A Client-Server Engine for Parallel Computation of High-Resolution Planes

D.P. Gavidia*, E.V. Zudilova, and P.M.A. Sloot

Section Computational Science
Faculty of Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, the Netherlands
daniela@cs.vu.nl, {elenaz,sloot}@science.uva.nl

Abstract. The paper describes a visualization mechanism that permits fast extraction of high-resolution slices of interest from large biomedical datasets stored remotely. To provide the user with real-time interaction, a sub-sampling mechanism of the 3D images has been developed. The 3D sub-sampled view is detailed enough to provide a context and allow a user to orient him/herself. The result of clipping is a high-resolution plane, which can be explored independently from the 3D image. The clipping engine is built as a client-server application, where the computation of high-resolution planes is executed on a server-site in parallel. This approach results in remarkably good performance.

Keywords: Interactive visualization, clipping, data parallelism, client-server, VTK

1 Introduction

Biomedical data is data collected by applying imaging techniques [2] to an object of biological or medical origin. Information sources may be widely distributed, and the volume of datasets to be processed is usually extremely big. As a result, biomedical imaging requires more and more computational resources. Processing, visualization and integration of information from various sources play an increasingly important role in modern biomedicine [9].

Current 3D imaging systems have the capability to produce overwhelming amounts of data converted into visual representations in order to be analyzed. Specialists in medicine and biology rely on these visualizations to achieve a better understanding of the structures and relationships presented in their subjects of study [2].

The improvements in imaging techniques have led to the development of higher resolution devices resulting in ever increasing amounts of data being produced. The problem, though, is not the lack of techniques for extracting information from 3D datasets and visualizing them accordingly, but the shift that is necessary between the traditional practice in biomedicine of visualizing slides in two dimensions and doing so in 3D [7]. Even though 3D data is available, the traditional practice of analyzing

* The author is now a PhD-student at the Department of Computer Science, Vrije Universiteit (Amsterdam, the Netherlands).

the information using 2D slices is still predominant. In order for such shift to occur, applications that use 3D data need to present it in a way that makes exploration and analyses intuitive.

Even though clipping techniques have been known for at least 15 years, they have been usually considered as an additional feature provided by a visualization package or a toolkit. However, clipping can also be an interactive visualization technique that permits to bring to the forefront the information that might be obscured in the density of a dataset, allowing a user to explore data intuitively. More importantly, clipping can help, for instance, in reducing the need for exploratory surgery as well as for dissection, by presenting both the 3D reconstruction of an object and a set of 2D slices, similar to normal scans that physicians usually use in their everyday practice. In this respect the 3D representation helps physicians to orient themselves within an object's geometry and 2D allows them to have a closer look into a familiar setting. The clearness and accuracy of the representation of a slice of interest for most real cases is more important than for the 3D representation of an object as a whole.

The research presented in this paper is focused on the development of a new visualization mechanism for the fast extraction of high-resolution clipping planes from a large datasets stored remotely, for example, on the GRID [9]. Section 2 describes the main design concept on which the clipping engine is based. The technical aspects of the implementation are given in section 3. Section 4 focuses on the parallel computations needed for the calculation of a high resolution plane done on the server site. The results showing the benefits of parallel processing are presented here as well. Conclusions and final discussions can be found in Section 5.

2 Design Concept

3D biomedical data is usually acquired by means of such imaging techniques as Computed Tomography (CT), Magnetic Resonance Imaging (MRI) or Positron Emission Tomography (PET). The choice of the imaging technique is determined by the structure or anomaly that needs to be observed, given that some techniques are better suited for certain cases than others.

Data produced by imaging techniques can vary from relatively small (e.g., 64 x 64 x 32 voxels) to extremely large (e.g., 2048 x 2048 x 1024 voxels). In the case of large datasets, when heavy processing is involved, the interactivity is usually hindered by the time required both for the data transfer from the remote repository and visualizing it on the user site.

One possible solution to reduce this delay is data compression, which provides good performance results (see for instance [3]) and is widely used. Other works (e.g. [1]) report on methods like sub-sampling for finding a balance between interactive performance and accuracy of the data presented in 3D, where interaction is possible with an acceptable delay and the result presented to a user is detailed enough to allow a thorough understanding of the data analyzed.

The potential users of a clipping engine are physicians who are used to work with 2D scanned images. 3D representation hardly plays a role for them, while 2D representation of a slice of interest is of the utmost importance. That is why it has been decided to use sub-sampling to provide a user with the real-time interaction capabilities.

Fig. 1. Clipping engine: GUI screenshot

The goal of this project is to allow for visualization of data resulting from biomedical imaging by interacting with a 3D representation of a dataset. This is achieved through the use of iso-surfaces as a means of giving a user an idea of the content of the dataset and clipping planes that allow a more in-depth analysis of the characteristics of the data observed. Surface rendering has been selected for the visualization of a static object in 3D because it gives similar possibilities for data exploration as volume rendering but is less computational and time consuming.

The GUI of the application contains two viewers (see Fig. 1): one presents the 3D object and another one shows a high-resolution slice of interest, which has been generated as a result of the intersection of a 3D object with a clipping plane built by a user.

3 Communication Considerations

3.1 The Client-Server Model of the Application

Imaging devices can produce enormous amounts of data, which are frequently stored remotely in high-capacity storage facilities. The size of these data files makes it inconvenient or impossible to have them stored locally. This leads to the natural choice for building a clipping engine as a client-server application. Additionally, there are two other reasons to implement the clipping engine using a client-server model:

1. The computation of a high-resolution plane is expensive due to the need to find the points that make up the plane through interpolation. This can be quite time-consuming. Assuming that better computational resources are available at the server, doing this operation at the server-site might yield better performance.
2. If high-performance computational resources are available at the server (e.g., a parallel super computer), performance can be further improved by conducting the computations of a high-resolution plane in parallel on the server site.

The clipping engine functions as follows. A sub-sampled dataset is transferred to the client-site and used for further interactions. A user can manipulate the 3D representation of the data and choose the clipping plane in real-time. Once a selection has been made, the client communicates with the server to obtain the desired high-resolution slice of interest.

The client-server implementation relieves the client from having to store enormous data files and having a fast processor to compute planes. His/her main concern would then be to have a fast enough graphics workstation for visualization and rendering.

The flow of data between client and server occurs in an asynchronous manner, with the server sending subsets of the scanned data by request through Remote Method Invocations (RMI). According to the flow direction, the data can be defined as follows. *Control data* is the data sent from the client to the server, comprised of the requests for data made by the client, such as the definition of the desired clipping plane or the sample rate for a sub-sampled dataset. *Computed data* is the data sent from the server to the client. In general, it is a subset of the scanned data sent to the client by request.

Control data is limited to a few bytes since it is comprised of requests for data. On the other hand, computed data, being a subset of the scanned data, results in a significantly larger amount of information being sent.

3.2 Implementation

The system runs on Linux and Unix computers. For the purpose of visualization and image processing, the Visualization Toolkit (VTK) [8] has been chosen. Besides visualization algorithms provided by the VTK, support for parallelization and running remote processes has played an important part in the development of the application. The GUI was built using the QT GUI application framework. VTK is cleanly integrated into the QT interface through the use of the VtkQt library [10]. VtkQt is an extension of VTK developed at the Section Computational Science of the University of Amsterdam. It allows for the insertion of VTK windows in the QT interface and transparent use of VTK classes alongside Qt, simplifying the development.

The CT scans of the coral “*Madracis mirabilis*” used for experiments have been made in a joint project on modeling and analysis of growth and form of corals in collaboration with the Netherlands Institute for Sea Research and Cooperative Institute for Marine and Atmospheric Studies, USA [5].

3.3 Operations and Communication

Figure 2 gives a general overview of the way the client-server clipping engine operates. The communication between client and server is triggered by two events: a request for a sub-sampled dataset and a request for a clipping plane. To obtain a sub-sampled data set, a sample rate value is sent to the server, triggering the execution of a sampling routine and generation of the sub-sampled dataset. In a similar manner, the client can request a clipping plane. In this case, the plane definition - a point in the plane and the normal of the plane - is sent to the server, triggering a routine for extracting the plane. The result is then made available to the client.

Fig. 2. Pipeline of the operation in the client-server application. The communication between client and server is handled by *vtkSocketController*, *vtkOutputPort* and *vtkInputPort*

The communication between client and server is handled by three VTK classes: *vtkSocketController*, *vtkOutputPort* and *vtkInputPort*, as illustrated in Fig. 2. The control data is sent from the client to the server through the *vtkSocketController*. After the necessary computations are performed on the server-site, the resulting datasets (computed data) are sent back to the client through the *vtkOutputPort* and *vtkInputPort* objects. These two classes allow for the construction of a pipeline that starts on the server-site and continues on the client-site.

Several RMI methods are written on the server and registered with the *vtkSocketController*. The *vtkSocketController* on the client connects to the one on the server using the name of the server and a designated port number. Once the connection is made, the client application triggers the RMIs to communicate changes in the parameters controlled by a user, namely sample rate for the dataset and definition of the clipping plane. The server then modifies these parameters in the server-site pipeline and makes the results available through the *vtkOutputPort* objects.

With the sub-sampled dataset, the client can generate iso-surfaces in accordance with the parameters set interactively by a user through the GUI. Iso-surfaces are rendered in a 3D render window along with the low-resolution clipping plane. A high-resolution slice of interest is generated only after a corresponding request is sent to the server. Then a high-resolution 2D image is calculated in a parallel on the server and rendered on the client-site.

Fig. 3. Data parallelism. Each process, except one reserved for appending the results, computes part of a high-resolution plane. In the figure, the computation is done using 3 processes. Process 0 appends the results from all the processes together.

4 Parallel Data Processing

For the extraction of the arbitrary plane from the dataset, the class *vtkCutter* has been used. This class is a filter that cuts through a dataset. It creates a polygonal surface corresponding to the implicit function $F(x, y, z) = \text{value}(s)$. In this case, the implicit function is a plane function with the desired position and orientation. Cutting with an implicit function is time-consuming, which hampers interactive performance. Taking advantage of VTK's support for parallel computing, a parallel implementation of the plane computation is in order to speed up the process. VTK supports task, pipeline and data parallelism. Data parallelism is useful for processing extremely large datasets, not unlike the ones produced by medical [1].

In order to reduce the time required to obtain the high-resolution plane after its request by the client, the calculation of the plane has been done in parallel. The implementation uses multiple processors that communicate through MPI (see Fig. 3). Each process works on a different piece of the original dataset producing a section of a high-resolution plane. These results are appended together by process 0 using a *vtkAppendPolyData* filter, which has as inputs the sections of the plane calculated by each of the other parallel processes. The output is a single high-resolution plane.

VTK handles data parallelism through parallel streaming, where the data is partitioned into independent subsets using VTK's streaming data model [6]. The *vtkAppendPolyData* filter implements this functionality. Coding is simplified considerably since it is not needed to explicitly assign a different subset of data for each process to work on. Instead, in parallel streaming mode the *vtkAppendPolyData* filter asks for a different piece of the plane polydata from each of its inputs (the outputs of the other processes). When the update request is propagated up the pipeline

Fig. 4. Time measurements (left) and speedups (right) for the calculation of a high-resolution plane of dimensions 512x512 with 16 bits per sample when done in parallel. Process 0 was not taken into account when considering the number of processes running in parallel.

of each process, the request for a subset of polydata is translated into a request for an independent subset of the input dataset (i.e. the volumetric data).

Fig. 4 shows the time required to calculate the plane in parallel using an increasing number of parallel processes and the associated speedup, demonstrating significant improvements in performance. The parallel implementation of the high-resolution plane computation was tested on the Distributed ASCI Supercomputer 2 - DAS-2 [4].

5 Conclusions and Future Work

Interactivity and accuracy are main concerns for the exploration of the biomedical data. Sub-sampling has been used to make interactivity possible in real-time, but the 2D slices extracted from a sub-sampled dataset might not be detailed enough. It is desirable to use the totality of the data available for the extracted slice. However, the computation of a high-resolution plane can be very time-consuming when done on a single processor, crippling the performance. The parallel computations on the server-side ease this problem and speedup the process significantly.

In respect to the graphics hardware available on the client and the complexity of the dataset, a higher or lower sample rate would be required for the interaction in real-time. Currently, the determination of an appropriate sample rate is given to a user. Though finding the optimal sample rate falls beyond the scope of this project, implementing an intelligent way of doing this would certainly prove beneficial. Moreover, in cases when the client is able to handle the entire dataset without a noticeable effect towards the interaction capabilities, it would be reasonable to avoid sub-sampling and to work with the entire dataset on the client.

Acknowledgements. The authors would like to thank Dr. Jaap Kaandorp and Dr. Robert Belleman for their input to the project and Denis Shamonin for his technical assistance.

References

1. Ahrens J., Law C., Schroeder W., Martin K., Papka M.: A parallel approach for efficiently visualizing extremely large time-varying datasets. Tech. Rep. LAUR-00-1620, Los Alamos National Laboratory, 2000.
2. Belleman R.G. and Sloot P.M.A.: Simulated vascular reconstruction in a virtual operating theatre. Proceedings of the CARS2001 Conference, Elsevier Science B.V., Berlin, Germany, June 2001, pp. 938-944.
3. Belleman R.G., Shulakov R.M.: High performance distributed simulation for interactive simulated vascular reconstruction. Proceedings of The 2002 International Conference on Computational Science (ICCS 2002), Amsterdam, the Netherlands, April 2002. Series Lecture Notes in Computer Science (LNCS) volume 2331, ISBN 3-540-43594-8, Springer-Verlag.
4. Distributed ASCI Supercomputer 2 home page: <http://www.cs.vu.nl/das2/>.
5. Kaandorp J.A., Koopman E.A., Sloot P.M.A., Bak R.P.M., Vermeij M.J.A. and Lampmann L.E.H.: Simulation and analysis of flow patterns around the scleractinian coral *Madracis mirabilis* (Duchassaing and Michelotti). Phil. Trans. R. Soc. Lond. B DOI:10.1098/rstb.2003.1339, 2003.
6. Law C.C., Schroeder W.J., Martin K.M., Temkin J.: A multi-threaded streaming pipeline architecture for large structured datasets. Proceedings of Visualization'99 (October 1999), IEEE Computer Society Press.
7. Robb R.A.: Handbook of Medical Imaging: Processing and Analysis. Isaak N. Bankman, Academic Press, 2000, Ch. Three-Dimensional Visualization in Medicine and Biology, pp.685-712.
8. Schroeder W., Martin K., Lorensen L: The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. 2nd Edition. Prentice-Hall, Inc., 1997.
9. Sloot P.M.A., Albada van G.D., Zudilova E.V., Heinzlreiter P., Kranzlmüller D., Rosmanith H., Volkert J.: Grid-based Interactive Visualisation of Medical Images, Proc. of the 1st European HealthGrid Conference, Lyon, France, January 2003, pp.57-67.
10. VtkQt web-page: <http://carol.wins.uva.nl/dshamoni/VtkQt>