Transfer functions for Volume Rendering applications and implementation results with the VTK

Martin Gasser

Institute of Computer Graphics and Algorithms
Vienna University of Technology
Vienna / Austria

Abstract

In this paper an overview over some methods of transfer functions design for volume rendering applications will be given. First, TF design in general and multidimensional transfer functions in special will be presented. After this rather theoretic introduction a short overview over existing solutions for TF specification will be given and two rather complementary approaches for mastering the problem of how to find an appropriate transfer function will be introduced, one beeing targeted at providing a user friendly interface for specifying a multidimensional transfer function in an interactive volume rendering application, the other providing a quality measure for isovalues in a volume dataset.

1 Introduction

In the field of scientific visualization, techniques for visualizing volume datasets (e.g. medical data which is acquired from computer tomography) are used heavily. One such technique is Volume Raycasting. Probably the main problem in applying this method is the specification of a proper transfer function (the mapping from scalar data values to visual properties like opacity or color).

1.1 Volume Raycasting Basics

As mentioned before, one method widely used for volume rendering is raycasting. Basically, raycasting means, that a ray is sent from the viewer location through each pixel. The pixel color is then obtained by integrating the light emission and absorption contributions along this ray ("compositing").

Max [7] modelled the complex physics associated with this quite simple idea with a differential equation. He showed that the solution to this equation can be approximated by the following recursion:

$$Out_i = In_i(1 - \alpha_i) + C_i\alpha_i \tag{1}$$

The above equation applies for back-to-front compositing. In the following are the equations for front-to-back compositing:

$$Col = Col + (1 - \alpha_{acc})C_i\alpha_i \tag{2}$$

for the accumulated color and

$$\alpha_{acc} = \alpha_{acc} + (1 - \alpha_{acc})\alpha_i \tag{3}$$

Solving the two equations and rearranging them yields the same result, whereas front-to-back compositing can increase rendering performance by employing early ray termination (i.e. the compositing procedure can be stopped when the accumulated opacity is already 1 or sufficently close to 1).

The process of volume rendering can be visualized in a pipeline When dealing with transfer function specification we are talking about the *classification* part of the rendering pipeline.

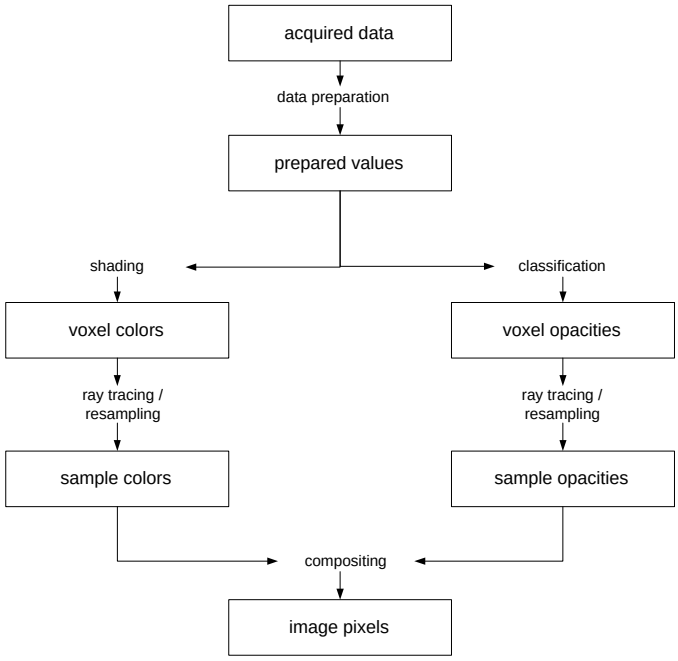

Figure 1: Volume Rendering pipeline

1.2 Classification Procedure

When using the above mentioned equations (1, 2), a color and an opacity value at each voxel location instead of one scalar value (e.g. density value from a computer tomograph) are required. That's where transfer functions come into play. So, a transfer function provides a mapping from scalar data values to color and opacity.

2 Flaws of conventional transfer functions and how to find remedy

The simplest form of a transfer function is one that provides a one-to-one-mapping from a scalar value to a visual property (e.g. density to color). Transfer functions traditionally have been limited to a one dimensional domain, whereas the nature of the data to be visualized often is inherently multi-dimensional. For example, when visualizing a medical dataset created from CT scans, it is often very difficult to identify boundaries between multiple materials based only on its density values. As soon as one data value is associated with multiple boundaries, a one dimensional transfer function cannot render them in isolation. See Fig. 4 for an example of the benefits of 2D transfer functions over one dimensional ones. This is a computer tomography of a turbine blade, which has a simple two material (air, metal) composition. Air cavities within the blade have slightly higher values than the air outside, leading to smaller edge strength for the internal air-metal boundaries. Whereas 1D transfer functions cannot selectively render the internal structures, 2D transfer functions can avoid opacity assignment at regions of high gradient magnitude, which leads to finer detail in these regions. Fig. 5 shows an example where features cannot be extracted using a 2D transfer function, whereas it is possible to reveal them with a 3D transfer function, incorporating the second derivative along the gradient direction as the third dimension into the transfer function domain.

2.1 Related work

2D transfer function where introduced to volume rendering by Levoy [5] . He incorporated gradient magnitude at the desired voxel location as the second dimension, using a method of finite differences for the generation of gradients. He introduced two styles of transfer function, both multidimensional, and both using the gradient magnitude as the second dimension. One function was intended for the display of surfaces between materials (*region boundary surfaces*) and the other for the display of isosurface values (*isovalue contour surfaces*). This semi-automatic method aims at isolating those portions of the volume that form the material boundaries. Figure 2 shows a plot of an isovalue contour transfer function, whereas Figure 3 shows a region boundary transfer function Unfortunately, the complexities in designing transfer functions explode with every extra dimension. In general, even for one dimensional transfer functions, finding the appropriate values is accomplished by trial and error. So, it's apparent, that there is a strong need for effective user interfaces and/or algorithms which aid the user in finding optimal configurations when exploring volume data.

Kindlmann et al. [3] propose an approach where the data itself should suggest an appropriate transfer function that brings out the features of interest. They try to utilize the relationship between the three quantities *data value*, *first derivative* and *second derivative* (along gradient direction), which they capture in a datastructure called *histogram volume*. They describe the theoretical relevance of these quantities and how transfer functions can be constructed from them. Finally they develop an algorithm for automatic transfer function generation based on analysis of the histogram volume.

Kniss et al. [4] propose a method for the specification of 3D transfer functions, where the user can interactively select features

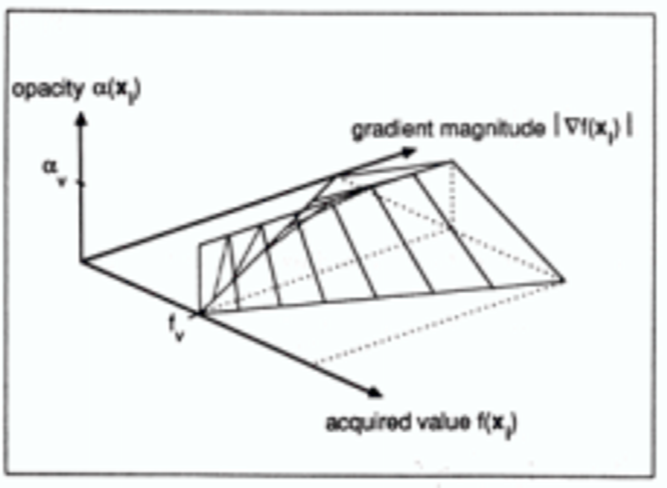

Figure 2: isovalue contour transfer function

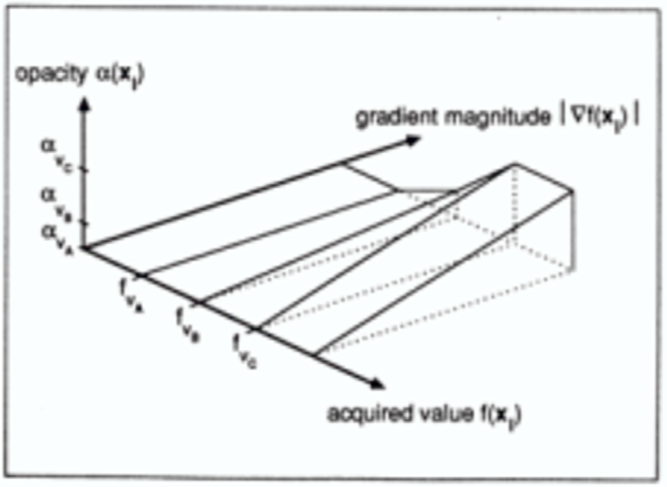

Figure 3: region boundary transfer function

of interest in the spatial domain through specialized *widgets* (See section 3.1). They pursue a rather explorative approach in finding interesting features in volumetric data.

The *Contour Spectrum* [1] presents the user a set of signature functions, such as surface area, volume and the gradient integral over the isovalue range. These data characteristics are drawn as overlapping 1D plots over the set of isovalues as an aid to the user when selecting a specific isocontouring level (e.g. it is likely that a "good" isovalue has a high gradient integral over the corresponding isosurface).

Pekar et al. [8] propose a similar approach, their method finds isovalues by utilizing laplacian weighted histograms and a theorem originating from the field of electrodynamics (see 3.2).

3 Methods for Transfer Function Specification

The task of designing transfer functions is heavily dependent on user knowledge, both about the domain the data originates from, and also concerning technical issues in transfer functions design.

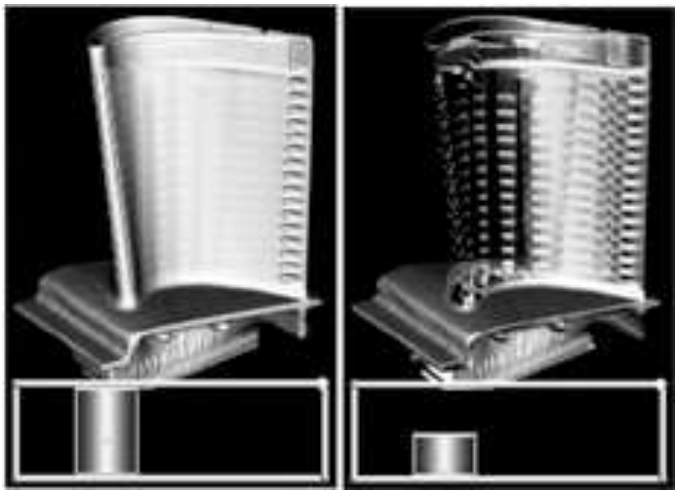

Figure 4: 1D (left) and 2D (right) transfer function

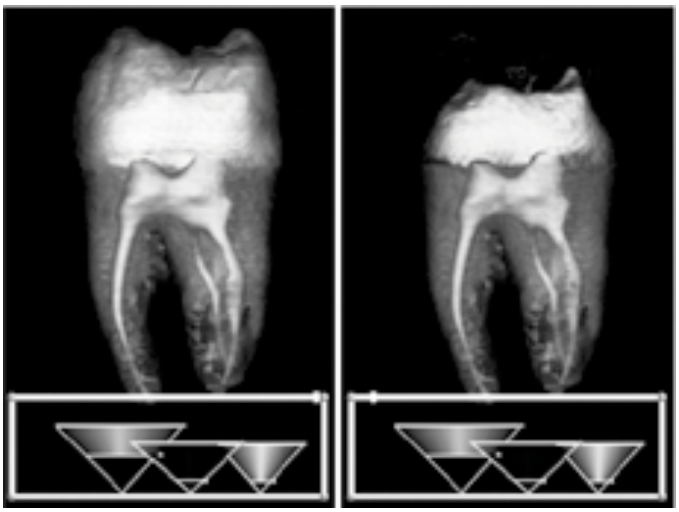

Figure 5: 2D (left) and 3D (right) transfer function

The traditional trial-and-error-based approach to transfer function design is very time consuming, so robust methods are needed to explore the data set of interest without any special knowledge of the interesting domains.

Whereas it is always a good idea to aid the user in finding appropriate parameters for transfer functions, it is not desirable to exclude the user completely from the process of choosing parameters. Therefore appropriate user interfaces are needed, where the user can select areas of interest (e.g. isovalues in a volume dataset) and which give an immediate visual feedback about the results of his selection.

3.1 Specialized User Interfaces

As we already have seen, using multidimensional transfer functions may be very complicated. To give the user a more natural way to specify transfer function parameters, intuitive user interfaces are needed. Kniss et al. [4] propose a method which is based on three major paradigms:

1. Multidimensional Transfer Functions
2. Linked UI Widgets/Dual Domain Interaction
3. Interactivity

They propagate the use of 3D transfer functions, where the scalar value, the first and the second derivative span up the space of the transfer functions, which is mapped to the RGBA space of color/opacity values. Because this space is inherently nonspatial and the quantities at each point in this room are not scalar, a principle of the widgets is to link interaction in one domain with feedback in another. On the one hand, the user can explore the data volume in the spatial domain by the use of a *data probe widget* or a *clipping plane widget* to select a particular region of interest in the volume dataset, seeing the scalar value and the derivatives at this point and its local neighbourhood. So, in opposition to classical volume rendering systems, where the proces of transfer function specification involves the setup of a sequence of linear ramps, which define the mapping of data values to opacity and color, this system operates the opposite way round. The transfer function is set by direct interaction in the spatial domain, with observation of the transfer function domain. To provide interactive framerates for their visualization system, they heavily incorporated modern hardware in their method. By using an OpenGL extension (i.e. *Pixel Textures*), an extra speedup to the traditional hardware volume rendering methods has been gained.

Another image-centric approach proposed by J. Marks et al. [6], namely *Design Galleries*, works by offering a set of automatic generated thumbnail images to the user, who can then select the most appealing ones out of them. The method works by rating a set of randomly generated input vectors (the components of the vectors are the transfer function parameters). Input vectors that generate similar images are grouped in a simple interface, for easy browsing of the transfer function space.

3.2 Automatic Detection of Isosurfaces

Pekar et al. [8] propose a data driven approach to semi automatic detection of isosurfaces. They use a laplacian weighted gray value histogram and the divergence theorem to develop an efficient method for the computation of optimal gray value thresholds. Besides their method also offers some other useful features, such as computation of isosurface volume, isosurface area and mean gradient.

In the following a short description of their algorithm: Let the gray value at position x be $I(x)$. Each gray value threshold

$T_i \in [I_{min}, I_{max}]$ yields a binary volume $\mathcal{B}(T_i) = \mathcal{B}_i$, where a voxel has a binary value $\mathcal{B}_i(x) = 1$ for $I(x) > T_i$ and 0 otherwise, yielding a set of contour surfaces Γ_i which divide the data volume into regions of $B_i(x) = 0$ and regions of $B_i(x) = 1$. The aim is to find a certain gray value threshold T_{opt} that optimally transfers the voxels of a given volumetric data set into a binary volume. The threshold value is considered to be appropriate if it corresponds to a material transition in the data volume. To detect intensity transitions between different material types, we have to look for contour surfaces consisting of voxels with high gray value gradients. These contour surfaces typically correspond to highly contrasted boundaries in the volume data. The objective function to measure the goodness of a certain partitioning $\mathcal{B}(T)$ can be defined as the surface integral of the intensity gradient magnitude $g = \nabla I$ over the set of surfaces Γ:

$$F(T) = \int_{\Gamma} |g| d\gamma \tag{4}$$

This integral can be computed for each threshold T_i by finding the partitioning surfaces and computing their gradient vectors at all points. A threshold value can be considered as appropriate if the objective function $F(T)$ which we call the *total gradient integral* takes a maximum at this value.

A very efficient approach to compute the objective function is based on the divergence theorem [2]. It states that an integral of a vector field g over a contour surface can be replaced by the volume integral of the divergence $\nabla \dot{g}$ over the volume Ω enclosed by the surface.

By taking into account that the gradient is orthogonal to an isosurface and by using the divergence theorem we can rewrite the objective function as:

$$F(T) = \int_{\Gamma} |g| d\gamma = -\int_{\Gamma} g \cdot n d\gamma = -\int_{\Omega} \nabla \cdot g dx \tag{5}$$

where the divergence of the gradient vector field $\nabla \cdot g$ is equal to the Laplace operator applied to the data volume, which leads us to

$$F(T) = -\int_{\Omega} \mathcal{L}(x) dx \tag{6}$$

A discrete approximation of the above integral yields

$$F(T) = -\sum_{I(x) \geq T} \mathcal{L}(x) \tag{7}$$

where $I(x) \geq T$ identifies all voxels in a volume dataset having a gray value above the threshold T.

By leveraging a cumulative laplacian weighted histogram, F(T) can be computed very efficiently, requiring only one pass through the dataset.

The corresponding algorithm can be described as follows:

1. Compute the laplace value $\mathcal{L}(x)$ at each voxel location x.

2. Build a histogram from the filtered dataset, incrementing a histogram bin $I(x)$ (where $I(x)$ is the image intensity at voxel location x) by the Laplace value $\mathcal{L}(x)$ at voxel location x:

$$G(I(x)) \Leftarrow G(I(x)) + \mathcal{L}(x) \tag{8}$$

3. Accumulate the histogram in such a way, that each cumulative histogram value $G_c(T)$ is set as the sum of all higher histogram values $G(I)$ with $I \geq T$. This corresponds to the evaluation of (7) for a certain gray value threshold. Each cumulative histogram value $G_c(T)$ gives a discrete approximation of the volume integral (6) over voxels with a gray value $I \geq T$.

4. The optimal threshold T_{opt} which corresponds to the most dominant material transition can be found as the global maximum of the histogram curve $L_c(T)$

See section 4 for an implementation of this method.

4 Implementation

The language chosen for the implementation of the algorithm explained in [8] was Java. Furthermore, we used the *Visualization Toolkit* [9] for the visualization of the isovalues generated by the algorithm. Due to its simplicity and the tight integration of the VTK, Java is a powerful alternative to C++, particularly when doing rapid application development of small and simple applications.

Basically the application *Volumez* is a simple raycaster. Transfer Functions are one dimensional, mapping a scalar data value to an opacity and a color. To aid the user in finding the optimal density thresholds for isosurfaces, an accumulated laplace weighted histogram is drawn in the transfer functions specification window (See Figure 6 - 9).

Listing 1: LaplacianWeightedHistogram.java

```
package volumez;
/*
 * LaplacianWeightedHistogram.java
 *
 * Created on 02. Juni 2002, 12:39
 */

import vtk.*;

public class LaplacianWeightedHistogram {
    private int numbins;
    private int[] data_dimensions;

    private double[] normalizedBins;
    private double[] bins;
    private double highestAccumLaplace;

    int optimalValue;

    private vtkImageData imageData;

     /*
      * Creates a laplacian weighted histogram from the
          data in the vtkImageData object
      */
    public LaplacianWeightedHistogram(vtkImageData
        _image_data) {

        imageData = _image_data;

        float lowerbound = (float) imageData.
            GetScalarRange()[0];
        float upperbound = (float) imageData.
            GetScalarRange()[1];
        data_dimensions = imageData.GetDimensions();

        numbins = (int) (upperbound - lowerbound) + 1;
        bins = new double[numbins];

        for (int i = 0; i < bins.length; i++) {
          bins[i] = 0.0f;
        }

        for (int k=0; k < data_dimensions[2]; k++)
            for (int j=0; j < data_dimensions[1]; j++)
                for (int i=0; i < data_dimensions[0]; i
                    ++) {
                    int value = (int) Math.round(
                        getDataValue(i,j,k));
```

```
                    bins[value] += getLaplaceValue(i, j,
                        k);
                }

            // accumulate the laplace values (integrate)
            highestAccumLaplace = 0.0f;
            for (int i = numbins - 2; i >= 0; i--) {
              bins[i] += bins[i+1];

              if (bins[i] > highestAccumLaplace) {
                highestAccumLaplace = bins[i];
                optimalValue = i;
              }
            }
        }

        private double getDataValue(int i, int j, int k) {

          vtkPointData point_data = this.imageData.
              GetPointData();
          vtkDataArray scalars = point_data.GetScalars();

          // if we are outside the volume boundaries, return
              zero
          if (i < 0 || j < 0 || k < 0
          || i >= data_dimensions[0] || j >= data_dimensions
              [1] || k >= data_dimensions[2])
              return 0.0f;

          int[] ijk = new int[3];

          ijk[0] = i; ijk[1] = j; ijk[2] = k;
          int pointId = imageData.ComputePointId(ijk);
          return scalars.GetTuple1(pointId);
        }

        private double getLaplaceValue(int i, int j, int k) {
          return
            ((getDataValue(i, j , k) - getDataValue(i+2, j, k
                )) - (getDataValue(i-2, j , k) -
                getDataValue(i, j, k))) +
            ((getDataValue(i, j , k) - getDataValue(i, j+2, k
                )) - (getDataValue(i, j-2 , k) -
                getDataValue(i, j, k))) +
            ((getDataValue(i, j , k) - getDataValue(i, j, k
                +2)) - (getDataValue(i, j , k-2) -
                getDataValue(i, j, k)));
        }

        /** Returns the histogram as is.
         */
        public double[] getHistogram() {
            return bins;
        }

        /** Returns the histogram with every value divided by
              the largest value
         */
        public double[] getNormalizedHistogram() {
            if (normalizedBins == null)
                normalizedBins = new double[numbins];

            for (int i = 0; i < numbins; i++)
                normalizedBins[i] = (double) bins[i] /
                    highestAccumLaplace;

            return normalizedBins;
        }

        /** Returns the value where the histogram value is
             maximal.
         */
        public float getOptimalThreshold() {
            return optimalValue;
        }
}
```

Listing 1 shows the Sourcecode of the class responsible for the calculation of the Laplacian weighted histogram. Instead of directly calculating the laplace value with a 3D-laplace kernel, duplicate calculation of central differences was chosen to approximate the second derivative at a voxel location (the Laplace value at this location). This method leads to better results an is much more efficient than convoluting the neighbouring voxels with a generalized filter kernel.

Figure 6: Laplacian weighted histogram of the feet dataset

5 Results

See Figures 7 - 9 for some sample pictures generated with the proposed algorithm.

6 Conclusions

While the main attention of research has been turned on developing efficient and high quality methods for volume rendering, there is still a lack of results of research in the field of user interface design for volume rendering applications. Although the method presented in [4] provides a good explorative approach to transfer function specification, it is still rather difficult to master the design of a proper transfer function, and it requires a lot of routine when a certain region of interest should be revealed. Therefore, data driven methods need to be incorporated into volume rendering applications for the purpose of TF design. In [8], one such method is presented. The implementation of this method showed, that the usability of volume rendering applications can be greatly improved by providing means of exploring data quickly and finding regions of interest faster than it would be possible with a naive trial-and-error approach.

References

[1] Chandrajit L. Bajaj, Valerio Pascucci, and Daniel Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.

[2] J.D.Jackson. *Classical Electrodynamics*. Wiley, 1962.

[3] Gordon Kindlmann and James W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium on Volume Visualization*, pages 79–86, 1998.

[4] Joe Kniss, Gordon Kindlmann, and Charles Hansen. Interactive volume rendering using multi-dimensional transfer functions

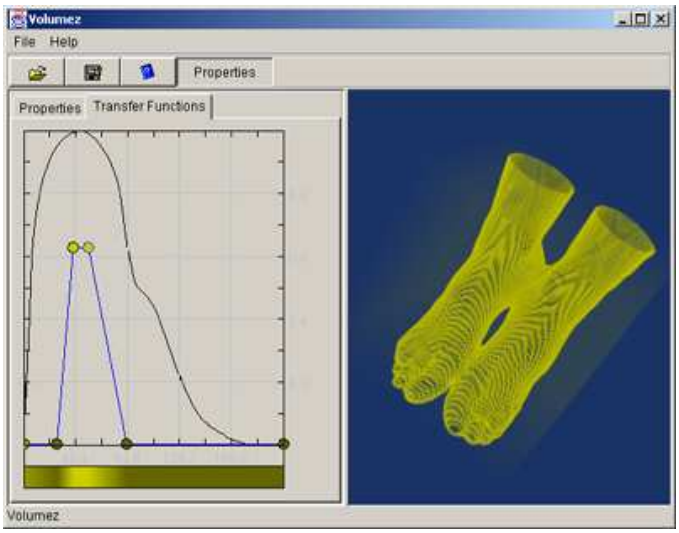

Figure 7: A scan of a foot, with the skin emphasized in the transfer function

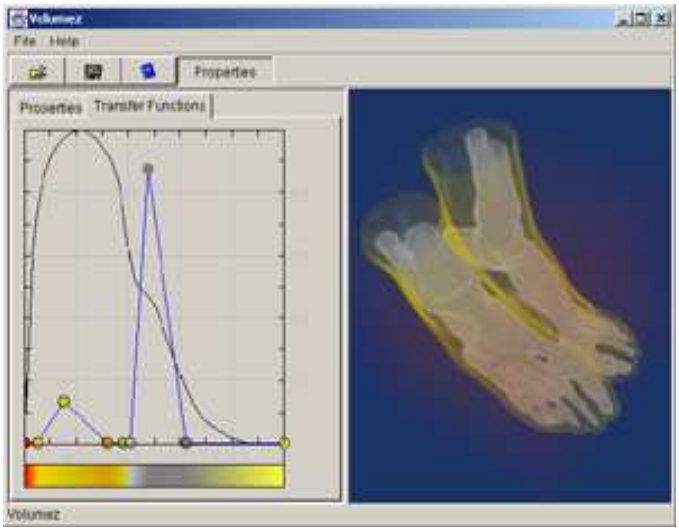

Figure 8: Skin and bone of the feet dataset

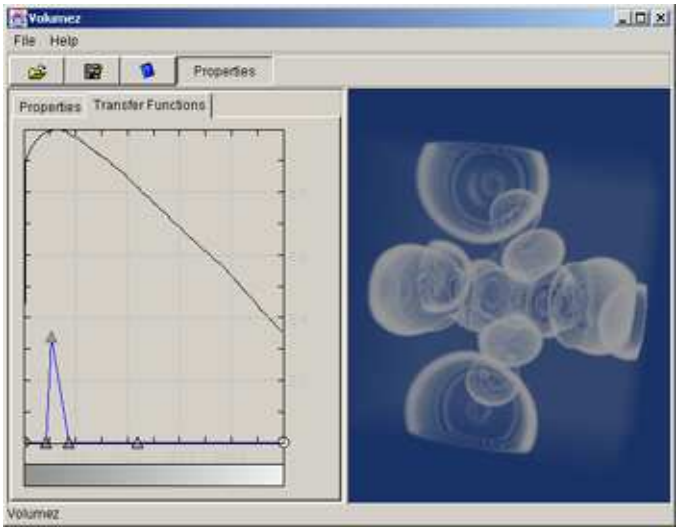

Figure 9: Iron molecule

and direct manipulation widgets. In *IEEE Visualization 2001*, October 2001.

[5] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, May 1988.

[6] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. *Computer Graphics*, 31(Annual Conference Series):389–400, 1997.

[7] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):97–108, June 1995.

[8] Vladimir Pekar, Rafael Wiemker, and Daniel Hempel. Fast detection of meaningful isosurfaces for volume data visualization. In *IEEE Visualization 2001*, October 2001.

[9] W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit: An Object-Oriented approach to 3D Graphics*. Prentice Hall, 1998.