of, and response to comments must be made by the accords of due process defined by the accredited standards organization. After the draft standard is voted on and approved, its specifications are considered *dejure* until amended or withdrawn. Occasionally an approved standard is further standardized by its submission to a larger standards organization to emphasize its broader significance, such as CD-ROM and the International Standards Organization.

In developing a draft standard, necessary milestones include:
(1) review technical and administrative requirements,
(2) the design of the pixel model and its coupling to HDF,
(3) the assimilation, encapsulation and management of community metadata,
(4) the mechanism for integrating communities' file formats using community-specific modules within an HDF-VFS paradigm,
(5) the definition of a software API, and
(6) the internal data structures to coordinate the previous items.

8 Conclusion

"Simplicity is the ultimate sophistication." — Leonardo da Vinci

"There are risks and costs to a program of action. But they are far less than the long-range risks and costs of comfortable inaction." — John F. Kennedy

Digital imagery has become indispensable in science. It is therefore imperative that these disparate images be endowed with long-term archival value that can be later integrated into larger scientific models and insights by future researchers.

Digital images are initially created through instrument acquisition and computer simulations, which are then packaged into various incompatible file formats for use by

downstream research communities. In tandem, the acquisition and simulation communities are developing larger dimensional images having new pixel modalities, resulting in overall image sizes that are becoming computationally unmanageable; this is further complicated with unnecessary duplications of pixels due to file format conversions. When independent communities interact with the images, this has resulted in very different ontologies of metadata that must be sequestered and kept in context with the pixels in order to be maintained as archival-ready image datasets.

The existing incompatible image format infrastructures are:

(1) ill-suited to optimize pixel organization relative to computer system designs, and

(2) unable to assimilate heterogeneous metadata ontologies by the various independent communities interacting with the pixels.

These implications are profound and strategic. The failure to establish a scalable n-dimensional scientific image standard in a manner that results in efficient, interoperable, and open public specifications, results in merely a less than optimal research environment and a less certain future capability for image repositories and researchers.

The en-masse coordination of the vast array of scientific image formats into an advanced scientific image standard is a rare opportunity; while at the same time solving a wider set of requirements defined by downstream communities. This can only be accomplished by broad consensus and commitment by the scientific communities, government, and industry.

A simple and universal scientific image standard must be devised to solve these underlying technical problems through formal standard processes. Which will enable the respective scientific communities to transparently interoperate their existing parochial image formats within a unified image model and storage framework.

Acknowledgements

This work was funded by the National Center for Research Resources (P41-RR-02250) and the National Institute of General Medical Sciences (5R01GM079429); and through conferences facilitated by Consortium for Management of Experimental Data in Structural Biology (MEDSBIO) by grants from the Department of Energy (ER64212-1027708-0011962), National Science Foundation (DBI-0610407), and the National Institutes of Health (1R13RR023192-01A1).

I thank Hugh Kress, Steve Sucher, Frank Brown, and Skye Hughes for critically reading the manuscript. And I would also like to thank Wah Chiu (NCMI), Mike Folk (The HDF Group), Herbert Bernstein (Dowling College), Kevin Eliceiri (LOCI/UW-M), Werner Benger (CCT/LSU), Erez Zadok (Stony Brook University), Mike Schmid (NCMI), and Steve Ludtke (NCMI) for their generous consultations and encouragement.

References

[Arms, 2003] Caroline Arms, C., & Fleischhauer, C., *"Digital Formats: Factors for Sustainability, Functionality, and Quality"*, Office of Strategic Initiatives, Library of Congress.

[Becla, 2005] Becla, J., & Wang, D.L., *"Lessons Learned from Managing a Petabyte",* Proceedings of the 2005 CIDR Conference.

[Bethel, 2003] Bethel, E.W., "Interoperability of Visualization Software and Data Models is Not an Achievable Goal", Proceedings of the 14th IEEE Visualization Conference (VIS'03). P. 603.

[CCSDS, 2002] Consultative Committee for Space Data Systems, *"Recommendation for Space Data System Standards: Reference Model for an Open Archival Information System (OAIS)*", CCSDS 650.0-B-1, January 2002.

[CENDI, 2007] CENDI Digital Preservation Task Group, *"Formats for Digital Preservation: A Review of Alternatives and Issues",* March 1, 2007.

[DICOM, 2006] *"Digital Imaging and Communications in Medicine Strategic Document",* October 24, 2006., http://medical.nema.org/.

[Edwards, 2004] Edwards, P.N., *"A Vast Machine: Standards as Social Technology",* Science, Volume 304, Number 5672, p. 827.

[Folk, 1999] M Folk, M., Cheng, A., & Yates, K., (1999), *"HDF5: A file format and I/O library for high performance computing applications",* Proceedings of Supercomputing'99.

[Folk, 2003] Folk M., & Barkstrom B.R., *"Attributes of File Fomats for Long-Term Preservation of Scientific and Engineering Data in Digital Libraries"*, Joint Conference on Digital Libraries 2003, May 27, 2003.

[Gray, 2005] Gray, J., Liu, D., Nieto-Santisteban, M., Szalay, A., DeWitt, D., & Heber, G., *"Scientific Data Management in the Coming Decade",* CTWatch Quarterly, Volume 1, Number 1, February 2005.

[ICSTI, 2004] The International Council for Scientific and Technical Information (ICSTI) & US Federal Information Managers Group (CENDI), *"Digital Preservation and Permanent Access to Scientific Information: The State of the Practice".*

[ICSU, 2004] International Council for Science, *"International Council for Science (ICSU) Report of the CSPR Assessment Panel on Scientific Data and Information: Managing Data and Information for Current and Future Research",* p. 20-23.

[Ifrah, 1999] Georges Ifrah, G., *"The Universal History of Numbers : From Prehistory to the Invention of the Computer",* Wiley 1999.

[MacTavish, 1999] MacTavish S.H., Pickard M.R., *"Electronic Digital Imaging Standards for Archiving Records"*, U.S. Department of Defense report number GA22F042.

[Marciano , 2006] Marciano, R., Hou, C., Moore, R., Rajasekar, A., Burnstan L., & Kreisler, H., *"Long-Term Preservation of Large-Scale Multimedia Collections: A Digital Preservation Workflow Approach",* Archiving 2006, Ottawa, Canada, May 23, 2006, p. 88-91.

[NCRR, 2002] National Center for Research Resources/NIH, *"Data and Collaboratories in the Biomedical Research Community"*, September 16-18, 2002. www.ncrr.nih.gov/biotech/collabmtg2002.asp., Section 2.2, Challenges, p. 7.

[Szeredi, 2009] Szeredi, M., http://fuse.sourceforge.net/

[Tanner, S., 2006] Tanner, S., *"Managing Containers, Content and Context in Digital Preservation: Towards a 2020 Vision",* Archiving 2006, Ottawa, Canada, May 23, 2006, p. 19-23.

[Traeger, 2008] Traeger, A., Joukov, N., Wright, C.P., & Zadok, E., *"A Nine Year Study of File System and Storage Benchmarking"*, ACM Transactions on Storage (TOS), 2008, Volume 4, Issue 2, p. 25-80.

Post-processing Pipeline Optimization for Interactive Exploration of Multi-Block Turbine Propulsion Simulation Datasets

Andreas Gerndt*[a], Rolf Hempel[a], Edmund Kügeler[b], Torsten Kuhlen[c]
[a]German Aerospace Center (DLR), Simulation and Software Technology, Germany
[b]German Aerospace Center (DLR), Institute of Propulsion Technology, Germany
[c]RWTH Aachen, VR Group, Aachen, Germany

ABSTRACT

With the increasing power of current supercomputers, flow field simulations become larger and larger. Those datasets are too large to fit into the main memory of a visualization workstation and too large to be processed in a reasonable time on a stand-alone system. Distributed post-processing can eliminate dominant bottlenecks. With parallel computer systems, the post-processing can show a considerable speed-up. In this paper, we address issues to optimize several phases of the pipeline-based post-processing for interactive exploration of time-dependent, multi-block flow field simulation datasets. Firstly, we present the distributed post-processing framework, its approach to manage multi-block data structures, and how nested parallelization can improve the overall runtime. Scalability, balancing, and efficiency of algorithms optimized for unsteady multi-block datasets are presented.

However, complex feature extractions are still time-consuming. For interactive exploration approaches in virtual environments, further strategies like streaming of intermediate data from post-processing backend to visualization frontend are needed. If the extracted visualization objects are too complex to be rendered with interactive frame rates, view-dependent multi-resolution techniques can help to present essential details without losing real-time rendering. To improve the interactivity, several integrated strategies are evaluated. Run-time measurements prove the efficiency of the approaches. An outlook for future steps concludes this paper.

Keywords: Virtual Reality, Scientific Visualization, Large-scale Datasets, Multi-Block Data Structure, Data Streaming, Multi-Resolution Approaches

1. INTRODUCTION

To be capable to investigate very complex flow-fields, an advanced flow solver makes use of massive parallel supercomputers. Nevertheless, the computation of thousands of time steps with moving geometries that show millions of data points still need days, weeks, or even months. Often, the CFD (Computational Fluid Dynamics) results are not satisfactory, and the simulation has to be repeated with modified input parameters. An essential problem of the analysis of the resulting datasets is the lack of appropriate tools for the post-processing, i.e., for the procedure to extract important features out of the huge grid-based simulation results and to visualize them for a convenient analysis. Often, 2D diagrams of single quantities are sufficient for the confirmative analysis of such 3D simulations. To find new behaviors or features of a flow field, a more interactive evaluation approach is necessary. In this case, Virtual Reality (VR) systems are the best choice to present multi-dimensional and spatial aspects of unsteady 3D flow fields.

Nowadays, even for high-end VR systems, off-the-shelf components are used to build up the visualization hardware infrastructure. Here, a particular bottleneck is the small main memory so that the local processing of terabytes of simulation data requires out-of-core approaches (cf. [21]). As the main goal for interactive visualization environments is to render images as fast as possible, it is recommendable to relieve those systems from the time-consuming and I/O intensive post-processing. Therefore, distributed post-processing environments may be a preferred solution. All resources available on the visualization frontend can then be used for the real-time evaluation of VR devices and to render the extracted visualization objects.

In an optimal case, the post-processing can be moved to those supercomputers which are used to simulate the flow-fields. But also PC-clusters as well as multi-core workstations become more and more an option. On such multi-processor systems, parallelization approaches (cf. [1]) may reduce the processing time considerably. However, their efficiency is primarily restricted by communication overhead (cf. [15]). Especially when datasets are already partitioned into multiple blocks, data parallelism can be very efficient. As long as heavy communication can be avoided, post-processing scales almost linearly with the number of processors involved in the feature extraction. Scalability is the main evidence of the

*andreas.gerndt@dlr.de; phone +49 531 295-2782; fax +49 531 295-2767; www.dlr.de

usefulness of the proposed distributed approach. Therefore, one topic of this paper is the assessment of scalability of algorithms performed on shared-memory as well as distributed memory systems.

The remaining paper is structured as follows: The next section reflects related work in the field of large-scale data in virtual environments. Then, the distributed post-processing architecture that has been developed is described. Section 4 explains the multi-block propfan dataset and the topology meta-data in more detail. In subsequent sub-sections, advanced nested parallelization aspects, ways to stream intermediate and multi-resolution data from backend to frontend, and eventually view-dependent multi-resolution rendering strategies are presented. The paper concludes with an outlook on future work.

2. Related Work

The processing of large datasets requires a large amount of main memory and may lead to an intensive CPU load. Thus, the execution on real-time systems is not feasible so that researchers have been developing distributed systems over the last three decades. A widely used analysis system was Plot3D already presented in 1985 by Buning and Steger [4]. Further popular systems were e. g. FAST by Bancroft et al. [2], pV3 by Haimes [10], and UFAT by Lane [13]. A comparison of these early frameworks is given in [14].

One of the first available systems for VR-based flow visualization was the Virtual Wind Tunnel presented by Bryson et al. An extension connected the virtual environment with a post-processing back-end which was responsible for the feature extraction [3]. COVISE is a system with a focus on collaboration. Originally designed as a data flow oriented post-processing toolkit for desktops, the optional COVER module also allows its usage in virtual environments [17]. The data flow and applications of distributed post-processing for interactive visualization is also investigated by Kuhlen et al. [12]. Their framework, as most of the applications available today, makes use of post-processing algorithms implemented in the Visualization Toolkit (VTK) [20].

3. Distributed Post-Processing Framework

The goal of the developed framework was to hide post-processing implementation details and the usage of VR technologies in order to offer a straightforward application programming interface. At the same time, it should be open enough to be suitable for a large range of engineering problems. In this section, the developed framework is presented.

3.1 Visualization Frontend

The main aspects of the developed VR-based visualization framework are covered by toolkits freely available from public domain or open source projects. Almost all data processing functionality is offered by the Visualization Toolkit (VTK). It is based on a pipe and filter concept that uses the output of one processing step as input for the successive filter. The end of such a pipeline yields a set of polygons needed for the rendering process. Although VTK also offers rendering capabilities, this task is taken over by ViSTA FlowLib [19]. The main goal of this toolkit is the time-correct visualization of unsteady datasets. For this purpose, it creates data containers as simple arrays for each dataset used in an application. In general, a simulation time step of an unsteady dataset is considered as input of a post-processing pipeline. The processed result is then stored in a bin of the data array.

Actually, ViSTA FlowLib does not care about the post-processing. It only expects that the data arrays are filled with data which are defined in a format supported by the selected renderer. ViSTA FlowLib offers a set of renderers for various visualization approaches. The default renderer visualizes data which is defined in VTK polydata format.

3.2 Parallelization Backend

In general, the application is responsible to assign data to the array bins. On the other hand, the user can also decide that the data is produced by an external post-processing application. Here, Viracocha comes into play [7]. It is a parallelization framework with a communication protocol which allows receiving requests from ViSTA FlowLib to compute data and sending results back for rendering. Running on a supercomputer, Viracocha can also carry out the requests in parallel. The application configures the requests and determines how data is transmitted, i.e., whether partial results by individual Viracocha processes are sent in parallel to ViSTA FlowLib or bundled to one large packet before being sent. Communication aspects and data handling are managed automatically by the base toolkits. The application developer does not have to control the data flow. As soon as results arrive at the visualization frontend, they are rendered automatically. The general data flow is depicted in Fig. 1.

The main components of Viracocha are the scheduler and a fixed amount of workers. When the visualization host sends a request to get new extracted flow features, the scheduler receives it and creates an appropriate task controller. As soon as enough workers are available to process the request, a local worker group is created to compute the task in parallel. A worker object and a task controller object are just organizing the work flow. The algorithms themselves are defined as commands and are created by means of command factories. The visualization host sends command IDs as part of the

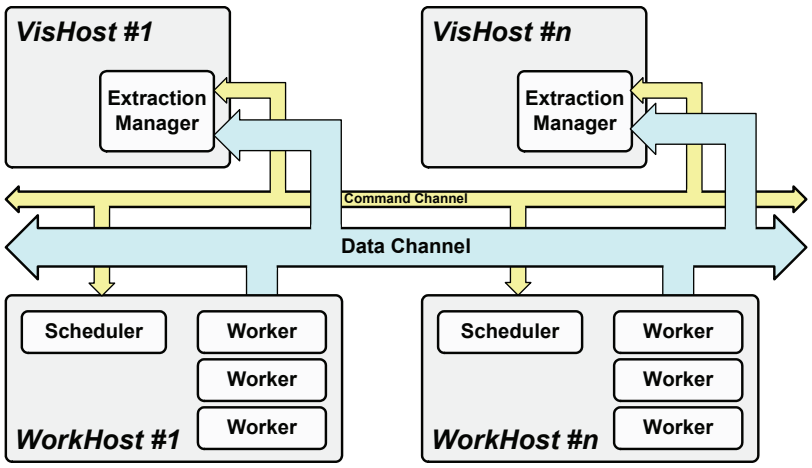


Fig. 1. Principal concept of the distributed framework

request which specifies the actual algorithm to be processed. This mechanism (cf. Fig. 2) allows to plug in new algorithms into the framework without changing base components.

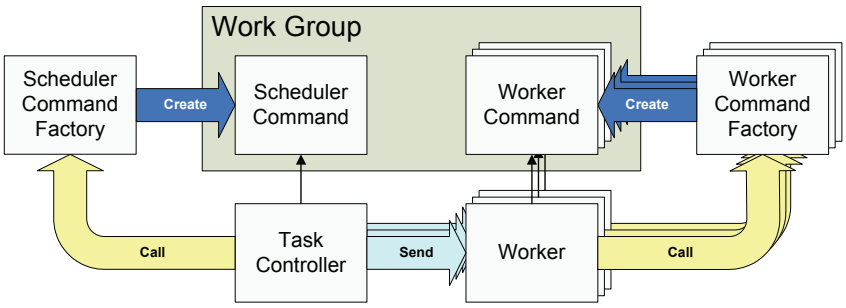


Fig. 2. Essential interfaces and workflow to create a worker group.

To be more flexible, the communication between scheduler and workers as well as the communication within such a local worker group is based on message passing [9]. The Message Passing Interface (MPI) hides the architecture used to compute the tasks in parallel. Therefore, Viracocha can run on shared memory as well as on distributed memory systems.

3.3 Data Management System

Viracocha offers an open parallelization architecture which can be used for a variety of advanced post-processing approaches. Its main bottleneck is data loading. Workers often have to wait for the data before a computation can be continued. This typically leads to load imbalances. Some post-processing algorithms implemented in Viracocha address those issues, but they are problem-specific and not available for general use. This was the reason for the development of an integrated data management system (cf. Fig. 3) called VDMS (Viracocha Data Management System) that provides data handling functions to all integrated post-processing methods.

The main goal was to bring the data as close as possible to where it is processed. Since data access times differ by orders of magnitude depending on whether the data is in main memory already or stored on a file server, moving the data through the memory hierarchy before it is needed in a computation can reduce processor idle times and load balancing problems. The main strategies implemented are:

- **Caching**: Already loaded data is stored in main memory as long as sufficient space is available. The next request by any process gets the data directly from the cache. A second-level cache moves the data to a local hard drive from where it can be loaded faster than from a file server.
- **Prefetching**: Data not requested yet is already loaded into the main memory by a second thread while the main thread continues with data processing. Later, when the main thread needs the prefetched data, it can access it immediately.
- **Load Strategies**: Depending on hardware resources and data structure, optimal strategies can be selected. For instance, if a parallel file system is installed, appropriate strategies can be integrated.
- **Data Service**: The VDMS Server as a central organizing component is responsible to assign data to one of the parallel workers whenever it has finished its current task. This can result in a better balanced computation.

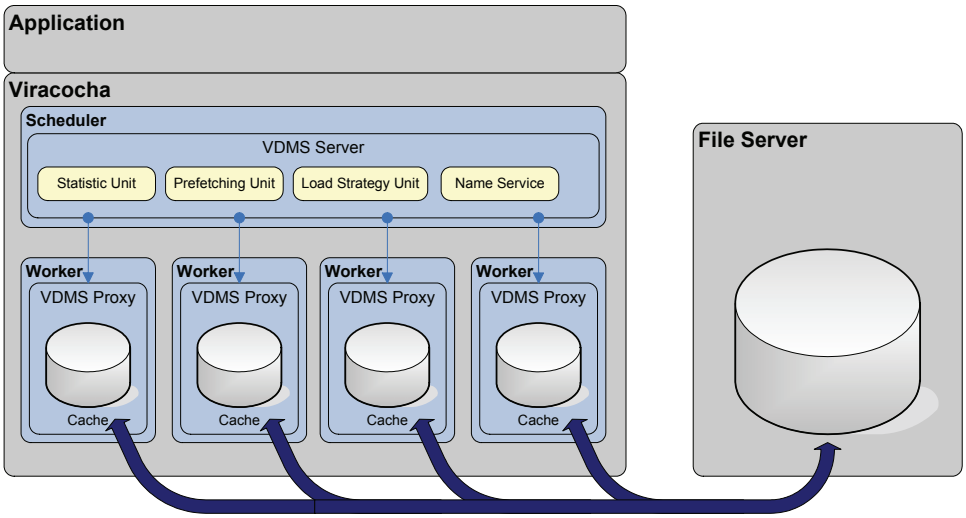


Fig. 3. Main components of the Viracocha Data Management System (VDMS) to reduce the time a process has to wait for requested data.

4. Distributed Multi-Block Post-Processing

Our main interest is the evaluation of large-scale simulations of turbine flow fields. Over the last years, at the German Aerospace Center (DLR) we have been developing a highly accurate multi-block solver called TRACE (Turbo machinery Research Aerodynamic Computational Environment) [6]. The entire considered flow field is partitioned in a block-structured way. The dataset evaluated in this paper is a counter-rotating turbine with two rows of 12 blades each. For reasons of symmetry, only the flow around one blade is computed (cf. Fig. 4). 12 blocks are required to decompose the blade's flow field. A complete rotation is computed in 50 time steps. Because of the counter-rotating behavior, the blocks change their relative positions so that the domain decomposition is different for each time step and stored separately. For a comprehensive rendering of the entire propfan, the blocks are replicated for all blades, which results in 144 blocks for each of the 50 time steps. Each block is stored in a separate VTK file. The size of the entire mesh is 2,533,356 nodes and does not change over time. With 1 vector and 5 scalar quantities computed at each node, the datasets of the completed propfan has a size of 5.1 GByte.

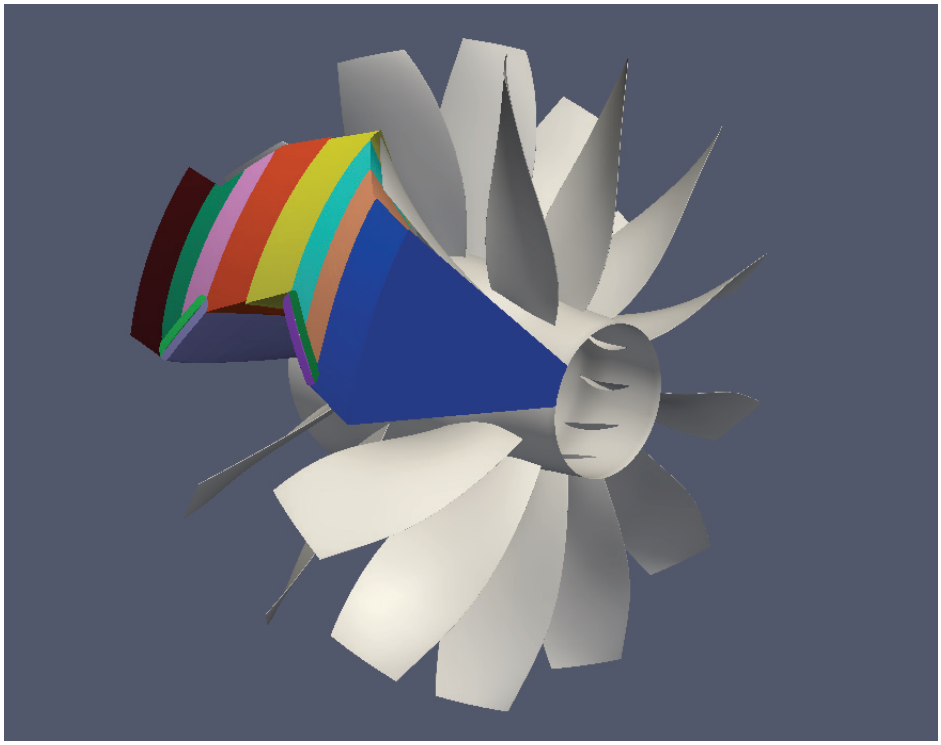


Fig. 4. Multi-block structure used to describe the flow field for the first time step of the counter-rotating propfan, each block colored differently.

A single block is defined as a rectilinear grid. Thus, the neighbor relationship of cells within a block is defined by the so-called *i-j-k* coordinates of the computational space (cf. [18]). Between blocks, appropriate information is missing. For a faster prost-processing, an additional multi-block topology specification has been developed. For most of the cell-based extraction algorithms, this does not play an important role. But for global approaches, like particle tracing, multi-block topologies are very helpful. If the connected sides of neighbored cells of adjacent blocks match exactly, as in the case of the considered multi-block propfan, just some few additional meta-information has to be retrieved and stored. The data structure of the multi-block topology is depicted in Fig. 5.

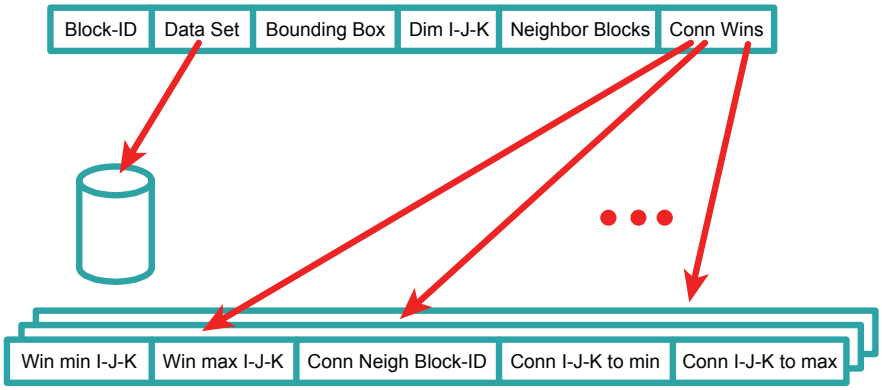


Fig. 5. Meta-data for multi-block topology.

Not only neighbor block IDs are stored to specify the connectivity but also so-called connection windows are computed. They specify the *i-j-k* cell range where a current block is connected to a neighbor block. This creates a very accurate topology graph which can be used for fast cell search algorithms in multi-block datasets.

4.1 Nested Parallelization of Critical Point Computation

For the analysis of flow fields, the topology of the velocity field plays an important role. The core features of the topology is a critical point that defines the location where the vector magnitude becomes zero, i.e., where the velocity vanishes. Starting from those feature points, the entire flow field can be sub-divided in homogeneous segments. They are also often used to identify turbulences in flows. For the propfan, they mainly appear at the blade tips (cf. Fig. 6).

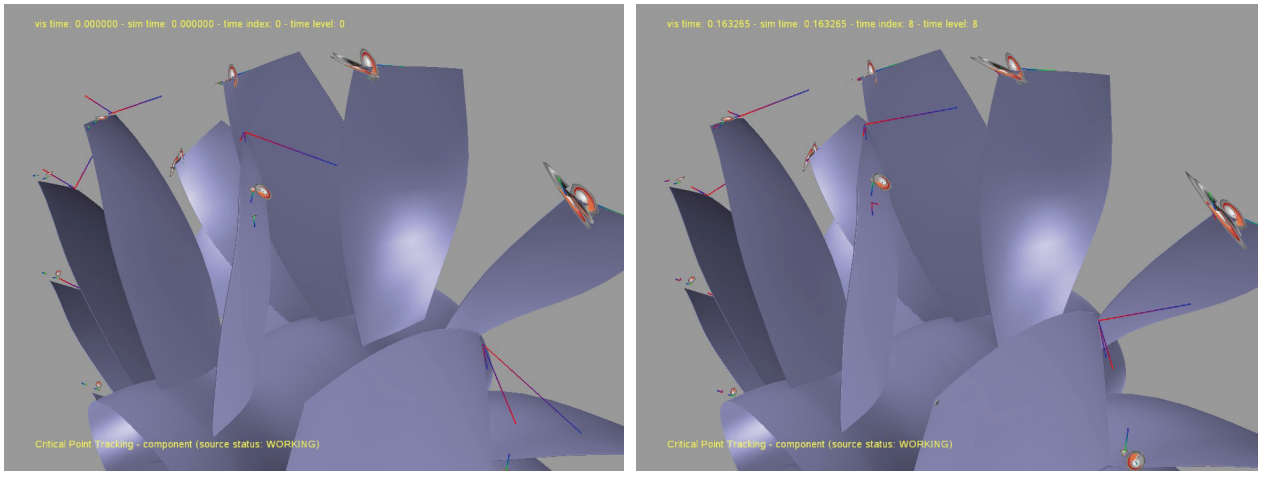


Fig. 6. Critical points (different eigensystem icons are depicted) are primarily detected at blade tips. Subsequent time steps show that critical points remain pretty stable in this propfan dataset.

Computing and classifying critical points in a hexahedral grid is a cell-based approach (cf. [8]). Although no topology or neighbor information are needed, it is a very time-consuming task. Parallelization can decrease the computation time for critical points extraction considerably. On our test system – a SunFire E6800 supercomputer node with 48 processors – the Viracocha approach using MPI to employ its parallel workers does not show an optimum speedup. Therefore, a further parallelization level was included. Not only blocks but also the cells of a block were now distributed to processors. As this is not supported by the generic approach of Viracocha, this can only be done on algorithm level. A very straightforward parallelization approach is to use OpenMP [5] instead of MPI. As OpenMP primarily aims at multi-threading parallelization, its application is restricted to shared-memory systems.

To assess both parallelization interfaces, we investigated the run-time behavior when merely blocks are distributed. For this measurement, one Viracocha worker was used who subsequently dispatched its blocks to OpenMP threads. In

comparison to the MPI result, the speed-up (defined as runtime with one processor divided by runtime gained with *n* processors) with OpenMP was clearly improved (cf. Fig. 7). The main reason for this improvement can be found in the strategy how blocks are assigned to workers. Viracocha by default assigns a static list of blocks to a worker without considering the varying sizes of single blocks. OpenMP, however, is able to dispatch blocks dynamically to its threads. This results in a better load balancing.

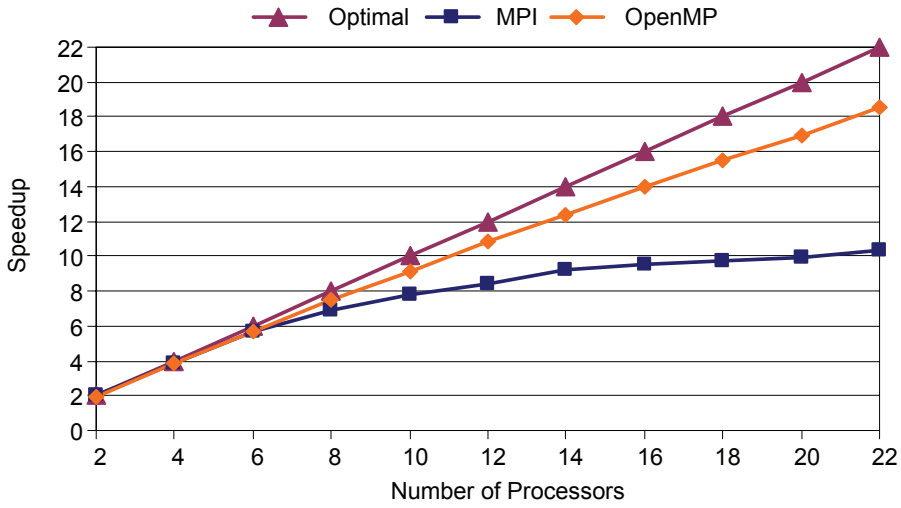


Fig. 7. Speed-up using *n* Viracocha workers for MPI-parallel critical points computation (blue) and using one worker but *n* OpenMP processes (orange) on a shared memory system. Time steps 10 to 15 were processed.

The SunFire cluster also supports Nested OpenMP, e.g., an OpenMP thread can start further sub-threads, and so on. In the case of critical point computation in a multi-block dataset, this can be exploited by dispatching the 144 blocks in a first stage and assigning all 2,373,600 cells in an inner loop. Because of very different block sizes and imbalanced computation loads, OpenMP runs pretty quickly into balancing problems. Especially when the number of cells assigned to a thread are too small, the time to coordinate threads increases and the efficiency (defined as speed-up divided by number of processors) of nested parallelization drops. This is clearly visible in Fig. 8.

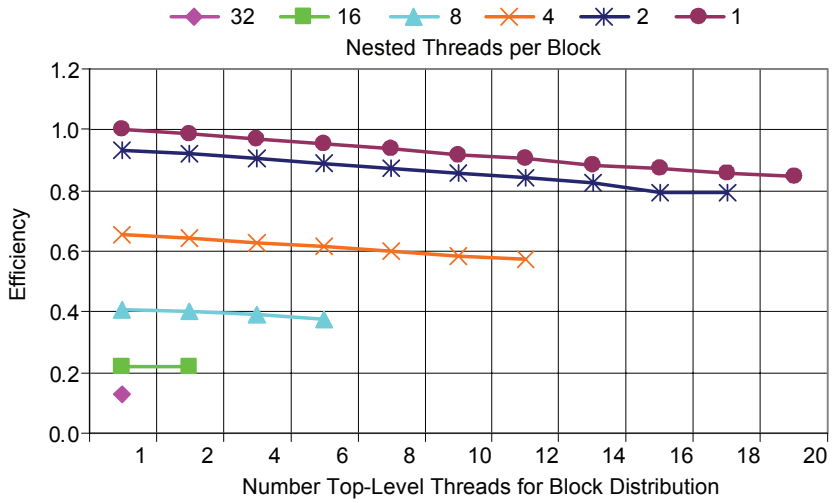


Fig. 8. Efficiency measurement for the computation of critical points for time steps 10 to 15 using Nested OpenMP.

4.2 Feature Extraction Data Streaming

Relieving the visualization host from time-consuming post-processing may enable interactive exploration of already extracted features. But if a user in virtual environments has to wait too long for requested features, this interactive post-processing approach will not be accepted. Parallelization and data management can reduce computing time considerably, but as soon as very complex algorithms have to be carried out, the latency becomes an issue. Data streaming offers a solution to bridge the time between request and transmitting the final results. It presents intermediate results so that the user can start the evaluation process very early. Often preliminary results can already offer an impression of the final

result. With the option to cancel a running task on the backend and to re-send the request with modified parameters, the interactivity may be increased.

The multi-block structure supports data streaming. A straightforward solution is that a worker transmits its partial results back to the visualization frontend whenever one block is processed. For unstructured grids, this approach can be reached by processing just a certain amount of cells before the partial results are sent back. In this case, it might be possible that a large amount of unconnected polygons has to be visualized with consequences on the rendering performance. This is typically not the case in multi-block datasets. Nevertheless, block-based streaming does not guarantee that the user obtains a good and fast forecast of the final result (cf. Fig. 9).

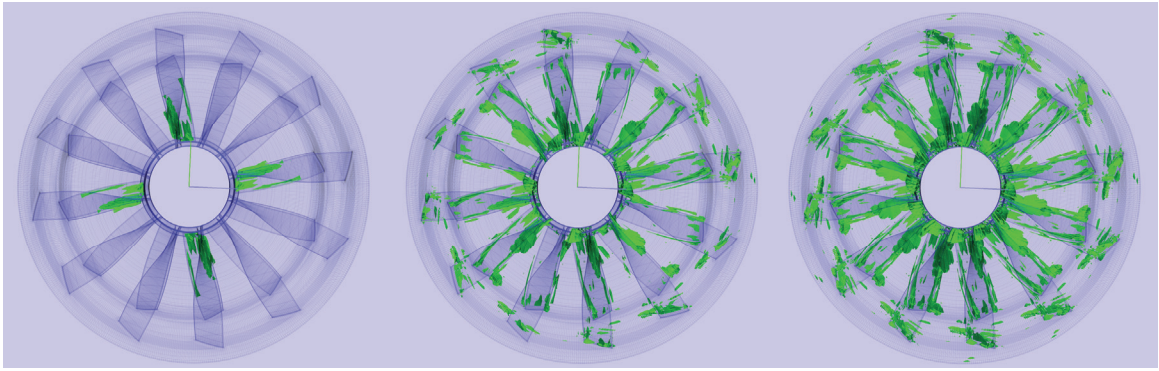


Fig. 9. Multiple streaming steps of Lambda-2 vortex extraction computed by 4 workers and then streamed independently to the visualization frontend.

Post-processing algorithms that can be used for block-based streaming are e.g. isosurface extraction and cutplane computation. The computation in one block can be performed very quickly. But when many workers produce many packages for the frontend in a very short time, this can lead to a communication bottleneck as depicted on Fig. 10. An increasing number of parallel streaming workers make it worse. More data packages arrive at the visualization host than it can process. This hinders the communication and can even slow down the rendering performance. This emphasizes the importance of efficient network communication and optimized communication protocols. As soon as the computation becomes more complex, for instance, when computing vortex structures based on the Lambda-2 method, the communication bottleneck does not play a prominent role anymore.

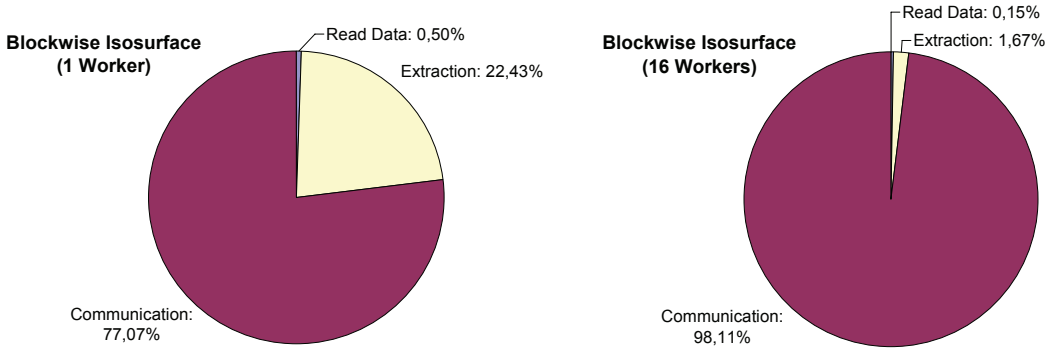


Fig. 10. Communication overhead with increasing number of workers.

To provide the user with relevant information, it may be useful to submit data first which is close to the user and will not be occluded by other objects later on. To implement a straightforward view-dependent approach, blocks are not processed in terms of their block IDs but are sorted according to their bounding-boxes and then processed front to back. With increasing numbers of blocks, this approach offers already a good approximation of view-dependent streaming, works also in parallel, and is very fast. A more accurate way exploiting the user position is based on a binary space partitioning (BSP) tree. But this requires that all cells of a time step are loaded. Therefore, the multi-block dataset must be first merged to an unstructured grid. Although such tree data structures can also help to speed-up the feature extraction for instance of isosurfaces, online merging of large-scale multi-block datasets or storing an unstructured version of them on a file server is generally not feasible because of the time needed to merge the data and the space allocated on file servers.

Also view-dependent streaming can not guarantee that first incoming data presents already a good impression of the final result. Streaming that is based on a multi-resolution version of the dataset can offer this. A level-of-detail approach sub-samples each block multiple times. Each thinned out dataset is stored to the file server. When now a feature extraction request arrives, the dataset level with the lowest level is selected first and the post-processing algorithm is performed.

After sending this low-resolution result to the visualization host, the next higher level is loaded and processed. This goes on until extraction data from the original dataset can be presented. Loading low-resolution data into memory is very fast and data can be computed almost immediately so that the user can start with the analysis shortly after the request. The disadvantage of this approach is the additional space on the file server and the time the user has now to wait until the original data is processed.

Most wanted are progressive algorithms that present early a low-resolved but complete result with data points which are not removed but completed by next arriving partial data. These are hardly to develop as for this purpose the resulting geometry has to be known in advance. One post-processing algorithm with a pre-known geometry result is the cut function based on point sampling. Instead of computing the intersection points of a cut object with the grid, points on the surface of the cut object are used to determine the scalar value at that location of the grid. As one example, a sampling algorithm has been developed using a cylinder as cut object. By defining the radius of the base circle, the height of the cylinder, and a tessellation degree, the cylinder surface can be regularly triangulated with an arbitrarily resolution (cf. Fig. 11).

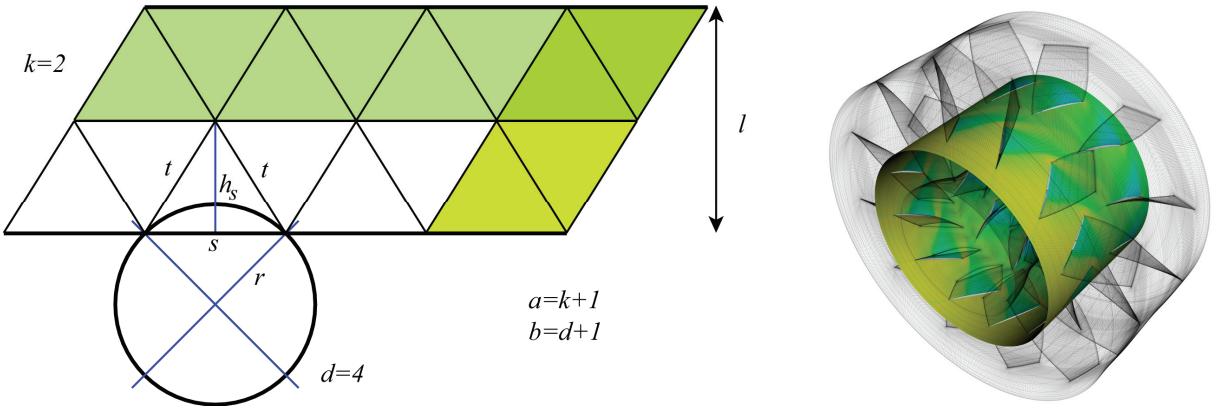


Fig. 11. Triangulation parameters of a cut cylinder (left), resulting cut through the propfan (right).

The computation can now be parallelized in different ways. For instance, the number of triangle strips can be distributed to workers. The time steps, of course, may be computed in parallel as well. The streaming would still not be progressive. But the tessellation approach depicted in Fig. 11 points already to a first progressive solution. If the resolution *d* on the base circle and the resolution *k* of the cylinder height is a multiple of primes, these primes can be used to nest the point sampling computation.

A more elegant way to integrate a progressive cut streaming is the surface subdivision scheme proposed by Kobbelt [11] (cf. Fig. 12). Only the set of newly added surface points are taken to determine their scalar values which are then streamed back to visualization host. Although those sampling approaches are pretty expensive to determine scalar values, the progressive streaming approach is very effective in improving the interactivity in virtual environments.

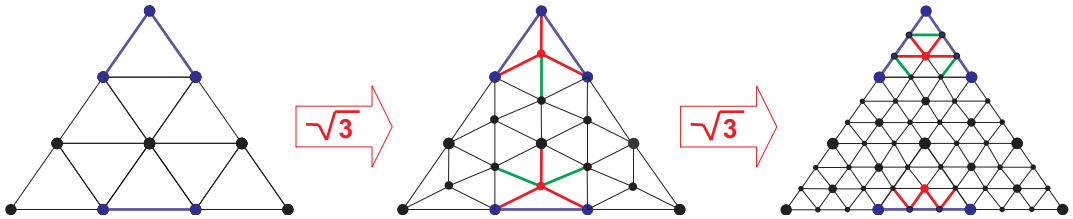


Fig. 12. The Sqrt-3 surface subdivision scheme (special boundary and vertex treatments (blue) are required: new edges (red), edge flips (green)).

4.3 Multi-Resolution Geometry

After the final extraction data was streamed, all intermediate data might be deleted. On the other hand, to gain interactivity on the frontend, it could be useful to switch back to a lower resolution in dependence of the current framerate. This is enabled by a multi-resolution data object that can be instantiated on the visualization host in order to cache arriving streaming data. A multi-resolution manager measures the current framerate and selects an appropriate resolution which shows as many details as possible without losing interactivity.

The multi-resolution manager is also capable to evaluate the current view-point for a view-dependent resolution of multi-resolution data object. Meshes based on the Sqrt-3 data structure support this. A more time-efficient progressive approach uses the FastMesh algorithm introduced by Pajarola et al. [16]. A result of our implementation is presented in Fig. 13.

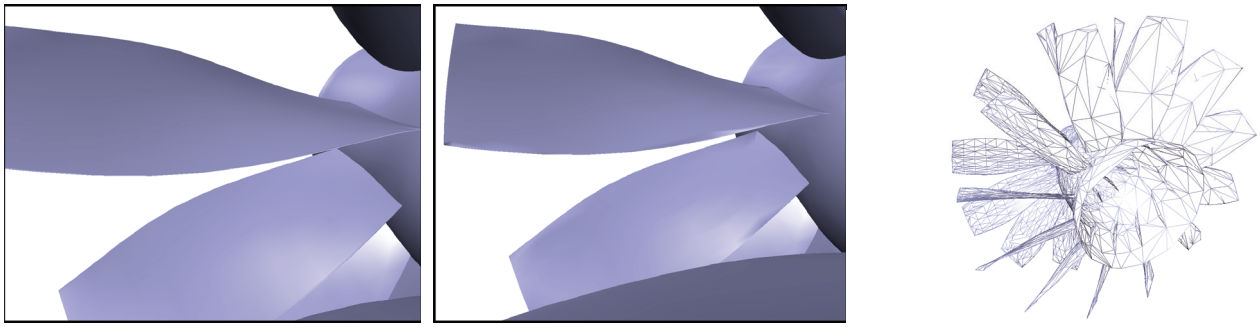


Fig. 13. View-dependent optimized view of propfan blades, original with 340,000 triangles (left), view-dependent optimized with 8,532 triangles (mid), and wireframe presentation of the entire view-dependent optimized model (right).

5. Conclusion and Future Work

This paper has presented the ongoing work to optimize several stages of a distributed post-processing pipeline. The primary goal is to evaluate very large-scale CFD datasets of turbo-machinery designs in interactive environments. The dataset used in this paper comes with a moderate size but was useful to develop and evaluate new algorithms. A currently started project has to deal with 1678 blocks and 1792 time steps (cf. Fig. 14). The goal is that the approaches depicted in this paper have also the ability to explore such terabyte flow fields datasets interactively in virtual environments.

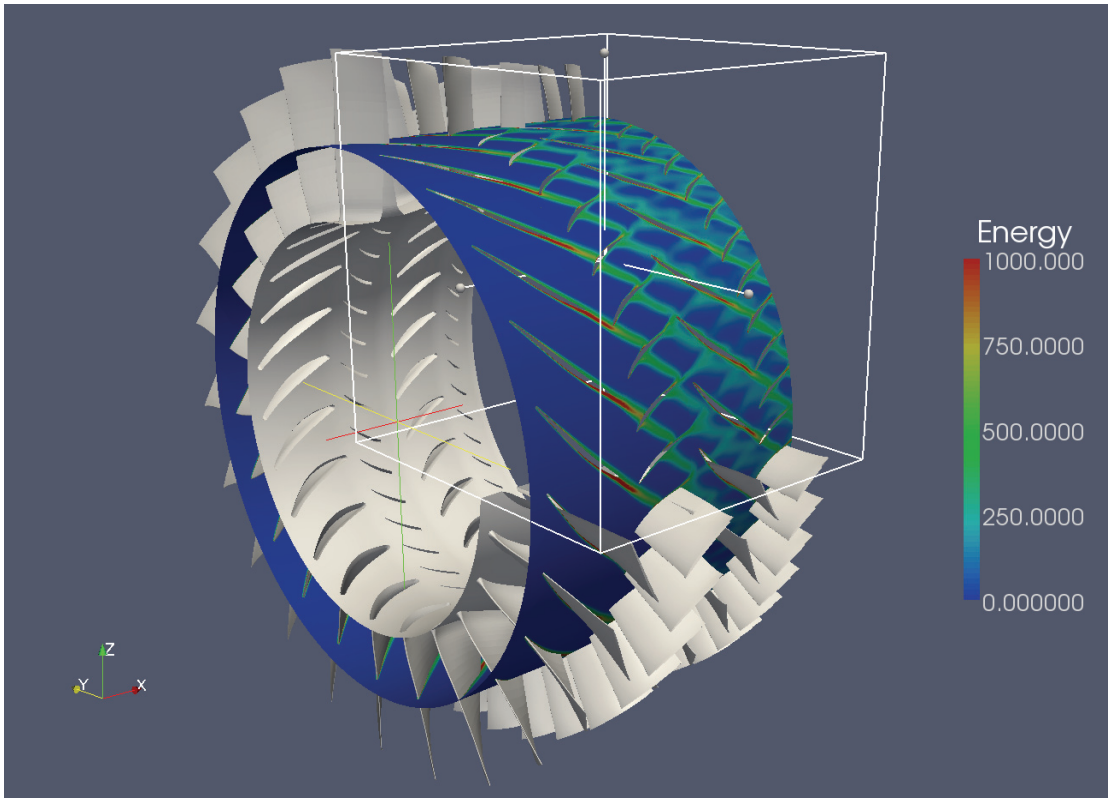


Fig. 14. The next step: a 4.4 terabyte unsteady multi-block turbine dataset. Scalars are computed on a cut cylinder and a part of geometry is clipped by a clipping box.

An advanced framework has been developed for that purpose which comes with a flexible parallelization and data management concept. Special approaches have been integrated for multi-block data structures. It could be shown that nested parallelization strategies can result in a more balanced computation and therefore are capable to reduce the time a user has to wait for requested features. However, time-consuming post-processing algorithms demand further strategies to reduce the latency of the distributed system. Several approaches have been presented based on data streaming to address this problem. Progressive strategies are the most adequate solutions but unfortunately not available for most of the post-processing algorithms. It is an ongoing research activity to find and develop more sophisticated approaches in this field. Finally, we have presented first results how to render polygonal models with a high complexity with interactive framerates. The multi-resolution manager makes use of streamed data as well as optimized view-dependent

data structures. As the generation of such data structures for very large polygonal objects is also time-consuming, we are working on parallel strategies to compute adaptive meshes in parallel on the computation backend as well.

REFERENCES

[1] Ahrens, J. P., Law, C. C., Schroeder, W. J., Martin, K., and Papka, M., "A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Datasets", Technical Report #LAUR-00-1620, Los Alamos National Laboratory, 2000.

[2] Bancroft, G. V., Merritt, F. J., Plessel, T. C., Kelaita, P. G., Kelaita, R. K., and Globus, A., "FAST: A Multi-Processed Environment for Visualization of Computational Fluid Dynamics", A. Kaufmann (Ed.), Proceedings, IEEE Visualization, San Francisco, CA, pp. 14 – 27, 23. – 26. October 1990.

[3] Bryson, S. and Gerald-Yamasaki, M. J., "The Distributed Virtual Windtunnel", Proceedings, Supercomputing, Minneapolis, Minnesota, USA, pp. 275 – 284, 1992.

[4] Buning, P. G. and Steger, J. L., "Graphics and Flow Visualization in Computational Fluid Dynamics", Proceedings, 7. Computational Fluid Dynamics Conference, AIAA-1985-1507, Cincinnati, OH, 15. – 17. July 1985.

[5] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R., "Parallel Programming in OpenMP", Morgan Kaufmann, 2000.

[6] Franke, M., Kügeler, E., and Nürnberger, D., "Das DLR-Verfahren TRACE: Moderne Simulationstechniken für Turbomaschinenströmungen", Proceedings, Deutscher Luft- und Raumfahrtkongress, Friedrichshafen, Germany, September 2005.

[7] Gerndt, A., Hentschel, B., Wolter, M., Kuhlen, T., and Bischof, C, "VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments", Proceedings, Supercomputing 2004, Pittsburgh, PA, USA, 2004.

[8] Globus, A., Levit, C., and Lasinski, T., "A Tool for Visualization the Topology of Three-Dimensional Vector Fields", G. M. Nielson, L. Rosenblum (Eds.), Proceedings, IEEE Visualization, San Jose, CA, pp. 33 – 40, 22. – 25. October 1991.

[9] Gropp, W., Lusk, E., and Thakur, R., "Using MPI-2, Advanced Features of the Message-Passing Interface", MIT Press, 1999.

[10] Haimes, R. "pV3: A Distributed System for Large-Scale Unsteady CFD Visualization", Proceedings, 32. Aerospace Science Meeting and Exhibit, AIAA 94-0321, 1994.

[11] Kobbelt, L. "Sqrt-3 Subdivision", Proceedings, ACM Siggraph, ACM Press, pp. 103 – 112, 2000.

[12] Kuhlen, T., Gerndt, A., Assenmacher, I., Hentschel, B., Schirski, M., Wolter, M., and Bischof, C., "Analysis of Flow Phenomena in Virtual Environments – Benefits, Challenges and Solutions", Proceedings, 11. International Conference on Human Computer Interaction, HCII 2005, Las Vegas, NV, USA, 2005.

[13] Lane, D. A., "UFAT: A Particle Tracer for Time-Dependent Flow Fields", D. Bergeron, A. Kaufmann (Eds.), Proceedings, IEEE Visualization, Washington D.C., IEEE Computer Society Press, pp. 257 – 264, 17. – 21. October 1994.

[14] Lane, D. A., "Scientific Visualization of Large-Scale Unsteady Fluid Flows", G. M. Nielson, H. Hagen, H. Müller (Eds.), „Scientific Visualization – Overviews, Methodologies, Techniques", IEEE Computer Society Press, pp. 125–145, 1997.

[15] Law, C. C., Martin, K. M., Schroeder, W. J., and Temkin, J. E., "A Multi-Threaded Streaming Pipeline Architecture for Large Structured Data Sets", Proceedings, IEEE Visualization, Washington D.C., IEEE Computer Society Press, 1999.

[16] Pajarola, R. and DeCoro, C., "Efficient Implementation of Real-Time View-Dependent Multiresolution Meshing", IEEE Transaction on Visualization and Computer Graphics, Vol. 10:3, May / Juni 2004.

[17] Rantzau, D. and Lang, U., "A Scalable Virtual Environment for Large Scale Scientific Data Analysis", Future Generation Computer Systems, 14: 3 – 4, Elsevier Science B. V., pp. 215 – 222, 1998.

[18] Sadarjoen, I. A., van Walsum, T., Hin, A. J. S., and Post, F. H., "Particle Tracing Algorithms for 3D Curvilinear Grids", in: G. M. Nielson, H. Hagen, H. Müller (Eds.), „Scientific Visualization – Overview, Methodologies, Techniques", IEEE Computer Society Press, pp. 311 – 335, 1997.

[19] Schirski, M., Gerndt, A., van Reimersdahl, T., Kuhlen, T., Adomeit, P., Lang, O., Pischinger, S., and Bischof, C., "ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments", Proceedings, 7th International Immersive Projection Technologies Workshop, and 9th Eurographics Workshop on Virtual Environments, ACM Siggraph, Zurich, Switzerland, pp. 77 – 85, 2003.

[20] Schroeder, W., Martin, K., and Lorensen, B., "The Visualization Toolkit – An Object-Oriented Approach to 3D Graphics", 4th edition, Kitware Inc., 2006.

[21] Silva, C. T., Chiang, Y, El-Sana, J., and Lindstrom, P., "Out-Of-Core Algorithm for Scientific Visualization and Computer Graphics", Course Notes, IEEE Visualization, Boston, MA, October 2002.

TOWARDS AN INTERACTIVE AND DISTRIBUTED VISUALIZATION SYSTEM FOR EXPLORING LARGE DATASETS

ANDREI HUTANU[1,2], JINGHUA GE[1], CORNELIUS TOOLE, JR.[1,2], AND GABRIELLE ALLEN[1,2]

Abstract. We analyze the problem of volume rendering of large remote data. Existing systems and approaches for distributing the visualization process are analysed and a new system that enables high data-transmission rates by adding on-demand instantiation of a distributed data server in the network is proposed. Our current results include a prototype implementation that successfully demonstrated the viability of the proposed approach during SC08

1. Introduction

Visualization and analysis is a core need for the computational scientists developing and running their science and engineering applications on national cyberinfrastructure. Simply because of its size, scientists today face the problem of not being able to effectively analyse and visualize the data that they generate and in consequence, much of the data that is computed is actually discarded and never analysed.

The scenario of interest is illustrated in Figure 1. We look first at the case where the simulation data is stored on a local storage system attached to the supercomputer that was used to generate it. The user is connected using a high-speed network infrastructure to the data server. At various locations in the network, compute and rendering resources are available that could be used to improve the visualization experience provided to the user. In Section 3 we discuss how the data could be streaming from a live simulation, rather than a static file. We are mainly interested in systems where the control over the visualization process is given to the user. The user decides what data he wants to visualize, and has continuous control over the visualization parameters. The system needs to be interactive (at least 5-10 frames per second video rate) and responsive (maximum 1-2 seconds response time to changes in parameters or data).

2. Related work

Shalf and Bethel, [20], discuss how successful visualization systems can be extended to grid and distributed computing environments. They propose distributed visualization architecture to support a variety of distributed visualization applications by hiding the complexity in the allocation and use of distributed resources.

There are various ways in which a visualization application can be created to solve the problem, such as running the visualization on the data server (Figure 2)

[1] Center for Computation & Technology, Louisiana State University, Baton Rouge, LA, USA.
[2] Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA.

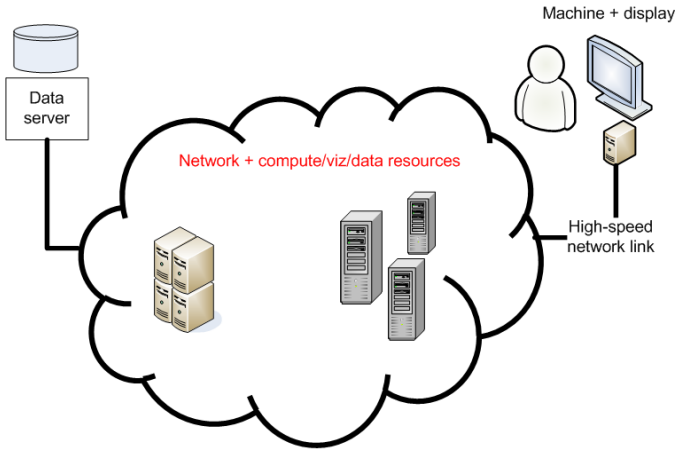


FIGURE 1. Basic visualization scenario.

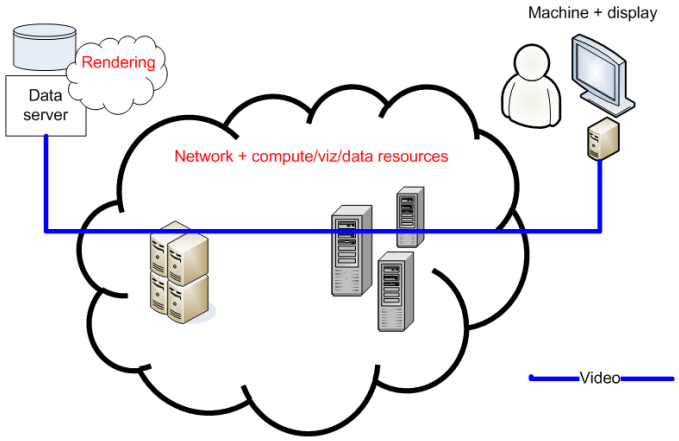


FIGURE 2. Remote visualization (video streaming). Here the visualization is performed on the data server and a video stream carries the resulting images to the user.

and running a video stream to the client, or run the visualization on the local client (Figure 3) and run a data stream between the data server and the client.

These two solutions do not utilize resources in the network and are limited by the visualization power of the data server or local machine respectively. Another possible solution is to build a three-way distributed system that uses a visualization cluster in the network, data streaming from the data server to the cluster and video streaming from the cluster to the local client (Figure 4)

The TeraGrid(TG) [2] remote visualization services is done through two different models:

Server provided visualization: Remote visualization sessions can be launched on Spur, TACC's TeraGrid Visualization System, through the use of a VNC server running on Spur and a VNC client running in a Web browser on the your local

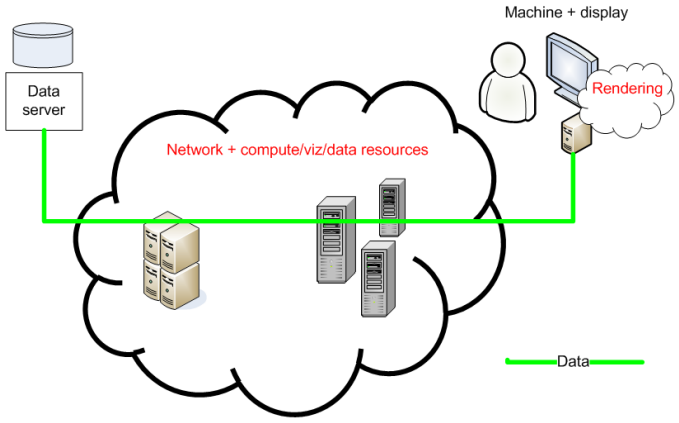


FIGURE 3. Local visualization (data streaming). Here the data is streamed to a visualization client on the local machine for immediate visualization.

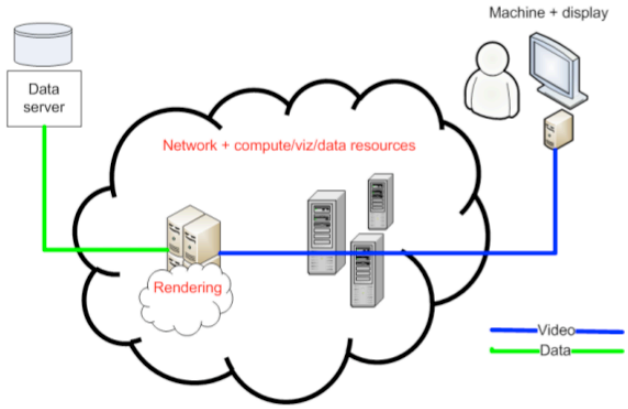


FIGURE 4. Three-way distributed visualization where an intermediate visualization cluster provides the rendering capabilities.

resource [1]. The system supports a wide variety of rendering systems. VNC is an example of remote visualization using video streaming(Figure 2). Web portal assisted client-server visualization: TG visualization gateway [website] enables you to launch a ParaView [8] server, a parallel visualization application for large datasets, on the UC/Argonne TeraGrid Visualization cluster, and connect to it via a ParaView client running on your local resource. The resource allocation can be submitted and processed through the web portal. This is similar with the TACC solution also a case where the rendering is running on the machine local to the data, using video streaming to transport the result images. ParaView also supports a mode where the data server is separated from the rendering server which in turn is located somewhere in the network(like in Figure 4).

VisIt [9, 4] is a visualization software designed to use client-server distribution. The parallelization of data processing on the remote data server allows for large

data sets to be processed. Visit's visualization pipeline can be either server-side rendering and image streaming to client (Figure 2), such as parallel ray-tracing, or server performing data processing and data is transferred to local machine to do interactive accelerated visualization (similar to Figure 3).

Most commonly, the server is as a stand alone process that reads in data from files. An alternative exists where a simulation code can directly deliver data to VisIt. This allows for visualization and analysis of a live running simulation. In the current system, VisIt's server-side component and the simulation are running on the same cluster.

Visapult [5]is a distributed, parallel, volume-rendering application. Visapult's processing pipeline has three components: a raw data source, a viewer, and a visapult back-end reader. The Data source component, usually placed near a distributed parallel storage system, feeds the multi-process visapult back-end using multiple parallel data streams. Each process of the visapult back-end renderer feeds image to user client which combines them using an image-based rendering-assisted volume rendering (IBRAVR) [17]. This approach is similar to the one illustrated in Figure 4, however with a slightly more complex client-side processing module that does more than just displaying an image to the screen.

Semotus Visum [16], is a framework for distributing the visualization pipeline in two components between client and server machines. This framework allows several various server-client configurations in which the visualization stages can either reside within the client, server or shared between both.

gViz [6] developed an XML format to describe the visualization process and grid enabling of modular visualization environments (MVEs) that can be combined in a flexible way. The work on gViz was built upon in a system called eViz whose goal is to provide an adaptive infrastructure for distributed collaborative visualization [7]. Resource selection is made based on information gathered through simulating various resource allocations options.

3. Proposed solution. Discussion

We look at the situation in which the graphics resources available at either location (user or data) are insufficient, and rendering clusters in the network need to be utilized. The results of the rendering process will be streamed in parallel to the user(s). We are exploring various options and alternatives for video streaming, using compressed/uncompressed schemes, hardware-assisted and purely software-based streaming methods.

Realistically, even the graphics capabilities of the most powerful rendering machines today will not be able to interactively render data in terabyte or petabyte range, so we propose an application where instead of transferring the entire file to the rendering machine(s) only particular sections of interest will be transferred. When the data is received, the visualization is updated and the user can move to another section of interest, interactively exploring the dataset [18].

To reduce the amount of data transferred in a single operation and improve the responsiveness of the application "progressive visualization" [14] can also be used where low resolution versions of the data are transferred first and then higher resolution data. Another optimization for progressive visualization is that instead of transferring a large data block in a single operation, the application splits the request into smaller operations. The operation size is an application optimization

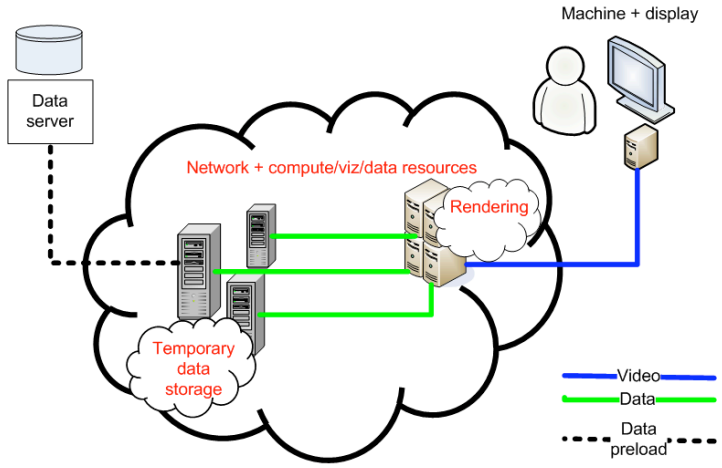


FIGURE 5. Architecture of visualization system which involves a temporary distributed data server allocated on-demand to improve sustained data transfer rates.

choice. There cannot be too many operations as the rendering should be able to keep up with the updates. On the other hand, the blocks cannot be too large as the visualization should be updated as often as possible. Depending on user interaction, the application may also initiate a new set of operations even before the previous set was completed. In consequence, the application will have multiple concurrent remote operations active. The rendering component needs to have the capability to asynchronously update the visualization as new data arrives.

This is already a complex solution that will work in many situations but in addition to parallel remote rendering, we are investigating methods of speeding up the data transfer from the data source to the rendering process. High-capacity, possibly dedicated links connecting the data to the rendering process cannot be efficiently utilized using standard TCP. One direction that we are looking at is supporting experimental protocols in addition to their tuning options so that it will be able to achieve maximum throughput in any network. Another option is to distribute the data in advance into the network and possibly load it in the main memory of compute resources in the network to reduce the load time experienced by the visualization application. A distributed data server can sustain much higher data transfer rates than the single data source shown in Figure 4. Transferring data from the remote memory is faster than transferring it from the disk and this improves the responsiveness of the application as seen by the user. Determining how the data is loaded in advance in order to maximize throughput as experienced by the user is a complex optimization problem but we have proposed an algorithm based on maximum flow that will help us tackle this issue [10].

The architecture of our visualization approach is illustrated in Figure 5. A remote interaction system is necessary in order for the user to be able to connect and steer the visualization. Interaction with the remote parallel renderer is necessary to modify visualization parameters such as the direction of viewing or the level of zoom.

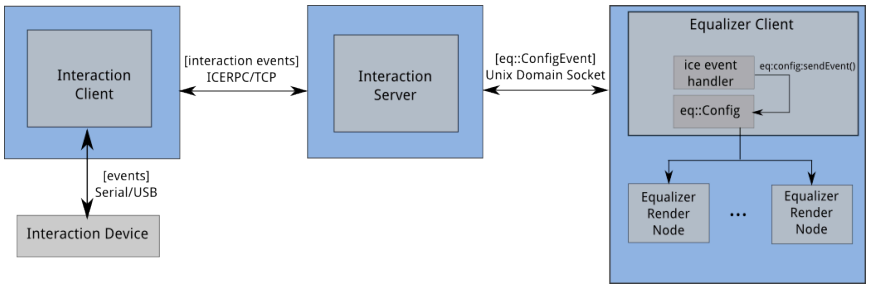


FIGURE 6. Architecture of the remote interaction system.

This architecture can be applied for the visualization of live simulation data as well. Instead of transferring data from a file to the distributed storage(cache), the live simulation can upload the data of interest to the distributed storage. In this case a policy on what data to be discarded and on data priority needs to be applied.

4. Current results. Future plan

We have developed a prototype system that combines remote data access, parallel rendering and remote interaction.

High-performance data transmission over wide-area networks is difficult to achieve. One of the main factors that can influence performance is the network transport protocol. Using unsuitable protocols on wide area network, can result in very bad performance — for example a few Mbps throughput on a 10Gbps dedicated network connection using TCP. TCP can however be very efficient in local area networks with low contention. The application needs to use protocols that are suitable for the network that is utilized. Our remote data access supports configurable transport protocols and currently the UDT [12, 3] library and the standard TCP library are supported. Another issue is blocking on I/O operations. This reduces the performance that is seen by the application and the solution is to use a completely non-blocking architecture. Our system is built to have a pipeline architecture, and blocking on remote data access only happens when the rendering component requires it.

Parallel rendering on HPC or visualization clusters is utilized to visualize large datasets. We have experimented with Equalizer (a parallel rendering framework) [11] and Visit. Visit does not support parallel hardware-accelerated rendering and although it is interactive, we are currently experiencing performance issues when using Equalizer (2 frames per second on an 8-node cluster). The visualization application used in Equalizer is a test application that is supplied with the framework and does not support the progressive visualization scenario or interactive exploration that we described in Section 3 and we will need to implement.

Since the visualization application is not local, an interaction system consisting of three components was developed. The components are a local interaction client running on the local machine, an interaction server running on the rendering cluster and an application plug-in that connects the interaction server to the application and inserts interaction commands in the application work-flow. The architecture is illustrated in Figure 6.

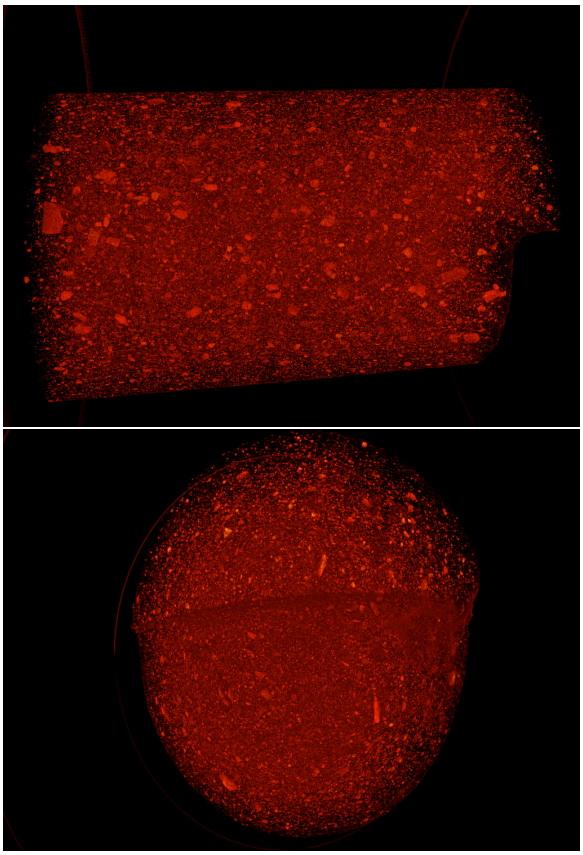


FIGURE 7. Visualization of the tomography data sets.

For our demonstration we used specialized interaction devices developed by the Tangible Visualization Laboratory at CCT that are very useful in long-latency remote interaction systems and can support collaboration (collaborative visualization) from multiple sites.

The final component of the system is the video streaming. Images that are generated from the remote visualization cluster need to be transported to the local client for the user to see. In the past we have successfully utilized hardware-assisted systems running videoconferencing software (Ultragrid) [15] and the VirtualGL system.

4.1. **SC08 experiment.** At Supercomputing 2008 we have shown how our current system can be used to visualize a 3D uniform volume dataset (data resolution $2048\times2048\times2048$ floats) representing a flame retardant distribution in a polystyrene solution acquired using synchrotron x-ray tomography [13]. The size of the dataset used was 2 Gbytes($1024 \times 1024 \times 2048$ unsigned char).

The hardware configuration for the demonstration at the Supercomputing 2008 conference in Austin is shown in Figure 8. A visualization streaming client was available on the show-floor, connected using Internet2 DCN services to LSU in

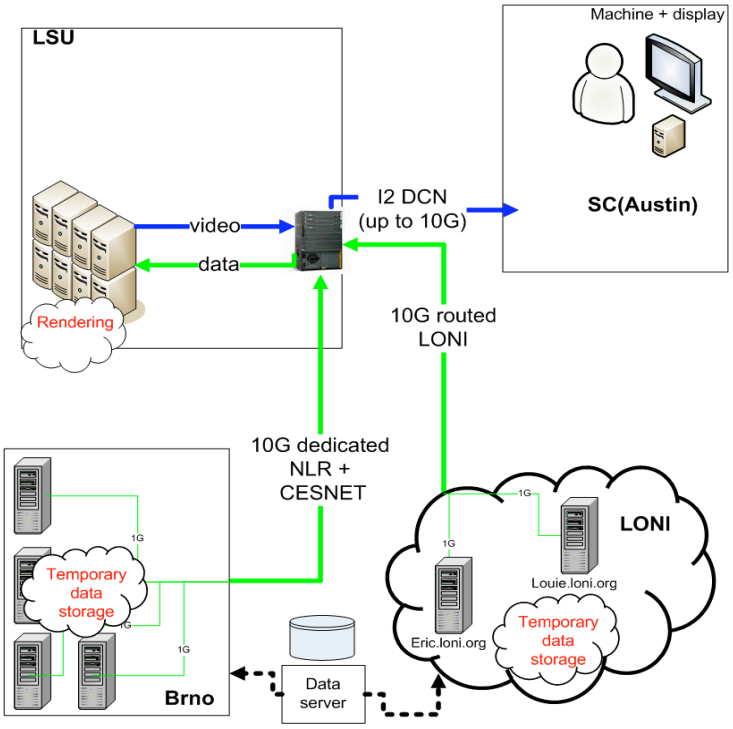


FIGURE 8. Hardware set-up for the SC08 demonstration (arrows show flow of video-blue and data-green)

Baton Rouge. DCN is a new service offered by Internet2, where the user can allocate network bandwidth between points of interest.

A visualization cluster of 8 nodes in Baton Rouge was connected to data servers in Brno, Czech Republic with a 10Gbps dedicated connection provided by NLR and CESNET, and to LONI clusters in Louisiana using the LONI research network. The LONI network provides a capacity of 10Gbps over shared network links. The bandwidth utilization for the data transport from the data servers to the visualization cluster reached a peak pf around 5Gbps (this transfer is very short 2-3 seconds). Video transmission using VirtualGL was using a few tens of Megabits/s.

4.2. **Concluding Remarks. Future work.** We have so far successfully shown the viability of the proposed approach of distributing the visualization system in three components. The visualization system can interactively render large amounts of data, and can retrieve data quickly from the temporary data servers. The requirements for the local system are very low (a display client connected to the network), all the powerful resources are available in the network. We are currently seeing limited frame rate performance and we plan to investigate using other rendering resources and software configurations to improve it.

Regarding the functionality of the system, we will need to move from the test rendering module to a complete visualization solution that we will integrate in Equalizer, or to add data browsing, exploration and asynchronous visualization

update capabilities to the existing rendering module so that the system will be able to visualize more than the initial data and support progressive visualization.

We are planning to integrate software-based video streaming using SAGE [19] for high-performance high-quality video streaming and explore the possibilities of using the system in a multiple-user setting.

5. Acknowledgements

This work was funded in part by the NSF EPSCOR CyberTools project under award #EPS-0701491 and by the Center for Computation & Technology at Louisiana State University, and used the resources of the Louisiana Optical Network Initiative (LONI) and of the CESNET MetaCentrum Project (MSM 6383917201). We would like to thank Prof Les Butler from the Department of Chemistry at LSU for providing the data sets used for this work.

References

[1] Spur user guide: http://www.tacc.utexas.edu/services/userguides/spur/.
[2] Teragrid, web page, documentation, publications: http://www.teragrid.org/userinfo/data/vis/vis_portal.php.
[3] Udt web page, documentation, publications: http://udt.sourceforge.net/.
[4] Visit, web page, documentation, publications: http://visitusers.org.
[5] E.W. Bethel and J. Shalf. Grid-Distributed Visualizations Using Connectionless Protocols. IEEE Computer Society, 2003.
[6] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood. Visualization in grid computing environments. *IEEE Visualization*, pages 155–162, 2004.
[7] K.W. Brodlie, J. Brooke, M. Chen, D. Chisnall, C. Hughes, N.W. John, M.W. Jones, M. Riding, M. Turner, and J.D. Wood. Adaptive infrastructure for visual comuting. 2007.
[8] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote Large Data Visualization in the ParaView Framework. pages 162–170, 2006.
[9] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A Contract Based System For Large Data Visualization. pages 25–25, 2005.
[10] Jr. Cornelius Toole and Andrei Hutanu. Network flow based resource brokering and optimization techniques for distributed data streaming over optical networks. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–8, New York, NY, USA, 2008. ACM.
[11] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola. Equalizer: A scalable parallel rendering framework. In *IEEE Transactions on Visualization and Computer Graphics*, 2008.
[12] Yunhong Gu and Robert L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Comput. Networks*, 51(7):1777–1799, 2007.
[13] Kyungmin Ham, H.A. Harriett, and L.C Butler. Burning issues in tomography analysis. In *Computing in Science & Engineering*, pages 78–81, 2008.
[14] Hans-Christian Hege, Andrei Hutanu, Ralf Kähler, André Merzky, Thomas Radke, Edward Seidel, and Brygg Ullmer. Progressive retrieval and hierarchical visualization of large remote data. In *Proceedings of the 2003 Workshop on Adaptive Grid Middleware*, pages 60–72, September 2003.
[15] Petr Holub, Ludek Matyska, Milos Liska, Lukás Hejtmánek, Jirí Denemark, Tomás Rebok, Andrei Hutanu, Ravi Paruchuri, Jan Radil, and Eva Hladká. High-definition multimedia for multiparty low-latency interactive communication. *Future Generation Comp. Syst.*, 22(8):856–861, 2006.
[16] Eric J. Luke and Charles D. Hansen. Semotus visum: a flexible remote visualization framework. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 61–68, Washington, DC, USA, 2002. IEEE Computer Society.
[17] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. IBR-Assisted Volume Rendering. In *Proc. IEEE Visualization*, pages 5–8, 1999.
[18] Steffen Prohaska, Andrei Hutanu, Ralf Kahler, and Hans-Christian Hege. Interactive exploration of large remote micro-ct scans. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 345–352, Washington, DC, USA, 2004. IEEE Computer Society.

[19] Luc Renambot, Byungil Jeong, Hyejung Hur, Andrew Johnson, and Jason Leigh. Enabling high resolution collaborative visualization in display rich virtual organizations. *Future Gener. Comput. Syst.*, 25(2):161–168, 2009.

[20] J. Shalf and EW Bethel. The grid and future visualization system architectures. *Computer Graphics and Applications, IEEE*, 23(2):6–9, 2003.

Interactive Large-Scale Volume Rendering

R. Parys, G. Knittel
WSI/GRIS, Tuebingen University
72076 Tuebingen, Germany
[parys | knittel]@gris.uni-tuebingen.de

ABSTRACT

We present a system capable of interactive rendering of massive volumetric data sets on a 64 megapixel display wall. The data is compressed down to 0.75 bits per RGB voxel using Residual Vector Quantization and is rendered directly from the compressed representation. We use a decompressed data cache to improve rendering speed by exploiting temporal coherence and to reduce redundant data processing in Screen Space Decomposition algorithm. We use smart load balancing methods to reduce pixel data transfers and network collisions, and to increase cache hit ratios. As an example we compress the full color Visible Human dataset from about 21 GB down to 700 MB, and render it from the compressed representation stored in video memory at interactive frame rates.

Keywords
Distributed and Parallel Graphics, Volume Rendering, GPU Programming

1. INTRODUCTION

Nowadays, volume rendering has become a standard in medical applications and visualization of scientific simulations. Visualization requirements are increasing: data sets and display resolutions are getting larger. This leads to more complex rendering algorithms. In this work we present a quite extreme example: we render a very large data set (about 7.5G voxels) on a high-resolution display wall (65,536,000 pixels). Our display system (see Fig. 6) is built using sixteen LCDs of a resolution 2560x1600 each. A cluster of 16 PCs drives the display wall. Each PC is connected to one LCD, and is equipped with a Intel Core 2 Duo 2.4GHz processor, NVidia Geforce 8800GT graphics card with 1GB of video memory and 4GB of system memory. The PCs are connected via GBit Ethernet.

Todays consumer graphics cards are aproaching 150GB/s of peak memory bandwidth and a teraflop of computational performance. By introduction of a general GPU programming frameworks, GPUs can outperform CPUs in many tasks. In order to achieve the highest performance in the rendering, the most computation-intensive tasks are off-loaded to the GPUs by using NVidia CUDA SDK. While the computing platform presented here is quite powerful, it has a few bottlenecks that can seriously limit the performance and should be avoided. One of the bootlenecks is the slow gigabit ethernet and and the other is the PCIe 1.x bus. Our system is designed to minimize any data traffic send over those mediums.

2. RELATED WORK

In this section we describe work related to our project from the areas of GPU-based volume rendering, parallel volume rendering, and data set compression.

2.1 Data Set Compression

In [6] the application of lossless compression methods for volume data is presented. The authors focus on reducing storage requirements, rather than improving rendering speed. Maximum data reduction reported is 50% for selected data sets.
Vector quantization for volume rendering was first introduced in [18], with some improvements in [22]. The presented system renders directly from compressed data, but nearest-neighbor interpolation limits its rendering functionality.
The use of Block Truncation Coding in space filling way to limit the memory bandwidth was introduced first in [12].
In recent years Wavelet-based coding has received most attention [9],[17],[20],[21]. Some extensions based on a hierarchical wavelet representation of large datasets is used in [7]. The claimed compression rate without noticeable artifacts in the image was 30:1. The authors minimize the number of voxels processed by deriving a quality measure from the wavelet representation and achieve interactive rendering speeds for large data sets on standard PCs. Decompression is done on the CPU, and sending uncompressed voxels to the graphics card over the bus can seriously limit performance (see Table 1).

2.2 GPU-based Volume Rendering

The graphics hardware was used for volumetric rendering in [1], [3] and [4]. Data was kept in a 3D texture and screen-aligned slices were drawn with properly computed mapping coordinates and blended together. Later the visual quality was improved with gradient shading [15], multi-dimensional transfer functions [11], pre-integrated transfer functions [5], and the processing of pre-segmented data sets [8]. We integrate these state of the art methods into our system, where it is possible and useful. Although,

some features have lower priority, for example data segmentation is not included due to the large effort required for this task. This feature may be supported in later versions.

2.3 Parallel Volume Rendering

As described in Section 3.3, parallel rendering can be implemented in two ways: object-space partitioning and screen space partitioning. Object-space partitioning usually is limited by alpha blending of the intermediate images. Solutions are proposed in [14], [23] and [24]. In [24], it is pointed out that the CPU is used that for alpha-blending, instead of the much better suited GPU. Although, GPU can be used to speed up the computation, large data streams are still difficult to handle in the network.

Isosurface rendering on a display wall of about 63M pixels is described in [16]. Example shown for isosurfaces build from 470M triangles, the rendering takes about 15 seconds.

3. THE GIGA-VOXEL SYSTEM

In our rendering system we off-load all time consuming computations to the GPU. This choice is supported by a few benchmark figures. Results obtained on a Dell XPS700 workstation, equipped with an Intel Core 2 Duo CPU at 2.13GHz and an NVidia GTX280 (optionally an 8800GT) are summarized in Table 1. To measure the bandwidth we have used "bandwidthTest" from the NVidia CUDA SDK [19].

Test	Bandwidth
CPU ↔ Cache	98,520
CPU ↔ Memory, 16MB Blocks	2,100
CPU ↔ Graphics Card (PCIe 1.x)	1,500
GPU ↔ Video Memory GTX280	110,028
GPU ↔ Video Memory 8800GT	43,357

Table 1: Bandwidth Measurements [MB/s]

Interesting measurement is the internal CPU cache bandwidth and the bandwidth to the external video memory on the GTX280. As it can be seen, the latter one is better. Our design target was set to keep all necessary data locally in video memory and to avoid frequent transfers of big amounts of data between GPU and GPU. The GPU is used for all compute-intensive tasks like decompression, shading and ray casting. Each cluster node must have a copy od the data in order to reduce network traffic to minimum. When the size of typical data sets it taken into account, it becomes clear, that a compression scheme must be employed. There are contradicting requirements for the compression scheme: high compression rate at high quality and extremely fast decompression time. An interesting candidate is *Residual Vector Quantization* (RVQ).

The data needs to be compressed in an offline processing step. This step is performed only once. Later the compressed data is reused for rendering. Compression and rendering is described in the following subsections.

3.1 Residual Vector Quantization

Residual Vector Quantization has been initially described in [10]. RVQ and related techniques are described in [2]. RVQ is based standard vector quantization (VQ). In VQ, a set of vectors is represented by a smaller set of vectors called *codevectors*, that is minimizing overall error. Usually, clustering methods like *k-means* [13] are used to compute a set of codevectors called a *codebook*. K-means is starting from an initial set of random codevectors (*seeds*), and assigns each vector to its nearest codevector in order to create clusters. In next step, the codevectors are moved to the center of gravity of their cluster. This step is repeated until the codevectors no longer move. Each vector is represented by a pointer to its codevector in the codebook. Decompression step for a vector is returning a codevector pointed by the pointer.

In case of the large or unknown (at the time of codebook construction) set of vectors, a subset of vectors (*training set*) can be used to create the codebook. Vectors from the outside of the training set are replaced by a pointer to its nearest neighbor vector.

For RVQ the data set is decompressed and for each original vector, the error vector is computed. The set of error vectors is compressed with VQ, to obtain the second set of pointers and the second codebook. This process is repeated for the desired number of levels. Compressed vector is represented by a set of pointers to codebooks. To decompress such a vector it is necessary to add the codebook vectors associated with a compressed vectors.

The code length for RVQ is defined by number of bits required to represent a set of pointers (indices) to the codebooks.

Experiments for large number of images have shown, that larger codebooks should be preferred over a high number of levels in order to achieve higher PSNR. We have performed a number of tests showing that good quality can be achieved with 4 levels and 4096 codebook vectors per level, what gives a codelength of 48 bits.

We use "Visible Human Female" color (RGB) data set for our tests. Each voxel is defined by 3 channels, 8 bit each. We create vectors of dimension 192 from 4×4×4 voxel cubes. The 64 voxels in a cube are represented with 48 bits, giving a compression rate of 32:1, or 0.75 bits per voxel.

We store codebook vectors in higher precision to decrease the influence of rounding errors. 32-bit values used to represent one codebook vector voxel. 11 bits are used for red and green, and 10 bits for blue. One codebook vector is 256 bytes, and the whole codebook for 4 levels takes 4MB of video memory.

3.1.1 Compressing the Visible Human Female

Visible Human data set is available for download as a set of images of a resolution of 2048×1216 pixels [25]. There are 5189 images. The body was frozen in a blue gel. We treat this as "empty space", so we had to remove it from the data set. We have cropped the images to a final resolution of 1608×896 pixels,

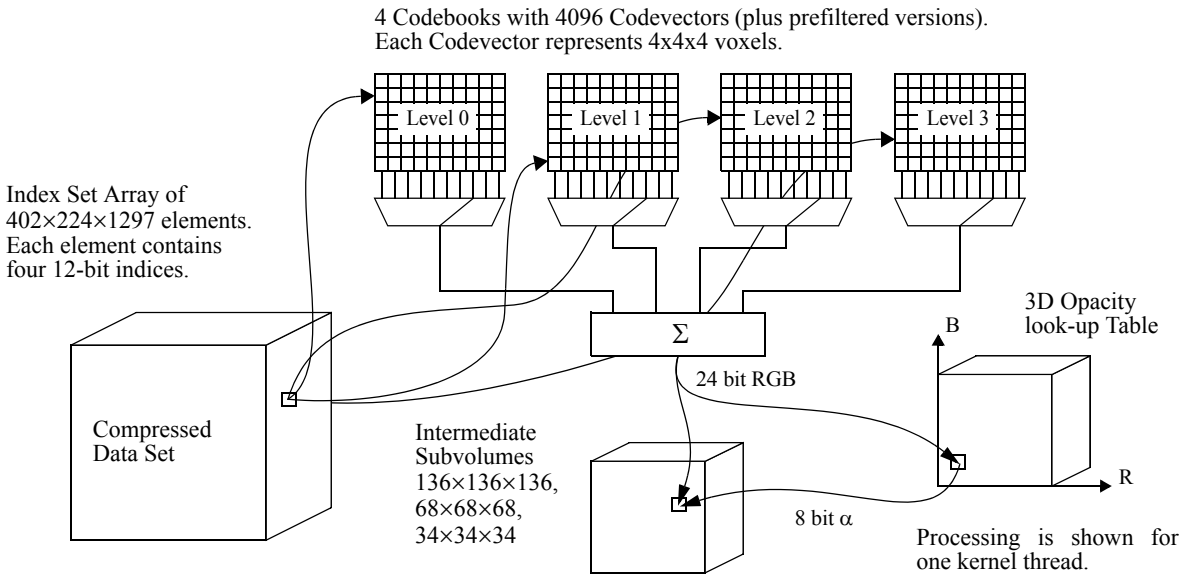


Fig. 1: Decompression and classification

because there is too much of empty space. The total input data size is 20.9GByte.

In order to limit the compression time, so we have selected a training set equivalent in size to 300 images. Training phase to obtain four codebooks took about 21 hours on eight-core machine. Codebook construction time can be improved significantly performing nearest neighbor searches in parallel.

Reusing the codebook to compress the data set took additionally about 10 hours.

We have computed the quality of decompressed data set in terms of PSNR. We didn't include 4×4×4 cubes that are forming the empty space. The overall PSNR is about 27dB. An example of an original image versus the decompressed image is shown in Fig. 2a and b.

The compressed data is an array of 402×224×1297 = 116,792,256 index sets of 48 bits each, for total size of 700,753,536 Bytes. The entire compressed data set along with the codebooks fits on a graphics card with 1GByte of video memory. We only consider the case that the compressed dataset fits completely into the video memory. On other case swapping from main memory or even hard disk would be required.This also would benefit from the high compression rate.

3.2 Rendering

The dataset is divided into subsets of subvolumes with the same Manhattan distance. We render each previously mentioned subset to an off-screen buffer and blend resulting image with the framebuffer. To render a subset of subvolumes we repeat steps of decompressing a subvolume from this set into a 3D texture buffer and rendering the 3D texture using ray-casting. In the decompression step we integrate classification using 3D lookup table. Optionally gradient extraction and shading is also integrated into the decompression step in order to not slow down the ray caster. We use early ray termination on a per-ray basis. We use occlusion culling for an early exit on a subvolume basis. We apply empty space skipping to subvolumes after classification. When a visible contribution of a subvolume is under a user-supplied threshold, the subvolume is discarded from rendering. This test is done according to the actual transfer function. Multi-resolution rendering is integrated in an elegant way - we downsample the codebooks to create additional two levels of detail. In the following subsections we present individual steps of the rendering pipeline in detail.

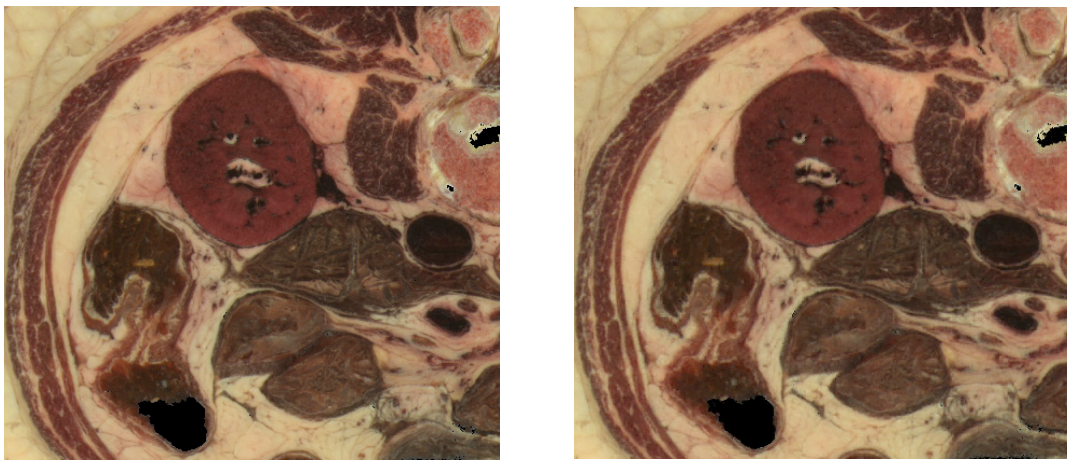
3.2.1 Decompression

The decompression code is implemented with NVidia CUDA technology. We make use on-chip shared memory buffer in order to reduce transfers from video memory. Processing is as follows.

We load 256 index sets (worth 16k voxels) into the shared memory. Each index set consists of 48 bits, which are unpacked into four 16-bit indices again into shared memory. For each voxel to be generated, there is one thread in the kernel. Each thread reads from memory those elements of the codebook vectors which it needs for its voxel. The codebook vector element is unpacked from R11G11B10 format (see Section 3.1), and accumulated in shared memory.

Unpacked RGB values of a voxel are used to access a 3D lookup-table with opacity (α) values. The α-value is again written into the shared memory, which completes the voxel generation. When a certain number of voxels is decompressed, they are written to an intermediate 3D texture. We take care that memory transfers are mostly large bursts, to maximize the bandwidth. Decompression performance is 1.86G voxels/s on the GTX280, and 0.60G voxels/s on the 8800GT.

We partition the volume into subvolumes of 128×128×128 voxels. We extend each subvolume in each direction by two layers of 4×4×4 voxels, so the



a.) b.)

Fig. 2:a.) Original image. b.) Decompressed image.

final subvolume size is 136×136×136. In this way we solve the problem of missing voxels at boundaries for the reconstruction filter in the ray caster (tri-linear interpolation) and during gradient extraction (see Section 3.2.2). In this way we add the overhead of about 20%. Decompression performance for different levels of detail is summarized in Table 2. Overall decompression and classification flow is presented on Fig. 1

3.2.2 Gradient Extraction and Shading

After a subvolume has been decompressed, our system can perform gradient shading as an option. We compute the gradient from the opacity, because steep changes in opacity represent the surfaces.

GPU	Level	Voxels/s	Subvolumes/s
8800GT	0	0.60G	254
GTX280	0	1.86G	776
8800GT	1	0.20G	716
GTX280	1	0.72G	2463
8800GT	2	0.03G	841
GTX280	2	0.06G	1667

Table 2: Decompression Performance

We use a variant of a 3×3×3 Sobel filter (see Fig. 3). We need to address two problems:

- high computation costs due to the large kernel,
- a certain amount of noise still in the volume.

Both problems are solved by using downsampled versions of the subvolume for gradient estimation (see also section 3.2.5, Multi-Resolution Rendering). We generates two additional levels of detail: a 68^3, and a 34^3 subvolume. We compute the gradients only on the lowest-resolution grid, directly on the GPU, using a CUDA-kernel. Performance is given in Table 3.

GPU	Gradients/s	Subvolumes/s
8800GT	17.8M	570
GTX280	63.9M	2045

Table 3: Gradient Estimation Performance

We do not affect the raycaster performance by the shading operation, because gradient extraction and shading are done at the voxel positions, and the contributions from specular reflection are added to the just decompressed RGB-quantities. The decompression speed does not suffer too much because of the regular memory access pattern.
For gradient extraction the system can use a Central Difference (CD) operator. Gradient shading is again implemented as a CUDA-kernel in a way that each thread processes one voxel. Each thread reads a certain subset of the required voxel neighborhood, so that by the end of this step a large block of voxels resides in shared memory.
For the performance reasons we assume that light sources are located at infinity and a constant viewing direction throughout the subvolume (only for the shading, not for the raycasting). Thus, we do not need to recompute the halfway vectors for each voxels. This approximation does not produce disturbing effects caused by misplaced highlights. Exponentiation is done by a look-up in a precomputed table. Gradient shading speed for CD and one light source is summarized in Table 4.

3.2.3 The Raycaster

We use the raycaster from the NVidia SDK. No attempt was made to optimize this code, because it is not the scope of this work.
The average rendering time of this raycaster was measured 3.97ms per subvolume (on the GTX280 graphics card), with early ray termination disabled. It is about 4 times slower than the pure decompression. Recurring decompression can be tolerated fairly well,

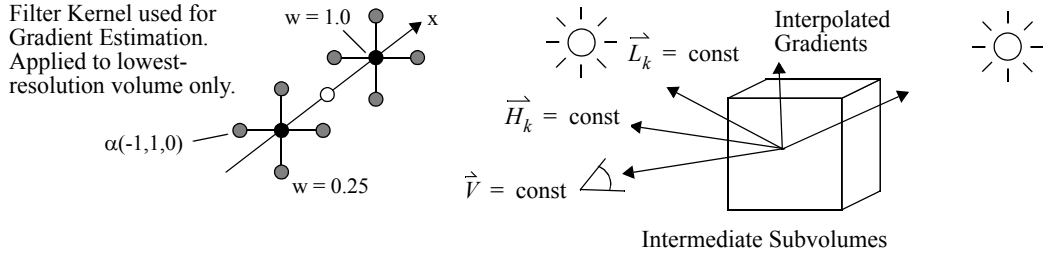


Fig. 3: Gradient shading

GPU	Level	Voxels/s	Subvolumes/s
8800GT	0	0.227G	111
GTX280	0	0.411G	201
8800GT	1	0.309G	966
GTX280	1	0.546G	1712
8800GT	2	0.271G	7426
GTX280	2	0.482G	13158

Table 4: Gradient Shading Performance (CD)

because most of the rendering time was spent in ray casting.

3.2.4 Blending and Occlusion Culling

The subvolumes are rendered in front-to-back order. We divide subvolumes into non-overlapping subsets (in screen space) This is done by sorting subvolumes according to their Manhattan Distance to the viewer. The result of the rendering of one subvolume subset is a private frame buffer of RGBα-values. This buffer is α-blended with the compound frame buffer, which in the end contains the final image.

During blending, a Z-buffer is also updated. Whenever the α-value of a pixel in the compound frame buffer exceeds a threshold, the corresponding entry in the Z-buffer is set to Z-front. This is then used to exclude subvolumes which are occluded by opaque structures in the data set from decompression and rendering. To this end, an OpenGL occlusion query is submitted with the bounding box of the subvolume, which returns the number of visible pixels. Depending on a user-defined parameter, the subvolume is rendered or discarded.

Occlusion queries can be accelerated by submitting a set of bounding boxes. In our system, all subvolumes with the same Manhattan Distance could be rendered in parallel and in any order, so they are queried in one batch. We exclude subvolumes which are located at any of the visible faces of the entire volume from the occlusion query. Blending process in illustrated on Fig. 4 (a).

3.2.5 Multi-Resolution Rendering

As mentioned before, a downsampled version of the subvolume can be generated from a *downsampled version of the codebooks*. Thanks to this fact there is no need to store separate index sets for each level of detail in video memory. We need only a small amount of extra memory for storing downsampled codebooks. Different resolutions of subvolumes are used to avoid subsampling of the data during ray casting and to speed up the rendering of distant sub volumes.

The system can decompress subvolumes with 136, 68, and 34 voxels along each axis. The raycaster automatically performs proper voxel access and filtering by using normalized texture coordinates. Only the opacity must be adjusted, we accomplish this by using a separate 3D look-up table for each resolution.

3.3 Parallel Rendering on the Cluster

There are two possible ways of distributing work among rendering nodes: object-space partitioning (OSP) and screen-space partitioning (SSP). For the first method, a workpackage is a subvolume that is rendered into a private frame buffer. A subvolume can be visible on more than one display, so it may be necessary to send a subset of pixels over the network to the receiving node. This subset of pixels is then blend into local composite frame buffer on the target node. Since the contribution of many subvolumes may influence one pixel, multiple transfers over the network may be required to compute final color. Many advanced algorithms have been designed to optimize this operation [14], [23], but for slow GBit Ethernet it may be still a bottleneck. The advantage of this approach is that each subvolume is processed only once.

In SSP, the screen is split into a set of tiles. A machine that is assigned to a certain tile, renders all subvolumes contained in the tile view frustum and sends the final pixels to the destination screen. We have selected this method, because in our setup it will give higher frame rate. Often it may happen that a subvolume is visible on multiple tiles – in this case it has to be processed multiple times. In case of ray casting there is no redundant processing, but the decompression code can generate only complete subvolumes.

Since the viewport can be in arbitrary location, each node needs to store complete copy of the data set. In the cluster we have only 1GB of video memory per graphics card, the compressed dataset takes about 700 MB. We use high resolution off-screen buffers for blending and a certain amount of video memory is allocated for internal OpenGL data structures.

3.3.1 Subvolume caching

For a significant performance improvement, a caching scheme for decompressed subvolumes is used. In order to reduce multiple processing of the same sub-

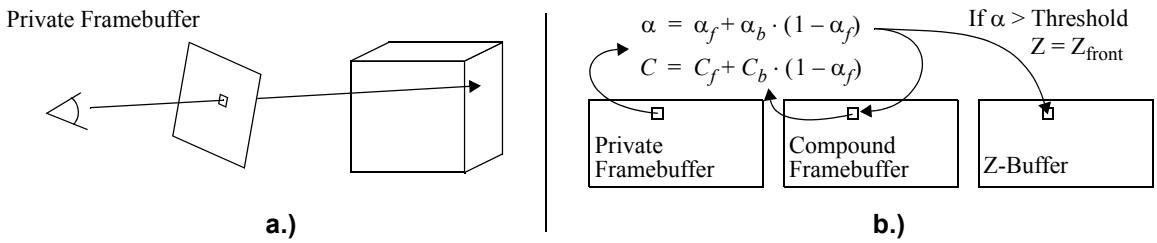


Fig. 4: Ray casting (a) and alpha-blending (b)

volumes by the decompressor we cache 128 most frequently used subvolumes. We use caching in a fast preview mode only, when lowest level of detail is used. On higher levels of detail, decompressed subvolumes consume too much memory. Preview mode is used for camera motion, and we sacrifice rendering quality for speed. Caching scheme has proven to be very efficient also in case of the camera motion, when temporal coherency is exploited. The effect of caching on the rendering speed in preview mode is sumarrized in Table 5. For a first frame of the rendering, when temporal coherency cannot be exploited yet, we achieve an average hit ratio of 58%. For consequent frames, when temporal coherency can be exploited we achieve an average hit ratio of 89%.

Rendering method	Cache hit ratio	Time(s)
no caching	-	0.50
caching	58%	0.39
caching with temporal coherency	89%	0.25

Table 5: Impact of caching on the rendering speed.

3.3.2 The Tile Manager

Further performance optimization are achieved by a smart load balancing scheme based on dynamic scheduling and a set of work assignment rules.

Assignment of tiles to the rendering nodes is done on demand. A tile management thread (*tile manager*) is running on one of the cluster nodes. When a node needs to have a tile assigned it is sending request to the tile manager and as a response gets a tile number to render. Since remote rendering causes network data transfers, the tile manager is using some heuristics during tile assignment to reduce network traffic and avoid network collisions. In the Ethernet, network collisions can happen when a few nodes send data to the same machine at the same time. It can cause large delays and it can limit target machine network bandwidth. The tile manager uses the following rules for a tile assignment to a rendering node: (1) At first it tries to assign a local tile to a rendering node. (2) If it is not possible, it tries to assign a tile that was assigned to this node in previous frame. (3) If it is not possible, it tries to assign a tile that is located on a different machine than previously assigned tile. The first rule tries to eliminate network traffic. Rules (1) and (2) improve cache hit ratio for local and remote rendering. Rule (3) reduces per-node network traffic and collisions for remote tiles, by trying to guarantee that tiles rendered in parallel on different machines are not send to the same node. The tile manager runs 16 threads that are accessing a shared data structure holding information about the tile assignment. Threads are synchronized with a critical sections. Only one thread at a time can access the shared structure, while other threads waiting to enter the critical section are descheduled. This approach does not overload the CPU.

Each machine is running one rendering thread and 15 threads used for receiving pixel data from the network. Rendered tiles are received in background of the rendering process and they do not affect the rendering speed on our dual core machines. Rules used by the tile manager try to guarantee, that number of threads receiving data at the same time is minimized. At the end of the frame, each node checks if it has all tiles. After receiving a synchronization signal, each node plots its tiles to screen. In the preview mode tiles are rendered in lower resolution by shooting one ray per 4×4 pixel block, so the amount of data transferred is reduced 16 times. As an option rendered tiles can be compressed before sending over the network. The image compression is described in the following subsection.

3.3.3 Real-Time Image Compression and Decompression

We have implemented a variant of S3TC compression algorithm. The build-in texture compression functionality of OpenGL is not suitable to use in our case. While there is a dedicated S3TC decompression unit on the graphics hardware, the process of compression is implemented on the CPU and it is slow. In other applications it usually does not have an impact on the performance, because textures are compressed just once, and they are reused many times during the application run time. However in our case every frame we need to compress tiles before sending them off and we cannot afford an expensive compression algorithm. The other disadvantage of original S3TC is a compression ratio of 4 bits per pixel.

We have implemented a variation of S3TC algorithm directly on the GPU using NVidia CUDA technology. It has the advantage of massively parallel processing and access to the data stored on the GPU. Rendered tile is kept in video memory in a frame buffer object. With current implementation of CUDA it is not possible to use frame buffer object directly. Frame buffer

Fig. 5: The original and the decompressed image

has object to be copied to a pixel buffer object before CUDA kernel can process the rendered data. However it is much faster to copy within video memory than between system and video memory.

Our implementation of S3TC algorithm is compressing blocks of 4×4 pixel in a lossy manner. An average luminance L of a pixel block is computed. Then the average luminance is used to divide pixels into two sets. Pixels with higher luminance then L are assigned to the first set and the remaining pixels to the second set. For each set an average color is computed. In a compressed representation we store two average colors and for each pixel we store value pointing to one of the colors in a lookup table. Each color is stored in R5G6B5 format, what gives 16 bits, and the lookup table stores 16 one-bit values. Finally 48 bits are needed to represent a tile of 4×4 pixels giving a ratio of 3 bits per pixel. Performance figures for compression and decompression are presented in Table 6. The GPU times shows only kernel times and do not include memory copy time.

	CPU	GTX280	8800GT
compression	35.35 ms	0.18 ms	0.55 ms
decompression	4.65 ms	1.10 ms	0.70 ms

Table 6: Image compression and decompression speed.

Comparison between original and decompressed image is presented on Fig. 5

4. PERFORMANCE

A photo of the display wall showing a rendering of the Visible Human Female is shown in Fig. 6. The average rendering speed for different levels of detail and without caching is summarized in Table 7. The column "Res" shows the tile resolution level, for example 'f' means full resolution, 'f/4' means 1 ray for a pixel block of 2×2, and 'f/16' means 1 ray for a pixel block of 4×4. In the preview mode, one ray was shot for each 4×4 pixel block and the dataset was decompressed at lowest level of detail.

LOD	Res	Shading	Time (s)
0	f	n	2.6
1	f	n	1.4
2	f	n	1
0	f/4	n	1.7
1	f/4	n	0.8
2	f/4	n	0.55
0	f/16	n	1.4
1	f/16	n	0.6
2	f/16	n	0.25
0	f	y	3.7
1	f	y	1.7
2	f	y	1.4
0	f/4	y	2.5
1	f/4	y	1
2	f/4	y	0.9
0	f/16	y	1.8
1	f/16	y	0.7
2	f/16	y	0.6

Table 7: Rendering speed

5. CONCLUSIONS

Our system for parallel volume rendering presented in this paper provides the following features:

- Rendering massive data sets in a very high resolution on a display wall,
- storing massive data sets completely in video memory and replication among rendering nodes by using effective compression down to 0.75bpp,
- very fast decompression done by the GPU,
- classification and gradient shading done on-the-fly during rendering,
- empty-space-skipping and occlusion culling on a per-subvolume basis,
- multi-resolution rendering,
- load balancing of parallel rendering by screen-space partitioning.

Rendering of Visible Human data set is done at interactive speed directly from compressed data. Lossy

compression preserves small details of the anatomy, like blood vessels.

6. REFERENCES

[1] K. Akeley, "*RealityEngine Graphics*", Proc. ACM Siggraph 93 Conference, pp. 109-116

[2] C. F. Barnes, S. A. Rizvi, *"Advances in Residual Vector Quantization: A Review"*, IEEE Trans. on Image Processing, Vol. 5, No. 2, 1996

[3] B. Cabral, N. Cam, J. Foran, "*Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware*", Proc. ACM Symposium on Volume Visualization 1994, pp. 91-97

[4] U. Cullip, U. Neumann, "*Accelerating Volume Reconstruction with 3D Texture Hardware*", UNC Tech Report TR93-0027, 1993

[5] K. Engel, M. Kraus, T. Ertl, "*High-quality pre-integrated volume rendering using hardware-accelerated pixel shading*", Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2001

[6] J. E. Fowler, R. Yagel, "*Lossless Compression of Volume Data*", Proc. ACM Symposium on Volume Visualization 1994, pp. 43-50

[7] S. Guthe, M. Wand, J. Gonser, W. Strasser, "*Interactive Rendering of Large Volume Data Sets*", Proc. IEEE Visualization Conference 2002, Boston, MA, pp. 53-60

[8] M. Hadwiger, C. Berger, H. Hauser, "*High-quality two-level volume rendering of segmented data sets on consumer graphics hardware*", Proc. IEEE Visualization Conference 2003, pp. 301-308

[9] I. Ihm, S. Park, "*Wavelet-based 3D compression scheme for very large volume data*", Proc. Graphics Interface 1998, pp. 107-116

[10] B. H. Juang, A. H. Gray, *"Multiple Stage Vector Quantization for Speech Coding"*, Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, Vol. 1, Apr. 1982, pp. 597-600

[11] J. Kniss, G. Kindelmann, C. Hansen, "*Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets*", Proc. IEEE Visualization Conference 2001, pp. 255-262

[12] G. Knittel, *"High-Speed Volume Rendering Using Redundant Block Compression"*, Proc. IEEE Visualization '95 Conference, Atlanta, GA, October 29 - November 3, 1995, pages 176-183

[13] S. P. Lloyd, *"Least squares quantization in PCM"*, IEEE Trans. on Information Theory, Vol. 28, 1982, pages 129-137

[14] K.-L. Ma, J. S. Painter, C. D. Hansen, M. F. Krogh, "*Parallel volume rendering using binary-swap compositing*", IEEE Computer Graphics and Applications, Vol. 14, No. 4, 1994, pp. 59-68

[15] M. Meissner, U. Hoffmann, W. Strasser, "*Enabling Classification and Shading for 3D Texture Mapping Based Volume Rendering*", Proc. 10th IEEE Visualization Conference 1999 (VIS '99), p. 32

[16] K. Moreland, D. Thompson, "*From Cluster to Wall with VTK*", IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003, October 20-21, 2003, Seattle, Washington, USA, pp. 25-31

[17] K. Nguyen, D. Saupe, "*Rapid high quality compression of volume data for visualization*", Computer Graphics Forum 20, 13, 2001

[18] P. Ning, L. Hesselink, "*Vector Quantization for Volume Rendering*", Proc. ACM Workshop on Volume Visualization 1992, Boston, MA, pp. 69-74

[19] NVIDIA Corporation, "*CUDA Zone*", http://www.nvidia.com/object/cuda_home.html#

[20] F. Rodler, "*Wavelet based 3D compression with fast random access for very large volume data*", Proc. Pacific Graphics 1999, pp. 108-117

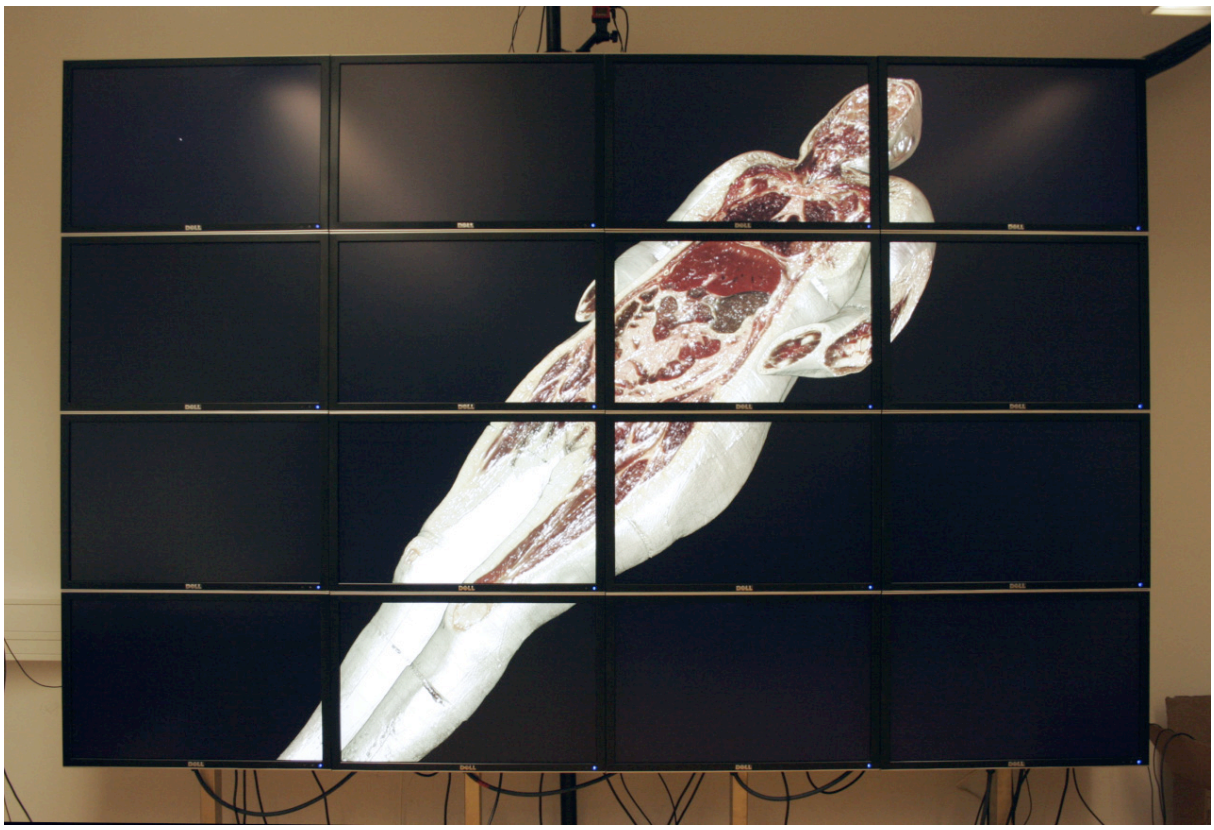
[21] S. Roettger, S. Guthe, D. Weiskopf, T. Ertl, W. Strasser, *"Smart Hardware-Accelerated Volume Rendering"*, Proc. EG/IEEE TCVG Symposium on Visualization 2003, pp. 231-301

[22] J. Schneider, R. Westermann, "*Compression Domain Volume Rendering*", Proc. 14th IEEE Visualization Conference 2003 (VIS'03), p. 39

[23] A. Stompel, K.-L. Ma, E. B. Lum, J. Ahrens, J. Patchett, "*SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering*", IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003, October 20-21, 2003, Seattle, Washington, USA, pp. 33-40

[24] M. Strengert, M. Magallon, D. Weiskopf, S. Guthe, T. Ertl, "*Hierarchical Visualization and Compression of Large Volume Datasets Using GPU Clusters*", Proc. EG Symposium on Parallel Graphics and Visualization 2004, pp. 41–48

[25] United States National Library of Medicine, "*The Visible Human Project*", http://www.nlm.nih.gov/research/visible/getting_data.html



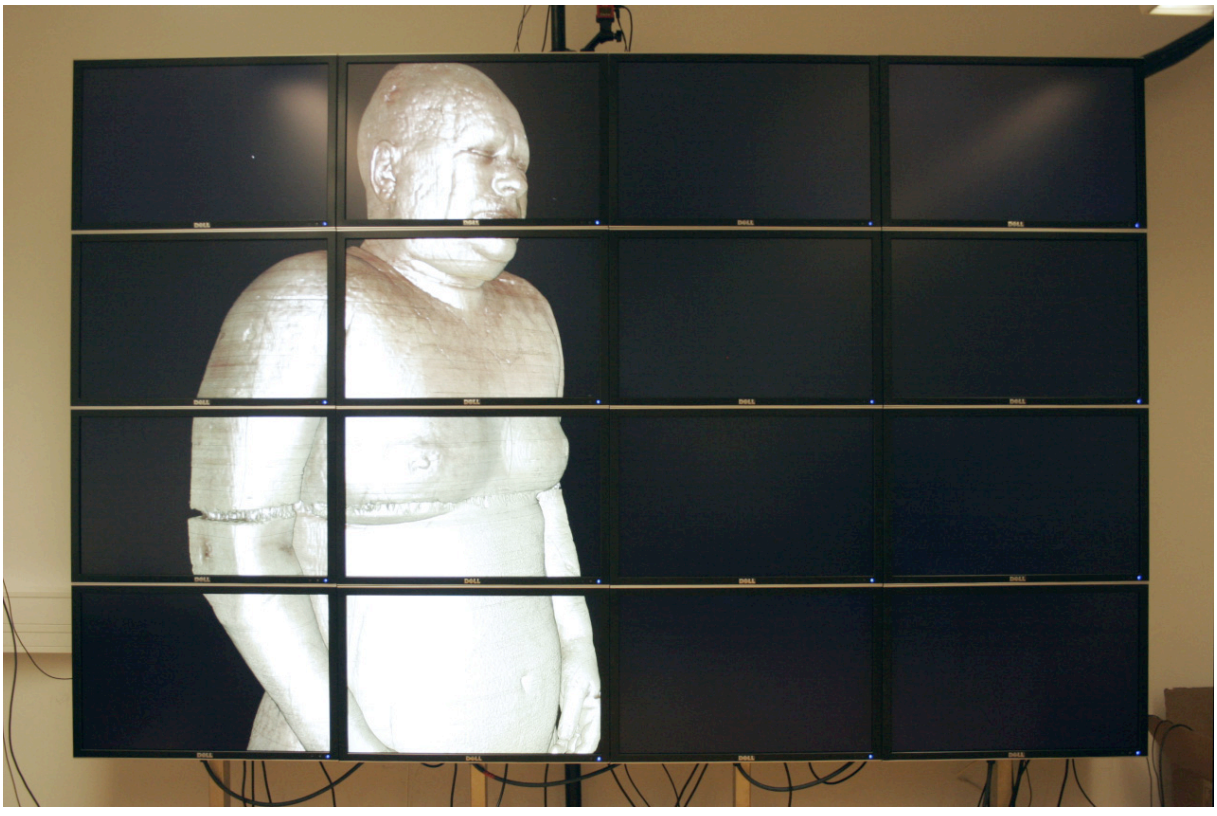


Fig. 6: The Display Wall.

Remote Rendering of Large Biological Datasets

Wolfram Schoor[1,3] and Marc Hofmann[1] and Simon Adler[1,3] and Werner Benger[2] and Bernhard Preim[3] and Rüdiger Mecke[1]

[1]Fraunhofer Institute for Factory Operation and Automation, Germany

wolfram.schoor,marc.hofmann,simon.adler,ruediger.mecke@iff.fraunhofer.de

[2]Center for Computation and Technology, Louisiana State University, USA

werner@cct.lsu.edu

[3]Otto von Guericke University Magdeburg, Germany

preim@isg.cs.uni-magdeburg.de

Abstract

This paper presents remote rendering technologies to support the analysis and exploration of large biological datasets evaluated for a medium sized biology research institute. Reconstructed three-dimensional models and other data domains stored in a database are the foundation for the visualization. Remote Rendering via a web-based client enables a widespread access to the data with interactive frame rates and high-quality rendering results without the demand of new client hardware. Moreover, the rendering server has the capability to manage multiple independent or collaborative sessions and creates only a very small network traffic over time. Two worst case scenarios were tested i) a network connection with very small bandwidth, and ii) a thin graphical server solution for rendering.

1 Introduction

Remote Rendering commonly means interactive visualization of three-dimensional datasets via a client/server architecture over a network. Visualization of complex 3D models is still a task hard to accomplish without specialized hardware. Furthermore, ad-hoc visualizations over distance are playing a key role in future visualization trends. One problem of visualization is that huge amounts of data must be handled, which requires not only a fast CPU and high memory capacities (RAM), but also a high degree of graphics output manageable by modern graphics cards.

These resources cannot be covered and the data amount is increasing faster than processors [Moore, 1998]. Another aspect is that the data must be stored somewhere, and a heterogeneous data storage on different clients is not acceptable.

These drawbacks can be overcome by the concept of remote rendering, also called remote visualization. The datasets are computed on a rendering machine with high graphics capabilities - the so-called *server*, and only rendered images are sent to the working station - called *clients*. For remote visualization purposes the following key issues denote indicators of quality.

- Network latency
- Network throughput
- Error recovery
- Security issues
- Hardware/software incompatibility
- Graphics performance (fps)
- Scalability (user number)
- Network robustness
- Balance (local/remote resources)

The work presented here is motivated by a given shortcoming regarding biological model visualization, namely limited bandwidth, slow clients with poor graphics capabilities, large biological 3D models and the demand to real-time interaction. The latter cannot be covered by existing conventional remote rendering techniques. Therefore, a remote rendering architecture with reduced network traffic generating correct high-quality renderings for low level end user PCs with interactive frame rates is given in this paper.

2 Related Work

This section briefly shows the stages of remote rendering techniques from the past to today, presents a possible classification of remote rendering applications dealing with polygonal models, and introduces established applications. The treatment of other model representation forms will not be part of this paper. Nevertheless, the principle way of converting these representations into a polygonal representation as pre-step can be performed.

2.1 Historical Outline

The idea of remote rendering/visualization is not new. Various remote rendering applications have been developed in the past.
One of the most popular remote rendering applications for 2D systems was the X Window protocol [Scheifler & Gettys, 1986] for X systems. The extension to transfer 3D primitives is the OpenGL Stream Codec (GLS) [Dunwoody, 1996] introduced by SGI.
A standard technique for remote display of 3D applications is GLX [Womack & Leech, 1998], the "OpenGL Extension to the X Window System"[1]. Using GLX for remote rendering, GL commands are encoded by the client side's API and sent to the X server within GLX packages. These commands are decoded by the X server and submitted to the OpenGL driver for rendering on graphics hardware at the server.

[1] The pendant for Windows operating systems is *WGL* and *CGL* for Mac OS X respectively.

2.2 Classification

Adapted from [Schmalstieg & Gervautz, 1996], remote rendering techniques of 3D geometry models can be classified as follows:

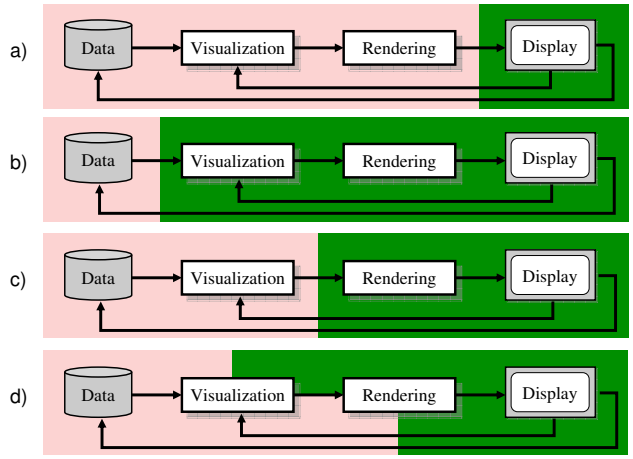


Figure 1: Remote rendering types a) image-based, b) geometry replication, c) immediate-mode, and d) hybrid rendering

1. *Image-based:*
   Rendering is performed by the sender, and the resulting stream of pixels is sent over the network as provided in [Scheifler & Gettys, 1986] (see Figure 1 (a)).

2. *Geometry-based:*
   A copy of the geometric database is stored locally to be accessed by the rendering process. The database can either be available before application start (kept on local hard disk), or downloaded just before usage, such as VRML / X3D browsers can do [Gueziec et al., 1999], [Web3D Consortium, Inc., 1998] and [Web3D Consortium, Inc., 2004] (see Figure 1 (b)).

3. *Immediate-mode drawing:*
   The low-level drawing commands used by drawing APIs are issued by the application performing the rendering. They are not immediately executed, but sent over the network as a kind of remote procedure call. The actual rendering is then performed by the remote machine (e.g. distributed GL [Shreiner et al., 2005]), see Figure 1 (c).

4. *Hybrid rendering:*
   In this approach a 3D geometry as low resolution representation is rendered on the client's side and the server transmits images to the client as well. The used mode depends on the performed user action (computational vs. network load) [Koller et al., 2004] or is dynamically adapted to the available network capacities (see Figure 1 (d)).

Another quite different hybrid rendering approach is presented in [Lin et al., 2007]. JPEG images are used to stream embedded 3D model information decoded as RGB information.

2.3 Existing Remote Visualization Tools

A variety of remote visualization applications exists in the IT universe. Many of them are well established and cover a wide range of different visualization capabilities or even have their focus on cluster applications.
In the following, a brief overview of existing remote visualization tools will be given. This list is not intended to be exhaustive.

Application	Image-based	Geometry-based	Hybrid	Collaboration
HP Remote Rendering Service [Hewlett-Packard, 2007]	+	-	-	+
SGI VizServer [SGI Inc., 1999]	+	-	-	+
Sun Shared Visualization [Sun Microsystems, Inc., 2008]	+	-	-	+
Chromium [Humphreys et al., 2002]	+	+	o	o
VR-Juggler [Bierbaum et al., 2001]	o	+	-	+
Scan View [Koller et al., 2004]	+	+	+	-

Table 1: Comparison of remote rendering tools,
(+) available, (-) not available, (o) unknown

2.4 Findings

Summarized, existing remote rendering techniques cover a wide range of problem statements including image-based remote renderings (suitable for fast network connections) or geometry-based remote renderings (e.g. cluster rendering with sufficient client's graphics hardware). Some of these approaches support collaborative remote rendering as well.
Hybrid rendering methods for their capabilities could be identified as adequate rendering method for low and medium level clients using networks with medium network speed.

3 System Demands

In this section the requirements of the involved underlying system architecture and its components as well as special user interests are discussed with respect to the chosen use case; a medium sized biology research institute.
Main issues are the individual clients' hardware configurations in the network, the data transfer capabilities and the state of scientific data in the investigated institution.

3.1 Clients Configuration

A medium-sized scientific biology research institute with approximately 300 potential clients was evaluated. These clients are completely heterogeneous, they cover a big inventory of out-of-date machines[2] as well as a few state-of-the-art hardware machines. The utilized systems are as diverse as the clients' hardware configurations. Whereas the processor speed is widely feasible, the graphics hardware is not. An nVidia GeForce2 MX is for example capable of rendering 20 million triangles per second (see `www.nvidia.com`). This means that at most $800,000$ triangles can be rendered per frame in real-time during interaction. One biological dataset consists of more than 80% more triangles than this hardware can manage. These preconditions have to be taken into account in the design of a remote visualization application for large biological models.

3.2 Server Configuration

Two different server machines could be used for testing the remote visualization performance.
The first one is a high performance computer with two Quad-Core Opteron 2.6 GHz, 16 GB RAM, a Quadro FX 5600 graphics card and furthermore a Tesla S870 processor with 500 GFlop and a shared 1 GBit connection (bottleneck) to the "Deutsches Forschungsnetz" (DFN) and to the biology research institute. An average server upload B_{up} of 2 MBit with average ping rates t_{avg} lesser than 23 ms could be identified ($\sigma_{B_{up}} \approx 0.2$ MBit, $\sigma_{t_{avg}}3$ ms).
The second server is a minimal graphics server configuration with a 2.8 GHz Pentium IV with 4 GB RAM and *WindowsXP* as operating system. The rendering job is done by one *Nvidia 7800 GTX* graphics card. This server is directly located at the biology research institute. The minimum network connection for this setup is bound to the local switches and achieves 100 MBit.

3.3 Data Transfer

The data transfer capabilities are one major aspect in the remote data visualization and can be formulated regarding to Equation 1.

$$B \geq \frac{(R_w \; R_h) \; C_{depth} \; f \; n}{c_{factor}} \tag{1}$$

Without any compression c_{factor} and for a screen resolution R_wxR_h (XGA), a color depth C_{depth} of 3 Bytes/pixel and a frame rate f of 26 frames/sec for only one client n, this would require at least a bandwidth B of:

$$B \geq \frac{(1024 * 768) \; pixel * 3 \; Bytes/pixel * 26 \; sec^{-1} * 1}{1}$$

$$B \geq 60 \; MByte/s \tag{2}$$

which is still high and only possible with at least 1 GBit/s ethernet[3] (see example calculation in Equation 2).
Using JPEG compression this can be reduced by a factor c_{factor} of $10-300$ (depending

[2] e.g. *Win98* Pentium III's with less or equal 500 MB RAM
[3] An $SXGA$ screen resolution requires respectively around 100 MByte/s.

on the resulting image quality) and will also work with slower connections. A compression factor of 50 will result in a data transfer rate of approximately 1200 kByte/s which might fit in a 10 MBit data connection, if the other users produce only very low amounts of traffic. This means a permanent data stream would load the network traffic and also prevent highly detailed images on the clients' side.
The local network speed at the biology research institute depends on the used switches and includes 100 MBit as well as 1 GBit connections. This fact prevents a standard remote rendering strategy due to the reason that high-quality renderings would not fit into a 100 MBit connection if a couple of other users share the same server connection (big n).

3.4 Data Resources

The biological data pool consists of different data domains from biological material. The internal knowledge of the data's specific characteristics can be advantageous to find solutions for a well suited remote visualization approach.
For the visual analysis, specifically microtome serial section data of different stages of barley seed development (*days after flowering* or DAF) are used, similar to the method described in [Gubatz et al., 2007].
Each dataset consists of approximately $2,000$ slice images of barley caryopsis, which were obtained from a microtome at 3 μm slice thickness. Digitized with a color CCD camera, images originally measured 1600×1200 pixels at a spatial resolution of 1.83×1.83 μm per pixel. Since color-space analysis revealed an almost linear correlation and a limited range, it was possible to reduce the optical resolution of 12 bits for each RGB channel to 8 bit grayscale images without significant losses [Schoor et al., 2008a], thus reducing the data volume for each dataset to approximately 5 GB. Depending on the developmental stage of the specimen, sections can be too large to be captured in a single frame at the given spatial resolution.
Available data include gene expression assays utilizing mRNA *in-situ hybridization* (ISH) probing of histological cross-sections. Whereas *in-situ* data allows visualization of spatial gene expression patterns, staining with gene-specific samples requires complex chemical protocols, for that ISH image data origins from different individual caryopsis.
Magnetic resonance spectroscopy measurements (^{1}H-NMR) of caryopses at different points in time were sampled in a specific device. While being nondestructive, the detected proton distribution has a lower spatial resolution of approximately $10\mu m$ (isotropic) per voxel, and does not resolve histology or structural features.
Other potential datasets, such as macroarray gene expression data or metabolomic assays by laser microdissection (LMD), could be integrated into the database to enhance synergy and facilitate the inference of analysis results. Figure 2 presents the database with different data domains and an example image, and Figure 3 a pie chart breaking down data distribution per model. The organization is derived from the *entity-attribute-value* (EAV) design as in [Anhøj, 2003]. This approach has the advantage of high flexibility when extending the database with other data domains.

3.5 Visualization Aspects

Another very important precondition is the range of functions of the remote rendering application to develop. At the moment the bio researchers test a software application

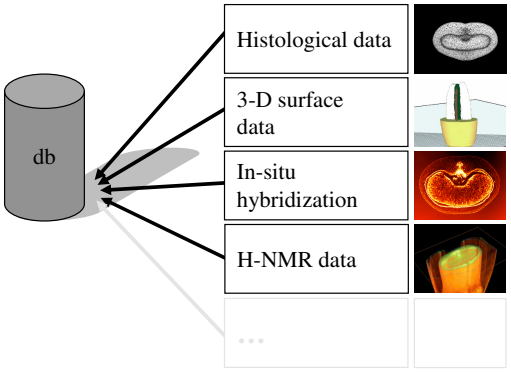


Figure 2: Schematic view: Integration of different data domains by using an *EAV* model for the visualization database.

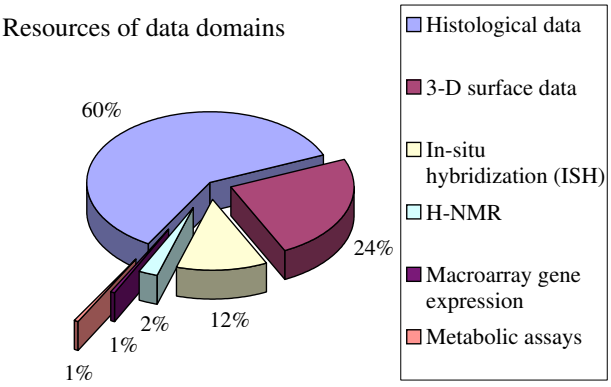


Figure 3: Data distribution for one model, the absolute data totaling around 9 GB

which is especially tailored to them, namely the *BioVizIt* application. This software is initially a single-user stand alone application, which is not network compatible. Among others, the scope of this solution covers the following features and actions:

- scene graph,
- transparency and color settings to objects,
- hide, copy, delete, create, rename, group and ungroup objects,
- affine transformations, model deformations and model measurements,
- variable and textured clipping planes per object,
- image segmentation techniques,

- annotations

Another point the application must meet is the production of high-quality renderings on demand. All these features and actions have to be considered as part of the resulting remote rendering application to best possibly support the biologist's work.

3.6 Findings

As conclusion, there are different possibilities to perform remote visualization tasks as proposed in Section 2. As possible candidates only those solutions are appropriate which meet the requirements of the Section 3.1-Section 3.5 .
The clients' hardware denotes that the datasets to be worked with are too large and too complex to be viewed with local resources. The standard approach for remote visualization (pure image transfer)is not sufficient for this application due to the fact that rendered images with complex color patterns and differing inter-frame correlations typically lack in being processed and transmitted with interactive frame rates.
As solely remote rendering approach which fits the system requirements a hybrid rendering approach is applicable.
Due to the specific visualization requirements depicted in this section a conception with an explicit control to data exchange mechanisms and adaption possibilities of the visualization is necessary. A solution to achieve this is to extend the existing *BioVizIt* application with specific hybrid remote rendering capabilities.

4 The Concept of a Hybrid Remote Rendering Approach

In the following, the remote visualization concept is described, where a hybrid rendering approach is the basis. Furthermore, the architecture's adaptions of the BioVizIt platform, referring to network compatibility and approaches addressing limited bandwidth, are described.

4.1 Hybrid Rendering

Figure 4 illustrates the idea of the hybrid approach. In the beginning of the rendering process (initial rendering request) a request with datasets' names and initial settings is sent to the server. As a result, the server application is loading the requested file from the database, and in addition a low resolution 3D geometry model (*level of detail* also *LOD*) is transferred to the client. Moreover, the graphics server submits the first remote-rendered image to the client.
If in the following, user navigation is performed (consecutive rendering requests), the local 3D model is rendered by the client itself. Fast movements can be achieved without having performance problems due to network limitations. The local 3D model can be reduced (detail loss) up to one per mill (approximately 4-5 MB) of the original data amount of the 3D model. This is equivalent to the data amount necessary performing real-time server-sided image rendering in XGA mode for two frames. Even if the bandwidth is less than 10 MBit an initial loading delay for an unoptimized data transfer of 4 seconds for the complete low resolution model would be acceptable to

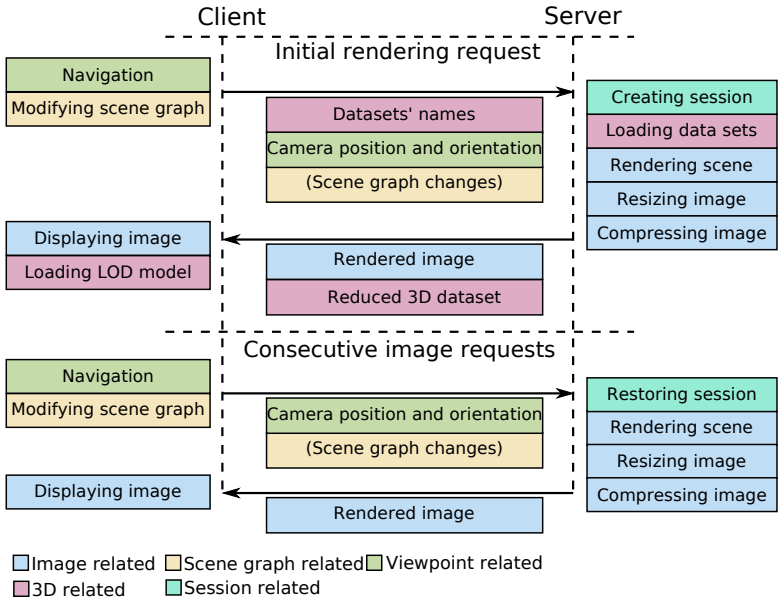


Figure 4: Sequential diagram of the proposed hybrid rendering approach

the user, if afterwards a stable real-time interaction can be performed.
In case of stopping the navigation or changing the scene graph for example, a command is immediately sent to the server (containing at least the new camera position and orientation) to request a new rendered image. This approach ensures real-time model interactions with a high-quality rendering result during the model exploration.
A comparable approach is used by a variety of programs which display only a subset of the whole dataset or even a faster rendering mode (point rendering) to ensure real-time model interaction, for example *Polyworks*[4].

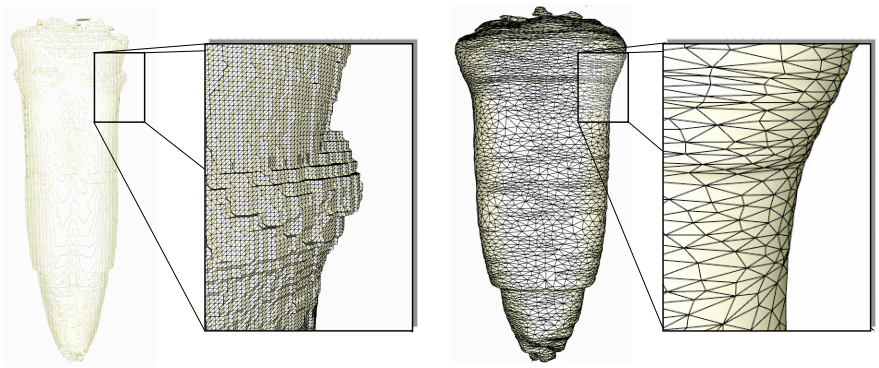
Geometry Submission

The server stores the information of the original 3D models and also in coarser resolutions (*LOD*). This storage overhead is uncritical because data storage is not the limiting size for this application. Datasets are generated according to the proposed technique in [Schoor et al., 2008b] (see Figure 5). If a dataset is requested, the original dataset is loaded into the server application. At the same time a low resolution model is transferred from the database to the client.

Image Submission

The image submission by demand is a concept to overcome the server network load. Additionally, image compression algorithms [Taubman & Marcellin, 2001] and residual images (see [Yoon & Neumann, 2000]) are used to reduce the network traffic.

[4] `http://www.innovmetric.com`



a) b)

Figure 5: Surface models of different grains' outer hull at different resolutions a) grain with a mesh size of 560,000 triangles and detail with highlighted faces b) reduced mesh with approximately 10,000 highlighted faces and detail with highlighted faces

4.2 Network Interconnect

Another important aspect besides the underlying data transfer per se is how the data will be transported. To achieve high resolution rendering results on the client, only a reliable data transfer method is an option. This excludes for example *UDP* or *UTP* [Thibault et al., 2002] data transfer, because in such data transfer mode the correct rendering result is not warranted due to unordered or lost packages. The network protocol layer *TCP* is sufficient for the required purposes, albeit *UDP* is faster, for example.
By contrast the TCP connection ensures the correct receiving of either geometry models or rendered images.

4.3 Architecture Adaption of the BioVizIt Platform

For an extension of the existing solution as server variant the range of functions must be appended to network and session routines. Multiple network connections through clients must be promptly accepted and processed, using multi-threading techniques. User events are registered in a message queue. This queue is held in the operating system and members of the queue are passed to the application (for example panning the window). Received messages by the application are then handled in a procedure of the primary thread.
To avoid blockages in the processing of this thread, a new thread – the "server thread" – is generated during the program start to wait for new client requests via Transmission Control Protocol (TCP).
If a client connects to the server over a predefined port, a new "client thread" is generated immediately for this client. This thread is responsible for further communication with the client.
The client thread reads the request from the server's socket thread and interprets it

through the HyperText Transfer Protocol (HTTP). By accessing a virtual document, additionally the document area of the HTTP request is used, which includes the remote rendering control commands of the client for the server. These commands are processed sequentially.
At the end of this processing the client is enqueued to a list, which contains clients awaiting an image synthesis step.

5 Implementation and Results

In this section, the essential aspects of the implementation are presented. The identified key issues for remote visualization are examined regarding the presented approach.
Network latency The latency at the beginning of a session depends on how the geometry transmission is handled. The coarse 3D model can first be accessed after full model upload. This can take a few seconds for the geometry transmission ($3 - 4$ s for server case 2). This is an initial step which is only performed once a model is loaded. After that, the interaction is performed with latencies < 30 ms due to local rendering.
The rendering result in high resolution takes less than 1 second after the mouse button is released (indicating the end of user navigation). The time needed to display the result on the client is the sum of the steps which are shown in Figure 6.

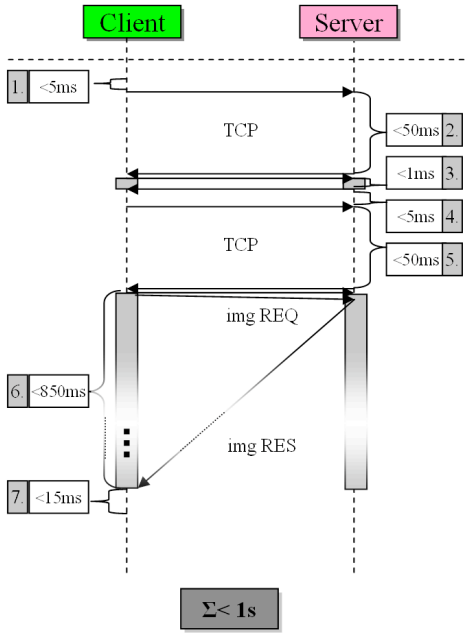


Figure 6: Sequential diagram of server and client transmissions with approximate time spans for one remote rendering step for server case 2

The sequences in Figure 6 are:

1. the control signal from interaction device (mouse) to the client

2. establishing TCP connection (3-way handshake)
3. sending the rendering information via html request (position, orientation, fov, etc.)
4. doing the rendering job on the server
5. establishing TCP connection again
6. sending an image request to the server and getting the respective data
7. using these data to display the scene in high resolution

Graphics performance The graphics performance of the client rendering stage depends on the clients hardware and on the level of detail in which the 3D model was submitted. For the tested application even on a Pentium III client a coarse 3D model with $20,000$ points could be interactively visualized.
During the observation of the model (no model interaction) no rendering step is required. That means that a real-time application can be guaranteed for the model interaction.
Network throughput The network throughput per high resolution image request is equivalent to the physical size of this image and corresponds to approximately 300 kB. With an initial geometry transfer of 4-5 MB for the low resolution 3D model and an average image request of 3-5 s, a data volume per client of 10 MB is transmitted within the first minute.
Scalability The approach provided here has a good scalability. The rendering of scenarios, such as described in Section 3, can be accomplished on graphics servers without problems with interactive frame rates (Case 2: minimum graphics server solution). That means, in the worst case[5] the remote rendering image of the last handled client will be displayed with an additional delay of approximately 1 s if the data transfer is ignored. This is the theoretical maximum number of users which can be handled by the server under the assumption that a delay of more than 2 seconds is not acceptable to users.
Taking the 100 MBit connection as lower bound for the data channel (see Section 3), more than 40 clients can request a full resolution image at the same time without causing additional delay at a client station due to the data transfer. Assumed that a 2 MBit connection is the average connection configuration for the server case 1, and a full image resolution is requested, only one client request per second is possible without causing an additional delay on the client.
Error recovery The error recovery is automatically performed by the network protocol layer (TCP) which provides explicit error corrections. Furthermore, a transmission control and data ordering is also part of this protocol. This is paid with a higher latency of the remote rendering.
Network robustness In general, the failure of a network connection in the beginning of a work session leads to an insufficient remote visualization. During the session a temporary failure of the network connection in the time scale of seconds can be bridged without problems due to the local downsampled version of the 3D model. A loss of connectivity within a larger time range disallows a practical use of the here provided technique due to the missing detail information the remote-rendered image provides.

[5] e.g. if the server gets a synchronous request of 25 clients

Security issues The data pool of plant biological data consists of confidential information. Even an internal use must contain at least a simple protection mechanism due to project internal information. Therefore, a simple login query was integrated to authenticate the users with respective project rights. Furthermore, each session on the remote visualization server is initially hidden to other users.
This approach does not cover any wiretap operation or similar attacks via network. The image data as well as the 3D low resolution models are submitted plain without protection mechanism up to now. Possible are techniques like in [Koller et al., 2004] which provides small shifts in the submitted image data (jitter) to prevent a reconstruction of the high resolution 3D model data or use additionally a secure transfer protocol (e.g. HTTPS).
Balance between local/remote resources The architecture of the hardware at the scientific institution is naturally very heterogeneous. A uniform strategy to extensively use clients' hardware is therefore not applicable. As the least common denominator, a low resolution 3D model, which is tailored to the actual hardware configuration, is used at the clients' side to achieve real-time interaction.
Hardware/software incompatibility The visualization software is written in *C++*, uses only libraries like *wxWidgets* and *OpenGL*, and is therefore platform-independent for supported systems. The visualization service tests for hardware premises, if specific features are not supported by actual server hardware, alternatively slower CPU-based implementations exist (e.g. deformation model).
The client's visualization front-end uses a platform-independent web interface, based on $Java^{®}\ Version\ 1.5$, $Java\ 3D^{™}$. This front-end was tested with *Firefox 3.0*, *Internet Explorer 6.0 / 7.0*, *Opera*, and *Chrome Beta*. The application even runs on a mobile device with *Windows CE*. Incompatibilities are not known so far. The tests covered only the general operating.

Figure 7 shows a screenshot of the provided web interface and the visualization options.

6 Conclusion

In this paper a remote rendering approach was presented, which is applicable to medium-sized enterprises who have to deal with a large amount of scientific data. With a minimum of effort and the use of existing hardware this approach allows to visualize, explore and modify data. It could be shown that a simultaneous use of 5 high-level sessions could be managed with no recognizable loss of performance even with the server case 1.
The clients hardware constitution must not be state-of-the-art which makes this approach of visualization furthermore attractive in a financial manner. Instead of the high end graphics server (case 1), the local visualization server with the minimal graphics configuration (case 2) is to prefer. The network bandwidth could be identified as limiting constraint respective to the number of clients contemporaneously used.

7 Future Work

If the remote rendering application will be used with many users, the aid of more than one rendering machine can be usefull. In the next term a larger focus will be

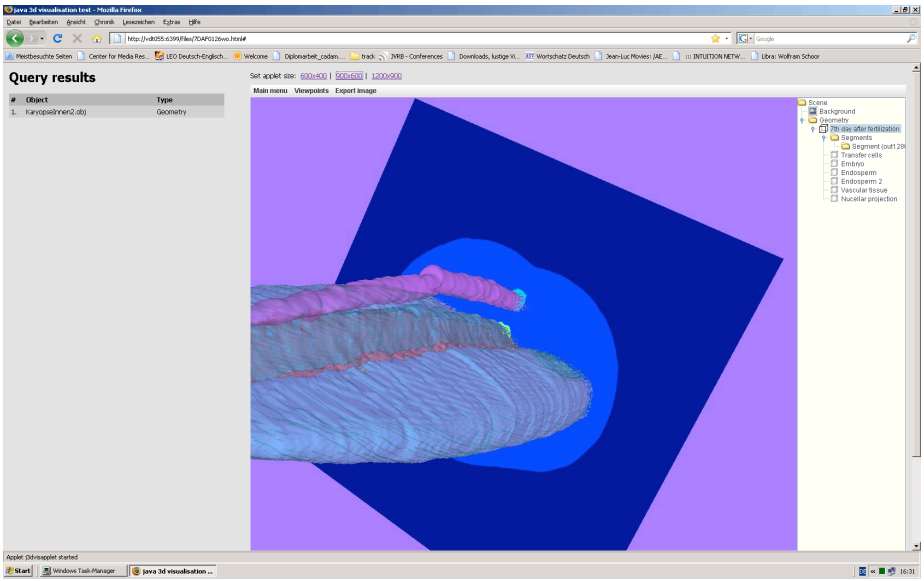


Figure 7: Screenshot of the clients web interface showing a reconstructed model of a caryopsis with clipping plane

placed on data modification possibilities and supporting full functionality to biologists in comparison to the stand-alone BioVizIt application.
A linking of the VISH platform [Benger et al., 2007] with the provided Java web-interface is planned.
Applying non photorealistic rendering (NPR) aspects like the line drawings based on multi-resolution meshes [Ni et al., 2006] or feature-line-based city visualizations [Quillet et al., 2006] to remote rendering applications, this can improve the performance if using a purely image-based remote rendering approach.
It is planned to expand the intranet-based solution to the internet for selected public datasets so that interested users can interactively explore information within the provided 3D models from the institutes website.

8 Acknowledgments

This research work is being supported by the BMBF grants 0313821B and 0313821A. The authors would also thank the DFG for supporting the cooperation preparative between the Fraunhofer Institute and the Louisiana State University by the grant SCHO 1346/1-1.

References

[Anhøj, 2003] Anhøj, J. (2003). Generic design of web-based clinical databases. *Journal of Medical Internet Research*, 5(4), e27. URL: `http://www.jmir.org/2003/4/e27/`.

[Benger et al., 2007] Benger, W., Ritter, G., & Heinzl, R. (2007). The Concepts of VISH. In *Proceedings of the 4th High-End Visualization Workshop: Open issues in visualization with special concentration on applications in astrophysics, numerical relativity, computational fluid dynamics and high-performance computing*.

[Bierbaum et al., 2001] Bierbaum, A., Just, C., Hartling, P., Meinert, K., Baker, A., & Cruz-Neira, C. (2001). Vr juggler: A virtual platform for virtual reality application development. In *VR '01: Proceedings of the Virtual Reality 2001 Conference (VR'01)* (pp.8̃9). Washington, DC, USA: IEEE Computer Society.

[Dunwoody, 1996] Dunwoody, C. (1996). *The OpenGL stream codec : A specification.* Technical report, Silicon Graphics, Inc.

[Gubatz et al., 2007] Gubatz, S., Dercksen, V. J., Brüß, C., Weschke, W., & Wobus, U. (2007). Analysis of barley (Hordeum vulgare) grain development using three-dimensional digital models. *The Plant Journal*, 52(4), 779–790.

[Gueziec et al., 1999] Gueziec, A., Taubin, G., Horn, B., & Lazarus, F. (1999). A Framework for Streaming Geometry in VRML. *IEEE Computer Graphics and Applications*, 19(2), 68–78.

[Hewlett-Packard, 2007] Hewlett-Packard (2007). Advantages and Implementation of HP Remote Graphics Software HP Remote Graphics Software enables 2D and 3D real-time interactive graphics and collaboration from a distance. White Paper.

[Humphreys et al., 2002] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., & Klosowski, J. T. (2002). Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3), 693–702.

[Koller et al., 2004] Koller, D., Turitzin, M., Levoy, M., Tarini, M., Croccia, G., Cignoni, P., & Scopigno, R. (2004). Protected interactive 3d graphics via remote rendering. *ACM Trans. Graph.*, 23(3), 695–703.

[Lin et al., 2007] Lin, N.-S., Huang, T.-H., & Chen, B.-Y. (2007). 3D Model Streaming based on JPEG 2000. *IEEE Transactions on Consumer Electronics*, 53(1), 182–190.

[Moore, 1998] Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1), 82–85. URL: `http://dx.doi.org/10.1109/JPROC.1998.658762`.

[Ni et al., 2006] Ni, A., Jeong, K., Lee, S., & Markosian, L. (2006). Multi-scale line drawings from 3d meshes. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (pp. 133–137). New York, NY, USA: ACM.

[Quillet et al., 2006] Quillet, J. C., Thomas, G., Granier, X., Guitton, P., & Marvie, J. E. (2006). Using expressive rendering for remote visualization of large city models. In *Web3D '06: Proceedings of the eleventh international conference on 3D web technology* (pp. 27–35). New York, NY, USA: ACM.

[Scheifler & Gettys, 1986] Scheifler, R. W. & Gettys, J. (1986). The X window system. *ACM Trans. Graph.*, 5(2), 79–109.

[Schmalstieg & Gervautz, 1996] Schmalstieg, D. & Gervautz, M. (1996). Demand-driven geometry transmission for distributed virtual environments. *Computer Graphics Forum*, 15(3), 421–431. URL: `citeseer.ist.psu.edu/schmalstieg96demanddriven.html`.

[Schoor et al., 2008a] Schoor, W., Bollenbeck, F., Hofmann, M., Mecke, R., Seiffert, U., & Preim, B. (2008a). Automatic Zoom and Pseudo Haptics to Support Semiautomatic Segmentation Tasks. In V. Skala (Ed.), *16th WSCG 2008*, WSCG'2008 Full Papers Proceedings (pp. 81–88).: WSCG University of West Bohemia. Full Paper.

[Schoor et al., 2008b] Schoor, W., Bollenbeck, F., Weier, D., Weschke, W., Preim, B., Seiffert, U., & Mecke, R. (2008b). VR-Based Visualization and Exploration of Plant Biological Data. *Journal of Virtual Reality and Broadcasting*. submitted.

[SGI Inc., 1999] SGI Inc. (1999). SGI® OpenGL Vizserver™ 3.5 Visualization and Collaboration (Application-Transparent, Remote, Interactive). White Paper.

[Shreiner et al., 2005] Shreiner, D., Woo, M., Neider, J., & Davis, T. (2005). *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2 (5th Edition) (OpenGL)*. Addison-Wesley Professional.

[Sun Microsystems, Inc., 2008] Sun Microsystems, Inc. (2008). Seeing the Future with the Sun™ visualization system: Scaling, sharing, and scheduling visualization resources. White Paper.

[Taubman & Marcellin, 2001] Taubman, D. S. & Marcellin, M. W. (2001). *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Norwell, MA, USA: Kluwer Academic Publishers.

[Thibault et al., 2002] Thibault, S., Cavin, X., Festor, O., & Fleury, E. (2002). Unreliable transport protocol for commodity-based opengl distributed visualization. In *Workshop on Commodity-Based Visualization Clusters, Boston, MA*.

[Web3D Consortium, Inc., 1998] Web3D Consortium, Inc. (1998). *ISO/IEC 14772-1:1998: Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding*. International Organization for Standardization.

[Web3D Consortium, Inc., 2004] Web3D Consortium, Inc. (2004). *ISO/IEC 19775-1:2004 Information technology — Computer graphics and image processing: Extensible 3D (X3D) — Part 1: Architecture and base components*. International Organization for Standardization. URL: `www.web3d.org/x3d/specifications`.

[Womack & Leech, 1998] Womack, P. & Leech, J. (1998). *OpenGL graphics with the X Window System, version 1.3*. Technical report, Silicon Graphics, Inc.

[Yoon & Neumann, 2000] Yoon, I. & Neumann, U. (2000). Ibrac: Image-based rendering acceleration and compression. *Eurographics 2000, Vol*, 19, 321–330.

Strategies for Efficient Transparency Determination Based on Depth Peeling

Lars Uebernickel[1] Wolfram Schoor[1,2]
Bernhard Preim[1]
[1]Otto von Guericke University, Magdeburg
[2]Fraunhofer Institute for Factory Operation and Automation, Magdeburg
uebernic@cs.uni-magdeburg.de
wolfram.schoor@iff.fraunhofer.de
preim@isg.cs.uni-magdeburg.de

Abstract

The rendering of semi-transparent polygonal surfaces is a common task in computer graphics and very important to present inner structures of 3D objects by preserving the mean information of the object's 3D shape. A variety of approaches exists for transparency determination with different features and drawbacks. Rendering large and complex scenes in a correct manner is hard to accomplish in real-time for order-dependent transparency determination. In this paper, different methods are introduced, which support application-specific transparency and are based on the depth peeling principle: i) *Reverse Depth Peeling* for preserving Video Random Access Memory (VRAM), ii) *Adaptive Depth Peeling* reducing the render passes by composing backmost depth layers into quadripartite division layers, and iii) *Stop criteria* to achieve real-time interaction at the expense of rendering accuracy.
A practical evaluation for selected example models is given.

1 Introduction

On the majority of modern consumer-level graphics hardware, visible surface determination is implemented using a depth buffer. This is very efficient, as it fits well into the pipeline-based design of this hardware. However, it does not produce correct results for non-refractive transparent surfaces, as individual fragments are not drawn in the correct back-to-front order or might not be rendered at all (due to early depth rejection). To circumvent this limitation, transparent surfaces can be depth-sorted and rendered back to front. Polygons which intersect themselves (in view space) have to be split and rendered separately. This method is computationally expensive for the CPU, as the sorting and splitting has to be recalculated for each frame and cannot be

done on a graphics chip. A different approach to this problem is *Depth Peeling* [Everitt, 2001]. This technique separates each unique depth *layer* in the scene into a separate layer and composes them back-to-front into the framebuffer. However, this approach is very resource intensive, as the scene has to be drawn multiple times; once for each peeled depth layer. This work proposes various approaches to modify the classic depth peeling algorithm to adapt to different application needs. Especially in technical visualizations [Diepstraten et al., 2002] or [Elmqvist et al., 2007] transparency is a valuable feature. Trade-offs for rendering accuracy, speed, and memory consumption are presented. The presented implementations are compatible to graphics cards supporting OpenGL 2.1 [Rost, 2006]. First, two modifications to the classical depth peeling algorithm are presented in Section 3 and Section 4. Section 5 introduces various *Stop criteria* along with their properties.

2 Related Work

In this section, different techniques for transparency determination are presented. Starting with a brief review of existing order-dependent transparency determination algorithms, the focus is subsequently set to image-based depth peeling algorithms. Many algorithms were introduced in the past to solve the problem of correct transparency in virtual 3D scenes. These algorithms worked in general in the object space and used a depth sorting step [Catmull, 1974]. The drawback of algorithms working in object space is that the performance of the application directly correlates with the triangle count. These algorithms have therefore an insufficient scalability. Furthermore, the CPU processing resources are loaded. Due to this, order-dependent transparency approaches are out of the scope of this paper. For the correct handling of all OpenGL blending modes [Shreiner, 2004], the visible transparent surfaces have to be blended in a back-to-front order. The depth buffer algorithm cannot handle transparent surfaces correctly [Catmull, 1974]. This can be solved by utilizing the A-buffer algorithm which stores a linked list of fragments for each pixel [Carpenter, 1984]. The final color of a pixel is computed by depth-sorting the fragments ascending and then recursively blending the fragments from back to front by accumulating them. In the following, different image-space-based strategies for transparency determination are presented.

Classic Depth Peeling The concept of classic depth peeling by [Everitt, 2001] assumes that after rendering the scene with a depth test, the depth buffer of the nearest fragment to the observer is bound to a fragment shader (cf. also [Mammen, 1989]). The source and destination depth buffers are permanently swapped every pass to avoid problems by modifying the buffer (double buffering). If the present fragment depth is less or equal to the regarding depth from the previous pass, then this fragment will be discarded and the next layer underneath is returned by the depth test. The major drawback of this algorithm is that one geometry pass is needed per peeled layer. Thus, for render-

ing the scene of depth complexity N, classic depth peeling requires N geometry passes, which can be extremely computational expensive for large N. Two depth buffers are used concurrently for determining the farthest unprocessed transparent surface for each pixel. The process is repeated until all transparent surfaces have been blended. Order-independent transparency is computed correctly since the per-pixel sorting is performed implicitly using selection sort. The peeling order is shown in Figure 1.

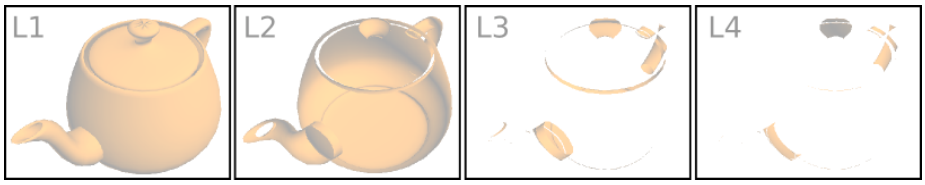


Figure 1: Classic Depth Peeling according to [Everitt, 2001]. The object is peeled from the front to the back, showing the results of each depth layer comparable to clipping planes.

A Hybrid Rendering Approach for Separate Opacity and Transparency Handling In [Wexler et al., 2005] a hybrid approach was presented for rendering complex scenes with high quality. In a first step only opaque objects are rendered. For transparency determination a depth peeling approach was implemented as second render step. Opacity thresholding and a new method called z-batches speed-up the transparency determination. At high spatial sampling rates, the implementation is considerably faster than CPU-based rendering approaches for typical scenes.

Multi-Layer Depth Peeling via Fragment Sort [Liu et al., 2006] describe an accelerated depth peeling algorithm, which in contrast to other depth peeling algorithms peels multiple layers simultaneously per rendering pass. This multi rendering approach is achieved by a fragment sort program, which sorts and writes multiple fragment colors and depths via Multiple Render Targets (MRT). Liu et al. report a speed-up up to eight times of classic depth peeling [Everitt, 2001]. The algorithm is specified as robust against unreliable parallel read-after-write behavior in current graphics hardware.

Multi-Fragment Processing using the K-Buffer A technique called *k-buffer* as generalization for the depth buffer was described in [Bavoil et al., 2007], allowing multiple fragments per pixel with read-modify-write capabilities applicable for a variety of rendering algorithms as well as for fast transparency determination. First introduced in the field of direct volume rendering [Callahan, 2005] and [Callahan et al., 2005], with the *k-buffer* it is possible to determine transparency with only one single geometry pass. This requires only a small fixed additional amount of memory.

Depth Peeling using Delay Streams [Aila et al., 2003] presented a modification of the pipeline of modern graphics hardware architecture. They added a *Delay Stream* between the vertex and pixel processing units. This allows to determine occlusion information of subsequent receiving triangles, while the present triangle stays in the delay stream. Furthermore, the memory and bandwidth requirements of order-independent transparency can be drastically reduced in most cases. In addition to the *R-Buffer* technique (see [Wittenbrink, 2001]), not only the *ARGBZ*, blend mode, and coordinates are stored, but all render state changes as well as compressed vertices and coordinates of unprocessed pixels are stored. Attributes like color, depth and others are computed from the vertices and render states during peeling by using the existing hardware units. Few hidden primitives are stored due to the efficient occlusion culling. Depth peeling is performed using only one frame buffer and one depth buffer. By using multiple depth buffers simultaneously [Wittenbrink, 2001], the number of passes over the order-independent transparency stream can be further reduced.

Order-independent Alpha Blending [Meshkin, 2007] presented an approach for order-independent alpha blending by a simplification of the formula for calculation of multiple translucent layers. It could be shown that this simplification achieves better results than the full term (discarding the order-dependent term), respective to the error occurring by blending the layers. For small alpha values as well as for low intensity values, results with small artifacts could be achieved. Higher alpha values can lead to larger artifacts up to complete inaccuracies. Due to the simple formula only two render passes and a single render target are required. Blending only one translucent layer, correct transparent rendering results can be guaranteed. Summed up, this is a comparably fast approach providing suitable results for low alpha values. MRTs can be used to speed up the computational time by simultaneously solving the blending equation in parts.

Weighted Average Based on the idea of [Meshkin, 2007], [Bavoil & Myers, 2008] introduced a real weighted average technique which does not ignore any term. It builds on the following observation. Under the assumption that all RGBA colors were identical for a given pixel, the result would be independent on the order in which the fragments are alpha blended. In the usual case, where the RGBA colors are not uniform, the multiple colors per pixel are replaced by a uniform color per pixel. Non-uniform alphas can be handled applying an alpha-weighted average of the colors for every pixel. The average RGBA color for each pixel is generated by rendering the transparent geometry into an accumulation buffer implemented as a 16-bit floating-point texture. The result is an accumulated RGBA color and the number of fragments per pixel equates the depth complexity. After this geometry pass, the accumulation buffer is passed to a post-processing full screen pass, which performs a weighted average of RGB by alpha, yielding a weighted average color C. This

is a fast approach for determining the transparency of a scene, because only one rendering pass is needed.

Dual Depth Peeling Dual depth peeling [Bavoil & Myers, 2008] quite bisects the number of geometry passes of the classic depth peeling from N to $N/2 + 1$ by applying depth peeling from front and from back simultaneously. The hardware depth buffer must be explicitly switched off to perform a custom depth test for a blending without read-modify-write hazards. This approach only works on new graphics hardware (GeForce 8 Series and higher).

3 Reverse Depth Peeling

The classical depth peeling algorithm stores a framebuffer-sized image for each depth layer and composes those layers back-to-front in its final step. Hence, the amount of graphics memory consumed by this technique grows linearly with the number of peeled layers. This might be undesirable for some applications, as the graphics memory might be needed for different data, e.g. high quality textures.

The *reverse depth peeling* approach alleviates this problem by peeling the layers from the back, starting with the furthest one in the scene (see Figure 2). Since the layers are peeled in the same order as they are composed into the frame buffer, this can happen incrementally after each peeling step. Only two frame buffers are needed (in addition to the usual two depth buffers), one for the current layer, and one for the accumulated final image. Hence, the memory usage of this approach is fixed and does not depend on the depth complexity of the rendered model.

However, this technique has a significant drawback: Since the nearest peeling layer is handled last by the algorithm, all layers of a scene must be peeled in each frame. This should not be a problem for scenes for which the depth complexity is fairly small and constant across all viewing angles, as it is the case for some visualization tasks.

For scenes in which the absolute depths are fairly well known (e.g. the scene is always viewed from a specific distance or angle), this problem can somehow be alleviated by clearing the initial depth buffer with a value smaller than 1.0. This will discard depths greater than this value, possibly resulting in fewer overall layers.

4 Adaptive Depth Peeling

A huge drawback of the classic depth peeling algorithm is its extensive need for video memory. A full color, framebuffer-sized texture must be stored for each layer, in addition to two depth buffers. The *Reverse Depth Peeling* approach (cf. Section 3) tries to circumvent this by peeling the layers from the back, but it is not as adaptable in its need for GPU usage (cf. Section 5).

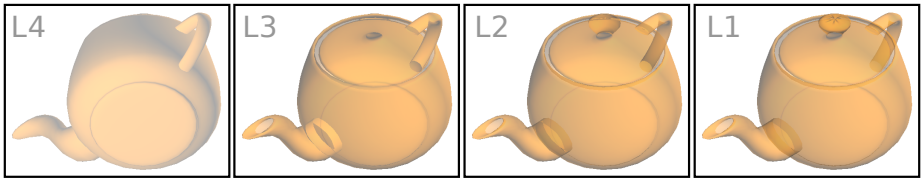


Figure 2: Scheme of Reverse Depth Peeling. The object is peeled from the back (depth layer L4) to the front (depth layer L1). The result is immediately blended together in one buffer. Adding the information of the frontmost depth layer in a last step yields to the final rendered image.

The *Adaptive Depth Peeling* algorithm takes advantage of the fact that the color contribution of a layer to the final scene decreases with each additional layer. It does so by lowering the resolution of far-away layers. This degrades the quality of the final image, being especially visible at crossings between objects due to slightly jagged object boundaries. Nevertheless, it decreases the amount of memory needed and allows for faster rendering times than the classical depth peeling approach, as not as many samples need to be drawn for each layer.

The specific amount of memory saved compared to classical depth peeling depends on the intensity of the resolution reduction and the amount of layers rendered in each resolution. For example, when halving the resolution in each dimension for all remaining layers q after the first full quality layers f of n total layers, the following memory and fill-rate savings S yield:

$$S = \quad n - (f + \lceil \tfrac{1}{4}\ q \rceil) \tag{1}$$

$$= \quad n - (f + \lceil \tfrac{(n-f)}{4} \rceil) \tag{2}$$

In contrast to the *Reverse Depth Peeling*, this algorithm is more adaptable to specific performance and quality trade-offs: First, it can be used with the same *Stop criteria* as the classic depth peeling (cf. Section 5). Furthermore, it can be specified how much the resolution should be decreased and after which layers.

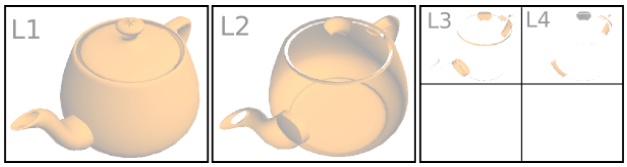


Figure 3: Principle of the Adaptive Depth Peeling. A distinct number of backmost depth layers with reduced resolution is arranged into one buffer.

5 Stop Criteria

Since depth peeling is an iterative algorithm, it can be stopped before it completes in order to save computation time and / or memory. This results in a loss of image quality, which might be acceptable when trying to save computational resources.

This is especially desirable when taking the inherent properties of the peeled layers into account: Firstly, the amount of samples drawn steadily decreases with each layer. Hence, cutting off some of the last layers will never influence the image quality as much as rendering the model without any depth cues.

Additionally, the amount of samples drops heavily after a first couple of layers (usually two) for many real-world models (see Figure 4), resulting in many layers in the "back" of the scene with only relatively few samples in each case. Dedicating as much rendering time to those layers as to the first ones seems wasteful.

Furthermore, the highest depth complexity often occurs on parts of the model with high curvature but little screen coverage. Since it is almost impossible to properly perceive the depth appearance of these parts of a model when they are rendered transparently, sacrificing image quality at these points provides better results than doing so on parts of the model with greater screen coverage and less depth complexity.

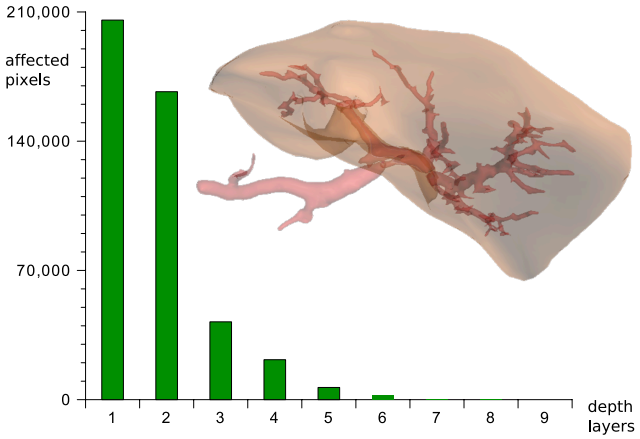


Figure 4: Influence of a depth layer against the amount of affected pixels shown on the Liver model.

This section presents multiple *Stop criteria* which limit the amount of layers produced in each frame, taking into account the previously discussed properties of the peeled layers. All of them can be applied to the traditional depth peeling algorithm as well as *Adaptive Depth Peeling*. Due to peeling the layers from the back of the scene, *Reverse Depth Peeling* is not suited for any of them.

No Samples Left Generate all layers until nothing is drawn into the frame buffer anymore. This stop criterion ensures the highest quality for depth complexities, as it generates all existing layers. It can be implemented by counting the samples passing the depth test for each layer (i.e. an occlusion query), stopping when this count reaches zero. This results in always rendering one layer more than is necessary. Even though all samples fail the depth test in the last layer, generating it might not be negligible for large models. Another disadvantage of this criterion is that it might have substantially different running times for different views on the same scene.

Few Samples Left This slight variation of the above criterion stops the generation of layers if the number of samples drawn for the previous layer is below a user-specified threshold. Since the amount of samples steadily decreases with each layer, this criterion will always cut off layers which contribute less samples to the final image than their predecessors. Furthermore, it conveniently omits those parts of the model which have a high depth complexity but low screen coverage and are therefore hard to perceive anyway.

For each view on a scene, a threshold exists for this criterion which executes the depth peeling algorithm in its entirety without rendering any unnecessary layers (cf. the empty layer in *No Samples Left* or superfluous layers in *Fixed Amount of Layers*).

Fixed Amount of Layers This stop criterion imposes an upper limit on the amount of layers generated for the final image. Therefore, the upper limit of the rendering time can be controlled more precisely, as it does not depend solely on the scene's depth complexity. Furthermore, an implementation can allocate the fixed amount of depth buffers beforehand, saving the costs to allocate additional buffers in each frame and helping to keep the video memory defragmented. Finally, the occlusion query test that is necessary for each rendered layer for the *No Samples Left* and *Few Samples Left Stop criteria* can be omitted here. This might result in performance gains, because for each occlusion query, the graphics pipeline has to be flushed (resulting in pipeline bubbles) in order to get the amount of samples rendered.

However, this approach does more work than necessary in regions of the scene with low depth complexity, and produces artifacts in regions with high depth complexity, as it cuts off the last layers.

Contribution to the Final Image Stop generating layers as soon as the maximum contribution any sample could make to the final value of the corresponding pixel falls below a certain threshold. The contribution C of a sample to the final image can be calculated by

$$C = \prod_{i=0}^{n} (1 - \alpha_i) \tag{3}$$

where n is the amount of layers which are in front of the current layer, and α_i are their respective opacity values. This value has to be calculated once for each sample and can be accumulated in an OpenGL accumulation buffer or a seperate render target.

Fixed Amount of Time per Frame Stop as soon as a fixed amount of time has passed. This is useful in highly interactive applications which need to maintain a specific frame rate at the cost of rendering accuracy. A problem which might occur with this is that no explicit signal can be sent to interrupt the peeling process immediately. In practice this criterion is implemented using a predefined amount of layers (similar to the criterion *Fixed Amount of Layers*), for this reason this criterion is not treated separately.

6 Experimental Results

In order to validate the usefulness of the presented algorithms and stop criteria, experiments with different types of 3D models have to be conducted to compare our methods to the traditional depth peeling approach.

6.1 Experimental Setup

The three algorithms - *classic*, *adaptive* (cf. Section 4), and *reverse* (cf. Section 3) depth peeling - as well as the *Stop criteria No samples Left*, *Few Samples Left*, and *Fixed amount of Layers* (cf. Section 5) have been implemented. This implementation was realized in the C programming language using the OpenGL API to access the graphics hardware.

Four 3D models were chosen for the tests: The *Stanford Bunny* for its non-trivial polygon count but relatively low depth complexity. Due to its low visual complexity, this model is well fitted to compare the algorithms for its rendering quality. The second model is a reconstruction (about 11000 polygons) of the *Stanford Dragon*, which has a high depth complexity when viewed directly from the front (up to 13 layers). This helps making it independent of possible rasterization or fill-rate bottlenecks which might occur with more detailed models. Finally, two practical models from the medical domain: A liver organ including its vessels for VR surgery, and a plant biological model (barley grain) with its inner structure. A list of these models along with their defining characteristics is given in Table 1.

In order to measure the performance of the algorithms independently from other possibly limiting factors in the graphics pipeline, the models were rendered with plain materials of varying colors and 30% opacity. Furthermore, as the models are rendered multiple times per frame, their geometry was copied into the graphics memory utilizing *OpenGL*'s Vertex Buffer Objects. Hence, performance bottlenecks resulting from repeatedly copying vertex data over the graphics bus have been eliminated. The experiments were run on a standard desktop system with an Nvidia QuadroFX 4600 graphics card.

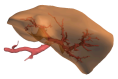
Model	Icon	# of Faces	Depth Complexity
Stanford Bunny		69452	7
Stanford Dragon		11103	13
Liver		51130	11
Barley Grain		2268306	11

Table 1: The 3D models used in the experiments.

6.2 Results

Table 2 shows the results of the four tested models. For each algorithm and stop criterion, the achieved frames per second (FPS) measured on the implemented test system is shown. In order to quantify the *correctness* of an image produced by one of the algorithms, it is compared to the one generated by *classic depth peeling*: First, the number of differing pixels is counted, which yields the *absolute error* (AE). In Table 2, it is given as the fraction of the total amount of pixels that the model occupies in the frame buffer. Second, the average distance between the differing pixels in the Euclidean space is shown as *mean absolute error* (MAE).

The rows titled *All layers* and the ones below present the performance measurements for the stop criterion *Fixed amount of layers*. *All layers* is equivalent to the depth complexity of the tested model and has hence the same output as the classic depth peeling approach. The values below it are the measurements with the last one, two, three, or four layers removed.

		Stanford Bunny			Stanford Dragon		
Algorithm	**Stop Criterion**	**FPS**	**AE**	**MAE**	**FPS**	**AE**	**MAE**
Classic	No Samples Left	90	0.00	0.000	114	0.00	0.000
	1%	156	0.49	0.791	224	2.16	0.187
	5%	240	7.11	74.709	296	19.84	11.076
	10%	242	7.11	74.709	358	44.73	59.503
	All layers	104	0.00	0.000	123	0.00	0.000
	−1	115	0.00	0.000	131	0.00	0.000
	−2	135	0.00	0.002	142	0.00	0.000
	−3	158	0.43	0.210	153	0.00	0.000
	−4	192	3.34	7.178	175	0.00	0.000
Adaptive	No Samples Left	92	5.07	23.671	180	20.36	22.709
	1%	170	5.20	24.112	300	21.00	18.504
	5%	254	7.46	82.248	375	31.55	27.493
	10%	254	7.46	82.248	430	47.50	71.848
	All layers	114	5.07	23.671	186	20.35	18.430
	−1	128	5.07	23.671	196	20.35	18.430
	−2	147	5.07	23.671	210	20.35	18.430
	−3	171	5.20	23.769	223	20.35	18.430
	−4	208	5.20	24.117	241	20.35	18.430
Reverse		90	0.00	0.000	114	0.0	0.000

		Liver			Barley Grain		
Algorithm	**Stop Criterion**	**FPS**	**AE**	**MAE**	**FPS**	**AE**	**MAE**
Classic	No Samples Left	52	0.00	0.000	9	0.00	0.000
	1%	108	3.86	4.064	16	3.53	0.649
	5%	156	12.28	32.62	22	40.16	47.811
	10%	199	18.46	143.6	36	43.36	160.012
	All layers	59	0.00	0.000	9	0.00	0.000
	−1	63	0.00	0.000	10	0.00	0.000
	−2	68	0.00	0.000	11	0.00	0.000
	−3	74	0.00	0.000	12	0.00	0.000
	−4	83	0.00	0.000	14	0.00	0.000
Adaptive	No Samples Left	55	11.74	23.590	10	34.35	13.705
	1%	120	12.90	26.417	16	34.59	13.908
	5%	170	15.55	50.799	22	43.12	53.357
	10%	210	18.46	147.90	36	43.35	160.01
	All layers	64	11.74	23.590	10	34.35	13.705
	−1	70	11.74	23.590	10	34.35	13.705
	−2	76	11.74	23.590	11	34.35	13.705
	−3	84	11.74	23.590	13	34.35	13.705
	−4	93	11.74	23.590	14	34.35	13.705
Reverse		52	0.0	0.000	9	0.00	0.000

Table 2: Results for the tested models

Few Samples Left As can be seen in the graphical representation of the results for the *Few Samples Left* criterion in Figure 6 (a), it increases performance for all of the tested models. It does so even for a parameter of 1% (for the *Bunny* model the FPS increases by 73%), for which the error is negligibly small (cf. Figure 5). Furthermore, Figure 5 depicts the influence of the *Few Samples Left* criterion to the resulting image quality of all tested models. For the parameter 1%, throughout a good image quality could be achieved, but especially for the parameters 5% and 10% the results between models differ a lot. In case of the *Dragon* each fifth pixel is effected, but only in a very small way (in average by 11 units in the Euclidean space). In contrast, in the image of the *Bunny* only each 14th pixel is effected (quite a rate of 1/3 of the *Dragon*), but therefore, the average change of 74 units is significantly higher and causes a worse image quality.

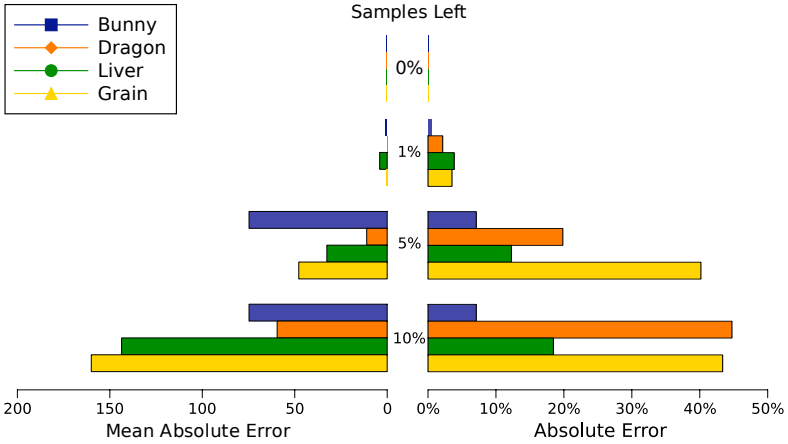


Figure 5: Error metrics for the tested models rendered with the *Few Samples Left* criterion with different parameters indicating the image quality.

Fixed Amount of Layers As expected in Section 5 *Fixed Amount of Layers*, rendering the exact amount of layers necessary is substantially faster for all of the models, see Figure 6 (b). This is due to the fact that this method does not need an occlusion query for each layer. However, the exact number of depth layers necessary has to be known beforehand and changes for different views on a model.

Contrary to the expectation, the tests for the *Limited amount of layers* stop criterion showed for all of the tested models, that the last few layers contribute nothing or negligibly little to the final image. For the *Bunny* model, two layers can be omitted with virtually no degradation of quality, increasing the performance by 52%. The number of layers that can be omitted is even higher for models with greater depth complexity (such as the tested *Liver* model) or higher polygon count (*Barley Grain* model).

Reverse Depth Peeling As expected, the *Reverse Depth Peeling* algorithm performs similar and produces the same output as *classical depth peeling* for all of the tested models. It does however use substantially less video memory.

Adaptive Depth Peeling For models with high polygon counts, the *Adaptive Depth Peeling* approach has not proven to be as effective as expected. The constraining factor for those models seems to be the great amount of faces that the graphics system has to process rather than the actual filling of those faces, which the *Adaptive Depth Peeling* approach is designed to speed up. Therefore, it increases the rendering performance of the relatively low-polygon *Stanford Dragon* model substantially, but has virtually no effect on the other, higher polygon models.

However, *Adaptive Depth Peeling* might also not be a good choice for lower polygon models, as it introduces artifacts into the resulting image. In Figure 7, one can see that many more changes are introduced compared to classic depth peeling with the stop criterion *Few Samples Left* of 1%. The latter is not only more correct, it also outperforms the *Adaptive Depth Peeling* version.

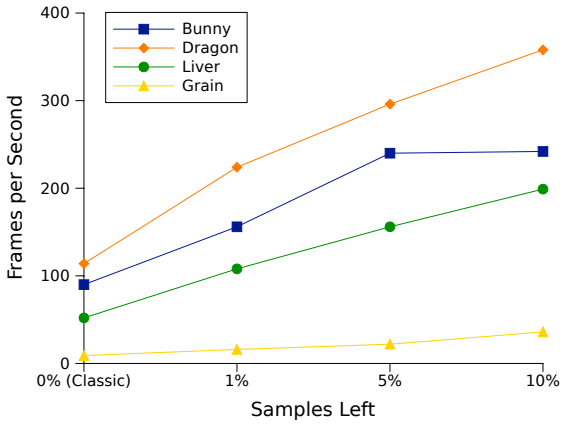
If video memory is the constraining factor, this approach might be adequate (cf. Section 4). Otherwise, the proposed *Stop criteria* yield better results for all tested models and should be preferred.

7 Conclusion

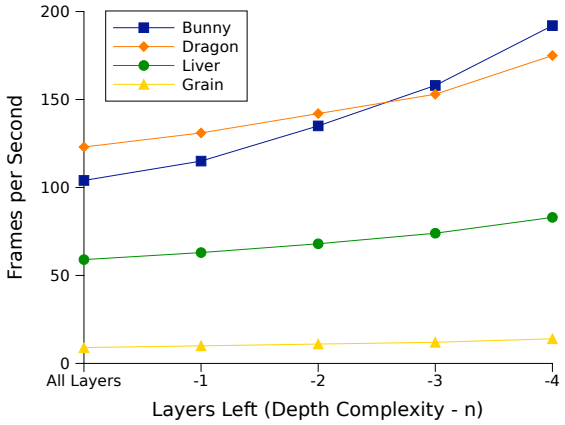
In this paper, different strategies for real-time rendering of complex transparent polygonal models with respect to the number of faces and also to the models' depth complexity were presented. It could be shown that the *Reverse Depth Peeling* approach is suitable for applications where the VRAM is the limiting size. As a drawback, the introduced *Stop criteria* cannot be applied to *Reverse Depth Peeling*. This causes losses in the rendering performance.

The introduced concept of *Adaptive Depth Peeling* by aggregating four rear layers at a time to one layer underperformed especially with respect to the rendering quality. The frame rate could not be increased significantly in comparison to Everitt's classic algorithm with applied *Stop criteria*.

The final concept presented, was the *Stop criteria*. Similar to the early termination of rays in ray tracing [Weiskopf et al., 2004] or integration stop in the radiosity [Prikryl et al., 1999], stop criteria can be used to speed up the rendering of the models to be visualized. The results state that cutting of the rendering process when only a small amount of pixels is still left for computation and / or reducing the depth complexity are very powerful add-ons for depth peeling algorithms. The speed up is partially based on the expense of the rendering quality.



(a)



(b)

Figure 6: Performance characteristics of the criteria: (a) *Few Samples Left* and (b) *Fixed Amount of Layers*.

8 Future Work

For real-time interaction it might be useful to combine the presented concepts. For example, a fast navigation through the model can be done with a reduced quality, whereas in still images full quality transparency determination can be performed.

The next step will be applying the presented *Stop criteria* to the more present depth peeling algorithms which are described in the related work section.

(a) (b) (c)

Figure 7: Comparison of the image quality of *Adaptive depth peeling* (a) and the stop criterion *Few Samples Left* with a threshold of 1% (b) against the correct rendering result produced by the classic depth peeling (c). Pixels which differ from the optimal result (c) are highlighted red.

References

[Aila et al., 2003] Aila, T., Miettinen, V., & Nordlund, P. (2003). Delay Streams for Graphics Hardware. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers* (pp. 792–800). New York, NY, USA: ACM.

[Bavoil et al., 2007] Bavoil, L., Callahan, S. P., Lefohn, A., Comba, J. L. D., & Silva, C. T. (2007). Multi-fragment effects on the GPU using the k-buffer. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (pp. 97–104). New York, NY, USA: ACM.

[Bavoil & Myers, 2008] Bavoil, L. & Myers, K. (2008). *Order Independent Transparency with Dual Depth Peeling*. Technical report, NVIDIA.

[Callahan, 2005] Callahan, S. (2005). The K-Buffer and Its Applications to Volume Rendering. Master's thesis, University of Utah.

[Callahan et al., 2005] Callahan, S., Ikits, M., Comba, J., & Silva, C. (2005). Hardware-Assisted Visibility Ordering for Unstructured Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 11(3), 285–295.

[Carpenter, 1984] Carpenter, L. (1984). The A-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph.*, 18(3), 103–108.

[Catmull, 1974] Catmull, E. E. (1974). *A subdivision algorithm for computer display of curved surfaces*. PhD thesis, The University of Utah.

[Diepstraten et al., 2002] Diepstraten, J., Weiskopf, D., & Ertl, T. (2002). Transparency in Interactive Technical Illustrations. In *Proc. of EuroGraphics '02*.

[Elmqvist et al., 2007] Elmqvist, N., Assarsson, U., & Tsigas, P. (2007). Employing Dynamic Transparency for 3D Occlusion Management: Design Issues and Evaluation. In *Proceedings of INTERACT 2007* (pp. 532–545).

[Everitt, 2001] Everitt, C. (2001). *Interactive Order-Independent Transparency*. Technical report, NVIDIA Corporation.

[Liu et al., 2006] Liu, B., Wei, L.-Y., & Xu, Y.-Q. (2006). *Multi-Layer Depth Peeling via Fragment Sort*. Technical report, Microsoft. URL: `http://research.microsoft.com/research/pubs/view.aspx?tr_id=1125`.

[Mammen, 1989] Mammen, A. (1989). Transparency and Antialiasing Algorithms Implemented with the Virtual Pixel Maps Technique. *IEEE Comput. Graph. Appl.*, 9(4), 43–55.

[Meshkin, 2007] Meshkin, H. (2007). Sort-Independent Alpha Blending. Perpetual Entertainment, Game Developers Conference (GDC).

[Prikryl et al., 1999] Prikryl, J., Purgathofer, W., & L, C. (1999). Perceptually-driven termination for stochastic radiosity. In *In Seventh International Conference in Central Europe on Computer Graphics and Visualization (Winter School on Computer Graphics)* (pp. 418–425).

[Rost, 2006] Rost, R. J. (2006). *OpenGL(R) Shading Language (2nd Edition)*. Addison-Wesley Professional.

[Shreiner, 2004] Shreiner, D. (2004). *OpenGL(R) 1.4 Reference Manual (4th Edition)*. Redwood City, CA, USA: Addison Wesley.

[Weiskopf et al., 2004] Weiskopf, D., Schafhitzel, T., & Ertl, T. (2004). GPU-Based Nonlinear Ray Tracing. *Computer Graphics Forum (Eurographics 2004)*, 23(3), 625–633.

[Wexler et al., 2005] Wexler, D., Gritz, L., Enderton, E., & Rice, J. (2005). GPU-accelerated high-quality hidden surface removal. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (pp. 7–14). New York, NY, USA: ACM.

[Wittenbrink, 2001] Wittenbrink, C. M. (2001). R-buffer: a pointerless A-buffer hardware architecture. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (pp. 73–80). New York, NY, USA: ACM.