

Medical Volume Rendering Techniques

Wenhui Zhang^{1*}, James K. Hahn¹

Abstract

Medical visualization is the use of computers to create 3D images from medical imaging data sets, almost all surgery and cancer treatment in the developed world relies on it. Volume visualization techniques includes iso-surface visualization, mesh visualization and point cloud visualization techniques, these techniques have revolutionized medicine. Much of modern medicine relies on the 3D imaging that is possible with magnetic resonance imaging (MRI) scanners, functional magnetic resonance imaging (fMRI)scanners, positron emission tomography (PET) scanners, ultrasound imaging (US) scanners, X-Ray scanners, bio-marker microscopy imaging scanners and computed tomography (CT) scanners, which make 3D images out of 2D slices. The primary goal of this report is the application-oriented optimization of existing volume rendering methods providing interactive frame-rates. Techniques are presented for traditional alpha-blending rendering, surface-shaded display, maximum intensity projection (MIP), and fast previewing with fully interactive parameter control. Different preprocessing strategies are proposed for interactive iso-surface rendering and fast previewing, such as the well-known marching cube algorithm.

Keywords

Volume rendering , Iso-surface visualization , Alpha-blending rendering, Marching cube algorithm

¹Department of Computer Science, The George Washington University, Washington D.C., United States of America

*Corresponding author: wenhui@gwmail.gwu.edu

Contents

Introduction		3.2 Fast object-order techniques 23
1 Thirty Year Overview of Medical Volume Visualization		Hierarchical splatting • Extraction of surface points
2		
2 Volume-rendering Methods	4	3.3 Hybrid acceleration methods 24
2.1 Indirect volume rendering 4		Shear-warp factorization
Space domain volume rendering: Marching Cubes • Frequency domain volume rendering:Fourier Transform		
2.2 Direct volume rendering 12		4 Challenges of Medical Volume Rendering 25
Image order methods • Object order methods		Acknowledgments 26
3 Acceleration Techniques	20	References 26
3.1 Fast image order techniques 20		
Hierarchical data structures • Early ray termination • Distance transformation		

Introduction

Medical imaging currently plays a crucial role throughout the entire clinical applications from medical scientific research to diagnostics and treatment planning. However, medical imaging procedures are often computationally demanding due to the large 3D medical datasets to process in practical clinical

applications.

In this report, we gave a compact 30-year overview of the history of medical visualization research. Based on the history we categorized the whole survey report into direct volume rendering and indirect volume rendering techniques. Detailed view about volume rendering in general is also described. Software only acceleration methods are presented as well.

Research challenges of the coming decade are identified and discussed in the last section.

1. Thirty Year Overview of Medical Volume Visualization

Since the offspring of magnetic resonance imaging (MRI) and computed tomography (CT) scanners around the early 70s, and the consequent tons of medical volume data, medical visualization has undergone significant development and is now a primary branch of visualization.

Medical visualization includes the use of various techniques to either invasively or noninvasively image the structure/anatomy and function/pharmacology of human being bodies. Structural medical visualization explains physical properties of the recording system, while functional medical visualization reveals physiological constraints of the recording system. Regular clinical methods for creating images and visualization of human being structures include Computed Tomography (CT), Structural Magnetic Resonance Imaging (MRI), Endoscopy, Ultrasound Image (US) while regular clinical methods for creating images of human being function includes Electroencephalography(EEG), Magnetoencephalography(MEG), Positron Emission Tomography (PET) and Functional Magnetic Resonance Imaging (fMRI).

Medical Visualization finds its applications in diagnostic purposes and treatment purposes, therefore medical visualization has an important role in the improvement of public health in all population groups. For example virtual colonoscopy, in treatment, for example surgical planning and guidance, and in medical research, for example visualization of diffusion tensor imaging data [5] . As importance of

medical visualization is revealed in both the acute and postacute rehabilitation settings for patients, like brain visualization with ruptured aneurysm brain tumor, lesions and other common 560 brain diseases like Alzheimer's, Parkinson's, Altruism, and Anxiety Disorder, 3D medical visualization interactive tools needed to be built up to meet the satisfaction of current therapeutic and diagnostic clinical standards.

Furthermore, medical visualization is justified also to follow the course of a disease already diagnosed and treated. During the past decades, there was rapid development in medical image acquisition technology, which make it now possible to acquire much more complex data than human beings can achieve ever before. Take High Angular Resolution Diffusion Imaging (HARDI) for example, 40 or more diffusion weighted volumes are acquired, thus made it possible to calculate and visualize water diffusion and, indirectly, structural neural connections in the brain [7] . In fMRI based full brain connectivity, time based correlation of neural activity is indirectly measured between all pairs of voxels in the brain, thus giving insight into the functional neural network [1] . Moreover, advanced medical visualization techniques enforce us on answering more and more complex answers.

In 1978, Sunguroff and Greenberg published their work on the visualization of 3D surfaces from CT data for diagnosis, as well as a visual radiotherapy planning system, also based on CT data [6].

In 1983, Vannier et al. published their results on developing a system for the computer based pre-operative planning of craniofacial surgery [8] . The system was based on the extraction and visualization of 3D hard and soft tissue surfaces from CT data. Through the integration of an industrial CAD (computer aided design) application, it was also possible to perform detailed 3D measurements on the extracted surfaces.

In 1986, Hohne and Bernstein published a paper on using the gray-level gradient to perform shading of surfaces rendered from 3D CT data [3].

In 1987, Lorensen and Cline published the now famous Marching Cubes isosurface extraction al-

gorithm [33], which enabled the fast and practical extraction of 3D isosurfaces from real world medical data.

In 1988, Levoy published a paper on volume raycasting [4].

In 1988, the first multimodal volume rendering paper was published by Hohne [2], in which registration and combined visualization of CT and MRI was introduced and nicely presented.

One of the first real applications of medical visualization is known as therapy planning, which remains important to this day. In 1993, Altobelli published their work on using CT data to visualize the possible outcome of complicated craniofacial surgery [9].

Basser published a paper in 1994, introducing Diffusion Tensor Imaging (DTI), an MRI based acquisition modality, which yields 3×3 symmetric diffusion tensors as its native measurement quantity [11].

In 1995, Hong introduced virtual colonoscopy (VC) [16], after which, medical visualization is serving as an more and more important medical application, namely screening for colon cancer.

Time varying medical volume data visualization was brought on the table in 1996 by Behrens, to support the examination of dynamic contrast enhanced MRI mammography data with the display of parameter maps, the selection of regions of interest (ROIs), the calculation of time intensity curves (TICs), and the quantitative analysis of these curves [12].

In 1998, Basser and his colleagues published one paper on extracting data from fibertract trajectories from DTI data, they were known as the first to extract and visualize fibertract trajectories from DTI data of the brain at that time [10]. After two years, the visualization community includes tensor lines for tractography [27] and direct volume rendering of DTI data [18] [19] spring out with tons of innovative methods.

In 2000 Ebert introduced the term illustration in this work [15]. Illustrative visualization is primarily motivated by the attempt to create renditions that consider perceptual capabilities of humans. Boundary enhancement based on gradient approximation

[14] and curvature-based transfer functions [20] are landmarks in illustrative medical visualization. Tietjen et al. applied silhouettes and other feature lines for various scenarios in liver surgery planning [24]. Besides silhouettes technique, hatching has great potential to reveal details of shapes [17].

In 2001, Tory presented methods for visualizing multi time pointed MRI data of a multiple sclerosis patient, where the goal was to study the evolution of brain white matter lesions over time, which sets the milestone for multi subjects medical visualization [25].

For a long time, it was not possible to apply illustrative visualization techniques in practice due to performance constraints. And in 2003, GPU raycasting was introduced by Kruger [21], with advances in graphics hardware and algorithms, it is now feasible from a computational standpoint.

The upcoming technology of DTI initiated a whole body of medical visualization research dedicated to the question of how best to visually represent and interact with diffusion tensor data in particular and multifield medical data in general. In 2007, Blaas presented a paper on a visual analysis inspired solution to this problem based on linked physical and feature space views [13].

Medical visualization has also started to work on the problem to work with multi subjects data. These datasets include measurements and imaging for more than one subject at a time. The aim of this paper is to be able to extract and analysis certain patterns that affect subgroups of the whole collection. LifeLines2, an early information visualization system to visualize and compare multiple patient histories or electronic medical records [26]. Work has been done on the interactive visualization of the multi subject and mixed modality datasets acquired by medical cohort studies [130]. In these studies, mixed modality data, including imaging, genetics, blood measurements, is acquired from a group of subjects in order to be analyzed for diagnosing or predicting the clinical outcome of that group. It was demonstrated by Stenwijk to create a highly interactive coupled view visualization interface, integrating both information and scientific visualization techniques, with which patterns, and

also hypotheses, could be extracted from the whole data collection.

Area of medical visualization is very complex and, depending on a context, requires supplementary activities of medical doctors, medical physicists, biomedical engineers as well as technicians.

2. Volume-rendering Methods

Volume rendering is a technique for visualizing sampled functions of 3D data by computing 2D projections of a colored semitransparent volume. It involves the following steps: the forming of an RGB-Alpha volume from the data, reconstruction of a continuous function from this discrete data set, and projecting it onto the 2D viewing plane (the output based on screen space) from the desired point of view.

The raw datasets we got for medical purpose include, cloud point data, data by slides(nii file in neuroscience field, e.g. MRI), surface data, to show these dataset in volumetric way needs some special techniques and transfer functions to transfer them into RGB-Alpha dataset modality.

An RGB-Alpha volume is a 3D four-vector data set, where the first three components are the familiar R, G, and B color components and the last component, Alpha, represents opacity. An opacity value of 0 means totally transparent and a value of 1 means totally opaque. Behind the RGB-Alpha volume an opaque background is placed. The mapping of the data to opacity values acts as a classification of the data one is interested in. Isosurfaces can be shown by mapping the corresponding data values to almost opaque values and the rest to transparent values. The appearance of surfaces can be improved by using shading techniques to form the RGB mapping. However, opacity can be used to see the interior of the data volume too. These interiors appear as clouds with varying density and color. A big advantage of volume rendering is that this interior information is not thrown away, so that it enables one to look at the 3D data set as a whole. Disadvantages are the difficult interpretation of the cloudy interiors and the long time, compared to surface rendering, needed to perform volume rendering.

There are four main paradigms in which volume rendering is performed in nowadays: raycasting [76] [66], splatting [91], shear warp [?], cell projection [82] [119], texture mapping hardware assisted [120] [121] [122], and via custom hardware [123] [124].

In this report, we are only interested in software based volume rendering technics, and volume rendering techniques are clustered into two categorites, indirect volume rendering and direct volume rendering. Indirect volume rendering, where in a preprocessing step the volume is converted to an intermediate representation which can be handled by the graphics engine. In contrast, the direct methods process the volume without generating any intermediate representation assigning optical properties directly to the voxels.

2.1 Indirect volume rendering

Indirect volume rendering technique extracts polygonal surface from volume data and represents an isosurface, it is also known as as 3D contours. The most popular algorithm for indirect volume rendering is marching cube algorithm [33].

Indirect methods aim at the visualization of iso-surfaces defined by a certain density threshold. The primary goal is to create a triangular mesh which fits to the isoregions inside the volume. This can be done using the traditional image processing techniques, where first of all an edge detection is performed on the slices and afterwards the contours are connected. Having the contours determined the corresponding contour points in the neighboring slices are connected by triangles. This approach requires the setting of many heuristic parameters thus it is not flexible enough to use them in practical applications. A more robust approach is the “marching cubes” isosurface reconstruction [33], which marches through all the cubic cells and generates an elementary triangular mesh whenever a cell is found which is intersected by an iso-surface. Since the volumetric data defined in the discrete space is converted to a continuous geometrical model, the conventional computer graphics techniques, like ray tracing or buffering can be used to render the iso-surfaces.

Another indirect volume-rendering approach is known as 3D Fourier transform (3D FT), where the intermediate representation is a 3D Fourier transform of the volume rather than a geometrical model [38] [39] [40]. This technique aims at fast density integral calculation along the viewing rays. Since the final image is considered to be an X-ray simulation, this technique is useful in medical imaging applications. The main idea is to calculate the 3D Fourier transform of the volume in a preprocessing step. This transformation is rather expensive computationally but it has to be executed only once independently on the viewing direction. The final image is calculated performing a relatively cheap 2D inverse Fourier transformation on a slice in the frequency domain. This slice is perpendicular to the current viewing direction and passes through the origin of the coordinate system. According to the Fourier projection-slice theorem the pixels of the generated image represent the density integrals along the corresponding viewing rays.

2.1.1 Space domain volume rendering: Marching Cubes

Isosurface is an operation that given a scene outputs a connected surface as a binary shell. Connectedness means that within the output shell it is possible to reach to any shell element from any shell element without leaving the shell. If the input is a binary scene, the shell constitutes a connected interface between 1-cells and 0-cells. If the input is a grey scene, the interface between the interior and exterior of the structure is usually difficult to determine. Thresholding can be used to determine this interface, in which the shell constitutes essentially a connected interface between cells that satisfy the threshold criterion and cells that do not. In a particular thresholding operation specified by a single intensity value, the resulting surface is called an iso-surface. The common iso-surfacing algorithms are Opaque Cubes (Cuberille) [28] [29], Marching Cubes, Marching Tetrahedra [31], and Dividing Cubes [30]. Of which, the most popular one used for medical visualization today is Marching Cubes.

Marching Cubes algorithm, developed by Lorensen and Cline in 1987 [33] is used to approximate an

isosurface by subdividing a region of space into 3D array of rectangular cells, which is the most popular method for isosurface rendering. Another popular isosurface extraction method is a propagation-based marching cubes method in 1986 by Wyvill et al.[32]. That method is somewhat similar to Marching cubes, yet they have some shortcomings. The isosurfaces they extract also differ. Due to the differences, and since most teams who have described application of a marching cube, in the report we restrict the Marching cubes designation to the Lorensen's approach.

The basic idea of Marching Cubes is that voxel could be defined by the pixel values at the eight corners of the cube. If one or more pixels of a cube have values less than the user-specified isovalue, and one or more have values greater than this value, we know the voxel must contribute some component of the isosurface. By determining which edges of the cube are intersected by the isosurface, we can create triangular patches which divide the cube between regions within the isosurface and regions outside. By connecting the patches from all cubes on the isosurface boundary, we get a surface representation.

In the eighties the volume-rendering research was mainly oriented to the development of indirect methods. At that time no rendering technique was available which could visualize the volumetric data directly without performing any preprocessing. The existing computer graphics methods, like ray tracing or z-buffering [34] had been developed for geometrical models rather than for volume data sets. Therefore, the idea of converting the volume defined in a discrete space into a geometrical representation seemed to be quite obvious. The early surface reconstruction methods were based on the traditional image-processing techniques [37] [35] [36] , like edge detection and contour connection. Because of the heuristic parameters to be set these methods were not flexible enough for practical applications, they are lacking detail and introducing artifacts. Lorensen and Cline [33] came up with the idea of creating polygonal representation of constant density surfaces from 3D array of data. Existing methods of 3D surface generation by Wyvill et

al.[32] trace contours within each slice then connect with triangles (topography map), create surfaces from “cuberilles” (voxels), perform ray casting to find the 3D surface using hue-lightness to shade surface and gradient to shade, and then display density volumes. There are some shortcomings of Wyvill et al’s techniques. One thing is that they throw away useful information in the original data, in cuberilles level they use thresholding to represent surface, during the process of ray casting, they use depth shading alone or approximates shading using unnormalized gradient. Another thing is that these methods lack hidden surface removal, and volume models display all values and rely on motion to produce a 3D sensation. Thus Marching Cubes algorithm is introduced, for Marching Cubes algorithm uses all information from source data, derives inter-slice connectivity, surface location, and surface gradient, also result of Marching Cubes can be displayed on conventional graphics display systems using standard rendering algorithms, and also does not rely on image processing performed on the slices and requires only one parameter which is a density threshold defining the isosurface.

In summary, marching cubes creates a surface from a three-dimensional set of data as follows [33]:

1. Read four slices into memory;
2. Scan two slices and create a cube from four neighbors on one slice and four neighbors on the next slice;
3. Calculate an index for the cube by comparing the eight density values at the cube vertices with the surface constant;
4. Using the index, look up the list of edges from a precalculated table;
5. Using the densities at each edge vertex, find the surface and edge intersection via linear interpolation;
6. Calculate a unit normal at each cube vertex using central differences. Interpolate the normal to each triangle vertex;
7. Output the triangle vertices and vertex normals.

After having an isosurface defined by a density threshold, all the voxels are investigated whether they are below or above the surface, comparing the densities with the surface constant. To locate the surface, it uses a logical cube created from eight pixels, 4 each from 2 adjacent layers, slice k and slice $k + 1$. This binary classification assigns the value of one to the voxels of densities higher than the threshold and the value of zero to the other voxels, which sets cube vertex to value of 1 if the data value at that vertex exceeds or equals the value of the surface we are constructing otherwise, sets cube vertex to 0. If a vertex = 1 then it is “inside” the surface, if a vertex = 0 then it is “outside”. Any cube with vertices of both types is “intersected” by the surface. The algorithm marches through all the intersected cells, where there are at least two corner voxels classified differently. For such cells an index to a look-up table is calculated according to the classification of the corner voxels as shown in Figure 1 .

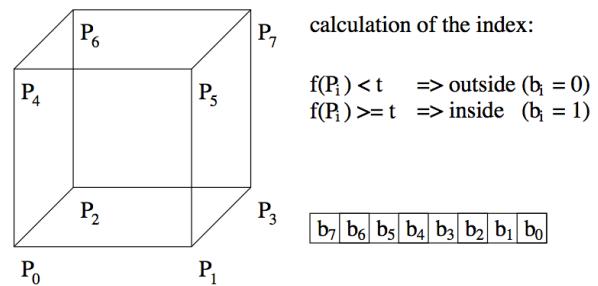


Figure 1. Calculation of the index to the look-up table

For each cube, we have 8 vertices with 2 possible states each, inside or outside. This gives us 2^8 possible patterns, which is 256 cases. An index system is built, which contains eight bits associated with the eight corner voxels of the cubic cell and their values depend on the classification of the corresponding voxels. This index addresses a look up table containing all the 256 cases of elementary triangular meshes. Because of symmetry reasons, there are just 15 topologically distinct cases among

these patterns thus in practice the look-up table contains 15 entries instead of 256. Figure 2 shows the triangulation of the 15 patterns.

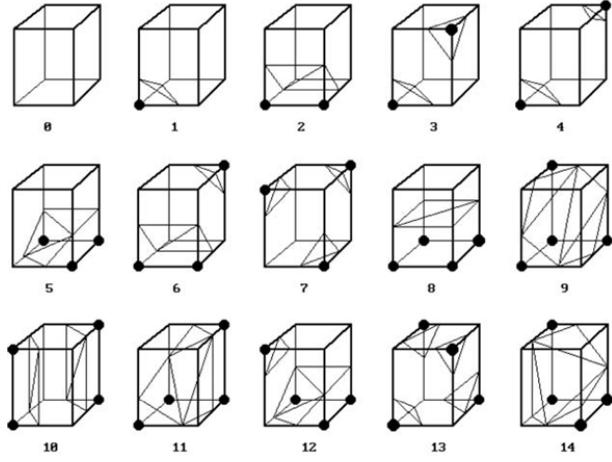


Figure 2. Marching Cubes

After having the list of intersected edges read from the look up table, the intersection points along these edges are calculated using linear interpolation. Vertex bit mask is used to create an index for each case based on the state of the vertexes, and then index will tell which edge the surface intersects, we can then can linearly interpolate the surface intersection along the edge. The position of vertex $V_{i,j}$ along the edge connecting corner points P_i and P_j is computed as (1):

$$V_{i,j} = ((t - f(P_i) * P_j)) + \frac{(f(P_j) - t) * P_i}{f(P_j) - P_i} \quad (1)$$

assuming that $f(P_i) < t$ and $f(P_j) > t$, where f is the spatial density function and t is the threshold defining the isosurface.

Since the algorithm generates an oriented surface with a normal vector at each vertex position, the last step is the calculation of the surface normals.

To calculate surface normal, we need to determine gradient vector, G , which is the derivative of the density function. To estimate the gradient vector at the surface of interest, we first estimate the gradient vectors at the vertices and interpolate the gradient at the intersection. The gradient at cube

$V_{i,j,k}$, is estimated using central differences along the three coordinate axes by:

$$G_{i,j,k}(x) = \frac{f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k)}{\delta x} \quad (2)$$

$$G_{i,j,k}(y) = \frac{f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k)}{\delta y} \quad (3)$$

$$G_{i,j,k}(z) = \frac{f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1})}{\delta z} \quad (4)$$

The normals $n(x_i, y_j, z_k)$ at the cube vertices are determined using central differences (5):

$$n(x_i, y_j, z_k) \approx \frac{1}{2} * \begin{bmatrix} f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k) \\ f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k) \\ f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1}) \end{bmatrix} \quad (5)$$

After dividing the gradient by its length produces the unit normal at the vertex required for rendering. Then the algorithm linearly interpolates this normal to the point of intersection. At an intersection point $V_{i,j}$ along the edge connecting grid points P_i and P_j the surface normal $N_{i,j}$ is calculated using linear interpolation between the corresponding normals denoted by N_i and N_j respectively (6):

$$N_{i,j} = ((t - f(P_i) * N_j)) + \frac{(f(P_j) - t) * N_i}{f(P_j) - P_i} \quad (6)$$

The continuous geometrical model generated by the marching cubes algorithm can be rendered using the traditional computer graphics techniques. The conventional graphics acceleration devices which are based on the hardware implementation of the -buffering hidden surface removal can render such a model in real time using Phong shading or Gouraud shading. The pseudocode of generalized Marching Cubes algorithm is shown as following:

Pseudocode: Generalized Marching Cubes [33]

```

1 Create an edge table
2 Read in 3 2d slices of data
3 while(moreDataSlices)
4 {
5   Read in next slice of data
6   while(moreCubes)
7   {
8     Fill cube index
9     Assign 1 or 0 to vertex index
10    edgesToDrawBetween = edgeTable[cubeIndex]
11    Interpolate:
12      triangle vertexes from edge vertexes
13    Determine triangle vertex normals
14    Draw triangle(s)
15    currentCube++;
16  }
17  Discard data slice
18 }

```

Marching cubes algorithm has been applied in many application areas, including biochemistry [41], biomedicine [42], deformable modeling [43], digital sculpting [44], environmental science [45], mechanics and dynamics [46], natural phenomena rendering [47], visualization algorithm analysis [48], etc. Processing involving depth maps [49] has also been influenced by Marching cubes isosurface extraction, especially in the development of methods based on distance fields [50].

An preliminary implementation of marching cubes is shown as below Figure 3 .



Figure 3. Marching Cubes of Metaballs

Although employed in many arenas, isosurface creation is heavily utilized in medical visualization [51] and computer-aided diagnosis applications [52]. Isosurfaces recreate the digitized images taken by computed tomography (CT), magnetic resonance (MR), and single-photon emission computed tomography(SPECT).

Pseudocode: Marching Cube for Medical Image Dataset [53]

```

1 for each image voxel

```

```

2   a cube of length 1 is placed on
3   eight adjacent voxels of the image
4   for each of the cube's edge{
5     if (the one of the node voxels has value
6       greater than or equal to t and
7       the other voxel has value less than t)
8     then
9       (calculate the position of a point on
10      the cube's edge that belongs to the
11      isosurface,
12      using linear interpolation)
13    }
14    for each predefined cube configurations{
15      for each of eight possible rotations{
16        for the configuration's complement{
17          compare the produced cube configuration
18          of the above calculated isopoints to the
19          set of predefined cube configurations
20          and produce the corresponding triangles
21        }
22      }
23    }
24 }

```

Marching Cubes algorithm is capable only for isosurface rendering thus the internal structures of the volume cannot be visualized. After the pre-processing, the original data values are not available anymore thus cutting planes are not supported. Cutting operations are rather important in medical imaging applications, where the physician can define an arbitrary cross section of the 3D model and render the slice displaying the original gray scale data values. Furthermore, the modeling of semi-transparent tissues, which is the most important feature of direct volume rendering, is not supported either.

The main disadvantage of the marching cubes algorithm is the computationally expensive pre-processing. Especially having high resolution data sets the number of the generated triangles can be enormous. Since the interactivity is strongly influenced by the complexity of the model usually some post-processing is performed on the initial mesh in order to simplify it [54]. Furthermore the triangular mesh is not uniform because the vertices are located on the edges of cubic cells, therefore some mesh refinement is also required.

Advantages of Marching Cubes:

1. Uses all information from source data;
2. Derives inter-slice connectivity, surface location, and surface gradient;
3. Result can be displayed on conventional graphics display systems using standard rendering algorithms;
4. Allows Solid modeling capability: cutting and capping.

Disadvantages of Marching Cubes:

1. Requires user input;
2. There is a loss of accuracy when visualizing small or fuzzy details;
3. The assumptions which are made about the data may not necessarily be valid. This particularly applies to the assumption that the surfaces exist within the data to map the geometric primitives onto;
4. Unless the original information is stored along with the geometric representation, the information on the interior of the surfaces is lost;
5. Mainly limited to medical images with clear contiguous intensity boundaries: constant density;
6. Is performing a modified form of thresholding.

2.1.2 Frequency domain volume rendering: Fourier Transform

Fourier Volume Rendering (FVR) developed by Levoy, Totsuka and Malzbender [40] is based on the frequency spectrum of the 3D scalar field by utilizing the Fourier Slice Projection theorem [56] [55]. This theorem allows us to compute integrals over volumes by extracting slices from the frequency domain representation. It states that a projection of a 3D data volume from a certain view direction can be obtained by extracting a 2D slice perpendicular to that view direction out of the 3D Fourier spectrum and then inverse Fourier transforming it [57], as is shown in Figure 4 :

Malzbender proposes various filters for high-quality resampling in frequency domain. Totsuka and Levoy [40] extended this work with depth cues and shading performing calculations in the frequency domain during slice extraction.

Illumination models for FVR were studied in the work of [58]. They describe methods to integrate diffuse lighting into FVR. One approach is based on gamma corrected hemispherical shading and is suitable for interactive rendering of fixed light sources. Another technique uses spherical harmonic functions and allows lighting using varying light sources. These shading techniques, however, require a large amount of memory and are

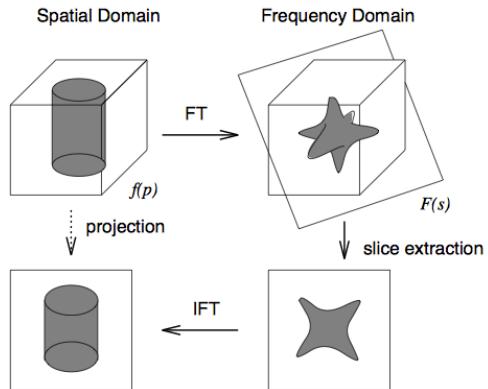


Figure 4. Frequency domain rendering

not well suited for visualization of large data sets. Another approach that produces images which are similar to FVR is based on importance sampling and Monte Carlo integration [59] thus the samples are not aligned on a regular grid. This technique overcomes the limitation of parallel projection and the overall computational complexity $O(N^2)$ is better than in case of FVR.

Fourier Slice Projection theorem, also known as Fourier slice theorem or projection slice theorem in mathematics, is a theorem states that the results of the following two calculations are equal. The first calculation procedure is stated as, take a three dimensional function $f(r)$ project it onto a two dimensional plane, and do a Fourier transform of that projection. The second calculation procedure is stated as take that same function, but do a three dimensional Fourier transform first, and then slice it through its origin, which is parallel to the projection line.

For a 3D volume, the theorem states that the following two are a Fourier transform pair, the 2D image obtained by taking line integrals of the volume along rays perpendicular to the image plane, and the 2D spectrum obtained by extracting a slice from the Fourier transform of the volume along a plane that includes the origin and is parallel to the image plane.

The inverse process of Fourier Volume Rendering can be seen as a reconstruction method. Here, a set of pre acquired projections are Fourier transformed and then put slice by slice into a Fourier

Volume, which is initialized with zeros. If enough projections are available the Fourier Volume will be filled completely after a while. Consequently, by applying the inverse Fourier transform to the reconstructed Fourier Volume the spatial representation of the object, described by the projections, can be computed.

The general pipeline of the Fourier Volume Rendering technique can be divided into two basic steps. At first a computationally expensive one-time preprocessing step is performed. In a second rendering step arbitrary view directions can then quickly be computed by carrying out a two dimensional slicing operation and an inverse frequency transform.

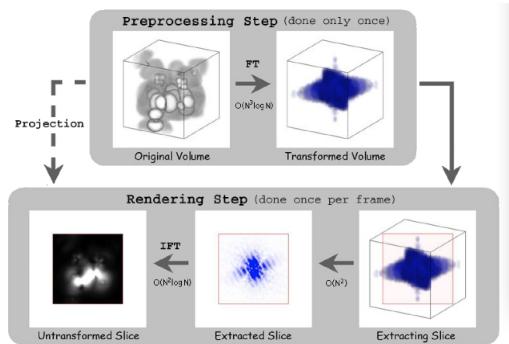


Figure 5. Fourier Volume Rendering pipeline

Figure 5 gives a more detailed overview of the involved operations. Here, the preprocessing step transforms the spatial domain volume into its frequency domain representation. This is usually accomplished with either the three dimensional Fast Fourier Transform or the Fast Hartley Transform and has a run time complexity of $O(N^3 \log N)$ assuming an $N \times N \times N$ input data set. Due to the Fourier Projection Slice Theorem a view from any arbitrary angle can then easily be computed in a second rendering step by slicing the frequency volume along a plane oriented perpendicular to the viewing direction and crossing exactly through the center of the frequency volume. On account of the two dimensional character of the resultant slice, the overall complexity of this operation is $O(N^2)$. Finally the sought-after projection is derived by taking an inverse two-dimensional Fast Fourier or Fast Hartley Transform of the frequency slice. The com-

plexity of this operation is $O(N^2 \log N)$, which is the asymptotic running time of the rendering step.

In summary, FVR algorithm works as following:

1. Load the volume data into memory and compute its 3D Fourier transform using the FFT algorithm;
2. When the view direction changes, take a 2D slice of the volume, passing through its center and perpendicular to the view direction. Many operations can be performed, at this stage, on the 2D slice to get various effects such as thresholding, depth cue, etc, through the use of filters;
3. Take the 2D inverse Fourier transform of this slice. Rescale its values to valid intensity range and display the slice.

A straightforward implementation of the Fourier transform is not suitable for high performance FVR. The inverse two dimensional transform must be computed at high speed to achieve interactive frame rates. Therefore fast variants of the Fourier Transform are used in FVR implementations. The original idea of the Fast Fourier Transform (FFT) was introduced by Cooley [60]. Their algorithm decomposes the Discrete Fourier Transform (DFT) into $\log(2N)$ passes, where N is the size of the input array. Each of these passes consists of $N/2$ butterfly computations. Each butterfly operation takes two complex numbers a and b and computes two numbers, $a + wb$ and $a - wb$, where w is a complex number, called principal N th root of unity. After $\log(2N)$ passes the butterfly operations result into the transformed data. One of the fastest implementations available, is the FFTW library [61].

The Fast Hartley Transform (FHT) [62] performs as an alternative to FFT. The transform produces real output for a real input, and is its own inverse. Therefore for FVR the FHT is more efficient in terms of memory consumption. The Multidimensional Hartley Transform, however, is not separable. The N dimensional transform cannot be computed as a product of N one dimensional transforms. Bracewell and Hao propose a solution to this problem [62] [63]. They suggest to perform N

one dimensional transformations in each orthogonal direction followed by an additional pass that corrects the result to correspond to the N dimensional Hartley transform.

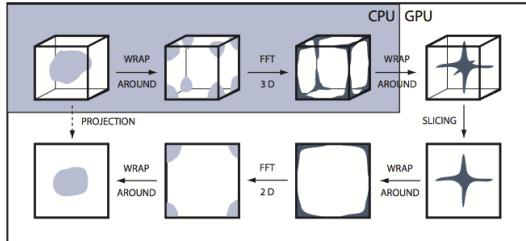


Figure 6. Frequency domain volume rendering pipeline on GPU

Based on the work of Bracewell and Hartley, a GPU based FVR algorithm [64], and which is accelerated by factor of 17 by mapping the rendering stage to the GPU. The three dimensional transformation into frequency domain is done in a preprocessing step. The rendering step is computed completely on the GPU. First the projection slice is extracted. Four different interpolation schemes are used for resampling the slice from the data represented by a 3D texture. The extracted slice is transformed back into the spatial domain using the inverse Fast Fourier or Fast Hartley Transform. The rendering stage is implemented through shader programs running on programmable graphics hardware achieving highly interactive framerates, and the pipeline is as shown in Figure 6.

Pseudocode: Fourier Volume Rendering on GPU [64]

```

1  /* Initializing necessary variables */
2  InitVars();
3  /* Initialize CUDA context (GL context has been created already) */
4  InitCUDAContext(argc, argv, true);
5
6  /* Reading & initializing volume data */
7  InitData();
8
9  /* Creating float volume & releasing byte data */
10 CreateFloatData();
11
12 /* CUDA or Hybrid pipeline */
13 if (!GPU)
14 {
15     /* Wrapping around spatial volume */
16     WrapAroundVolume();
17     /* Creating spectrum complex arrays */
18     CreateSpectrum();
19     /* Wrapping around spectral volume */
20     WrapAroundSpectrum();
21     /* Packing spectrum complex data into texture
22     * array to be sent to OpenGL */
23     PackingSpectrumTexture();
24 }
25 else
26 {
  
```

```

27     /* Spectral texture for OpenGL compatibility */
28     mTextureArray =
29         (float*) malloc (mVolumeSize * 2 * sizeof(float));
30     /* Run the FVR on the CUDA pipeline */
31     CUDA_Way();
32 }
33
34 /* Uploading spectrum texture to GPU for slicing */
35 SendspectrumTextureToGPU();
36 /* We don't need float data anymore as it resides in the
37 * GPU texture memory */
38 delete [] mVolumeDataFloat;
39
40 /* Intersecting QUAD with the texture */
41 SetDisplayList();
42
43 /* CUDA timer */
44 cutCreateTimer(&mTimer);
45 cutResetTimer(mTimer);
46
47 /* Register OpenGL callbacks */
48 glutDisplayFunc(DisplayGL);
49 glutKeyboardFunc(KeyBoardGL);
50 glutReshapeFunc(ReshapeGL);
51 glutIdleFunc(IdleGL);
52
53 /* Initializing OpenGL buffers */
54 InitOpenGLBuffers();
55
56 /* Start main rendering loop */
57 glutMainLoop();
58
59 /* Clean Up */
60 CleanUp(EXIT_FAILURE);
61
62 /* Exiting ... */
63 // shrEXIT(argc, (const char**)argv);
  
```

Advantages of Fourier Volume Rendering:

1. Big improvement in speed;
2. Permits the rendering of compressed datasets by using only portions of the frequency spectrum. Fourier Volume Rendering also allows the application of lowpass, highpass, and bandpass filters with little overhead, since the volume data are already available in an adequate representation. By exploiting this property other operations such as successive refinement can easily be achieved just by successively adding higher and higher frequencies to the resultant image. This also permits the rendering of compressed datasets by using only portions of the frequency spectrum, which might be useful for web applications.

Disadvantages of Fourier Volume Rendering:

1. Lack of occlusion and hidden surfaces, the projection obtained by the Fourier projection slice theorem is a line integral normal to the direction of view. Voxels on a viewing ray contribute equally to the image regardless of their distance from the eye. The image therefore lacks occlusion, an important visual cue.

While some users prefer integral projections since nothing is hidden from view, this characteristic would be considered a drawback in most applications. Since for the calculation of the density integrals a distance-dependent weighting function or an opacity manipulation cannot be used. In the next section it will be shown that direct volume-rendering methods are much more flexible in this sense supporting the modeling of several optical phenomena like emission, reflection, and attenuation.

2. Exclusive support of orthogonal viewing;
3. Contrast of high interpolation costs versus ghosts;
4. Significantly higher memory costs.

All problems are technical in nature and several solutions are proposed, yet the lack of occlusion is fundamental and so far no projection slice theorem is known that reproduces the integral differential equation approximated by volume rendering algorithms.

2.2 Direct volume rendering

Direct volume rendering is a method which renders the data set directly without using any intermediate representation. The optical attributes like a color, an opacity, or an emission are assigned directly to the voxels. The pixel colors depend on the optical properties of the voxels intersected by the corresponding viewing rays.

In comparison to the indirect methods presented in the previous section, direct methods display the voxel data by solving the equation of radiative transfer for the entire volumetric object. In direct volume rendering, the scalar value given at a sample point is virtually mapped to physical quantities that describe the emission and absorption of light at that point. This mapping is also often termed classification. It is usually performed by means of a transfer function that maps data values to color (emission) and opacity (absorption). These quantities are then used for a physically based synthesis of virtual images.

Similar to a divide and conquer-strategy, algorithms for direct volume rendering differ in the way the complex problem of image generation is split up into several subtasks. A common classification scheme differentiates between image order and object order algorithms. The direct volume rendering pipeline is shown in Figure 7 :

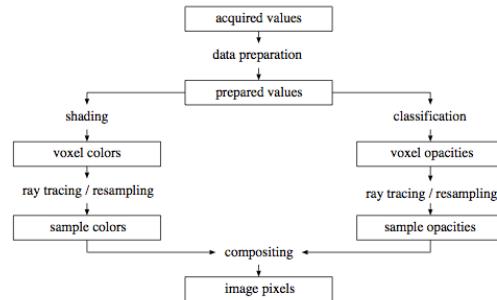


Figure 7. The Direct Volume Rendering Pipeline

The first step in direct volume rendering is data preparation, before any render pipeline operation is performed on the data, it may need some sort of preparation first. Filtering, anti aliasing, contrast enhancement, domain switching are some common operations that are used. The prepared array is the input of the shading process where colors are assigned to the voxels depending on their densities. The assigned colors are shaded. The shading model requires a normal vector at each voxel location. In gray level volumes the normals can be obtained as the estimated gradients calculated from the central differences as shown below,

$$\nabla f(x_i, y_j, z_k) \approx \frac{1}{2} \begin{bmatrix} f(x_{i+1}, y_j, z_k) - f(x_{i-1}, y_j, z_k) \\ f(x_i, y_{j+1}, z_k) - f(x_i, y_{j-1}, z_k) \\ f(x_i, y_j, z_{k+1}) - f(x_i, y_j, z_{k-1}) \end{bmatrix} \quad (7)$$

where $f(x_i, y_j, z_k)$ is the discrete 3D density function. The output of the shading process is an array of voxel colors. In a separate step, classification is performed yielding an additional array of voxel opacities. After having the color and opacity values assigned to each voxel, rays are cast from the view point (image order) or object point (object order) and perform the process of resampling the

volume. The treated samples are rendered to screen in the final step. A number of optical and illumination models can be used depending on how realistic the final image has to be or how computationally complex the operation is allowed to be.

The simplest visualization models directly map the density profile onto pixel intensities. For instance, one possibility is to calculate each pixel value $I(x, w)$ as the density integral along the corresponding viewing ray defined by origin x and direction w :

$$I(x, w) = \int_t f(x + w * t) dt \quad (8)$$

This model is equivalent with the Fourier volume rendering resulting in simulated X-ray images. Similar visual effect can be achieved approximating the density integrals by the maximum density value along the viewing rays, as below:

$$I(x, w) = \max_t f(x + w * t) \quad (9)$$

There are several models available as shown in Figure 8 :

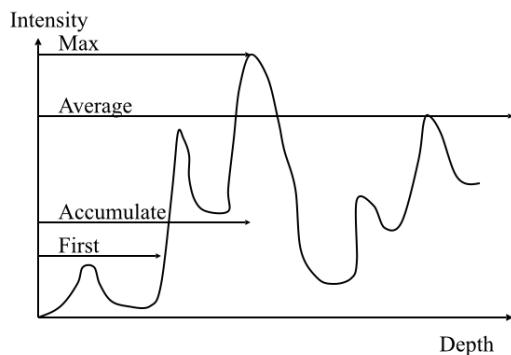


Figure 8. Ray Traversal Schemes

For early methods of direct volume rendering, transparency was not considered till 1988, researchers did not consider sophisticated light transportation theory, and were concerned with quick solutions, hence models at that time were more or less applied to binary data, since non binary data requires sophisticated classification and compositing methods. The first ray traversal was only considering the first

volume data the ray reaches for extracting the iso surface, it was developed by Tuy in 1984 [66].

And then researchers came up with the idea of taking average value ray traversed for representation, and this method produces basically an X-ray picture. Some researchers came up with another idea that they treated Maximum Intensity Projection for magnetic resonance image (MRI) rendering. Accumulate opacity method was developed by Levoy in 1988, while compositing colors this method makes transparent layers visible. [76]

Among all models introduced by tons of researchers, one well known in medical imaging as maximum intensity projection (MIP) and it is mainly used for visualization of blood vessel structures. Assuming that a trilinear filter is applied for function reconstruction the exact maximum densities can be analytically calculated [86]. In practice the density profile is approximated by a piecewise constant function taking a finite number of evenly located samples, and the maximum density sample is assigned to the given pixel.

The main drawback of maximum intensity projection is the loss of depth information. For example, in a medical application it might be confusing that a higher density blood vessel can “hide” other blood vessels which are closer to the view-point. In order to avoid this problem Sato proposed a technique called local maximum intensity projection (LMIP) [85]. Instead of the global maximum along the corresponding viewing ray the first local maximum which is above a predefined threshold is assigned to each pixel, as is shown below Figure 9 :

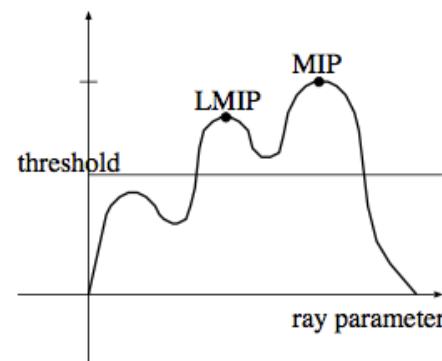


Figure 9. Local Maximum Intensity Projection

In order to model physical phenomena like scattering or attenuation optical properties are assigned to the volume samples as functions of their density values. Each pixel intensity $I(x, w)$ is composed from the assigned properties of the samples along the associated viewing ray according to the well known light transport equation [76] [80] [79] [81]:

$$I(x, w) = \int_s e^{\int_0^s \sigma(t) dt} I(s) ds \quad (10)$$

or in its discrete form, as in the limit as the sample spacing goes to zero, is approximated by a summation of equally spaced samples [82] [83] :

$$I = \sum_{0 \leq i \leq n} \left(\prod_{0 \leq i \leq j} e^{-\int_j^{j+1} \sigma(t) dt} \right) I(i) \quad (11)$$

where x is the origin of the ray, w is the unit direction of the ray, $\sigma(t)$ s the differential attenuation at $x + w * t$, and $I(s)$ is the differential intensity scattered at $x + w * s$ in the direction $-w$.

Introducing $\alpha(j)$ as the accumulated opacity of ray segment $[j, j + 1]$, this method is also known as alpha blending :

$$\alpha(j) = 1 - e^{-\int_j^{j+1} \sigma(t) dt} \quad (12)$$

I can be evaluated recursively running through each i th sample in back to front order:

$$I_{out} = \alpha(i) * I(i) + (1 - \alpha(i)) * I_{in} \quad (13)$$

where I_{in} is the intensity before the evaluation of the i th sample, $I(i)$ is the scattering of the i th sample, and I_{out} is the intensity after having the contribution of the i th sample added to the weighted sum. The initial value of I_{in} is the ambient light. In fact, $I_{out} = \alpha(i) * I(i) + (1 - \alpha(i)) * I_{in}$ is the Porter Duff over operator used for compositing digital images [84]. In the following subsections two different strategies are presented for approximating the light transport equation using the over operator.

Pseudocode: Alpha Blending

```

1  rgb AlphaBlending(int x, int z) {
2      int y, i, segment;
3      rgb color = BLACK;
4      voxel v;
5      double trans = 1.0; for(y = 0; y < 8; y++) {
6          segment = mask[z][y][x]);
7          while(segment) {
8              i = Trace(segment);
9              v = volume[z][y][x-offset[y*32+i];
10             trans *= 1.0 - v.opacity;
11             if(trans < threshold) return color;
12             color += v.color * v.opacity * trans;
13             segment &= ~(0x80000000 >> i);
14         }
15     }
16     return color;
17 }
```

In this report, direct volume rendering techniques are classified further into two categories. The object order methods process the volume voxel by voxel projecting them onto the image plane, while the image-order methods produce the image pixel by pixel casting rays through each pixel and resampling the volume along the viewing rays.

The direct techniques represent a very flexible and robust way of volume visualization. The internal structures of the volume can be rendered controlled by a transfer function which assigns different opacity and color values to the voxels according to the original data value. Although there is no need to generate an intermediate representation direct volume rendering is rather time consuming because of the enormous number of voxels to be processed.

2.2.1 Image order methods

Image order rendering, also called backward mapping, ray casting, pixel space projection, or image-space rendering, is fundamentally different from object order rendering. Image order techniques consider each pixel of the resulting image separately. For each pixel, the contribution of the entire volume to this pixels's final color is computed.

By the late 80's, a number of surface extraction techniques had been developed, like the marching cubes algorithm. Surface extraction and rendering using polygons works fairly well on arbitrary data, but it does have one big drawback; aliasing effects due to the difficulty of classifying where the actual surface is. This problem is further amplified by the fact that the marching cubes algorithm uses a binary classification scheme which means that a data point is either on the surface, or it is not. Obviously, that kind of classification is not very precise, and it does not work well even with real numbers. Polygons

are generally not well suited to display, complex, fine details, especially since a near infinite number is needed. The classification errors produce aliasing artefacts, meaning that surface features that do not exist in the dataset are embedded in the final rendered picture. For applications used in the medical field in particular, aliasing or any artefacts at all are not acceptable.

One solution to the classification problem is to use a technique called raycasting. The basic algorithm is simple, rays are cast into a data volume and samples are taken along each ray by interpolation of the surrounding voxels. This means that no intermediate geometry is constructed and thus the classification problem is solved. Another advantage is that the volume can be rendered semi-transparent, and as a result it is possible to display many surfaces within each other.

Rene Descartes introduced ray tracing back in 1637, the idea of tracing light rays and their interaction between surfaces. He applied the laws of refraction and reflection to a spherical water droplet to demonstrate the formation of rainbows. The first ray casting algorithm used for rendering was presented by Arthur Appel in 1968 [87]. The idea behind ray casting is to shoot rays from the eye, one per pixel, and find the closest object blocking the path of that ray. Using the material properties and the effect of the lights in the scene, this algorithm can determine the shading of this object. The simplifying assumption is made that if a surface faces a light, the light will reach that surface and not be blocked or in shadow. The shading of the surface is computed using traditional 3D computer graphics shading models. One important advantage ray casting offered over older scanline algorithms is its ability to easily deal with non-planar surfaces and solids. If a mathematical surface can be intersected by a ray, it can be rendered using ray casting. Elaborate objects can be created by using solid modeling techniques and easily rendered.

In 1980, Turner Whitted [88] used the basic ray casting algorithm but extended it. When a ray hits a surface, it could generate up to three new types of rays, reflection, refraction, and shadowing. A reflected ray continues on in the mirror reflection

direction from a shiny surface. It is then intersected with objects in the scene; the closest object it intersects is what will be seen in the reflection. Refraction rays travelling through transparent material work similarly, with the addition that a refractive ray could be entering or exiting a material. To further avoid tracing all rays in a scene, a shadow ray is used to test if a surface is visible to a light. A ray hits a surface at some point. If the surface at this point faces a light, a ray is traced between this intersection point and the light.

Back in 1984, method cast parallel or perspective rays from the pixels of the image plane was proposed, Tuy's work known as binary ray casting determines only the first intersection points with a surface contained in the volume [66]. Binary ray casting aims at the visualization of surfaces contained in binary volumetric data. Along the viewing rays the volume is resampled at evenly located sample points and the samples take the value of the nearest voxel. When the first sample with a value of one is found the corresponding pixel color is determined by shading the intersected surface point.

Then in 1988, Levoy [76] published a paper with a raycasting algorithm which since has become the definition for raycasting. Direct volume rendering of gray level volumes is not restricted to surface shaded display like in the case of binary data sets. Here a composite projection of the volume can be performed by evaluating the light transport equation along the viewing rays. Composition requires two important parameters, the color and an opacity at each sample location. Levoy [76] proposed an image order algorithm which assigns these parameters to each grid location in a preprocessing step. The opacity and color values at an arbitrary sample point are calculated by first order interpolation.

In this report, ray casting developed by Levoy [76] is considered as a typical image order algorithm and will be explained in the following section. Images generated by ray casting represent the reference results in terms of image quality, which is shown as Figure 10 :

In ray casting, rays are cast into the dataset. Each ray originates from the viewing point, and penetrates a pixel in the image screen, and passes

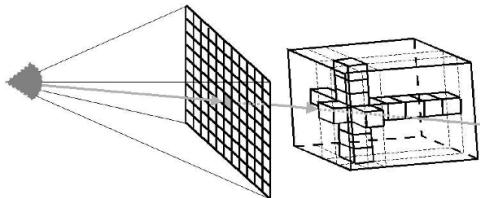
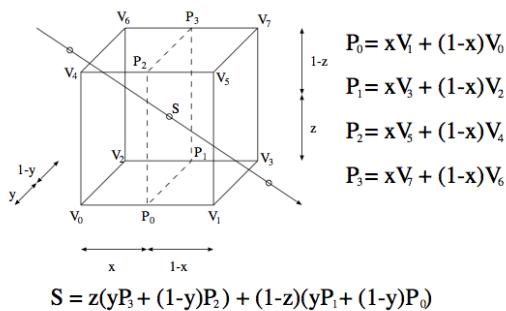


Figure 10. Image Order Rendering

through the dataset. At evenly spaced intervals along the ray, sample values are computed using interpolation [ref: Figure 11]. The sample values are mapped to display properties such as opacity and color. A local gradient is combined with a local illumination model at each sample point to provide a realistic shading of the object. Final pixel values are found by compositing the color and opacity values along the ray. The composition models the physical reflection and absorption of light [65]. Composite ray casting is a flexible approach for visualizing several semi transparent surfaces contained in the data and produces high quality images. However, the alpha blending evaluation of viewing rays is computationally expensive, especially when super sampling is performed trilinearly interpolating each single sample.



$$S = z(yP_3 + (1-y)P_2) + (1-z)(yP_1 + (1-y)P_0)$$

Figure 11. Resampling using trilinear interpolation.

Pseudocode: Standard Recursive Algorithm [76]

```

1 For each pixel in image {
2   Create ray from eyepoint passing through this pixel
3   Initialize NearestT to INFINITY and NearestObject to NULL
4   For every object in scene {
5     If ray intersects this object {
6       If t of intersection is less than NearestT {
7         Set NearestT to t of the intersection
8         Set NearestObject to this object
9       }
10    }
11  }
12  If NearestObject is NULL {
13    Fill this pixel with background color
14  }

```

```

15  Else {
16    Shoot a ray to each light source to check if in shadow
17    If surface is reflective, generate reflection ray: recurse
18    If surface is transparent, generate refraction ray: recurse
19    Use NearestObject and NearestT to compute shading function
20    Fill this pixel with color result of shading function
21  }
22 }

```

Another alternative is discrete ray casting or 3D raster ray tracing [67], where the continuous rays are approximated by discrete 3D lines generated by Discrete Bresenham algorithm [68] or continuous scan conversion algorithm. While traditional ray tracers are capable of rendering only objects represented by geometric surfaces, discrete ray casting is also attractive for ray tracing 3D sampled datasets like 3D MRI imaging, and computed datasets like fluid dynamics simulations, as well as hybrid models in which such datasets are intermixed with geometric models, such as scalpel superimposed on a CT image, radiation beams superimposed on a scanned tumor, or a plane fuselage superimposed on a computed air pressure [70]. Unlike nonrecursive ray casting techniques, discrete ray casting, which recursively considers both primary and secondary rays can model shadows and reflections for photorealistic imaging. Discrete ray casting offers the use of ray tracing for improved visualization of sampled and computed volumetric datasets [69].

Pseudocode: Discrete Ray Casting [69]

```

1 algorithm to determine closest object intersected by a ray
2   for each object in the scene
3     if the ray intersects with the object then
4       if the object is closest to the ray source then
5         the object is the closest intersected by the ray
6       endif
7     endif
8   endfor
9 endalgorithm

```

The closest intersection points are stored for each pixel and afterwards an image space depth gradient shading [72] [71] can be performed. Better results can be achieved applying object space shading techniques like normal based contextual shading [74] [75]. Normal computation methods based on surface approximation try to fit a linear [73] or a biquadratic [77] [78], where function to the set of points that belong to the same iso surface. These techniques take a larger voxel neighborhood into account to estimate the surface inclination.

There are some advantages and disadvantages of raycasting algorithm.

Advantages of Raycasting:

1. Realistic simulation of lighting, better than scanline rendering or ray casting;
2. Effects such as reflections and shadows, which are difficult to simulate using other algorithms, are a natural result of the ray tracing algorithm;
3. Relatively simple to implement yet yielding impressive visual results.

Disadvantages of Raycasting:

1. Performance is very poor;
2. Scanline algorithms and other algorithms use data coherence to share computations between pixels, while ray tracing normally starts the process anew, treating each eye ray separately;
3. However, this separation offers other advantages, such as the ability to shoot more rays as needed to perform anti-aliasing and improve image quality where needed. Although it does handle inter-reflection and optical effects such as refraction accurately;
4. Other methods, including photon mapping, are based upon ray tracing for certain parts of the algorithm, yet give far better results.

2.2.2 Object order methods

Object order algorithms start with a single voxel and compute its contribution to the final image. This task is iteratively performed for all voxels of the data set. Object order rendering is also called forward rendering, or object space rendering or voxel space projection. It loops through the data samples, projecting each sample onto the image plane, which is shown as Figure 12 :

The simplest way to implement viewing is to traverse all the volume regarding each voxel as a 3D point that is transformed by the viewing matrix and then projected onto a Z-buffer and drawn onto the screen. The data samples are considered with a

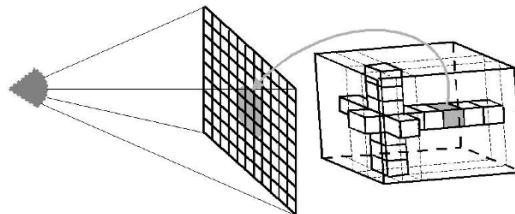


Figure 12. Object Order Rendering

uniform spacing in all three directions. If an image is produced by projecting all occupied voxels to the image plane in an arbitrary order, a correct image is not guaranteed. If two voxels project to the same pixel on the image plane, the one that was projected later will prevail, even if it is farther from the image plane than the earlier projected voxel. This problem can be solved by traversing the data samples in a back to front or front to back order. This visibility ordering is used for the detailed classification of object order rendering.

The first object order algorithm reported in literature was a rendering method presented by Upson and Keeler [93], which processed all voxels in front to back order and accumulated the values for the pixels iteratively. Similarly to the early image order methods, aimed at the rendering of binary volumes. Processing each sample, only the voxels with value of one are projected onto the screen. The samples are projected in back to front order to ensure correct visibility. If two voxels are projected onto the same pixel, the first processed voxel must be farther away from the image plane. This can be accomplished by traversing the data plane by plane, and row by row inside each plane. For arbitrary orientations of the data in relation to the image plane, some axes may be traversed in an increasing order, while others may be considered in a decreasing order. The ordered traversal can be implemented with three nested loops indexing $x-$, $y-$, and $z-$ directions respectively. Such an implementation supports axis-parallel clipping planes. In this case, the traversal can be limited to a smaller rectangular region by simply modifying the bounds of the traversal. The depth image of the volume can be easily generated. Whenever a voxel is projected onto a pixel, the pixel value is overwritten by the distance of the given voxel from the image plane. Similarly to the

early image-based methods the distance image can be passed to a simple 2D discrete shader.

Further development of this idea lead to splatting [92] [91] [90] [95] are not restricted to the rendering of binary volumes. Splatting is an algorithm which combines efficient volume projection with a sparse data representation. In splatting, each voxel is represented as a radially symmetric interpolation kernel, equivalent to a sphere with a fuzzy boundary. Projecting such a structure generates a so called footprint or splat on the screen. Splatting traditionally classifies and shades the voxels prior to projection.

Many many improvements since Westover's approach was published. Crawfis introduced textured splats, Swan and Mueller solved anti aliasing problem, Mueller himself developed image aligned sheet based splatting and post classified splatting in 1998 and 1999 respectively [105]. Object order approaches also comprise cell projection [89] and 3D texture mapping.

In Mueller's approach, a gray scale volume is treated as a 3D discrete density function. Similarly to the ray casting method a convolution kernel defines how to reconstruct a continuous function from the density samples. In contrast, instead of considering how multiple samples contribute to a sample point, it is considered how a sample can contribute to many other points in space. For each data sample $s = (x_s, y_s, z_s)$, a function C defines its contribution to points noted as (x, y, z) in the space:

$$C(x, y, z) = h(x - x_s, y - y_s, z - z_s) * f(s) \quad (14)$$

where $f(s)$ is the density of sample s . The contribution of a sample s to an image plane pixel (x, y) can be computed by an integration:

$$C(x, y) = f(s) * \int h(x - x_s, y - y_s, u) du \quad (15)$$

where u coordinate axis is parallel to the viewing ray. Since this integral is independent of the sample density, and depends only on its (x, y) projected location, a footprint function F can be defined as follows:

$$F(x, y) = \int h(x - x_s, y - y_s, u) du \quad (16)$$

where (x, y) is the displacement of an image sample from the center of the sample's image plane projection. The footprint kernel F is a weighting function which defines the contribution of a sample to the affected pixels. A footprint table can be generated by evaluating the integral, on a grid with a resolution much higher than the image plane resolution. All the table values lying outside of the footprint table extent have zero weight and therefore need not be considered when generating an image. A footprint table for data sample s is centered on the projected image plane location of s , and sampled in order to determine the weight of the contribution of to each pixel on the image plane.

Computing a footprint table can be difficult due to the integration required. Although discrete integration methods can be applied to approximate the continuous integral, generating a footprint table is still a costly operation. However, in case of orthographic projection, the footprint table of each sample is the same except for an image plane offset. Therefore, only one footprint table needs to be calculated per view. Since this would require too much computation time anyway, only one generic footprint table is built for the kernel. For each view, a view transformed footprint table is created from the generic footprint table. The generic footprint table can be precomputed, therefore it does not matter how long the computation takes.

Pseudocode: Splatting an i-axis row of a Data Slice [105]

```

1  /* screen coords of transformed point */
2  float scr[2];
3  /* screen space step for each step along i-axis of data */
4  float step[2];
5  /*pointer to footprint array*/
6  float *foot;
7  float alpha, thresh;
8  /*first and last on-screen points in row*/
9  int firstpt, lastpt;
10 /*index into data set*/
11 int index;
12 /*pixel address offsets for kernel loop*/
13 int off[];
14 /*data set axes indices*/
15 int i, j, k;
16 /*lowest pixel coords of footprint coverage */
17 int lo[2];
18 /*size of kernel*/
19 int ksize;
20 int lopix, p;
21 /* screen position of first point */
22 scr[] = transformed(k, j, firstpt);
23 /* dda increment for each i-step */
24 step[] = f(view_transform);
25 /* index of first point in j'th row and k'th slice */
26 index = f(k, j, firstpt);
27 /* do all on-screen points in row */
28 for (i = firstpt; i <= lastpt; i++) {
29    /* load data point value */

```

```

30 alpha = data[index++];
31 /* test for significant data */
32 if (alpha > thresh) {
33 /* footprint is function of fractional screen position */
34 foot = f(scr[]);
35 /* find lowest pixel of footprint coverage */
36 lo[] = (int)(scr[] - extent);
37 /* index to lowest pixel */
38 lopix = f(lo[]);
39 /* loop over kernel size */
40 for (p = 0; p < ksize; p++) {
41 /* accumulate contribution */
42 image[lopix + off[p]] += alpha * foot[p];
43 }
44 }
45 /* step to position of next point on screen */
46 scr[] += step[];
47

```

There are three modifiable parameters of the splatting algorithm which can strongly affect the image quality. First, the size of the footprint table can be varied. Small footprint tables produce blocky images, while large footprint tables may smooth out details and require more space. Second, different sampling methods can be used when generating the view-transformed footprint table from the generic footprint table. Using a nearest neighbor approach is fast, but may produce aliasing artifacts. On the other hand, using bilinear interpolation produces smoother images at the expense of longer rendering times. The third parameter which can be modified is the reconstruction filter itself. The choice of, for example, a cone function, Gaussian function, Sinc function or bilinear function affects the final image.

Alpha blending composition is also supported by the splatting algorithm. The voxel contributions of slices mostly perpendicular to the viewing direction are evaluated on associated sheets parallel to the image plane. After having each sheet generated image composition is performed applying the Porter-Duff over operator.

Using the splatting algorithm approximately the same image quality can be achieved as applying a composite ray casting. The advantages of splatting over ray casting are the following. First, the cache coherency can be exploited since the voxels are sequentially traversed in the same order as they are stored in memory. In contrast, ray casting requires random access to the voxels. Furthermore, the splatting approach supports incremental rendering in back to front or front to back order. Using splatting smooth surfaces can be rendered without staircase artifacts, unlike in the case of ray casting. The main drawback of splatting is that the gener-

ated images are blurred because of the spherically symmetric reconstruction kernel. In contrast, using ray casting with trilinear reconstruction sharp object boundaries are obtained.

Advantages of Splatting:

1. Footprints can be pre-integrated, which ensured fast voxel projection;
2. Fast: voxel interpolation is in 2D on screen;
3. More accurate integration (analytic for X-ray);
4. More accurate reconstruction (afford better kernels);
5. Only relevant voxels must be projected.

Disadvantages of Splatting:

1. Mathematically, the early splatting methods only work for X-ray type of rendering, where voxel ordering is not important, bad approximation for other types of optical models;
2. Object ordering is important in volume rendering, front objects hide back objects need to composite splats in proper order, else we get bleeding of background objects into the image (color bleeding);
3. However, this separation offers other advantages, such as the ability to shoot more rays as needed to perform anti-aliasing and improve image quality where needed. Although it does handle inter-reflection and optical effects such as refraction accurately;
4. Axis aligned approach add all splats that fall within a volume slice most parallel to the image plane, composite these sheets in front to back order, incorrect accumulating on axis aligned face cause popping;
5. A better approximation with Riemann sum is to use the image aligned sheet based approach.

3. Acceleration Techniques

Early implementations of volume rendering used brute-force techniques that require on the order of 100 seconds to render typical data sets on a workstation. Algorithms with optimizations that exploit coherence in the data have reduced rendering times to the range of ten seconds but are still not fast enough for interactive visualization applications.

Many of the three dimensional data sets that need to be visualised contain an interesting range of values throughout the volume. By interesting, it is meant those parts of the volume to which the viewer's attention must be drawn in order for the viewer to gain insight to the physical phenomena the data represents. If the range of values is small, as for example the visualisation of the human skull from CT scans, then a surface tiling method will suffice.

Volume rendering offers an alternative method for the investigation of three dimensional data, such as surface tiling as described by Jones [96], marching cubes supported iso surface rendering by Lorensen and Cline [33], octree acceleration for faster iso surface generation by Wilhelms and Van Gelder [97], special data structure for rendering by Wyvill et. al. [98] and surface mapping method by Payne and Toga [99].

Surface tiling can be regarded as giving one particular view of the data set, one which just presents all instances of the threshold value. All other values within the data are ignored and do not contribute to the final image. This is acceptable when the data being visualised contains a surface that is readily understandable, as is the case when viewing objects contained within the data produced by CT scans. In certain circumstances this view alone is not enough to reveal the subtle variations in the data, and for such data sets volume rendering was developed [76] [83] [93] [100].

Most data sets do not fall into this category, but rather have a larger range of values or several different values which need to be represented in the visualisation. Such data sets need a method which can display the volume as a whole and visualise correctly those data values in which the viewer is

interested.

There are several widely used optimization methods, for early ray termination and empty space skipping. Early ray termination compares accumulated opacity against threshold, such as marching cubes algorithm [33], and in such a way accelerates rendering process. Empty space skipping method utilize additional data structure [98], and encoding empty space in volume, such as Octree algorithm [97], which encodes measure of empty within 3D texture read from fragment shader, and performs raymarching fragment shader can modulate sampling distance based on empty space value.

In this report, we will present the fast image order and object order methods respectively. It will be shown that the advantageous properties of these two different approaches are complementary. Therefore, hybrid methods which combine image order and object order techniques have been proposed by several authors will be described right after. One of them is a two pass raycasting and back projection algorithm which exploits the frame to frame coherency. Another one is the classical shear warp algorithm, which is based on run length encoding of volume and image scanlines exploiting the volume and image coherency respectively.

3.1 Fast image order techniques

3.1.1 Hierarchical data structures

Hierarchical data structures like octrees, k-d trees, or pyramids are used for image order volume rendering to efficiently encode the empty regions in a volume. Such data structures are widely used in computer graphics for accelerating traditional algorithms, like ray tracing. Among which, the most widely used method is octree. The use of octrees for 3D computer graphics was pioneered by Donald Meagher at Rensselaer Polytechnic Institute in 1980. [101] The idea of using octree is to quickly find the first intersection point for an arbitrary ray without evaluating the intersections with all the objects. Ray tracing in continuous analytically defined scenes requires a hierarchical structure with arbitrarily fine resolution. In contrast, in volume rendering the discrete representation of the scene can be exploited. Then a pointerless complete

octree represented by a pyramid was introduced by Levoy [80] [102].

Assuming that the resolution of the volume is $N * N * N$, where $N = 2^M + 1$ for some integer M . A pyramid is defined as a set of $M + 1$ volumes. Volumes are indexed by a level number $m = 0, 1, 2, \dots, M$, and the volume at level m is denoted by V_M . Volume V_0 measures $N - 1$ cells on a side, volume V_1 measures $(N - 1)/2$ cells on a side and so on up to the volume V_M which is a single cell.

Levoy applied a binary pyramid in order to quickly traverse the empty regions in a volume. [80] A cell of V_0 represents the rectangular regions between eight neighboring voxels in the original data. The value of a cell in V_0 is zero if all of its eight corner voxels have opacity equal to zero, otherwise its value is one. At higher levels of the pyramid zero is assigned to a cell if all the corner cells at one level lower have value of zero.

For each ray, first the point where the ray enters a single cell at the top level is calculated. Afterwards the pyramid is traversed in the following manner. Whenever a ray enters a cell its value is checked. If it contains zero the ray advances to the next cell at the same level. If the parent of the next cell differs from the parent of the previous one then the parent cell is investigated and the ray is traced further at one level higher. If the parent cell is empty then it can be skipped, and the iteration continues until a non-empty cell is found. In this case, moving down in the hierarchical structure, the first elementary cell is determined which has at least one opaque corner voxel. In such an elementary cell samples are taken at evenly spaced locations along the ray and compositing is performed. Using such a hierarchical ray traversal larger empty regions can be easily skipped. Since the non-empty cells in the binary pyramid represent the regions, where opaque voxels are present the algorithm is called presence acceleration.

In 1992, Denskin and Hanrahan [79] improved this algorithm using pyramids not only for skipping the empty ray segments but for approximate evaluation of homogeneous regions. Therefore, their technique is called homogeneity acceleration. Instead

of using a binary pyramid they construct a so called range pyramid which contains the maximum and minimum values of subvolumes at one level lower. If the maximum and minimum values of a cell are nearly the same then it is considered homogeneous and an approximate evaluation is performed.

3.1.2 Early ray termination

To reduce the time complexity of volume rendering, Leovy [76] came up with a technique named as early ray termination (ERT), which adaptively terminates accumulating color and opacity values in order to avoid useless ray casting. This technique reduces the execution time by roughly a factor of between 5 and 11.

ERT reduces the computational amount by avoiding accumulation of color and opacity values that do not have influence on the final image. Associating an accumulated opacity to each pixel of the image plane ray casting can be performed evaluating the rays in front to back order, in case of back to front composition, all the samples along the ray have to be taken into account. This computation is usually redundant since several samples can be occluded by a ray segment which is closer to the viewer and has accumulated opacity of one. Therefore, these samples do not contribute to the image. In contrast, using front to back composition, the rays can terminate when the accumulated opacity exceeds a predefined threshold [80]. This technique is well known as early ray termination or α acceleration.

This acceleration method introduces a systematic bias in the image because of the predefined threshold. In order to avoid this, a technique called Russian Roulette can be used for unbiased estimation of pixel values [79].

Earlier parallel schemes can be classified into two groups: screen parallel and object parallel rendering as illustrated in Figure 13 :

Screen parallel rendering exploits the parallelism in screen space. In this scheme, the screen is divided into p subscreens, where p represents the number of processors, and tasks associated with each subscreen are assigned to processors. Because each processor takes responsibility for the entire of a ray as it does in sequential schemes, ERT can

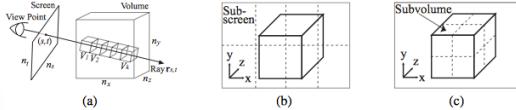


Figure 13. Ray casting and its parallel schemes : (a) ray casting by exploiting the parallelism in screen space and in object space (b) Screen-parallel rendering and (c) object-parallel rendering parallelize, respectively.

easily be applied to this scheme, as illustrated in Figure 14 (a). Furthermore, by assigning the tasks in a cyclic manner, this scheme statically balances the processing workloads. However, it requires large main memory to provide fast rendering for any given viewpoint, because every processor need to load the entire volume into memory. Thus, though screen parallel rendering is a good scheme for small datasets, which require no data decomposition, it does not suit for large scale datasets.

In contrast, object parallel rendering exploits the parallelism in object space. This scheme divides the volume into p subvolumes, and then assigns tasks associated with each subvolume to processors. Parallel rendering of each subvolume generates p distributed subimages, so that image compositing is required to merge subimages into the final image. Thus, this scheme allows us to distribute subvolumes to processors, so that is suitable for large scale datasets. However, because accumulation tasks of a ray can be assigned to more than one processor, it is not easy to utilize global ERT in this scheme.

Figure 14 shows an example of local ERT in object-parallel rendering. In this example, voxels from V_1 to V_4 are visible from the viewpoint while voxels from V_5 to V_9 are invisible. These voxels are assigned to three processors, so that each processor takes responsibility for three of the nine voxels. In object parallel rendering, the reduction given by ERT is localized in each processor, because processors take account of the local visibility instead of the global visibility.

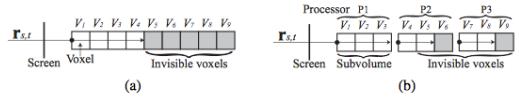


Figure 14. Early ray termination (ERT). (a) Global ERT for sequential and screen-parallel rendering, and (b) local ERT for object-parallel rendering. While global ERT terminates the ray immediately before invisible voxel V_5 , local ERT fails to avoid accumulating locally visible but globally invisible voxels: V_5 , V_7 , and V_8 . Voxels V_6 and V_9 are invisible locally as well as globally.

3.1.3 Distance transformation

The main drawback of acceleration techniques based on hierarchical data structures is the additional computational cost required for the traversal of cells located at different levels of the hierarchy. Furthermore, the rectangular cells only roughly approximate the empty regions.

Cohen proposed a technique called proximity clouds for fast 3D grid traversal. Here the geometric information required for empty space skipping is available with the same indices used for the original volume data [103]. The data structure is a simple 3D distance map generated from binary volumes. The input binary volume encodes the transparent and opaque cells similarly to Levoy's approach [80]. In the distance volume each cell contains the distance to the nearest non-empty cell. The distance between two cells is represented by the Euclidean distance of their center points.

Ray casting is performed applying two different cell traversal strategies. The algorithm switches between these two strategies depending on the distance information stored in the current cell. Using fixed-point arithmetic and integer division it is easy to find the cell which contains the current ray sample. If the current sample is in the vicinity of an object a simple incremental ray traversal is performed.

If this is not the case, the distance value d stored in the current cell is used for fast skipping of empty regions. The new sample is determined by adding the unit direction of the given ray multiplied by

$d - 1$ to the current sample location. The distance from the nearest object has been calculated from the center of the current cell, therefore using a stepping distance $d - 1$, skipping beyond the free zone can be avoided.

Distance maps can be generated based on several distance metrics, like City Block, Euclidean, or Chessboard distance. Approximate distance maps are usually calculated applying the efficient Chamfering method. The basic idea is to use a mask of local distances and propagate these distances over the volume. The generation of a more accurate distance map requires a larger mask, therefore the pre-processing time is longer. On the other hand, less samples have to be taken in the raycasting process when more exact distance information is available. Therefore, the applied distance metric is a compromise between the preprocessing and rendering times.

3.2 Fast object-order techniques

3.2.1 Hierarchical splatting

Object-order volume rendering typically loops through the data, calculating the contribution of each volume sample to pixels on the image plane. This is a costly operation for high resolution data sets. One possibility is to apply progressive refinement. For the purpose of interaction, first a lower quality image is rendered. This initial image is progressively refined when a fixed viewing direction has been selected.

For binary data sets, bits can be packed into bytes such that each byte represents a $2 * 2 * 2$ portion of the data [66]. The volume is processed bit by bit to generate the full resolution image but lower resolution images can be produced processing the volume byte by byte. A byte is considered to represent an element of an object if it contains more than four non zero bits, otherwise it represents the background. Using this technique, an image with one half the linear resolution is produced in approximately one eighth the time.

A more general method for decreasing data resolution is to build a pyramid data structure, which for an original data set of N^3 samples, consists of a sequence of $\log(N)$ volumes. The first volume is

the original data set, while the second volume of one-eighth the resolution is created by averaging each $2 * 2 * 2$ group of samples of the original data set. The higher levels of the volume pyramid are created from the lower levels in a similar way until $\log(N)$ volumes have been created. An efficient implementation of the splatting algorithm, called hierarchical splatting [104] uses such a pyramid data structure. According to the desired image quality, this algorithm scans the appropriate level of the pyramid in a back to front order. Each element is splatted onto the image plane using the appropriate sized splat. The splats themselves are approximated by polygons which can be rendered by conventional graphics hardware.

Laur and Hanrahan [104] introduced hierarchical splatting for volume rendering using Gouraud shaded polygons. Researchers like Mueller [105], Swan [106], and Zwicker [113] focus mainly on the improvement of the visual quality of texture splatting; however, the techniques described in these papers only apply to the reconstruction of continuous functions, take volume rendering of regular grid data for example, and they do not address adaptive rendering or data size reduction. Additionally, there exist a number of non-realtime rendering systems for large point based data sets, e.g. for rendering film sequences [108].

Using points as rendering primitives is a topic of ongoing research. However, almost all publications in this area deal with the rendering of geometric surfaces. Alexa [109], Pfister [110], Rusinkiewicz and Levoy [111], Wand [112], and Zwicker [113] showed different methods to create data hierarchies of surfaces represented by sample points and how to render them efficiently. As the intrinsic model of points describing a surface is fundamentally different to the model used for scattered data, their clustering techniques cannot be applied in our case. Pauly [115] used principal component analysis for clustering, but with a different hierarchy concept compared to our approach. Some systems [111] [114] use quantized relative coordinates for storing the points in a hierarchical data structure, but these approaches were not optimized for fast GPU access because the data structures had to be interpreted

by the CPU. Additionally, the presented rendering techniques have been designed to create smooth surfaces without holes and they allow no or only few layers of transparency.

3.2.2 Extraction of surface points

Although there are several optimization techniques based on empty space leaping or approximate evaluation of homogeneous regions, because of the computationally expensive alpha blending compositing the rendering is still time demanding.

One alternative to alpha blending volume visualization is the extraction of relevant voxels and the optimization of the rendering process for sparse data sets. Following these approach interactive applications can be developed which support flexible manipulation of the extracted voxels. In medical imaging systems, for example, the cutting operations are rather important, where a shaded isosurface and an arbitrary cross sectional slice can be rendered at the same time.

Sobierajski [116] proposed a fast display method for direct rendering of boundary surfaces. From the volume data only those boundary voxels are extracted which are visible from a certain set of viewing directions. In many cases the six orthographic views are sufficient to obtain an approximate set of all the potentially visible boundary voxels. Better approximation can be achieved increasing the number of directions.

Taking only the six orthographic views into account the visibility calculations can be performed efficiently using a 2D boundary tracing algorithm on the slices perpendicular to each coordinate axis. The output of the surface extraction algorithm is a list of boundary voxels in which duplicate elements are removed. The generated list stores for each voxel all the attributes which are necessary for the rendering process, like the coordinates or the normal vector.

The set of surface points is passed to the rendering engine. Since adjacent voxels are mapped onto not necessarily adjacent pixels, holes can appear in the produced image. In order to avoid this problem one voxel is projected onto several pixels depending upon the viewing direction. Since the

rendering speed is directly related to the length of the voxel list, for a specific viewing direction the number of voxels to be projected can be reduced by voxel culling. This is similar to back face culling in polygon rendering [68]. If the dot product of the surface normal and the viewing direction is positive the given voxel belongs to a back face, therefore it is not rendered.

Since the presented algorithm follows a direct volume rendering approach, cutting planes can be easily implemented. The projected boundary surface points are shaded according to the lighting conditions and the voxels intersected by the cutting plane are rendered by projecting their original density values onto the image plane.

A fast previewing algorithm proposed by Saito [117] is also based on the extraction of boundary voxels. Similarly to the previous method a set of surface points is stored in a list and it is passed to the rendering engine. In contrast, in order to increase the rendering speed, only a subset of the boundary voxels are extracted according to a uniform distribution. The extracted surface samples are converted to geometrical primitives like crosslines perpendicular to the surface normal and projected onto the image plane.

3.3 Hybrid acceleration methods

3.3.1 Shear-warp factorization

The shear-warp algorithm is a purely software based renderer. Shear-warp was invented by Lacroute [118] and can be considered a hybrid between image order algorithms, such as raycasting, and object-order algorithms, such as splatting. In shear warp, the volume is rendered by a simultaneous traversal of run length encoding (RLE) encoded voxel and pixel runs, where opaque pixels and transparent voxels are efficiently skipped during these traversals. Further speed comes from the fact that a set of interpolation weights is precomputed per volume slice and stays constant for all voxels in that slice. The caveat is that the image must first be rendered from a sheared volume onto a so called base plane, aligned with the volume slice most parallel to the true image plane Figure 15 :

After completing the base plane rendering, the

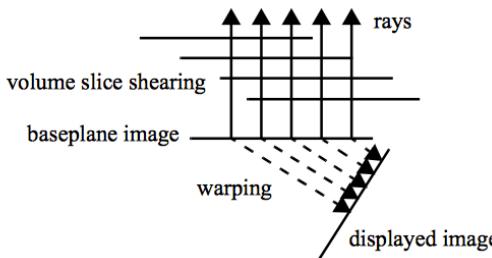


Figure 15. A sketch of the shear-warp mechanism

base plane image is warped onto the true image plane and the resulting image is displayed.

Pseudocode: Pseudocode of the standard shear warp algorithm.

```

1  Shear_Warp (voxel_data)
2  if New(Mview)
3  shade_table = Construct_Shade_Cube(Mview, L, E);
4  if New(transfer_function)
5  RLE_Encode(voxel_data, RLE_X, RLE_Y, RLE_Z);
6  if (major_viewing_axis == X) current_RLE = RLE_X;
7  if (major_viewing_axis == Y) current_RLE = RLE_Y;
8  if (major_viewing_axis == Z) current_RLE = RLE_Z;
9  Factorize(Mview, Mshear, Mwarp);
10 ShearParameters(Mshear, shear_u, shear_v, trans_u, trans_v);
11 base_image = Render(current_RLE);
12 display_image = Warp(base_image, Mwarp);
13
14
15 Render (current_RLE)
16 base_image.Initialize();
17 num_scanline_pixels = slice_width;
18 num_scanlines = slice_height;
19 for k = front_slice, k<=end_slice, k++
20 Composite_Slice (k);
21 return(base_image);
22
23
24 Composite_Slice (k)
25 slice_u = k.shear_u + translate_u;
26 slice_v = k.shear_v + translate_v;
27 slice_u_int=floor(slice_u);
28 slice_v_int=floor(slice_v);
29 weights[4]=Voxel_Weights(u, u_int, v, v_int);
30 for j= 0, j<=num_scanlines-1, j++
31 for i= 0, i<= num_scanline_pixels-1, i++
32 bot_run=Skip_Transparent_Voxels (i, j, k, current_RLE);
33 top_run=Skip_Transparent_Voxels (i, j+1, k, current_RLE);
34 pixel_start=Min (bot_run.start voxel, top_run.start voxel);
35 a=bot_run.start voxel + bot_run.length;
36 b=top_run.start voxel + top_run.length;
37 pixel_end=Max (a,b);
38 for pixel=pixel_start, pixel<=pixel_end, pixel++
39 pixel =Skip_Opaque_Pixels (pixel, j, base_plane_image.opacity_RLE);
40 if (pixel > pixel_end)
41 break;
42 voxel_square=Get_Voxel_Square (bot_run, top_run, pixel);
43 composited_opacity=Composite_Pixel (voxel_square, weights);
44 if composited_opacity > 0
45 Update_Opacity_RLE (pixel, base_plane_image.opacity_RLE);
46 i=pixel;
47
48
49 RLE_Encode (voxel_data, RLE_X, RLE_Y, RLE_Z)
50 for all voxels in voxel_data with AlphaTF[voxel.density] > 0
51 RLE_voxel.shade_index=Calc_Shade_Index (voxel.gradient);
52 RLE_voxel.density=voxel.density;
53 RLE_X.Add (RLE_X, RLE_voxel);
54 RLE_Y.Add (RLE_voxel);
55 RLE_Z.Add(RLE_voxel);
56
57
58 Composite_Pixel (voxel_square, weights)
59 voxel_square.opacities=AlphaTF[voxel_square.densities];
60 pixel_opacity=Interpolate (voxel_square.opacities, weights);
61 if pixel_opacity > 0
62 voxel_square.shades=
63   Get_Shades (shade_table, voxel_square.shade_indices);
64 voxel_square.colors=
65   Calc_Colors (voxel_square.shades, voxel_square.densities, TF);

```

```

66   pixel_color=Interpolate (voxel_square.colors, weights);
67   composited_opacity=
68     Composite (color, opacity, base_plane_image.pixels);
69   return (composited_opacity);

```

4. Challenges of Medical Volume Rendering

In the past decade, commercial CT scanners have become available that can take five 320 slice volumes in a single second. Toshiba's 320 slices CT scanner, the Aquilion One, was introduced in 2007. [125] This is fast enough to make 3D videos of a beating heart. Rapid advances in the dynamic nature and sheer magnitude of data force us to make improvements to existing techniques of medical visualization to increase computational and perceptual scalability.

Images of nerve bundles and muscle fibres is important for areas of study in neuroscience and biomechanics. High Angular Resolution Diffusion Imaging (HARDI) [127] and Diffusion Spectrum Imaging (DSI) [126] datasets contain hundreds of diffusion-weighted volumes describing the diffusion of water molecules and hence indirectly the orientation of directed structures, which are calling for new visualization techniques.

Then there are the imaging techniques that work on the level of molecules and genes. Up to now, most of the visualization research has been focused on small animal imaging [129] [128], but due to its great diagnostic potential, molecular imaging will see increased application in humans. The great potential of these is that they can reveal pathological processes at work long before they become apparent on the larger scale, such as bioluminescence (BLI) and fluorescence (FLI) imaging's applications on tumor detection.

Acquiring the image data is just one part of the challenge. Representing it visually in a way that allows the most effective analysis is also hugely difficult but again there have been huge advances.

One of the most spectacular is the representation of medical data topologically, in other words showing the surfaces of objects. That makes it possible to more easily see the shapes of organs and to plan interventions such as surgery.

Another of the most spectacular is the interactive representation of multi-modality medical data. Medical visualization research often combines elements of image analysis, graphics and interaction, and is thus ideally equipped to address the challenge of developing and validating effective interactive segmentation approaches for widespread use in medical research and practice. Also, integration of simulation models and MRI or CT is crucial for diagnostics as well. A huge challenge for the future and the subject of much current research, is to create images of the potential outcome of interventions that show the result of the surgery.

The most recent image processing techniques allow the addition of realistic lighting effects creating photo realistic images. Beyond this, hyper realistic images can show what lies beneath certain layers.

Another area of growing importance is the visualisation of multi-subject data sets, this is required as researchers want to study the onset and progression of disease, general aging effects, and so forth in larger groups of people. such as the Rotterdam Scan Study focusing on neuro degeneration [131] and the Study of Health In Pomerania (SHIP) focusing on general health [132], Steenwijk setting the first steps for the visualization of population imaging by applying visual analysis techniques to cohort study imaging data [130].

Last but not least, with increasing population of mobile device, such as iPhone and iPad, cheaper and easy compiled visualization softwares are needed for medical purposes, to free doctors from desktops. The biggest challenge of all is to find ways of making powerful medical visualisation techniques cheap enough for everyone.

Acknowledgments

Thanks for Gelman library and Prof. Hahn for supporting.

References

- [1] Greicius, M.D., Krasnow, B., Reiss, A.L., Menon V., *Functional connectivity in the rest-*

ing brain: A network analysis of the default mode hypothesis. Proceedings of the National Academy of Sciences 100(1), 253–258, 2003.

- [2] Hohne, K.H., Bomans, M., Tiede, U., Riemer, M., *Display of multiple 3D objects using the generalized voxel model.* In Medical Imaging II, Part B, Proc. SPIE 914, pp. 850–854. Newport Beach, 1988.
- [3] Hohne, K.H., Bernstein, R., *Shading 3D-Images from CT using gray-level gradients.* IEEE Transactions on Medical Imaging 5, 45–47, 1986.
- [4] Levoy,M., *Display of surfaces from volume data.* IEEE Computer Graphics and Applications 8(3), 29–37, 1988.
- [5] Preim, B., Bartz, D., *Visualization in Medicine.* Morgan Kaufmann, 2007.
- [6] Sunguroff, A., Greenberg, D., *Computer generated images for medical applications.* In Proceedings of the 5th annual conference on Computer graphics and interactive techniques, SIGGRAPH '78, p. 196–202. ACM, New York, NY, USA , 1978.
- [7] Tuch,D.S., Reese,T.G., Wiegell,M.R., Makris,N., Belliveau, J.W., Wedeen, V.J., *Highangular resolution diffusion imaging reveals intravoxel white matter fiber heterogeneity.* Magnetic Resonance in Medicine 48(4), 577–582, 2002.
- [8] Vannier,M.W.,Marsh,J.L., Warren,J.O., *Three dimensional computer graphics for craniofacial surgical planning and evaluation.* SIGGRAPH Comput. Graph. 17(3), 263–273, 1983.
- [9] Altobelli, D.E., Kikinis, R., Mulliken, J.B., Cline, H., Lorensen, W., Jolesz, F., *Computer assisted three dimensional planning in craniofacial surgery.* Plastic and Reconstructive Surgery 92(4), 576–585; discussion 586–587, 1993.
- [10] Basser, P., *Fiber tractography via diffusion tensor MRI (DT-MRI).* In Proceedings of the 6th Annual Meeting ISMRM, Sydney, Australia, vol. 1226, 1998.

- [11] Basser, P., Mattiello, J., LeBihan, D., *MR diffusion tensor spectroscopy and imaging*. Biophysical Journal 66(1), 259-267, 1994.
- [12] Behrens,U., Teubner, J., Everts, C., Walz,M., Jurgens, H.,Peitgen, H.O., *Computer-assisted dynamic evaluation of contrast enhanced MRI*. In Proceedings of Computer Assisted Radiology, pp. 362-367, 1996.
- [13] Blaas, J., Botha, C.P., Post, F.H., *Interactive visualization of multifield medical data using linked physical and feature space views*. In Proceedings Eurographics / IEEE-VGTC EuroVis, p. 123-130 , 2007.
- [14] Csebfalvi, B., Mroz, L., Hauser, H., Konig, A., Groller, E., *Fast visualization of object contours by Non Photorealistic volume rendering*. Computer Graphics Forum 20(3), p. 452-460, 2001.
- [15] Ebert,D.,Rheingans,P., *Volume illustration : non-photorealistic rendering of volume models*. In Proceedings of the conference on Visualization '00, VIS '00, p. 195-202. IEEE Computer Society Press, Los Alamitos, CA , USA , 2000.
- [16] Hong, L., Kaufman, A., Wei, Y.C., Viswambharan, A., Wax, M., Liang, Z., *3D virtual colonoscopy*. In Biomedical Visualization, 1995. Proceedings., pp. 26-32, 83. IEEE , 1995.
- [17] Interrante, V., Fuchs, H., Pizer, S., *Enhancing transparent skin surfaces with ridge and valley lines*. In IEEE Conference on Visualization, 1995. Visualization '95. Proceedings, pp. 52-59, 438. IEEE, 1995.
- [18] Kindlmann, G., Weinstein, D., *Hue balls and lit tensors for direct volume rendering of diffusion tensor fields*. In Proceedings of the conference on Visualization 99: celebrating ten years, VIS 99, p.183-189. IEEE Computer Society Press, Los Alamitos, CA, USA,1999.
- [19] Kindlmann, G., Weinstein, D., Hart,D., *Strategies for direct volume rendering of diffusion tensor fields*. IEEE Transactions on Visualization and Computer Graphics 6(2), 124-138, 2000.
- [20] Kindlmann,G.,Whitaker,R.,Tasdizen,T.,Moller,T., *Curvature based transfer functions for direct volume rendering: Methods and applications*. In Proceedings of the 14th IEEE Visualization 2003 (VIS'03), VIS 03, p. 67 IEEE Computer Society, Washington, DC, USA, 2003.
- [21] Kruger, J., Westermann, R., *Acceleration techniques for GPU-based volume rendering*. In Visualization Conference, IEEE, p. 38. IEEE Computer Society, Los Alamitos, CA, USA, 2003.
- [22] Preim, B., Bartz, D., *Visualization in Medicine*. Morgan Kaufmann, 2007.
- [23] Steenwijk, M.D., Milles, J., Buchem, M.A., Reiber, J.H., Botha, C.P., *Integrated visual analysis for heterogeneous datasets in cohort studies*. IEEE VisWeek Workshop on Visual Analytics in Health Care, 2010.
- [24] Tietjen,C.,Isenberg,T.,Preim,B., *Combining silhouettes, surface, and volume rendering for surgery education and planning*. IEEE/Eurographics Symposium on Visualization (EuroVis), p. 303-310, 2005.
- [25] Tory,M.,Rober,N.,Moller,T.,Celler,A.,Atkins,M.S., *4D space time techniques : a medical imaging case study*. Proceeding IEEE Visualization 2001, p. 473-476. IEEE Computer Society, Washington, DC, USA, 2001.
- [26] Wang, T.D., Plaisant, C., Quinn, A.J., Stanchak, R., Murphy, S., Schneiderman, B., *Aligning temporal data by sentinel events: discovering patterns in electronic health records*. Proceeding of the twenty sixth annual SIGCHI conference on Human factors in computing systems, CHI '08, p. 457-466. ACM, New York, NY, USA, 2008.
- [27] Weinstein,D.,Kindlmann,G.,Lundberg,E., *Tensorlines: advection diffusion based propagation through diffusion tensor fields*. roceedings of the conference on Visualization '99: celebrating ten years, VIS '99, p. 249-253. IEEE Computer Society Press, Los Alamitos, CA, USA , 1999.
- [28] Herman, G.T., and Liu, H.K., *Optimal Surface Reconstruction from Planar Contours*. Computer Graphics and Image Processing, 1-121, 1979.

- [29] Roberts, J.C., *An Overview of Rendering from Volume Data including Surface and Volume Rendering*. Technical Report 13-93*, University of Kent, Computing Laboratory, Canterbury, UK, 1993.
- [30] Borouchaki, H., Hecht, F., Saltel, E., and George, P.L., *Reasonably Efficient Delaunay-based Mesh Generator in 3 Dimensions*. Proceedings of 4th International Meshing Roundtable, 3-14, 1995.
- [31] Cignoni, P., Montani, C., Puppo, E., and Scopigno, R., *Optimal Isosurface Extraction from Irregular Volume Data*. Proceedings of Volume Visualization Symposium, 31-38, 1996.
- [32] Wyvill G, McPheeers C, Wyvill B *Data structures for soft objects*. Visual Computer 1986;2(4):227–34.
- [33] William E. Lorensen, Harvey E. Cline, *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. SIG ‘87.
- [34] L. Szirmay-Kalos, *Theory of Three Dimensional Computer Graphics*. Akademia Kiado, Budapest, 1995.
- [35] S. S. Trivedi, G. T. Herman, and J. K. Udupa, *Segmentation into three classes using gradients*. In Proceedings IEEE Transactions on Medical Imaging, pages 116–119, June 1986.
- [36] J. K. Udupa, *Interactive segmentation and boundary surface formation for 3D digital images*. In Proceedings Computer Graphics and Image Processing, pages 213–235, March 1982.
- [37] E. Artzy, G. Frieder, and G. T. Herman, *The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm*. In Proceedings Computer Graphics and Image Processing, pages 1–24, January 1981.
- [38] T. Malzbender, *Fourier volume rendering*. ACM Transactions on Graphics, Vol.12, No.3, pages 233–250, 1993.
- [39] L. Lippert and M. H. Gross, *Fast wavelet based volume rendering by accumulation of transparent texture maps*. Computer Graphics Forum, Proceedings EUROGRAPHICS ‘95, pages 431–443, 1995.
- [40] T. Toska and M. Levoy, *Frequency domain volume rendering*. Computer Graphics, Proceedings SIGGRAPH ’93, pages 271–278, 1993. <http://www-graphics.stanford.edu/papers/fvr/>
- [41] Heiden W, Goetze T, Brickmann J, *Fast generation of molecular surfaces from 3D data fields with an enhanced marching cube algorithm*. Journal of Computational Chemistry 1993;14(2):246–50.
- [42] Yim P, Vasbinder G, Ho V, Choyke P., *Iso-surfaces as deformable models for magnetic resonance angiography*. IEEE Transactions on Medical Imaging 2003;22(7):875–81.
- [43] Lin F, Seah HS, Lee YT, *Deformable volumetric model and isosurface: exploring a new approach for surface boundary construction*. Computers and Graphics 1996;20(1):33–40.
- [44] Ferley E, Cani M-P, Gascuel J-D, *Practical volumetric sculpting*. Visual Computer 2000;16(8):469–80.
- [45] Stein R, Shih A, Baker M, Cerco C, Noel M, *Scientific visualization of water quality in the chesapeake bay*. Proceedings of visualization ’00, Salt Lake City, 2000. p. 509–12.
- [46] Matsuda H, Cingoski V, Kaneda K, Yamashita H, Takehara J, Tatewaki I, *Extraction and visualization of semitransparent isosurfaces for 3D finite element analysis*. IEEE Transactions on Magnetics 1999;35(3):1365–74.
- [47] Trembilski A, *Two methods for cloud visualization from weather simulation data*. Visual Computer 2001;17:179–84.
- [48] Kim K, Wittenbrink C, Pang A, *Data level comparison of surface classification and gradient filters*. Proceedings of joint international workshop on volume graphics 2001, Stony Brook, New York, 2001. p. 249–63.
- [49] Savarese S, Rushmeier H, Bernardini F, Perona P, *Shadow carving*. Proceedings of the eighth international conference on computer vision, Vancouver, 2001. p. (I)190–7.

- [50] Friskin S, Perry R, Rockwood A, Jones T, *Adaptively sampled distance fields: a general representation of shape for computer graphics*. Proceedings of SIGGRAPH 2000, New Orleans, 2000. p. 249–54.
- [51] Dedual, N., Kaeli, D., Johnson, B., Chen, G., Wolfgang, J., *Visualization of 4D Computed Tomography Datasets*. Proceeding 2006 IEEE Southwest Symposium of Image Analysis and Interpretation, March 2006.
- [52] Meng-fei Li; Ruo-feng Tong, *CASS: A computer aided surgical system based on 3D medical images*. Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on, On page(s): 1 - 5, Volume: Issue: , 21-23 June 2011.
- [53] K.S. Delibasis, G.K. Matsopoulos, N.A. Mouravliansky, K.S. Nikita, *A novel and efficient implementation of the marching cubes algorithm*. Computerized Medical Imaging and Graphics 25 (2001) 343-352.
- [54] R. Shekhar, E. Fayad, R. Yagel, and F. Cornhill, *Octree-based decimation of marching cubes surfaces*. In Proceedings of IEEE Visualization '96, pages 335–342, 1996.
- [55] Oppenheim, A.V., and Schafer, R.W., *Digital Signal Processing*. London, Prentice Hall International Inc., 1975.
- [56] Lim, J.S., *Two Dimensional Signal and Image Processing*. London, Prentice Hall International Inc., 1990.
- [57] Levoy, M., *Volume Rendering Using Fourier Slice Theorem*. IEEE Computer Graphics and Applications, 3, 202-212, 1992.
- [58] ENTEZARI, A., SCOGGINS, R., MOLLER, T., and MACHIRAJU, R., *Shading for fourier volume rendering*. In Proceedings of Symposium on Volume Visualization and Graphics 02, 131–138, 2002.
- [59] Balazs Csébfalvi, László Szirmay-Kalos., *Monte carlo volume rendering*. In Proceedings of the 14th IEEE Visualization 2003, p.59, October 22-24, 2003.
- [60] COOLEY, J. W., AND TUKEY, J. W., *An algorithm for the machine calculation of complex Fourier series*. Math. Comput. 19, 297–301, 1965.
- [61] FRIGO, M., AND JOHNSON, S. G. , *FFTW: An adaptive software architecture for the FFT*. In Proceedings of ICASSP'98, vol. 3, 1381–1384., 1998.
- [62] BRACEWELL, R. N., BUNEMAN, O., HAO, H., AND VILLASENOR, *Fast two-dimensional Hartley transform*. Proceedings of the IEEE 74, 1282–1282., 1986.
- [63] HAO, H., AND BRACEWELL, R. N., *A three-dimensional DFT algorithm using the fast Hartley transform*. Proceedings of the IEEE 75, 264–266., 1987.
- [64] Viola, Ivan and Kanitsar, Armin and Groller, Meister Eduard, *GPU-based Frequency Domain Volume Rendering*. Proceedings of the 20th Spring Conference on Computer Graphics, p55-64, 2004.
- [65] Ray, H., Pfister, H., Silver, D., and Cook, T.A. , *Ray Casting Architectures for Volume Visualization*. IEEE Transactions on Visualization and Computer Graphics, 5(3), 210-233, 1999.
- [66] H. K. Tuy and L. T. Tuy. , *Direct 2D display of 3D objects*. IEEE Computer Graphics and Applications, Vol.4, No.10, pages 29–33, 1984.
- [67] R. Yagel, D. Cohen, and A. Kaufman. , *Discrete ray tracing*. IEEE Computer Graphics and Applications, Vol.12, No.5, pages 19–28, 1992.
- [68] L. Szirmay-Kalos, *Theory of Three Dimensional Computer Graphics*. Akade mia Kiado, Budapest, 1995.
- [69] Yagel, R., Kaufman, A. and Zhang, Q., *Realistic Volume Imaging*. Proceedings of Visualization'91, San Diego, CA, 226-231, October 1991.
- [70] Kaufman, A., Yagel, R. and Cohen, D., *Intermixing Surface and Volume Rendering*. 3D Imaging in Medicine. Algorithms, Systems, Applications, K. H. Hoehne, H. Fuchs and S. M. Pizer, (eds.), Springer-Verlag, 217-227, 1990.

- [71] Y. W. Tam and W. A. Davis, *Display of 3D medical images*. In Proceedings Graphics Interface, pages 78–86, 1988.
- [72] D. Cohen, A. Kaufman, R. Bakalash, and S. Bergman, *Real time discrete shading*. The Visual Computer, Vol.6, No.1, pages 16–27, 1990.
- [73] J. Bryant and C. Krumvieda, *Display of 3D binary objects: I-shading*. Computers and Graphics, Vol.13, No.4, pages 441–444, 1989.
- [74] L. S. Chen, G. T. Herman, R. A. Reynolds, and J. K. Udupa, *Surface shading in the cuberille environment*. IEEE Computer Graphics and Applications, Vol.5, pages 33–43, 1985.
- [75] G. T. Herman and J. K. Udupa, *Display of three dimensional discrete surfaces*. In Proceedings of the SPIE, Vol.283, pages 90–97, 1981.
- [76] M. Levoy, *Display of surfaces from volume data*. IEEE Computer Graphics and Applications, Vol.8, No.3, pages 29–37, 1988.
- [77] R. E. Webber, *Ray tracing voxel data via biquadratic local surface interpolation*. The Visual Computer, Vol.6, No.1, pages 8–15, 1990.
- [78] R. E. Webber, *Shading voxel data via local curved-surface interpolation*. Volume Visualization, (A. Kaufmann, ed.), IEEE Computer Society Press, pages 229–239, 1991.
- [79] J. Danskin and P. Hanrahan, *Fast algorithms for volume ray tracing*. In Workshop on Volume Visualization 1992, pages 91–98, 1992.
- [80] M. Levoy, *Volume rendering by adaptive refinement*. The Visual Computer, Vol.6, No.1, pages 2–7, 1990.
- [81] N. Max, *Optical models for direct volume rendering*. Journal IEEE Transactions on Visualization and Computer Graphics, Vol.1, No.2, pages 99–108, 1995.
- [82] N. Max, P. Hanrahan, and R. Crawfis, *Area and volume coherence for efficient visualization of 3D scalar functions*. Computer Graphics (San Diego Workshop on Volume Visualization), Vol.24, No.5, pages 27–33, 1990.
- [83] P. Sabella, *A rendering algorithm for visualizing 3D scalar fields*. Computer Graphics (Proceedings SIGGRAPH 1988), Vol.22, No.4, pages 51–58, 1988.
- [84] T. Porter and T. Duff, *Compositing digital images*. Computer Graphics (Proceedings SIGGRAPH 1984), Vol.18, No.3, pages 253–259, 1984.
- [85] Y. Sato, N. Shiraga, S. S. Nakajima, Tamura, and R. Kikinis, *LMIP: Local maximum intensity projection*. Journal of Computer Assisted Tomography, Vol.22, No.6, 1998.
- [86] G. Sakas, M. Grimm, and A. Savopoulos, *Optimized maximum intensity projection (MIP)*. In EUROGRAPHICS Workshop on Rendering Techniques, pages 51–63, 1995.
- [87] G. Sakas, M. Grimm, and A. Savopoulos, *Some techniques for shading machine renderings of solids*. In Proceedings of the April 30–May 2, 1968, spring joint computer conference (AFIPS '68 (Spring)). ACM, New York, NY, USA, 37–45, 1968
- [88] Turner Whitted, *An improved illumination model for shaded display*. Commun. ACM 23, 6 (June 1980), 343–349., 1980.
- [89] J. Wilhelms and Van Gelder A., *A Coherent Projection Approach for Direct Volume Rendering*. In Proceeding SIGGRAPH, 1991.
- [90] L. Westover, *Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, UNC Chapel Hill, 1991.
- [91] L. Westover, *Footprint Evaluation for Volume Rendering*. In Proceeding SIGGRAPH, 1990.
- [92] L. Westover, *Interactive Volume Rendering*. In Chapel Hill Volume Visualization Workshop, 1989.
- [93] C. Upson and M. Keeler, *V-BUFFER: Visible Volume Rendering*. In Proceeding SIGGRAPH, 1988.
- [94] K. Mueller, T. Moller, and R. Crawfis, *Splatting Without the Blur*. In Proceeding IEEE Visualization, 1999.

- [95] K. Mueller and R. Yagel, *Fast Perspective Volume Rendering with Splatting by Using a Ray Driven Approach*. In Proceeding IEEE Visualization, 1996.
- [96] M. W. Jones, *The visualisation of regular three dimensional data*. University of Wales Swansea, UK, July 1995.
- [97] J. Wilhelms and A. Van Gelder, *Octrees for faster isosurface generation*. ACM Transactions on Graphics, 11(3):201–227, July 1992.
- [98] G. Wyvill, C. McPheeers, and B. Wyvill, *Data structures for soft objects*. The Visual Computer, 2:227–234, 1986.
- [99] B. A. Payne and A. W. Toga, *Surface mapping brain function on 3D models*. IEEE Computer Graphics and Applications, 10(5):33–41, September 1990.
- [100] L. Carpenter R. A. Drebin and P. Hanrahan, *Volume rendering*. In Proceedings SIGGRAPH '88 (Atlanta, Georgia, August 1-5, 1988), volume 22(4), pages 65–74. ACM SIGGRAPH, New York, August 1988.
- [101] Meagher, Donald, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Rensselaer Polytechnic Institute (Technical Report IPL-TR-80-111), October, 1980.
- [102] J. Wilhelms and A. Van Gelder, *Octrees for faster isosurface generation*. Computer Graphics, Vol.24, No.5, 1990.
- [103] D. Cohen and Z. Sheffer, *Proximity clouds, an acceleration technique for 3D grid traversal*. The Visual Computer, Vol.11, No.1, pages 27–38, 1994.
- [104] D. Laur and P. Hanrahan, *Hierarchical splatting: A progressive refinement algorithm for volume rendering*. In Computer Graphics ,Proceedings SIGGRAPH '91, pages 285–288, 1991.
- [105] MUELLER, K., MOLLER, T., and CRAWFIS R., *Splatting without the Blur*. In Proceeding Visualization '99, IEEE, 363–370. , 1999.
- [106] SWAN, J. E., MUELLER, K., MOLLER, T., SHAREEF, N., CRAWFIS, R., AND YAGEL, R., *An Anti Aliasing Technique for Splatting*. In Proceeding Visualization '97, IEEE, 197–204., 1997.
- [107] ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M., *Surface Splatting*. In Proceeding SIGGRAPH '01, ACM, 371–378. , 2001.
- [108] COX, D. J. , *Cosmic Voyage: Scientific Visualization for IMAX film*. In SIGGRAPH '96 Visual Proceedings, ACM, 129,147., 1996.
- [109] ALEXA, M., BEHR, J., COHEN OR, D., FLEISHMAN, S., LEVIN, D., AND SILVA, C. T. , *Point Set Surfaces*. In Proceeding Visualization '01, IEEE, 21–28., 2001.
- [110] PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M., *Surfels: Surface Elements as Rendering Primitives*. In Proceedings SIGGRAPH '00, ACM, 335–342., 2000.
- [111] RUSINKIEWICZ, S., AND LEVOY, M. , *QSpplat: A Multiresolution Point Rendering System for Large Meshes*. In Proceedings SIGGRAPH '00, ACM, 343–352. , 2000.
- [112] WAND, M., FISCHER, M., PETER, I., MEYER AUF DER HEIDE, F., AND STRASSER, W., *The Randomized z-Buffer Algorithm*. In Proceedings SIGGRAPH '01, ACM, 361–370., 2001.
- [113] ZWICKER, M., PFISTER, H., VAN BAAR, J., AND GROSS, M. , *Surface Splatting*. In Proceedings SIGGRAPH '01, ACM, 371–378., 2001.
- [114] BOTSCHE, M., WIRATANAYA, A., AND KOBBELT, L. , *Efficient High Quality Rendering of Point Sampled Geometry*. In Proceeding Eurographics Workshop on Rendering '02, EG., 2002.
- [115] PAULY, M., GROSS, M., AND KOBBELT, L. , *Efficient Simplification of Point Sampled Surfaces*. In Proceedings Visualization '02, IEEE, 163–170., 2002.

- [116] L. Sobierajski, D. Cohen, A. Kaufman, R. Yagel, and D. E. Acker., *A fast display method for volumetric data*. The Visual Computer, Vol.10, No.2, pages 116–124, 1993.
- [117] T. Saito., *Real time previewing for volume visualization*. In Proceedings of Symposium on Volume Visualization '94, pages 99–106, 1994.
- [118] P. Lacroute, *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Ph.D. dissertation, Technical Report CSL-TR-95-678, Stanford University, 1995.
- [119] P. Shirley and A. Tuchman, *A polygonal approximation to direct scalar volume rendering*. Computer Graphics, vol. 24, no. 5, pp. 63-70, (San Diego Workshop on Volume Rendering), 1990.
- [120] B. Cabral, N. Cam, and J. Foran, *Accelerated volume rendering and tomographic reconstruction using texture mapping hardware*. Symposium on Volume Visualization '94, pp. 91-98, 1994.
- [121] K. Engel, M. Kraus, and T. Ertl, *High Quality PreIntegrated Volume Rendering Using Hardware Accelerated Pixel Shading*. Proceeding SIGGRAPH Graphics Hardware Workshop '01, pp. 9-16, 2001.
- [122] J. van Scheltinga, J. Smit, and M. Bosma, *Design of an on chip reflectance map*. Proceeding Eurographics Workshop on Graphics Hardware '95, pp. 51-55, 1995.
- [123] M. Meissner, U. Kanus, and W. Strasser, *VIZARD II: A PCICard for Real-Time Volume Rendering*. Proceeding Siggraph/Eurographics Workshop on Graphics Hardware '98, pp. 61–67, 1998.
- [124] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, *The VolumePro realtime raycasting system*. Proceeding SIGGRAPH '99, p. 251-260, Los Angeles, CA, August 1999.
- [125] Hsiao, E.M., Rybicki, F.J., Steigner, M., *CT coronary angiography: 256-slice and 320-detector row scanners*. Current cardiology reports 12(1), 68–75 , 2010.
- [126] Hagmann, P., Jonasson, L., Maeder, P., Thiran, J.P., Wedeen, V.J., Meuli, R., *Understanding diffusion MR imaging techniques: from scalar diffusion-weighted imaging to diffusion tensor imaging and beyond*. Radiographics: A Review Publication of the Radiological Society of North America, Inc 26 Suppl 1, S205–223, 2006.
- [127] Tuch, D.S.,Reese, T.G.,Wiegell, M.R.,Makris, N., Belliveau,J.W., Wedeen,V.J., *Highangular resolution diffusion imaging reveals intravoxel white matter fiber heterogeneity*. Magnetic Resonance in Medicine 48(4), 577–582, 2002.
- [128] Kok, P., Dijkstra, J., Botha, C.P., Post, F.H., Kaijzel, E., Que, I., Lowik, C., Reiber, J., Lelieveldt, B.P.F., *Integrated visualization of multi angle bioluminescence imaging and micro CT*. In Proceedings SPIE Medical Imaging 2007, vol. 6509, 2007.
- [129] Kok, P., Baiker, M., Hendriks, E.A., Post, F.H., Dijkstra, J., Lowik, C.W., Lelieveldt, B.P., Botha, C.P. , *Articulated planar reformation for change visualization in small animal imaging*. IEEE Transactions on Visualization and Computer Graphics 16(6), 1396–1404, 2010.
- [130] Steenwijk, M.D., Milles, J., Buchem, M.A., Reiber, J.H., Botha, C.P. , *Integrated visual analysis for heterogeneous datasets in cohort studies*. IEEE VisWeek Workshop on Visual Analytics in Health Care, 2010.
- [131] de Leeuw, F.E., de Groot, J.C., Achten, E., Oudkerk, M., Ramos, L.M.P., Heijboer, R., Hofman, A., Jolles, J., van Gijn, J., Breteler, M.M.B., *Prevalence of cerebral white matter lesions in elderly people: a population based magnetic resonance imaging study. the rotterdam scan study*. Journal of Neurology, Neurosurgery and Psychiatry 70(1), 9-14, 2001.
- [132] John, U., Hensel,E., Ludemann,J., Piek, M.,Sauer, S.,Adam,C., Born,G., Alte,D., Greiser, E., Haertel, U., Hense, H.W., Haerting, J., Willich, S., Kessler, C., *Study of health in pomerania (SHIP): a health examination survey in an east german region: Objectives and design*.

Social and Preventive Medicine 46, 186–194,
2001.