FAKULTÄT FÜR !NFORMATIK

Visibility Histograms in Direct Volume Rendering

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Informatik

eingereicht von

Gerlinde Emsenhuber
Matrikelnummer 0026580

am:
Institut für Computergraphik und Algorithmen

Betreuung:
Betreuer: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Stefan Bruckner

Wien, 4. November 2008

(Unterschrift Verfasserin) (Unterschrift Betreuer)

Abstract

This thesis introduces visibility histograms as a method for analyzing volumetric datasets. These histograms show how much the data points within a 3D dataset that have the same scalar value influence the image which is created by rendering the dataset with a particular transfer function and from a particular viewing direction. These histograms can be used to gain insights into the internal structure of volumetric datasets, in particular information about occlusions. Furthermore, the possibility of automatically calculating transfer functions which generate a particular visibility histogram when applied to a dataset from a particular viewing direction is explored. Two methods which can be used to calculate a matching transfer function for a visibility histogram are explained, one of which is based on a genetic algorithm approach, while the other is an heuristic.

Zusammenfassung

In dieser Diplomarbeit werden Visibility Histogramme als Methode um volumetrische Datensätze zu analysieren eingeführt. Diese Histogramme zeigen an, welchen Einfluss die Datenpunkte mit demselben Skalarwert in einem solchen Datensatz auf das Gesamtbild, welches entsteht wenn der Datensatzes mit einer bestimmten Transferfunktion und von einer bestimmten Blickrichtung aus mithilfe einer Direct Volume Rendering Methode wie zum Beispiel Volume Raycasting dargestellt wird, haben. Die Histogramme können benutzt werden um Aufschluss über die interne Struktur von volumetrischen Datensätzen, insbesonders Verdeckungen von verschiedenen Strukturen zu erhalten. Weiters wird auf die Möglichkeit eingegangen, mithilfe der Histogramme automatisch Transferfunktionen zu berechnen, welche bei Anwendung auf einen Datensatz aus einer bestimmten Blickrichtung eine gewünschte Sichtbarkeitsverteilung erzeugen. Zwei Ansätze wie dies realisiert werden kann werden vorgestellt: Der erste Ansatz basiert auf einem genetischen Algorithmus, der andere auf einer Heuristik.

Contents

Chapter 1

Overview

This thesis belongs to the research field of volume visualization, in particular to the field of direct volume rendering, also commonly referred to as DVR. In traditional computer graphics 3D objects are created using high-level surface representations such as polygonal meshes, NURBS or subdivision surfaces. The visual properties of surfaces are described by means of shading algorithms, e.g. the Lambertian diffuse reflection model or BRDFs. Using surface rendering, typically only the surface points of an object contribute to the final image, while the interior points of an object and the light interaction that takes place between them are unaccounted for. In contrast, direct volume rendering takes the interior points of an object into account. Direct volume rendering techniques typically generate images from a set of three dimensional scalar data. Often, these data sets come in the form of regular volumetric grids: they consist of a group of slice images with the slices acquired in a regular pattern, with each single slice containing a regular amount of pixels in a regular pattern. The elements of such a grid are also called voxels, and are the three dimensional counterpart to pixels. Based on their scalar value, these voxels are assigned a certain color and opacity. This is done with the help of a transfer function, witch can be a simple ramp, a piecewise linear function or even a random table. In order to generate an image, the three dimensional set of voxels is projected onto the image plane. The value of a single pixel is influenced by the voxels that are projected onto it.

Volume rendering has become a very important area of research over the last decades. Its applications reach from medical imaging to scientific visualization of data. It is used in computer graphics, where is can be employed to model natural objects that are difficult to describe with surface models, such as clouds,

fog or fire. It also plays a very important role in medicine, due to its potential use of giving a three dimensional insight into the internal structure of the human body, organs etc.

Since volume rendering deals with 3D data sets, its possible that a structure of interest is occluded by another structure. With the help of the transfer function, it can be specified which points of the data set should be completely transparent and which ones should be opaque. This is done by assigning specific alpha values to data points. Finding a good transfer function that shows the desired structures can be a rather complex task, often finding an appropriate transfer function is done by trial-and-error. A very simple approach would be to assign the data values that are of interest full opacity, while making all the other data values completely transparent, but in doing so, the context surrounding the structure of interest would be lost. In this thesis visibility histograms are introduced as a way to give insight into the internal structure of a dataset. ”Visibility” is a measure for the amount of influence that a data point within a volumetric data set has on an image generated by a direct volume rendering algorithm. It is higher for points which are not occluded by other structures or have a high opacity and lower for points which are occluded or have a low opacity. The visibility information is displayed in the form of a histogram, where data points with the same scalar value are sorted into the same histogram bin. Visibility histograms represent the visibility distribution in a data volume, and are similar to the concept of color histograms which are commonly used in computer graphics and give an insight into the color distribution a image. These histograms are used for object recognition, and as appearance-based signature to classify images for content-based image retrieval systems [43, 34].

The second part of this thesis deals with automatically generating transfer functions that enhance the visibility of certain data points within the dataset, while trying to preserve the visibility of the the rest of the data points if possible. Since the generation of transfer functions can be time-consuming, methods for automatically generating them have become an important area of research as well. Unfortunately, calculating a transfer function which produces exactly a specified visibility histogram is computationally expensive, plus there is no guarantee that such a transfer function can actually be found. Therefore, two approaches for

generating a matching transfer function to a visibility histogram are presented: one is based on genetic programming, the other is a heuristic approach.

1.1 Structure

The thesis is structured in the following way: Chapters 2 and 3 provide an overview of related work that this thesis is based on. Chapter 2 deals with volume visualization in general, explaining several surface rendering and volume rendering techniques and dealing with the theory behind direct volume rendering methods. Chapter 3 gives an overview of genetic algorithms. Chapter 4 deals with visibility histograms, explaining what precisely is meant by the term visibility. Chapter 5 deals with the automatic generation of transfer functions that match a given visibility distribution, explaining the two approaches that were implemented to reach this goal. Chapter 6 deals with implementation details, and in chapter 7, result images that were obtained with the implemented methods are presented. Chapter 8 is a summary of the presented work.

Chapter 2

State of the Art in Volume Rendering

This chapter gives an overview of the different techniques used in volume visualization. Section 2.1 explains the structure of the data sets that are used in volume visualization. Section 2.2 introduces several commonly used surface rendering algorithms, while section 2.3 deals with volume rendering techniques. Subsection 2.3.1 takes a closer look at the optical models that are used in volume rendering, while subsection 2.3.2 explains several important volume rendering methods. Section 2.4 explains the role of transfer functions in direct volume rendering, and introduces several techniques that are used for automatic transfer function generation. Finally, section 2.5 presents several Focus+Context visualization techniques for direct volume rendering.

2.1 Datasets used in Volume Rendering

The input datasets for volume visualization can be generated by different methods and devices. Primary sources of volumes are:

- sampled data of real objects
- computed data produced by computer simulations
- modeled data generated from a geometric model

One of the earliest scientific fields that adopted volume rendering was medical imaging. In this field, the 3D data is typically acquired by some kind of scanning device. In particular, CT (computerized tomography) and MRI (magnetic resonance imaging) are two methods which are often used to generate datasets. Medical imaging devices typically generate discretized volume datasets, where

a single scalar value is given for a discrete point in 3D space. Another source of data volumes are simulation results. For example, CFD (computational fluid dynamics) simulations in engineering or computed electromagnetic fields in physical sciences can generate volume datasets. Voxelization is an example for a method that generates volume datasets from a surface descriptions of a 3D object. Volume datasets span three dimensions, and assign one or more values to each position in the data volume. If there is exactly one value per position, the volume is also referred to as scalar field, otherwise it it called vector field. Datasets generated by measuring or sampling typically are discretized, i.e. they consist of a number of discrete sample points. The values between the sample points are not known, and have to be estimated, e.g. by interpolating the values from the surrounding sample points. On the other hand, datasets that were generated by modeling can be continuous, i.e. they are defined by functions, and the exact value at any point within the volume can be calculated. A continuous representation of a dataset can be transformed into a discretized version by simply evaluating the functions describing the volume at a number of sample points. This thesis deals with volume datasets typically expect datasets in the form of a discrete scalar field, such as the datasets generated by medical imaging. The scalar value given for a position in the dataset is also referred to as density value.

The data points within the data volume are typically assumed to be arranged on a *uniform grid*. What precisely this means is that the vectors from each point in the volume the adjacent neighbors in the three dimensions are expected to create a Cartesian coordinate system with axes parallel to the Cartesian coordinate system of the entire volume. The distance between adjacent points can be different in the x, y and z direction, but for a particular direction, the distance is assumed to be constant for the whole volume. If the data points are arranged on a uniform grid, they form a number of cuboid cells of equal size. Most practical applications assume that the datasets are described by a uniform grid.

One particular example of an uniform grid is a Cartesian grid: in a Cartesian grid, the distance between adjacent data points is equal in all directions, i.e. the cells formed by the data points are cubes of equal size. A uniform grid can be transformed into a Cartesian grid by choosing a set of sample points that form a Cartesian grid, and interpolating the values at the sample points from the original

dataset. This process is also referred to as resampling of the data volume.
An important term that is associated with volume visualization is a *voxel*, short for ”volume element”. Voxels are the 3D equivalent of pixels. Just like pixels are the data elements in 2D images that hold the color values, voxels are the data elements in 3D images, holding the scalar value. There are several different possible interpretations of what precisely a voxel is used in literature. One possible view is that a voxel is a small cube that fills a small volumetric region with its associated data value. The other interpretation is that a voxel corresponds to a grid point in a uniform grid. They are defined as points in 3D space, with an interpolation scheme for calculation the value in between the data points.

2.2 Surface Rendering Methods

With surface rendering algorithms, surface representations such as polygonal models are extracted from the 3D data set, and subsequently rendered. The advantage of these algorithms is that once the model has been extracted from the data, it can typically be rendered very fast with the help of graphics hardware acceleration. For this reason, surface rendering algorithms were very popular for a long time. The disadvantage of surface rendering techniques is that while in the original data set, data values are available for every point in the volume, after extracting the surface model only the surface points are used to compute the final image. Also, depending on the algorithm which is used, the extraction of the surface model can be a computationally expensive task.
A straight-forward surface rendering algorithm that was developed very early is the *cuberille algorithm*, first introduced by Herman and Liu [16]. Cells from the data set are selected for rendering if the value of one or more corners is higher than the threshold value, and the value of one or more corners is lower than the threshold value. These cells represent the border between the cells that lie inside the object and the cells that lie outside of the object. This way, a surface model consisting of small cubes is generated.
Another surface rendering algorithm is the so called *contour connecting algorithm* developed by Keppel [21]. In this method, a three dimensional model is generated from contours. Starting with a particular threshold value, the algorithm first

searches for closed contours within each individual slice of the volume; This is accomplished by first searching for the threshold value in a particular slice, and by subsequently searching for a contour with values close to the threshold value if it is contained within the slice. The detected contours are then connected be geometric primitives between the slices.
Another widely used surface rendering algorithm is the *marching cubes algorithm*, which was first introduced by Lorensen and Cline [29]. The algorithm takes a certain threshold value and then traverses the data volume, taking eight adjacent data points at a time, thus forming an imaginary cube, and checking the scalar values at the eight data points against the threshold value. These eight data points are the replaced with an appropriate set of polygons: If the scalar value at a particular data point is higher than the threshold, it is marked with a 1, else it is marked with a 0. The eight values can be interpreted as integer in binary format, an can be used as an index into a table that contains the 256 possible polygon configurations (for some of the polygon configurations, see Figure 2.1. Finally each vertex of the generated polygons is placed on the appropriate position along the cube's edge by linearly interpolating the scalar values of the data points that are connected by that edge. The individual polygons are then fused together to form the desired surface.

During the last 10 to 15 years, surface rendering algorithms have lost a lot of their appeal. While rendering the extracted surface models used to be a lot more efficient and fast than computing an image with direct volume rendering methods, during the last couple of years graphic cards have steadily become faster, and capable of handling bigger amounts of data. For hardware-based volume rendering the size of the memory is especially important as the volume data set has to be stored on the graphics card: If the entire data set does not fit in the memory of the graphics card then it is impossible to execute the entire rendering process on the graphics card. E.g. a volume data set with a size of 512^3, i.e. 512 data points in each direction, where each voxel value is stored with 16 bit has a size of approximately 256 MB. A dataset of this size can be stored directly on the graphics card, and larger values are also possible.

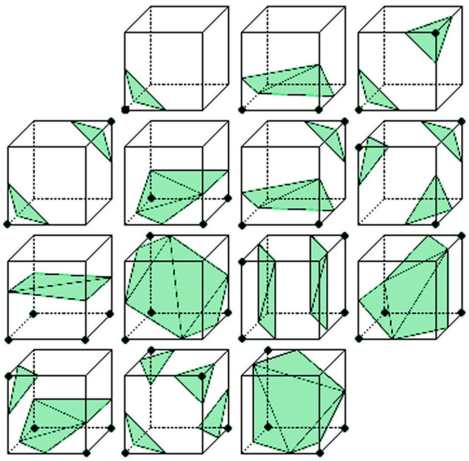


Fig. 2.1: The basic polygon configurations used by the marching cube algorithm for generating a surface model. The rest of the 256 configurations can be generated from these basic configurations by rotation and mirroring.

2.3 Direct Volume Rendering Methods

Direct volume rendering (DVR) methods generate images without computing an intermediate model of the data set. In DVR, images are generated by directly projecting the data volume onto the image plane, and voxels which belong to the same pixel in the image plane are blended together. With this approach, the internal points of the volume are taken into account when generating the image.

2.3.1 Optical Models

The physical basis for volume rendering relies on geometric optics, in which light is assumed to propagate along straight lines unless an interaction between the light and the medium takes place. The three most important types of interaction are:

- *Emission*: material actively emits light, increasing the light energy
- *Absorption*: material absorbs light, thereby reducing light energy
- *Scattering*: light can be scattered by the participating media, which changes the direction of light propagation. If the wavelength is not changed by scattering, the process is called elastic scattering, otherwise it is called

inelastic scattering.

The Rendering Equation

The so-called rendering equation introduced by Kajiya [20] describes a physically based model of how the light emitted by one or several light sources propagates through a scene. With this equation, it is possible to completely describe the light transfer that takes part within a scene. In Equation 2.1, one way to write the rendering equation is given.

$$I(x,x') = g(x,x') * [\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')\,dx''] \tag{2.1}$$

In the equation above, $I(x,x')$ stands for the light energy that arrives at point x from point x', and $g(x,x')$ is a geometric factor between the two points. The term $\epsilon(x,x')$ denotes the light which is emitted from point x' to point x, and $\rho(x,x',x'')$ is related to the intensity of the light scattered from point x'' to point x via a patch of the surface at point x'. The above formula is impossible to solve analytically, as it contains an integral over the whole space as well as infinite recursions, and therefore has to be simplified before it can be used.

The Volume Rendering Integral

The most commonly used model used in volume rendering is the emission-absorption model. In this model, scattering and indirect illumination are ignored. It is a popular model, because it provides a good compromise between generality and efficiency of computation. The emission-absorption model can be described by the volume rendering integral:

$$I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)dt} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)dt} ds. \tag{2.2}$$

The term I_0 represents the light entering the volume from the background at position $s = s_0$; $I(D)$ is the irradiance leaving the volume at position $s = D$ and reaching the camera. The first part of Equation 2.2 denotes the light from the

background attenuated by the volume. The second term represents the integral contribution of the source terms attenuated by the participating medium along the remaining distances to the camera. The term

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(t)dt \tag{2.3}$$

represents the optical depth between positions s_1 and s_2. The optical depth indicates how long light may travel through the volume before it is absorbed, thus indicating the typical length of light propagation before scattering occurs. Small values for the optical depth mean that the medium is rather transparent, while large values indicate that it is rather opaque. Defining the transparency between two point s_1 and s_2 as

$$(s_1, s_2) = e^{-\tau(s_1,s_2)} = e^{-\int_{s_1}^{s_2} \kappa(t)dt} \tag{2.4}$$

yields a simpler form of the volume rendering integral:

$$I(D) = I_0 T(s_0, D) + \int_{s_0}^{D} q(s)T(s, D)ds. \tag{2.5}$$

The volume rendering integral in its classic form, as shown in Equation 2.2, represents the emission-absorption model accurately, but does not take scattering into account. *Single scattering* of the external light can be included in order to achieve greater realism. Single scattering is often is often approximated by a local illumination model that imitates local surface rendering, such as the Lambert or the Blinn-Phong [3] illumination models. In such a case, the gradient of the scalar field serves as the normal vector for the local illumination model, as the gradient is identical to the normal vector on an isosurface through the respective point in space.

Discretization

In order to solve the volume rendering equation, numerical methods have to be applied as the integral cannot be evaluated analytically. A common approach

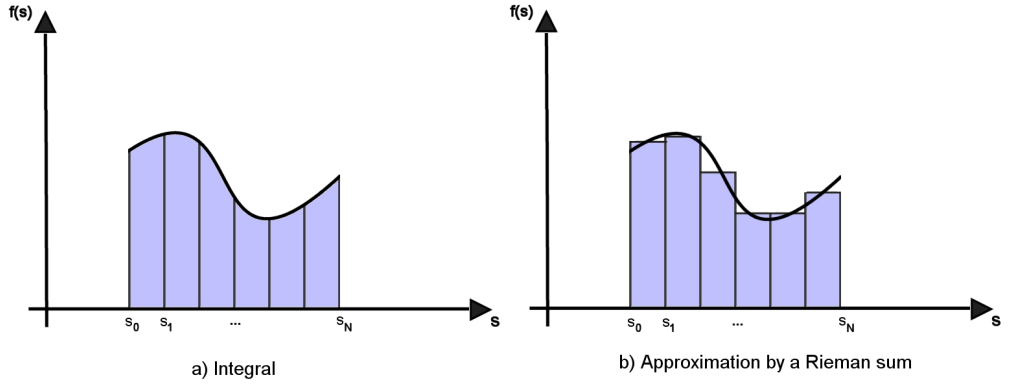


Fig. 2.2: A) Partitioning the integration domain into several intervals and b) approximating the integral by a Rieman sum.

is to split the integration domain into n subsequent intervals. The intervals are described by sample points $s_0 < s_1 < ... < s_n$, with s_0 being the starting point and $s_n = D$ being the endpoint of the integration. Using this approach, the radiance at location s_i can be expressed by the following equation:

$$I(s_i) = I(s_{i-1})T(s_{i-1}, s_i) + \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)ds. \tag{2.6}$$

A simpler notation for the color (radiance) c_i and the opacity T_i at sample point s_i is typically used in literature:

$$T_i = T(s_{i-1}, s_i)c_i = \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)ds$$

The radiance at the endpoint is given by:

$$I(D) = I(s_n) = I(s_{n-1})T_n + c_n. \tag{2.7}$$

Equation 2.7 can also be written as

$$I(D) = \sum_{i=0}^{n} c_i \prod_{j=i+1}^{n} T_j. \tag{2.8}$$

In order to solve Equation 2.8, the color and transparency contributions of the n intervals need to be known. The most common approach to compute these values is to approximate the volume-rendering integral by a Riemann sum over n equidistant segments. The color and opacity values are thus approximated by piecewise-constant functions.

Compositing

The basic idea behind compositing is to iteratively compute the discretized volume-rendering integral (Equation 2.8) by splitting the summations and multiplications into several simpler operations that are executed sequentially.
Two different methods for compositing exist:

- front-to-back compositing and
- back-to-front compositing.

Using the front-to-back compositing scheme, the viewing rays are traversed from the eye point into the volume.

$$C_{dst} \leftarrow C_{dst} + (1 - \alpha_{dst})C_{src} \tag{2.9}$$

$$\alpha_{dst} \leftarrow \alpha_{dst} + (1 - \alpha_{dst})\alpha_{src} \tag{2.10}$$

By reversing the traversal direction, the back-to-front compositing scheme is obtained

$$C_{dst} \leftarrow (1 - \alpha_{src})C_{dst} + C_{src} \tag{2.11}$$

2.3.2 Volume Rendering Techniques

There are two subclasses of direct volume rendering algorithms, object-order and image order algorithms. The difference between object-order and image-order algorithms is that image order algorithms attempt to find all the voxels which affect a single pixel in the image plane, while object order algorithms attempt to find for each single voxel the pixel it belongs to. Figure 2.3 shows the basic difference between these two concepts.

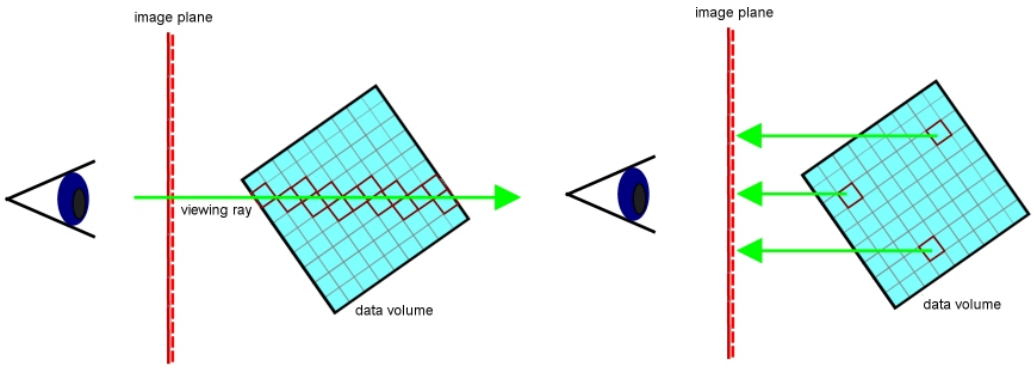


Fig. 2.3: The concept of a) image-order volume rendering methods and b) object-order volume rendering methods.

Object-Order Approaches

A typical object-oder algorithm is *splatting*, introduced by Westover [49]. In this algorithm, each voxel in the volume is projected back onto the image plane in the form of a 3D reconstruction kernel. This 3D kernel then produces a 2D footprint on the image plane. The process of projecting the voxel onto the image plane can be imagined like throwing a snowball at a wall: upon impact, the snowball spreads out around the point where it hit the wall. The 2D footprint represents the spreading out of the voxel as it hits the image plane. Typically, a Gaussian distribution function is used as a footprint. Based on the distribution function, color and opacity are calculated for each pixel that is affected by the footprint and the values of overlapping footprints are blended together. Splatting is a flexible algorithm which can be applied to a traditional volume dataset based on a uniform grid, which is traversed in a voxel-by-voxel fashion, as well as datasets which consist of arbitrarily distributed data points.
Another object-order approach is *cell projection* [41]. The basic principle of this algorithm is similar to splatting, but instead of voxels, cells are projected back onto the image plane. Cell projection is used for datasets represented by complex unstructured cell meshes. The algorithm works by first sorting all the cells according to their distance to the camera, then generating a set of triangles which represent the projection cell for every cell, and finally rendering the triangles in the right order.

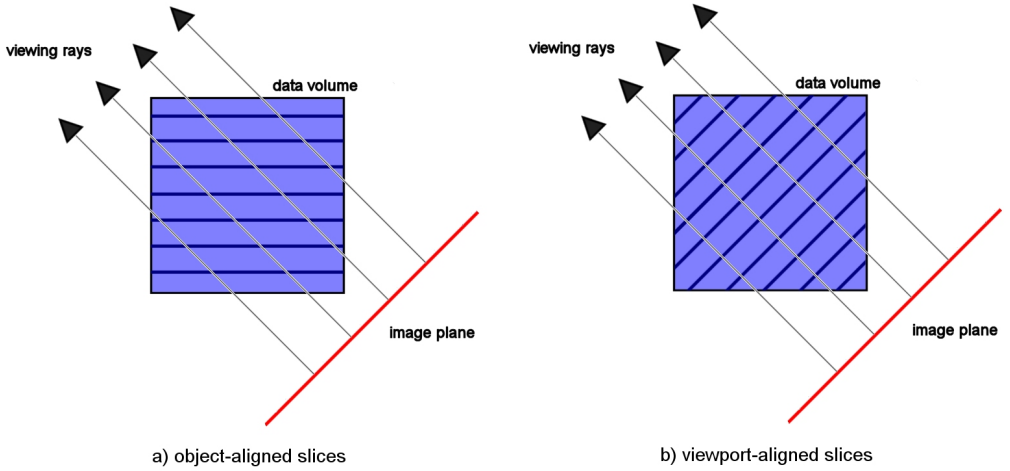


Fig. 2.4: The concept of a) object-aligned and b) viewport-aligned slices in texture slicing.

Texture slicing is the dominant object-order method for GPU-based volume rendering. The basic principle is to use 2D slices that are located in the 3D object space to sample the volume. These slices are projected onto the image plane and combined according to the compositing scheme. Texture slicing is directly supported by the graphics hardware as it only needs texture support (used for storing the dataset and the transfer functions) and blending (used for compositing). There are three different methods for implementing texture slicing:

- using 2D textures with object-aligned slices,
- using 3D textures with viewport-aligned slices,
- using 2D textures with viewport-aligned slices.

Object-aligned slicing aligns the slices aligns the stack of planes with one of the principal axis of the volume. For an arbitrary viewing direction, the slices are always aligned with the main direction which points in the viewing direction. When the viewing direction changes, the direction the slices are aligned with can change rather abruptly, which causes artifacts in the final image. Figure 2.4.a depicts the concept of object-aligned slices. When using viewport-aligned slices the planes are always oriented in the direction of the viewer, thus no artifacts are caused by a rotation of the volume. On the downside, it is harder to create the geometry of the slices because they are not all quads such as in object-aligned

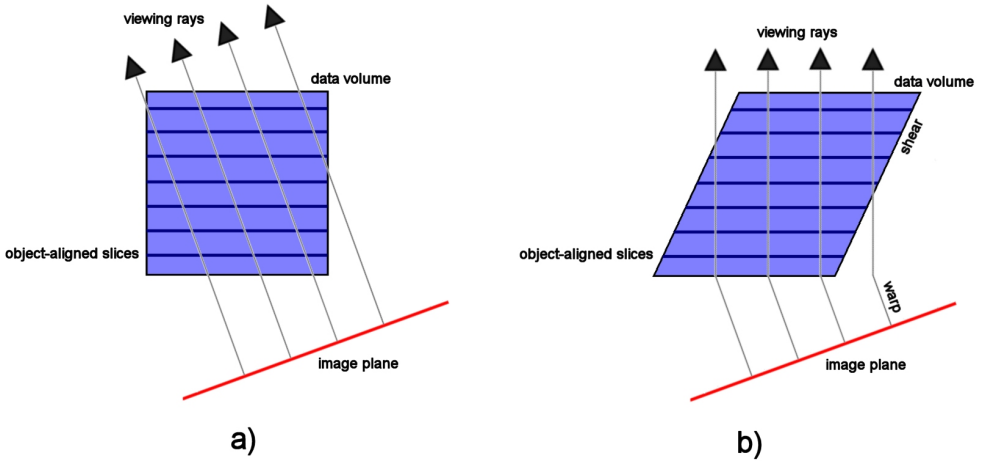


Fig. 2.5: Shear-warp volume rendering.

slicing. In Figure 2.4.b, viewport-aligned slices are shown.
Shear-warp factorization is a an object-order algorithm that was first introduced by Lacroute and Levoy [26]. It is the fastest known purely software based volume rendering algorithm, however, the quality of the resulting images is often not as good as the quality of images generated by other volume rendering techniques. The principle behind shear warp factorization is to transform the volume into an intermediate coordinate system called the sheared object space. In this space, all viewing rays are parallel to the third coordinate axis (see Figure 2.5.b). For each of the three coordinate axes a separate stack of slices has to be computed; the stack which is most perpendicular to the viewing direction is chosen and transformed. The slices are then blended together back-to-front, and a final warping step eliminates distortions in the image and performs the rotation around the viewing axis.

Image-Order Approaches

The most commonly used image-order algorithm is *volume ray casting*, which was first introduced by Levoy [28]. In this algorithm, a ray is shot into the data volume for every pixel in the image plane. The volume data is resampled at a number of

evenly spaced locations (sample points) along the ray. At each sample point, color and opacity are determined by interpolating the surrounding area based on the transfer function. In the last step the sample points are blended with each other and the background, typically using the front-to back compositing scheme, as the rays originate from the image plane. Sometimes, more than one ray is shot into the volume for each pixel in the image plane; this is called super sampling. Ray casting is the most important method for CPU volume rendering, and has been used for quite sometime.
Since the development of faster GPUs with a higher functionality, *GPU ray casting* has become popular. GPU ray casting can be either implemented by multipass rendering or by single-pass rendering. In single-pass GPU ray casting, advancing the viewing rays, accessing the data values that correspond to the ray position, and compositing the values happens in a loop in the fragment program. Multipass ray casting was developed before single-pass rendering, because the functionality that allowed a loop to be carried out in a fragment program was not available at that time. In multipass GPU ray casting, advancing the viewing rays is done by the CPU, by rendering the front faces of the volume's bounding box several time in a row while shifting the previous position along the ray direction. The accumulated colors and opacities are passed along as textures.

2.4 Transfer Functions

In direct volume rendering, using an appropriate transfer function is crucial for obtaining useful and informative result images. The rendering equation assumes that the optical properties, e.g. emission and absorption coefficients for each point in the data volume are known - however, a typical volume dataset contains abstract scalar data values that describe some spatially varying physical property. In reality, obtaining emission and absorption coefficients from a data volume can not be done "naturally", instead the user must decide how the different structures in a given dataset should look like by assigning the aforementioned coefficients to all the data points using arbitrary mappings. A transfer function is a way to define such a mapping from data values to optical properties. In general, the mapping

done by a transfer function can be described by the following equation:

$$tf(\vec{h}(x,y,z)) \rightarrow \vec{p} \tag{2.12}$$

In Equation 2.12, the function tf stands for the transfer function while the function vector $\vec{h}$ entails all the different functions which are applied to a specific point with position (x, y, z) in the data volume. The complexity of a transfer function depends on the dimension of the vector $\vec{h}$: if the vector has dimensionality n, the transfer function is called n-dimensional transfer function. The vector $\vec{p}$ describes the different optical properties which are assigned by the transfer function. Typical optical properties that are defined by a transfer function are

- the emitted radiance, which is represented by a RGB value
- the opacity which represents the absorption, represented by a scalar value between 0 and 1

One-dimensional transfer functions are the most commonly used transfer functions, as they are easy to generate manually. They use the scalar value at a particular position (x, y, z) as input function. In Equation 2.13, a typical 1D transfer function is given:

$$tf(f(x,y,z)) \rightarrow (\vec{c}, \alpha) \tag{2.13}$$

The function $f(x, y, z)$ return the scalar value at position (x, y, z). The optical properties that are assigned by the transfer function are the opacity value α, and emitted radiance $\vec{c}$, which consists of the three color components red, green, and blue. However, higher dimensional transfer function are also possible.

2.4.1 Transfer Function Design

Manually Generated Transfer Functions

Trial and error is used commonly for finding an appropriate transfer function for a certain dataset. This approach is easy to handle, but can be very time consuming. The transfer function is typically generated by the user, with the help of a transfer

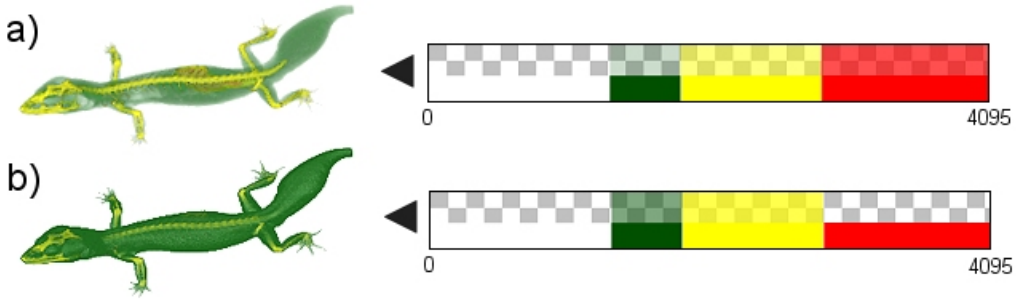


Fig. 2.6: The same dataset, rendered with two different one-dimensional transfer functions. The transfer functions are shown next to their corresponding image

function editor. Afterwards, the dataset is rendered with the previously specified transfer function; If the resulting image does not meet the users expectations, the transfer function is modified.

The problem with this approach is that the user has lots of possibilities for generating and modifying transfer functions, and it might take a huge number of steps to find an appropriate one. Another manual approach is the generation of a variety of random transfer functions. Then the user selects the transfer function which yields the best result. This procedure is repeated until the transfer function converges against a useful setup. He et al. [15] introduce an approach for assisting the user in exploring appropriate transfer functions by treating the search as a parameter optimization problem and addressing it with stochastic search techniques.

In order to make the manual transfer function setup easier, some methods which analyze the data volume and automatically identify potentially interesting structures within it have been developed. In an approach developed by Bajaj et al. [1], the so called contour spectrum is used to achieve this: the contour spectrum locates scalar values which correspond to uniquely shaped iso surfaces, so that the user can select an appropriate iso value. Potts and Möller [36] propose a logarithmically scaled transfer function editor, and argue that such a scale relates the height of the transfer function to the rendered intensity of a region of particular density in the volume almost directly, resulting in much improved, simpler manual transfer function editing. Takanashi et al. [45], on the other hand, introduce an interactive classification technique for volume data, called ISpace, which uses

Independent Component Analysis (ICA) and a multidimensional histogram of the volume data in a transformed space. The result of this technique is an opacity transfer function defined for rendering multivariate scalar volume data.
According to Brodlie et al. [4], the fundamental drawback of transfer function based methods is that they are injective, therefore the separation of features with the same statistical properties is impossible. For example, tissue can be separated from bone in CT scans, but a single bone cannot be separated from other bones. The flexibility of transfer functions can be improved by extending them to higher dimensions. The most commonly used type are two-dimensional transfer functions, which depend on both the scalar value and its first-order derivate, the gradient magnitude. Other variations take the second-order derivate into account [18, 23]. Transfer functions of higher dimensions are also sometimes proposed [46, 24]. While multi-dimensional transfer functions offer more degrees of freedom and flexibility, the already time-consuming process of manually setting up an appropriate transfer function gets even more complex with each extra dimension that gets added. This lead to the development of several different methods for (semi)-automatically generating transfer functions.

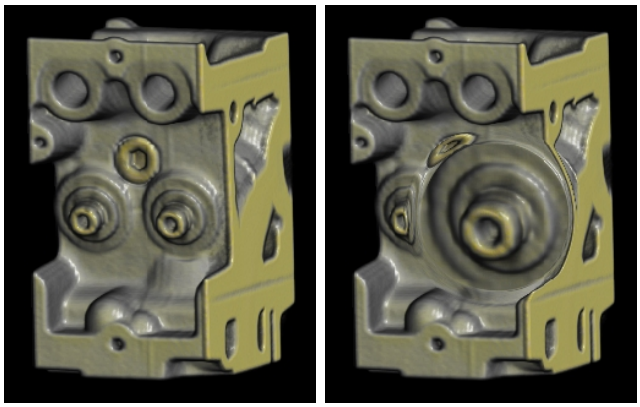
(Semi)-Automatic Transfer Function Generation

A common approach is to extract critical iso surfaces from the data volume, analyze the topological structure of these surfaces and create a transfer function based on the analysis. The topological method proposed by Fujishiro et al. [11] is based on so called *hyper reeb graphs*.
A different method was introduced by Kindlmann and Durkin [22]. This method was designed specifically for volume datasets where the regions of interest are the boundaries between different materials, which actually constitute a large class of volume datasets. A transfer function which makes boundaries readily visible can be generated from the relationship between three quantities: the data value and its first and second directional derivatives along the gradient direction. The relationship between these quantities is captured in a so called *volume histogram* throughout the volume in a position independent fashion. The histogram volume is a three dimensional representation of the three aforementioned properties, so

that each property has its own coordinate axis, where the position of each bin in the histogram volume represents the three values at a small range and the value of the bin itself represents the number of voxels, which correspond to the three values of the bin's position.
Brodlie et al. [4] developed a method for the automatic setup of multi-dimensional transfer functions by adding spatial information to the histogram of a volume. The method is based on the fact that a feature by definition is a spatially connected region in the volume domain with a unique position and certain statistical properties, therefore only using the statistical properties - the histogram of the data volume - ignores an important part of the definition of a feature. The method produces a two-dimensional transfer function, where the spatial information has been used to derive the color, and a combination of the statistical and spatial information is used to setup the opacity. The opacity is calculated in such a way that uninteresting regions with low gradients are blended out and the material boundaries with high gradients are emphasized. The color is determined by separating the data volume into a group of different features by the use of an heuristic - this is done because doing an exact segmentation of the dataset would be too time consuming. The accuracy of this heuristic segmentation can be controlled by the user by specifying a control parameter. After the features have been determined, each of them is assigned an individual color.

2.5 Focus+Context in Direct Volume Rendering

According to Card et al. [6], Focus+Context start from three premises: First, the user needs both overview (context) and detail information (focus) simultaneously. Second, information needed in the overview may be different from that needed in detail. Third, these two types of information can be combined within a single (dynamic) display, much as in human vision. Volume datasets, especially those used for visualizing medical images often contain relatively small regions of interest, occluded by the surrounding tissue. Using transfer functions, it is possible to hide the tissue that surrounds the area of interest, but in doing so, the context is lost. Therefore, during the past few years, Focus+Context display in direct volume rendering has become an important area of research.



(a) dataset without distortion (b) dataset with distorted focal region

Fig. 2.7: A volume dataset rendered with a distortion based Focus+Context Technique, taken from [27]

Traditional Focus+Context visualization techniques, e.g. magnification lenses have been adapted to work with direct volume rendering [48, 27]. Figure 2.7 shows a volume dataset where an area has been magnified with the help of a nonlinear distortion technique. Non-photorealistic or illustrative rendering methods are a very active field of research. In many technical or medical textbooks, illustrations are used to visualize complex relationships and procedures. Approaches for illustrative volume visualization employ non-photorealistic rendering techniques to mimic the style of traditional illustrations.

In volume visualization, Levoy [28] was the first to propose modulation of opacity using the magnitude of the local gradient. This is an effective way to enhance surfaces in volume rendering by suppressing homogenous regions. Based on this proposal, Ebert and Rheingans [8] developed a collection of volume illustration techniques that adapt and extend non-photo realistic rendering techniques to volume objects, e.g. feature enhancement, feature halos and adding depth and orientation cues, and proposed to apply these methods locally for regional enhancement.

Lu et al. [30] present a framework for an interactive direct volume illustration system that simulates stipple drawings.

Another approach, inspired by cut-away views, which are commonly used in

Fig. 2.8: Importance driven rendering example, rendered from different angles; Taken from [47]

technical illustrations, was introduced by Viola et al. [47]. With cut-away views, less important parts of a scene are suppressed or completely removed to reveal the more important underlying information. The proposed method assigns different *object importances* to the parts in the data volume, which are used to determine which structures should be readily discernible and which structures are less important. In those image regions, where an object occludes more important structures it is displayed more sparsely than in those areas where no occlusion occurs; Thus the objects of interest are clearly visible. For each object several levels of sparseness are specified.

Hauser et al. [14] propose *two-level volume rendering*, a technique which allows for selectively using different rendering techniques for different subsets of a three dimensional dataset. Different structures within the dataset are rendered locally on an object-by-object basis by either direct volume rendering, maximum intensity projection, surface rendering, value integration (x-ray-like images), or non-photorealistic rendering. Globally all the results of subsequent object renderings are combined in a merging step (usually compositing). This allows to selectively choose the most suitable technique for depicting each object within the data, while keeping the amount of information contained in the image at a reasonable level.

Lum and Ma [31] present a multi-dimensional transfer function method for enhancing surfaces which works through the modification of surface shading instead of the variation of opacity. The technique uses a lighting transfer function that takes into account the distribution of values along a material boundary.

The approach of using distance to emphasize and deemphasize different regions

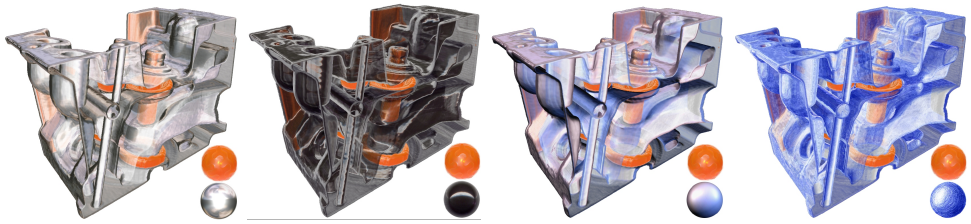


Fig. 2.9: Engine block rendered using different style transfer functions. The lit sphere maps used in the transfer function are depicted at the bottom right corner of each image. Image taken from [5]

of a data volume was introduced by Zhou et al. [50]. The method uses the distance between voxels in the data volume and the focus of interest in order to determine the importance of the current voxel for the whole rendering.

The concept of *style transfer functions* was introduced by Bruckner and Gröller [5]. Instead of traditional transfer functions, style transfer functions represent specific styles captured from existing artwork. Capturing the specific styles is based on the *lit sphere shading model* [42]. The basic idea of this model is to capture color variations of an object as a function of normal direction. As a sphere provides coverage of the complete set of unit normals, an image of a sphere under orthographic projection will capture all such variations on one hemisphere.

Chapter 3

Genetic Algorithms

In this chapter, an overview of *genetic algorithms* is given. This is important, as genetic algorithms were used in the second part of this thesis in order to automatically compute a transfer function for a volume data set with desired attributes.
A genetic algorithm is a search technique used in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. They are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination). Genetic algorithms are used in many different areas, such as molecular biology, Pattern recognition and data mining, routing and scheduling, and robotics.
In section 3.1 a brief historical overview of genetic algorithms is given. Section 3.2 takes a closer look at the methodology of genetic algorithms. In particular, different methods for crossover and selection are presented.

3.1 Historical Overview

The earliest instances of genetic algorithms appeared appeared in the late 1950s and early 1960s, programmed on computers by evolutionary biologists who were explicitly seeking to model aspects of natural evolution [2, 10]. By 1962, researchers such as Friedman and Bremermann [9] had all independently developed evolution-inspired algorithms for function optimization and machine learning, but their work attracted little follow up.
A more successful development in this area came in 1965, when Rechenberg [38]

introduced a technique he called evolution strategy, though it was more similar to hill-climbers than to genetic algorithms. In this technique, there was no population or crossover; one parent was mutated to produce one offspring, and the better of the two was kept and became the parent for the next round of mutation. Later versions introduced the idea of a population.
The next important development in the field was introduced in 1966 by Fogel et al. [12] who introduced in America a technique called evolutionary programming. In this method, candidate solutions to problems were represented as simple finite-state machines, and like Rechenberg's evolution strategy, their algorithm worked by randomly mutating one of these simulated machines and keeping the better of the two. Genetic algorithms in particular became popular through the work of Holland [19] in the early 1970s. His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next generation, known as *Holland's Schema Theorem*.
These foundational works established more widespread interest in evolutionary computation. By the early to mid-1980s, genetic algorithms were being applied to a broad range of subjects, from abstract mathematical problems like bin-packing and graph coloring to tangible engineering issues such as pipeline flow control, pattern recognition and classification, and structural optimization. At first, these applications were mainly theoretical. Over time, however, genetic algorithms migrated into the commercial sector, their rise fueled by the exponential growth of computing power and the development of the Internet.

3.2 Methodology

Figure 3.1 gives a general overview of the principle of genetic algorithms. Basically, in genetic algorithms, an initial population of abstract representations of a number of candidate solutions evolves towards better solutions by repeatedly applying the techniques

- Selection
- Mutation

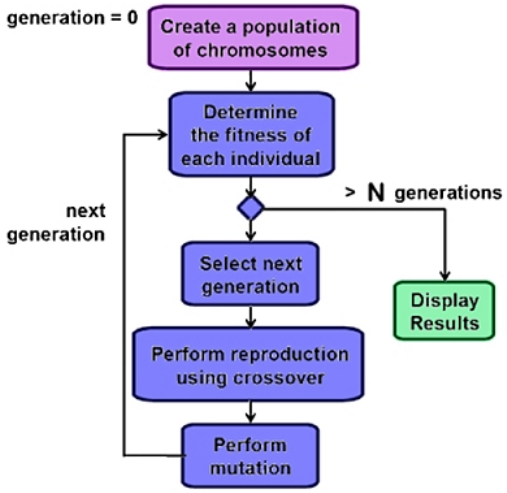


Fig. 3.1: An overview of genetic algorithms.

- Crossover

. After each evolution step, a new generation is formed. This process is repeated until a certain termination criterion is met, e.g. a maximum number of generations is reached, or the individuals in the current generation are ”good enough”. The most common way to abstractly represent the solutions is by encoding them in binary as strings of 0s and 1s, although other encodings are also possible. The initial population can either be generated randomly, or by seeding a particular area of the search space, where optimal solutions are likely to be found. The initial population the evolves towards better solution in a certain number of generations. In each generation, the fitness of the individuals within the population is evaluated, multiple individuals are stochastically selected from the current population based on their fitness. The selected solutions are then modified by using mutation and recombination and form a new population, which is used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached. Two important things that need to be defined in order to use a genetic algorithm are

- a genetic representation of the solution domain and
- a fitness function to evaluate the solution domain

3.2.1 Methods of Representation

Using same-length arrays of bits as genetic representation of the solution domain is the most commonly used method of representation as it facilitates the crossover operation. When using arrays of differing lengths, crossover is more complex. Another, similar approach is to encode solutions as arrays of integers or decimal numbers, with each position again representing some particular aspect of the solution. This approach allows for greater precision and complexity than the comparatively restricted method of using binary numbers only. Instead of numbers, letters can also be used to represent aspects of the solution. Another strategy, developed by Koza et al. [25] and called genetic programming, uses tree-like representations. In this approach, random changes can be brought about by changing the operator or altering the value at a given node in the tree, or replacing one subtree with another.

3.2.2 Selection

The *fitness function* is defined over the genetic representation and serves as a measure for the quality of the solutions. It is therefore always problem specific. For some problems, it is very difficult to find an appropriate fitness function, in some cases it is even impossible. In such a case, interactive genetic algorithms can be used: instead of using a fitness unction, the solutions are evaluated by humans. In every generation, a number of solutions are selected based on the defined fitness function and subsequently used to generate the next generation. There are many different techniques which a genetic algorithm can use to select the individuals to be copied over into the next generation. Some of these methods are mutually exclusive, but others can be and often are used in combination. Some of the most commonly used techniques are explained below: *Elitist selection* selects the fittest members of each generation. This selection method is rarely used in practice, as selecting only the fittest members in each generation can lead the

algorithm to converge towards a local optimum. Instead, fitness proportionate selection algorithms typically select fitter individuals with a higher likelihood, but it is not certain that they are selected. The selection pressure is ratio of the best individual's selection probability to the average selection probability of all individuals in the selection pool.

Roulette-wheel selection is a form of fitness-proportionate selection in which the chance of an individual's being selected is proportional to the amount by which its fitness is greater or less than its competitors' fitness. The probability of selecting a single solution can be described by 3.1:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}. \tag{3.1}$$

Conceptually, this can be represented as a game of roulette - each individual gets a slice of the wheel, but more fit ones get larger slices than less fit ones. The wheel is then spun, and whichever individual owns the section on which it lands each time is chosen.

Scaling selection: As the average fitness of the population increases, the strength of the selective pressure also increases and the fitness function becomes more discriminating. This method can be helpful in making the best selection later on when all individuals have relatively high fitness and only small differences in fitness distinguish one from another.

Tournament selection: In this selection method, subgroups of individuals are chosen from the larger population, and members of each subgroup compete against each other. Only one individual from each subgroup is chosen to reproduce. Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Hierarchical selection: Individuals go through multiple rounds of selection each generation. Lower-level evaluations are faster and less discriminating, while those that survive to higher levels are evaluated more rigorously. The advantage of this method is that it reduces overall computation time by using faster, less selective evaluation to weed out the majority of individuals that show little or no promise, and only subjecting those who survive this initial test to more rigorous and more

computationally expensive fitness evaluation.

3.2.3 Reproduction

Once selection has chosen fit individuals, they must be randomly altered in hopes of improving their fitness for the next generation. The two basic strategies which are used to accomplish this are mutation and crossover.
The concept of mutation is simple: Just as mutation in living things changes one gene to another, so mutation in a genetic algorithm causes small alterations at single points in an individual's code.
On the other hand, crossover entails choosing two individuals to swap segments of their code, producing artificial "offspring" that are combinations of their parents. This process is intended to simulate the analogous process of recombination that occurs to chromosomes during sexual reproduction. Figure 3.2 shows the four most commonly used crossover techniques: Using *one-point crossover* (Figure 3.2.a), one point common on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms.
Two-point crossover (Figure 3.2.b) calls for two points to be selected on the parent organism strings. Everything between the two points is swapped between the parent organisms. Another crossover variant, the *cut and splice approach* (Figure 3.2.c), results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point. Finally, using *Uniform crossover* (Figure 3.2.d), the value at any given location in the offspring's genome is either the value of one parent's genome at that location or the value of the other parent's genome at that location, chosen with 50/50 probability.

3.2.4 Termination

The process of selection and recombination is repeated until a certain termination condition is met. Termination conditions that are commonly used include:

- A solution is found that satisfies minimum criteria.

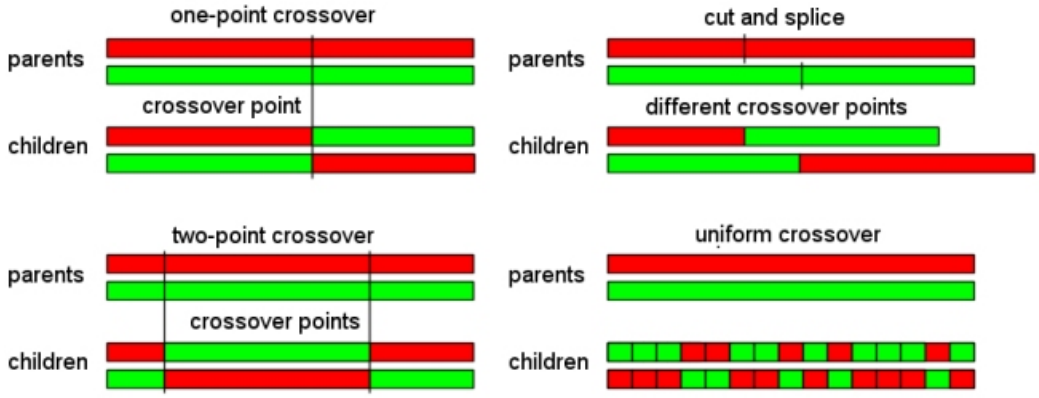


Fig. 3.2: The basic principles of the different crossover methods. In Figure c), using the cut-and-splice approach, the length of the offspring differs from the length of the parents, while in Figures a), b) and d) the length stays equal

- A fixed number of generations is reached.
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results.
- An allocated budget e.g. computation time or money is reached.

Combinations of the above reasons can be used as well.

Chapter 4

Visibility Histograms

This chapter introduces the idea of so called *visibility histograms*. The basic idea behind a visibility histogram is to show for a given volume dataset that is rendered with a specific transfer function, how much each individual density value in the dataset contributes to the final image.

Basically, a histogram is a graphical display of tabulated frequencies. It is the graphical version of a table that shows what proportion of cases fall into each of several or many specified categories. The categories are usually specified as non-overlapping intervals of some variable, and are adjacent. Histograms are often used in computer graphics as a tool for visualizing information.

One area where histograms are used in computer graphics, is in the form of *color histograms* [34]. A color histogram is a representation of the distribution of colors in an image, derived by counting the number of pixels of each of a given set of color ranges in a typically two-dimensional or three-dimensional color space. They can be built from images in various color spaces, e.g. RGB or rg chromaticity. A histogram of an image is produced first by discretization of the colors in the image into a number of bins, and counting the number of image pixels in each bin. Such a color histogram provides a compact summarization of the distribution of data in an image. The color histogram of an image is relatively invariant with translation and rotation about the viewing axis, and varies only slowly with the angle of view. By comparing histograms signatures of two images and matching the color content of one image with the other, it is possible to recognize an object of unknown position and rotation within a scene. In photography, color histograms are frequently used in digital cameras for estimating the scene illumination, as part of the camera's automatic white

balance algorithm. Color Histograms are also commonly used as appearance-based signature to classify images for content-based image retrieval systems.
Another area where histograms are often used, are image editors. Such programs often have provisions to create an image histogram of the image being edited, i.e. a brightness histogram plots the number of pixels in the image with a particular brightness value. Algorithms in the digital editor allow the user to visually adjust the brightness value of each pixel and to dynamically display the results as adjustments are made. This way, improvements in picture brightness and contrast can be obtained.
In this chapter, the basic idea behind visibility histograms will be explained. Section 4.1 explains the mathematical background of the term ”visibility”, while section 4.2 deals with computing a visibility histogram for a data set, and explains which factors influence the visibility distribution for a particular dataset. Section 4.3 explains how visibility histograms can be used to analyze and visualize certain aspects of volume datasets.

4.1 Visibility

The basic idea behind a visibility histogram is to show how much the points in a volume data set with a certain density value affect the final image, i.e. how ”visible” they are. In chapter 2.3, different methods that are commonly used in direct volume rendering were explained. In order to understand what precisely is meant by the term visibility, a closer look needs to be taken at the most popular image-order method for volume rendering, volume ray casting.
The basic idea behind volume ray casting is to shoot viewing rays through the data volume for every pixel in the image plane, and samples are taken at evenly spaced points (see Figure 4.1.a) and blended together using the front-to-back compositing scheme defined by Equation 2.10 to calculate the intensity value for the pixel in the image plane from which the ray originated.
Based on the transfer function which assigns opacity values to the densities in the data volume, the accumulated opacity increases after every sample point, which means that the next sample points have less of an influence on the final intensity value of the pixel. Figure 4.1.b shows how the accumulated opacity increases as

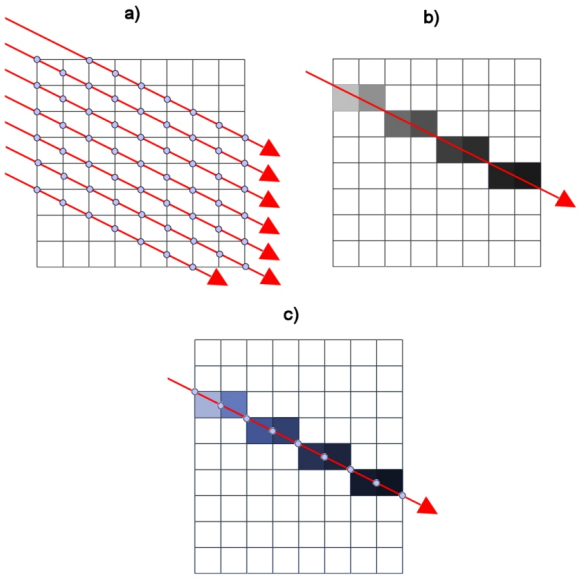


Fig. 4.1: a) In volume raycasting, rays are shot through the data volume and the color and opacity are evaluated at evenly spaced sample points. b) As the rays pass through the volume, the accumulated opacity along every single ray increases. c) The contribution of an interval between two sample points to the accumulated opacity is defined as the visibility of the interval.

the ray passes deeper into the volume. Between two sample points s_{n-1} and s_n, the amount by which the opacity increases as the ray travels through the space between the sample points can be described as

$$op(s_{n-1}, s_n) = \alpha_{s_n} - \alpha_{s_{n-1}} \tag{4.1}$$

where α_{s_i} denotes the accumulated opacity at sample point i. The value $op(s_n, s_{n+1})$ in Equation 4.1 is the visibility of sample point s_n. In chapter 2.3, the formula for front-to-back alpha compositing was expressed as

$$\alpha_{dst} \leftarrow \alpha_{dst} + (1 - \alpha_{dst})\alpha_{src} \tag{4.2}$$

Using the front-to-back alpha compositing formula, the visibility can be calculated as

$$op(s_{n-1}, s_n) = (1 - \alpha_{DST}) * \alpha_{SRC} \tag{4.3}$$

The opacity values in a given alpha transfer function are defined with respect to a particular sampling rate which is assumed to be constant throughout the volume. A problem arises when the sampling rate needs to be changed: the discretized opacity contributions need to be modified because their values depend on the sampling distance. The alpha values for two different sampling distances are related by

$$\widetilde{\alpha} = 1 - (1 - \alpha)^{\frac{\Delta\tilde{x}}{\Delta x}}. \tag{4.4}$$

Equation 4.4 describes the *opacity correction* which needs to be made whenever a sampling distance different from the original sampling distance for which the alpha transfer function was designed needs to be used. It allows for varying sampling rates even though the optical properties are only given with respect to a discrete version of the volume rendering integral.

4.2 Visibility Histograms

In Chapter 4.1 the mathematical meaning of ”visibility” for a single sample point in a data volume was explained. In order to calculate a visibility histogram for a given dataset, the visibility of all sample points with the same scalar value is added together. Intuitively, this value is a measure for the influence that the data points with a certain density have on an image that is created by rendering the dataset with a volume rendering method such as ray casting.
The visibility value for a certain density is normalized by the number of sample points in the volume with that particular density. Thus, the value that is displayed in a visibility histogram for a particular density is a measure of how much influence, on average, a data point with that density has on the final image.
Something that is not very intuitive about these visibility values is that increasing the opacity value for a particular density, while decreasing the opacity values for all the other density values in a data volume does not necessarily achieve a high increase in the visibility of the data points with particular density: Increasing the opacity value for a specific scalar value will make the data points with that density which are closest to the image plane more visible - however, these data points may occlude many other data points with the same density and decrease their visibility.

Achieving an average visibility that is close to 1 is not possible most of the time. For a single ray traversing the data volume, the maximum visibility achievable for a certain scalar value on that ray is $1/n$, with n being the number of sample points with the scalar value. In order to achieve this value, the alpha value for the scalar value in question would have to be set to 1, while the alpha value for all the scalar values of sample points that are encountered before the sample points with the scalar value in question have to be set to zero. The visibility histogram for this ray would then be zero for all scalar values other than the scalar value that is of interest. In reality, such a visibility distribution is not very useful, since hiding all scalar values other than one particular value would generally not generate particularly meaningful images. In general, a range, or even several ranges of scalar values within a dataset are of interest, leading to more occlusions and a lower average visibility of the area of interest. Also, hiding the scalar values that do not belong to the area of interest destroys context information. Given a particular dataset, the visibility distribution is dependent on two factors:

- the transfer function
- the viewing direction

4.2.1 Influence of the Transfer Function

A visibility histogram depicts the visibility distribution for a given data set only with respect to a certain transfer function. Every time the alpha transfer function changes, the visibility histogram has to be computed anew. Figure 4.2 shows how different transfer functions applied to the same dataset can lead to different visibility distributions.

4.2.2 Influence of the Viewing Direction

A visibility histogram for a particular dataset is dependent on the direction from which the dataset is looked at, i.e. the direction of the viewing rays which traverse the data set. When rotating the dataset, the sample points that are encountered along the viewing rays change, causing a change in the visibility distribution. Even rotating a dataset by 180° will produce a different visibility distribution (see

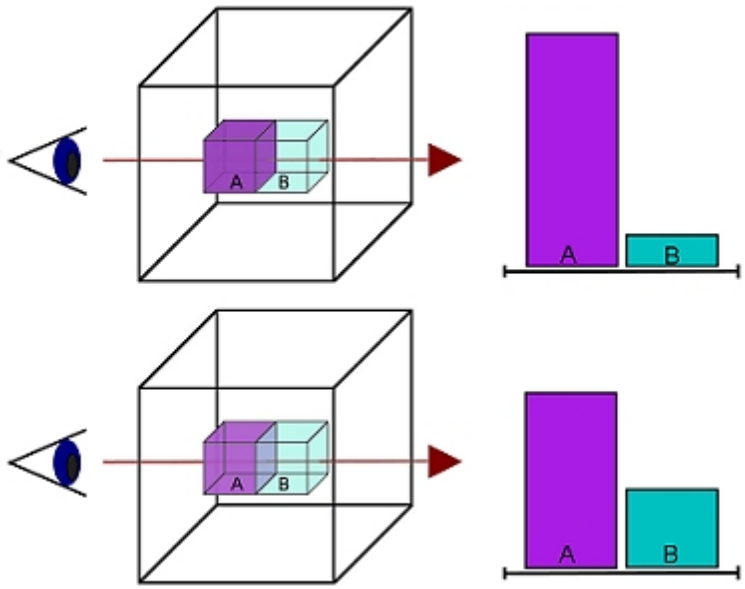


Fig. 4.2: The influence of differing transfer functions on the visibility distribution.

Figure 4.3).

4.3 Visibility Histograms as Tool for Analyzing Volume Datasets

Since the visibility histogram of a volume dataset is dependent on the viewing direction and the transfer function it can be used to analyze the structure of the dataset.
Rendering a dataset with a fixed viewing direction, but differing transfer functions can be used to gain insight into the internal structure of the data set, particularly how different tissues occlude each other and interfere with each others visibility. For a dataset that is viewed from a certain direction, calculating the visibility histograms for several transfer functions where the alpha values for a range of scalar values vary slightly can be used to analyze the effect that this particular data range has on the overall visibility distribution.
On the other hand, when rendering a dataset from different angles, the overall

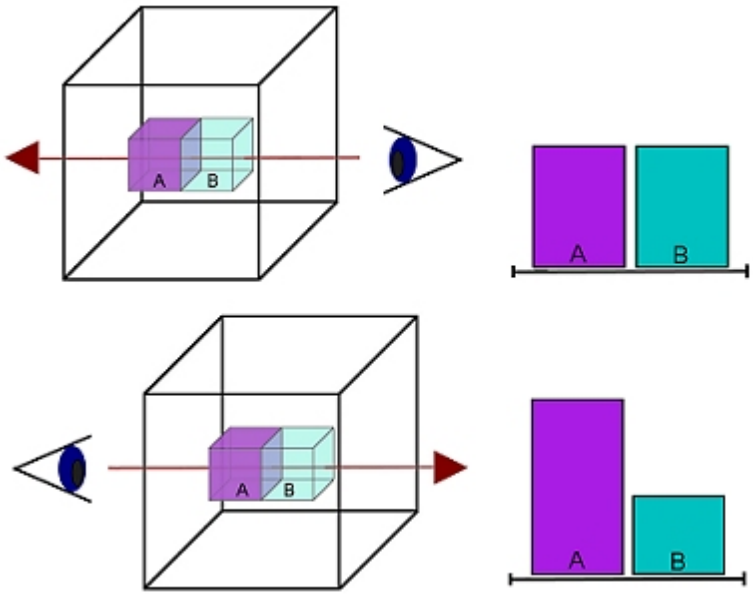


Fig. 4.3: The influence of the differing viewing direction on the visibility distribution.

visibility distribution tends to change rather slowly with the viewing direction. Visualizing the resulting visibility distributions makes it possible to spot patterns and draw conclusions on the internal structure, i.e. if the dataset is symmetric, the resulting visibility distributions will reflect this. For those scalar ranges with a rather uniform distribution throughout the dataset, the visibility distributions will likely not vary a lot while the viewing direction changes. On the other hand, if a certain range of scalar values has a high visibility only from certain angles, it points toward an asymmetry in the data set.

Chapter 5

Visibility Manipulation

In the previous chapter, visibility histograms were introduced. It was shown, how given a particular transfer function which assigns an opacity value to each data point in a volumetric dataset, the average visibility of all the data points with a particular scalar value can be computed. This chapter deals with the reverse operation: instead of computing the visibility histogram for a transfer function, we want to find a transfer function which generates a specified visibility histogram when applied to a dataset.
In section 5.1, an overview of transfer functions in general is given. Section 5.2 shows how theoretically, a transfer function could be computed for a particular visibility distribution, and explains why an approach that does not exactly solve the problem, but approximates it, is preferable over finding the exact solution. Section 5.3 deals with how genetic algorithms can be used to find a suitable transfer function for a given visibility histogram, whereas section 5.4 explains how a heuristic approach could be used to find a suitable transfer function.

5.1 Automatic Transfer Function Generation with Visibility Histograms

Trying to find a suitable transfer function for a volumetric dataset is typically done with an idea in mind of what the final image generated should look like: certain areas in the data volume are more interesting than others, and are thus desired to be more visible than others. The data points which represent those areas typically are assigned a higher opacity value, while those data points belonging to areas which are of lesser interest are assigned a lower opacity value or may even be

made fully invisible. In chapter 4, the connection between the opacity values which are assigned to the data points in a volumetric dataset, and their visibility in the generated image was explained. Given a particular transfer function, it can be computed how visible each of the scalar values in the dataset is. If a way could be found to automatically generate a transfer function that matches a a user-specified visibility distribution - i.e. a transfer function that generates the specified visibility histogram when applied to a volumetric dataset - this method could be used to facilitate the process of finding a suitable transfer function.
Given a certain 3D dataset and a transfer function that assigns every point in the dataset an opacity value, computing the visibility histogram from a certain viewing direction is a rather straightforward task, but unfortunately, the reverse operation is not easy. Given a 3D dataset that is viewed from a certain direction and a visibility histogram, there are some challenges that make automatically generating an alpha transfer that function complicated.
Assuming that a visibility distribution is given which assigns the desired average visibility V_k to every bin m_k in the histogram $M = m_k$, a transfer function needs to be found which assigns to every data point p_{xyz} in the volume an opacity value so that

$$\frac{\sum_{l=0}^{k} visibility(p_{xyz})}{k} = V_k \quad p_{xyz} \in m_k. \tag{5.1}$$

In chapter 3.1, it was mentioned that the formula for calculating the visibility of a single data point is based on the front-to-back compositing scheme (see Equation 4.3). This means that the visibility of a data point is dependent on the alpha values of all the data points preceding it on a ray which traverses the volume in the viewing direction. In order to calculate a transfer function for a given visibility distribution, it would thus be necessary to store all the points that lie on each single viewing ray in the right order. With this information, the visibility at point p_n along a viewing ray can be calculated as

$$visibility(p_n) = \sum_{i=0}^{n} (-1^{n+i} * \prod_{j=i}^{n} \alpha_j) \tag{5.2}$$

where α_j is the opacity at point p_j with $0 < j < n$ that lies on the same viewing ray as p_n. Combining equations 5.1 and 5.2 gives a system of k (k being the

number of bins in the visibility histogram) nonlinear equations, and as much variables as there are different opacity values. By solving the resulting system of equations, the appropriate transfer function for a desired visibility distribution could be found - assuming that the system of equations has a solution.
Assuming a 3D transfer function which assigns an opacity value to each data point based on the position $p(x, y, z)$ of the data point, and a visibility histogram with a separate bin for every scalar value in the dataset, the amount of variables in the system of equations is a lot higher than the amount of equations; in such a case, infinitely many solutions could exist for the system. For example, the datasets used for the scope of this thesis typically contained 10^7 data points, with densities ranging from 0 to 4095.
In the typical case of an 1D transfer function, where an opacity value is assigned to the data points based on the scalar value, a system of equations could be generated where the number of equations is equal to the number of variables. This could be achieved by assigning all the scalar values that fall into a same bin within the visibility histogram the same opacity value.
If an application were to implement this method, it would require the user to specify the following things:

- the viewing direction
- the number of bins in the visibility histogram
- the desired visibility value for every bin in the histogram

The problem with this approach is that there is no guarantee that a transfer function which generates the desired visibilities can even be found: depending on the internal structure of the dataset, i.e. the way structures with different densities occlude each other, it might not be possible to find a suitable transfer function. The maximum possible visibility of a single data point is bounded by the accumulated opacity up to the data point, i.e. a data point hidden behind a very opaque region has a rather low visibility, even if it has a high associated alpha value. In the dataset in Figure 5.1, for example, the data points in the middle are occluded by the surrounding teapot. If the visibility of the teapot is set to a certain value α_i, the maximum visibility that can be reached for the interior points is $1 - \alpha_i$. For the data points that belong to the teapot the situation is even more complex, as

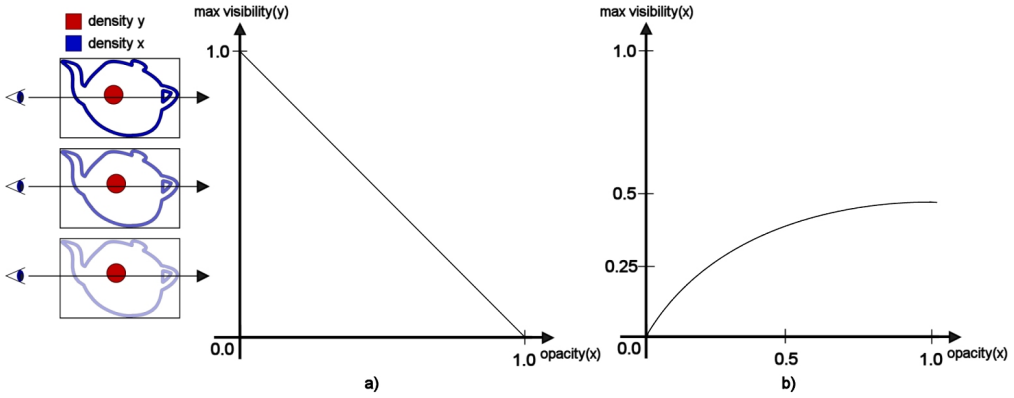


Fig. 5.1: Graph a) shows how the maximum achievable visibility of the red data points is bounded by the opacity value of the blue data points. Graph b) shows how the maximum achievable visibility of the blue data points is bounded by their own opacity values, due to self occlusion.

they have the same density and thus share the same alpha value. As the visibility of the data points at the back is bounded by the visibility of the front points, the total visibility for data points is bounded.

Allowing a user to specify a visibility distribution in absolute values is an infeasible idea, as no guarantee can be given that a matching transfer function can be found. Another approach would be to let the user specify a desired visibility distribution, and then attempt to find a transfer function which produces a visibility distribution that comes close to the user-specified visibility distribution.

5.2 Genetic Algorithm Approach

In the last section, it was shown why trying to find a transfer function that produces exactly the desired visibility histogram for a particular dataset is an infeasible approach. However, finding an exact solution is not always necessary: another approach is to search for a transfer function which generates a visibility histogram that comes ”as close as possible” to the desired visibility distribution. Thus, the problem of generating a transfer function can be viewed as search problem: The space to be searched consists of all possible transfer functions that assign an opacity value in the range of 0-1 to each point in the data volume, and

the transfer function searched for is the one where the difference between the generated visibility histogram and the desired visibility histogram is minimal. In chapter 3, genetic algorithms were introduced as a method that is commonly used to solve search problems. If a fitness function that can be used to evaluate a single transfer function, as well as a way to represent the individual transfer functions genetically can be found, a genetic algorithm can be used to find an appropriate transfer function for a desired visibility histogram.

5.2.1 Genetic Representation

In chapter 3, it was established that for a genetic algorithm, a genetic representation of the solution domain is needed. In this particular case, the solution domain consists of transfer functions which assign an opacity value to the individual points in the data volume.
When designing the genetic representation, an important thing to keep in mind is that the crossover and mutation operation have to be applied to the genetic representation. It should be easy to carry out these operations.
In section 5.1, it was explained that a transfer function can be n-dimensional, where n stands for the number of functions that are applied to a point at position (x, y, z) (see Equation 2.12). In such a case, the functions which are used as input for the transfer function, plus the transfer function itself, could be used as genetic representation, where each single function represents a gene. Using this approach, the crossover and mutation operation can be very complicated. Another approach is to calculate the exact value of the transfer function for each individual point in the volume, and interpreting the resulting values as an array. Given a 3D dataset based on a uniform grid, with nx data points in the direction of the x-axis, ny points in the direction of the y axis, and nz points in the direction of the z-axis, evaluating the transfer function yields $nx * ny * nz$ alpha values between 0 and 1, one for every point with position (x, y, z) in the data volume. These opacity values can be arranged as an array where the index of the alpha value $\alpha(x, y, z)$ for a particular point can be calculated as $x + y * nx + z * nx * ny$. Using this approach, all the transfer functions can be represented as arrays of equal length, with values between 0 and 1. As mentioned in chapter 3, arrays of equal length

are the preferred way to represent the solution domain, as crossover and mutation are extremely easy to realize that way.
The the case of the most commonly used transfer function, a 1D function which assigns an alpha value to every scalar value in the volumetric dataset, using arrays of equal lengths is once again an easy method for obtaining a genetic representation. In this case, the index where the alpha value for a particular scalar value can be calculated with the help of the scalar value itself.

5.2.2 Fitness Function

The other important thing that needs to be defined for the genetic algorithm is a fitness function to evaluate the solution domain. The solution domain consists of different transfer functions which assign an opacity value to each individual point in the 3D data set. The best approach for measuring the fitness of such a transfer function is to evaluate the visibility histogram that is generated by applying the transfer function to the data set when viewed from a fixed direction.
The goal of the genetic algorithm is to compute a transfer function which, when applied to a specific data set, generates a visibility histogram that is as similar as possible to another visibility histogram that is specified. Therefore, what is needed is a measure for the dissimilarity between two visibility histograms. The dissimilarity $d_{(H,K)}$ between histograms $H = h_i$ and $K = k_i$ is also commonly referred to as distance.
The different methods for calculating the distance between two histograms can be divided into two groups: *Bin-by-bin* dissimilarity measures compare the contents of corresponding histogram bins. *Cross-bin* dissimilarity measures also compare non-corresponding bins. For some cross-bin dissimilarity measures, a ground distance has to be defined between the bins in the histograms that don't correspond to each other, that allows for weighting the difference between two bins based on their ground distance.

Minkowski-Form Distances

In the euclidean space R^n, the distance between two points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ is usually given by the euclidean distance (also called 2-norm distance). The euclidean distance d is defined as

$$d = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{5.3}$$

Similarly to Equation 5.3 *minowski distance* of order p (also called p-norm distance) is defined as

$$d = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{5.4}$$

Interpreting a histogram with n bins as an n-dimensional point in R^n, Minkowski-form distances, especially the euclidean distance, can be used to calculate the distance between two histograms. Minkowski-form distances are an example of bin-by-bin dissimilarity measures. The Minkowski distance of order 1 is often used for computing dissimilarity between color images [44]. One drawback of the Minkowski distance is that when used as a measure for the similarity between histograms, it does not match perceptual similarity well. Only the correspondence between bins with the same index is accounted for, while information across bins is disregarded. Figures 5.2.a and 5.2.b illustrate this problem: in Figure 5.2.a, the Minkowski distance between the two histograms on the left is larger than the perceptual distance. In 5.2.b, it is shown how the desired distance should be based on correspondences between bins in the two histograms and on the ground distance between them.

Histogram Intersection

The histogram intersection, another bin-by bin dissimilarity measure, is attractive because it can handle partial matches when the areas of the two histograms - i.e. the sum over all the bins - are different. If the areas of two histograms are equal, the the histogram intersection is equivalent to the normalized 1-norm distance.

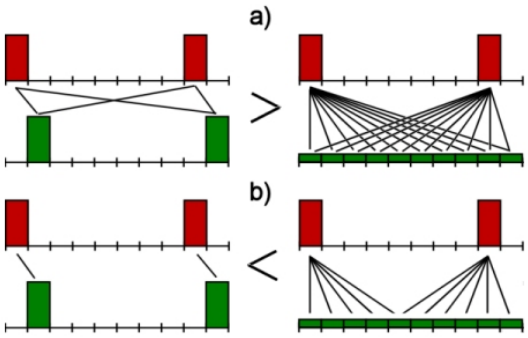


Fig. 5.2: An example where the Minkowski distance does not match the perceptual dissimilarity. In Figure a), the Minkowski distance between the histograms on the left side is greater than the Minkowski distance between the histograms on the right side. Figure b) shows the desired distance, which is based on perceptual similarity.

The histogram intersection can be calculated as

$$d_{(H,K)} = 1 - \frac{\sum_i min(h_i, k_i)}{\sum_i k_i}. \tag{5.5}$$

Quadratic Distance

A well known cross-bin dissimilarity measure is the *quadratic distance*, first introduced by Hafner et al. [13]. It calculates a weighted similarity between histograms instead of simply comparing the values per bin. The computational cost for the quadratic histogram distance is high. Equation 5.6 expresses the quadratic distance between two histograms.

$$d_{(H,K)} = \sqrt{(h-k)^t * A(h-k)} \tag{5.6}$$

In Equation 5.6, h and k are vectors that contain all the entires in H and K; $A = [a_{ij}]$ refers to an $n * n$ matrix, and a_{ij} is the similarity coefficient between the bins i and j. The similarity coefficient can be calculated by the formula

$$a_{ij} = \frac{1 - d_{ij}}{d_{max}} \tag{5.7}$$

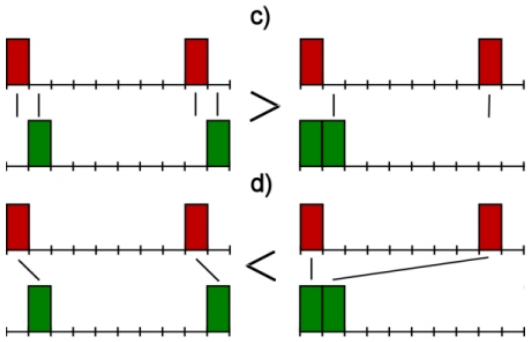


Fig. 5.3: An example where the quadratic distance does not match the perceptual dissimilarity. In Figure a), the quadratic distance between the histograms on the left side is greater than the quadratic distance between the histograms on the right side. Figure b) shows the desired distance, which is based on perceptual similarity.

where the ground distance $d_{ij} = |x_i - y_i|$ is a measure for the similarity between bins i and j.

The quadratic-form distance does not enforce a one-to-one correspondence between mass elements in the two histograms: The same mass in a given bin of the first histogram is simultaneously made to correspond to masses contained in different bins of the other histogram. In Figure 5.3.a and 5.3.b, it is shown how just like with the minowski distance, the perceptual dissimilarity can conflict with the quadratic distance between two histograms.

Match Distance

The *match distance* is another cross-bin dissimilarity measure. It is calculated with the formula

$$d_{(H,K)} = \sum_i |\hat{h}_i - \hat{k}_i. \tag{5.8}$$

The match distance between two histograms is defined as the 1-norm distance between their respective cumulative histograms. The cumulative histogram of a a histogram h_i is defined as

$$\hat{h}_i = \sum_{i \leq j} h_j. \tag{5.9}$$

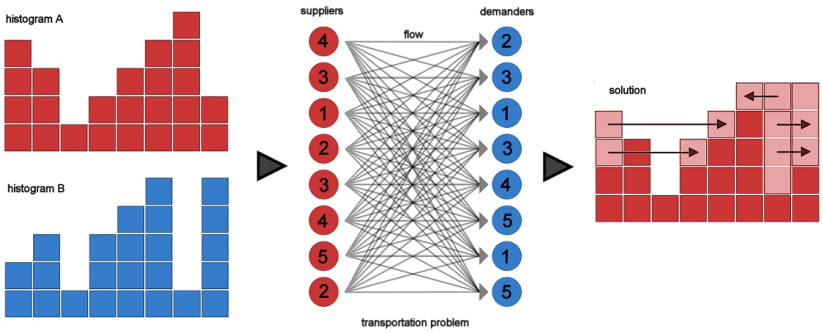


Fig. 5.4: The steps involved in calculating the Earth Mover's distance between two histograms.

Earth Mover's Distance

A histogram can be represented by a set of clusters where each cluster is represented by its mean (or mode) m_j, and by the fraction of the distribution that belongs to that cluster, w_{m_j}. Such a representation is also called a *signature* $s_j = (m_j, w_{m_j})$. The earth mover's distance can be interpreted in the following way: Given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. The earth movers distance measures the least amount of work needed to fill the holes with earth. In this metaphor, a unit of work corresponds to transporting a unit of earth by a unit of ground distance.

Computing the earth mover's distance is based on a solution to the well-known transportation problem [17]. Suppose that several suppliers, each with a given amount of goods, are required to supply several consumers, each with a given limited capacity. For each supplier-consumer pair, the cost of transporting a single unit of goods is given. The transportation problem is then to find a least-expensive flow of goods from the suppliers to the consumers that satisfies the consumers' demand.

Using signatures, a comparison between two histograms can be modeled as a transportation problem: One histogram is defined as the supplier and the other as the consumer. The cost for a supplier-consumer pair to equal the ground distance between an element in the first signature and an element in the second.

This can be formalized as a linear programming problem: By interpreting

histogram P as signature with m clusters $P = (p_1, w_{p_1}), ..., (p_m, w_{p_m})$ and histogram Q as $Q = (q_1, w_{q_1}), ..., (q_n, w_{q_n})$ with n clusters, and defining the ground distance matrix $D = [d_{ij}]$ where d_{ij} is the ground distance between clusters p_i and q_j. The goal of the linear optimizations is to find a flow $F = [f_{ij}]$, with f_{ij} being the flow between p_i and q_j, that minimizes the overall cost

$$Work(P, Q, F) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij} \tag{5.10}$$

The flow F is subject to the following constraints:

$$\begin{aligned}
&f_{ij} \geq 0 \\
&\sum_{i=1}^{m} f_{ij} \leq w_{q_i} \\
&\sum_{j=1}^{n} f_{ij} \leq w_{p_i} \\
&\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} = \min \left(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_i} \right)
\end{aligned}$$

The first constraint allows moving supplies from P to Q and not vice versa. The next two constraints limits the amount of supplies that can be sent by the clusters in P to their weights, and the clusters in Q to receive no more supplies than their weights; and the last constraint forces to move the maximum amount of supplies possible. This amount is also referred to as total flow. Once the transportation problem is solved, and the optimal flow is found, the earth mover's distance is defined as the work normalized by the total flow:

$$EMD(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} \tag{5.11}$$

The normalization factor is introduced in order to avoid favoring smaller signatures in the case of partial matching. The Earth mover's distance was first introduced as distance measure for monochromatic images by Peleg et al. [35] and is commonly used in image applications [40, 39].

Parameter-based Dissimilarity Measures

These methods first compute a small set of parameters from the histograms, either explicitly or implicitly, and then compare these parameters. Both bin-to-bin and cross-bin parameter based dissimilarity measures exist [37, 7].

5.3 Heuristic

In the section before it was explained how a genetic algorithm can be used to find an alpha transfer function that generates a visibility histogram for a given data set. However, using this approach has some disadvantages.
First, starting from a random population of possible solutions, it takes a rather long time for the algorithm to converge toward good solutions. Depending on the population size, in each generation a lot of visibility histograms have to be computed, and their distance to the desired visibility histogram has to be evaluated. Also, several very different transfer functions can generate the same visibility histogram. Some of these transfer functions might be more desirable than others, yet it is difficult to mathematically define which transfer functions are desirable and which ones are not. However, in order to include this criterion into the fitness evaluation that happens during the selection stage of a genetic algorithm, it would be necessary to define it in a form which can be computationally evaluated.
A different approach that takes into account is the use of a heuristic. A heuristic is a method to help to solve a problem, commonly informal. It is particularly used for a method that often rapidly leads to a solution that is usually reasonably close to the best possible answer. According to Michalewicz and Fogel [33], heuristics stand for strategies using readily accessible, though loosely applicable, information to control problem-solving in human beings and machines.
Concerning visibility histograms, an intuitive attempt at creating a simple heuristic that generates a transfer function from a given visibility distribution is to simply increase the alpha values for those regions where the user desires an increased visibility, while decreasing the alpha values everywhere else. However, estimating just by how much the alpha values of the scalar values within the region of interest

should be increased is already complicated, since these scalar values tend to occlude each other, and increasing the alpha value of one of these scalars can drastically change the visibility for the rest.
A possible way to refine the heuristic is to take the distance of a certain sample point to the viewing plane into account. The idea behind this is simple: since sample points that are close to the viewing plane potentially occlude many of the sample points that are within the range of interest, their alpha value has the most influence on the overall visibility. Overall, these sample points should have a rather low alpha value. For sample points which are far away from the viewing plane, the alpha value can be higher, since they are less likely to occlude important data points. The general idea of making regions closer to the viewing plane more transparent, while making regions further away more opaque can be refined further by counting, for every slice in the volume, just how many sample points for which the visibility should be increased because their scalar value falls within a region of interest are potentially occluded by sample points in the slice, and by adjusting the alpha values based on that number.

Chapter 6

Implementation

This chapter deals with the implementation details of the presented approach. In chapter 6.1 the implementation of the algorithm for calculating visibility histograms is described; Chapters 6.2 and 6.3 deal with the implementation of the genetic algorithm and the heuristic that were described in chapter 5.

6.1 Implementation of Visibility Histograms in General

In chapter 4, the theoretical steps for computing visibility histograms were described; this this section deals with the implementation. The goal of the application was to generate a method for displaying several visibility histograms for a given dataset in an OpenGL-enabled application.
In order to calculate a visibility histogram, rays need to be cast trough the data volume in the viewing direction, taking samples at a number of evenly spaced sample points. The values calculated for each sample point must be sorted into bins based on the density of the sample point, added together and then normalized by the total number of samples in the bin. The process of ray casting can be either be CPU based or GPU based. Typically, implementations that utilize the graphics hardware are faster than purely software-based implementations. As mentioned before, the end goal of is not to compute a single visibility histogram, but a number of visibility histograms that show the visibility distribution for the dataset when viewed from different directions. Therefore, it is important that a single visibility histogram can be computed fast, making a GPU-based implementation the better solution. The main challenge in GPU-based ray casting is that graphics hardware only supports polygonal rendering primitives, while a volume data set consists

of a three dimensional scalar field. It is necessary to decompose the volumetric object that should be rendered into primitives supported by the GPU. The basic idea behind GPU-based object-order volume rendering is that a discrete 3D scalar field can be represented as a stack of 2D slices. A 3D data set can be rendered by displaying a high number of semi-transparent 2D slices extracted from it. In this approach, the polygons that correspond to the slices are the geometric primitives used for rendering. These geometric primitives represent a proxy geometry which describes the shape of the data domain, usually the bounding box, not the shape of the object contained in the data. The data itself is supplied to the graphics hardware in the form of texture images. These texture images are then mapped to the slices. The interpolation of the opacity values at the sample points, as well as the compositing of the computed values to calculate the visibility at a particular sample point, is done by the GPU in a fragment program.

6.1.1 Geometry Setup

The first step is to decompose the volume data set into a series of polygonal slices which are parallel to the viewing plane. This is done by rendering a series of quads which are parallel to the viewing plane, one for each slice that should be rendered. Clipping planes are used to discard the fragments of each quad that do not belong to the data volume. The fragments which do actually belong to the data volume are not clipped, and processed by the vertex and fragment shader, where the actual visibility calculation takes place.

6.1.2 Texture Setup

The data volume and the transfer function are supplied to the GPU in the form of textures. The data volume is passed to the GPU in the form of a 3D texture, where the value that is stored at a certain point in the texture represents the scalar value at that point in the volume. When a fragment is processed in the fragment shader, looking up the value in the texture at the position of the fragment yields the scalar value at that position. The scalar value can then be used as a parameter for looking up the opacity value in the texture that stores the transfer function.
The transfer function maps an opacity value to each individual point in the dataset.

The simplest case is to assign an opacity value based on the density value on each position in the volume. In this case, an 1D texture can be used to store the transfer function. However, sometimes a transfer function which is dependent on more than one variable is desirable. Since the goal of the application is to compute visibility histograms for different viewing directions, it also makes sense to use different transfer functions for different viewing directions. Thus, a 2D texture is used to store the transfer functions, and the scalar value plus the viewing angle are used to look up the alpha value.
However, when computing a visibility histogram, each slice polygon that is processed by the fragment program contains the visibility values for all the data points that are present in the slice. It is therefore necessary to access each of the processed slice polygons individually, as the visibility values need to be sorted by density value and summed up. Furthermore, as the visibility value of a data point is computed by subtracting the accumulated opacity from before the light passes through the data point from the accumulated opacity after the data point, a way to pass the accumulated opacity of the previous slice to the fragment program is needed. Both of these problems can be solved by the use of two additional 2D textures. One texture is used to store the values that are generated by the fragment program when processing a single slice polygon. That texture then gets passed along with the data volume and the transfer functions when the next polygon is rendered. The other texture, which was used to pass the results along when the previous slice was processed, now becomes the texture into which the results of the fragment program are written. The two textures always alternate between storing the values generated when the previous slice polygon was processed, and passing the results to the fragment program.

Framebuffer Objects

Since the output of each slice that is processed by the fragment program is needed as input for the next slice in the form of a texture, the best approach is to have the fragment program render its values directly into a texture. This is accomplished by using the OpenGL framebuffer object extension. This extension allows for rendering to destinations other than the buffers provided to the GL

by the window-system. These newly defined rendering destinations are known as framebuffer-attachable images. The extension provides a mechanism for attaching framebuffer-attachable images to the GL frame buffer as one of the standard GL logical buffers: color, depth, and stencil. When a framebuffer-attachable image is attached to the framebuffer, it is used as destination of fragment operations. Rendering to a texture is accomplished by defining a texture as frame buffer-attachable image. The ”render to texture” semantics are similar to performing traditional rendering to the framebuffer, followed immediately by a call to CopyTexSubImage() to copy the contents of the framebuffer to the specified texture. By using the frambuffer object extension, an application can achieve the same effect, but with the advantage that the GL can usually eliminate the data copy from the framebuffer to the texture.

Pixelbuffer Objects

Each slice polygon contains the visibility values for all the sample points in the slice. These visibility values have to be sorted into bins based on the density at the sample point. This means that the contents of the frame buffer have to be read after every slice that is rendered. The standard command used for reading the contents of the frame buffer is glReadPixels(), which has the drawback of being a synchronous read. This means that after glReadPixels() is called, the driver will typically send a readback command to the hardware, and then wait for all of the data to be available before returning control to the application. In the case of the visibility histogram, this means that after each slice polygon is rendered, the application would have to wait until the contents of the frame buffer are read, before it can proceed to sort the visibility values calculated for the slice, and subsequently drawing the next slice. This rather inefficient approach can be improved by the use of pixel buffer objects offered by the OpenGL Pixel Buffer Object extension. Pixel buffers offer asynchronous reads, which means that the application does not have to wait until the read is finished, but can do something else in the meanwhile. Figure 6.1 shows how the frame buffer objects and the pixel buffer objects work together: after rendering slice n, its contents are written into frame buffer object FB_0. These values are then read into the pixel buffer

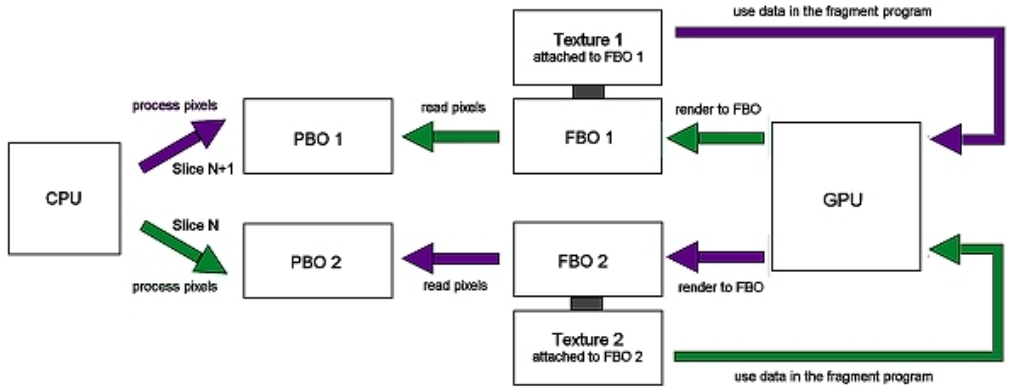


Fig. 6.1: An overview of how two pixel buffer objects and two frame buffer objects alternate between storing the output of the fragment shader for processing by the CPU, and passing data to the fragment shader as 2D texture.

object PB_0 in an asynchronous read. While waiting for the results of the readout, the values of slice $n-1$ which were rendered into FB_1 and read into PB_1 can be evaluated by the CPU, as they are ready at this point. Afterwards, the slice $n+1$ can be rendered into FB_1 while the texture bound to FB_0 is passed along, after which the next asynchronous readout for PB_1 is started. At this point, the values in PB_0 are ready for CPU evaluation.

6.1.3 Shaders

Each slice that is rendered is first processed by the vertex shader, and subsequently by the fragment shader. The vertex shader processes the four corner points associated with each slice, and assigns two sets of texture coordinates to each vertex: One set of texture coordinates is used to look up the accumulated opacity from the previous slice in the 2D texture, the other is used to look up the density value at the current position in the 3D texture that stores the volume. These texture coordinates are the interpolated for each fragment and used for the texture look ups. Shown below is the code of the vertex shader:

```
1 uniform mat4 TextureMatrix;
  void main()
3 {
      gl_Position = ftransform();
```

```
5     vec4 vecEyeVertex = gl_ModelViewMatrix * gl_Vertex;
      gl_ClipVertex = matTexture*vecEyeVertex;
7
      /* calculate the coordinates to look up the density */
9     gl_TexCoord[1] = TextureMatrix*vecEyeVertex;
      /* these texture coordinates are used to look up the accumulated opacity */
11    gl_TexCoord[0] = gl_MultiTexCoord0;
}
```

For each fragment that gets processed by the fragment shader, the accumulated opacity is looked up in the 2D texture. The alpha value for the fragment is calculated by first looking up the scalar value in the 3D texture that represents the data volume, then using the scalar value to look up the alpha value in the texture representing the transfer function. With the old accumulated opacity and the alpha value, the new accumulated opacity and the visibility are calculated. The RGBA-value that is written for each fragment contains the new accumulated opacity, which is used for the next slice, the visibility and the density which are needed to calculate the histogram, and 0 as alpha value. The alpha value is used to test whether a fragment actually belongs to the data volume, or to the background, as the values outside of the data volume are clipped and the alpha value of the background color is set to 1. The fragment shader is shown below:

```
uniform sampler3D DataVolume;
2 uniform sampler2D TransferFunction;
uniform sampler2D PreviousVisibility;
4 uniform float ViewingAngle;
void main (void)
6 {
      /* look up the accumulated opacity of the previous slice */
8     float PreviousOpacity = texture2D(PreviousVisibility, gl_TexCoord[0].xy).g;

10    /* look up the alpha value */
      float CurrentDensity = texture3D(DataVolume, gl_TexCoord[1].xyz).a;
12    vec2 vecLookup = vec2(CurrentDensity, ViewingAngle);
      float CurrentAlpha = texture2D(TransferFunction, vecLookup.xy).a;
14
      /* calculate the new accumulated opacity, and the visibility */
16    float CurrentOpacity = PreviousOpacity + (1.0-PreviousOpacity)*CurrentAlpha;
      float CurrentVisibility = (1.0-PreviousOpacity)*CurrentAlpha;
18
      /* the new accumulated opacity will be used in the next slice */
20    /* visibility and density at are needed to calculate the histogram */
      vec4 vecCurrent = vec4(CurrentVisibility, CurrentOpacity, CurrentDensity, 0.0f);
22    gl_FragColor = vecCurrent;
}
```

6.2 Implementation of the Genetic Algorithm and the Heuristic

For the scope of this thesis, two different methods for generating a transfer function when given a specific visibility histogram were implemented: the first implementation was in the form of a genetic algorithm, the second in the form of a heuristic.
Both the approach based on the genetic algorithm and the heuristic approach require a specific goal visibility histogram, which needs to be specified by user input. One possible method of specifying the goal visibility histogram is in the form of an editor similar to transfer function editors, where the user can specify a piecewise linear function with the help of control points that describes the desired visibility distribution. This approach has the advantage of allowing the user to specify any desired visibility distribution. However, as was mentioned before, when specifying an arbitrary visibility histogram, it is not guaranteed that a transfer function that produces similar visibilities can be found.
Instead of being able to specify a complete visibility distribution, the user is allowed to manipulate an already existing visibility distribution. This initial visibility histogram is generated by applying a very simple starting transfer function to the dataset. The user can the select a continuous interval out of the density values within the dataset, and then either increase or decrease the visibility of the points in the volume whose densities fall into the specified interval, by a certain amount. After specifying the desired interval, and by how much the visibility of the data points should be increased or decreased, the goal histogram is calculated, which is then used as input for the genetic algorithm or the heuristic.

6.2.1 Implementation of the Genetic Algorithm

The implementation of the genetic algorithm is fairly straightforward: First, an initial population of candidate transfer functions is created. In order to evaluate the fitness of these transfer functions, the visibility histogram has to be calculated for every single transfer function, and compared to the user specified visibility histogram. Afterward, pairs of individuals are chosen from the current generation

and recombined to form the next generation. The following an overview of the genetic algorithm in pseudo code:

```
/* initialization */
for (int i=0; i<PopulationSize; i++)
   seed the starting population p[i];

/* iteration */
for (int j=0; j<NumberOfGenerations; j++)\
{
    initialize current population n[i] to zero;
    for (int i=0; i<PopulationSize/2; i++)
    {
        /* selection */
        select two individuals p[a] and p[b];

        /* recombination */
        generate a random number r1 between 0 and 1;
        if (r1>CrossoverThreshold)
        {
            generate random crossover point c;
            n[i], n[i+1] = crossover(p[a], p[b], c);
        }
        else
        {
            n[i] = p[a]; n[i+1] = p[b];
        }
        generate a random number r2 between 0 and 1;
        if (r2<MutationThreshold) mutate(n[i]);
        generate a random number r3 between 0 and 1;
        if (r3<MutationThreshold) mutate(n[i+1]);
    }

    for (int i=0; i<PopulationSize; i++) p[i] = n[i];
}
```

Initialization

The algorithm starts by generating an initial population of n transfer functions. Transfer functions are represented by arrays containing floating point values between 0 and 1, and all transfer functions have the same length. A certain amount of the transfer functions are generated in a totally random fashion, by assigning a random number between 0 and zero as alpha value for each scalar value within the dataset. The other transfer functions are based on the initial transfer function, which was used to generate the initial visibility histogram. They are created by dividing the initial transfer function into a number of evenly spaced intervals, assigning a random scale factor to each of the intervals, and scaling the alpha values of the original transfer function that fall into a particular interval with the scale factor assigned to the interval. After the scaling, the alpha values are

clipped back into the interval $[0, 1]$. The scale factor is random for all the intervals that do not overlap with the user defined interval where the visibility should be increased or decreased, and weighted within the intervals that do overlap with the user defined interval.

Fitness Evaluation

In order to evaluate the fitness of the generated transfer functions, the earth mover's distance was implemented. Calculating the earth mover's distance between two histograms can be seen as transportation problem, where one histogram is assumed to be the supplier, and the other histogram is assumed to be the demander. In the implementation, the goal visibility histogram was used as the supplier, and the visibility histogram generated by the transfer function which should be evaluated was the demander. The bins from the goal histogram become supplier nodes, and the bins from the newly generated visibility histograms become the demander nodes. The ground distance d_{ij} between two histogram bins with indices i and j is simply defined as

$$|i - j|$$

This ground distance fulfills alls the conditions to be a metric, and as was mentioned in section 5.2, the earth mover's distance between two histograms that is based on a ground distance which is a metric, is a metric itself.
The algorithm takes the goal visibility histogram, $H1$, and the visibility histogram generated by the transfer function, $H2$, and begins by comparing the total supply offered by $H1$ to the total demand in $H2$. If the supply is greater than the demand, an additional demander that can "soak up" the extra supply is added to the transportation problem, and the ground distance to all supplier nodes is set to zero. If, on the other hand, the total demand by $H2$ is greater than the supply offered by $H1$, and additional supplier node is added to the problem that fulfills the extra demand. The ground distance between this supplier node and the demander nodes is again set to zero.
The cost matrix $C = [c_{ij}]$ is calculated. It stores the cost for moving one unit between supplier i and demander j, which is equal to the ground distance d_{ij}.

The matrix $F = [f_{ij}]$ stores the flow between supplier i and demander j, and is initialized to zero. Finally, the matrix $W = [w_{ij}]$ stores the reduced cost between each of the supplier nodes and demander nodes. The values in this matrix are calculated as

$$w_i j = c_i j - u_i - v_j.$$

The values u_i and v_j are the dual variables. They are associated with the so called basic cells, i.e. all the cells in the flow matrix where the flow $f_{ij} > 0$. The dual variables are chosen so that the reduced cost for the basic cells is zero. The reduced cost matrix, and the dual variables have to be calculated anew every time the flow changes, and are initialized to zero. Once the problem is set up like this, a starting solution is generated. The starting solution sets up an initial flow f_{ij} between each of the supplier nodes and each of the demander nodes. The cells (i, j) in the matrix F where $f_{ij} > 0$ become the basic cells, and the w_{ij}, u_i and v_j are calculated accordingly. The basic cells all have $w_{ij} = 0$, but the other cells take on positive and negative values. The condition for optimality for the transportation problem is that all cells must have nonnegative reduced costs. As long as that condition is not fulfilled, the given solution to the transportation problem can be improved. In order to improve the solution, the cell (i_0, j_0) in F with the minimal reduced cost $w_{i_0 j_0}$ in W is chosen to become a new basic cell. The new basic cell always determines a cycle that consists of the new cell and several basic cells of the current solution. This cycle is found by starting at the new basic cell (i_0, j_0) in F and moving from one cell to another in horizontal or vertical moves. Once this cycle has been identified, the flow is updated along the cycle: The minimum positive flow f_{min} from all the basic cells within the cycle is either subtracted or added to the cells in the cycle: starting at the new basic cell, f_{min} is added, changing the the total flow within this cell from 0 to f_{min}, then subtracted from the next cell in the circle, then added again in an alternating pattern. By doing this, the flow in one of the original basic cells is set to zero: this cell is no longer a basic variable. The new set of basic variables and their associated f_{ij} represent the new solutions. The w_{ij}, u_i and v_j are the updated to reflect the changes in the set of basic variables. The process of finding the cell (i_0, j_0) in F with the minimal reduced cost $w_{i_0 j_0}$ in W, searching for a cycle

and updating the f_{ij} along the cycle is repeated until the optimality condition is fulfilled, i.e. the reduced cost for all the cells is nonnegative. At this point the earth mover's distance can be calculated as

$$EMD = \frac{\sum_i \sum_j f_{ij} * d_{ij}}{\sum_i \sum_j f_{ij}}. \tag{6.1}$$

The following listing gives an overview of how the earth mover's distance is computed in pseudocode.

```
/* initialization */
initialize matrices F = f[i][j] = 0, C = c[i][j] = 0 and W = w[i][j] = 0;
initialize vectors U = u[i] = 0 and V = v[j] = 0;
find an initial solution;
calculate the u[i] and v[j] so that c[i][j]-u[i]-v[j]=0 for all (i,j) where f[i][j] != 0;
calculate the w[i][j] = c[i][j]-u[i]-v[j];

/* iteration */
while(1)
{
    /* optimality test */
    if (all w[i][j] >= 0) break;

    /* improve old solution */
    find a, b so that w[a][b] = min(all w[i][j]);
    find a cycle consisting of cells where f[i][j] != 0 and f[a][b];
    find the minimal flow fMin in the cycle;
    starting with f[a][b] alternate between adding or subtracting fMin;
}

/* calculate EMD */
EMD = F*C;
```

Selection and Recombination

Roulette wheel selection was chosen as selection scheme. After evaluating the distance between the visibility histogram generated by a transfer function of the current generation and the goal visibility histogram, the fitness is stored in an array. The fitness F_i of a solution is calculated as

$$F_i = \frac{1}{D_i}.$$

In the equation above, D_i stands for the earth mover's distance between the visibility histogram which is generated by applying the transfer function to the dataset, and the goal visibility histogram.

In each generation, a certain number of ”unfit” individuals are excluded from the selection and recombination step, in order to improve the rate of convergence. When the fitnesses of all transfer functions in the current generation have been evaluated, the total fitness of the remaining individuals is calculated, and their fitness values are normalized with the total fitness. In order to select an individual from the population, a random number between 0 and 1 is generated. Imagining that each individual in the population is assigned a pocket on a roulette wheel with the size of the pocket depending its normalized fitness, this random number is the equivalent of the roulette ball which lands in a certain pocket, with its value determining the individual that should be selected.
In order to create the n individuals that make up the next generation, $n/2$ pairs of individuals are selected from the fittest members of the current population. Each individual can be selected multiple times, however, each of the $n/2$ pairs of individuals must consist of two different individuals.
Since the transfer functions are represented by two arrays of floating point numbers of equal length m, the recombination step is rather straightforward: A crossover point is chosen by generating a random integer i_0 between 0 and $m-1$, which represents an index into the arrays. By swapping the values of the two arrays for all indices $i \geq i_0$, one-point crossover can be implemented. For the two-point crossover method, two indices i_{min} and i_{max} between 0 and $m-1$ are generated, and the values with indices $i_{min} \leq i \geq i_{max}$ are swapped. Mutating a transfer function is accomplished by iteration trough the array elements, and, with a very low chance, changing the value in a single array element to a random floating point number. Crossover and mutation can be carried out in a single step.

6.3 Implementation of the Heuristic

A heuristic approach was implemented as well. The heuristic approach is based on the following idea: in order to increase (or decrease) the visibility in a certain interval, the alpha values of the data points which fall into this interval are increased (or decreased), while the alpha values of all the other data points are decreased (or increased). However, increasing the alpha values for an interval does not always bring a high increase in the overall visibility of the interval, as a

certain amount of self-occlusion happens within the data volume. The approach also makes use of the fact that different transfer functions can lead to the same visibility distribution, and extends the transfer function from one dimension into two dimensions. The idea is to make the alpha values of the individual data points slice-dependent: data points which are in the front slices - i.e. the slices that are closest to the viewing plane - occlude many other points, while the data points that are located in the slices in the back of the volume occlude less points. Therefore the alpha values of points in the back can safely be set to higher values without having to worry about occlusion, while the points in the front slices have to be assigned alpha values that are relatively lower.

The algorithm works as follows: The data points for which the visibility should be increased are counted in each slice. The counting can be done with the same algorithm that computes a visibility histogram. With the values for each individual slice, it can be calculated for every slice how much of the total number of the data points that should be made more visible are potentially occluded by points within it. If the total number of points is c_0 and the number of points that lie behind the n-th slice is c_n, the fraction of potentially occluded points is $\frac{c_n}{c_0}$, a value between 0 and 1. The alpha value of all the points which belong to a slice then gets weighted with this value.

Chapter 7

Results

This chapter presents the result images that were obtained by the algorithms implemented in chapter 6. In section 7.1, result images of the general implementation of visibility histograms are presented. Section 7.2 presents some transfer function generated with the heuristic approach and the genetic approach, and compares the results of both approaches.

7.1 Visibility Histograms

In this section, the results which were obtained by the implementation presented in the previous section are be shown. The goal of the implementation was an application with which several visibility histograms for a given dataset could be calculated and displayed an a neatly arranged way.

Figure 7.1 shows a rendering of a dataset representing a human head, with several visibility histograms which show the visibilities from different viewing angles arranged in a circle around the dataset. The individual visibility histograms that are displayed around the dataset are generated by rotating the dataset around the y-axis in steps of 5°. The step size was chosen because it gives a good overview of how the visibilities of the individual data values change with the viewing direction, while still making the individual histograms easily distinguishable. Each visibility histogram contains 32 bins, representing same-sized intervals of the scalar values contained within the dataset.

A bin in the histogram is represented by a single quad with a particular brightness and color. The brightness of the quad shows how many data points within the dataset fall into that particular bin: A quad with a high brightness indicates that a

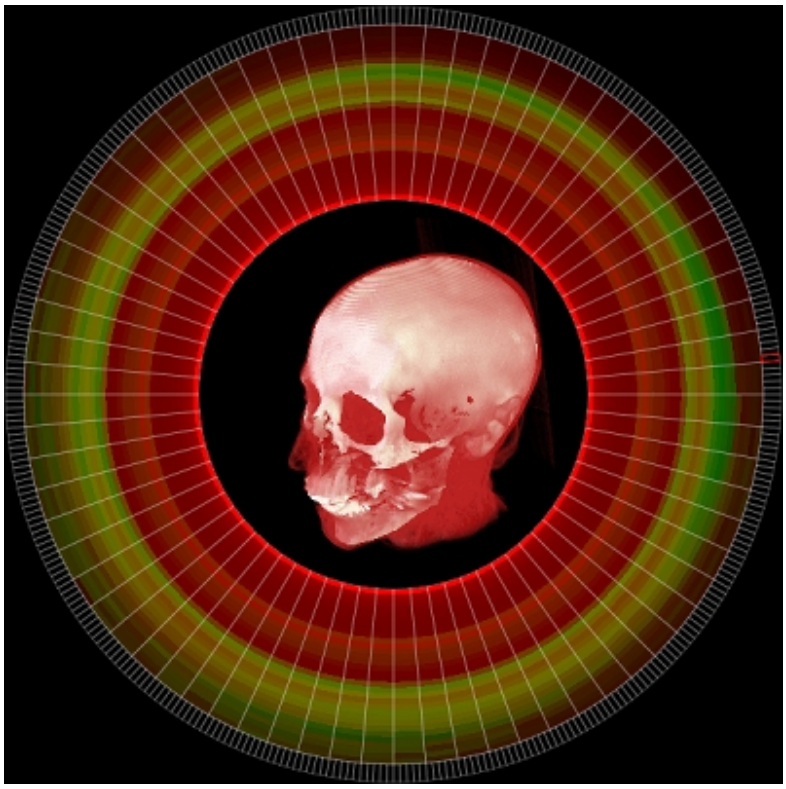


Fig. 7.1: A human head, rendering with the texture slicing method. The visibility histograms that are generated when rotating the dataset around the y-axis are displayed around the head. Green areas in the histograms represent a high visibility of the corresponding data points, while red ares represent a low visibility.

high number of points fall into that particular bin, while a low brightness indicates that only few data points belong to a particular bin. In Figure 7.1, there are two areas in the histograms where the brightness is very high: One represents the empty space around the head, which has an opacity of 0 and is therefore invisible, the other represents the data points which belong to the skin and flesh. These data points have a low opacity and are displayed in red in the rendering of the datasets. Even though this area is visible in the rendering of the dataset, the average visibility of a single point belonging to the area is rather low, as there are so many of them. The color of a quad indicates the average visibility of the data points that belong to the bin it represents. Color values range from pure green, which indicates a high visibility, to pure red, which indicates that the data points belonging to a particular bin are completely invisible. In Figure

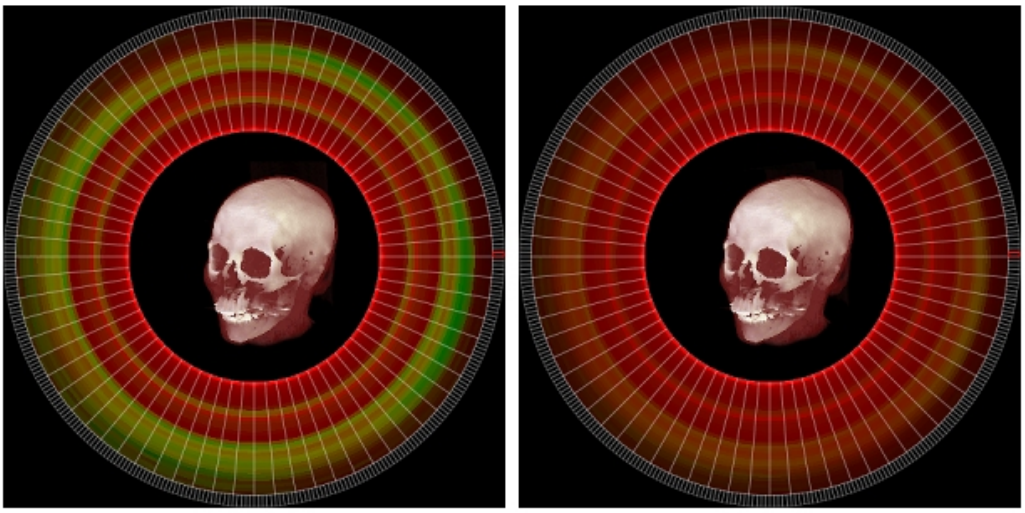
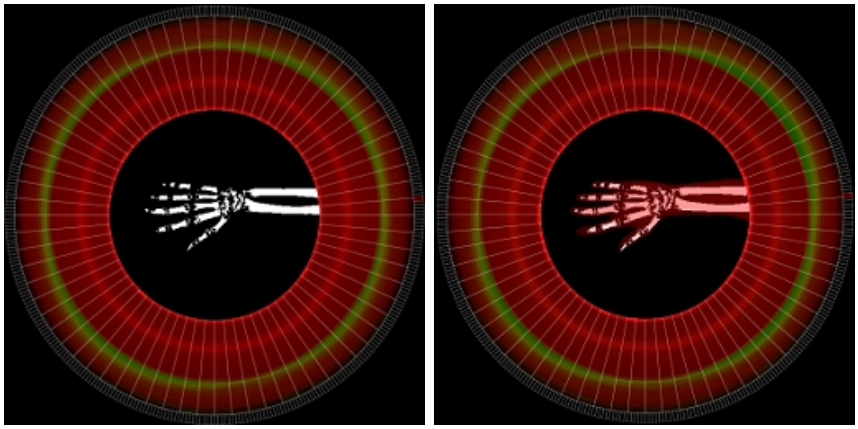
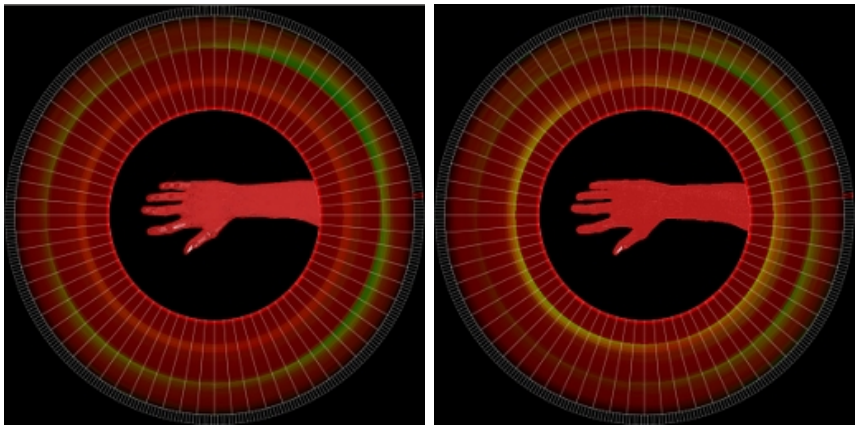


Fig. 7.2: The human head dataset with its respective visibility histograms. The visibility histograms on the left side use logarithmic scaling, while the visibility histograms on the right side display the unscaled values.

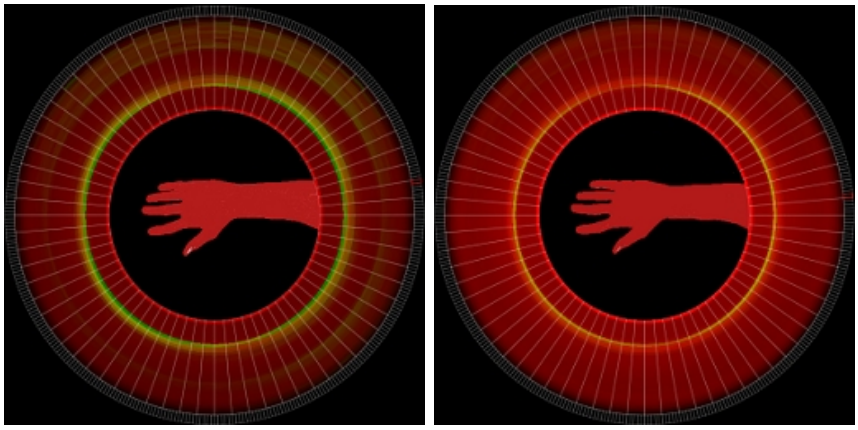
7.1, the bones have a high visibility which is represented by the green ring in the visibility histograms. It should be noted that the displayed visibility values are not absolute; Instead the displayed visibilities are normalized with the maximum visibility value that is contained within any of the displayed histograms, and scaled logarithmically in order to increase the contrast between areas of high visibility and areas of low visibility. Figure 7.2 displays the visibility histograms for the same dataset, once normalized and with logarithmic scaling and once in ”pure” form. In the pure form, differences in the visibilities are not very well recognizable.



(a) $\alpha_0 = 0$, $\alpha_1 = 0.8$ (b) $\alpha_0 = 0.0125$, $\alpha_1 = 0.8$



(c) $\alpha_0 = 0.05$, $\alpha_1 = 0.8$ (d) $\alpha_0 = 0.2$, $\alpha_1 = 0.8$



(e) $\alpha_0 = 0.4$, $\alpha_1 = 0.8$ (f) $\alpha_0 = 0.8$, $\alpha_1 = 0.8$

Fig. 7.3: The dataset is rendered with different transfer functions: the alpha α_1 value of the red tissue increases, while the alpha value α_0 of the white tissue stays constant.

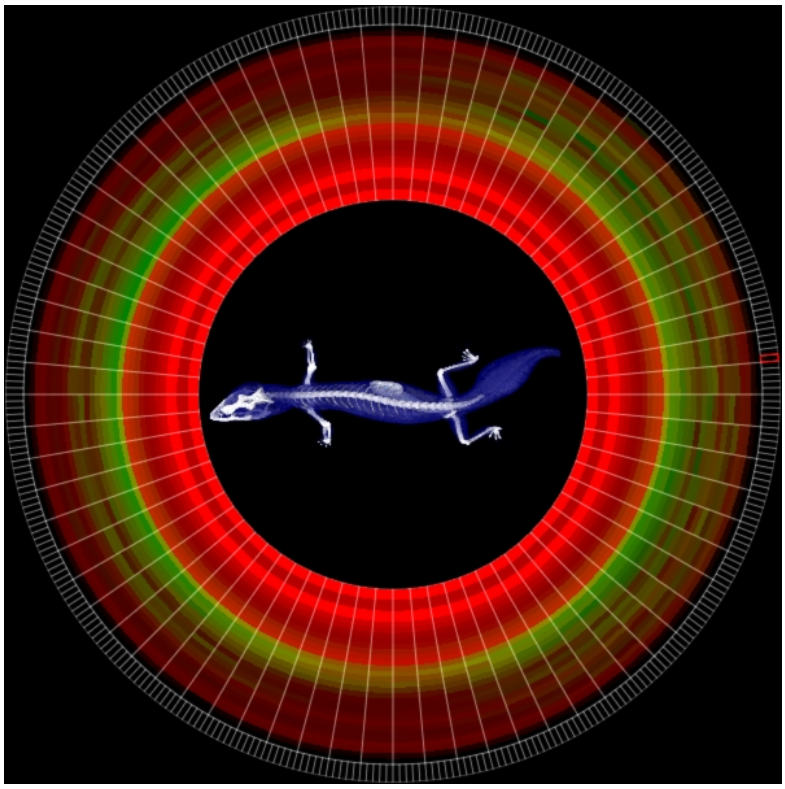
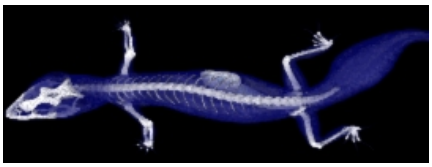
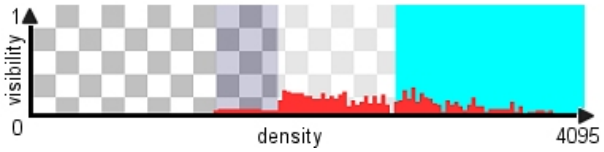


Fig. 7.4: A 3D dataset of a small lizard, with its corresponding visibility histograms.

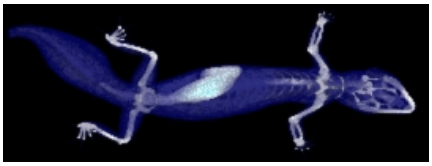
Figure 7.3 shows several renderings of a volumetric dataset of a human hand that was recorded using computerized tomography, each generated with a different alpha transfer function, with the alpha values of the red tissue which represents the flesh and skin increasing in each picture and the alpha values of the white tissue which represents the bones staying constant. In Figure 7.3.a, only the bones are displayed with an alpha value of 0.8. The visibility of the bones is represented by the bright green outer ring in the visibility histograms, while the rest of the scalar values are totally invisible. In Figure 7.3.b, the alpha value of the red tissue is set to 0.0125. Since the alpha value is low, and there is a lot of self-occlusion happening in the red tissue, the average visibility of a data points belonging to the red tissue is still close to zero. Because of this and the logarithmic scaling used to display the visibilities, the red tissue is not recognizable at all in the visibility histograms, while the visibility of the bones is still almost the same as in the previous picture. In Figure 7.3.c, the alpha value of the red tissue is increased



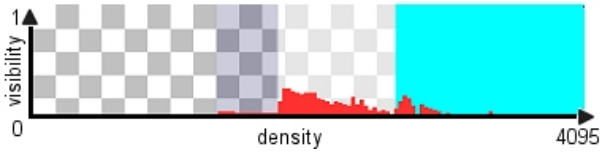
(a) the gecko dataset with rotation of 0° around the y-axis



(b) the corresponding visibility histogram

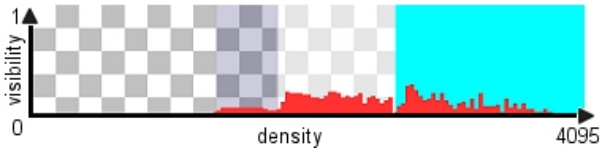


(c) the gecko dataset with rotation of 180° around the y-axis

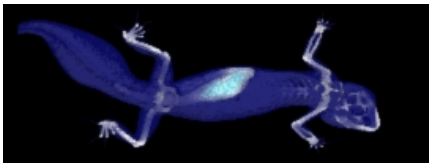


(d) the corresponding visibility histogram

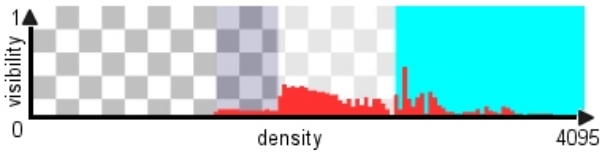
(e) the gecko dataset with rotation of 45° around the y-axis



(f) the corresponding visibility histogram

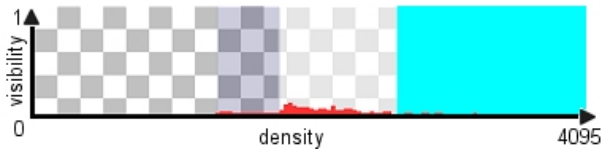


(g) the gecko dataset with rotation of 135° around the y-axis

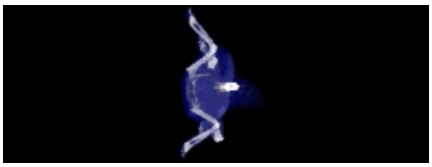


(h) the corresponding visibility histogram

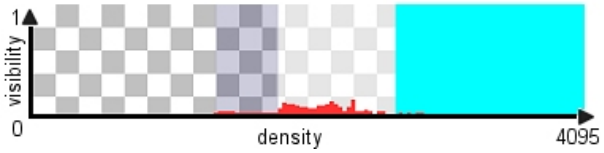
(i) the gecko dataset with rotation of 90° around the y-axis



(j) the corresponding visibility histogram



(k) the gecko dataset with rotation of 270° around the y-axis



(l) the corresponding visibility histogram

Fig. 7.5: the gecko dataset in detail, rotated around the y axis. the corresponding visibility histograms are displayed next to the datasets.

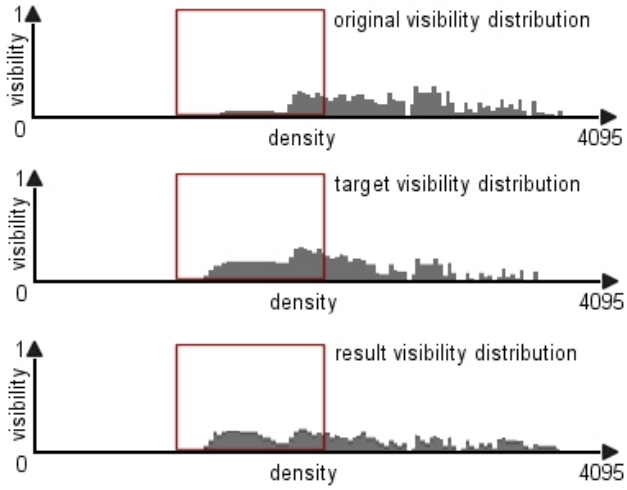
to 0.05. Here, the visibility of the red tissue begins to finally show in the form of the second yellow ring in the visibility histograms, while the visibility of the bones has decreased with respect to figures 7.3.a and 7.3.b. In Figure 7.3.d, the alpha value of the red tissue is set to 0.2. At this point, the data points belonging to the red tissue are clearly visible in the visibility histograms, while the bones are invisible from most viewing directions. In Figure 7.3.e, the alpha value of the red tissue is set to 0.4, which causes the red tissue to be displayed in bright green in the histograms, and the bones to become even more invisible. Finally, in Figure 7.3.f, the alpha value of the red tissue is set to 1.0: At this point, the bones become completely invisible. The self-occlusion of the red tissue is so high, that the overall visibility of the red tissue has decreased again.
By arranging several visibility histograms from different viewing directions at once, the user can get an overview of how the visibilities change with the viewing direction. In Figure 7.4, a dataset consisting of 256x256x88 data points which represents a small lizard is shown with its respective visibility histograms. Looking at the changes within the visibility distribution, it becomes clear that from certain angles, a lot of data points have a very low visibility due to occlusion. Especially when rotating the dataset 90° around the y-axis, the visibility of most of the points in the dataset becomes very low.
This is because when viewed from this angle, 256 slices have to be rendered, with many data points being occluded. For these two angles, the displayed visibility histograms are almost completely red, meaning that the average visibility for each scalar value is very low. The same situation happens when rotating the dataset 270° around the y-axis. Figure 7.5 shows the dataset viewed from various angles, plus the respective visibility histograms in detail. On the other hand, when not rotating the dataset at all, or by rotating 180° around the y-axis, only 88 slices are rendered, thus the least occlusion occurs.

7.2 Transfer Functions

In this section, the results which were obtained by the implementation presented in the previous section are be shown. The goal was to test how suitable the two methods presented in the previous section are at producing a transfer function

(a) dataset rendered with the original transfer function

(b) dataset rendered with the output transfer function

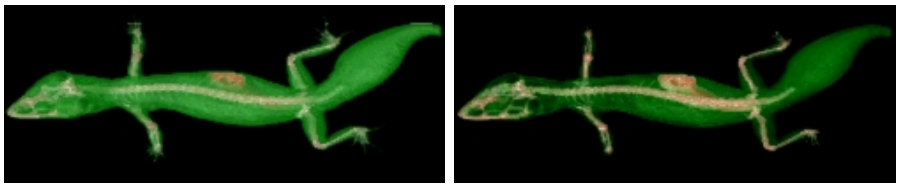


(c) the corresponding visibility histograms

Fig. 7.6: Results obtained with the genetic algorithm: (a) shows the original transfer function, while (b) shows the fittest member of the final population. Figure (c) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated, which corresponds mainly to the green tissue

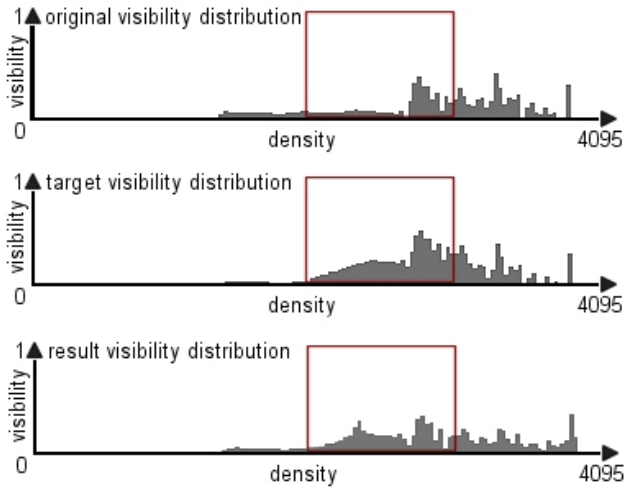
which matches the visibility histogram specified by the user.

When implementing the genetic algorithm, different values were tried out for the population size, the number of generations and the number of ”unfit” individuals which are not allowed to reproduce, as well as the rate of mutation and crossover. Since in every generation, the visibility histogram of every single individual within the population has to be calculated, the runtime of the genetic algorithm is very dependent on the population size, as well as the total number of generations. With a population size of 100 and 100 generations, 10000 visibility histograms



(a) dataset rendered with the original transfer function

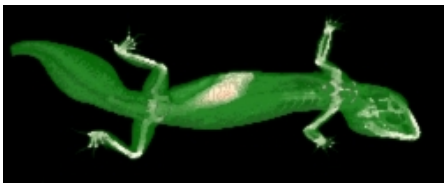
(b) dataset rendered with the output transfer function



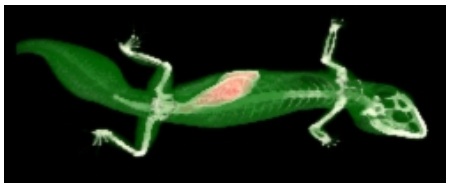
(c) the corresponding visibility histograms

Fig. 7.7: Results obtained with the genetic algorithm: (a) shows the original transfer function, while (b) shows the fittest member of the final population. Figure (c) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated, which corresponds mainly to the white tissue, i.e. the bone structure

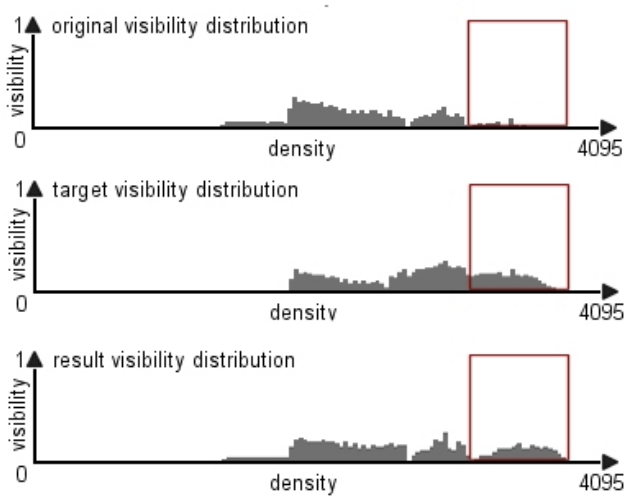
would have to be calculated and compared to the target histogram, which would take too much time. On the other hand, if the population size is too small, the genetic algorithm may not explore enough of the solution space to consistently find good solutions. The number of individuals excluded from possible selection and recombination, on the other hand, directly influences the selection pressure: If the selection pressure which is applied is too high, the algorithm might suffer form a well-known problem, premature convergence. If only the fittest members of each generation are allowed to reproduce, this can drive down the population's diversity



(a) dataset rendered with the original transfer function



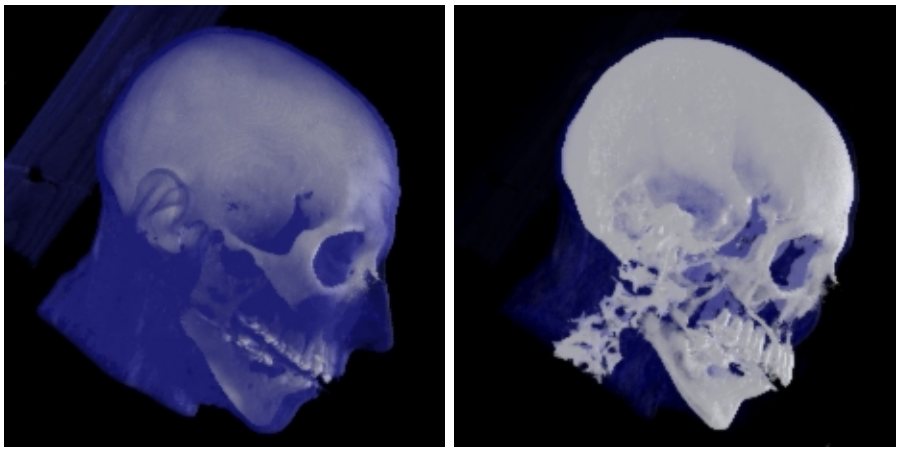
(b) dataset rendered with the output transfer function



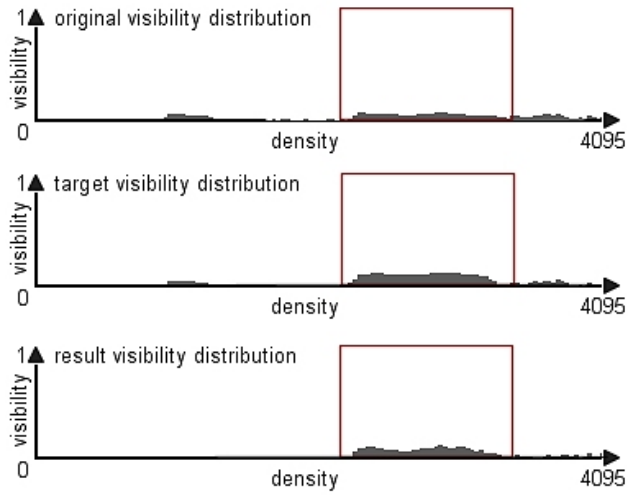
(c) the corresponding visibility histograms

Fig. 7.8: Results obtained with the genetic algorithm: (a) shows the original transfer function, while (b) shows the fittest member of the final population. Figure (c) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated.

too soon, leading the algorithm to converge on the local optimum than searching the fitness landscape thoroughly enough. In the end, a population size of 60 was chosen, and the 15 most unfit members were excluded from reproducing. The termination condition is simply that a maximum number of generations has been generated; Out of the individual within the last generation, the fittest individual is selected as the final result. The number of created generations was set to 20. In general, the most significant improvements within the total fitness of the individual generations were happening within the first 15 generations, with less overall improvement afterwards. The crossover- and mutation rates were set to



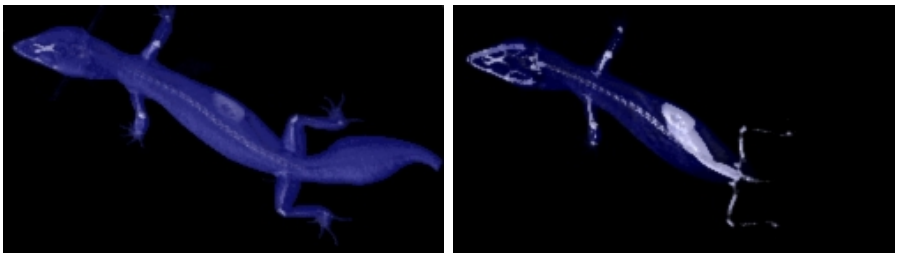
(a) Original transfer function (b) Output transfer function



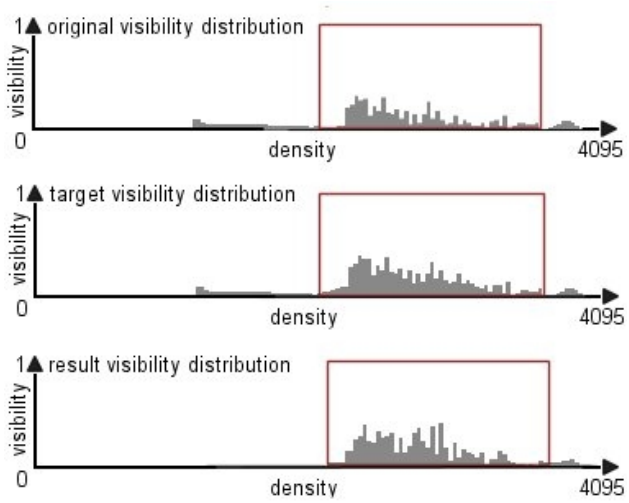
(c) Visibility histograms

Fig. 7.9: Results obtained with the heuristic approach: (a) shows the original transfer function, while (b) shows the generated transfer function. Figure (c) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated.

0.7 and 0.01, respectively. Figures 7.6, 7.7 and 7.8 show some results that were obtained with the implementation of the genetic algorithm. The figures show the fittest member - i.e. the transfer function when applied to the dataset - of the final generation, as well as the resulting visibility distribution. The visibility



(a) Original transfer function (b) Output transfer function



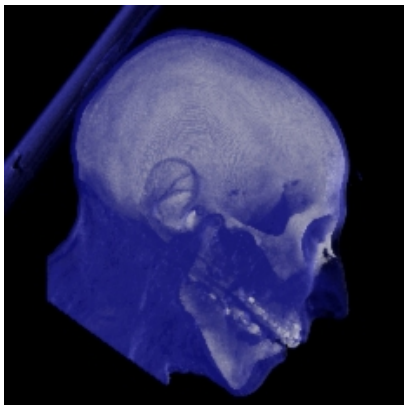
(c) Visibility histograms

Fig. 7.10: Results obtained with the heuristic approach: (a) shows the original transfer function, while (b) shows the generated transfer function. Figure (c) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated.

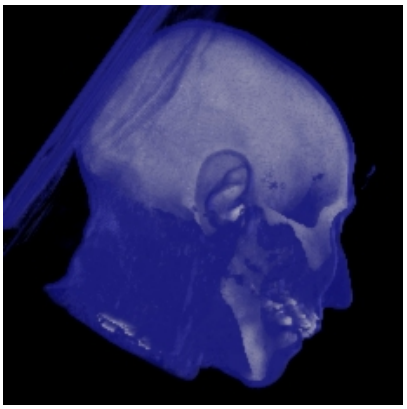
distribution of the result transfer function matches the overall shape of the desired visibility distribution rather well, however, the increase in the visibility is often not as high as desired.

In figures 7.9, and 7.10, transfer functions which were generated with the heuristic approach are shown. With the transfer functions generated with the heuristic approach, the resulting visibility distribution does not match the overall shape of the desired visibility distribution as well as with the transfer functions generated by the genetic algorithm. However, the achieved increase in the visibility comes

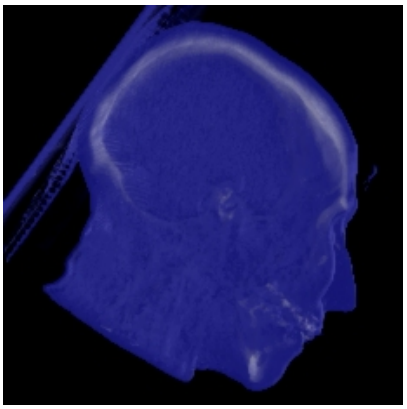
closer to the desired visibility increase than with the genetic algorithm.
In figures 7.11, and 7.12, the resulting transfer functions of both the genetic algorithm and the heuristic plus their respective visibility histograms are shown in comparison. In the images rendered with the resulting transfer functions, the data points for which the visibility was supposed increase, are more clearly visible than in the images rendered with the transfer function calculated by the genetic algorithm. Both algorithms do not always succeed at generating a transfer function which causes an a increase in visibility that is as high as desired, while decreasing the visibility of data points is, on the other hand, relatively easy. Generally, there is a connection between the amount of data points that should become more visible, and the the total increase that can be achieved: The higher the total number of data points, the more self occlusion occurs, thus the visibility increase that can be achieved decreases.



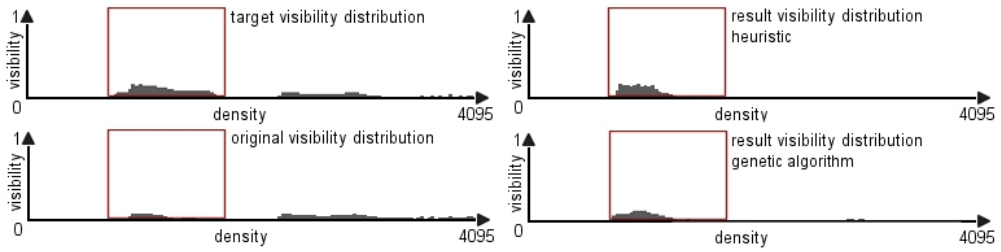
(a) Original transfer function



(b) Output of the genetic algorithm

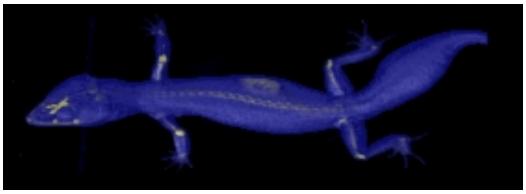


(c) Output of the heuristic

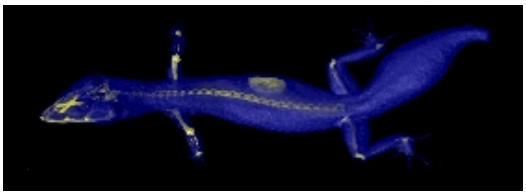


(d) Visibility histograms

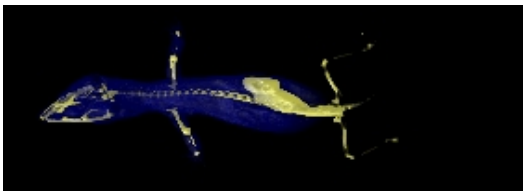
Fig. 7.11: Results obtained with the heuristic approach vs. the genetic algorithm: (a) shows the original transfer function, while (b) shows the the transfer function generated by the genetic algorithm. Figure (c) shows the transfer function generated by the heuristic. Figure(d) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated.



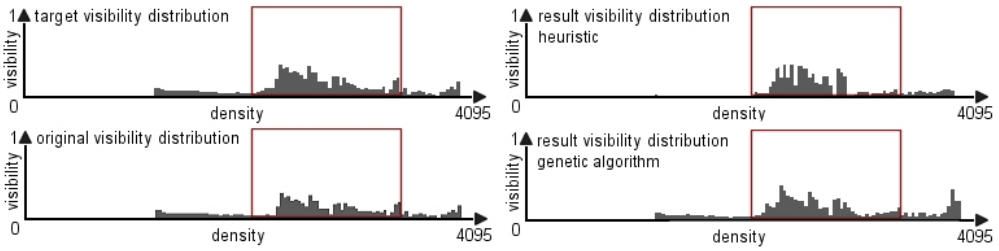
(a) Original transfer function



(b) Output of the genetic algorithm



(c) Output of the heuristic



(d) Visibility histograms

Fig. 7.12: Results obtained with the heuristic approach vs. the genetic algorithm: (a) shows the original transfer function, while (b) shows the the transfer function generated by the genetic algorithm. Figure (c) shows the transfer function generated by the heuristic. Figure(d) shows the visibility histograms of the original, the desired and the resulting transfer functions. The red box represents the area where the visibility was manipulated.

Chapter 8

Summary

8.1 Introduction

Volume rendering has become a very important area of research over the last decades. Its applications reach from medical imaging to scientific visualization of data. Volume datasets typically consist of a scalar values representing the density. In volume rendering, the definition of transfer functions is an important task for the generation of informative images. While manually setting up transfer functions via a trial-and-error approach is commonly used in many direct volume rendering applications, there are two main drawbacks about the approach:

- Setting up a proper transfer via trial-and-error can be a very time consuming task, since the parameter-space of all possible transfer functions is large.
- For higher dimensional transfer functions, which take additional properties like gradient magnitude, curvature or spatial information into account, the manual setup of the transfer function becomes unintuitive and takes even longer.

. Therefore, the development of approaches for automatic or semi-automatic transfer function generation has become an active are of research. Several approaches that assist the user at manually setting up transfer functions have been proposed [36, 15, 1]. Fujishiro et al. [11] propose techniques for automatically generating transfer functions by analyzing the topology of the data volume and extracting critical features. Kindlmann and Durkin [22] introduce a method that was designed specifically for volume datasets where the regions of interest are the boundaries between different materials, via the help of a volume histogram. Finally, Brodlie et al. [4] describe a method for the automatic setup of multi-

dimensional transfer functions by adding spatial information to the histogram of a volume.
Focus+Context in volume visualization has become another important topic of research. Wang et al. [48] and LaMar et al. [27] propose distortion techniques that have been adapted to work with direct volume rendering methods. In terms of illustrative rendering techniques, Levoy [28] was the first to propose modulation of opacity using the magnitude of the local gradient. This is an effective way to enhance surfaces in volume rendering by suppressing homogenous regions. Based on this proposal, Ebert and Rheingans [8] developed a collection of volume illustration techniques that adapt and extend non-photo realistic rendering techniques to volume objects. Lu et al. [30] present a framework for an interactive direct volume illustration system that simulates traditional stipple drawing, while Hauser et al. [14] propose *two-level volume rendering*, a technique which allows for selectively using different rendering techniques for different subsets of a three dimensional dataset. Lum and Ma [31] present a multi-dimensional transfer function method for enhancing surfaces which works through the modification of surface shading instead of the variation of opacity. The concept of *style transfer functions* was introduced by Bruckner and Gröller [5]. Instead of traditional transfer functions, style transfer functions represent specific styles captured from existing artwork.

8.2 Visibility Histograms

The basic idea behind visibility histograms is to show for a given volume dataset that is rendered with a specific transfer function, how much the density values contribute to the final image. For a dataset rendered with the help of a direct volume rendering algorithm, the visibility of a single sample n point is defined as

$$visibility(n) = \alpha_{s_n} - \alpha_{s_{n-1}}. \tag{8.1}$$

where α_{s_i} is the accumulated opacity at sample point i, i.e. the opacity that accumulates along a viewing ray that is cast into the volume in the viewing direction. The accumulated opacity can be calculated recursively by the following

formula:

$$\alpha_{dst} \leftarrow \alpha_{dst} + (1 - \alpha_{dst})\alpha_{src}, \tag{8.2}$$

which is the front-to back formula for alpha compositing. Therefore, the visibility of sample point n can be calculated recursively as

$$visibility(n) = (1 - \alpha_{DST}) * \alpha_n. \tag{8.3}$$

The visibility of sample point n is dependent on two things:

- The transfer function that is applied to the data volume, as the alpha value of sample point n, as well as the alpha values of all the sample points that are encountered by the viewing ray before n are needed for the visibility calculation.
- The viewing direction: By looking at the data volume from a different angle, the sample values that are encountered the viewing ray change, therefore the accumulated opacity before n changes as well.

Given these dependencies, visibility histograms can be used to analyze a volume dataset, either with respect to its sensitivity to changes in in the alpha values of a certain transfer function, or with respect to its sensitivity concerning the viewing direction. By calculating the visibility distribution for a dataset with a fixed viewing direction and an alpha transfer function that varies slightly for a range of scalar values, the effect of the sample that fall within the specified range on the overall visibility distribution can be visualized. Likewise, using a particular alpha transfer function while rotating the dataset around a virtual axis and calculating the corresponding visibility histograms generates an overview of how the occlusions within the dataset change with the viewing direction.

8.3 Automatic Transfer Function Generation with Visibility Histograms

Since the automatic setup of transfer functions has become an important topic of research, The main problem of this approach lies in the fact that, while it is relatively simple to calculate the visibility histogram for a given transfer function

and viewing direction, there exists exists no direct mapping from a given visibility histogram to a matching transfer function. Depending on the internal structure of the volume data set, there might be multiple transfer functions which produce a particular visibility histogram when applied to the dataset, or none at all. Therefore, instead of trying to let the user specify a certain visibility distribution and calculating an appropriate transfer function, it makes more sense to try and approximate the desired visibility distribution. The main idea is to let the user select a certain range of scalar values that should become either more visible or less visible, while keeping the visibility of the the other data values roughly the same.

8.3.1 Genetic Algorithm

The problem of generating a transfer function can be viewed as search problem: The space to be searched consists of all possible transfer functions that assign an opacity value in the range of 0-1 to each point in the data volume, and the transfer function searched for is the one where the difference between the generated visibility histogram and the desired visibility histogram is minimal. Genetic algorithms are commonly used to approximate the solution to such a problem [15, 32]. Basically, in genetic algorithms, an initial population of abstract representations of a number of candidate solutions evolves towards better solutions by repeatedly applying the techniques

- Selection
- Mutation
- Crossover

After each evolution step, a new generation is formed. This process is repeated until a certain termination criterion is met, e.g. a maximum number of generations is reached, or the individuals in the current generation are ”good enough”.
The two things that need to be defined for a genetic algorithm are

- a genetic representation of the solution domain and
- a fitness function to evaluate the solution domain.

Storing the corresponding alpha value to each scalar value in the dataset in an array works as a genetic representation of the solution domain. This representation makes genetic operations like crossover and mutation easy.
In order to evaluate the fitness of the transfer functions, the output that is generated by applying these transfer functions - i.e. the visibility histograms - need to be compared to the desired visibility histogram. A measure for the dissimilarity, also called *distance* between to histograms is needed. The different methods for calculating the distance between two histograms can be divided into two groups: *Bin-by-bin* dissimilarity measures compare the contents of corresponding histogram bins. *Cross-bin* dissimilarity measures also compare non-corresponding bins. For some cross-bin dissimilarity measures, a ground distance has to be defined between the bins in the histograms that don't correspond to each other, that allows for weighting the difference between two bins based on their ground distance.
The dissimilarity measure that is best suited for comparing visibility distributions is the earth movers distance. The earth mover's distance can be interpreted in the following way: Given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. The earth movers distance measures the least amount of work needed to fill the holes with earth. In this metaphor, a unit of work corresponds to transporting a unit of earth by a unit of ground distance. Computing the earth mover's distance is based on a solution to the well-known transportation problem [17].

8.3.2 Heuristic Approach

A heuristic solution is an alternative approach to the genetic algorithm. Its main advantage is that it is a lot faster than the genetic approach, as the genetic approach requires a lot of visibility histograms to be calculated. The goal of the heuristic approach was to find a quick solution, however depending on the volume dataset in question, the solution might not be particularly good. The heuristic approach that was chosen uses a two dimensional transfer function, which is dependent on the scalar value of the sample point, as well as its distance to the viewing plane. The idea behind the heuristic approach is to increase the alpha values of those

scalar values where the user wants to increase the visibility by a certain factor, while decreasing the alpha values every where else a little bit. In addition to that, the alpha values of those sample points which are farther away from the image plane get a higher alpha value assigned to them, since the chance that they might be occluding important structures is pretty low.

8.4 Conclusion

Visibility histograms can be used to gain insights into the internal structure of a volume dataset, and to visualize those insights. They can also be used to help the user of direct volume rendering applications in setting up transfer functions, however finding a transfer function that generates an exactly specified visibility distribution is generally not possible, therefore other approaches like heuristics and genetic algorithms have to be used.

List of Figures

Bibliography

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In *Proceedings of IEEE Visualization*, pages 167–173. IEEE Computer Society, 1997.

[2] N. A. Baricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, 9:143–182, 1957.

[3] J. F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Computer Graphics*, 11(2):192–198, 1977.

[4] K. W. Brodlie, D. J. Duke, K. I. Joy, S. Röttger, M. Bauer, and M. Stamminger. Spatialized transfer functions. In *Proceedings of EuroVis*, pages 271–278. Eurographics Association, 2005.

[5] S. Bruckner and E. Gröller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum*, 26(3):715–724, 2007.

[6] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., 1999.

[7] M. Das, E. M. Riseman, and B. A. Draper. Searching for multi-colored objects in a diverse image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 756–761. IEEE Computer Society, 1997.

[8] D. S. Ebert and P. Rheingans. Volume illustration: Non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization*, pages 195–202. IEEE Computer Society, 2000.

[9] D. B. Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, 1998.

[10] A. Fraser. Simulation of genetic systems by automatic digital computers. *The Australian Journal of Experimental Biology and Medical Science.*, 10:484–491, 1957.

[11] I. Fujishiro, T. Azuma, and Y. Takeshima. Automating transfer function design for comprehensible volume rendering based on 3D field topology analysis. In *Proceedings of IEEE Visualization*, pages 467–470. IEEE Computer Society, 1999.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[13] J. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):729–736, 1995.

[14] H. Hauser, L. Mroz, G. I. Bischi, and E. Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, 2001.

[15] T. He, L. Hong, A. Kaufman, and H. Pfister. Generation of transfer functions with stochastic search techniques. In *Proceedings of IEEE Visualization*, pages 227–234. IEEE Computer Society, 1996.

[16] G. T. Herman and H. K. Liu. Three-dimensional display of human organs from computed tomograms. *Computer Graphics and Image Processing*, 9(1):1–21, 1979.

[17] F. L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematical Physics*, 20:224–230, 1941.

[18] J. Hladuvka, A. König, and E. Gröller. Curvature-based transfer functions for direct volume rendering. In *Proceedings of the Spring Conference on Computer Graphics*, pages 58–65, 2000.

[19] J. Holland. *Adaptation in Natural and Artificial Systems,*. The MIT Press, 1975.

[20] J. T. Kajiya. The rendering equation. *SIGGRAPH Computer Graphics*, 20(4):143–150, 1986.

[21] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, 1975.

[22] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 79–86. ACM, 1998.

[23] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization*, pages 513–520. IEEE Computer Society, 2003.

[24] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization*, pages 255–262. IEEE Computer Society, 2001.

[25] J. Koza, F. Bennett, D. Andre, and M. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufman Publishers, 1999.

[26] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH*, pages 451–458. ACM, 1994.

[27] E. LaMar, B. Hamann, and K. I. Joy. A magnification lens for interactive volume visualization. In *Proceedings of the 9th Pacific Conference on Computer Graphics and Applications*, pages 223–232. IEEE Computer Society, 2001.

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

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

[30] A. Lu, Christopher J. Morris, D. S. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. In *Proceedings of IEEE Visualization*, pages 211–218. IEEE Computer Society, 2002.

[31] E. B. Lum and K. Ma. Lighting transfer functions using gradient aligned sampling. In *Proceedings of IEEE Visualization*, pages 289–296. IEEE Computer Society, 2004.

[32] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proceedings of SIGGRAPH*, pages 389–400. ACM Press/Addison-Wesley Publishing Cooperation, 1997.

[33] Z. Michalewicz and D. B. Fogel. *How To Solve It: Modern Heuristics*. Springer Verlag, 2000.

[34] C. L. Novak and S. A. Shafner. Anatomy of a color histogram. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 599–605. IEEE Computer Society, 1992.

[35] S. Peleg, M. Werman, and H. Rom. A unified approach to the change of resolution: space and grey-level. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):739–742, 1989.

[36] S. Potts and T. Möller. Transfer functions on a logarithmic scale for volume rendering. In *Proceedings of Graphics Interface*, pages 57–63. Canadian Human-Computer Communications Society, 2004.

[37] J. Puzicha, T. Hofmann, and J. Buhmann. Non-parametric similarity measures for unsupervised texture segmentation and image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 276–272. IEEE Computer Society, 1997.

[38] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[39] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, page 59. IEEE Computer Society, 1998.

[40] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 2(40):99–121, 2000.

[41] C. Silva, J. Comba, S. Callahan, and F. Bernandon. A survey of GPU-based volume rendering of unstructured grids. *Brazilian Journal of Theoretic and Applied Computing*, 12(2):9–29, 2005.

[42] P. J. Sloan, W. Martin, A. Gooch, and B. Gooch. The lit sphere: a model for capturing npr shading from art. In *Proceedings of Graphics Interface*, pages 143–150. Canadian Information Processing Society, 2001.

[43] M. Stricker and M. Swain. The capacity and the sensitivity of color histogram indexing. Technical report, University of Chicago, 1994.

[44] M. J. Swain. *Color indexing*. PhD thesis, The University of Rochester, 1990.

[45] I. Takanashi, E. B. Lum, K. Ma, and S. Muraki. Ispace: Interactive volume data classification techniques using independent component analysis. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, page 366. IEEE Computer Society, 2002.

[46] F. Tzeng, E. B. Lum, and K. Ma. A novel interface for higher-dimensional classification of volume data. In *Proceedings of IEEE Visualization*, page 66. IEEE Computer Society, 2003.

[47] I. Viola, A. Kanitsar, and E. Gröller. Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2005.

[48] L. Wang, Y. Zhao, K. Müller, and A. Kaufman. The magic volume lens: An interactive focus+context technique for volume rendering. In *Proceedings of IEEE Visualization*, pages 367–374. IEEE Computer Society, 2005.

[49] L. Westover. Footprint evaluation for volume rendering. *SIGGRAPH Computer Graphics*, 24(4):367–376, 1990.

[50] J. Zhou, A. Döring, and K. D. Tönnies. Distance based enhancement for focal region based volume rendering. In *Proceedings of Bildverarbeitung für die Medizin*, pages 199–203, 2004.