# Isosurface Extraction Techniques for Web-based Volume Visualization

Klaus Engel * Rüdiger Westermann † Thomas Ertl *

* University of Stuttgart, IfI, Visualization and Interactive Systems Group
† Dept. of Computer Science, University of Utah

## Abstract

The reconstruction of isosurfaces from scalar volume data has positioned itself as a fundamental visualization technique in many different applications. But the dramatically increasing size of volumetric data sets often prohibits the handling of these models on affordable low-end single processor architectures. Distributed client-server systems integrating high-bandwidth transmission channels and Web-based visualization tools are one alternative to attack this particular problem, but therefore new approaches to reduce the load of numerical processing and the number of generated primitives are required. In this paper we outline different scenarios for distributed isosurface reconstruction from large-scale volumetric data sets. We demonstrate how to directly generate stripped surface representations and we introduce adaptive and hierarchical concepts to minimize the number of vertices that have to be reconstructed, transmitted and rendered. Furthermore, we propose a novel computation scheme, which allows the user to flexibly exploit locally available resources. The proposed algorithms have been merged together in order to build a platform-independent Web-based application. Extensive use of VRML and Java OpenGL-bindings allows for the exploration of large-scale volume data quite efficiently.

**Keywords:** Volume visualization, Isosurface reconstruction, Distributed Systems, Web-based Applications

## 1 Introduction and Related Work

The problem of how to convey a visual representations of realistically sized scalar volume data is still one of the most challenging research areas in scientific visualization. In particular, the extraction of isosurfaces [13, 16] has positioned itself as a fundamental visualization technique and a lot of effort has been made during the last years to come up with optimized algorithms. The goal of these approaches is manifold: Mesh decimation and reorganization techniques [5, 9, 10, 17, 20, 21] try to convert the outcoming surface representation in such a way that the number of primitives to be processed is reduced in order to render the surface at improved speed. In these approaches the mesh optimization is left out as a post process, but it can also be directly accomplished during the reconstruction phase itself [19]. Other techniques explicitly address the problem of optimizing the cell traversal procedure in order to avoid the processing of regions that do not contain the surface. The benefits of octrees for faster reconstruction of isosurfaces from regular volume data were first recognized in [26]. Several related techniques have been developed during the last years, like improved partitioning and incremental update techniques [2, 22, 23], efficient cell search with interval data structures [3, 4, 12], and most recently techniques that try to take advantage of the hierarchical nature of multi-resolution representations [18, 25]. Hybrid approaches as proposed in [11] try to avoid reconstructing primitives that do not contribute to the actual view by effectively integrating view dependent occlusion culling.

Most importantly, the motivation in all these approaches is to develop algorithms that react to changes of mapping parameters (e.g. varying the iso-value) by almost immediately regenerating the corresponding geometrical representation which then ought be rendered with several frames per second. Only with this type of real-time interaction and navigation it is possible to effectively analyze an unknown data set and to compensate for the information lost during the projection of the 3D scene onto the screen. However, despite all the sophistication incorporated into these methods, does it seem that the data sets are growing faster than algorithmic progress is made. For example data volumes from 3D medical imaging like CT are approaching sizes of $512^3$ which amounts to more than 100 million of voxel cells.

Particularly in Web-based applications, where in general there's no access to large-scale computing environments, new techniques have to be developed that allow for an effective reduction of the computational expense of the reconstruction process and of the number of generated primitives to be viewed on affordable low-end computers. In addition, new systems have to be designed taking into account current Web-standards and visualization options.

Just recently, since platform-independent programming languages and data exchange formats like Java, VRML and HTML are available, first Web-based visualization services were presented. In [24] a visualization tool for flow data was developed, which allows the user to down-load local data on a remote server and to request visualization services. In [15] a platform independent visualization tool including isosurface reconstruction, cutting planes and elevation plots for 2D and 3D datasets were proposed based on the Java programming language. Progressive reconstruction and transmission of isosurfaces from tetrahedral grids was emphasized in [7]. The goal was to reduce the amount of data to be reconstructed and transmitted by giving the user a level-of-detail control. The global nature of the progressive isosurface generation, however, is also the main disadvantages of this system. When searching for particular details the user always has to request the highest level-of-detail of the isosurface. As a consequence the advantages of a progressive transfer and display vanish.

In order to develop efficient Web-based isosurface visualization techniques several different aspects have to be considered. Some of them we will refer to in the following sections. In section 2 we will outline new techniques to re-organize the isosurface during the reconstruction phase thus reducing the number of primitives to be processed. Section 3 focuses on hierarchical techniques, which allow for a considerable acceleration of the entire *mapping-render* cycle. Aspects of how to effectively compress the generated information will be emphasized in section 4. Finally, in section 5 we will describe our Web-based application, and we will compare and analyze our results.

Let us proceed by organizing the Marching Cubes algorithm into a pipeline, which transforms the volume data into several other data types prior to rendering the reconstructed primitives (Fig. 1). First the volume data is loaded into main memory by a read module. A filter module selects all cells containing the isosurface. The inter-

*Institut für Informatik, Abteilung für Visualisierung und Interaktive Systeme, Breitwiesenstr. 20-22, 70565 Stuttgart, Germany, Email: `engel@informatik.uni-erlangen.de`

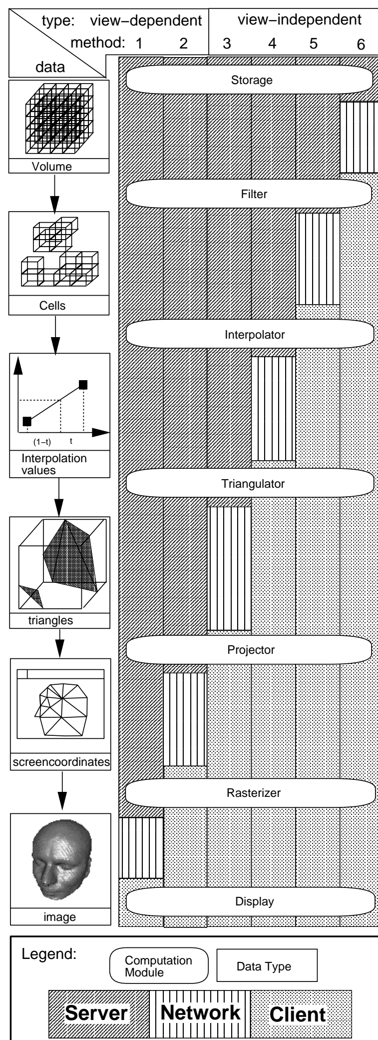


Figure 1: Marching Cubes Scenarios

polator module interpolates along the cell edges. The interpolation weights are successively used by a triangulator module that produces triangle data. Triangles are projected onto the viewing plane and mapped into screen coordinates by a projection module. Then they are rendered into the frame buffer by the rasterization module, and finally a display module shows the rendered images.

In a Web-based application these modules can be distributed in various ways between the clients and the servers:

1. All modules except for the display module are located on the server. The client serves as a display station for the transmitted images only. The surface extraction and rendering is completely done on the server.

2. In this scenario the screen coordiantes of triangle vertices along with the colors at the vertices are transferred to the client machine. The rasterization is done by local client hardware.

3. The client side is rendering triangles which were transferred from the server. The computation of the isosurface is done on the server side.

4. In this scenario the interpolation values along the cell edges and the Marching Cubes cases are transferred over the network connection. The setup of the triangles and the rendering is done on the client side.

5. The server is used to filter cells from the volume dataset which are intersected by the isosurface. These cells are transferred to the client, which is performing the rest of the computation and rendering.

6. In this case the server is only used for data storage and handling of volume datasets. The data is completely transferred to the client side, which is performing the computation and rendering of the isosurface.

If the client machine is not able to render the images by itself, i.e. because of lack of 3D graphics acceleration support, scenario 1 can be used. Most PC boards are very slow at transforming geometry, but have fast rasterization engines. In this case scenario 2 can be used. Scenario 6 can be applied if the volumes can be completely transmitted to the client because of their small size or because of the high bandwidth of the network. This scenario is also useful if only parameters of the volume have to be transmitted from the server and the volume data can be computed on the client (e.g. molecular orbital visualization).

A framework which is based on scenario 1 was presented in [8]. The framework enables the remote control of an OpenInventor application. Images generated by a high-end visualization server are streamed through the network to visualization clients using video-streaming codecs. The visualization on the server is controlled by CORBA (Common Object Request Broker Architecture) requests generated on the client machines. CORBA provides access to the visualization server from a large variety of client architectures. A web-based teleradiology system was presented which is based on the proposed techniques.

While scenarios 1 and 2 are view-dependent and require new transmission of data for new viewpoints, scenarios 3,4,5 and 6 allow local interaction with the transferred data. Since we are interested in visualizing large scale volume datasets on client machines with local rendering capabilities across large variety of network connections this paper will focus on the scenarios 3, 4 and 5.

# 2 Advanced isosurface representation

Throughout our scenarios we will assume that the distributed environment consists of a memory and compute server providing sufficient resources to store and process the data set, a client that is equipped with dedicated graphics hardware, e.g. a low-price graphics adapter, in order to support the rendering of 3D polygonal models, and a communication channel as it is commonly accessible by home site applications. In particular we will demonstrate that the server does not necessarily have to be a parallel architecture and that as a consequence the illustrated scenarios do not imply any sophisticated hardware requirements. Thus, one major advantage of the proposed scenarios is that they are not restricted to typical large-scale computing environments.

Even if the isosurface could be reconstructed on the server without any delay, in general the number of generated primitives is most likely to increase the amount that can be transmitted and rendered without considerably deteriorating the performance of the entire application. In our first approach we will address the problem of reconstructing the surface in such a way that reduces the load of geometry transfer and processing across the communication channel and on the client side.

## 2.1 Reconstructing stripped surfaces

One way to compress a triangle mesh is to reorganize multiple triangles into one triangle strip. For each vertex presented after the first two vertices one triangle is defined. Obviously, organizing triangle meshes as strip sets considerably reduces the number of

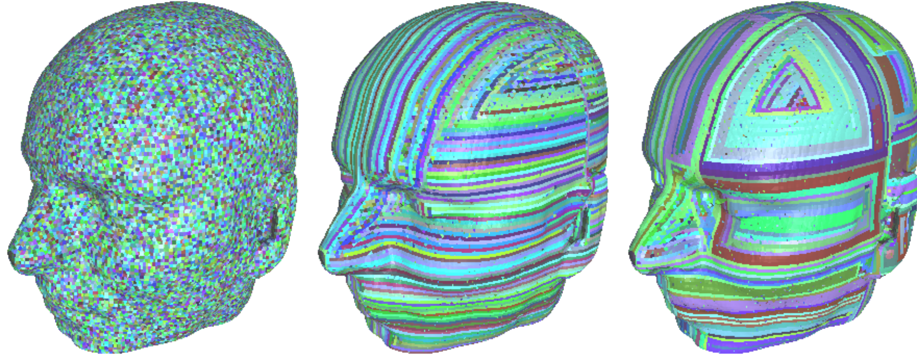


Figure 2: All three images demonstrate the results of different alternatives to directly reconstruct triangle strips (colored consistently) from scalar volume data. Left: strips are not connected across cell boundaries. Middle: strips are connected across cell boundaries but only into one particular direction. Right: the same as in the middle but different alternative directions are possible to continue a strip. The average strip length was 2, 7 and 14, from left to right.

vertices needed to represent the mesh and thus the number of operations necessary to render the mesh. By providing a method that is capable to directly reconstructing stripped isosurfaces we simultaneously address the problem to reduce the load on the transmission channel and to optimize the rendering performance.

Although on common graphics architectures a specific strip length is given that allows for optimal rendering performance it is obvious that the longer the generated strips are the more the transmission load is reduced. On the client side, however, strips of optimal length can be built prior to their rendering.

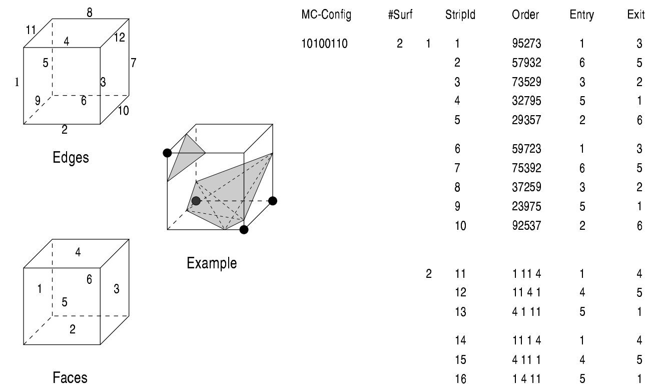
We start by reorganizing the standard MC-table in such a way that for each particular configuration the set of triangles is coded as triangle strips. Then the MC-algorithm is performed as usual and the reconstructed strips are sent to the client (leftmost image in Figure 2). However, as one would expect the average strip length is rather small since no effort has been made to connect strips from adjacent cells.

In order to increase the average strip length we extend the MC-table as outlined in Figure 3. For each configuration we store the number of surface patches passing through the cell, and we additionally store all possible regular strips together with the faces on which they start and stop. Note that for each strip we also store its "twin", which starts on the same face but with reversed vertices. At this point we should mention that different strips will apparently lead to different triangulations of the surface, but since in general no unique rule for the triangulation can be specified it can be chosen arbitrarily.

Once a start cell is selected the first strip is chosen from the appropriate table entry and the surface is tried to be continued to one of the cells direct neighbors. This neighbor is uniquely determined by the exit face and the last two vertices of the actual strip. All vertices of the next strip but the first and the second one have to be reconstructed.

We proceed by repeating this procedure until the next cell would be outside the volume or has already been processed. In order to check whether there are still unused strips in a cell we store one additional byte on a per-cell basis. Whenever a cell is processed for the first time each surface patch $n \in (1,\#\text{Surf})$ passing through that cell is coded by masking the appropriate bit $n$. When a patch is included to build a strip the corresponding bit is set to zero.

Since strips that are built in this way can only be connected to the neighbor determined by the exit face of the previous patch it is impossible to proceed even if there are still neighbors in other



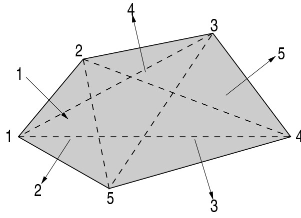

| MC-Config | #Surf | | StripId | Order | Entry | Exit |
|---|---|---|---|---|---|---|
| 10100110 | 2 | 1 | 1 | 95273 | 1 | 3 |
| | | | 2 | 57932 | 6 | 5 |
| | | | 3 | 73529 | 3 | 2 |
| | | | 4 | 32795 | 5 | 1 |
| | | | 5 | 29357 | 2 | 6 |
| | | | 6 | 59723 | 1 | 3 |
| | | | 7 | 75392 | 6 | 5 |
| | | | 8 | 37259 | 3 | 2 |
| | | | 9 | 23975 | 5 | 1 |
| | | | 10 | 92537 | 2 | 6 |
| | | 2 | 11 | 1 11 4 | 1 | 4 |
| | | | 12 | 11 4 1 | 4 | 5 |
| | | | 13 | 4 1 11 | 5 | 1 |
| | | | 14 | 11 1 4 | 1 | 4 |
| | | | 15 | 4 11 1 | 4 | 5 |
| | | | 16 | 1 4 11 | 5 | 1 |

Figure 3: The extended MC-table used to directly connect strips across cell boundaries. Note that surface patches are ordered with respect to decreasing number of vertices since longer strips should be selected first.

directions that have not been processed yet. This restriction results in the long but thin strips shown in the middle image of Figure 2.

In order to further increase the average strip length additional information is stored in the MC-table. Now we also include non-regular or degenerated strips, which are characterized by repeated vertices (see Figure 4).

Since for certain combinations of entry and exit cells there might be no regular strip we have to construct degenerated ones in order to connect the strips properly. At first glance it seems to be rather strange to include redundant information into the vertex stream, but it allows us to continue strips further on without starting a new one. Strips that only consist of one redundant vertex are always superior to a new one for which two new vertices have to be added. Problematic cases occur if a vertex is included more than twice. Then it would be more efficient to include one regular strip and to start a new one. Concerning the transmission load both strategies perform equally well, but if a degenerated strip is to be rendered there might be additional, possibly degenerated triangles that have to be rendered too. However, as will be demonstrated in the results sec-



| Consistent | Entry | Exit |
|---|---|---|
| 12534 | 1 | 5 |
| **Degenerate** | **Entry** | **Exit** |
| 125354 | 1 | 3 |
| 125423 | 1 | 4 |
| 1213415 | 1 | 2 |
| ↓ | | |
| 1243 415 | 1 | 2 |

Figure 4: For each strip all alternative representations are stored that are necessary to proceed to arbitrary neighbors. This has to be done for all possible entry faces. Now, strips are ordered with respect to increasing number of vertices since regular strips should be used first.

tion, constructing strips in this way still significantly decreases the overall number of vertices to be reconstructed and rendered.

Repeated vertices are coded quite effectively without increasing the transmission load. Once within a certain cell a vertex is used again we know at which position it was stored in the actual strip. The distance from this position to the current one is coded in the sign bits of the $(xyz)$-components of the actual vertex. The least significant bit is coded in the sign of the $z$-component and so on. Since we only have distances less than seven, three bits suffice to properly code the distance information. In order for the approach to work the volume bounding box is supposed to be positioned in the positive octant, and as a consequence the coding scheme is not in conflict with the vertex positions. Whenever the client receives a negative vertex component it decodes the sign bits into distances and uses the appropriate vertex from the already generated part of the strip.

## 2.2 Level-of-detail isosurface reconstruction

Although in the previous approach the number of primitives was reduced in a quite efficient way by reorganizing the geometric representation, usually interactivity cannot be achieved particularly for large-scale data sets. In general, we believe that in the proposed environment interactive frame rates can only be achieved by taking advantage of the hierarchical nature of multi-resolution techniques to effectively decimate the number of primitives needed to represent the surface.

In [25] an adaptive octree-based approach to the reconstruction of isosurfaces from regular volume data at arbitrary levels of detail was presented. The algorithm has been designed to enable real-time navigation through complex structures while providing user-adjustable resolution levels. Adaptive on-the-fly reconstruction and rendering is performed from a hierarchical octree representation thus allowing the user to interactively browse through all possible isosurfaces. Special attention was paid to the fixing of discontinuities where different levels are adjacent to each other.

In order to allow for the exploration of even the highest resolution data sets the user can manually select a region of interest in which the surface is reconstructed at the finest level. Outside of this region the surface is reconstructed at increasingly coarser resolution. In this way it is possible to adaptively select the size and the number of details that should be reconstructed at the same time achieving sufficient frame rates for interactive navigation.

In order to optimize this approach for its use in a distributed Web-based scenario several improvements have been added:

**Recursive reconstruction:** The reconstruction process is started from the focus point thus recursively growing the surface around the center of interest. This allows the user to stop the reconstruction if the desired features have been displayed. Whenever a particular inner node of the octree is going to be processed, its children are sorted with respect to their distance to the focus point and they are processed successively in this order. Note that sorting of the original cells is not accomplished in order to save unnecessary computations.

**Level-wise stripping:** We allow the user to select an arbitrary resolution level from which the stripped surface is to be reconstructed. Although stripping of a multi-resolution representation could be accomplished quite easily, it is in general not possible to connect strips across level transitions due to the special treatment of boundary cells. Since this results in rather short strips it has not been considered in our implementation.

**Object space clipping:** Arbitrary clipping planes can be selected on the client side. The plane parameters are sent to the server and considered during the octree traversal. It is now possible to view structures inside the object, which are usually hidden by others. Furthermore, a huge number of operations can be saved since only the visible half-space has to be traversed hierarchically.

**Optimized triangulation:** We did modify the treatment of level transitions in order to minimize the number of triangles while still achieving continuous surfaces. In [25] it was proposed to split triangles generated on the coarser level into triangle fans thus avoiding T-vertices (see Figure 5 A). In the current implementation a *continuous* scalar field is maintained even if the extraction level changes

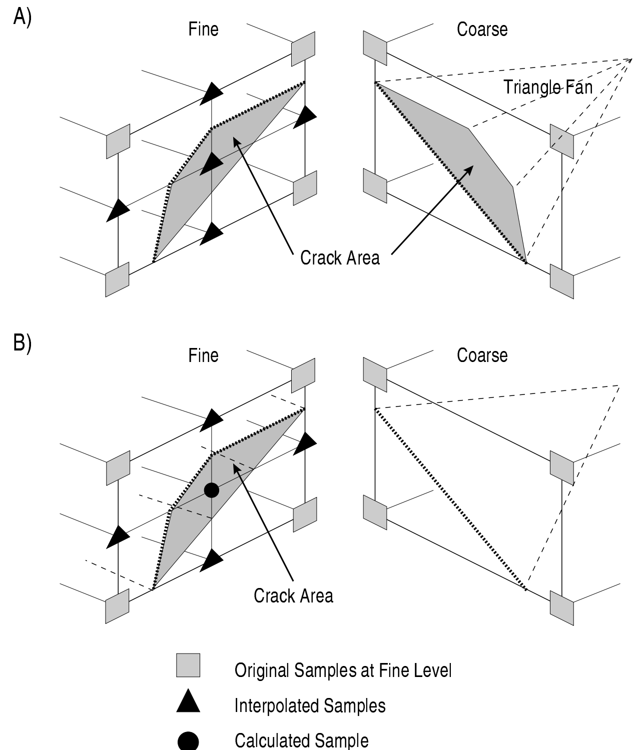



Figure 5: Reconstruction of continuous surfaces by multi-level crack fixing.

by adjusting the scalar values at those cell faces where a level transition occurs (see Figure 5 B): Continuity is established by letting the coarser cell sample the scalar field from the finer level at the even indexed voxels. The odd indexed voxels on the finer level have to be recomputed by linear interpolation in turn. The value at the cell face midpoint, however, has to be recalculated in order to obtain the same surface on either of both sides. This can easily be achieved by considering the line equation obtained from the intersection with the coarser cell edges and the already interpolated data values. Now the triangles generated on the coarser level can be used without any further modifications. Obviously this approach will result in T-vertices, but on the other hand it reduces the number of generated triangles and it allows one to use the standard MC-table without any modifications once the face midpoint has been calculated.

Figure 6 demonstrates the adaptive level-of-detail isosurface reconstruction from an octree hierarchy maintaining continuous transitions between different resolution levels.

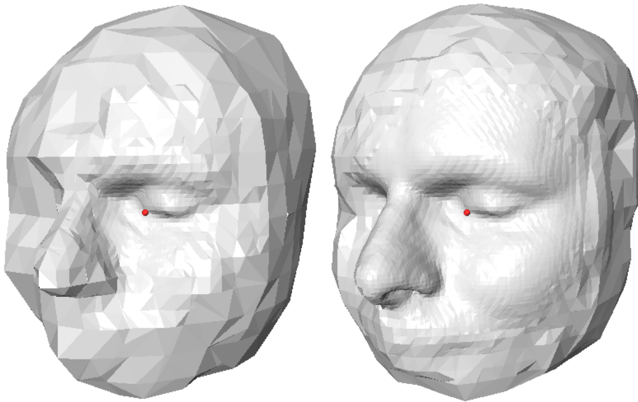


Figure 6: Adaptive level-of-detail isosurface reconstruction from a multi-resolution hierarchy. In the right image the radius of the region around the focus point in which the surface is reconstructed from the original data samples is increased as compared to the left image.

So far, we proposed different alternative approaches to reduce the number of geometric primitives that have to be reconstructed, transfered and rendered. We did not consider to further compress the transmitted data stream and we did not exploit any locally available capacities on the client. However, since usually the client is equipped with a more or less powerful processing unit it seems to be reasonable to make use of these resources in a distributed environment. In the following we will demonstrate how to effectively take advantage of these resources in order to further reduce the load on the transmission channel and to improve the performance of the entire reconstruction process.

# 3 Distributed isosurface reconstruction

By carefully profiling the previous approaches it turns out that a huge amount of time is spent in traversing the hierarchical data structure, testing cells for possible refinements and transferring the primitives across the communication channel. Quite often the client is waiting for the next chunk of data to be received. On the other hand, direct access to the data is in general not possible at the client side: Just because realistically sized data sets *cannot* be stored and processed on affordable low-end systems we focus our research on distributed isosurface reconstruction techniques.

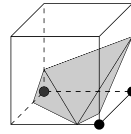



Figure 7: Transmission of vertices compared to the transmission of the MC-configuration and interpolation weights or data samples. All data except the configuration and the octree level is assumed to be encoded in four bytes per value. Note that in real data sets the selected MC-configuration is supposed to occur most frequently [16].

But we can exploit locally available computing power of the client system to reduce the idle times and to balance the load on all involved units more equally. Therefore we developed a distributed computation scheme which allows vertex positions to be calculated on the client. Two different scenarios are proposed, which effectively unload the server from numerical calculations.

First, we let the server perform the standard (maybe octree-based) MC-algorithm but it stops right before vertices have to be calculated from the already computed interpolation weights along the cell edges. These weights, the current MC-configuration and the octree level as well as the position of the cell within that level are sent to the client (see Figure 7A). Actually, the cell index is encoded in four bytes, which limits the maximal data size to $2048^2 \cdot 1024$.

From the illustrated example we observe that compared to the standard MC-algorithm and a cell-wise encoding of triangles into strips, a considerable amount of data to be transfered can be saved. Even if the surface patch can be inserted into a regular strip we can still make some progress. However, in order to make a fair comparison we also have to consider that multiple transmissions of interpolation weights from different cells along the same edge could be avoided. Furthermore it seems to be doubtful whether these values actually have to be encoded in four bytes. In our test cases we usually encode them in one byte without that we ever noticed visible differences in the reconstructed isosurface.

A different approach would be to just let the server act as a data manager, which stores and traverses the octree representation in main memory and transfers the appropriate data samples to the client for further use. The server stops the reconstruction process once the data samples defining the actual cell have been accessed and the MC-configuration is determined. The configuration, the samples necessary to compute the surface passing through that cell and the level as well as the position of the cell within that level are sent to the client (see Figure 7B).

Apparently, the amount of data to be transfered is not reduced since instead of one interpolation weight we have to send two data samples in general. However, if voxel values are given in 8 or 16 bit format this scheme might be superior to others since the transmission load can be further reduced without worsening the accuracy of vertex positions to be computed.

In both scenarios the same MC-table has to be stored on the server and the client and the position of the volume bounding box

has to be defined globally. Prepared with this information, in either case the exact vertex positions can be computed straight forwardly. All data specific information is sent from the server to the client once a new data set is loaded.

# 4 Rendering

Platform-independence and use of client-side rendering acceleration hardware were important objectives for the choice of the used programming languages and rendering APIs. Java was chosen because of its wide availability throughout platforms and Web-browsers.

The Virtual Reality Modeling Language (VRML)[1] and the External Authoring Interface (EAI) [14] enable the use of 3D graphics acceleration hardware for the visualization of isosurfaces while maintaining platform independence of Java code.

Alternatively, because of limitations of *VRML*, *OpenGL* bindings were chosen in order to satisfy the requirements of an interactive Web-based isosurface streaming system. In contrast to *VRML/EAI*, *OpenGL*-bindings allow for the direct access to the rendering pipeline. They are also available for a large number of platforms and provide hardware accelerated rendering inside browser windows.

Not only the visualization but also interaction with the displayed isosurfaces can be done using these APIs. A user is able to place a focus point and to move it along the isosurface just by clicking and moving the mouse. Alternatively the focus point can be moved freely in space by dragging arrows which are attached to it (see Fig. 11). The surface is updated and refined permanently inside an arbitrarily sized region around the focus point. The extent of the focus area is shown by a transparent sphere.

Besides these objectives the efficient decoding of the data stream from the server and the fast generation of the 3D scene from the received data stream was an important goal. As the request for reconstructing an isosurface usually results in an unknown amount of data to be received from the server memory has to be allocated dynamically in order to store this data. Dynamic memory allocation of large multidimensional arrays is a major performance bottleneck in current Java implementations. For this reason an array of vectors with a fixed length is initially allocated when the program is started. This array is filled with vertex data during the decoding of the data stream. As soon as the array is entirely filled the data is passed to the *VRML* or *OpenGL* visualization component and the array is reused further on.

Very different mechanisms are employed to generate the 3D scene from the received data stream when the *VRML* or the *OpenGL* binding visualization components are used.

## 4.1 VRML Rendering

As in previous work [7] a base scene with empty `IndexedFaceSet` nodes is used for the visualization of the isosurfaces using VRML. The EAI permits to fill `IndexFaceSet` nodes with vertex data by sending an array of vertices to the exposed field `point` of the `Coordinate` sub-node of an `IndexedFaceSet` node:

```
VRML:
Shape {
  appearance DEF m1
  Appearance {
    material DEF mat Material {
      diffuseColor  0.5 0.5 0.5
      specularColor 0.5 0.5 0.5
      }
    }
  geometry DEF tris0000 IndexedFaceSet {
    coord DEF points0000 Coordinate {
      point [ 0.0 0.0 0.0 ]
      }
    coordIndex [ 0 0 0 -1 ] solid FALSE}
  }
```

```
Java code:
Browser browser = Browser.getBrowser(applet);
Node node=browser.getNode("points0000");
EventInMFVec3f event=(EventInMFVec3f)
  node.getEventIn("point");
event.setValue(float[n][3]);
```

First a reference to a `Browser` object is obtained. The `getNode` method of this object is used to get a reference to a *VRML* node which was named using the `DEF` keyword. Then this node reference is used to get a reference to the `eventIn` field "point" which is filled with data using the `setValue` method.

The corresponding indices list is sent to the exposed field `coordIndex` as an event:

```
Java code:
Node node=browser.getNode("tris0000");
EventInMFVec3f event=(EventInMFVec3f)
  node.getEventIn("coordIndex");
event.setValue(int[n]);
```

Using this technique `IndexedFaceSet` nodes can only be filled up with data completely - no data can be added. For this reason the base scene consists of a number of `IndexedFaceSet` nodes, which are filled up from the first to the last one when an isosurface is build up. Remaining nodes, which are not needed are cleared with empty lists of vertices.

A `TouchSensor` node is placed in front of the `IndexedFaceSet` nodes in the scene graph. Whenever the user moves the mouse over the isosurface an event is generated which is sent to the Java applet. The Java applet can obtain the position where the touch event happened, place the focus point and update the isosurface accordingly.

Experiments using an retained mode API like VRML shows that rendering of large amounts of triangles becomes the limiting factor of our application (see Section 5). Each time an `IndexFaceSet` node is filled up with triangle data, the scene is completely rerendered thus slowing down the clients overall performance.

## 4.2 OpenGL Rendering

*OpenGL bindings* enable developers to write 3D applications in Java using the API defined by *OpenGL*. *OpenGL* calls issued in a Java implementation are directly converted into native *OpenGL* library call using the Java Native Interface (JNI). The wrapper library is only a minimal glue layer in between the Java program and the rendering library. *OpenGL* is an immediate mode API that allows full control of the rendering process.

Again we use a preallocated array of fixed length, which is successively filled with the vertex data directly received or created. When the array is entirely filled a *OpenGL* display list is created, which stores the rendering primitives built from the received data stream. It is rendered into the existing frame buffer content. When a display list is executed, the retained data is sent from the display list just as if it was sent by the application in immediate mode.

The placement of the focus point for local refinement of the isosurface is done by exploiting picking events. The `gluUnProject` function is used to locate the position in object space where the pick event was issued. Again the focus point is placed at this point and the isosurface is locally updated.

In contrast to the *VRML* implementation the rendering of newly available triangles can be performed much more efficiently using *OpenGL*-bindings. Having already stored a large number of triangles on the client the advantages become most obvious (see Section 5). These results clearly demonstrate the need for a standardization of a Web-based immediate mode rendering API.

## 5 Results and Analysis

In this section we show results for different models and we analyze some of the main features of the presented approaches. On the server side all tests were run on a SGI Octane MXE equipped with one 250 MHz R10000 processor and 1024 MB main memory. A SGI O2 workstation with 195 MHz R10000 processor and 128 MB main memory was serving as the client system. Both machines were linked via a 100 MB Ethernet connection. The client system was further equipped with the Netscape Communicator 4.07 with CosmoPlayer 2.1 Beta and the JDK1.1.6 with the Magician *OpenGL* Bindings 1.1 [6].

First, we analyze all three approaches for the reconstruction of stripped isosurfaces and we compare them with the standard MC-algorithm with and without compressed data transmission. Table 1 shows detailed results of the proposed techniques. The corresponding images are shown in Figures 9 and 10.

Table 1: Model characteristics and server timings for different isosurface reconstruction scenarios. The data size was $512^2 \cdot 256$.

| | Org | Cmp | Smp | Red | Opt |
|---|---|---|---|---|---|
| Reconstruct | 100% | 97% | 99% | 43% | 37% |
| Strip Length | 1 | - | 1.9 | 7.7 | 13.2 |
| #Vertices (Send) | 100% | - | 69% | 43% | 38% |
| #Bytes . | 100% | 13% | 70% | 46% | 40% |
| #Vertices (Rend) | 100% | 69% | 69% | 43% | 39% |

Method *Org* refers to the standard MC-algorithm. Triangles are transmitted independently. Methods *Cmp* and *Smp* perform the cell-wise reconstruction of stripped surfaces without ever connecting strips from adjacent cells. However, method *Cmp* stops the reconstruction right after the interpolation weights have been computed and sends these weights together with the MC configuration and the cells position to the client. Methods *Red* and *Opt* show the characteristics of stripping with and without a preferred direction.

The time necessary to reconstruct the isosurface on the server only slightly differs for methods *Org*, *Cmp* and *Smp* since the calculation of vertex positions from the already computed weights and also the organization into separate triangles does not contribute much to the overall time. Both methods that directly try to compute stripped surfaces are considerably faster than the others due to the reduced number of vertices to be calculated. Although a certain overhead has to be spent to traverse the extended MC-tables and to check adjacent cells for possibilities to continue the strip, the profit that can be made by avoiding multiple computations of the same vertices clearly dominates the overall performance. This also reflects in the different times for method *Red* and *Opt*, which basically result from the difference in the average strip length.

The number of vertices to be reconstructed strongly determines the transmission load and also the rendering performance on the client. At this point it is interesting to note that method *Opt* is still the most optimal one in terms of the number of reconstructed, transfered and rendered vertices although multiple vertices that are necessary to continue a strip have to be sent and rendered.

Obviously, concerning the transmission load method *Cmp* is ahead of all others since only the interpolation weights have to be transmitted. Later, this advantage will be slightly compensated due to the extra computations that have to be performed on the client to compute the vertex positions. However, as can be seen from the first row of Table 1 this only makes a small fraction of the overall time. We should also point to the small transmission overhead that is introduced by the communication protocol for stripped isosurfaces. For each strip an additional integer value has to be transmitted in which the length of the strip is encoded.

In Figure 10 we show the results of the standard MC-algorithm compared to the level-of-detail reconstruction of the same data set in order to demonstrate the advantages of the proposed multi-resolution representation. The original surface consists of about 6 million vertices. In our current Web-based implementation it took roughly 41 seconds to reconstruct the surface on the server. 72 MB of data had to be transfered from the server to the client. In the middle image we chose a focus point around the belly button. In this way the overall number of vertices was reduced to 157 K, which were reconstructed in 1.3 seconds. By just transferring the interpolation weights we saved approximately 97% of the data to be transmitted. The performance was further improved by selecting two object space clipping planes. The rightmost image in Figure 10 shows the remaining parts of the isosurface generated in 0.3 seconds thus effectively enabling real-time exploration of even the largest scale data sets.

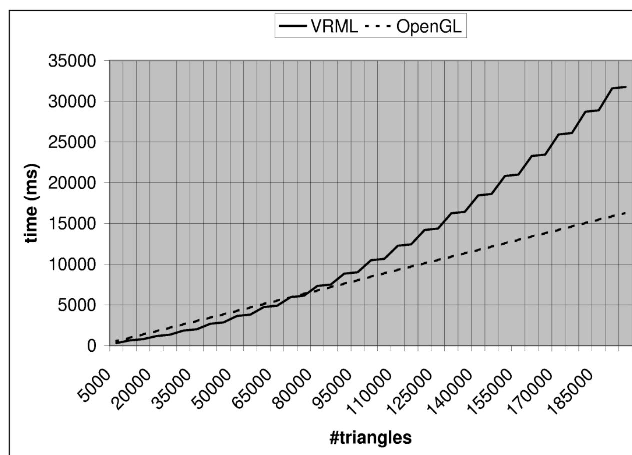



Figure 8: Transfer and display times

Figure 8 shows a detailed performance evaluation of the *OpenGL* and the *VRML* based implementation of the current client system. Upon receipt of 5000 triangles the client module sends these triangles to the rendering module. For low triangle counts the *VRML* implementation shows small advantages compared to the implementation using *OpenGL* bindings. However, the more triangles have to be received and rendered the more the *OpenGL* implementation becomes superior. An almost linear progression is achieved because newly received triangles are rendered directly into the frame buffer without ever processing the already previously ones. By exploiting the *VRML* node concept, on the other hand, the entire scene has to be rendered again each time the scene graph changes. This becomes the limiting factor when a large number of triangles is already stored in the node. The step-like shape of the performance curve can be explained by the VRML redraw mechanism. The scheduling of redraw events is exclusively controlled by the *VRML* plugin. Thus it might happen that although triangles are received and linked to the *VRML* node the scene graph is not rendered immediately, which results in a temporal delay and consequently in the step-like appearance of the curve.

Although the *VRML* implementation has some disadvantages for large amounts of triangles it is usable for our focus point oriented

isosurface extraction. The amount of triangles transmitted when moving the focus point is very small.

## 6 Conclusion

We have presented several different isosurface extraction techniques for distributed Web-based scenarios. The major contribution here is to reduce the number of geometric primitives to be reconstructed in order to minimize computational load and network traffic. Remote interactive exploration of large scale volume datasets is achieved in this way.

We showed that the stages of a pipelined Marching Cubes like reconstruction process can be relocated onto clients and servers in different ways. Several techniques that are especially well suited for each of those scenarios were introduced.

In the near future we plan to further improve the proposed techniques by omitting redundant computations in a more rigid way. A combination of the developed stripping technique and the transmission of Marching Cubes configurations with interpolation weights will considerably improve the network load thus resulting in shorter *mapping-render* cycles.

With direct access to the frame buffer contents becoming available through OpenGL bindings the ideas introduced in [11] for view-dependent isosurface reconstruction become a relevance also in distributed environments. Bi-directional protocols have to be established in order to further reduce the amount of primitives to be reconstructed and transmitted by only extracting the visible portion of the isosurface.

A Web-based visualization service is planned, which will allow the remote interactive exploration of very large datasets using standard Web-browsers and advanced OpenGL functionality. With volumetric textures becoming available on the PC market, integrated display of isosurfaces and soft tissue by means of 3D texture mapping will be provided even on low-end platforms.

## References


[1] ISO/IEC 14772-1:1997. The virtual reality modeling language. http://www.vrml.org/Specifications/VRML97/, 1997.

[2] C.L. Bajaj, V. Pascucci, and D.R. Schikore. Fast isocontouring for improved interactivity. In *ACM Symposium on Volume Visualization*, pages 39–46, 1996.

[3] P. Cignoni, P. Marino, C. Montani, E. Puppo, and R. Scopigno. Speeding up isosurface extraction using interval trees. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):158–170, 1997.

[4] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *1996 Symposium on Volume Visualization*, pages 31–39, 1996.

[5] M. Deering. Geometry compression. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, pages 13–20, 1995.

[6] Alligator Descartes. Magician Programmer's Guide for Java. http://www.arcana.co.uk/products/magician, 1998.

[7] K. Engel, R. Grosso, and T. Ertl. Progressive isosurfaces on the web. In *Late Breaking Hot Topics Proc. Visualization 98*, 1998.

[8] K. Engel, O. Sommer, C. Ernst, and T. Ertl. Remote 3d visualization using image-streaming techniques. In *ISIMADE - 11 TH Internationl Conference on Systems Research, Informatics and Cybernetics*, 1999.

[9] F. Evans, S. Skiens, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization 1996*, pages 319–326, 1996.

[10] Hugues Hoppe. Progressive meshes. In *Computer Graphics (SIGGRAPH 96 Proceedings)*, pages 99–108, 1996.

[11] Y. Livnat and C. Hansen. View dependent isosurface extraction. In *IEE Visualization 1998*, pages 175–181, 1998.

[12] Y. Livnat, H.-W. Shen, and C.R. Johnson. A near optimal isosurface extraction algorithm using span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[13] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[14] C. Marrin. Proposal for a vrml 2.0 information annex. http://cosmosoftware.com/developer/moving-worlds/spec/ExternalInterface.html, 1997.

[15] C. Michaels and M. Bailey. Vizwiz: a java applet for interactive 3d scientific visualization on the web. In *Proceedings IEEE Visualization '97*, pages 261–267, 1997.

[16] G. Nielson and B. Hamann. The asymptodic decider: removing the ambiguity in marching cubes. In *Visualization '91*, pages 83–91, 1991.

[17] K.M. Oh and K.H. Park. A type-merging algorithm for extracting an isosurface from volumeetric data. *The Visual Computer*, 12:406–419, 1996.

[18] M. Ohlberger and M. Rumpf. Hierarchical and adaptive visualization on nested grids. *Computing*, 59 (4):269–285, 1997.

[19] T. Poston, H.T. Nguyen, P.A. Heng, and T.T. Wong. Skeleton-climbing:fast isosurfaces with fewer triangles. In *Pacific Graphics 1997*, pages 117–126, 1997.

[20] W. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Proceedings SIGGRAPH '92*, pages 65–70, 1992.

[21] R. Shekhar, E. Fayyad, R. Yagel, and J. Cornhill. Octree-based decimation of marching cubes surfaces. In *Visualization '96*, pages 335–342, 1996.

[22] H. Shen and C. Johnson. Sweeping Simplices: A Fast Iso-Surface Axtraction Algorithm for Unstructured Grids. In *IEEE Visualization '95*, pages 143–150, 1995.

[23] H.-W. Shen, C. Hansen, Y. Livnat, and C. R. Johnson. Isosurfacing in Span Space with Utmost Efficiency (ISSUE). In *Proceedings IEEE Visualization '96*, pages 287–294, 1996.

[24] J. Trapp and H-G. Pagendarm. A prototype for a WWW-based visualization service. In *Proceedings Eurographics '97*, pages 23–30, 1997.

[25] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer; Preprint available at: http://www9.informatik.uni-erlangen.de/Persons/Westermann/lod.pdf.gz*, 1999.

[26] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. In *ACM Transactions on Graphics*, pages 201–227, 1992.

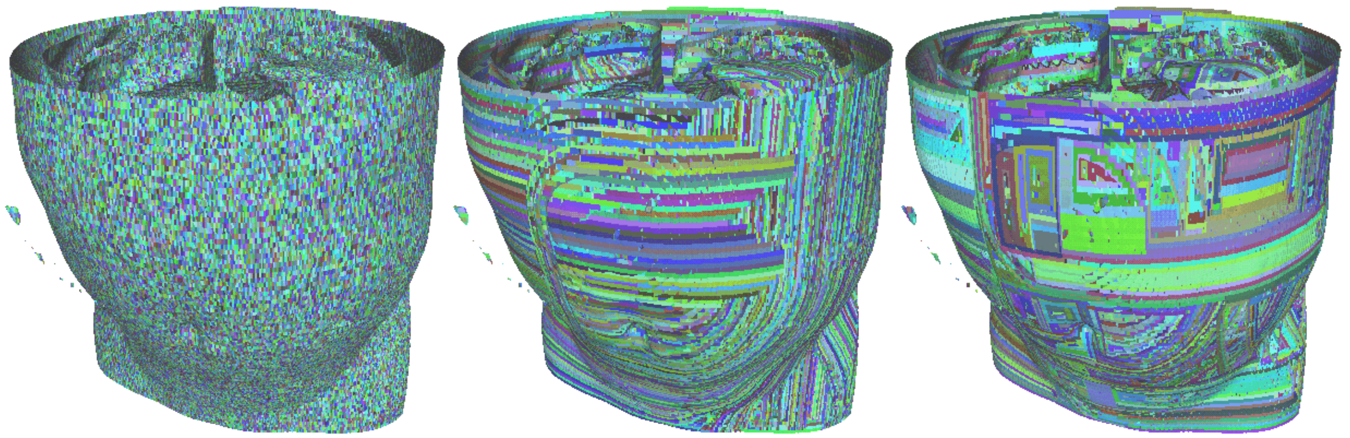


Figure 9: Results of different approaches to directly reconstruct triangle strips (colored consistently) from scalar volume data. Left: cell-wise stripping of the surface. Middle: strips are connected across cell boundaries into one particular direction. Right: strips are connected across boundaries into different alternative directions. The average strip length was 1.9, 7.7 and 13.4, from left to right.

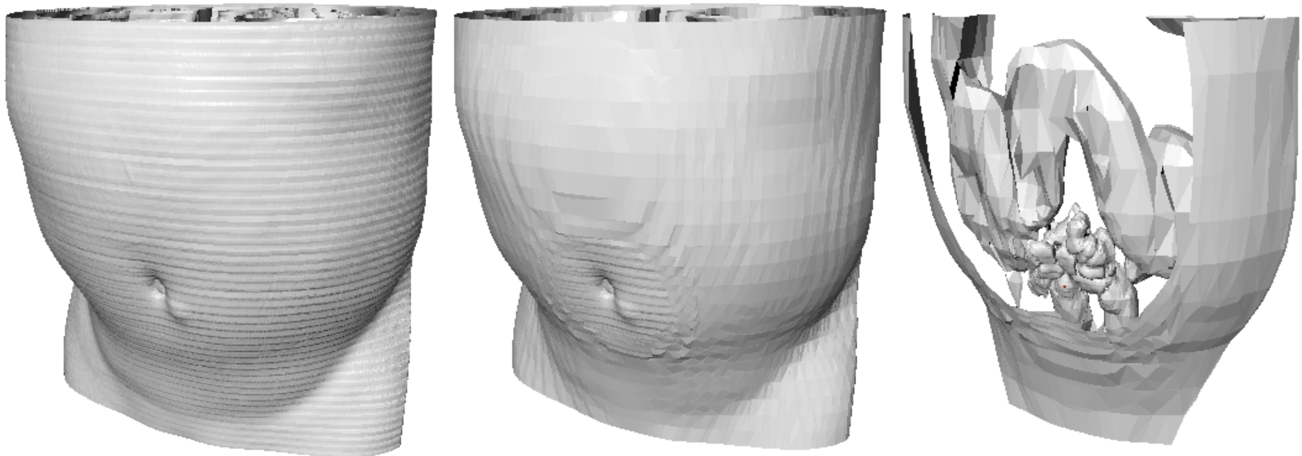


Figure 10: Hierarchical level-of-detail reconstruction: Left: isosurface as reconstructed from the original data samples (2.3 Mtriangles). Middle: LOD representation with the focus area around the belly button (135 Ktriangles). Right: use of two additional object space clipping planes.

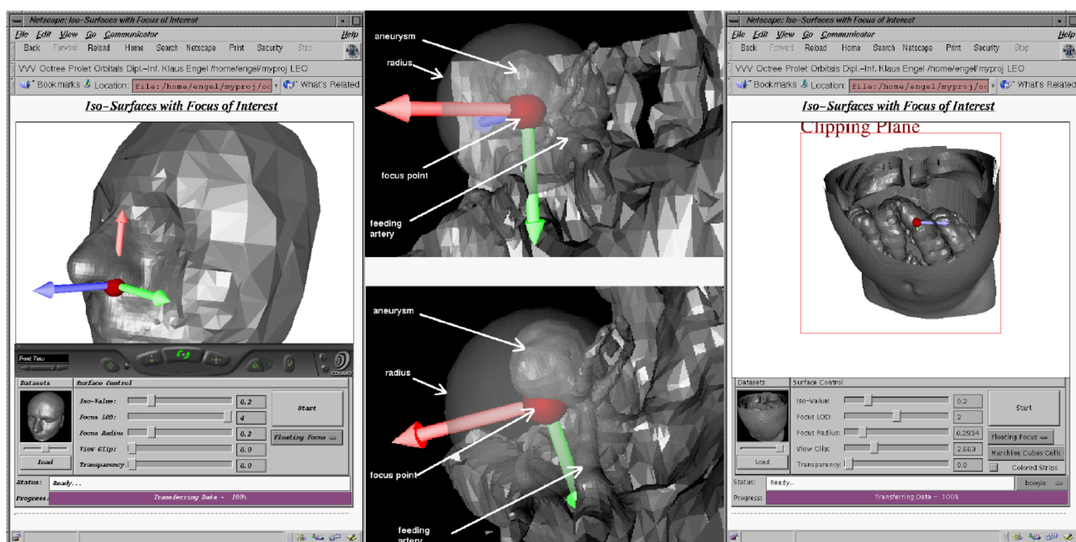



Figure 11: The client application: Left: VRML visualization with focus point. Middle: moving the focus point (radius shown by transparent sphere) on the cerebral artery to an aneurysm (berry-like blister of the cerebral artery): Right: VRML visualization with focus point and clipping plane.