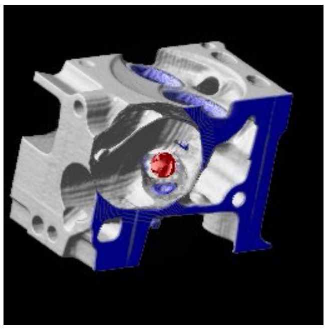
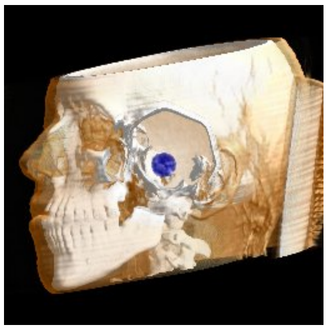
# Implementation of Importance-Driven Volume Rendering

Bernhard Pflugfelder
0027467

Christopher Thurnher
0125913

Institute of Computer Graphics and Algorithms
Vienna University of Technology





## Abstract

Common volume visualization methods mostly guarantee a high quality rendering of volume data with integrated object specific per-voxel based rendering. Especially, in medical or engineering data not only the representation of surfaces of visible volume objects are of particular interest, but also possible volume objects which are actually hidden by other objects in the view direction could contain important information. Thus, volume object occlusion becomes very important in volume visualization applications.

This report presents an efficient implementation using *(Importance-Driven Volume Rendering - IDVR)* to render all volume objects along the view direction in relation to the associated object importance values. Due to the higher importance of specific volume objects particular parts of objects with less importance will be cut or suppressed in their transparency if they are in front of more important object. Thus, the user is able to clearly view those important but hidden objects, regardless which view direction has been chosen.

*IDVR* actually is an addition to the common rendering pipeline and thus can be efficiently combined with other known volume visualization methods, in particular with realistic and non-photorealistic shading methods. Considering highly complex volume data sets containing various different volume objects, the *IDVR* enhancement of our application will introduce an easy solution to take care of entirely visualize individually defined volume objects despite of any occlusion and from which view direction they are looked at.

**Keywords:** Direct Volume Rendering, Two-Level Volume Rendering, Importance-Driven Volume Rendering, Contour Rendering, Tone Rendering

## 1 Introduction

To directly convey the informational contents of volumetric scalar fields various volume visualization methods have been developed in the last couple of years. These visualization methods are able to include different attributes of the volumetric data into the rendering process based upon the specific informational context the user might need. Since the advanced segmentation of basic volume data sets into smaller data entities which we want to call volume objects, separately rendering of the volumetric data can be achieved. This differently rendering of separate volume objects can be realized by the implementation of *Two-Level Volume Rendering (2lVR)* introduced in [10]. Based on these new possibilities to represent volumetric data parts with its own actual rendering attributes, the challenge of our volume rendering application is to specifically combine the data of those volume objects related to the current view direction and a predefined importance hierarchy of included objects which means the use of *Importance-Driven Volume Rendering (IDVR)* [9]. Depending on the user selection, any volume object of the data set can be represented in the final graphical results despite the fact that those objects may be partly or fully occluded in relation to the current view direction. Thus, the display of the critical information will be far more enhanced and various extended uses of IDVR based applications are clearly cogitable.

Main application areas of modern volume visualization have always the challenge to represent highly dense and complex volumetric scalar fields in a way that users can easily extract important information and quickly understand the relationship of every single image part to compose a correct general view of the visualized problem situation. Especially, medical or engineering data fields usually include many separate data entities like for example different tissue layers, various organs, bones or in relation to engineering data different engine parts. Due to the specific alignment of those volume objects within the data set, it is often not guaranteed that all currently critical data will be fully displayed to the user, particularly if the view direction would be changed by the user. For example a medical data set of a particular human abdomen created out of the slice images of Computer Tomography, one of the main attributes of such a data set is the high complexity regarding to a respectably large number of included separate volume objects. Basically, the problem we try to solve with our implementation is that some of these objects are at least partly occluded by other objects in relation to a specific view direction. Imagine the exemplary medical situation of a patient having possible colon cancer. Normally, the attending physician tries to figure out whether that cancer actually exists and at which state the caner already is. Due to the occlusion of parts of the colon, representing the important parts of that colon can be difficult by using ordinary volume visualization tools. Thus, the visibility of the critical area could be hardly affected and complicates the correct medical diagnosis. Due to the use of *IDVR*, the entire colon area can be correctly displayed independently to the chosen view direction and the general view of the current pathological situation of the patient will be provided, i.e. particular size of the tumor, location, near vital organs an possible ways of removal of the tumor.

By considering this small and simplified example it is particularly obvious to recent medical visualization applications but also in other application areas of volume visualization that an importance based volume rendering process has to be included in standard visualization methods to guarantee the correct representation of critical data to the user.

Current volume visualization methods are definitely able to achieve high quality images regarding different rendering attributes, rendering methods for entire volumetric data fields or single volume objects and user predefined properties like for example the view direction and light intensity. Rendering of single volume data objects can be only achieved in a step by step process considering the particular sequence of the volume objects along the current view direction. Based on front-to-back or back-to-front compositing of the precalculated sample points along the current ray, these color/opactiy values will be combined regardless which actual importance each volume object has been assigned by the data classification. Thus, volume objects which are represented by a high importance value but are actually hidden by other objects along the view direction can not be certainly visualized entirely to provide an satisfying graphical representation of the those hidden volume objects. Even through explicit changing of the opacity of volume objects lying in front of the interesting data entity, they will still have a visible influence on rendering results and will affect the quality of the information representation. Thus, the changing of object's opacity to visualize hidden data objects is not very effective, particularly, if the view direction is not to be meant as a constant property of the rendering process but can be altered anytime.

To integrate an advanced rendering method that allows the user to clearly view specific volume objects entirely and independent of the currently chosen view direction despite of any possible occlusion, our introduced application uses *Importance-Driven Volume Rendering (IDVR)* of [9]. *IDVR* is embedded into *Direct volume rendering (DVR)* and can be easily combined with standard rendering methods of volume visualization, i.e. *LMIP, Contour, Tone or Direct volume rendering using Phong illumination model* which are also implemented into our application. *DVR* is the basic volume visualization method which bases on the Raycasting algorithm and calculates all needed rendering values at specified sample point locations within the volume data set.

However, the use of *IDVR* enhancement to correctly visualize hidden volume objects additionally needs the specific importance value for each voxel of the volume data set. Thus, each voxel consists at least of two basic scalar values before the actual classification has been applied. These two values actually represent the density distribution and the importance hierarchy of the volume data set.

Regarding the enhancement of *IDVR* the next modification is the calculation of different levels of sparseness for each included volume data object. These levels of sparseness indicate various different transparency representation of a single volume object, for example from a very sparse one to the densest representation of a specific volume object. Thus, those levels of sparseness can be used to modify the opacity values of current sample points along the view direction depending on their actual importance status of the associated volume object. Afterwards, the synthesis of the resulting image will be accomplished based on those modified opacity values and already classificated color values. Since full transparency can be also assigned to specific volume objects (define low level of sparseness), it is possible to clip entire parts of the data set which may lay in front of some more important volume objects. However, two additional steps have to be integrated into in the standard rendering process to guarantee correct classification of importance of sample points and also automatic selection of sparseness representation which is called Importance Compositing. The classification of importance values will be done by using *Nearest Neighbor* interpolation based on the nearest voxel neighborhood and a single *footprint* (see Section 4) for each volume object has to be additionally calculated during the classification step. The compositing step selects the currently needed sparseness level of the volume objects along the view direction and assures the particular compositing of more important objects. To achieve this critical compositing of *IDVR* rendering various methods are introduced [9]. We only concentrate on the *Maximum Importance Projection (MImP)* compositing method which also has been implemented in our application.

This report describes an implemented *IDVR* application which is able to render arbitrary volume data sets in respect to the importance hierarchy of the included volume objects. For detailed description of the defined input file format of volume data sets please see the *API documentation* of our application [13]. The specific importance values of every single object have to be selected by predefined specific mask data files (file format also described in the *API-Documentation*) and will be directly applied to the actual voxels. Since the correct functionality of *IDVR* rendering depends on different handling of each single volume object, the application also includes Two-Level-Rendering which is utterly deciding to realize separate object rendering and, therefore, *IDVR* based rendering. Furthermore, the user has the choice of four different rendering methods, i.e. realistic and non-photorealistic shading models, and various rendering properties like view and light direction, early ray determination, gradient thresholds, sampling distance and more.

The remainder of this report is organized as follows. Section 2 shortly refers to related scientific work on which our application is based. Section 3 describes all implemented basic rendering methods which in particular are the implemented realistic and non-photorealistic rendering models including *Two-Level Volume Rendering*. Detailed explanation of the functionality of those listed rendering methods is also part of this section. Section 4 explains the *IDVR* functionality in detail and how the standard rendering pipeline has been enhanced to fulfill the key requirement of important based volume rendering. Section 5 introduces some implementation features of our application like the class structure of the rendering pipeline and also some additional constraints of the rendering process are described here. Furthermore, Section 6 presents various resulting images of *IDVR* rendering and discusses the quality of those results. Finally, Section 7 summarizes this report and comments future chances of *IDVR* integrated rendering applications in practice. The following Figure 1 indicates the basic difference between normal *DVR* with *Phong Illumination* and *IDVR* enhancement which obviously is the integrated cut-out of specific image parts.

## 2 Related Work

Our implemented application is based upon several sources of scientific work which provide theoretical models and explicit methods to realize our self defined rendering requirements. With respect to that importantly related scientific work we will give an overview to several basic scientific works in this section. First, we discuss methods and further enhancements of *Feature Classification* which are essential for high quality data classification. Besides, those methods are all viewpoint-independent and directly eligible by the user to ensure a high interactively adjustable rendering process. The second group of used methods is related to several compositing models, whereas volume object based rendering is of particular interest.

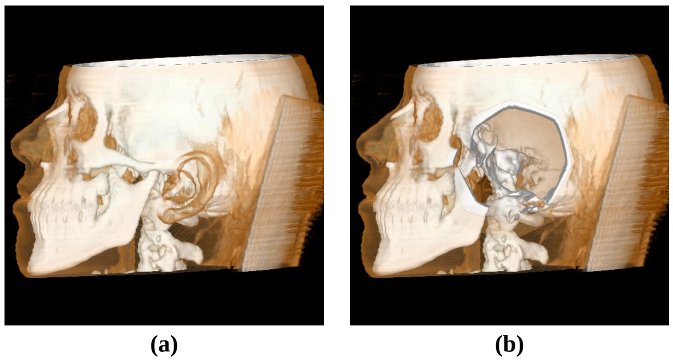


Figure 1: Illustrates the different results of using normal *DVR* or *IDVR* enhancement. Image (a) displays a human head viewed from the left side which was not rendered with *IDVR* and image (b) was rendered by *IDVR* enhancement of internal features.

Particularly, *Two-Level Volume Rendering* becomes a key method to realize the separate rendering of single volume objects. Finally, several shading models which have been included into our application will be also referred and important specific works in that area will be mentioned.

**Feature Classification** is mostly related to the transfer function specification which doubtlessly is a center part of the classification routine but surely is not the only key part of a high quality data classification which we tried to implement in our application. Typically, transfer function specification represents the mapping of basic density values of data samples to optical values like opacity and color. This function class is also called one dimensional transfer function because only the density value will be used as the input value of the specific function. Transfer functions can be designed in various way to represent the optical data classification of the entire data set or just for a single volume object, also it can include manually selected thresholds or boundaries which restrict the valid input value range of the specific transfer function. Additionally, various shading or compositing models, i.e. *MIP, Phong Illumination, Contour and Tone*, or *cut-out* calculating methods need information about the gradient distribution within the volume data set. Thus, gradient vectors for each single voxel of the volume data set have to be estimated by a part of the classification routine (see Section 5.4). Among several possible approximation methods to calculate gradient values of a volume data set, two methods based either on Central Differences in the neighboring gradient distribution (introduced in [5]) or Linear 4D Regression by calculating a hyper plane corresponding to the actual data sample have to be mentioned. This high quality gradient estimation method is introduced in [7]. Actually, the choice of the interpolation method is also directly connected to the quality of the resulting images, thus the standard *Trilinear Interpolation* is an important feature of our classification routine although the computational effort substantially increases. Besides the required transfer function based classification and gradient estimation, importance interpolation and *footprint* calculation (both in [9]) have to be also integrated into the actual classification to provide *IDVR* capability. The *footprint* of a particular volume object denotes the projection of a specific volume object to the image plane which is the base data to perform the correct *cut-out* (also in [9])) during *IDVR compositing*. See Section 4.1 for more information.

**Compositing models** produces the final optical values out of various sample point values along a specific ray during Raycasting. Especially, the segmentation of separate volume objects and the inclusion of importance value based opacity modulation (i.e. *MImP* of [9]) has direct influence to the final compositing method. Basically, Front-To-Back or Back-To-Front compositing methods as introduced in [4] describe the linear combination of the optical attributes opacity and color of all sample points along a cast ray. To take opacity modulation and therefore importance based compositing into account, those standard compositing methods have to be modified. The first important modification is *Two-Level Volume Rendering (2lVR)* [10] which founded a substantial graphical visualization field with comprehensive capabilities in detailed graphical representation of volume data. Methods of this field include volume object based rendering which basically denotes to the separate rendering of sample points based on the associated volume object rendering method. The membership of sample points to a specific volume object will be defined by an object identity number which every voxel needs. Thus, objects with different transfer functions or different shading models can be easily integrated into the rendering process to produce the final image. The second modification of our compositing method is the influence of the importance value based opacity modulation due to the classification process. To efficiently realize this, the addition of levels of sparseness is introduced to represent various transparency representation of a single data object, i.e. from full transparency to absolute opaqueness. However, to select the correct sparseness level of each single volume object along a specific ray during Raycasting, the compositing process has to be enhanced in term of differing between the actual importance of included volume objects. Several importance based compositing methods are introduced in [9] and in particular *Maximum Importance Projection (MImP)* introduced in [9] represents an easy way to clip unimportant parts of the volume data. Other compositing methods of [9] are more complex in term of implementation but have the main advantage to better control level of sparseness selection of each volume object. Thus, not only complete clipping of unimportant object parts is possible but also some kind of reduced opacity representation can be applied.

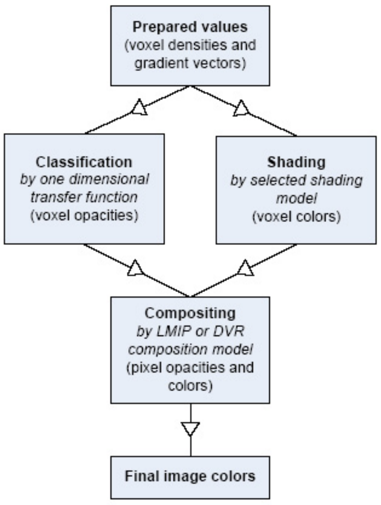
**Shading methods** calculate the final optical values of a sample before this sample can be actually synthesized with other samples to the final image. Various shading models exist in the graphical world and most of those methods have even not been designed in a visualization context, i.e. *Phong Illumination* with specular light, tone shading or more. However, to convey specific informational contents of volume data sets depended on the requirements of single users, different shading model have to be embedded into the rendering pipeline (see Section 5.2). *Phong Illumination* ([5]) guarantees higher realistic images, whereas *non-photorealistic* rendering methods, i.e. *Tone Rendering* [3] and *Contour Rendering* [2], are able to visualize important information in an abstract but sometimes more coherent way.

## 3 Volume Visualization Methods

In this section, there will be a short summary of the Direct Volume Rendering algorithm with Raycasting (see Section 3.1). Then, standard volume visualization methods will be described, like *Maximum Intensity Projection (MIP)* (see Section 3.1.1), *Direct Volume Rendering with Phong Illumination* (see Section 3.1.2), *Tone Rendering* (see Section 3.1.3) and *Contour Rendering* (see Section 3.1.4). Finally, the Two-Level Rendering will be described in Section 3.2.

### 3.1 Rendering Methods

Rendering huge volume data sets is a common issue in today's computer science. There are various approaches to this, one is a Raycasting algorithm also called *Direct Volume Rendering (DVR)*. One main advantage of this approach is that you can render the data set directly from the volume data without having to fit it to geometric primitives like polygons, which can be done e.g. with the *Marching Cubes* algorithm [6]. There are a lot of variations of *Direct Volume Rendering*, but they all rely on the same algorithm (see Figure 2) [4]. The acquired volume data set from e.g. a computer tomograph (CT) or from magnetic resonance imaging (MRI) consist of a 3-dimensional scalar field containing 1-dimensional density values, also called voxels. In some applications it is necessary to prepare the acquired data values first, e.g. the correction of nonorthogonal sampling grids in electron density maps. After that, the density values are classified and shaded. The classification step assigns to each voxel a certain opacity value, e.g. according to a transfer function. Now, every voxel has a density and opacity value. The shading process assigns to each voxel a color value, normally a RGB value. This assigned voxel color value consists of a basic color part which refers to the material color of the specific voxel and a shaded color part which depends on the actual shading model. The basic color is also defined by the transfer function and thus every density value corresponds to a specifically defined color value. Actually, shading is mostly the core of each direct volume rendering method. After classification and shading, each voxel has a density, opacity and color value. Some rendering methods also need a gradient for each voxel. This gradient, which can be considered as an estimated normal vector of a voxel, can be calculated with various methods. Central differences [4] and 4D linear regression [7], are two methods, which will be described in Section 5.



Prepared values
(voxel densities and gradient vectors)
Classification
by one dimensional transfer function
(voxel opacities)
Shading
by selected shading model
(voxel colors)
Compositing
by LMIP or DVR composition model
(pixel opacities and colors)
Final image colors

Figure 2: Illustrated the implemented *Direct Volume Rendering (DVR)* Pipeline. Note that all these rendering steps are actually embedded into a Raycasting method.

Now, the actual Raycasting algorithm can start (see Figure 2). Rays are cast into the scene from the observer's eye point. Each ray is then sampled with a certain sampling distance (e.g. 70 percent of the distance of two adjacent voxels) creating an array of ray samples along it. Of course, it is seldom that the ray samples have exactly the discrete voxel coordinates. Therefore, the values of the ray samples have to be interpolated. There are also various methods to do this, two of them will be described in Section 5, They are called nearest neighbor and trilinear interpolation. This interpolation can be done to the already calculated color and opacity values of the voxels surrounding the ray sample. A more accurate way is to interpolate only the density value of the voxel and then do the classification and shading step for each ray sample. After that, each ray has an array of ray samples whereas each ray sample along the ray has a certain color and opacity value. The third step of the algorithm, after classification and shading, is the compositing. The compositing calculates the final color value for the rendered image. It does this by blending the color and opacity values of all ray samples with a certain calculation rule in back-to-front or front-to-bach order onto the image plane. The resulting color values are the color values for the rendered image. There are of course again various methods of doing this compositing step, which will be described later on. As can be seen, the direct volume rendering algorithm with Raycasting consists of three steps: Classification - shading - compositing. Different methods of all three steps will be discussed in the following subsections.

There are also some acceleration techniques, that can make the Raycasting procedure a lot faster. Various thresholds can be added to the rendering pipeline. One is an opacity threshold, to stop the compositing of a certain ray if the specified threshold is reached. This is obvious, because if the opacity reaches a very big value, the ray samples that lie beyond will not really make any difference to the final image. Another possibility is to insert a gradient magnitude threshold into the pipeline. Voxels or ray samples with a very small gradient are normally those that do not lie on the surface of an object. If only the surface of an object is of interest, this threshold will make sense. All voxels or ray samples which have a gradient with magnitude smaller than the threshold can be discarded. An advantage of this approach is that the gradient estimation an interpolation is normally a very expensive operation.

### 3.1.1 Maximum Intensity Projection (MIP)

One simple and fast visualization method is called *Maximum Intensity Projection (MIP)*. It provides a very fast algorithm because there are not very many calculations to be done by the ray caster in order to produce the final image.

For the classification and shading step there is only a linear mapping of the density values in order to get the opacity values. That means, that small density values result in small opacity (intensity) values and big density values result in big opacity (intensity) values. his can also be seen as a linear transfer function. This linear mapping is very fast, because there are hardly any calculations to be done.

The compositing step is the actual reason why this method is called ”Maximum Intensity Projection”. The final color of a ray is just the maximum intensity value along it. All other values a simply discarded. One disadvantage is, that any spatial relationship in the scene is lost, because there is only one value along a ray that is responsible to the final image. To prevent this, there is also a compositing method called *Local Maximum Intensity Projection* or *LMIP* (see Figure 3).

With *LMIP*, it is necessary to have an intensity threshold. When the compositing is done in front-to-back order, the color value of the resulting image is the first intensity value along one ray that lies over the specified intensity threshold. All ray samples, that lie after the chosen values can be discarded, that means, that this compositing method is also an acceleration to the normal maximum intensity projection. Because of the consistence of many volume data sets, there will also remain a visible spatial relationship in the final image. The local maximum of a ray will often be the first hit of the ray with the surface of an object (see Figure 4).

### 3.1.2 Direct Volume Rendering with Phong Illumination

Based on the standard rendering pipeline introduced by [4] the combination of *Direct Volume Rendering (DVR)* and *Phong Illumination* produces the most photo realistic image quality of all our implemented visualization methods. Basically, *Phong Illumination*

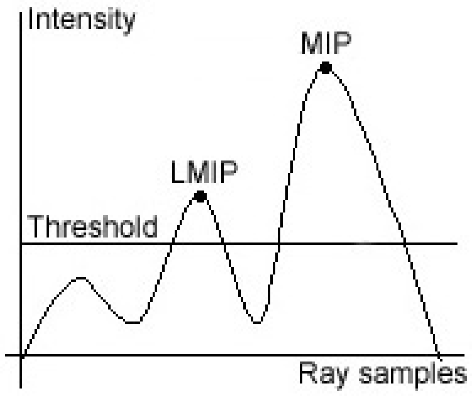


Figure 3: Local Maximum Intensity Projection (LMIP).

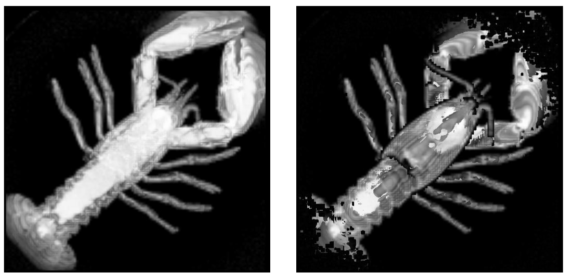


Figure 4: Displays the Lobster volume data set with two different *LMIP* compositing settings. Image (a) was synthesized by the standard *MIP* compositing, whereas the second image (b) was rendered by *LMIP* compositing with threshold 0.

includes the calculation of ambient, diffuse and specular light intensities at a specific sample point location in relation to the current light and view direction. Due to this light calculation the synthesized results obtain amongst others good depth effect and exceptional plastic appearance (see Figure 5). On the other hand *Phong Illumination* is very expensive in term of time and calculation consumption because not only gradient precalculation has to be done for every single sample point but also the costly vector based light intensity calculation has to be implemented.

Our realized method of *Direct Volume Rendering (DVR)* with *Phong Illumination* consists of three main rendering steps whereas the first, sample classification, and third, sample compositing, step are actually included in the standard rendering pipeline (see Figure 2) and will be used for the other introduced shading models (LMIP, Contour and Tone) as well. Moreover, these two main steps will also be modified to gain *2lVR* and *IDVR* enchantments. The Classification step is based on one dimensional transfer functions for opacity and color and will be also adapted to calculate volume object identity and importance value for each sample point. For more detail about the classification step see 5.2.

Basically, the compositing step is based on linear combination of sample points along a specific ray using the ”sample point blending” method of [4]. Thus, this method is embedded in a standard Raycasting algorithm and along each cast ray sample points will be calculated and finally composed to resulting opacity and color values. Those optical values directly represent that pixel color on the image plane through which the corresponding ray has been cast.

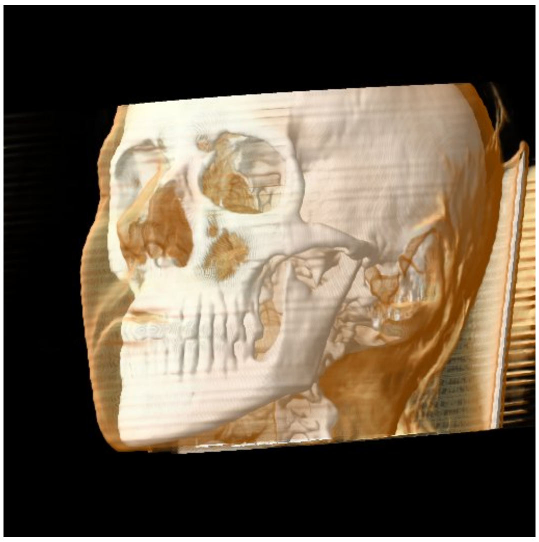


Figure 5: Illustrates a synthesized image with *Phong Illumination* based on a human head volume data set.

The sample points are located evenly spaced (distance defined by *Sampling Distance*) along the cast ray and all needed classification values like density, gradient, object identity and importance value have to be directly interpolated based on the corresponding values of the neighboring voxels. The actual compositing of the precalculated sample points values (opacity and color components (RGB)) has to be done for each cast ray and blends the values in *Front-To-Back (FTB)* order which means that linear combination of the opacity and the single color components will be executed exactly along the view direction from the front side of the volume set to the back side. The formula for the intensity blending calculation for the ray intensity $C_{out\lambda}(u_i)$ of ray $u_i$ (see Equation 1) is actually a simple linear interpolation of each single sample point intensity $C_\lambda(x_i)$ and opacity $\alpha(x_i)$ respectively at the location $x_i$ along the corresponding cast ray. Notice that $C_{in\lambda}(u_i)$ and $\alpha_{in}(u_i)$ corresponds to the current intensity and opacity values of the ray $u_i$ which both are based on the interpolation of the previous sample points. Additionally, the current ray opacity $\alpha_{out}(u_i)$ of ray $u_i$ has to be also modified by using the Equation 2 where $\alpha_{in}(u_i)$ describes the current ray opacity value (based on previous sample points). Thus, the interpolation calculation has to executed for each sample point along the ray in *FTB* order:

$$C_{out\lambda}(u_i) = C_{in\lambda}(u_i) + (1-\alpha_{in}(u_i)) * C_\lambda(x_i) * \alpha(x_i) \tag{1}$$

$$\alpha_{out}(u_i) = \alpha_{in}(u_i) + (1-\alpha_{in}(u_i)) * \alpha(x_i) \tag{2}$$

where

| | | |
|---|---|---|
| $C_{out\lambda}(u_i)$ | = | $\lambda$'the component of color at specific ray $u_i$ after blending calculation , $\lambda = r,g,b$. |
| $C_{in\lambda}(u_i)$ | = | $\lambda$'the component of color at specific ray $u_i$ before current blending calculation at sample point location $x_i$. |
| $C_\lambda(x_i)$ | = | $\lambda$'the component of color at sample point location $x_i$. |
| $\alpha_{out}(u_i)$ | = | opacity value at specific ray $u_i$ after blending calculation. |
| $\alpha_{in}(u_i)$ | = | opacity value at specific ray $u_i$ before blending calculation. |
| $\alpha(x_i)$ | = | opacity value at sample point location $x_i$. |

The second step of our rendering pipeline which corresponds to the shading step includes the calculation of final sample color values which refer to the modification of the basic color values of classification by a specific type of shading method. The final opacity value of the sample points will not be altered during the shading step and is identical to the classification opacity value. As you see in the other Sections, various shading models can be integrated into the pipeline and each model produces specific image results which convey a specific informational context. As already mentioned at the beginning, this Section describes the *Phong Illumination Model* which synthesizes more photo realistic images. Basically, we used the standard *Phong Model* which has been modified to the visualization context and also includes *Depth Cueing* to gain better depth appearance. Besides, gradient precalculation of every single sample point is utterly essential for the *Phong Model* and will be done by standard *Central Differences* [4] or *4D Linear Regression* [7]. Both estimation methods approximate the gradient vector of a specific voxel due to the neighboring density differences and the sample point's gradient will be finally interpolated based on the surrounding voxel gradients. *Central Differences* is a fast but low-quality method which use the approximation of the first derivative in $x$, $y$ and $z$ directions to calculate a weighted sum as the corresponding gradient component. Otherwise, *4D Linear Regression* guarantees higher quality in term of low approximation error but the computational effort also increases. See Section 5.4 for detailed description of those estimation methods. The implemented formula is now defined as following:

$$C_\lambda(x_i) = C_{p\lambda} k_{a\lambda} + \frac{C_{p\lambda}}{k_1 + k_2 d(x_i)} [k_{d\lambda}(N(x_i) * L) + k_{s\lambda}(N(x_i) * H)^n] \tag{3}$$

where

| | | |
|---|---|---|
| $C_\lambda(x_i)$ | = | $\lambda$'the component of color at sample point location $x_i$, $\lambda = r, g, b$. |
| $C_{p\lambda}$ | = | $\lambda$'the component of color of directional light source. |
| $k_{a\lambda}$ | = | ambient reflection coefficient for $\lambda$'the color component. |
| $k_{d\lambda}$ | = | diffuse reflection coefficient for $\lambda$'the color component. |
| $k_{s\lambda}$ | = | specular reflection coefficient for $\lambda$'the color component. |
| $n$ | = | exponent used to approximate highlight. |
| $k_1, k_2$ | = | constants used in linear approximation of depth-cueing. |
| $d(x_i)$ | = | perpendicular distance from image plane to sample point location $x_i$. |
| $N(x_i)$ | = | surface normal at sample point location $x_i$. |
| $L$ | = | normalized vector in direction of light source. |
| $H$ | = | normalized vector in direction of maximum highlight. |

Since a directional light source is used, $L$ is a constant normalized vector and $H$ is the approximation of the direction to the maximum highlight

$$H = \frac{V+L}{|V+L|}$$

with

| | | |
|---|---|---|
| $V$ | = | normalized vector in direction of observer. |
| $L$ | = | normalized vector in direction of the light source. |

Figure 6 exemplarily illustrates two images of the known Lobster data set and a human head data set to demonstrate the capability of image quality and, especially, the very good depth and plastic appearance because of the integration of light intensity into the shading calculations.

### 3.1.3 Tone Rendering

As seen in the previous section (see Section 3.1.2), *Direct Volume Rendering with Phong Illumination* can produce a very realistic image of the volume data set using a specified transfer function and a

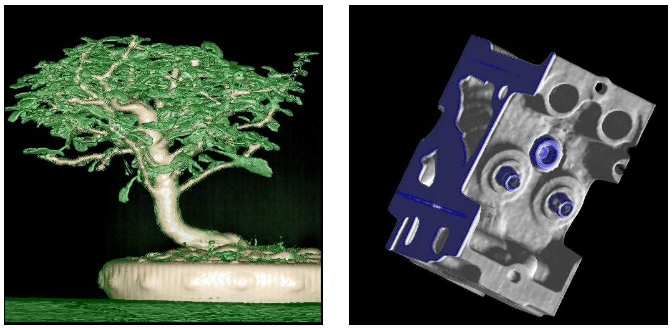


Figure 6: Two images of different volume data set indicate the realistic impression of *Phong Illumination*, whereas image (a) displays a bonsai data set and image (b) is based on a engine volume data set.

global illumination model. Apart from these photorealistic rendering methods there is another group called *Non-photorealistic rendering*. The goal of these methods is not to create a very close copy of nature, but to visualize the volume data in a rather artistic way. It can also be used for technical illustration, as it is needed e.g. in user manuals that involve 3D objects. One of these non-photorealistic rendering approaches is called *Tone Shading* which is introduced in [8] and [3].

Tone shading is often used in technical illustrations, but it is also used by painters. Its goal is to modify the tone of an object based on the orientation of that object relative to the light [8]. There are mainly two kinds of colors that are used to generate the desired effect: a warm color and a cool color. Surfaces facing towards the light will get the warm color, whereas surfaces not facing towards the light will get the cool one. As can be seen, this technique does not create a very realistic image, but as mentioned before, this is not the goal of non-photorealistic rendering techniques. Because of the coloring of the surface according to its relative position to the light, there will still remain a spatial impression of the object. A good choice for the warm color is yellow and for the cool one blue. Of course there can be any other combination of two colors, but blue and yellow produce very good results.

The final surface color of an object is formed by an interpolation between the warm and the cool color based on the signed dot product between the surface normal and the light vector. In volume rendering, the gradient vector is used instead of the normal vector. The illuminated object contribution is calculated using only the positive dot product, becoming zero at orientations orthogonal to the light vector.

The tone contribution for a ray sample is, as mentioned before, interpolated between the warm and the cool color, depending on the angle between the light vector an its gradient vector. This is summarized in Equation 4:

$$I_t = (k_w(1.0+\kappa)/2)C_w + k_c(1-(1.0+\kappa)/2)C_c \tag{4}$$

Whereas $\kappa$ is the dot product of the gradient and the light vector as shown in Equation 5. $k_w$ and $k_c$ is the ratio of warm and cool color contributing to the final result.

$$\kappa = G \cdot L \tag{5}$$

G is the gradient vector, $L$ is the vector in light direction, both must be normalized, so that the dot product between them will have a value between 0 and 1.

The values $C_w$

$$C_w = (R_w, G_w, B_w) \tag{6}$$

and $C_c$

$$C_c = (R_c, G_c, B_c) \tag{7}$$

describe the warm and the cool color components between which has to be interpolated. Here they are presented as RGB color values. $R_w$, $G_w$ and $B_w$ are the warm, $R_c$, $G_c$ and $B_c$ are the cool color components.

Now it should be clear, how the tone contribution of a ray sample can be calculated within the process of volume rendering. There is also the possibility, to combine the tone contribution to already calculated color values as e.g. from *Direct Volume Rendering with Phong Illumination* (see Section 3.1.2). This can be done by mixing the tone contribution with an already calculated color value, as described in Equation 8:

$$c = I_t \alpha + I_r(1-\alpha) \tag{8}$$

Whereas $\alpha$ must be a value between 0 and 1. $I_r$ is the already calculated color of the ray sample with any other rendering method, e.g. *Direct Volume Rendering with Phong Illumination*.

By now, there has only been one light source in the rendering process. Assuming that there can of course be more than one light source, that must also be taken into account. This can be done by summing up Equation 8 over all light sources:

$$c = \sum_{i=1}^{N_L} (I_{t_i} \alpha + I_{r_i}(1-\alpha)) \tag{9}$$

Whereas $N_L$ is the total number of light sources.

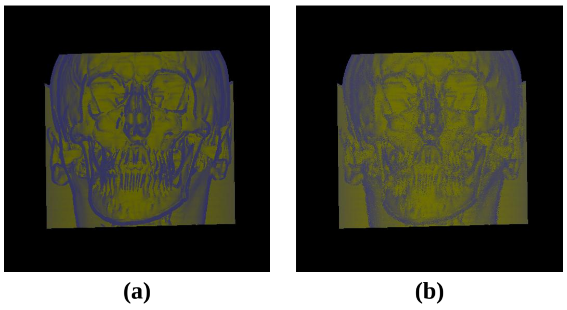


Figure 7: Illustrates two different tone shaded images based on a human head volume data set. Image (a) was rendered with warm color yellow and cool color blue, whereas image (b) was calculated with the sam color properties but different $k_w$ and $k_c$.

Figure 7 shows a tone shaded head with only the tone contribution and no further rendering method. The light direction is straightly ahead of the head. As can be seen, the result of tone rendering is far away of being realistic, but the spatial relationship of the head is still visible. It could also be enhanced by using a global illumination model like *Phong Illumination*. In the figures it should also be clearly visible, that surfaces, i.e. ray samples with a gradient vector oriented towards the light direction become more yellow, whereas ray samples with a gradient vector oriented away from the light direction become rather blue.

Now it should be clear on how the shader works. For the classification step, there can either a linear classification be used like described in Section 3.1.1 or, if you use a transfer function and mix the colors afterwards, the transfer function can also be used for the classification step, as described in Section 3.1.2.

For the compositing, there are also more possibilities. In Figure 7 *LMIP* compositing was used with a very low *LMIP* threshold. But there can also be used a transfer function compositing, like described in Section 3.1.2

### 3.1.4 Contour Rendering

As mentioned in the previous section (see Section 3.1.3), non-photorealistic rendering methods do not provide a close copy of nature, but can also be very useful in the term of volume rendering. Another non-photorealistic method is called *Contour Rendering* [2]. Contour rendering is a technique that does not depend on the data values, but on the gradient magnitude of the voxels and ray samples respectively. Its goal is to produce an image on which only the contour lines of an object are visible. Of course this rendering technique is a view dependent method which means, that the view direction must also be considered.

In the shading process, only those ray samples that have a gradient with a magnitude greater that a specified gradient magnitude threshold will be taken into account for the following rendering calculations. All other ray samples will be discarded. Of course it is beneficial to have a very sophisticated method of gradient estimation. 4D linear regression will produce a much better result as e.g. central differences, as will be described in Section 5. In order to decide, whether the gradient magnitude of any given ray sample satisfies the requirements to be used in the further rendering process, a windowing function is used:

$$w(|g|) = 1 \quad \text{if} \quad |g| > t_g \tag{10}$$

$$w(|g|) = 0 \quad \text{if} \quad |g| < t_g \tag{11}$$

Whereas $|g|$ is the magnitude of the gradient and $t_g$ is the gradient magnitude threshold. This function can be seen in Figure 8.

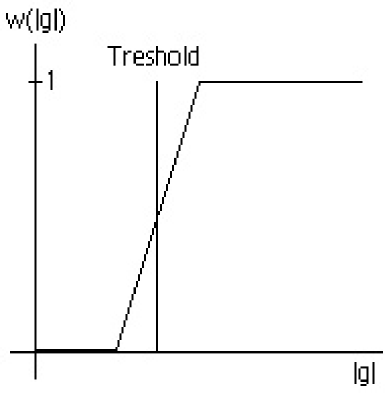


Figure 8: Windowing Function.

One of the advantages of using a windowing function lies in the nature of the gradient magnitude of a ray sample. In contour rendering, the main goal is, as mentioned above, to produce contour lines of an object surface. The bigger the gradient magnitude, the more likely the ray sample with the given gradient lies on an object surface.

As mentioned before, contour rendering is a view depended rendering method, so the view vector must be taken into account. It is obvious that the dot product between the gradient and the view vector is needed:

$$\kappa = V \cdot G \tag{12}$$

Whereas V is the view vector and G is the Gradient vector. If the dot product is 1, the angle between the view and the gradient vector is 0, so this can be used to determine, whether a ray sample lies on a contour line or not. Ray samples on an object contour have an closely 90 degree angle between the gradient and the view vector, and the value of the dot product is nearly 0. In order to achieve values between 0 and 1 for $\kappa$ in Equation 12, the two vectors have to be normalized first

$$s(\kappa) = (1-\kappa)^n \tag{13}$$

Equation 13 will give higher weights to voxels and respectively ray samples that belong to an object contour. The exponent n of Equation 13 controls the thickness of the contour lines. The bigger the exponent, the thinner the contour lines will be.

Now, in order to achieve the final intensity of a ray sample, there has to be a combination of the windowing function (Equation 10) and Equation 13:

$$I_c = w(|g|) \cdot s(\kappa) \tag{14}$$

Now it should be clear how the shading step of the contour renderer works. For classification, the intensity value of Equation 14 can also be used for the opacity of a ray sample, because the opacity of a contour line should be bigger than those that do not belong to an object contour.

As mentioned in Section 3.1, a volume rendering pipeline consists of three steps: classification - shading and compositing. The first two steps have already been explained above, only the compositing is missing by now. For compositing, a simple maximum intensity projection can be used, taking only the maximum intensity value of one ray. In order to maintain a better spatial relationship, there can also be used a (*LMIP*) as described in Section 3.1.1. This avoids that some ray samples that have a higher intensity hide the weaker contours that are closer to the viewer.

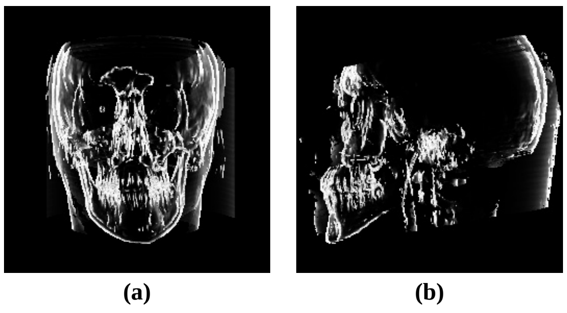


Figure 9: Displays two differently viewed images based on a human head volume data set. Both images (a) and (b) are rendered by *Contour Rendering* with indent contour properties.

Figure 9 shows two different images of a contour rendered human head which are actually viewed from different directions.

Contour rendering produces a very good first impression of the volume data set without having to know the data set a priori. It also works with only a few parameters and so can be used to get a first preview of the whole data set without missing important features. It also saves a time consuming transfer function specification, which is one of the most difficult tasks of volume rendering (see Section 3.1.2), particularly is the underlying volume data set is not known.

## 3.2 Two-Level Rendering

As already mentioned in this report, the standard implemented rendering pipeline is only able to treat all sample points which are directly interpolated based on the corresponding neighboring voxel values in the same manner regarding sample point shading and compositing. This approach relies on object based volume rendering. Particularly, this means that volume object membership of every sample point will not be used to specify and use special shading and compositing properties for each of those included volume objects. Thus, in *Direct Volume Rendering* only an appropriate choice of the transfer function makes visual distinction and informational differentiation of separate volume data objects possible within the volume data set but finding such a transfer function is both very complicated and produced images with still limited quality.

However, to visualize large and complex volume data set which contains high amounts of logically and graphically distinguishable volume objects (i.e. imagine a data set of the human abdomen with organs, muscles, different tissue layers, blood vessels and more) a better and more efficient approach has to be integrated which actually enhanced the already realized rendering pipeline represented by Figure 2. To use a per-voxel based approach for solving this problem like Two-Level Volume Rendering [10] exactly promises our defined requirements and, furthermore we need only to slightly modify most parts of the already implemented rendering pipeline. Thus, the specific object membership of every voxel which will be assigned during the segmentation will be used to distinguish between different object properties. Therefore, shading and compositing of sample points associated to specific single volume objects is possible during Raycasting. However, the necessary modification of segmentation and classification requires a more complex and larger data structure with containing all needed volume objects and also the object identity classification of the sample points. This includes the interpolation of valid object membership identity numbers per sample point by using *Nearest Neighbor* based on the surrounding voxel identity numbers. Each existing volume object has an own and explicit identity number and each associated voxel saves this number additional to the basic density value to refers to the correct volume object. Thus, this identity number works like a look-up table to the current associated volume object and only based on the single voxel all necessary properties regarding corresponding shading and compositing methods can be directly and easily extracted. This interpolation of the object membership identity number has been implemented in our application by using the basic *Nearest Neighbor* interpolation method.

Based on the use of explicit identity numbers corresponding to associated volume objects with individual rendering properties, each sample point can use its associated shading method which means that separate volume objects are able to include different shading methods. This feature increases the graphical quality of the resulting images because of the significantly better differentiation between separate volume objects. Thus, our object based rendering approach is based on *Two-Level Volume Rendering* to meet the discussed requirements. *Two-Level Volume Rendering* contains two main rendering passes, namely local and global rendering (see Figure 10). The local rendering pass includes both shading and compositing of single sample points but always in relation to the actually associated volume objects and their specified shading and compositing properties. Due to the explicit identity number of each volume object, sample point's opacity and color values can be calculated by the local shading method and all sample points associated to the same volume object will be blended during the classification step to a final representation of that specific volume object along the cast ray.

Afterwards, the second and global rendering pass will be applied and directly calculates the final optical values for the currently cast ray which actually represents the final color values for the corresponding pixel of the image plane. In other words, the already calculated object opacity and color values of the local rendering pass will be composited to the final color value by a predefined global compositing method. Figure 10 shows this *Two-Level Volume Rendering* concept with the local and global pass which is implemented in our application. It also points out that only those sample points will be blended together during the local rendering pass which firstly are associated to the same volume object and secondly are located in a direct neighborhood of the ray section. Figure 12 exemplarily indicates the enhanced capabilities of better graphical representation of volume data sets.

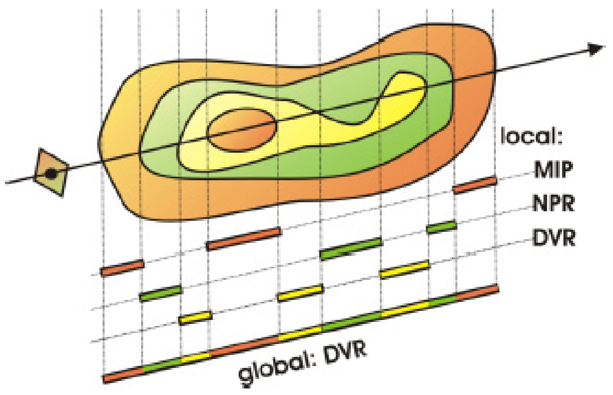


Figure 10: Schematic illustration of both rendering passes during Two-Level Volume Rendering. For the local compositing pass different methods are used in relation to the associated volume objects and, afterwards, global compositing will be applied. This image is courtesy to [11]

Unfortunately, *Two-Level Volume Rendering* of [10] not only requires additional modifications to the standard volume rendering pipeline (introduced in [4]) (see Figure 11) but also increases the amount of computation and, consequently time consumption. Especially, the efficiency will decrease significantly on only software based applications, and thus a GPU based approach will be necessary. To use the *IDVR*, as described in Section 4, this enhanced rendering pipeline is absolutely necessary. Figure 11 illustrates the modificated rendering pipeline which now includes the classification of volume object identity numbers and the two compositing passes.

## 4 Importance-Driven Volume Rendering (IDVR)

In this Section a rendering technique called *Importance-Driven Volume Rendering (IDVR)* [9] will be described. The two-level rendering approach described in Section 3.2 created the possibility to render a given segmented volume data set with a local renderer for each object and to combine these values in a global compositing step afterwards. But there are also some disadvantages this method comprises. In many applications such as medical imaging, there are regions of special interest within a volume data set, i.e. the examination of tumors in the kidneys, lesions inside the liver and findings of lung nodules. With two-level rendering it is possible to render these regions in a special way and so emphasize them. But first of all it is very difficult and time consuming to adjust each local

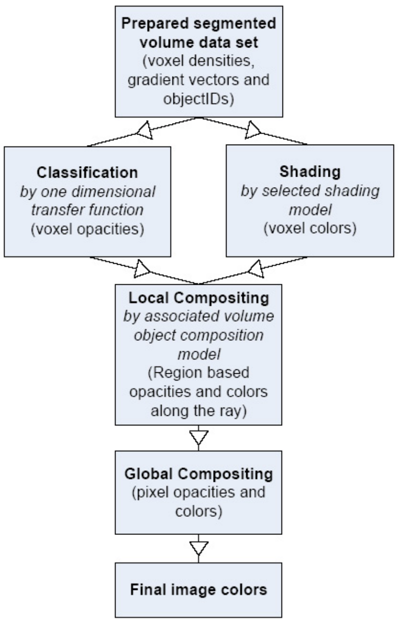

Prepared segmented volume data set
(voxel densities, gradient vectors and objectIDs)
Classification
by one dimensional transfer function
(voxel opacities)
Shading
by selected shading model
(voxel colors)
Local Compositing
by associated volume object composition model
(Region based opacities and colors along the ray)
Global Compositing
(pixel opacities and colors)
Final image colors

Figure 11: Illustrated the implemented *Two-Level Volume Rendering* Pipeline.

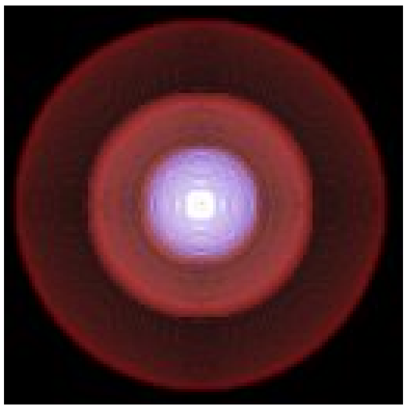


Figure 12: Three concentric spheres with different rendering properties which are actually synthesized by Two-Level Volume Rendering

renderer in a way that the result will be good and secondly, there will always be some regions of less interest that will overlap those with higher. Therefore, *Importance-Driven Volume Rendering* is proposed. It extends the Raycasting pipeline described in Figure 2 to provide a mechanism to create a view depended cut-away view to objects with high importance occluded by objects with low importance (see Figure 13).

The first difference is that there is a segmented data set needed in order to be able to render specified objects with different methods as in two-level rendering. After preparing these values, a third dimension besides opacity and color is added to the voxels in the *IDVR*-Classification step: The importance. The importance reflects the interest the user has to special regions, i.e. objects of the data set. Regions with higher importance should be visible from all view directions, regions that occlude them should be cut away. After that, the normal local rendering process for all ob-

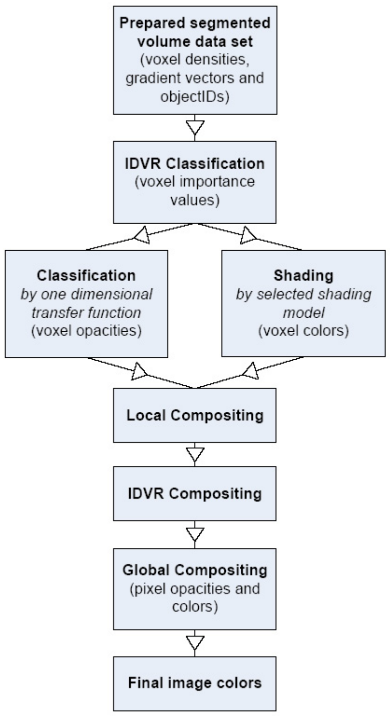

Prepared segmented volume data set (voxel densities, gradient vectors and objectIDs)
IDVR Classification (voxel importance values)
Classification *by one dimensional transfer function* (voxel opacities)
Shading *by selected shading model* (voxel colors)
Local Compositing
IDVR Compositing
Global Compositing (pixel opacities and colors)
Final image colors

Figure 13: Illustrated the implemented *IDVR Rendering* Pipeline.

jects is done, including classification, shading and local compositing. Before the global compositing along the rays, there is another step called *IDVR-Compositing*, which assigns a *Level of Sparseness* to each object,which will be described in Section 4.1, and then modifies the color and opacity values of the local compositing step. After that, the modified color and opacity values are blended in the global compositing step to produce the final image.

### 4.1 IDVR-Compositing

As mentioned above, the *IDVR-Compositing* assigns a sparseness value to each object. This value is different on each ray, dependent on how many and which objects intersect the given ray. Sparseness is loosely defined in terms of how much the display of an object takes up screen estate. A sparseness value 1 will mean that the object will be almost to complete transparent, while a sparseness value of 0 means that the object ´should be complete visible. In order to assign the sparseness value to the objects of a ray, there first of all needs to be created a footprint of the image (see Figure 14).

In Figure 14, there are three objects, while object two is occluded by object one and three. If object two lies behind object one and two and has the highest importance of all three objects, it will get a lower sparseness value than the other two objects, but only in those regions where the occlusion occurs. If there is no occlusion at all, the objects can be rendered in the most dense way. There are of course different possibilities of assigning the Level of sparseness to the objects, one is called ”Maximum Importance Projection” and will be describe in the next section.

According to Figure 14 it should be obvious, that the footprint of the volume is view depended. That means that the footprint has to be calculated each time the view direction changes.

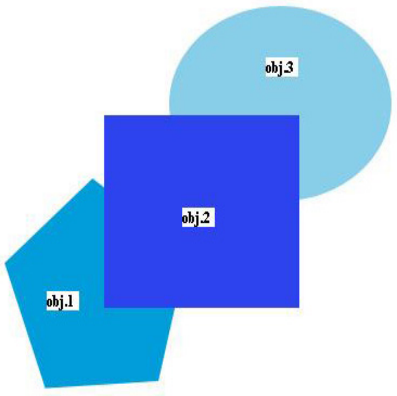


Figure 14: Footprint with three objects.

### 4.2 Maximum Importance Projection (MImP)

*Maximum Importance Projection* is a very simple but also very fast approach. As mentioned above (see Section 4.1), there first needs to be created a footprint of the scene. After that, the level of sparseness values of the objects must be assigned to all rays. *Maximum Importance Projection (MImP)* is very close to *Maximum Intensity Projection* described in Section 3.1.1. Along a ray, the object with the maximum importance value gets the lowest sparseness value, so that it displayed in the most dense way. All other objects get the highest value of sparseness. So, objects are either rendered with the most dense representation, or they are not rendered at all.

If *Maximum Importance Projection* is done in that simple way, there will only be a cylindrical countersink at the position of the most important object (see Figure 15 left). There are several disadvantages with this approach. For example, if there is parallel projection used, the spatial impression of the objects will get lost. This does not produce very good results.

A better approach is to consider the cut-away view as a translational sweep with the footprint of the desired object as cross section (generical cylinder). By moving the footprint towards the eye point and scaling it, a countersink geometry is being created as shown in Figure 15 right. The countersink geometry forms a conical clipping frustum. During Raycasting, this can easily be achieved by changing the starting points of those rays, that intersect this clipping frustum.

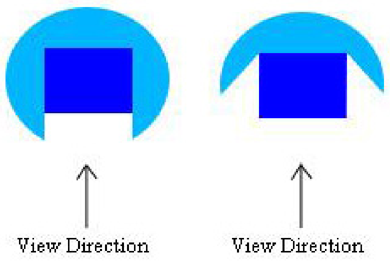


Figure 15: Maximum Importance Projection with cylindrical (left) and conical countersink (right).

To achieve this, there has to be some information about the depth of the objects. The last hit of a ray with an object along the viewing direction has to be known. For cylindrical *MImP* the ray samples ahead of the desired object (object with maximal importance and lowest sparseness value) are simply skipped. For conical *MImP* the footprint of the desired object has to be enlarged. Since the depth of the object is known, this can be done by image processing operations on the depth image. The enlarged depth footprint is processed by a 2D chamfer distance transform [1], a very fast way of doing this. Now the new ray starting points on the clipping frustum can be calculated as follows:

$$e_i = e_{max} - d_i \cdot s_c \tag{15}$$

$e_{max}$ is the maximal depth value of the desired object. $d_i$ is the distance of the ray to the object according to the depth footprint. $s_c$ is the slope of the countersink and $e_i$ is the calculated depth of the new starting point.

Now there should be a conical cut-out geometry. In order to simulate the cut-out correctly it is necessary to modify also the gradient vector on the clipping frustum, i.e. only the gradient of the first ray sample on any modified ray. This creates a countersink effect. Two components can be calculated using the 2D distance field, and for the z-component, the slope can be used. Since gradients inside objects are normally very small, the gradient also has to be scaled up.

### 4.3 Color an Opacity Modulation

The last point to consider is what to do with the level of sparseness values of the objects. As mentioned above, the objects have to be either dense or sparse, depending on the sparseness value. A very simple and fast method is called ”Color an Opacity Modulation”. As can be seen in Figure 13, the *IDVR* compositing takes place after the local rendering steps shading and compositing, so there are already calculated color and opacity values for each object along all rays. So, these values can be modified in this step. One possibility is to modify the saturation of the color of an object according to the sparseness value. Colors with a high sparseness attract the observer's eye more that colors close to gray. Hence, sparse objects will get a low saturation, whereas dense objects will maintain their saturation or even get an amplified one. Since the saturation does not change the occlusion of more important objects, the opacity has to be modulated at the same time.

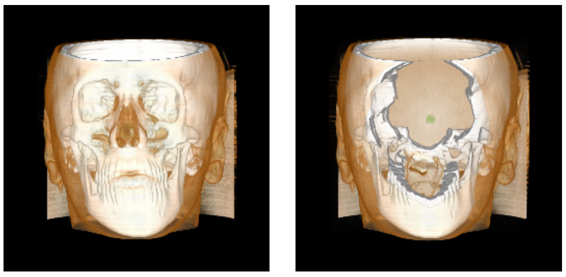


Figure 16: Illustrates two differently rendered images based on the same human head volume data set. Image (a) was rendered by using *DVR* and the introduced *Phong Illumination Model*. Otherwise, image (b) was sanitized by applying *IDVR*. Note that the head of image (b) contains a small green sphere with higher importance than the head itself.

Figure 16 obviously displays the same head. The left one is rendered with *Direct Volume Rendering* and *Phong Illumination*. The right one is the same, but it is rendered with *Importance-Driven Volume Rendering*. *Maximum Importance Projection* and *Color and Opacity Modulation* is used, whereas the opacity of the occluding objects is set to 0, so that it is fully transparent.

## 5 Implementation

In this section we will discuss specific implementation features of our application and also describe key elements of our software approach in more detail, particularly the implementation of the rendering pipeline. Furthermore, some additional features of the application which control the trade-off between general image quality versus rendering performance are shortly explained in the following section. For detailed description about the *Graphical User Interface (GUI)* of our application and comprehensive description how the application and how its different rendering modes works, we refer to the separate application documentations [14] and [13].

### 5.1 Ray Caster Implementation

We have already mentioned in this report that the basic rendering algorithm of our application is a standard Ray Casting routine which integrates all three rendering steps of the actual rendering pipeline. Ray Casting is an essential part of *Direct Volume Rendering (DVR)* in which separate single rays will be cast into the volume data set and evenly spaced sample points will be interpolated (opacity and color values) along a single ray. After the optical values of the single sample points have been calculated in the classification and shading step, they will be blended together by linear combination to represent the final optical representation for the corresponding ray.

Each ray will be exactly cast through the middle point of an associated pixel on the image plane and the final color value of the ray could be interpreted as the approximation of the actual color value for this specific pixel along the view direction. The image plane represents the viewable 2D projection of the volume data set depending on the view direction. It is always defined by its width dimension (note that the image plane is quadratic) and the corresponding resolution. Also the current location in relation to the volume data set coordinate system is an essential attribute of the Ray Casting routine. Due to simplification reasons we assume parallel projection during the rendering process, thus all rays exactly follow parallel to each other from the pixel middle point along the view direction into the volume data set. This assumption saves any additional projection calculation and simplifies the implementation of the rendering process.

To implement this basically described Raycasting algorithm into our application, we define following three main classes:

- The representation of the image plane *(Class ImagePlane)* contains all needed attributes of a current image plane instance which in particular are the orientation and location in relation to the world coordinate system, the current view direction and the current resolution of the image plane.
- The two render primitives ray sample and ray *(Classes RaySample and Ray)* are represented by helper classes to temporary saving of either all calculated values of a specific sample point (density, gradient, identity number, importance value, opacity, color) or all current sample points along a single cast ray.
- The actual Raycasting method (*Class SWRenderMachine)* actually contains three different methods all based on the Ray Casting routine to efficiently implement the three possible rendering modes *DVR, 2lVR and IDVR*.

The following *UML* class diagrams schematically illustrates the most important member variables and methods of those mentioned classes whereas Figure 18 refers to the *ImagePlane* class, Figure 17 presents the two basic primitive classes and, at last the *SWRenderMachine* class is been described by Figure 19.

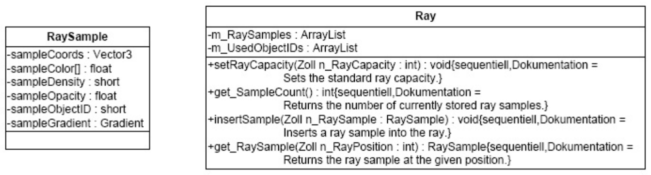

RaySample
-sampleCoords : Vector3
-sampleColor[] : float
-sampleDensity : short
-sampleOpacity : float
-sampleObjectID : short
-sampleGradient : Gradient
Ray
-m_RaySamples : ArrayList
-m_UsedObjectIDs : ArrayList
+setRayCapacity(Zoll n_RayCapacity : int) : void{sequentiell,Dokumentation = Sets the standard ray capacity.}
+get_SampleCount() : int{sequentiell,Dokumentation = Returns the number of currently stored ray samples.}
+insertSample(Zoll n_RaySample : RaySample) : void{sequentiell,Dokumentation = Inserts a ray sample into the ray.}
+get_RaySample(Zoll n_RayPosition : int) : RaySample{sequentiell,Dokumentation = Returns the ray sample at the given position.}

Figure 17: Shows the basic member variables and methods of the two primitives classes *RaySample* and *Ray*.

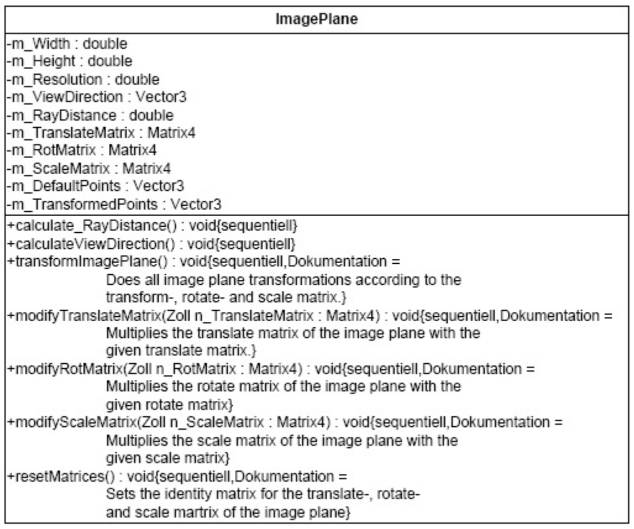

ImagePlane
-m_Width : double
-m_Height : double
-m_Resolution : double
-m_ViewDirection : Vector3
-m_RayDistance : double
-m_TranslateMatrix : Matrix4
-m_RotMatrix : Matrix4
-m_ScaleMatrix : Matrix4
-m_DefaultPoints : Vector3
-m_TransformedPoints : Vector3
+calculate_RayDistance() : void{sequentiell}
+calculateViewDirection() : void{sequentiell}
+transformImagePlane() : void{sequentiell,Dokumentation = Does all image plane transformations according to the transform-, rotate- and scale matrix.}
+modifyTranslateMatrix(Zoll n_TranslateMatrix : Matrix4) : void{sequentiell,Dokumentation = Multiplies the translate matrix of the image plane with the given translate matrix.}
+modifyRotMatrix(Zoll n_RotMatrix : Matrix4) : void{sequentiell,Dokumentation = Multiplies the rotate matrix of the image plane with the given rotate matrix}
+modifyScaleMatrix(Zoll n_ScaleMatrix : Matrix4) : void{sequentiell,Dokumentation = Multiplies the scale matrix of the image plane with the given scale matrix}
+resetMatrices() : void{sequentiell,Dokumentation = Sets the identity matrix for the translate-, rotate- and scale martrix of the image plane}

Figure 18: Shows the basic member variables and methods of the class *ImagePlane* which actually contains all necessary attributes of the image plane during the Raycasting process.

## 5.2 Class Structure of the Rendering Pipeline

Based on the basic rendering pipeline defined by [4], we use three main rendering steps to calculate the optical output values color and opacity starting from input values density, importance, gradient vector and object membership identity number. This three steps pipeline (see Figure 20) is completely embedded into the standard Raycasting method (see Section 5.1) and will be subsequently passed through for each cast ray. Since the *IDVR* enhancement has to integrated into this standard rendering pipeline, we also added an additional step to the pipeline which we called *IDVRCompositingModel*. Figure 20 shows that this new step is placed between the standard shading and compositing step and its main task is the modulation of the opacity of the ray samples based on the valid sparseness values associated to the included volume objects.

To guarantee more program flexibility and efficiency in the application we use a highly polymorphic approach to implement the different classes related to the rendering pipeline. Thus, each rendering step, i.e. classification, shading and compositing, consists of several different eligible classes which represent a separate but self-contained possibility to accomplish the associated rendering step. Only this flexible class structure makes an easy combination of a single classification model, shading model and compositing model possible which altogether compose a possible rendering process mode. With other words, this polymorphic class structure provides all arbitrarily selectable modes of the rendering process which con-

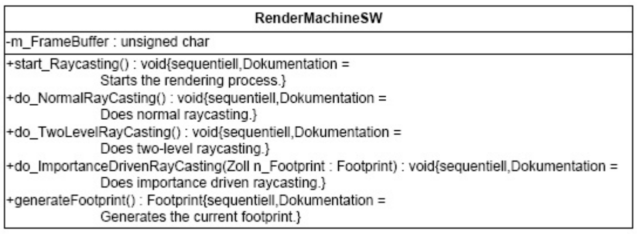

RenderMachineSW
-m_FrameBuffer : unsigned char
+start_Raycasting() : void{sequentiell,Dokumentation = Starts the rendering process.}
+do_NormalRayCasting() : void{sequentiell,Dokumentation = Does normal raycasting.}
+do_TwoLevelRayCasting() : void{sequentiell,Dokumentation = Does two-level raycasting.}
+do_ImportanceDrivenRayCasting(Zoll n_Footprint : Footprint) : void{sequentiell,Dokumentation = Does importance driven raycasting.}
+generateFootprint() : Footprint{sequentiell,Dokumentation = Generates the current footprint.}

Figure 19: Shows the basic member variables and methods of the class *RenderMachineSW* which actually implements the Raycasting process with the three different modes *DVR*, *2lVR* and *IDVR*.

sists of one classification model, one shading model and one composting model. To realize this required polymorphic class structure each of this three rendering steps are represented by a general interface which defines all basically needed method prototypes. Thus, all deviated classes of the rendering pipeline have to integrate the corresponding interface which relies to a specific rendering step.

We implemented two basic classification models which on the one hand base upon an arbitrarily chosen one dimensional transfer function and on the other hand integrate a predefined linear transfer function. Additionally, *IDVR* enhanced classes are provided to calculated *Footprints* of the included volume objects which are essential for *IDVR* compositing. The *Footprint* of a volume object is the corresponding projection on the image plane and will be used to estimate the cut-out. Figure 20 gives an overview to the interface class of the classification step.

The shading step contains four different classes which actually implements the possible shading models. For detailed information about these four shading models see Section 3.1. *LMIP, Phong Illumination, Contour* and *Tone Rendering* are implemented in the corresponding classes and provide the adequate calculation of the sample point's color and opacity values. Figure 20 illustrates the interface class of the shading step.

The following step of the rendering pipeline contains the opacity modulation of the current sample point to actually provide the required *IDVR* enhancement. The key issue of opacity modulation is the selection of a valid level of sparseness based on the associated volume object. As already mentioned *Maximum Importance Projection (MImP)* selects the current valid level of sparseness level of all included volume objects (see Section 4 and is basically implemented in *Class IDVRMImPCompositingModelSW* which has integrated the *IDVRCompositingModel* interface. Prior to this adapted compositing the *Class IDVRColAndOpModulatorSW* implements the necessary modulation of color (new specular intensity because of modificated gradients along the cut-away borders) and opacity of the current sample point. Thus, this class is directly included into the base class *IDVRMImPCompositingModelSW*.

Finally, the compositing step have to provide the two required compositing models which actually are *Direct Volume Rendering (DVR), Two-Level Volume Rendering (2lVR)*. We have implemented two separate classes to integrate the basic compositing modes which are *Class DvrCompositingModelSW* and Class *LMIP-Compositing*. The second compositing class actually performs no blending of sample points along a specific ray but chooses only one sample point next to a local maximum reference opacity value. This reduced compositing model is required to implement a correct *MIP* shading model which do actually not include the *DVR* compositing mechanism. Furthermore, *2lVR* is not included in an own class structure but the selected local and global compositing methods will

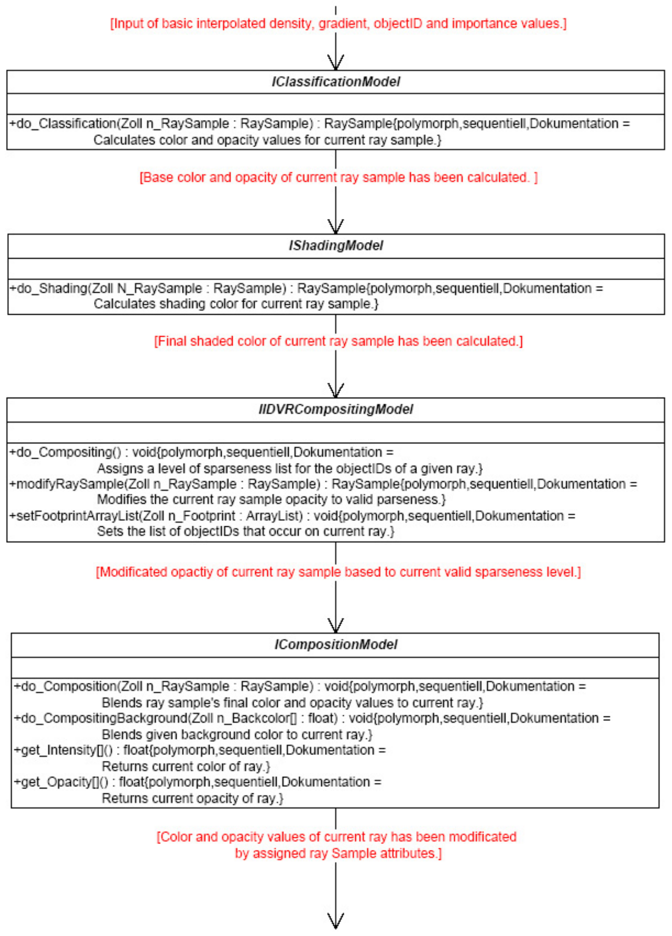

[Input of basic interpolated density, gradient, objectID and importance values.]
IClassificationModel
+do_Classification(Zoll n_RaySample : RaySample) : RaySample{polymorph,sequentiell,Dokumentation = Calculates color and opacity values for current ray sample.}
[Base color and opacity of current ray sample has been calculated. ]
IShadingModel
+do_Shading(Zoll N_RaySample : RaySample) : RaySample{polymorph,sequentiell,Dokumentation = Calculates shading color for current ray sample.}
[Final shaded color of current ray sample has been calculated.]
IIDVRCompositingModel
+do_Compositing() : void{polymorph,sequentiell,Dokumentation = Assigns a level of sparseness list for the objectIDs of a given ray.}
+modifyRaySample(Zoll n_RaySample : RaySample) : RaySample{polymorph,sequentiell,Dokumentation = Modifies the current ray sample opacity to valid parseness.}
+setFootprintArrayList(Zoll n_Footprint : ArrayList) : void{polymorph,sequentiell,Dokumentation = Sets the list of objectIDs that occur on current ray.}
[Modificated opactiy of current ray sample based to current valid sparseness level.]
ICompositionModel
+do_Composition(Zoll n_RaySample : RaySample) : void{polymorph,sequentiell,Dokumentation = Blends ray sample's final color and opacity values to current ray.}
+do_CompositingBackground(Zoll n_Backcolor[] : float) : void{polymorph,sequentiell,Dokumentation = Blends given background color to current ray.}
+get_Intensity[]() : float{polymorph,sequentiell,Dokumentation = Returns current color of ray.}
+get_Opacity[]() : float{polymorph,sequentiell,Dokumentation = Returns current opacity of ray.}
[Color and opacity values of current ray has been modificated by assigned ray Sample attributes.]

Figure 20: Illustrated the basic class structure of the rendering pipeline with *IDVR* enhancement (see 4) to calculate rendering values for a specific ray sample along the cast ray. The included classes actually are *Interface Class Definitions* to provide standardized use of different rendering methods within the Raycasting process.

be separately executed inside the Raycasting method (see Section 3.2).

## 5.3 Output of Rendering Results

Basically, two different ways to output the currently calculated rendering results are integrated in our application, whereas the display onto the screen is the main output. Additionally, the rendering results can be directly saved by a selected graphical file format. During the rendering process a texture based data structure is been used to save all already calculated results which actually consists of the three RGB color components. Thus, we use the texture based output of *OpenGL* to make the output process as efficient as possible. Texture objects of *OpenGL* provides a simple way to meet this requirement and will be directly initialized in the class *DrawModel* by using the texture format *GL_RGB*, the texel format *GL_UNSIGNED_BYTE* and, of course, size and actual data content of the calculated texture data structure.

Furthermore, the *DrawModel* class defines the needed polygon *(GL_QUAD)* to map the actual texture on it. This polygon actually represents the entire screen display plane. To guarantee an efficient sampling of this texture during size changes automatic *Trilinear* interpolation during *OpenGL* rasterization will be also used.

## 5.4 Additional Features of the Implementation

This Section shortly describes some important and additional features of our application to provide efficient and flexible processing during *Ray Casting*. Most of those following features describe rendering constraints of the implemented Raycasting process to control the obvious trade-off between higher image quality and lower computational expense. These constraints can be manually selected to provide individual preferences of the Raycasting process depending on desired image quality and maximum rendering speed.

The following listing gives an overview of significance and effect of the single features to the rendering process:

**Early Ray Termination (ERT).** To shorten the computational efford during Raycasting, this easily implemented feature gives the user control over calculation time and targeting image quality. Based on the method of [1] we provide the possibility to stop compositing of sample points along the current ray due to a specifically chosen opacity reference value. This reference value refers to the a lower boundary of valid opacity and, thus, if the current blended opacity value of the ray exceeds this given reference value the ray compositing will be stopped instead of going on until background has been reached.

Obviously, *ERT* always produces lower image quality because the entire amount of sample points along the ray will not be used for the compositing calculations and thus the optical representation of the corresponding pixel gains a higher approximation error.

**Sampling Distance.** Due to all sample points along an actual ray are evenly spaced the *sampling distance* measures this distance between to succeeding sample points along a ray in measurements of world coordinates. Thus, the higher the *sampling distance* has been selected the lower is the amount of sample points along the current ray and the approximation error of the actual optical values for the corresponding image plane pixel increases. The advantage of a higher *Sampling Distance* is faster computation of the rendering process but concurrent lower image quality should always be minded.

**Gradient Estimation.** During the rendering process of our application the actual gradient vector of each sample point becomes essential and therefore, the voxel-based estimation of those gradient vectors within the volume data set has to be included into the classification step of our rendering pipeline. We have implemented two gradient estimation methods to calculate each voxel gradient vector. Those two gradient estimation methods differ in approximation quality and computational effort. Obviously, the gained average error margin of the gradient vector approximation has a deciding impact to the final image quality, i.e. if the approximation error is low during *Phong Illumination* the specular effect will be visibly smoother and will look more ”realistic”.

The first and low quality estimation method is called *Central Differences* and measures the gradient vector of a specific voxel due to the explicit density values of the neighboring voxels. To calculate an approximation of the actual gradient vector the average sum of the first derivatives in the three main coordinate directions (parallel to the x, y and z axis) will be used, whereas the first derivatives will be simply assumed as the linear difference between two specific voxels. The second and high quality method approximates the gradient vector by estimating the density function itself in a local neighborhood due to the local 3D regression hyperplane of the calculated voxel.

**Interpolation Methods.** During the classification step all required basic attribute values of a sample point have to be interpolated in relation to the corresponding nearest local voxel neighborhood. We have defined that local voxel neighborhood as the eight surrounding voxels of the current sample point location in the volume data set. Particularly, the following attribute values of sample points must be interpolated by an adequately selected interpolation method:

- density value
- gradient vector
- object membership identity number
- importance value

We provide two interpolation methods which actually differ in approximation error margin and calculation effort. It only depends on the user's preferences which of those two interpolation methods is used to calculate the attributes above. Note that the actual amount of sample point attributes which have to be interpolated depends on the rendering mode. Thus, standard *(DVR* rendering only needs density and gradient values, *2lVR* additionally requires identity numbers and *IDVR* obviously includes importance values.

However, similar to the gradient estimation methods above two different interpolation methods are implemented in our application whereas quality of approximation results and computational effort considerably differ in both methods. *nearest neighbor interpolation* is a pretty fast interpolation method but produces extensively large interpolation error margins, otherwise, *trilinear interpolation* is batter and more sophisticated interpolation method but needs much more computational time.

## 6 Results

In this section we will summarize some important main properties of *IDVR* and mark out the difference to standard rendering methods based on example images. As already mentioned at various position in this report the key advantage of *IDVR* is the advanced visibility of included volume objects of the current volume data set and thus increase the quality of images results because of sophisticated graphical representation capabilities. We will demonstrate this main requirement in this section with various images rendered by our implemented application. Furthermore, some properties of the implemented *IDVR* process are also discussed which are different sizes of the cut-out area and the specular intensity modification due to the rearrangement of the gradient vectors along the current cut-out slopes. Thus, specular reflection will also be added to all sample points on the cut-out surfaces and not only on actual surfaces of single volume objects. Due to this important specular intensity modification the realistic effect of the rendered images will be guaranteed although *IDVR* actually relates to the non-photorealistic rendering domain.

The basic volume data set which has been used to synthesize the following image results allegorates a single human head and thus will be called Head data set. The resolution of this data set is 184*x*256*x*170 voxels and all included tissue layers and other anatomy parts are completely included. Two different transfer functions are applied to the example images. The first one extracts only skin and bone structure of the head and the second one additionally represents muscle and brain tissues. Besides, a second volume data set has also been used for example images to supplementarily demonstrate different *IDVR* properties. This volume data set has been synthetically defined and has a resolution of $256^3$ voxels. It contains two outer concentric spheres and a small cuboid inside the second sphere. Finally, all illustrated images were exclusively rendered with Phong Illumination to provide adequate specular intensity reflection.

Figure 21 illustrates the principal capability of *IDVR* which is to guarantee the utter visibility of entire volume objects based on the higher importance although possible occlusion of other volume objects occurs. All three images displays the left side of the Head data set with identical directional light coming from the view point position. Additionally, we selected the inner section of the head near the cerebellum and the top end of the spine as the most important part of the data set. This could be achieved by placing an translucent sphere (separate volume object) at that location. The left most image of Figure 21 was evidently rendered without any *IDVR* enhancement, the other two images were obviously synthesized by that rendering modification. Thus, the difference in presentation of included information of the Head data set is evidently because the inner bone structure of the head is clearly visible in the middle and right image. This bone structure appears to be the connection between the actual head and the top section of the spine. The correct light intensity calculation of the visible inner structure, in particular that single bone structure, also is observable and the specular reflection on the front side of the spine connection can be clearly viewed. Furthermore, the middle image differs from the right images in the cut-out size and thus a larger and less important part of the head surface has been obviously clipped.

The next example images which are indicated by Figure 22 displays again the Head data set with the identical transfer function of the previous figure but with a different definition of the importance hierarchy. Also the light direction was identically defined in relation to the view direction for both images of Figure 22. Instead of the inner bone structure a particular part of the brain, called hypophysis, is been defined as the most important volume object of the Head data set. We have simplicity used a small blue sphere to simulate the actual size and form of the hypophysis because the selection of a observable transfer function is very complex, particularly due to the similar density values of the neighboring areas. Furthermore, the hypophysis object was rendered with higher transparency and diffuse material property to reduce the specular intensity reflection. Note that the simulated hypophysis can be utterly viewed from different view directions although it is occluded by surrounding tissues and bone structures.

A synthetical volume data set was used to render the example images of Figure 23 which actually contains three volume objects, two outer concentric spheres and an inner small cuboid. The importance values for these volume objects increase from the outer sphere to the inner cuboid successively and thus the cuboid has the highest importance value. Those four images of Figure 23 differ in the actual size of the cut-out area which is largest in image (a) and is successively reduced to images (b), (c) and finally (d). Thus, the smaller sphere (second most important volume object) can be viewed better in image (d) because less parts are clipped away due to the cut-out. Furthermore, the specular intensity reflection of the cut-out surface is located at lower left corner which refers to the actual light direction from the upper right position (in relation to the view position).

Finally, the last group of example images demonstrates *IDVR* enhancement based on a more realistic transfer function of the Head data set. Figure 24 shows the human head from different view position, i.e. from the left side and the right side, and also varying light direction. Due to the more complex transfer function we tried to visualize a more realistic illustration of the human head which contains not only bone and skin structure but also muscle and brain tissue. Thus, the specular intensity reflection on the cut-out surface appears to be more visible than in Figures 21 and 22. Both images reply to the combination of possible realistic rendering model (together with adequate transfer function) and the cut-out of specific image parts. Exactly this point is the main advantage of the use of *IDVR* enhancement.

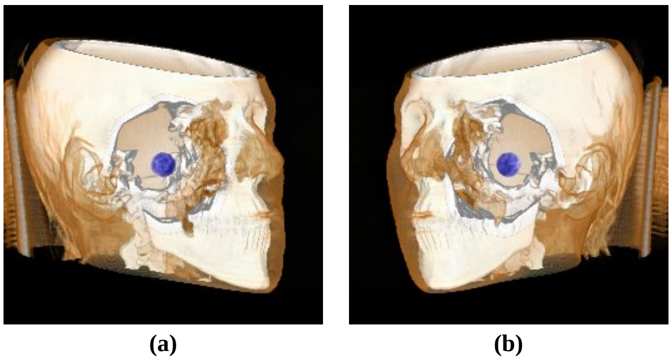


Figure 22: Illustrates two *IDVR* rendered image of the Head data set with included hypophysis as the most important volume object. The actual hypophysis is simulated by the small blue sphere in the middle of the brain section. Image (a) shows the right side of the head and image (b) display the left side. The directional light direction is identical to the view direction in both images.

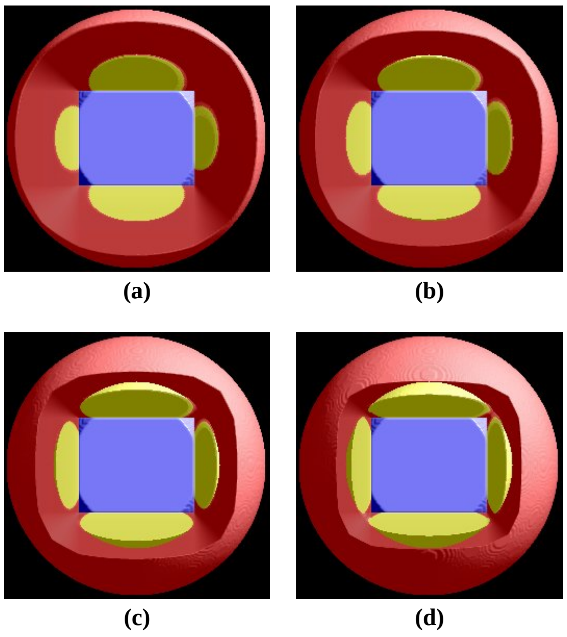


Figure 23: These four images display a synthetic volume data set which all consist of two concentric spheres and a small inner cuboid. The actual importance value of those volume objects increases from the outer sphere to the inner cuboid. Additionally, the cut-out size differs in all four images, whereas the size of the actual cut-out decreases from image (a), (b), (c) to image (d) successively.

## 7 Conclusion

We have shown that common volume visualization methods insufficiently provide mechanism to treat hidden volume objects. How-

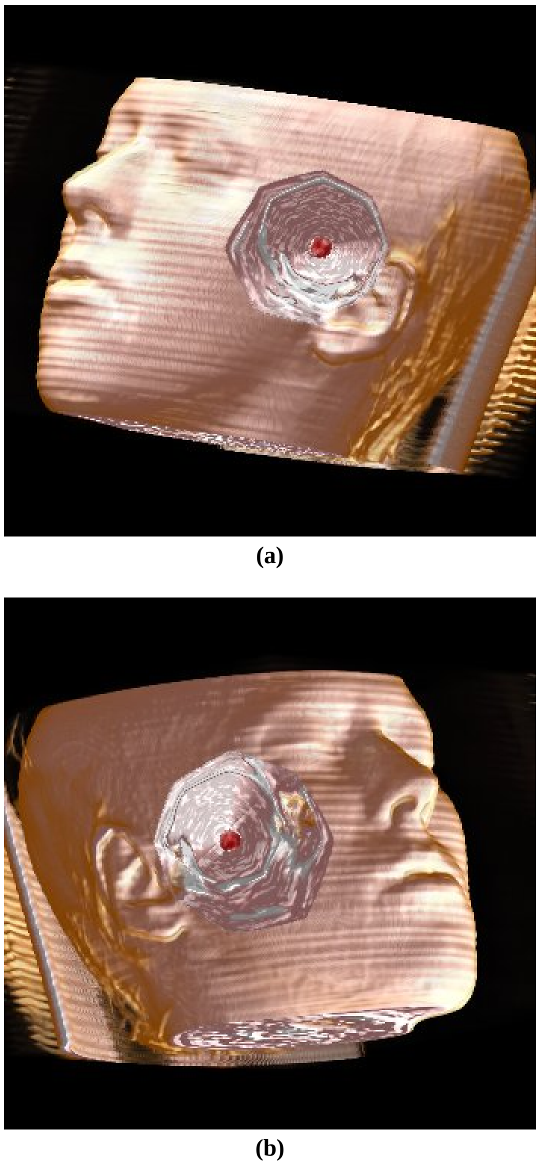


Figure 24: Illustrates two *IDVR* rendered image of the Head data set with different transfer function as in figure 22. A small red sphere was defined to create a synthetic cut-out in the middle section of the head. Image (a) shows the left side of the head and image (b) display the right side. Note that the light direction is targeted on the lower chin and differs in the particular view direction of both images. Thus, the specular reflection is located in relation to the light direction in images (a) and (b).

ever, volume object occlusion in relation to arbitrarily chosen view direction within highly complex volume data sets has become an urgent problem in volume visualization nowadays. Also in term of creating additional ways of convey informational content to users an importance-driven rendering approach promises new possibilities because of the capability of selective volume object rendering independent to the current view direction. Based on a importance driven rendering approach the pre-definition of a specific importance hierarchy absolutely is essential and that importance hierarchy actually represents the current relationship between all included volume objects within the data set. We also described that *IDVR* in-

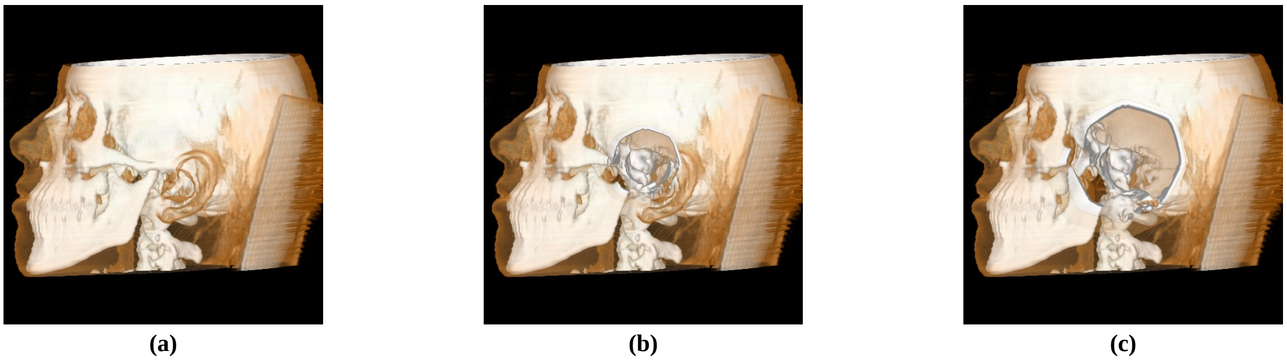


Figure 21: These three images illustrate different sizes of the cut-out view with the lower middle section of the inner head to be most important. The cut-out is that part of the image which has been clipped to ensure clear view to the important volume objects. Image (a) shows the head without any *IDVR* rendering, image (b) illustrates a small cut-out and image (c) includes a larger cute-away.

troduced by [9] provides those needed capabilities exactly. In particular medical visualization applications need some advanced image rendering to give users flexibility to choose which parts of the data (i.e. specific organs or tissues) set should be fully displayed despite of any occlusion. For example tumors surrounded by organs or other tissues could be fully visualized and thus gives physicians a better tool to treat those critical pathologies.

In this report we introduced an application which actually implements this importance driven approach embedded in a standard Raycasting algorithm. Furthermore, our application also integrates *Phong Illumination shading* and non-photorealistic rendering methods *Contour* and *Tone shading*. Thus, specific attributes of volume data sets and of volume objects can be expressed by the chosen rendering process in a more comprehensive way to increase the interpretation of the graphical results. To actually meet requirements of that described approach we implemented the Maximum Importance Projection method based on *Importance- Driven Volume Rendering (IDVR)* which has actually proofed to be a simple and very efficient way to fulfill our complements. We have also described that *IDVR* enhancement based on a standard Raycasting method with integrated *Direct Volume Rendering* and *Two-Level Volume Rendering* can be easily combined with standard realistic or non-realistic shading methods. Based on this new approach we created a more flexible and user-oriented visualization application than applications relying on standard rendering models, whereas the user can individually selects specific volume objects which will be displayed despite any possible occlusion in relation to the current view direction.

Our illustrated results indicate better informational representation and high quality of volume object representation. Since our application is only implemented on a CPU approach time consumption and efficiency still are main problems of this application and any use of large volume data sets (i.e. data sets larger than $1024^3$) would lead to unsatisfying calculation times. Interactivity is absolutely not possible without using advanced capabilities of at least consumer graphic cards. Thus, future applications based on *IDVR* enhancement should be directly integrated into a *GPU* approach [11] which certainly provides much higher efficiency and interactivity regarding *IDVR* rendering. Due to the increased capability of current consumer graphic cards standard Raycasting methods can be implemented into several corresponding *Fragment Shaders* instead of use some approximation methods like texture based approaches. Thus, the implementation of Two-Level volume rendering and *IDVR* becomes much easier and can be integrated into such *GPU* based rendering pipeline as [12] has introduced.

## ACKNOWLEDGEMENT

At the end of this report of our *IDVR* enhanced application both of us want to explicitly thank *Ivan Viola* who not only gratefully helped us during the entire project phase but also provided us with the basic scientific work (main author of [9]). Based on his work we were able to conduct this project which surely was very interesting and a good sight-in into the area of volume visualization. Also we want to thank *Prof. Meister Eduard Gröller* who was the supervisor of the project and actually leaded us to this specific subject by making the deciding proposal at the first meeting.

## REFERENCES

[1] Gunilla Borgefors. Distance transformations in digital images. In *Comput. Vision Graph. Image Process.*, 34(3):344–371, 1986.

[2] Balázs Csébfalvi, Lukas Mroz, Helwig Hauser, Andreas König, and Eduard Gröller. Fast visualization of object contours by non-photorealistic volume rendering. *Comput. Graph. Forum*, 20(3), 2001.

[3] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452. ACM Press, 1998.

[4] Marc Levoy. Display of Surfaces from Volume Data. In *IEEE Computer Graphics and Applications*, Vol. 8, No. 3, May, 1988

[5] Marc Levoy. Efficient Ray Tracing of Volume Data. In *ACM Transactions on Graphics*, Vol. 9, No. 3, July, 1990, pp. 245-261.

[6] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[7] László Neumann, Balázs Csébfalvi, Andreas König, and Eduard Gröller. Gradient estimation in volume data using 4d linear regression. *Comput. Graph. Forum*, 19(3), 2000.

[8] Penny Rheingans and David S. Ebert. Volume illustration: Nonphotorealistic rendering of volume models. In *IEEE Trans. Vis. Comput. Graph.*, 7(3):253–264, 2001.

[9] Ivan Viola, Armin Kanitsar, and Meister Eduard Groller. Importance-driven volume rendering. In *Proceedings of the IEEE Visualization 2004 (VIS'04)*, pages 139–146. IEEE Computer Society, 2004.

[10] Hauser, H.; Mroz, L.; Italo Bischi, G.; Groller, M.E. Two-level Volume Rendering. In *IEEE Transactions on on Computer Graphics and Visualization*, Volume 7, Issue 3, Page(s):242 - 252, 2001.

[11] Hadwiger Markus, Berger Christoph, Hauser Helwig. High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware. In *Proceedings of IEEE Visualization 2003*, pp. 301-308, 2003.

[12] Jens Krueger, and Ruediger Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization 2003*, 2003.

[13] API documentation of an IDVR based application, *http://www.cg.tuwien.ac.at/courses/projekte/vis/finished/ CThurnher_BPflugfelder/downloads/IDVR_APIDocumentation/index.html*, April 2005.

[14] GUI and user documentation of an IDVR based application, *http://www.cg.tuwien.ac.at/courses/projekte/vis/finished/ CThurnher_BPflugfelder/downloads/IDVR_UserDocumentation.pdf*, April 2005.