Unterschrift des Betreuers

TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

D I P L O M A R B E I T

Advanced Image-based Transfer Function Design

ausgeführt am

Institut für Computergraphik und Algorithmen
der Technischen Universität Wien

unter Anleitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Dipl.-Ing. Stefan Bruckner

durch

Leopold Kühschelm
Wolkersdorferstr. 23
2122 Ulrichskirchen

10. Dezember 2005 _____

Datum Unterschrift

Leopold Kühschelm

Advanced Image-based Transfer Function Design

Master's Thesis

supervised by

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Dipl.-Ing. Stefan Bruckner

Institute of Computer Graphics and Algorithms
Vienna University of Technology

Abstract

In volume visualization the definition of transfer functions is a critical task for the generation of informative images. We have the fact nowadays that the complexity of the images increases. So the problems of designing proper transfer functions are growing too. In this work an image-based approach is presented, which tries to simplify this task by providing multiple preview images for different function settings, so that the user can imagine more easily how the final image will look like.

Kurzfassung

Eine wichtige Aufgabe in der Volumensvisualisierung ist die Definition von Transferfunktionen, die es ermöglichen, Bilder mit einem hohen Informationsgehalt zu erstellen. Auf Grund der zunehmenden Komplexität in den Bilder wachsen auch die Schwierigkeiten bei der Definition der Transferfunktion. In dieser Arbeit wird eine Lösung für dieses Problem präsentiert, welche auf der Erstellung von Gallerien von Bildern basiert, die als Vorschau auf das endgültige Resultat dienen und verschiedene Einstellungen der Transferfunktion(en) repräsentieren. So wird es dem Benutzer ermöglicht, sich eine Vorstellung zu machen, wie das Endresultat aussehen wird und von Änderungen in der Transferfunktion auf Änderungen im Bild zu schließen.

Contents

Chapter 1

Introduction

See things as you would have them be instead of as they are

Robert Collier

The first part of this introduction will give a quick overview of the visualization process in volume visualization and in the second part we will present an outlook to the further chapters.

1.1 The Visualization Process

In the volume visualization process we want to enhance and transform the information in such a way, that the user has a better and more detailed insight what is covered in the data. We can identify different steps in this process (Figure 1.1). The first step is the data acquisition, followed by the data enhancement, the visualization mapping and finally the rendering of the information.

1.1.1 Data Acquisition

The data we operate on in volume visualization is a collection of sampled data points, which describe the density values of an object at discrete sam-

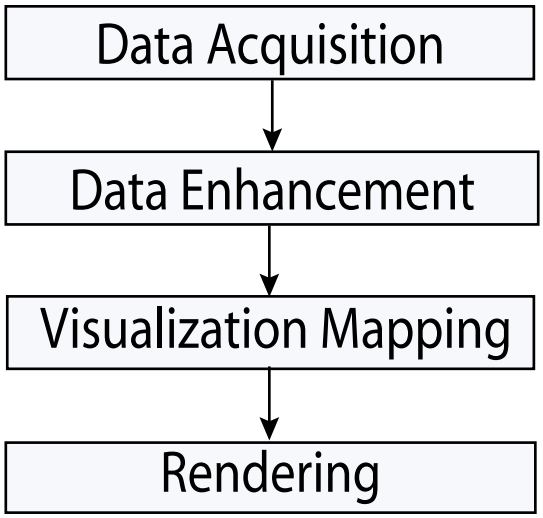


Figure 1.1: Visualization Pipeline

pling positions. This sampled data is usually created by tomographic imagining techniques like computed tomography (CT), magnetic resonance imaging (MRI), single-photon emission computed tomography (SPECT) or positron emission tomography (PET). Sometimes datasets are also created from geometric descriptions, this process is then called voxelization.

In CT imaging one important measure are the Hounsfield units, which describe the translucency of substance to x-ray irradiation . By definition the Hounsfield unit [39] of distilled water is zero - for more Hounsfield values of other substances see Table 1.1 .

1.1.2 Data Enhancement

Sometimes it is necessary to preprocess the acquired data. This preprocessing can include resampling or interpolation of additional data points, the calculation of gradients, or we have to correct motions of the object, which happend during the sampling process.

So the data enhancement step is extremely important for good results, because all the further calculations are based on the data, which is generated in this step. Therefore the quality of the data which is produced in this step

Substance	Hounsfield Unit
Air	-1000
Water	0
Bone	1000
Blood	40
Muscle	10 to 40
White Matter	46
Grey Matter	43
Cerebrospinal Fluid	15
Kidney	30
Liver	40 to 60
Fat	-50 to -100

Table 1.1: Hounsfield Units (values taken from [37])

is an upper bound of the quality of the final visualization.

1.1.3 Visualization Mapping and Rendering

Visualization mapping and rendering the information are densely coupled, so we will discuss this issues in one section. In volume visualization we have two main groups of rendering algorithms [8] and therefore also two different tasks in the visualization mapping.

The first group are the so called surface fitting or feature extracting algorithms. These algorithms produce surfaces by fitting lines of constant value with polygon patches together.

We have to specify in the visualzation mapping step the data value for which the surface should be extracted. After the generation of these patches we can use common rendering algorithms [2, 3, 4] for these patches to generate an image. Some algorithms of this group are cuberille [13], contour tracking [15], marching cubes [7] or marching tetrahedra [33]. The main drawback of these algorithms is that because of the decision of just one threshold value for the surface a large amount of information is lost.

The second group of algorithms are the direct volume visualization rendering techniques. In the direct volume rendering process the volume is seen

as a cloud of particles [22]. If a ray of light goes through a cloud there must be several effects measured like absorption, emission, scattering and shading. For simplification the resulting color is normally only computed by a model, which takes absorption and emission into account. So an approximation of the volume rendering integral is given by [23]:

$$I_\lambda(x,r) = \int_0^L C_\lambda(s)\mu(s)e^{-\int_0^s \mu(t)dt}ds \tag{1.1}$$

Here I_λ is the intensity of the light with wavelength λ received at the position x of the imageplane from the direction r. L is the length of the ray in direction r and $C_\lambda(s)$ is light of wavelength λ at the position s, which is emitted or reflected in the direction r. $\mu(s)$ is the density of the optical particles at position s and the term $e^{-\int_0^s \mu(t)dt}$ calculates the transparency of the volume along the ray until the position s.

Because the Equation 1.1 cannot be evaluated analytically [22] we have to approximate the integral with a numerical solution. So we discretize the integral into a series of intervals with a width of $\triangle s$ and compute the exponential term using a Taylor series. We get in that way an equation which can be solved more easily and is known as the common compositing equation [20]:

$$I_\lambda(x,r) = \sum_{i=0}^{L/\triangle s} C_\lambda(s_i)\alpha(s_i) \cdot \prod_{j=0}^{i-1}(1-\alpha(s_j)) \tag{1.2}$$

Here $\alpha(s_i)$ are the opacity samples along a ray and $C_\lambda(s_i)$ are the local color values derived from the illumination model. We can relate C and α to the local density value of the volume or other data properties like the density gradient there. So we can define a mapping between data properties and optical properties. This relation is commonly known as transfer function.

In general transfer functions can be seen as functions of a Cartesian product of data properties to a Cartesian product of optical properties [18]:

$$D_1 \times D_2 \times D_3 \times ... \times D_m \;\; \rightarrow \;\; O_1 \times O_2 \times O_3 \times ... \times O_n \tag{1.3}$$

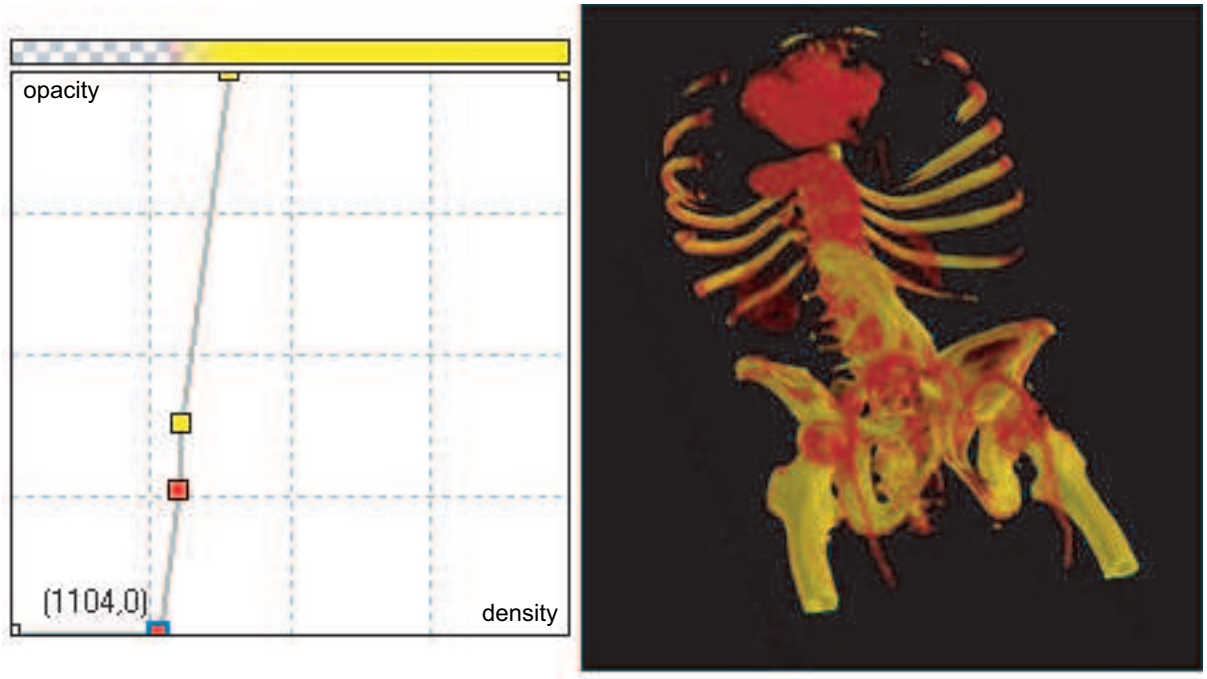
Normally transfer functions are implemented as lookup tables. This means that for a tupel of values of data properties, values of the optical properties are assigned. The simplest and most commonly used case is to define the density as data property (m=1) and the opacity and color values (red green and blue) as optical properties (n = 4). So we get a great variety of freedom to handle. A common implementation of an editor of such transfer functions will be a graphical tool where we can define piecewise linear functions through control points. We can assign opacity and color values at each control point. Here the intermediate values in the lookup table between the control points are calculated by linear interpolation. Even if we assume this restrictions we have a great variety of possible transfer functions. To demonstrate this we will look at some simple case i.e., one control point in the parameter space of the densities between density 0 and density 4095, 256 quantization levels for opacity and color values. So we get $4096 * 255^4$ possibilities to define a transfer function of this group of rather simple functions. Most of the time we want to define more complex transfer functions than a function with just one controlpoint, so we can see that there is a large parameter space, where we have to find out the best function for our purpose.

Another problem is that the relationship between transfer functions and the resulting image is often a non linear one. This means that small changes in the transfer functions often result in big changes in the final image, and sometimes if there are bigger changes made in the transfer function, there is nearly no effect in the image.

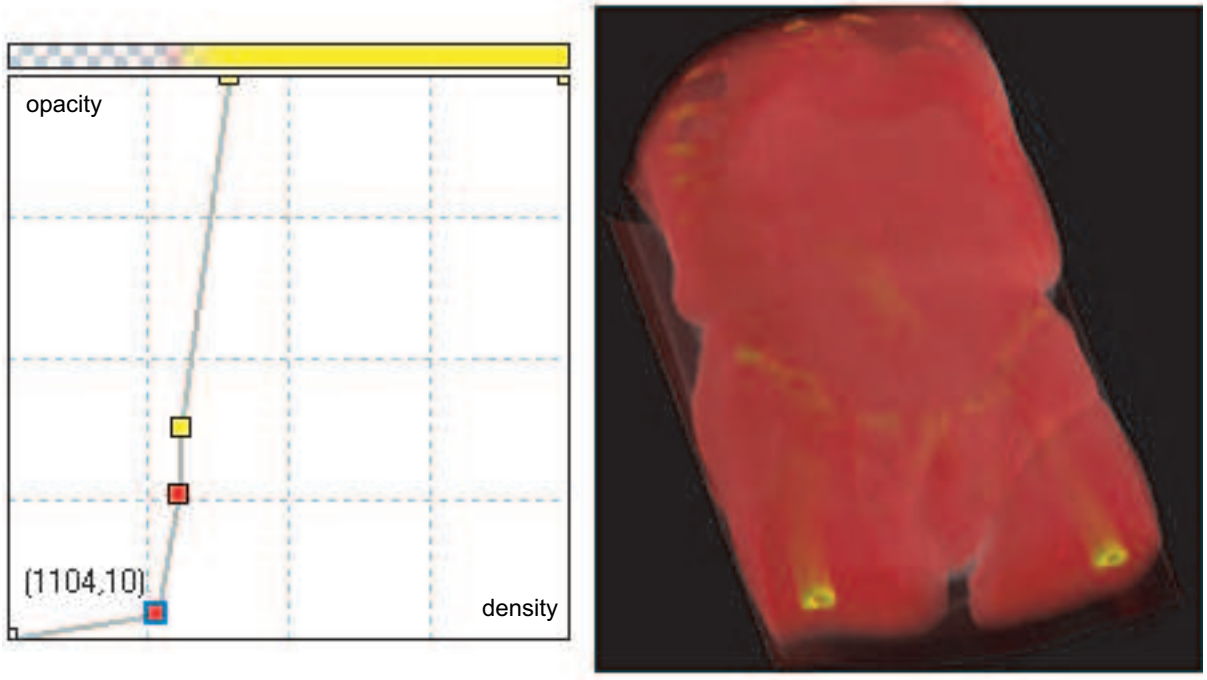
Before we will go on explaining some direct volume rendering techniques, we will show an example where a minor change in the transfer function makes a big effect in the image (Figure 1.2).

In the field of direct volume rendering we can divide the algorithms in three different groups: image-order, object-order and hybrid-order techniques.

In the image-order algorithms the image pixels are traversed and for each pixel the color is calculated (Figure 1.3). An image-order algorithm is ray-casting [20]. This method sends for every pixel of the image-plane a ray and depending on which voxels will be hit - the color of the pixel will be



(a)



(b)

Figure 1.2: Comparing transfer functions and resulting images: (a) First transfer function and resulting image (b) Second transfer function and resulting image

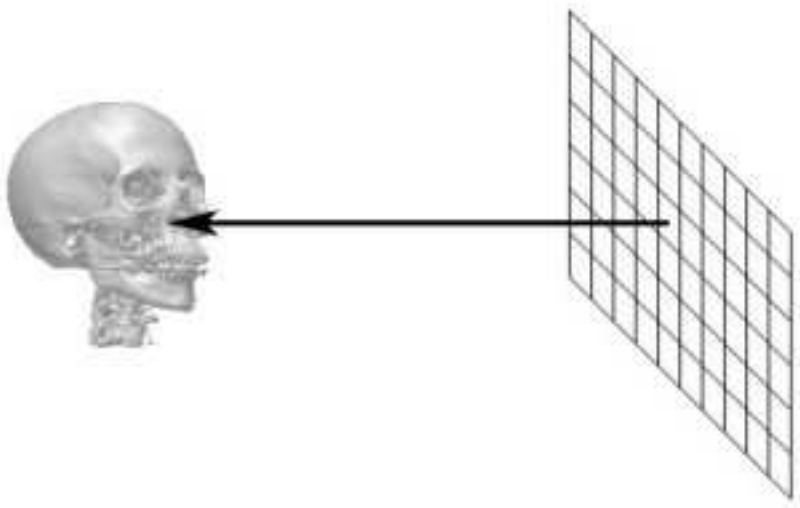


Figure 1.3: Image Order Traversal

calculated.

The object-order algorithms traverse through the volume and project each data sample to the image plane - see Figure 1.4. Members of this group are splatting [36] or texture mapping [27, 6] algorithms.

The hybrid-order algorithms try to combine the advantages of both groups (the object-order and image-order methods), one member of this group is the shear-warp algorithm [19]. It is based on a shear and warp factorization of the viewing transformation. The shear transformation has the property that after it has been applied all viewing rays are parallel to the viewing direction in the sheared object-space. So after the shear transformation the image- and object-space can be traversed simultaneously to produce an intermediate image. This intermediate image undergoes a two dimensional warp transformation to produce the final image.

1.2 Outlook to the next Chapters

In Chapter 2 we will present the state of the art method of transfer function definition. Corresponding to the image based approach, with the help of

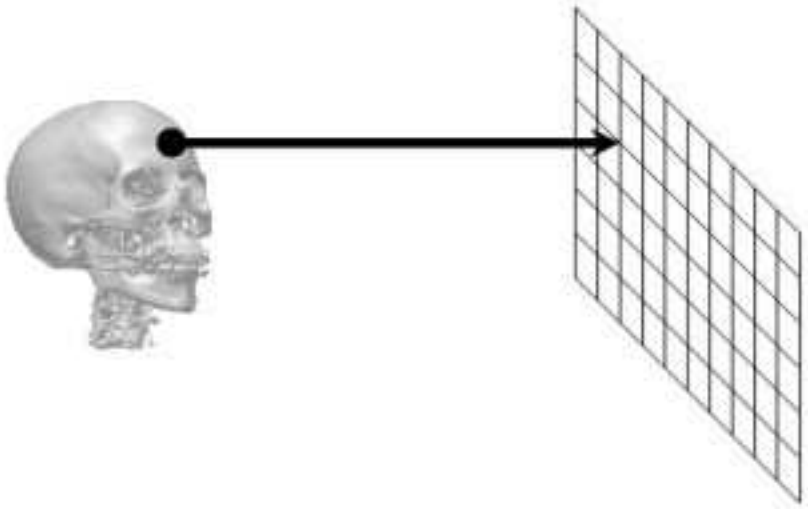


Figure 1.4: Object Order Traversal

design galleries, we will present different methods of generating such galleries efficiently in chapter 3. In Chapter 4 the implementational issues of this work are described. We will present the results in Chapter 5, which are not already presented in Chapter 3 to describe the methods there. Finally we will give a short summary of this work in Chapter 6.

Chapter 2

Transfer Function Design

The wisdom of life consists in the elimination of nonessentials.

Lin Yutang

We have already mentioned in the previous chapter the problems of defining an appropriate transfer function. So in this chapter we will introduce the four main groups of solutions to define a transfer function.

2.1 Four principal approaches specifying Transfer Functions

There are four main types of solutions to define transfer functions [26]. All these solutions have different advantages and disadvantages, which will be explained in the following parts of this chapter.

The first is the Trial-And-Error approach. Here the user tries to find an appropriate transfer function by manipulating the function by hand and rendering the image, to see what result comes out of the rendering process.

The second approach is the data-centric without data model technique. This method analyses the structure of a data-set in a first step and then

creates a transfer function based on this information.

The third approach (data-centric with data model) is mostly based on a mathematical model of boundaries between materials. Based on these detected boundaries the transfer function is created.

The last method is a quite simple approach (image centric). Here multiple images are rendered, with different transfer functions which are created through organized sampling through the data spectrum.

2.1.1 Transfer Function Specification through Trial-And-Error

As mentioned before the transfer function specification through Trial-And-Error approach is easy to handle but time consuming. The user has a lot of possibilities to create different transfer functions (color parameter, opacity, etc.). Therefore it is a hard task to find a transfer function which produces an appropriate image which meets the user's expectations. But the user gets a lot of knowledge about the data. So this method of defining a transfer function leads to an iterative process:

1. define the transfer function
2. render the image
3. assess the image - if it does not contain the information we wanted to visualize go back to 1 otherwise we are finished

The visualization pipeline gets one more step - the assessment of the image and an additional link backward to the visualization mapping step is added to model the iterative process (Figure 2.1).

Training can reduce the time to find a good transfer function using this method.

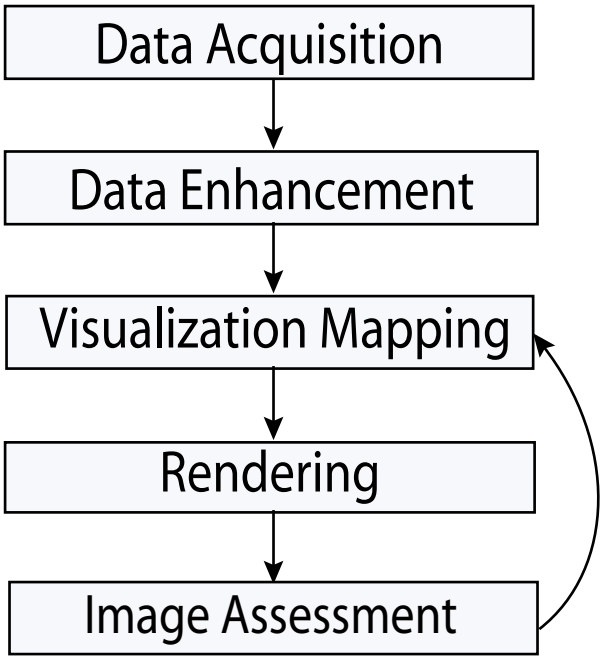


Figure 2.1: Visualization Pipeline for the Trial and Error Approach

2.1.2 Data centric Transfer Function Specification with no underlying data model

In a first step in most of these algorithms there are critical isosurfaces extracted. In a second step the topological structure of these surfaces is analysed and based on this data the transfer function will be generated.

One example of this type of method is transfer function design based on Hyper-Reeb-Graphs [9], which will be presented in the next part. An equal algorithm, which also works with topological analysis, is presented in [35]. In this paper the authors detect critical values by comparing the incident points (Figure 2.2).

The transfer function is created based on the critical values (these are the data values at the critical points) with the same method as described in the following section (Figure 2.5). Another method which corresponds to this group is described in [1].

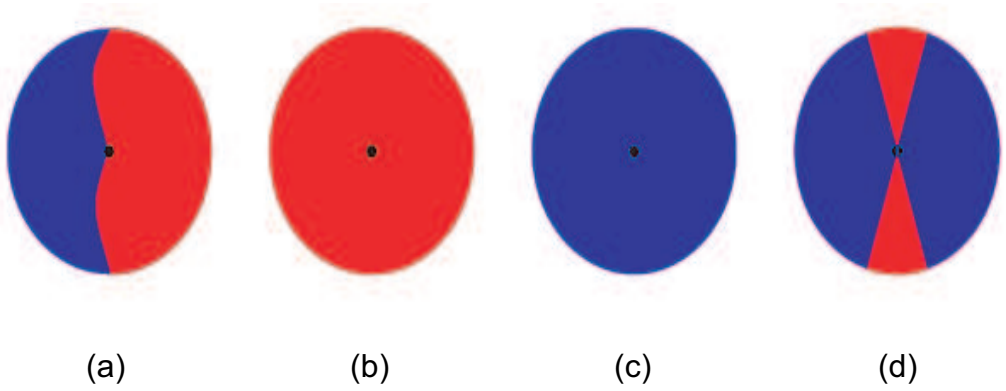


Figure 2.2: Critical Property
(a) Around a regular point the isosurface divides the volume into one single connected part with smaller and a single connected part with higher values, (b) Around a minimum all other points have a bigger value, (c) Around a maximum all other points have a smaller value, (c) At saddle points there are more than one single connected parts with smaller or higher values. The points in the cases (b), (c), (d) are called critical whereas the point in (a) is named regular as mentioned above. (Image taken from [35])

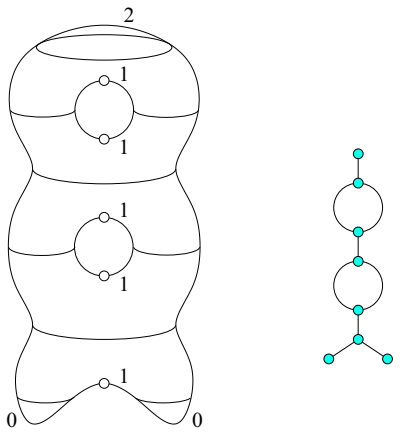


Figure 2.3: Double Torus with Cross-sections and resulting Reeb-Graph (image taken from [5]). The numbers stand for types of critical points (0 for minima, 1 for saddle points, 2 for maxima)

Transfer Function Design with the Help of Hyper-Reeb-Graphs

The transfer function design with the help of Hyper-Reeb-Graphs method interprets the volume dataset as a scalar field and specifies a metric on it, to analyse the structure of this scalar field and model this structure with the help of a hyper-reeb graph.

A reeb graph [5, 32, 10] is a structure, which depicts the topological structure of a manifold with the help of critical points. The definition of these critical points is based on the morse theory [41]. This theory is named after Marston Morse [24], an American mathematician. In his work on calculus of variations he introduced the technique of differential topology, which describes analysing a manifold based on differentiable functions. With this method substantial information on the manifold can be retrieved. This theory distinguishes between 3 types of critical points (pits, passes, peaks) also known as minima, saddle points and maxima - often indexed by the numbers 0, 1, and 2. One easy method to extract these critical points is to generate the cross-sections [32] of the object along a height function. For a better understanding see Figure 2.3. Here we see a torus with its cross-sections on the left and on the right you see the reeb representation of the torus.

Here we can see that on every critical point (pit-, saddle-, and peak-point) there is a topological change when taking the height function into account: at pit-points the figure starts, at saddle points the structure is split into or merged from two or more parts, at the peak-point the objects ends.

Going back to our dataset we can decompose it into multiple isosurfaces by its field values. For each surface we can obtain a reeb graph. By examining these graphs we can extract the information of critical isosurfaces in such a way, that we compare this surfaces on homo-topic equivalence. If we identify homo-topic changes we have found a critical isosurface and the referring field value is called critical field value.

A hyper-reeb-graph [9, 10] is a graph consisting of 2 layers. The top layer connects nodes with critical field values and the second layer depicts for each node or better for the node's critical value the reeb graph of the isosurface of the volume dataset. The figure 2.4 shows the 3Bloobies dataset and its hyper reeb representation [9].

In the design of the transfer function based on this graph Issei Fujishiro et al. [9] want to emphasis the critical surfaces, so they create a color transfer function which is uniform except at the critical value's position and an opacity transfer function which is constant except an elevation at the critical value's position (Figure 2.5).

2.1.3 Data centric Transfer Function Design with an underlying data model

An example of the next class of methods of defining transfer functions was published in [16]. It is based on an underlying data model - in this case the model tries to represent boundaries in the data-set.

The Boundary Model

The boundary model is used to detect changes of material based on the data values. As boundaries are always perceived as some rapid change of continuous areas, for instance in images, we can approximate a boundary by nothing other than a step function. But boundaries in real life are not always

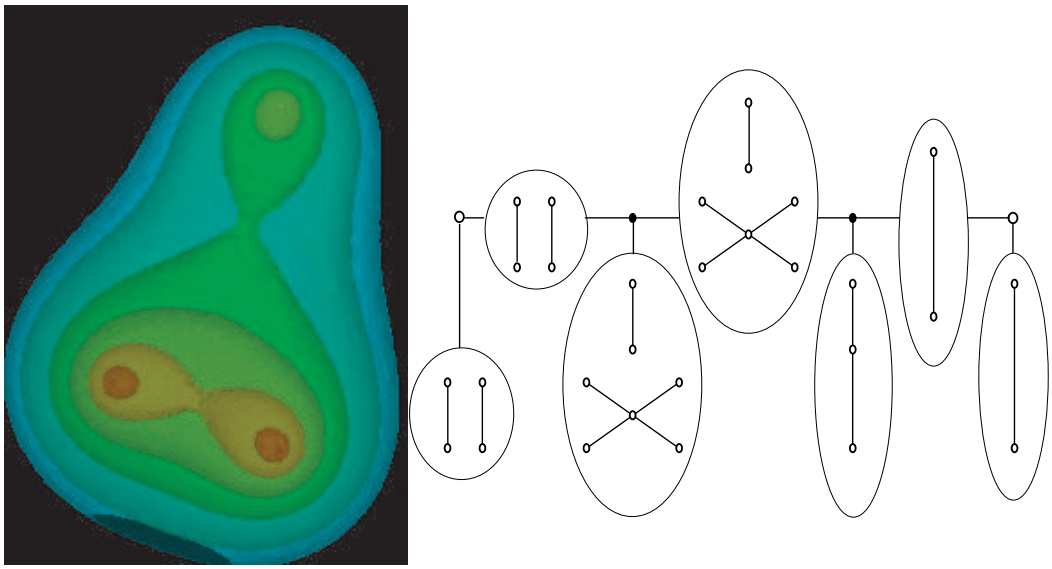


Figure 2.4: Critical isosurfaces of 3Bloobies dataset and resulting Hyper-Reeb-Graph (image taken from [9])

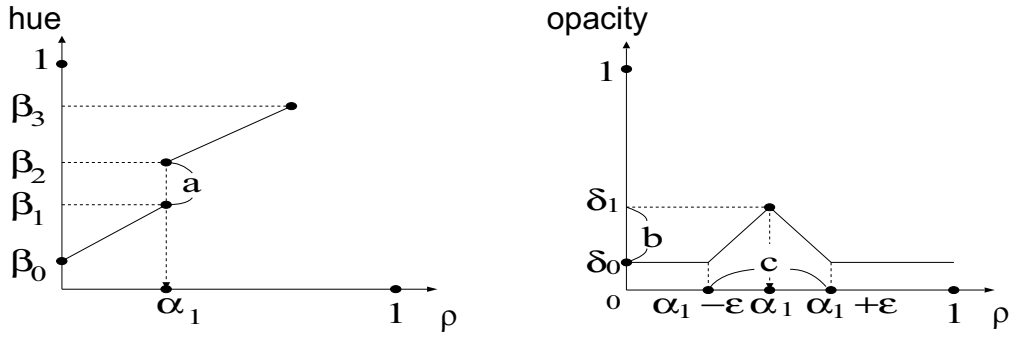


Figure 2.5: Transfer function design based on critical values (image taken from [9]) a) color function b) opacity function for critical field value α_1

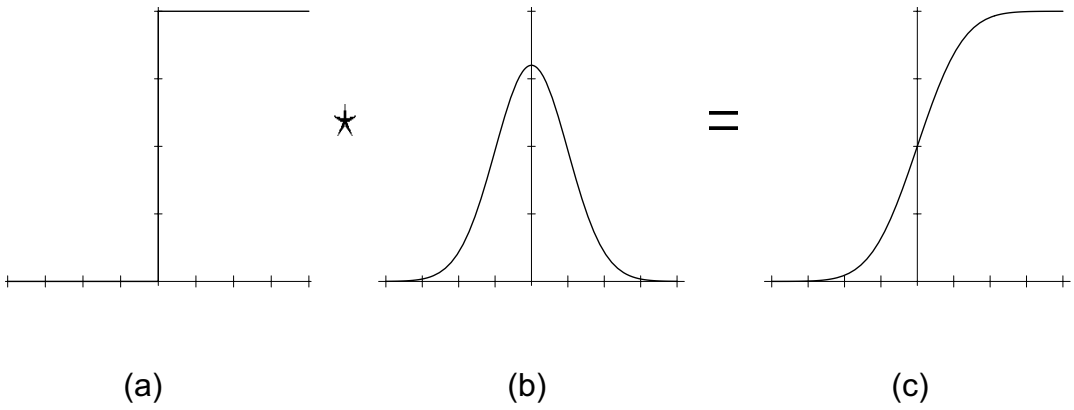


Figure 2.6: Boundaries blurred with Gaussian (image taken from [16]) (a) step function for representation of the boundary between different objects in the dataset (b) Gaussian function (c) Blurred Boundary Model

that sharp as modelled by a step-function therefore to get an appropriate function for boundaries we have to smooth it - in this case it was done with a Gaussian filter (Figure 2.6)

The Histogram Volume

An assumption made by Kindlmann et al. [16] is that they are visualizing continuous data. But they also use an inherent property of isosurfaces and gradients: gradients are always perpendicular to an isosurface. So the gradient direction is used as path through the volume to find the boundaries. The directional derivatives of first and second order along the gradient's direction are calculated to get a deeper understanding of the data.

Kindlmann et al. set the three data properties (data value, first and second derivative) in relation in a so called histogram volume. This is a three dimensional representation of the three properties, so that each property has its own coordinate axis:

1. The position of each bin in the histogram volume represents the three values at a small range

2. The value of the bin itself represents the number of voxels, which correspond to the three values of the bin's position.

As it is very important how to calculate the first and second directional derivatives and we also will need this values again in a later chapter we will shortly present how this values are measured. Mathematically the first directional derivative is calculated in the following way by Kindlmann et. al [16]:

$$D_v f = \nabla f \cdot v \tag{2.1}$$

According to the description above they go along the gradient therefore $v = \nabla f$ and so the first derivative is [16]:

$$D_{\nabla f} f = \nabla f \cdot \frac{\nabla f}{\parallel \nabla f \parallel} = \parallel \nabla f \parallel \tag{2.2}$$

This means that the first directional derivative along the gradient direction is the same as the magnitude of the gradient. Because of this property they interpret in [16] the second directional derivative as the gradient of the gradients' magnitudes. This measure is easy and computationally inexpensive to calculate, because in visualization the gradients can be used to compute the shading. The second derivative therefore can be defined in the following way [16]:

$$D^2_{\nabla f} f = D_{\nabla f}(\parallel \nabla f \parallel) = \nabla(\parallel \nabla f \parallel) \cdot \nabla f = \frac{1}{\|\nabla f\|} \nabla(\|\nabla f\|) \cdot \nabla f \tag{2.3}$$

We can calculate the histogram volume by the following algorithm (Algorithm 1).

Creating Opacity Transfer Functions

The next question is how to use the information from the histogram volume to generate transfer functions. Therefore we take a look at the boundary model, which can be defined mathematically in the following way in respect to the data value v of a boundary:

Algorithm 1 Histogram Volume Generation

1. Initialize the histogram volume to zero
2. Make one pass through the volume looking for the highest values of f' and f", and the lowest for f' and f". After that set the ranges of f' and f" accordingly
3. On a second pass: Measure f,f' and f" at each voxel, determine which bin corresponds to the values and increase the bin's value by one

$$v = f(x) = v_{min} + (v_{max} - v_{min})\frac{1 + erf(\frac{x}{\sigma\sqrt{2}})}{2} \tag{2.4}$$

Where erf() is the error function defined by $erf(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}dt$, v_{min} and v_{max} are the data values on both sides of the boundary and σ is the amount of Gaussian blurring of the boundary (Figure 2.6). So the first and second derivative are [16]:

$$f'(x) = \frac{v_{max} - v_{min}}{\sigma\sqrt{2\pi}}exp(-\frac{x^2}{2\sigma^2}) \tag{2.5}$$

$$f''(x) = -\frac{x\,(v_{max} - v_{min})}{\sigma^3\sqrt{2\pi}}exp(-\frac{x^2}{2\sigma^2}) \tag{2.6}$$

The next functions Kindlmann et al. need are $g(v)$, which is the average first directional derivative function of f, at the position where $f(x) = v$ and $h(v)$ which is likewise the same for the second directional derivatives. They also recalculate the boundary thickness through the maxima of $f'$and f''in [16]:

$$\frac{f'(0)}{f''(-\sigma)} = \sigma\sqrt{e} \tag{2.7}$$

Knowing the boundary thickness σ they calculate a function $p(v)$, which is used to determine on which side a voxel with value v lies - if it is closer to v_{min} it is negative, and otherwise if it is closer to v_{max} it is positive. We can calculate the function in the following way [16]:

$$p(v) = \frac{-\sigma^2 h(v)}{g(v)} \tag{2.8}$$

With the help of a function $b(x)$, which is only non-zero around zero (Figure 2.7) we can define the opacity function [16]:

$$\alpha(v) = b(p(v)) \tag{2.9}$$

Also see Figure 2.7 for the results of Kindlmann et al. using this method to define transfer functions.

2.1.4 Image centric Transfer Function Design

In this group of methods it is common to render multiple images and let the user choose some of them, to create a transfer function. It is an intuitive way for the users, but takes a lot of time for computational processing while the images the user can choose from are rendered. We will present in this chapter mainly one publication [18], which uses Volume Pro [25] technology to speed up the rendering process. It also shows clearly the main points of such algorithms for transfer function specification. Also Marks et al. [21] describe the use of this method by application of design galleries.

The definition process of a transfer function is divided into three steps by König et al. [18]:

1. Selection of the data range
2. Color selection
3. Select the opacity for the different data ranges

In the first step the user uses different geometric shapes to select which portion of density values should be selected. This geometric shapes are trapezoids, tents, boxes and ramps (Figure 2.8), which are defined round a peak point. The user can define multiple of these shapes with different positions and width by hand or he can choose a predefined selection. The system provides a preview image of each of these selections, so that the user can

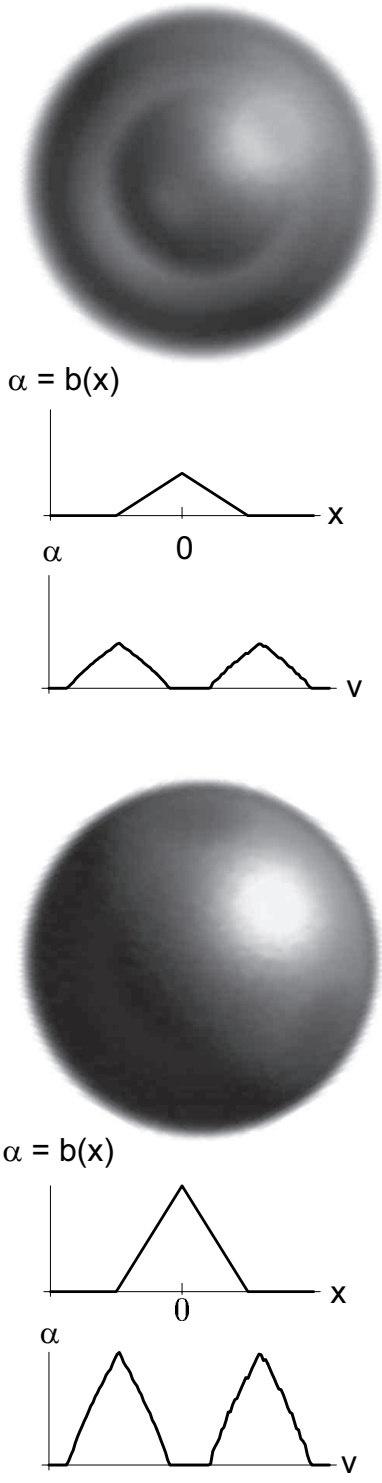


Figure 2.7: Examples for $b(x)$ and $\alpha(x)$ and their results (image taken from [16])

imagine how the final image would look like. The predefined selections are generated by the system first or can be loaded from a file by the user.

After the user has selected some of these shapes for further use, he has to define the color for these items. Again the user is supported by prerendered images and suggestive choices. As in step one he can select such a suggestion or modify a suggestion, or define a color for the shape by hand.

When the user is satisfied with the selections of the color settings for the shapes he has selected, he has to specify the opacity for them in the last step. Here again the user has the option to manually specify the opacity. He also can select and modify a predefined choice by the system.

The advancement in this approach is to separate the complex task of specifying a transfer function in three less complex subtasks. This technique, coupled with the idea of rendering preview images, makes it easier for the user to imagine how changes in the transfer function affect the final image.

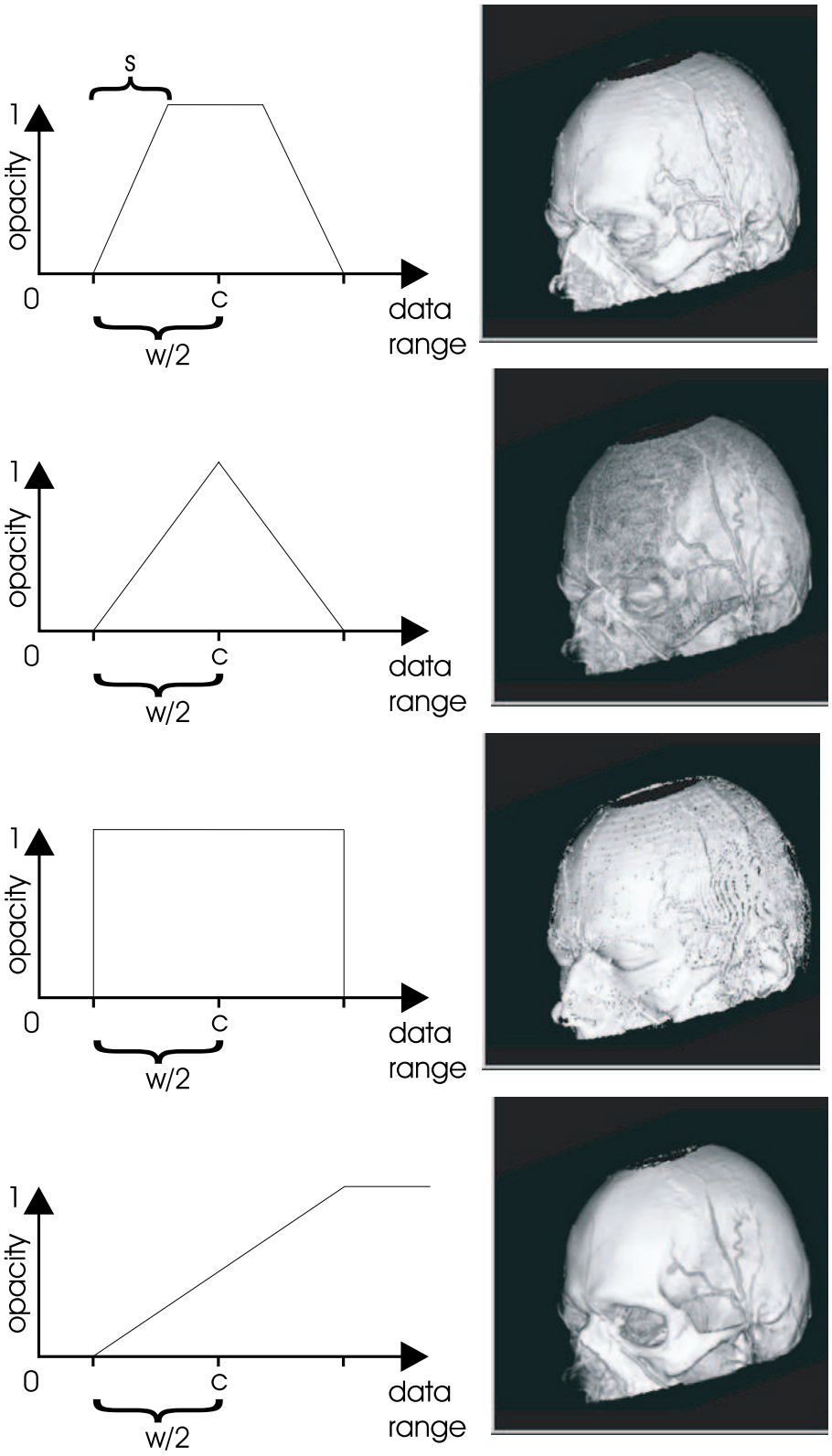


Figure 2.8: geometric shapes used to define data regions(image taken from [18])

Chapter 3

Design Gallery Generation

Read the folklore masters. Go to galleries. Walk in the woods. That's what you need to be an artist or storyteller.

Terri Windling

3.1 What are Design Galleries

We have already seen, it is rather difficult to define proper transfer functions, because of the large parameter space. We have discussed the four main approaches to cope with this problem. First we talked about an Trial-And-Error approach, which is rather simple but time consuming. The second solution were methods based on structural analysis of the data. The third approach we discussed was a solution based on a mathematical model of boundaries of objects and their representation in the data. The fourth approach was an image-based method. A series of images are generated using different automatically generated transfer functions. Then the user can select the image which matches best his expectations. This solution is easy to handle but is computationally rather expensive, because of the rendering of multiple images. The Design Gallery approach is a member of this class of solutions.

The interface usually provides an overview of the images and a function for their selection. Sometimes advanced functions are provided like selecting different viewpoints or generating additional resulting images from two selected ones [18]. The overview of the items of the gallery is provided through small thumbnail images and sometimes also in combination with a map where for all items some representatives are viewed [21].

In our raycasting system (see Chapter 4) we use this method for the definition of transfer functions of multiple objects rendered in one image. In this case the definition of transfer functions is even more difficult, than in the simpler case with one object, because of the interaction of two or more objects in one image, each having its own transfer function.

These difficulties lead the use of a design gallery. We generate transfer functions with the help of organized sampling. The resulting images of these functions are displayed in the gallery.

3.2 Generating Transfer Functions

This section describes the basic algorithm for transfer function generation. The core algorithm is presented in the next section and some additional refinements are then discussed in the further sections. The transfer function implementation we are talking about in the next sections is based on piecewise linear functions. This means there are some controlpoints which can be changed and the values (opacity, red, green, blue color) between the incident controlpoints are linearly interpolated. In this work's figures the opacity is depicted as the ordinate value of a controlpoint (Figure 3.1). The algorithms we are talking about in the following chapters will focus on manipulating the opacity to enhance the visibility of objects in the dataset. The color can be defined in the following step (see Section 4.2 the section about the Gallerymodule for more details).

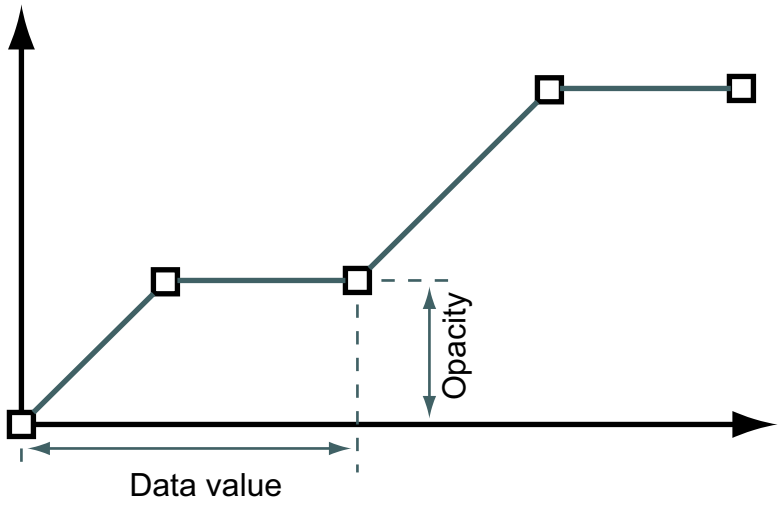


Figure 3.1: Schematic description of the piecewise linear transfer function

3.2.1 Transfer Function Generation through Organized Sampling

The basic idea of generating a sequence of transfer functions is to create some patterns (e.g. tent, box, ramp) at some specific points along the data axis for every transfer function. The first and easiest approach is to generate this sampling points at discrete intervals along the data axis.

Algorithm 2 Generate Transfer Function sequence with a constant offset between the data values, where the patterns (e.g. box, tent, ramp) are created

---

1. Initialize the sample-point to minimum value
2. As long as the sample-point is smaller or equal the maximum value do the following
3. Create a transfer function with zero opacity along the data axis
4. Set controlpoints for the selected pattern at the sample-position
5. Save the transfer function to a list
6. Go back to 2

This Algorithm (Algorithm 2) is the very basic step of generating the transfer function sequence. In some further refinement methods another algorithm was used which takes a list of specific sample points and creates the sequence of the transfer function not with constant offset between the samples, but with the positions from the list as creation points for the patterns. The idea of using the patterns (box, tent and ramp) was already used by A. König and E. Gröller [18]. In this paper an algorithm like Algorithm 2 is used for generating multiple transfer functions, which cover the whole data range. We took this idea and tried to improve it in different manners.

First we have to take a look on the essential aspect of these algorithms, and that are the patterns. We will reproduce the the properties of the patterns which were already discussed by König et al. [18]. We will introduce first the different geometric shapes.

The box pattern is rather simply defined: the height defines the opacity and the width the covered region of data values. Data values which lie at a distance of the half of the width of the box in both directions from the position are affected by this pattern (see Figure 3.2). The opacity is constant for the whole covered data region.

The tent pattern is defined around its peak in the middle. So the height is the opacity there. The covered data region is again defined by half of the width around the position, which is just in the middle of the tent (Figure 3.3). The opacity is obviously not constant for the region, but we have not such steep ascends and descends compared to the box.

The ramp pattern is well known in volume visualization as a windowing function. Beneath a certain limit all data values will be cut out (zero opacity) and therefore the values will not be represented in the image. The opacity of the data values which are in the scope of the slope are increased successively until the final opacity is reached, which is defined by the height of the ramp. The factor of height/width defines the ramp's slope. The reference point for the position of the ramp lies in the middle of the slope (Figure 3.4).

If we compare the achieved results from the box, tent and ramp pattern (Figure 3.5), we see that we get rather bad results from the box pattern, because it produces a lot of noise. This goes back to the fact, that if we use

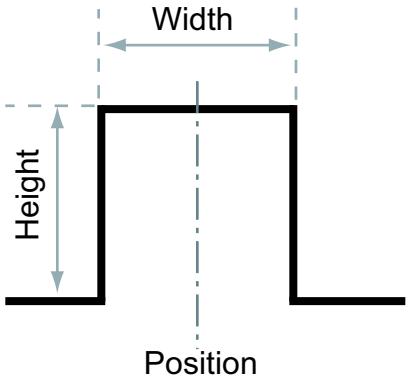


Figure 3.2: Description of the Box Pattern

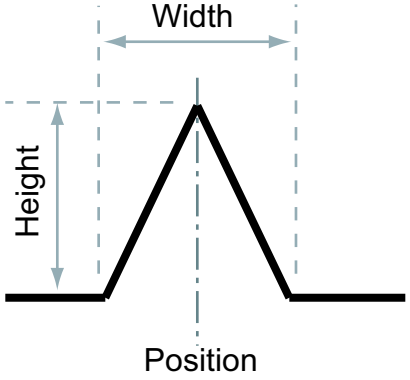


Figure 3.3: Description of the Tent Pattern

the box pattern we get a reconstruction kernel with a lot of high frequencies, which leads to a lot of noise. We can achieve nearly the same results with the tent pattern, only the quality is better, because we do not have the steep ascends and descends in this case. The ramp pattern shows the windowing behaviour we have discussed first. It shows the surfaces in a good quality, but we cannot see what lies beneath them. In this case the tent pattern is better, because it gives also some information of the interior.

As conclusion of this section we will now present a graphical representation of the algorithm. We use in this case the tent pattern (Figure 3.6). In this figure we see in the first row a line with the sampling points marked with

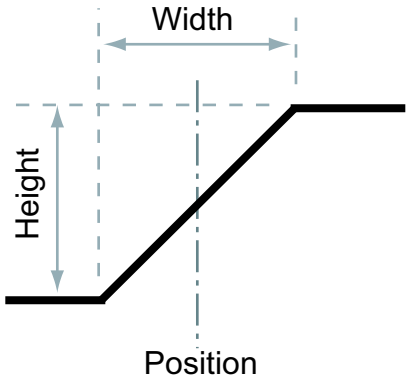


Figure 3.4: Description of the Ramp Pattern

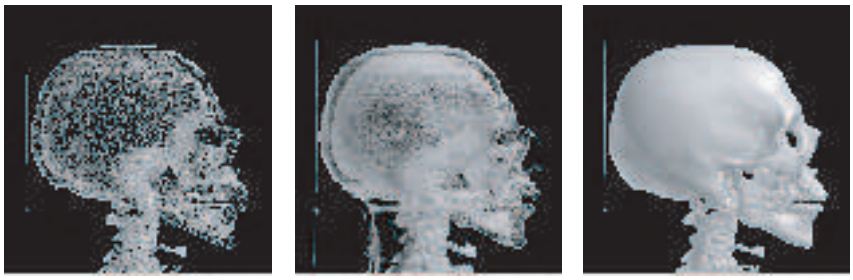


Figure 3.5: Results: (a) box pattern, (b) tent pattern, (c) ramp pattern

one, two and three. Above you see the transfer function sequence which will be generated (TF1, TF2, TF3). Another thing we can see is that the transfer functions of the sequence are nearly similar. This means each succeeding function in the sequence differs only from the function before by an offset of the pattern creation point.

To get galleries using the transfer function generation through organized sampling method which sufficiently cover every object in the dataset, we have to sample rather often. This increases the size of the galleries and they will be hard to handle and rendering them costs a lot of time. Our intention was to reduce the number of rendered images. But our focus was not only to reduce the number, we even wanted to have a meaningful spectrum of images. This means they have to be different and should depict as much information of interest as possible. In datasets are mostly some features we want to extract (e.g. bone, tissue, vessels, etc.). We wanted to design methods, which will enhance these features. We can achieve this enhancement through different approaches. The first method which comes in mind is based on the data value. The next section deals with this idea.

3.2.2 Refinement based on the data values

We search for big differences in neighbouring volume elements. This can be a hint, that there is a boundary between two objects. We use the density values there as sampling point in the data space of the transfer function. For the search of these differences we use the gradients. We can compute the gradient by

$$\bigtriangledown f(x,y,z) = \frac{\partial f}{\partial x}\widehat{x} + \frac{\partial f}{\partial y}\widehat{y} + \frac{\partial f}{\partial z}\widehat{z} = \begin{pmatrix} f(x+1,y,z) - f(x-1,y,z) \\ f(x,y+1,z) - f(x,y-1,z) \\ f(x,y,z+1) - f(x,y,z-1) \end{pmatrix} \quad (3.1)$$

where $\widehat{x}$, $\widehat{y}$ and $\widehat{z}$ are the unit vectors in the x-, y- and z-direction. This means the longer the gradient magnitude, the bigger is the value difference between adjacent voxels. We can assume on a position with a large gradient,

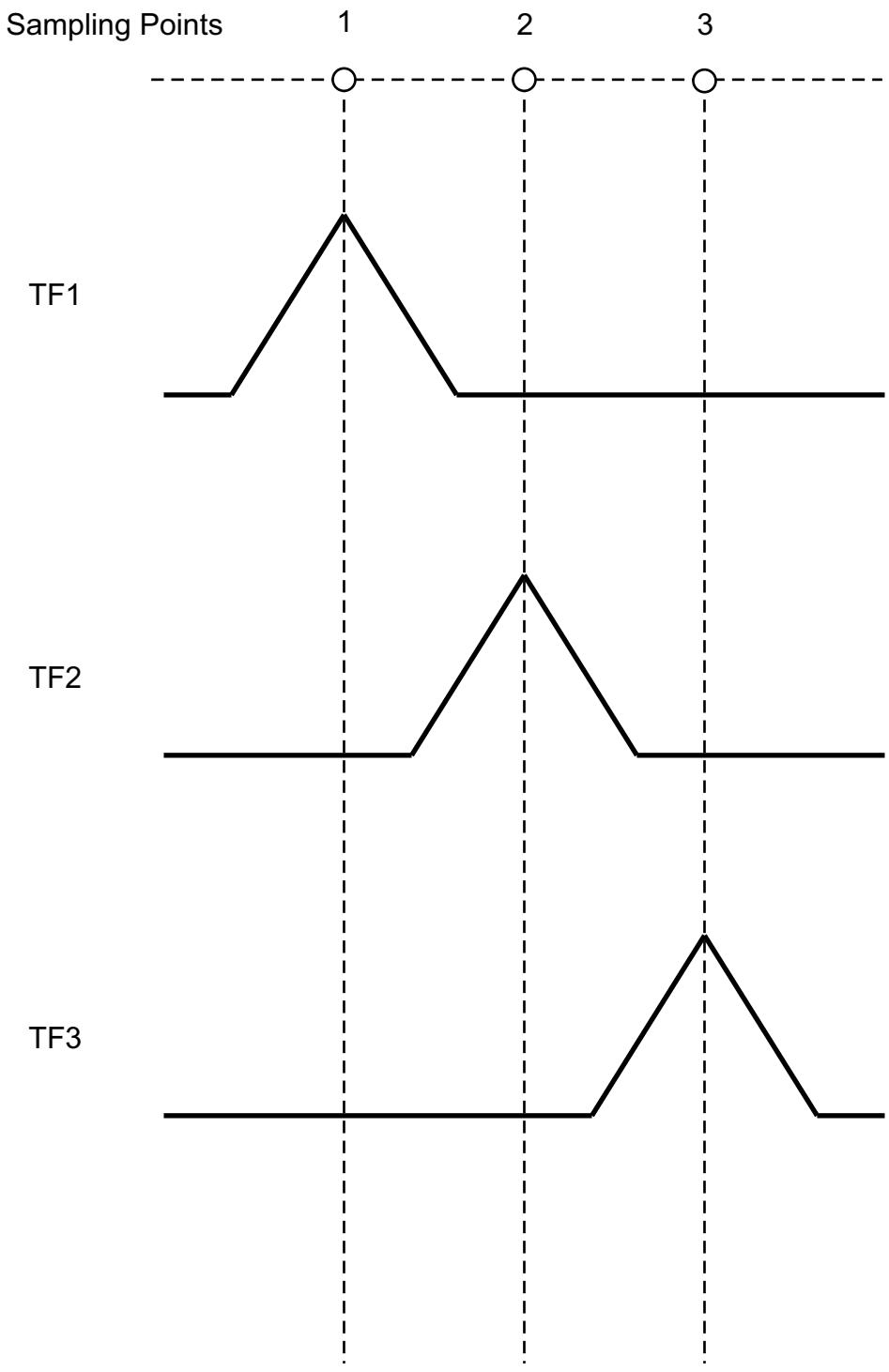


Figure 3.6: Overview of transfer function generation: Here by example the generation of the first three transfer functions of the sequence are depicted.

there is also a change in the material. As far as we want to display the different materials or better the border between them to distinguish the features in the dataset this can be a good measure to find this feature. So we can define a method for finding transfer functions to emphasize borders in the following way:

1. Specify a threshold (highest and smallest value) of the gradient magnitude

2. Go through the volume and calculate the gradient magnitude at every voxel

3. If the magnitude is between the threshold values and a transfer function using this density value as sampling point has not been already created, we create a transfer function with a pattern at this value.

The gradient magnitude can be seen as the first directional derivative (compare Equation 2.2) based on the voxel values. We can go one step further and calculate the second derivative. Then we can interpret the second derivative as magnitude of the gradient of the gradients' magnitudes - the curvature (compare Equation 2.3). Therefore we define an algorithm in nearly the same way as before, but instead of using the gradient's magnitude, we now use the magnitude of the curvature as reference value for comparing with the threshold values (low and high).

Both methods produce nearly the same result, but the algorithm for the second derivatives has a significantly higher runtime. As we can see in Figure 3.7 these methods generate very good results if we want to find isosurfaces. But both algorithms depend extremely on the underlying data. One problem is that there are usually a lot of differences, so we have a lot of different data values, for which the gradient value falls in the threshold interval. So if we choose for example the threshold values from 2350 to 2400 it produces a large gallery with 552 items. But if we choose a smaller interval with difference values of about 2000 we get a result like it is shown in Figure 3.7. Gradient values with values lower than 1000 even increase this problem, because the number of data values in the threshold interval even increases if we use smaller

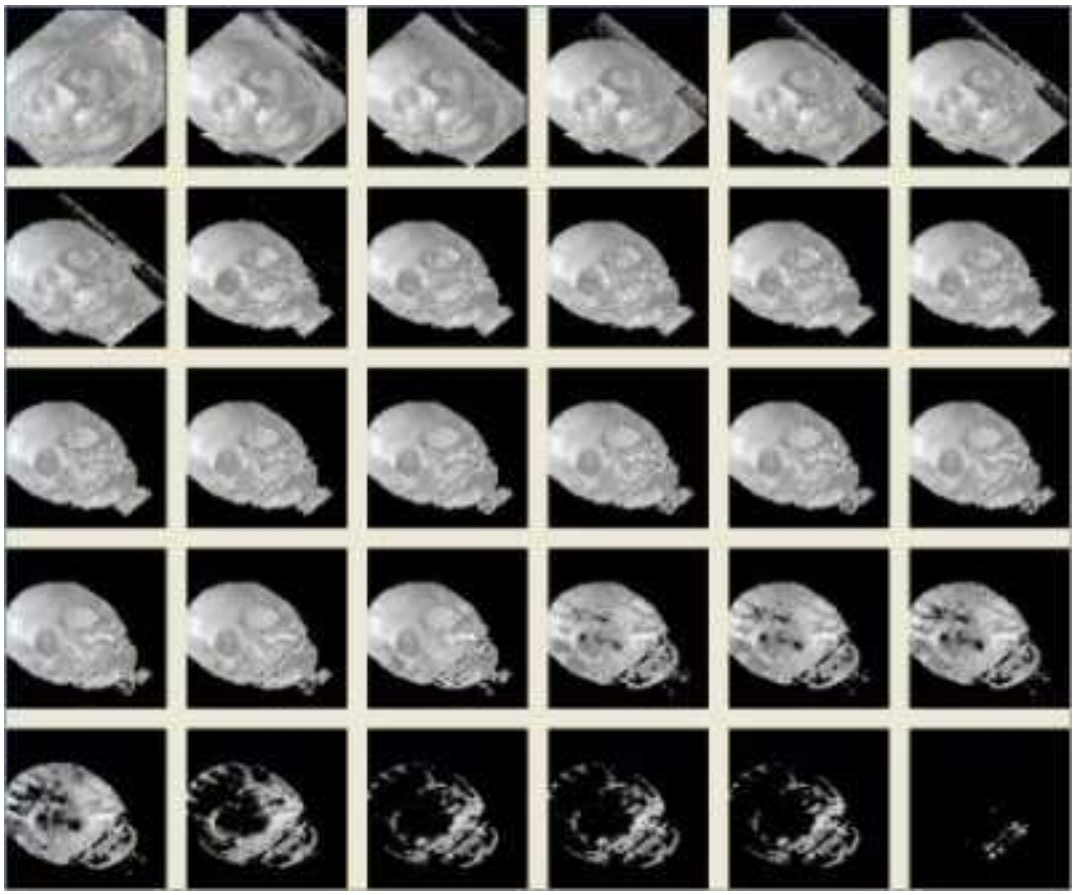


Figure 3.7: Gradient based method: Here a high threshold value of 2100, a low value of 2098 and the ramp pattern are used for transfer function generation

threshold values. But if we choose the threshold values too high (above 4000) we usually only get features with high densities (e.g. teeth) or no items.

Because of the small threshold intervals these two methods are rather limited. We tried another approach with the help of the data histogram to work around this problem.

3.2.3 Refinement based on the histogram

The data histogram is an array of scalar values, which lists for each density value the number of voxels which occur in the dataset. This array can be

easily produced by going through the volume and checking for each voxel the data value and increment the number of voxels which have been found with the same value in the array.

The first attempt was to apply a threshold method to the histogram, which was rather simple to implement by comparing each value in the histogram array with the threshold value. But this method does not lead to a good result, because of two reasons:

1. The characteristic of most datasets in medical visualization is that there are a lot of datapoints with low density values in it, because of the air which usually surrounds the scanned objects. This data is normally not wanted because it does not contain any useful information about the scanned object. These data items affect the thresholding method very strongly, because they represent far the most units in the dataset.

2. The thresholding process itself prerequesites a lot of knowledge, because we cannot state generally acceptable threshold values that will always lead to good results.

We have to think of a method to overcome this bad dataset characteristics. Our next idea was to use a method which measures differences of consecutive histogram values. The consideration behind that was that the different objects in the data are limited to some specific interval of data values and the following values should be very low until we reach the range of values for another object. We implemented a method which compares the difference of two consecutive histogram values with a threshold value. If the difference is bigger than the threshold value, we will place a pattern in the transfer function at the position of the first value. But as we can see in Figure 3.8 this method does not yield a good result. There are two reasons why this method does not work well:

1. The characteristic of the datasets, that we have a large amount of low data values of no interest also leads to big differences in this data range. So a lot of data values there are chosen for the patterns position.

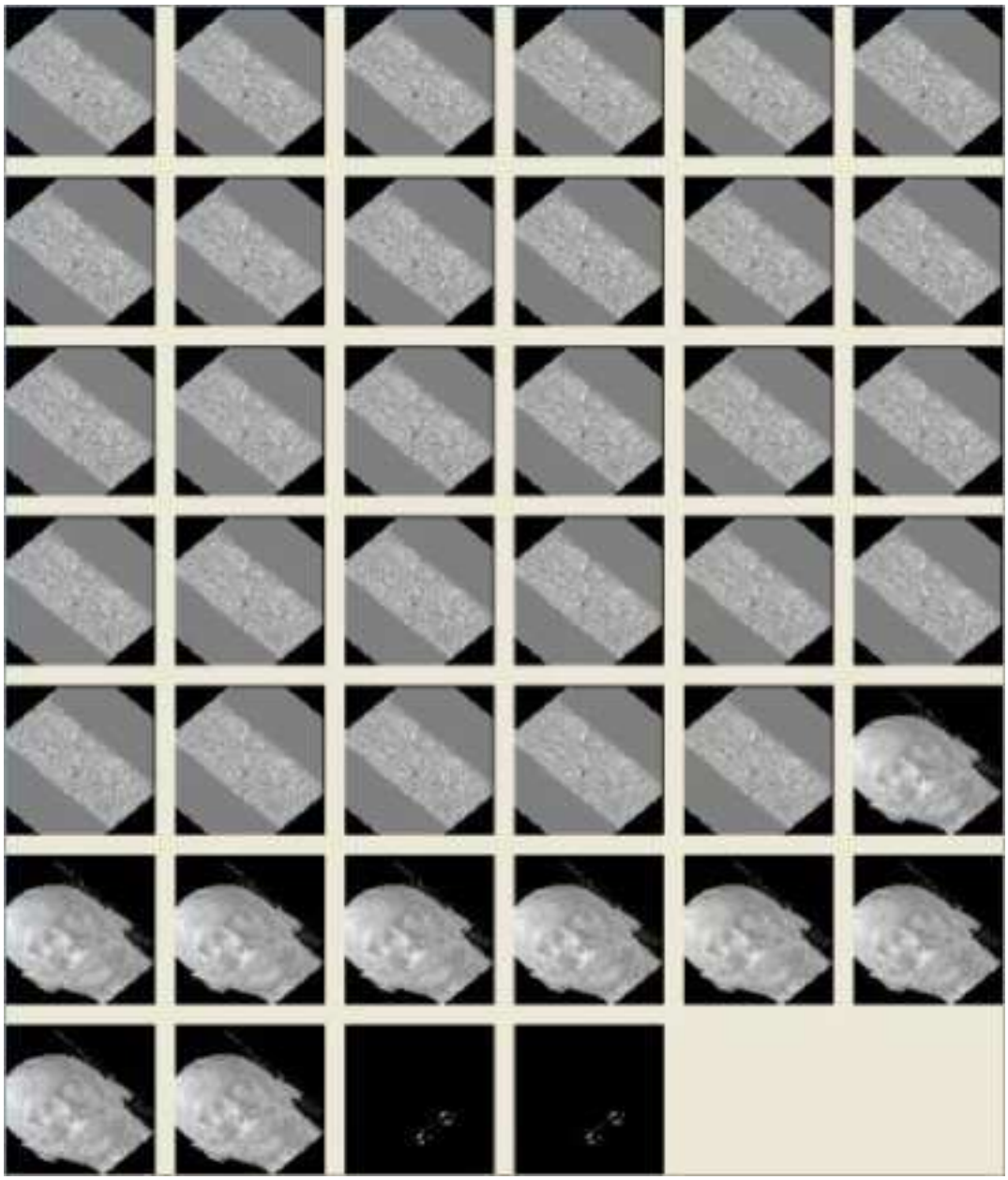


Figure 3.8: Result of Histogram Difference Thresholding with a threshold value of 3000

2. Our assumption that there are steep steps in the histogram between the data ranges of different objects is wrong. So the histogram seems to be more continuous between different objects.

The next method is based on finding local maxima in the sequence of histogram values. This algorithm extracts the peaks in the histogram as creation positions of the patterns in the transfer functions. We compute these positions by comparing consecutive histogram values and if we have a change from a monotonic increasing values to monotonic decreasing values, we have found a local maximum. This peak point is then used as creation position of the pattern in the transfer function. This method usually produces rather large galleries, because there are usually a lot of peaks in a histogram of a dataset. But we can use some image based method (these methods are discussed in the next section) to downsize the gallery (Figure 3.9). The local maxima method itself produces in this case a gallery of 990 items.

As conclusion of this section we can say that we have not found a method based on histograms to overcome the problem of the large amount of data with low density values. But even the last method seems to extract a lot of interesting images. The size of this gallery is far too big to provide the user a good overview. This leads us to the idea to add an extra processing step based on the produced images, to sort out similar images. We can generate galleries which give a representative summary of these large image collections. These image based methods for gallery reduction are discussed in the next section.

3.2.4 Refinement based on images

The following algorithms are based on a sequence of transfer functions like they are defined in a gallery configuration file and try to optimize this configuration for a special dataset. With these methods images are rendered, which are defined by the dataset and the transfer functions, and then these images are assessed according to some image measure or compared with another to get distinct images and reduce the size of the gallery in that way.

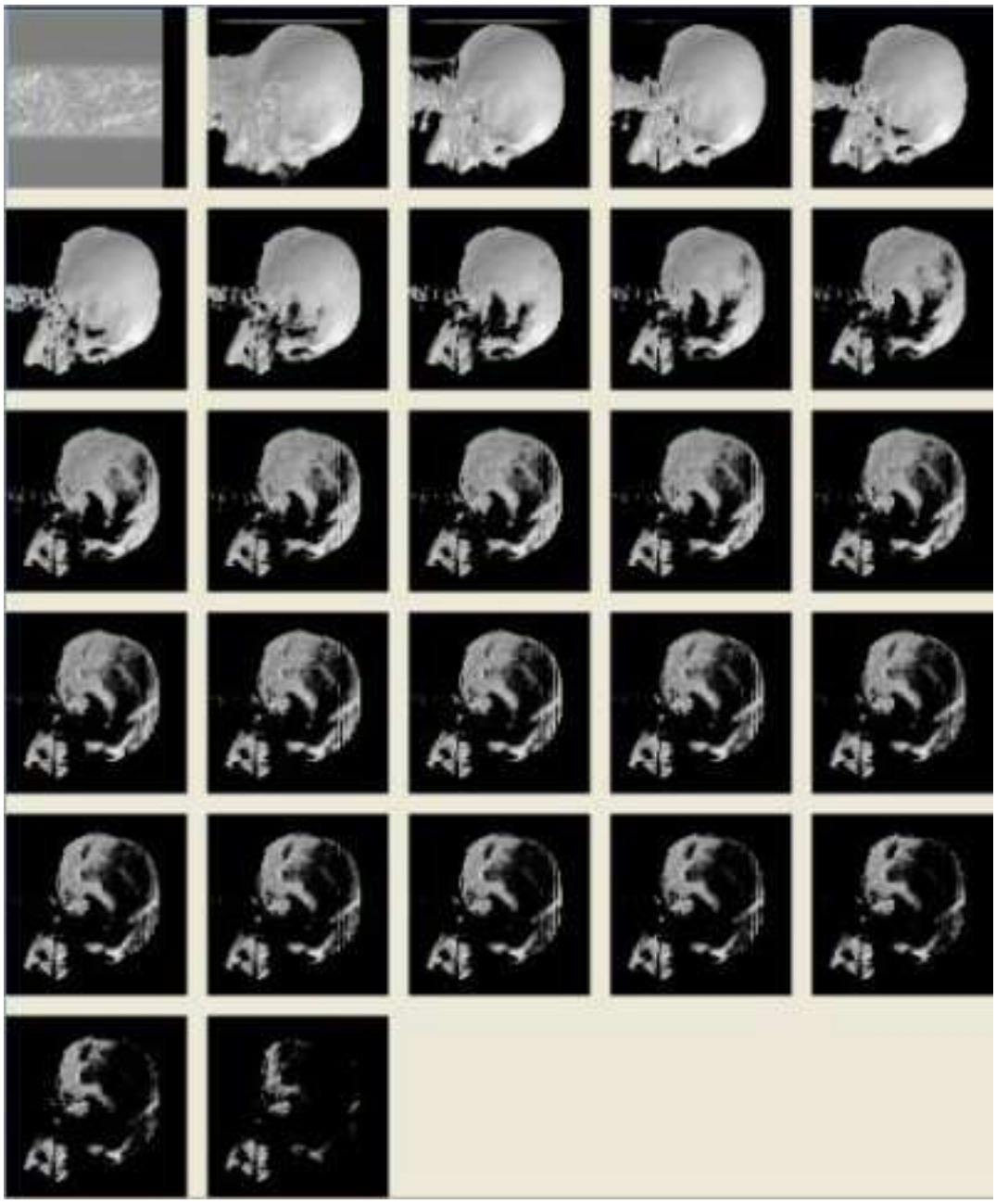


Figure 3.9: Result of Local Maxima Method: Here the local maxima method was combined with the pixel-difference method (Section 3.2.4) with a threshold value of 2500

We will discuss first the image measures and then we will go forward to their application in the different methods.

Image Measures

First of all we have to investigate the properties of the RGB-colorspace [34, 38], because the resulting images of our rendering system are RGB values. This colorspace consists of three stimuli for red green and blue, which are the three primary colors for an additive color system. This color system is mainly based on the color perception of the human eye. The photo-receptors for viewing colors are called cones. These cones respond to yellowish-green (564nm), green (534nm) and blue (420nm) light - in brackets we have mentioned the wavelength for each. The three primary colors result from the fact, that with light of red-, green- and blue-wavelengths these three type of cones can be stimulated nearly independently. We can now interpret this color system as a space with a three dimensional coordinate system. We can refer each of this three values to one coordinate axis. Each R,G,B-Triple has its point in this space (see Figure 3.10). Along with this information we can define a measure for the distance between two colors by the euclidean distance in that space:

$$dist(c1, c2) = \sqrt{(r_{c1} - r_{c2})^2 + (g_{c1} - g_{c2})^2 + (b_{c1} - b_{c2})^2} \tag{3.2}$$

Because of the fact, that algorithms sometimes work on luminance data, we have to calculate this value from the red green and blue values. For this task the following equation is commonly used [40]:

$$lum(c) = 0.3 * r_c + 0.59 * g_c + 0.11 * b_c \tag{3.3}$$

As we can see in the following image (Figure 3.11) this formula corresponds to the perception of the human visual system, where the blue factor contributes the fewest amount to the final luminance of the color and the green factor the most.

During the work of C. E. Shannon on information-theory [30] he defines the information content of a symbol or message by:

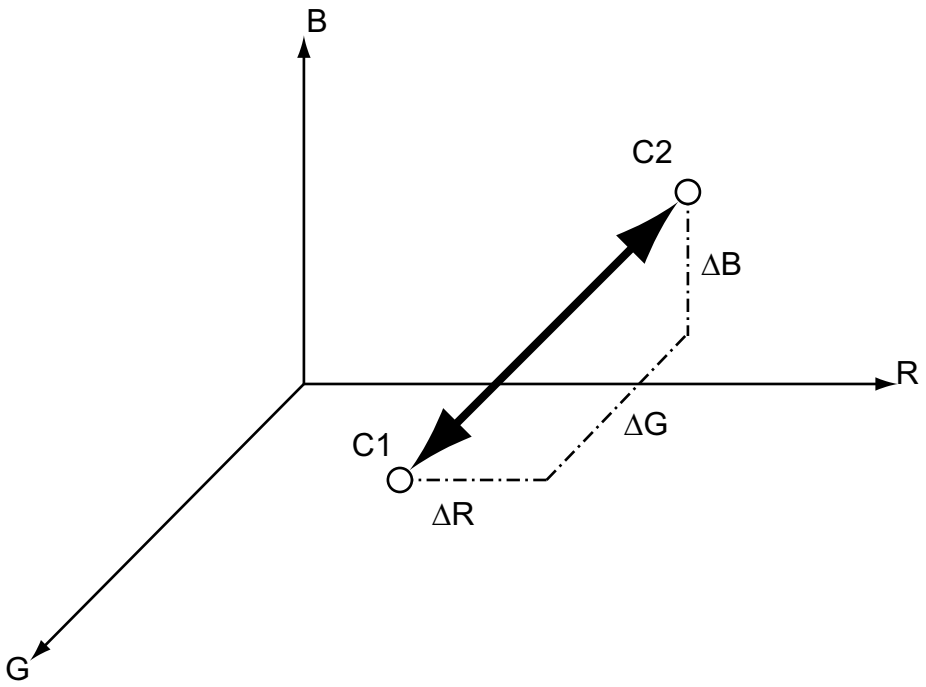


Figure 3.10: RGB Coordinate-system and euclidean distance

Figure 3.11: red, green, blue to gray level conversion

$$I(m) = -\log_2 p(m) \tag{3.4}$$

where p(m) is the probability of the occurrence of m in the output. So if the symbol or other pattern m occurs very often it has fewer meaning to the complete information, than a symbol that occurs fewer times. But also symbols which have nearly no occurance in the message have a lower meaning than others. So the complete information content of a string or another sequence of symbols with n different symbols (patterns) is defined by:

$$H(x) = \sum_{i=1}^{n} p(i) * I(i) = -\sum_{i=1}^{n} p(i) * \log_2 p(i) \tag{3.5}$$

This formula is also known as entropy. Our intention is to use this formula as a measure for information content in an image. For this topic we defined Algorithm 3.

Algorithm 3 Entropy of an image

1. We convert the RGB-image into a gray-level-value image with 255 gray-levels with the help of equation 3.3
2. In a first pass through the image the number of occurrences of each gray level is counted. After that we calculate the possibility of the occurrence of each gray level by dividing this value by the number of pixels of the image
3. Now we can use equation 3.5 to calculate the entropy of the image

Another measure based on images, which detects certain features like edges are gradients. For this task of finding gradients we again use the conversion of the RGB image to a luminance intensity image (see Equation 3.3). We can interpret this luminance image as a scalar field of values:

$$f : \mathbb{N} \times \mathbb{N} \rightarrow [0; 255] \tag{3.6}$$

This means we can define an operator to calculate gradients except at the boundaries of our image. We have to set a boundary condition. We can use that the gradient magnitude is zero at the boundary or forward and backward differencing as boundary condition. For the other positions we can define the gradient in the following way:

$$\bigtriangledown f(x,y) = \frac{\partial f}{\partial x}\widehat{x} + \frac{\partial f}{\partial y}\widehat{y} = \begin{pmatrix} f(x+1,y) - f(x-1,y) \\ f(x,y+1) - f(x,y-1) \end{pmatrix} \tag{3.7}$$

where $\widehat{x}$ is the unit vector in x-direction and $\widehat{y}$ is the unit vector in y-direction. We can use this operator to calculate the maximum or average gradient magnitude of an image to measure if there are strong edges in the image.

If we have defined the gradient magnitude as first order derivation of our scalar luminance image, we can go one step further. As we have defined the first order derivatives over the whole region of the image (including the boundary conditions), we can now define in nearly the same way a second order derivative of the luminance image. We can define the second order derivative in the context of the following scalar field:

$$g : \mathbb{N} \times \mathbb{N} \to \mathbb{R} : g(x,y) = \|\nabla f(x,y)\| \tag{3.8}$$

Because of this identity (Equation 3.8), the gradients of these field are defined by:

$$\nabla g(x,y) = \frac{\partial g}{\partial x}\widehat{x} + \frac{\partial g}{\partial y}\widehat{y} = \begin{pmatrix} \|\nabla f(x+1,y)\| - \|\nabla f(x-1,y)\| \\ \|\nabla f(x,y+1)\| - \|\nabla f(x,y-1)\| \end{pmatrix} \tag{3.9}$$

Refinement based on Image Entropy

The first idea was to use some measure for the information in an image. For this task we used the entropy [30] of an image. The method we use here compares the calculated entropy (see Algorithm 3) with a threshold value. If the entropy is bigger than this value the image is taken for the new gallery. We can manipulate the size of the gallery indirectly by defining

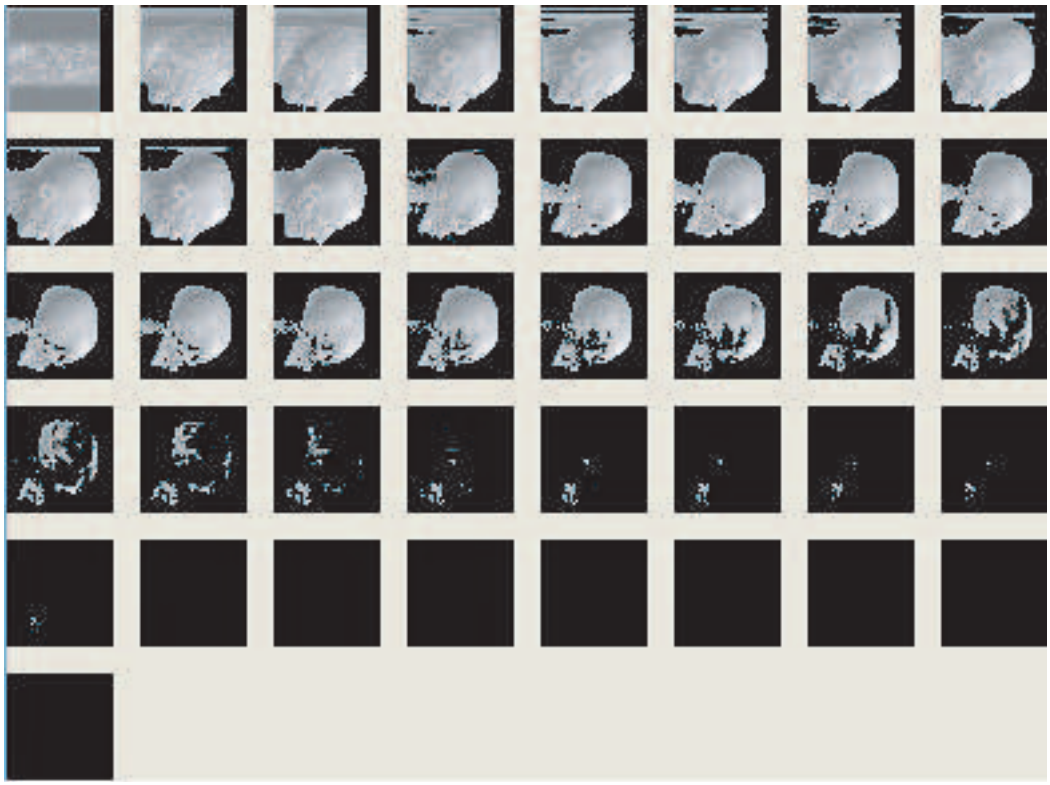
a bigger or smaller threshold value. This method can be used to sort out low information images of an existing gallery with the help of a threshold value. Because it is applied to an existing gallery the size of this collection is an upper bound for the size of the new gallery. For example we can generate a gallery first and reduce it to the images with the most information. Here we generated a gallery with the transfer function generation through organized sampling method with a sampling distance of 100 and the ramp pattern. Then we apply the entropy method to sort out the images with less information content. These are the images which only consist of a limited number of gray level values. You can see the results of this test in Figure 3.12. This method works very good and can be used quite easily to sort out low information images by using a small threshold value (e.g. 0.02). To verify this we tested the method on various datasets and the main viewing position along the x- y- and z-axis.

Refinement based on Image Gradient Magnitudes

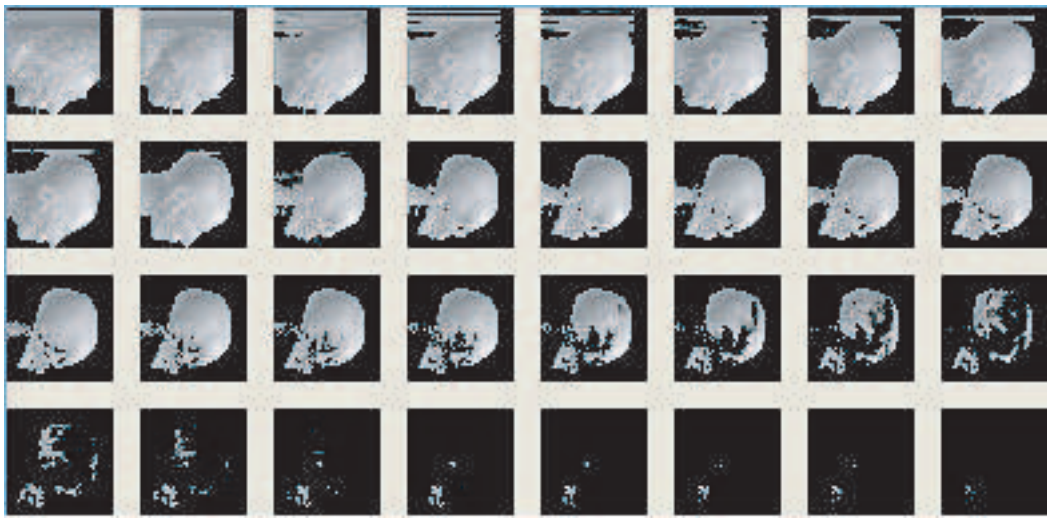
The next idea was to design a method to detect edges in images, because edges are usually a structure, which separate different objects in images. So our consideration was that if there are a lot of edges in the image, the image contains a lot of information of different objects in the dataset.

We have defined how to calculate image gradients (Equation 3.7). We will use that definition to design two algorithms based on gradients. The one algorithm measures the maximum gradient magnitude in an image and the other the average gradient magnitude. Both methods compare the calculated value with a threshold value. The significant difference in the results between them can be seen in Figure 3.13. We see that in the images of the maximum method we get finer details, whereas the average method selects also images which have a lot of small gradients.

We have also designed algorithms which are based on the curvature (Equation 3.9). We have again implemented methods to compute the maximum and the average value of this measure in an image. These values are again compared to a threshold value. If we compare both of these methods



(a)



(b)

Figure 3.12: Filtering images by image Entropy: (a) First we created a gallery with the transfer function generation through organized sampling method with a sampling interval of 100 and the ramp pattern (b) Then the images of less information are filted out with the help of the entropy method with a threshold value of 0.02

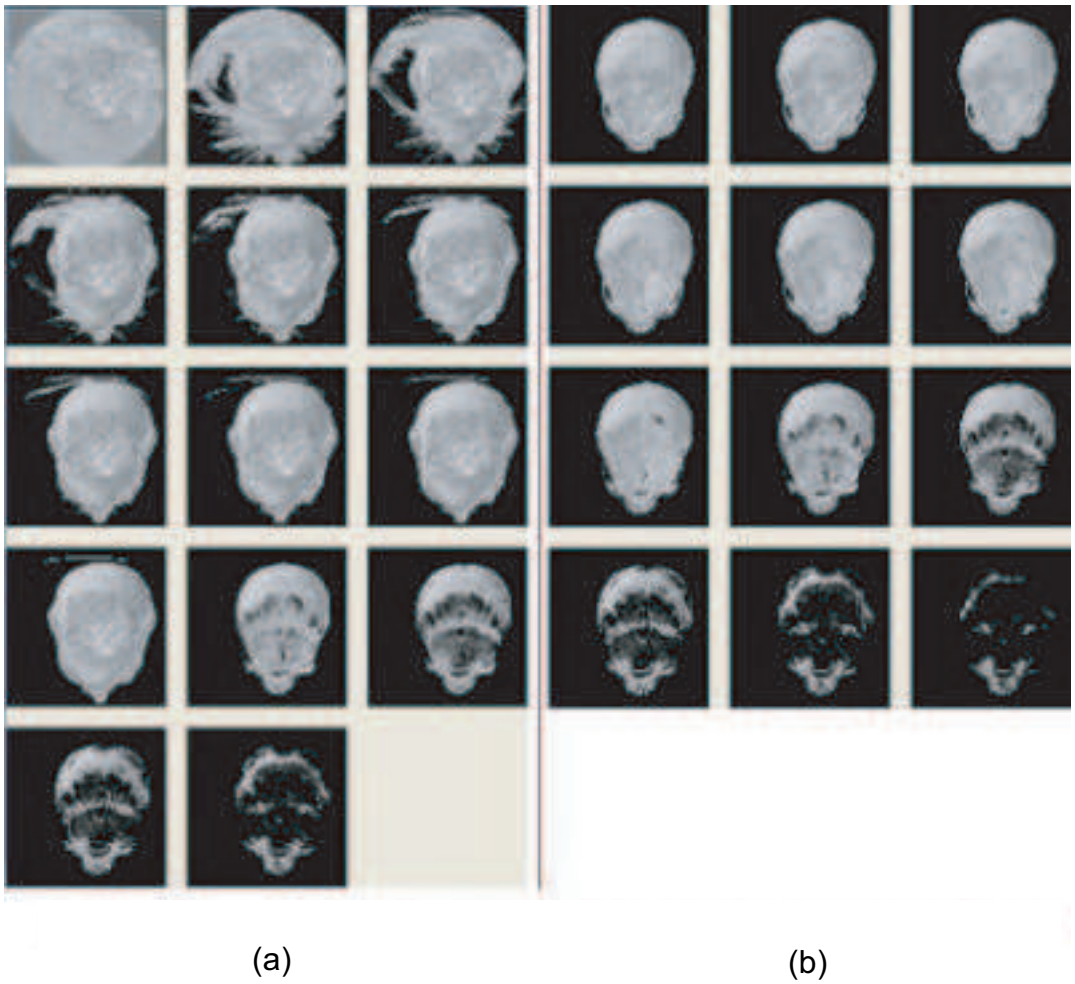


Figure 3.13: Comparison of Maximum and Average Gradient Magnitude Methods: We see that in (b) the maximum method selects more images with fewer objects which have high gradients than the average method in (a)

again, we can see the same effect as in the results of the methods which are based on the gradients: The maximum method usually shows finer detail than the average method, which selects images with more smaller gradients (Figure 3.14). We can also compare the results of the gradients methods and those of the curvature information methods. So we see that in the second group of algorithms there are more images with finer details selected. This corresponds to the fact that in the calculation of the curvature we emphasise edges more in the image by the second differential step. We can use these methods easily to create smaller galleries with the help of the threshold value.

Refinement based on Image Difference Calculation

We can consider that usually succeeding transfer functions in this sequence will create more similar images in the gallery than compared to the other functions in the sequence. Therefore our next idea was to design methods which compare the images of these neighbouring functions. These algorithms use difference calculation between images to select appropriate images. The question now is how do we measure the difference between images. As our images consist of pixels, which are defined by RGB-colors, we can use the RGB-RMS-color-difference formula (Equation 3.2) to measure the difference between two pixels. We can use this to define methods to compute the difference between two images. The first algorithm - the summed difference method - sums up the pixel differences and returns this sum as a measure of the difference between these two images. This is easy to implement:

1. We resample the two images to the same resolution.
2. We compare the pixel at each position from "image one"with the pixel of "image two"and calculate the difference
3. We sum up these measured differences

This calculation ends in some for the user unintuitive value and is therefore difficult to use, but it is a more precise measure than the second one we have implemented: In this method - the pixel-difference method - we define

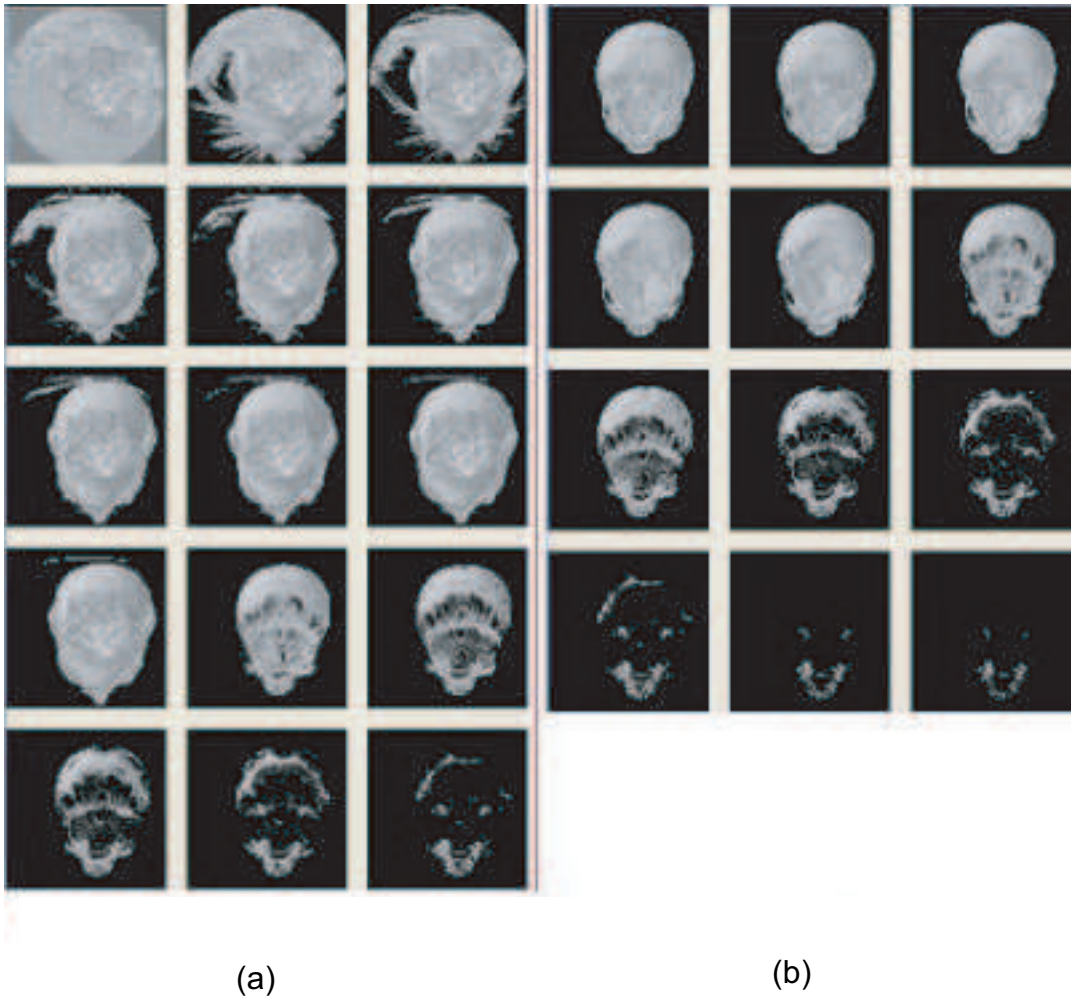


Figure 3.14: Comparison of Maximum and Average Curvature Method: We see that in (b) the maximum method selects more images with fewer objects which have high gradients than the average method in (a)

a threshold which is used to measure above which value the pixels should be identified as different. Then we count the pixels over an image which are different, this means the distance of their colors are greater than the threshold value.

We can use these measures to reduce the gallery size. We define a threshold value to measure how different the images in the gallery should be. In these methods we take the first image and put it in the final gallery. If the difference between this image and the next image is bigger than the threshold value we add the next image also to the gallery. If the difference is smaller than the threshold value we do not put the image in the final gallery. The next comparison is made between the image which was chosen last for the gallery and the next which was not already measured. We continue this procedure until we have reached the end of the gallery.

In this application the distance measure of counting different pixels will be more easily to handle - which means to define a threshold value. For the user this value is more obvious than the summed difference of color values. But as we have stated before the summed difference method is more precise (Figure 3.15). We can see in the figure, that the last two images were not chosen by the pixel-difference method, because the values of that images were smaller than the threshold value. The summed-difference method gives us more power to fine tune between the images which we would like to have and is not bound that much on the image resolution given by the system.

A third more advanced algorithm - advanced image selection method - was also created, which uses the summed-difference method. With this algorithm we can determine the final gallery-size explicitly and precisely by a number. Whereas in the other algorithms this can only be handled indirectly over the threshold value.

The method first selects the image from the middle of the source gallery for the final gallery. Then it tests which image in the source gallery is the most different from it and puts this image in the final gallery. The images which have already been taken into the final gallery are removed from the source gallery. So that we do not select the same image twice. The following images are tested against the images which are already in the final gallery.

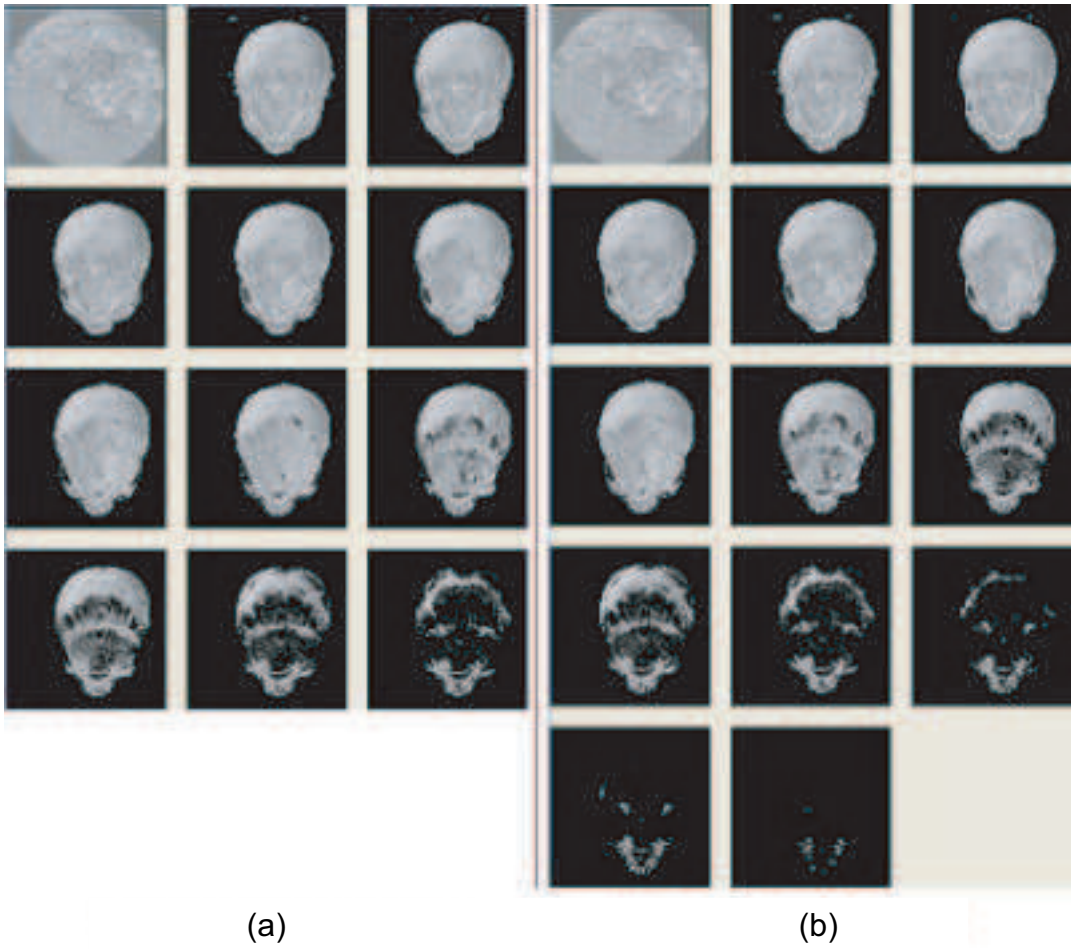


Figure 3.15: Comparison of Pixel- and Summed Image Difference Method: (a) Pixel Difference Method (b) Summed Image Difference Method

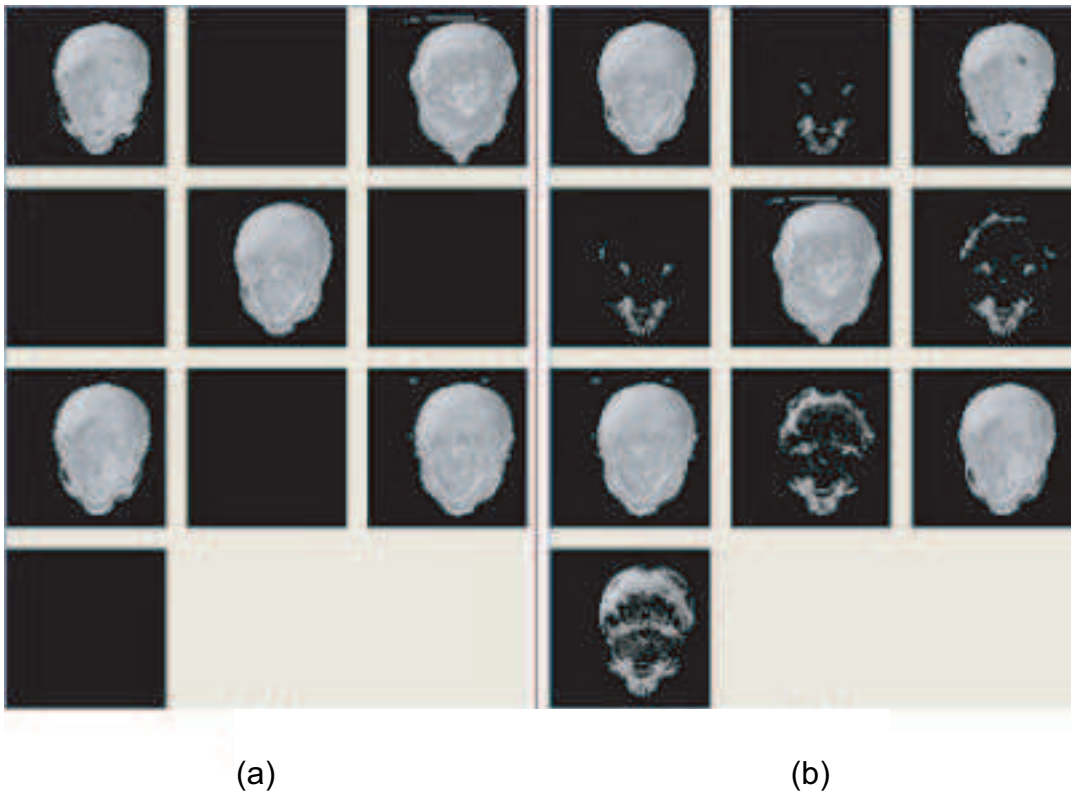


Figure 3.16: Advanced Image Selection Method Comparison: (a) shows the Advanced Image Selection Method applied on the gallery which was created by the brute force method with the ramp pattern. (b)shows the same method except that the gallery was before preprocessed with the average gradient method with a low threshold value (0.01).

This procedure is continued until we have reached the final gallery size. This method has some problem if there are too many black images in the source gallery. It selects black images multiple times because they are often the most distant to the images which have some content (Figure 3.16 (a)). This is the fact because we sum up the differences to the images in the final gallery. We can overcome this problem by removing the black images first from the source sequence of images. For this task we can use for instance the image entropy or gradient methods with a low threshold value (Figure 3.16 (b)).

3.2.5 Refinement based on Genetic Algorithm

In the following section we will present a genetic algorithm for searching the space of possible transfer functions [12] to find a population of transfer functions for which the rendered images correspond best to our expectation of finding different objects in the dataset.

First we will talk about genetic algorithms in general and then we will discuss the specific problems in our task of finding informative images.

Genetic Algorithms in General

Genetic algorithms [31, 17, 28, 29] try to copy the evolution process of nature to find solutions not only for combinatorial problems, but also for optimization of non-linear structures and continuous parameter optimization. It was introduced by Holland [14]. It is a general principle and can therefore be applied in different fields of problem solving. One common fact in these algorithms is that we are not only operating on a single item, but on a group of items (the individuals). This group of individuals is also called population. As already mentioned we try to copy the evolutionary process, which is used in nature. So we have to re-engineer this process for the problem we want to solve. We will present here a short summary of the most important principles of the evolutionary process in nature:

1. Selection: As Charles Darwin has defined in his work "On the origin of species by means of natural selection or the preservation of favoured races in the struggle for life" in 1859, selection is the process of choosing the individuals, which are the best (strongest) to survive and for reproduction.

2. Recombination: Process of reproduction of new individuals from the old ones.

3. Mutation: Random event which can lead to a new development in the reproduction process. If it leads to better individuals these are preferably chosen in the selection process otherwise the mutation does not take any effect.

As we have already come across the term individuals it is clear that we need some encoding for the individuals which form the population. Often this encoding is named chromosome and contains all parameters of possible solutions of the problem. These chromosomes are often represented as a vector of items, which are also named genes. Because a chromosome is the encoded form of a solution, which is also named genotype, there is obviously also the decoded form of this genotype - the phenotype, which is the solution of the problem we want to solve.

Next we will present how such an genetic algorithm works (Algorithm 4) then we will present some recombination operators and make some remarks on the use of genetic algorithms.

Algorithm 4 Genetic Algorithm

1. set generation g to zero
2. initialize the population: P(g)
3. evaluate the population P(g)
4. As long as termination condition is nor met do
   (a) $g = g + 1$
   (b) $P_s(g) = select(P(g-1))$
   (c) $P_r(g) = recombine(P_s(g))$
   (d) $P(g) = mutate(P_r(g))$
   (e) $evaluate(P(g))$
5. end

In a genetic algorithm we select chromosomes then recombine them, then mutate them and finally evaluate the new population. We have to provide a calculation of the fitness of each individual to evaluate the population. We have to define a measure how good the chromosome approximates the solution of the problem we want to solve. This fitness function can also be used in the selection to do a fitness proportional selection, which corresponds to Darwin's definition of selection.

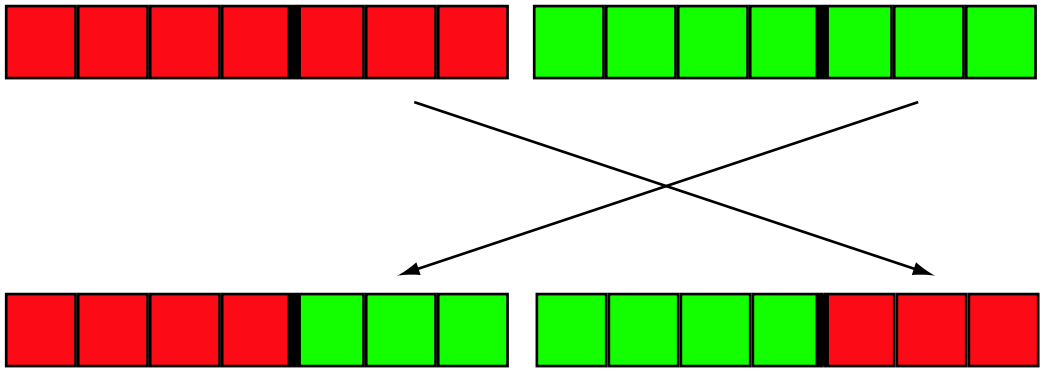


Figure 3.17: 1-Point-Crossover

The termination condition of the genetic algorithm can be for instance a maximum generation limit or a maximum / minimum fitness value. This depends on the fitness calculation and the problem we want to solve - if this fitness value should be maximized or minimized.

For the recombination of two chromosomes there are usually crossover-operators used. We can distinguish between 1-point-crossover, 2-point-crossover, multipoint-crossover.

1-point-crossover is rather simple: We divide the chromosomes in two parts for each chromosome at the same position and exchange the last parts (Figure 3.17). 2-point-crossover just selects 2 positions where the part in between these two points is exchanged (Figure 3.18). Multi-point-crossover is a generalization of these process to more partitions. One more operator is the uniform-crossover operator. It decides for every gene if it is kept or exchanged.

For mutation there should be an operator implemented which generates small random changes in the chromosomes. As a consequence we get for instance a new direction in searching our space of solutions. If the changes, which are made in this step, are too big, the genetic algorithm continuously jumps to different positions in the search space. The local solutions at each position will not be properly examined. The search process will degenerate to a random search of the whole space of solutions, which has nearly no chance to find the optimum in a big parameter space.

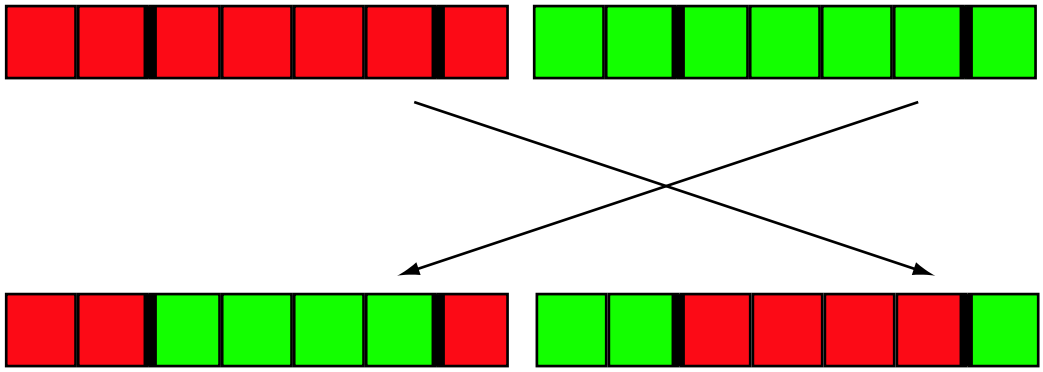


Figure 3.18: 2-Point-Crossover

One important factor in genetic algorithms is the selection pressure, which identifies how much a better individual is preferred over an individual with a worse fitness. If the selection pressure is too high the best individuals will increase too fast and therefore the different individuals in the population will decrease very fast - this means that the algorithm will often converge too fast to a locale optimum and cannot find the global optimum. If the pressure is to low there is nearly no increase in the best individuals and a lot of chromosomes with a bad fitness will be in the population. This means that the genetic algorithm converges very slowly.

Genetic Algorithm for the selection of informative images

The first issues we will meet are the issues regarding encoding of the genotypes and how to get the phenotypes. Then we will talk about different fitness functions and how we have implemented the selection task, followed by the definition of the recombination and mutation operators.

The phenotype is the final image, because this is the goal we want to reach - a good informative rendering. In our case the genotype is also obvious, it is the transfer function, as this is the main structure we can manipulate to get a different output in the images. We have defined the genotype as a piecewise linear function and the phenotype as an RGB-image. Now we have to specify a fitness function which should be applied on the phenotype to judge how go the result is. We have already defined such estimators in the previous

chapter, so it is an easy task to use them again. We use the entropy as a measure of how much information is in the image and the average gradient magnitude estimation as a measure of how many edges are depicted in the image. So we can again apply the methods of the previous section for this task.

The next topic we have to think about is the selection. We have implemented a fitness proportional selection. We implement this by generating an array of multiple individuals from the population. The number of each individual in this array depends on the fitness of the individual. The higher the fitness value is, the higher is the number of items of this individual in the array. Then we select a random number and take the item which is at the specified position in the array. This method is also known as roulette-wheel selection. So we select the individuals from the current population which should be recombined with each other. The number of selected individuals for recombination depends on the recombination rate, which is a factor of how many items of the new population should be created through recombination. The other items will be selected randomly from the old population to get again a population of the old size.

We have already talked about recombination with the help of the crossover operation. Here we will specialize the 1-point and 2-point-crossover for our domain (transfer functions). In the 1-point-crossover method we have to select a position where we want to separate the two transfer functions to exchange the second part with each other. This position is randomly chosen. We have to add controlpoints at the separating position and right before it, before we can exchange these parts which is simply done by exchanging the controlpoints in this part of the functions. The color and opacity values at these positions can be looked up in the lookup table of the transfer functions. So we create two new transfer functions out of the selected ones. These individuals are therefore usually named children of the two selected individuals.

The algorithm for 2-point crossover works analog except we have two more pairs of additional controlpoints to set we have two seperating points. We randomly choose two positions for the separating points. Then the internal

section between these points is swapped between the two transfer functions.

After the recombination and we have to refill the new population to the size of the old one by adding randomly chosen individuals from the previous generation. Afterwards we can apply the mutation. For this operation we have designed two algorithms: one which adds a controlpoint at a random position and another which randomly changes the existing controlpoints in opacity. The chromosomes, which should be mutated are selected randomly from the population. The number of selected individuals for the mutation is controlled by the mutation factor.

We have also applied another feature to this algorithm which is not generally part of a genetic algorithm: We have added a forward factor. This means we add for a specified number of the whole population the best individuals of the old to the new population. This increases the selection pressure. So this feature should be used with caution, because as we have explained before this can yield to an unwanted restriction of the search area.

The disadvantage of these stochastic search techniques is in general, that it always takes a lot of time until the algorithm converges to a final result, and the algorithm is very sensitive on the initial parameter settings So some parameter settings lead to a faster converging search process, but the solution would only be the local optimal solution of a small search space. We see that the result of such a fast converging search process is that there is no improvement in the fitness value (Table 3.1 the result with 30% forwarding). Or the setting can lead to a search process which continuously jumps to different positions in the search space. So the local neighbourhood of solutions at each position is not properly examined. This degenerates the search process to a complete random search, which has nearly no chance to find a good solution. We can see that in Table 3.1 the result with 90% mutation produces such a search procedure and therefore the resulting fitness (3.414) is not much higher than the initial one (3.115).

In the test runs we have made with different datasets, we have found out that setting the mutation rate to about 30 percent and the recombination rate to 70 to 80 percent are good values and will lead to a good convergency behaviour. The tests also showed that the two different fitness functions

For	Fitness : Init. Fit.	Recombination		Mutation		Gen	Runtime	Final Fit.
		1-Point	2-Point	Man.	Ins.			
0%	Ent.: 3.113		70%		30%	100	3m 51.515sec	4.227
0%	Ent.: 3.113		70%		30%	1000	35m 18.394sec	4.098
0%	Grad.: 0.137		70%		30%	1000	34m 39.49sec	0.312
0%	Ent.: 3.113		70%		30%	10000	6h 46m 49.03sec	5.344
0%	Ent.: 3.113	70%			30%	10000	6h 40m 20.05sec	3.211
30%	Ent.: 3.113		70%		30%	1000	35m 20.05sec	3.113
0%	Ent.: 3.113		70%		90%	10000	6h 48m 14,15s	3.414
0%	Ent.: 3.113		70%	30%		10000	6h 19m 40,323s	3.115

Table 3.1: Test-runs of Genetic Algorithm (For = Forwarding Factor, Init. Fit. = Initial Fitness of the best individual, Ent. = Entropy Method, Grad. = Average Gradient Method, Man. = Controlpoints Manipulation Method, Ins. = Controlpoint Insertion Method, Gen. = Generations, Final Fit. = Final Fitness of the best individual

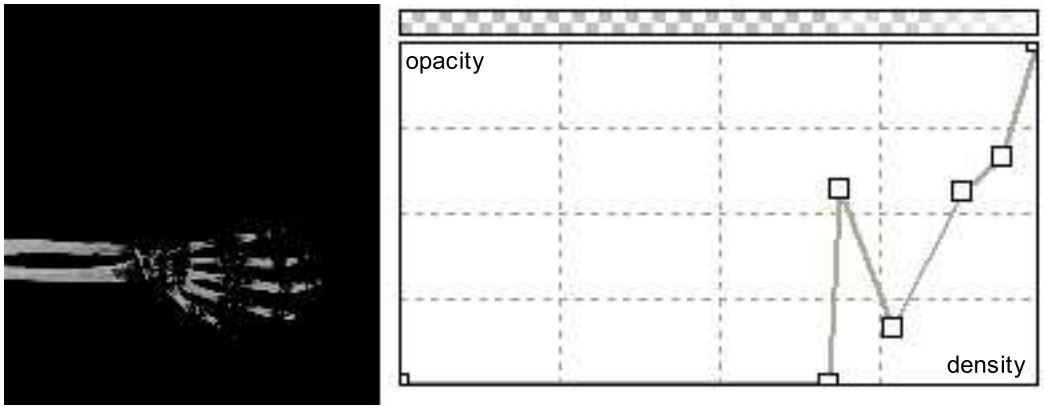


Figure 3.19: Best Result of the Gradient Fitness Function after 10000 Generations, Viewaxis: Z-Axis

(the first optimizes the average gradient magnitude of the resulting images and the second the entropy of the images) will lead to rather different best results (compare Figure 3.19 and Figure 3.20). This can be explained by the different goals of the gradient fitness function (improving edges) and entropy fitness function (improving information content in the image). In the tests it was also found out that the 1-point-crossover method will lead to a slower convergence of the genetic algorithm than the 2-point-crossover. In the context of the mutation algorithms it turned out, that the controlpoint insertion is better and leads to a faster convergence of the algorithm than the controlpoint manipulation method.

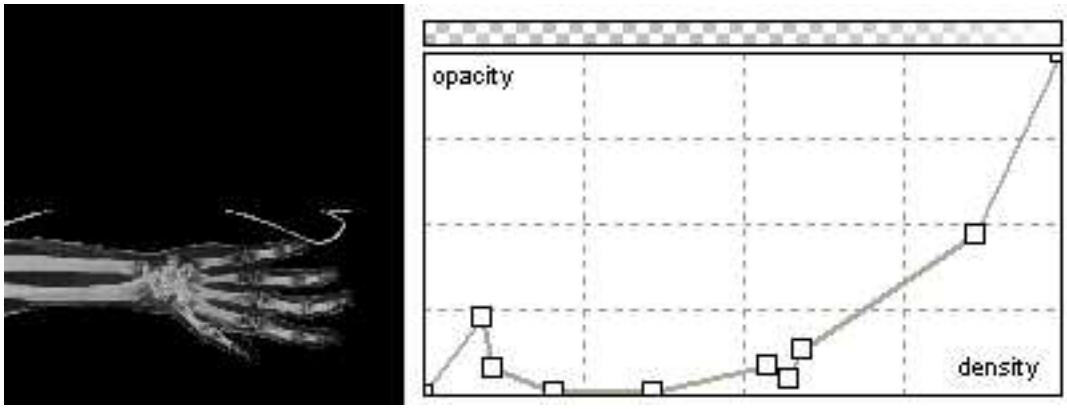


Figure 3.20: Best Result of the Entropy Fitness Function after 10000 Generations, Viewaxis: Z-Axis

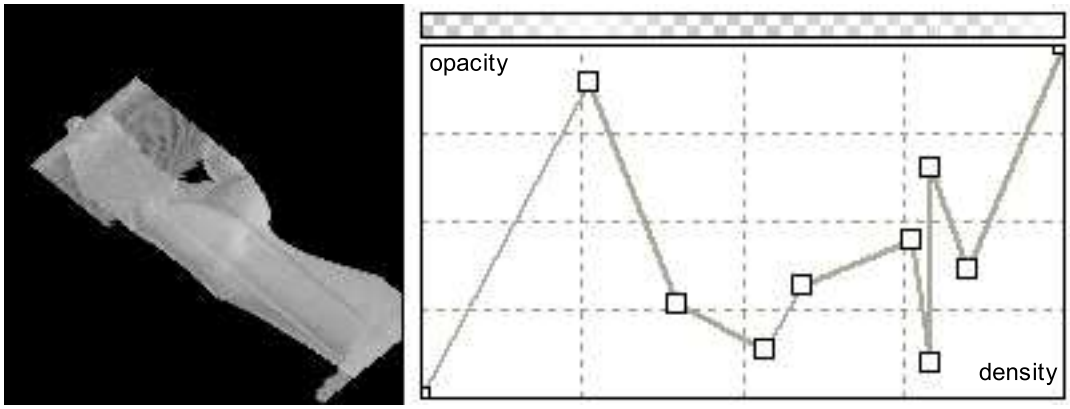


Figure 3.21: Best Result of the Entropy Fitness Function after 10000 Generations, Viewaxis: Y-Axis

We have not discussed until now that the image based methods are very dependent to the viewpoint. This leads also to big differences in the results of the genetic algorithm (compare Figure 3.19 and 3.21). We see that this leads to different results in the images and in the transfer functions. It is usual to genetic algorithms that the best results differ from time to time even if we start with the same initial solution. But we can see that in this case the differences are too big that we can assume that this variance in the fanal solution is based on the random behaviour of the genetic algorithm.

We have seen through out this chapter that the image based methods help a lot to avoid similar results in a gallery. The great disadvantage of these methods is the view dependency. We cannot be sure if we lose some good transfer functions through the fact that these functions will produce informative images by viewing the object from another viewpoint. Even more there can be cases that the results of the transfer functions are rather good from one viewpoint, but not from another one.

Chapter 4

Implementation

When there were no computers programming was no problem. When we had a few weak computers, it became a mild problem. Now that we have gigantic computers, programming is a gigantic problem

E. W. Dijkstra

In the following chapter we will give a short overview of the main framework (Figure 4.1) for volume visualization, which was programmed as part of this diploma thesis. Then we will present the tools which were used to analyse the different methods of gallery generation.

4.1 The Main Application

The idea was to design a framework for different methods in the field of medical visualization. Therefore the system has to be easily extendable, this means we need a modular design to integrate new functionality. We have created a scalable concept which is based on mainly five parts (Figure 4.2):

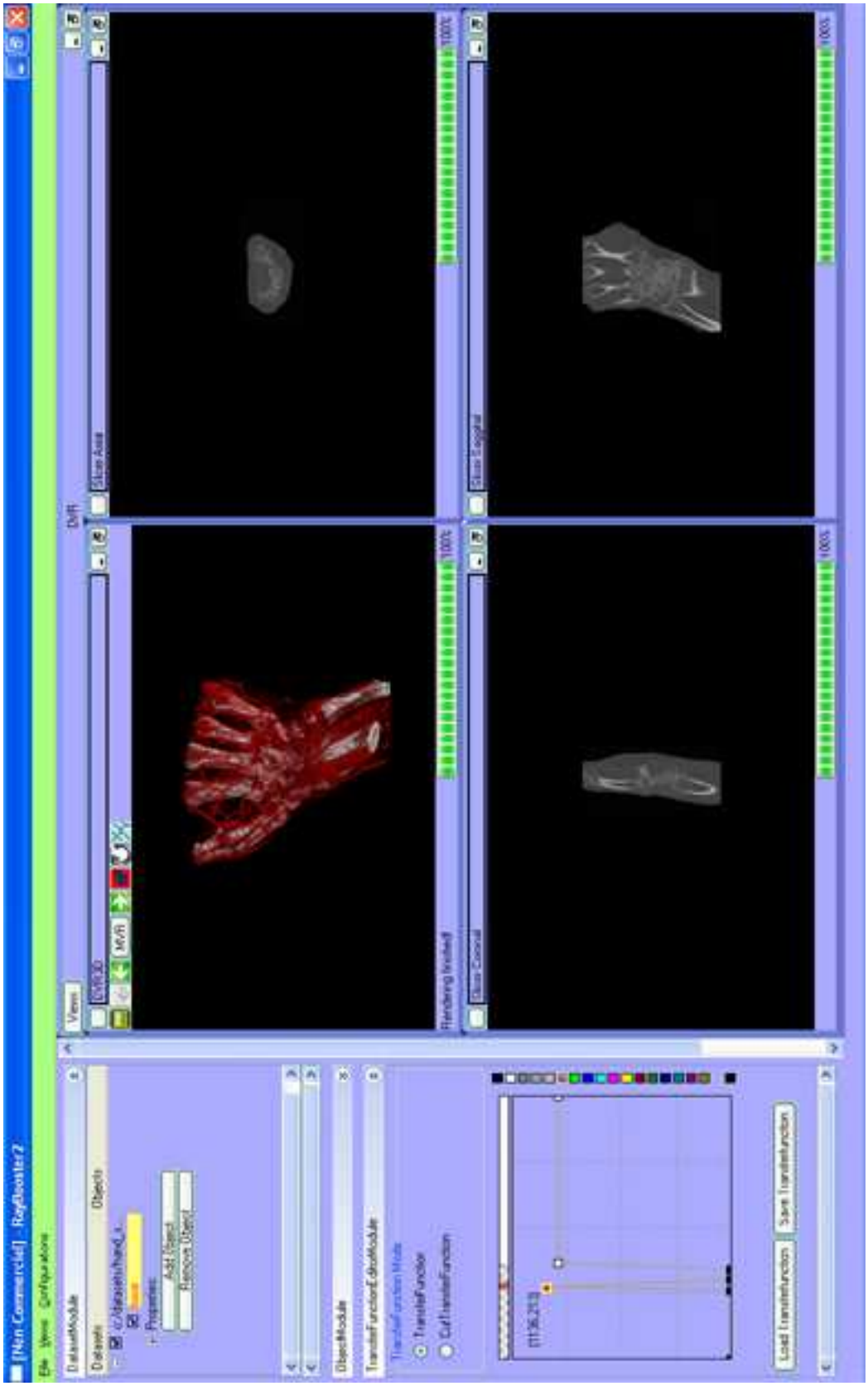


Figure 4.1: The Application

- Configurations
- Views
- Modules
- Renderers
- Actors

We will begin to describe them at the Renderers. These modules create the images, which are displayed in the View to which the renderer is applied to. The actors manage user interactions which are directly created at the space inside of the views like mouse moves and clicks and keyboard events. These events are forwarded to the renderers to set the necessary parameters there. The View is the visible GUI-frontend of one or multiple renderers and can be resized and has a basic menu where the user can save the current contents of the view in an image or create short movies. Some special views also have a toolbar to switch between different interaction modes - especially for manipulation of camera or object positions in 3D.

The top-level container which can group such views is the configuration. A configuration can have multiple views and modules. The modules and views which form such a configuration are defined in a separate file.

Modules are the items positioned at the sidebar, these items can be hidden or fully displayed by switching the arrows button (Figure 4.3). These items can be used to define parameters for the whole application (e.g. the preferences module to define the colors of different GUI-parts of the application) or just a single renderer (e.g. SlicerRenderer or DVRRenderer).

The user can manipulate the layout of such configurations by defining which modules and views should be loaded. In the application a user can then load and unload these different configurations at runtime by selection of the configuration in the Configurations-menu from the main menubar (Figure 4.3). If a user loads two or more configurations the views will connect and disconnect the signals to slots of the modules as the user switches between this configurations. This connect- and disconnect-process is also

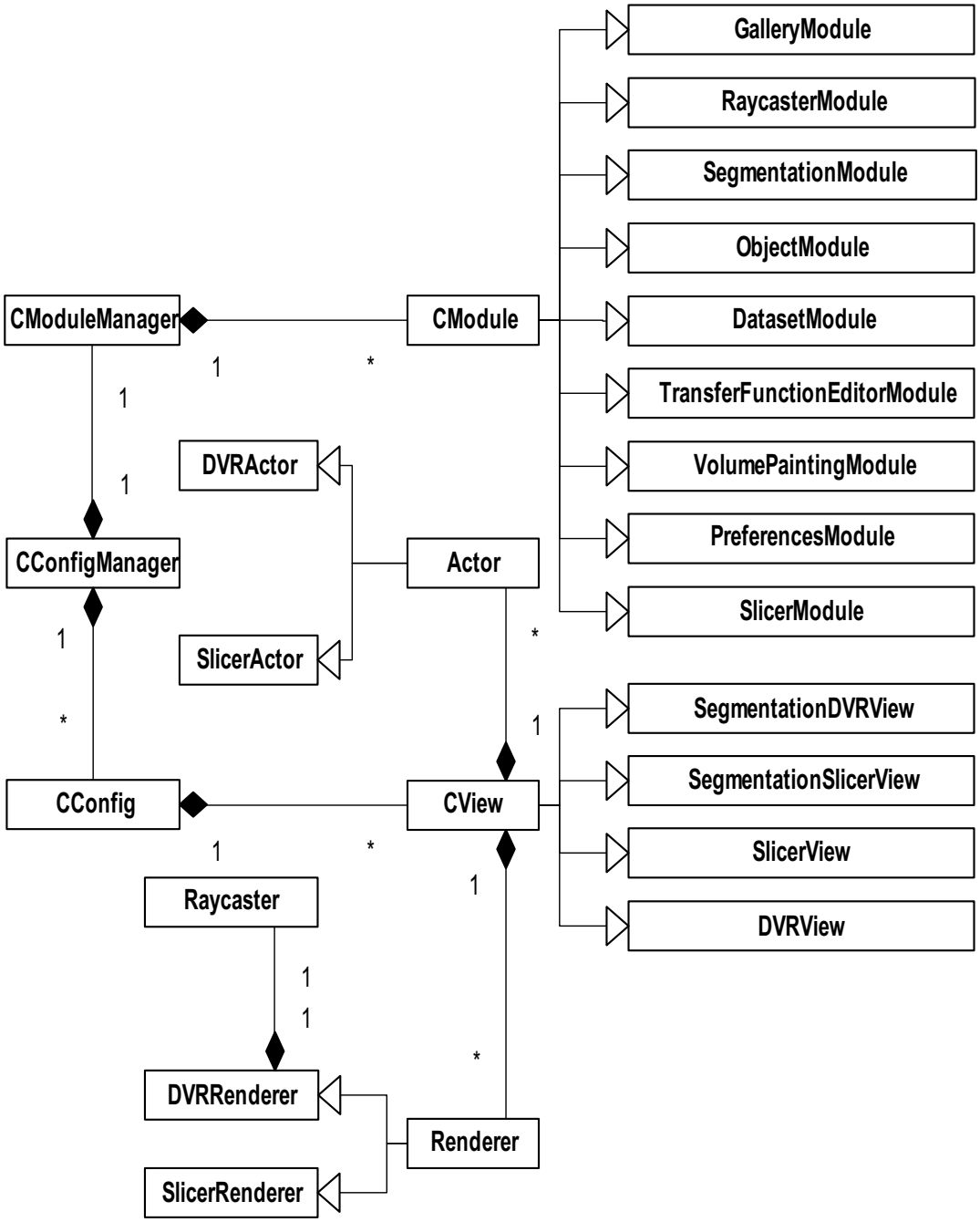


Figure 4.2: Class-diagram of Views, Modules, Actors, Renderers and Configurations

applied if a user switches in a configuration between two views. If we have loaded multiple SlicerViews for instance we can activate one View by clicking on it. We can for instance define which main-direction the slicer should view or which position. If the user then switches to another SlicerView the signals for manipulating the SlicerView through the SlicerModule are disconnected and connected to the currently activated SlicerView. If the user then switches back again the old settings in the module will be restored.

This extensive use of signals and slots mechanism makes it necessary to design an appropriate class to manage all this signals and slots. For this issue we have designed the SignalManager. So if a module sends a signal it must first connect this signal to this signal handler, which dispatches the signal to all slots of the views which are currently connected to this signal at the SignalManager.

Another problem is the management of datasets. For this task we have designed a module where the user can load and unload volumes and also add and remove objects for segmentation. The VolumeManager was programmed to load and manage these datasets at the different renderers. The renderers which are currently active are registered at the VolumeManager. The active renderers are those which interact with the currently active views. If the user switches between the views, the renderers will be registered and unregistered. If there was a dataset loaded or unloaded since the last time the renderer was active, the dataset is loaded or unloaded at the renderer. The information (the signals) from the dataset module are directly processed in the VolumeManager.

In the following section we will explain the tools for gallery manipulation, which are mainly standalone applications except the gallery module which is part of the framework and will also be presented next.

4.2 The Gallery Manipulation Utilities

We will describe the gallerymodule interface first to provide a basic knowledge how we use the galleries. It is part of the framework which was described in the previous section and helps the user in the transfer function designing

Figure 4.3: Sidebar

process. The interface is shown in Figure 4.4 and has two different parts.

The upper part consists of one or multiple folders which correspond to the datasets which are currently loaded. In this folder there are three radiobuttons provided to select different viewing positions (X-Direction, Y-Direction and Z-Direction). Also on the folder is a button to load a gallery-configuration-file, where the transfer functions are defined. For each transfer function a button with an image as content is generated in the view below the Load Gallery-button. This image shows the result if the dataset is rendered with the transfer function which corresponds to the button. We can navigate through this collection of preview images by using the scrollbar on the right-hand side of this view. We can click a button to select it and the image is rendered in the lower part of the module on the left side. Along with the image a checkbox is provided to choose which functions should be used for the final transfer functions. If we click on the image a dialog will be opened and we can manipulate the transfer function (colors and opacities of the controlpoints) which correspond to the image there. We can first select some candidate items by clicking in the upper part and then manipulate this items by clicking on the images in the lower part. The final result we will see just on the right side, because there is an imagebox which provides a preview of the selected images so far. In a last step we can set the current settings to the datasets in the framework by clicking on the Attach Settings-Button.

Along with the last selection step a problem comes up: We can already render different datasets with different transfer functions in one image by the multivolume rendering extension of the raycaster [11]. But to provide the full power of this way to define transfer functions, we have to design a system to render also different transfer functions for the same dataset in one image. This means we have to merge the transfer functions, which is achieved by merging the controlpoints and calculating the color and opacity at each point with the following equations:

$$\alpha = max(\alpha_1, \alpha_2) \tag{4.1}$$

Figure 4.4: GalleryModule

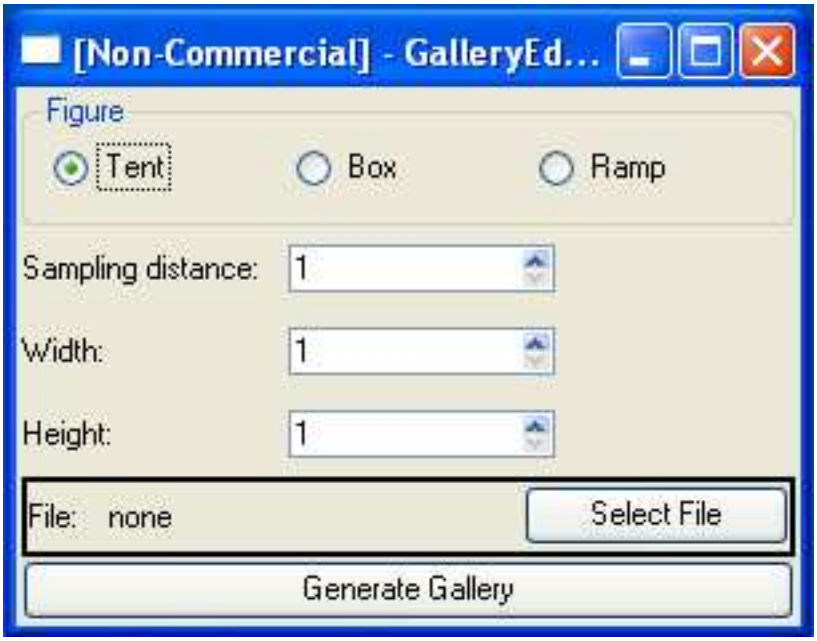


Figure 4.5: GalleryEditor

$$c = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2} \tag{4.2}$$

We get an environment which makes it easy to design transfer functions, but the next task will be to create gallery-configuration-files, so that we can use the GalleryModule efficiently. The theoretical background of these methods was already presented in Chapter 3, so we will give a quick overview here of the tools which provide a frontend for these methods.

The first program is the basic GalleryEditor (Figure 4.5). It provides the functionality for creating gallery-configurations with regular sampled patterns (as it was described in Section 3.2.1). We have three radiobuttons to select the pattern and can change the width and the height of the pattern. We can also manipulate the sampling distance between the patterns, which influences the gallery size.

The next program is the gallery generator for the data oriented methods (Section 3.2.2): like gradient magnitude thresholding and the thresholding method for the magnitude of the curvature (Figure 4.6). Here we can again

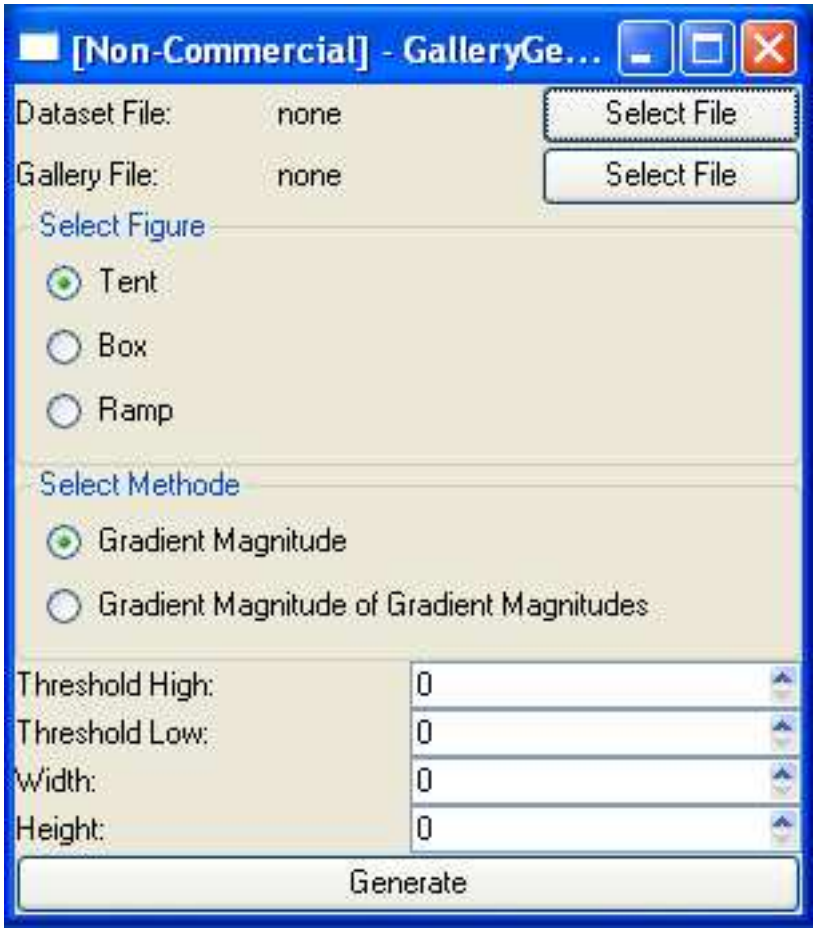


Figure 4.6: GalleryGenerator Data

select the patterns, and their width and length. We have two more input fields for the specification of the high and low threshold value. And last but not least we can select the method through radiobuttons. We have to select a file to load the dataset from, because for these methods also a specific dataset is needed. As before we have to select the configuration-file where we want the result to save.

In the next figure (Figure 4.7) we can see the interface for the histogram based methods. We can select the gallery configuration-file, the dataset the methods and the patterns again. Also the input fields for the width and the height of the pattern and for the threshold value are provided.

The second but last group of methods we have to provide an interface for is the one for the image based algorithms (Figure 4.8). These methods are based on a gallery configuration which was created before we can select such a configuration-file (Load Gallery File). We again provide widgets for the selection of the method, the dataset, and the threshold which again depends on the selected method. We can also specify the number of elements in the final gallery - this option is related to the advanced image selection method.

The last graphical frontend provides an interface for the genetic algorithm method (Figure 4.9). We can specify the dataset we want to work on. Then we can create a new population with the regular sampling method or load it from an existing configuration-file. The next option we have is to specify the factor of the best items of the last population which should be forwarded to the next as percentage of the population size. Then we can choose different methods of fitness calculations , recombination and mutation along with the factors of mutation and recombination. And as termination condition for the genetic algorithm we have to specify the number of generations. Finally we can select the file where we want to save the new configuration to.

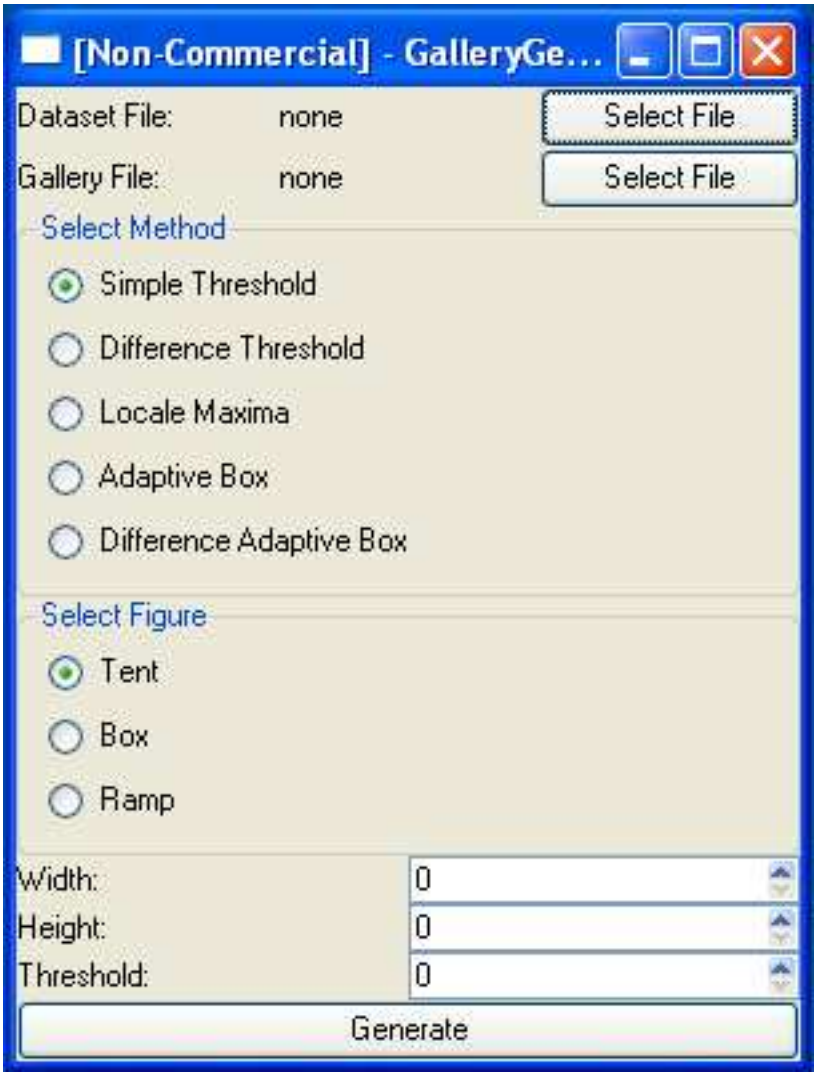


Figure 4.7: GalleryGenerator Histogram

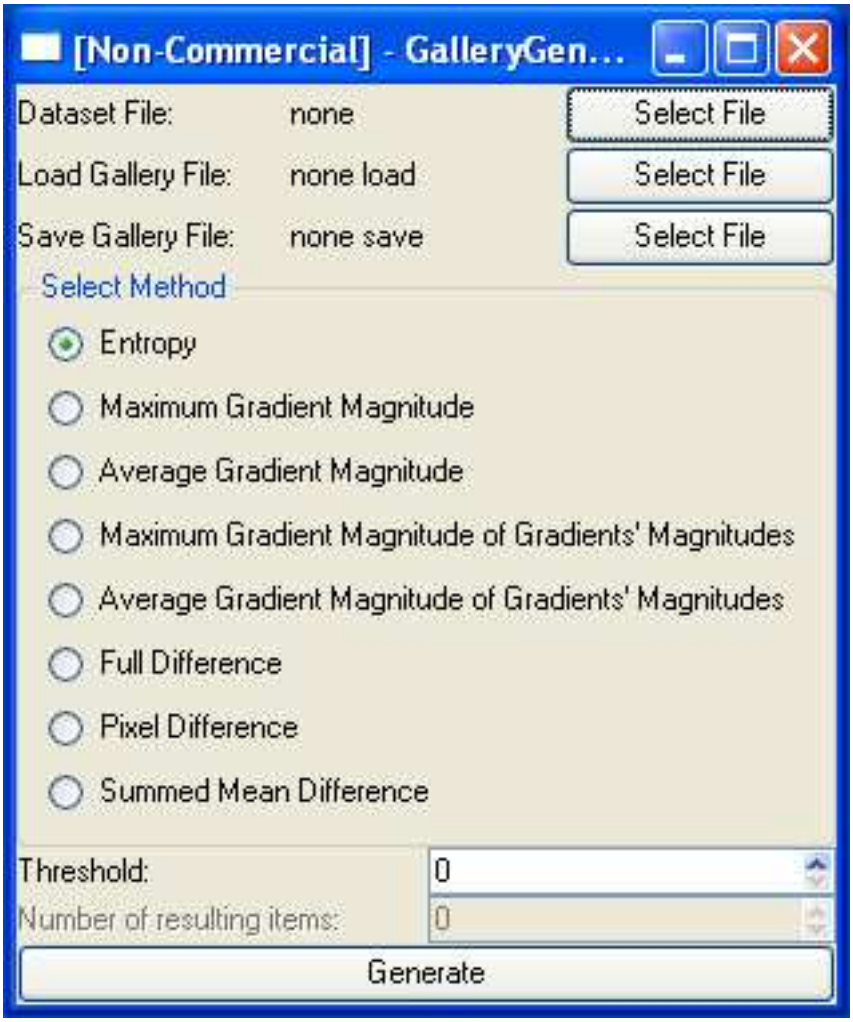


Figure 4.8: GalleryGenerator Image

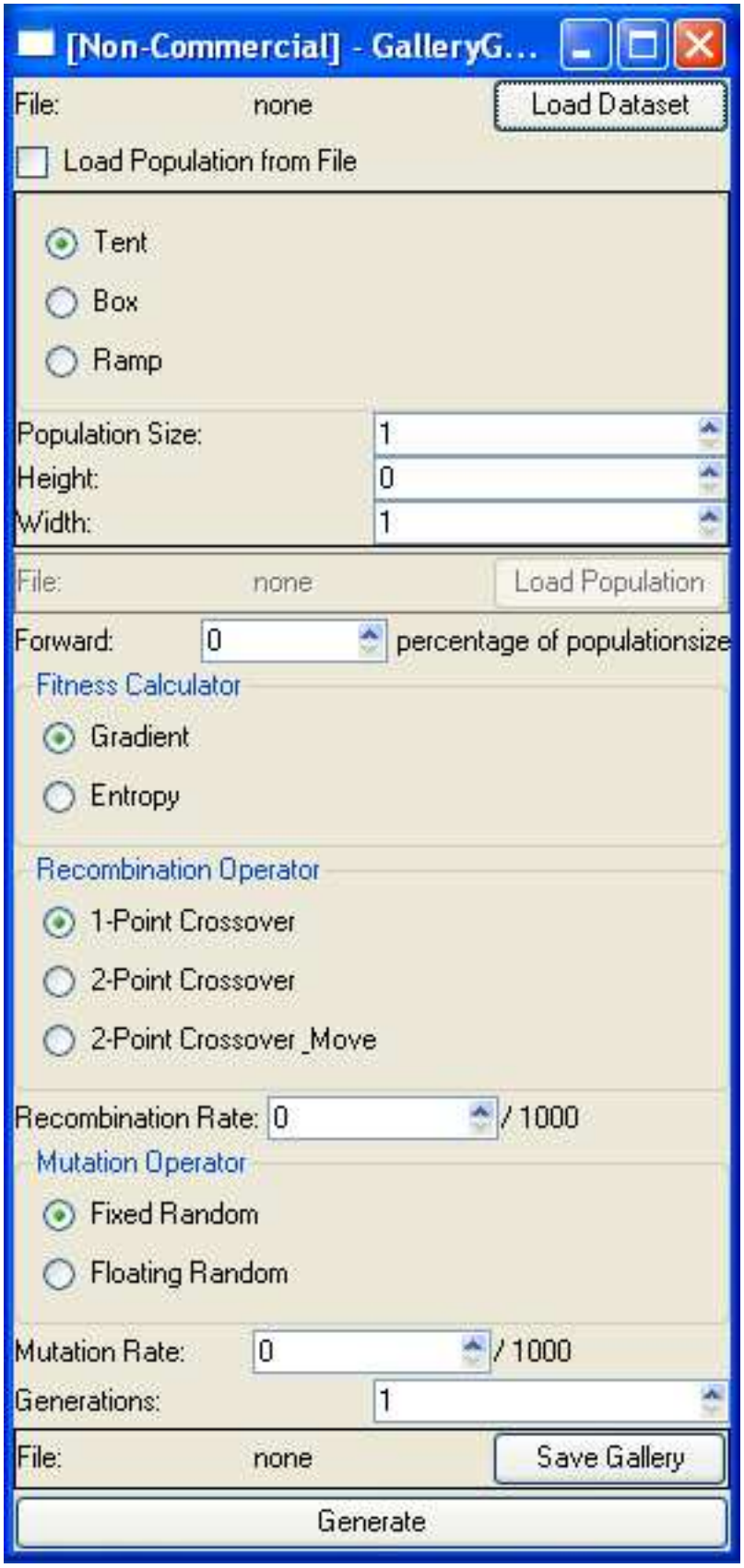


Figure 4.9: GalleryGenerator Genetic Algorithm

Chapter 5

Results

Truth hurts. Maybe not as much as jumping on a bicycle with a seat missing, but it hurts.

Drebin, Naked Gun 2 1/2

The results of the methods were already presented in Chapter 3, because they are used there to explain improvements and problems with certain methods.

The final conclusion to all these results would be that there is no single method which leads to a good gallery, with informative and different transfer functions.

One attempt to generate a really good representative gallery may be to combine some methods. First we can use the brute force with a dense sampling interval - this does not cost much time, as there is no additional data dependant processing needed. Then we can run an image based algorithm (like entropy estimation or gradient magnitude estimation) to sort out black images or other images with less information in it. Finally we can use one of the image difference calculation methods to further reduce the gallery size.

We will use this strategy now in an example, where we want to visualize a stag beetle. In this case we do not know exactly the density values which

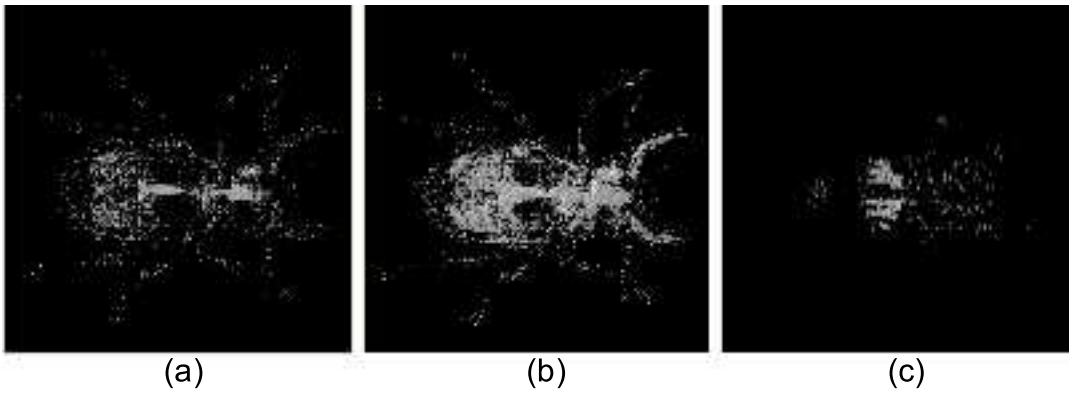


Figure 5.1: Transfer Functions of the generated Gallery used for the final Transfer Function: (a) 7th (b) 13th (c) last Transfer Function

show interesting features so the use of a design gallery wold be a good idea. In general in medical visualization of CT-datasets the interesting Hounsfield values are well known (see Table 1.1).

We first generate a gallery with a dens sampling interval. In our case we created the first gallery with the transfer function generation through organized sampling method (Section 3.2.1). We used a sampling intervall of 4 and generated the transfer functions with the help of the tent pattern with a width of 200 and a height of 200. The resulting gallery has a size of 1098 transfer functions. Then we reduced the gallery by the pixel-difference method (Section 3.2.4) with a difference threshold of 2000. This reduced the gallery to 51 transfer functions. After that we used the GalleryModule of our framework to assemble the final transfer function. For this task we choose the seventh, thirteenth and last transfer function in the gallery (see Figure 5.1 for images of these functions). We have to adapt the color and the opacity of these transfer functions and finally generated the image in Figure 5.2. The lower part of this figure shows the final transfer function.

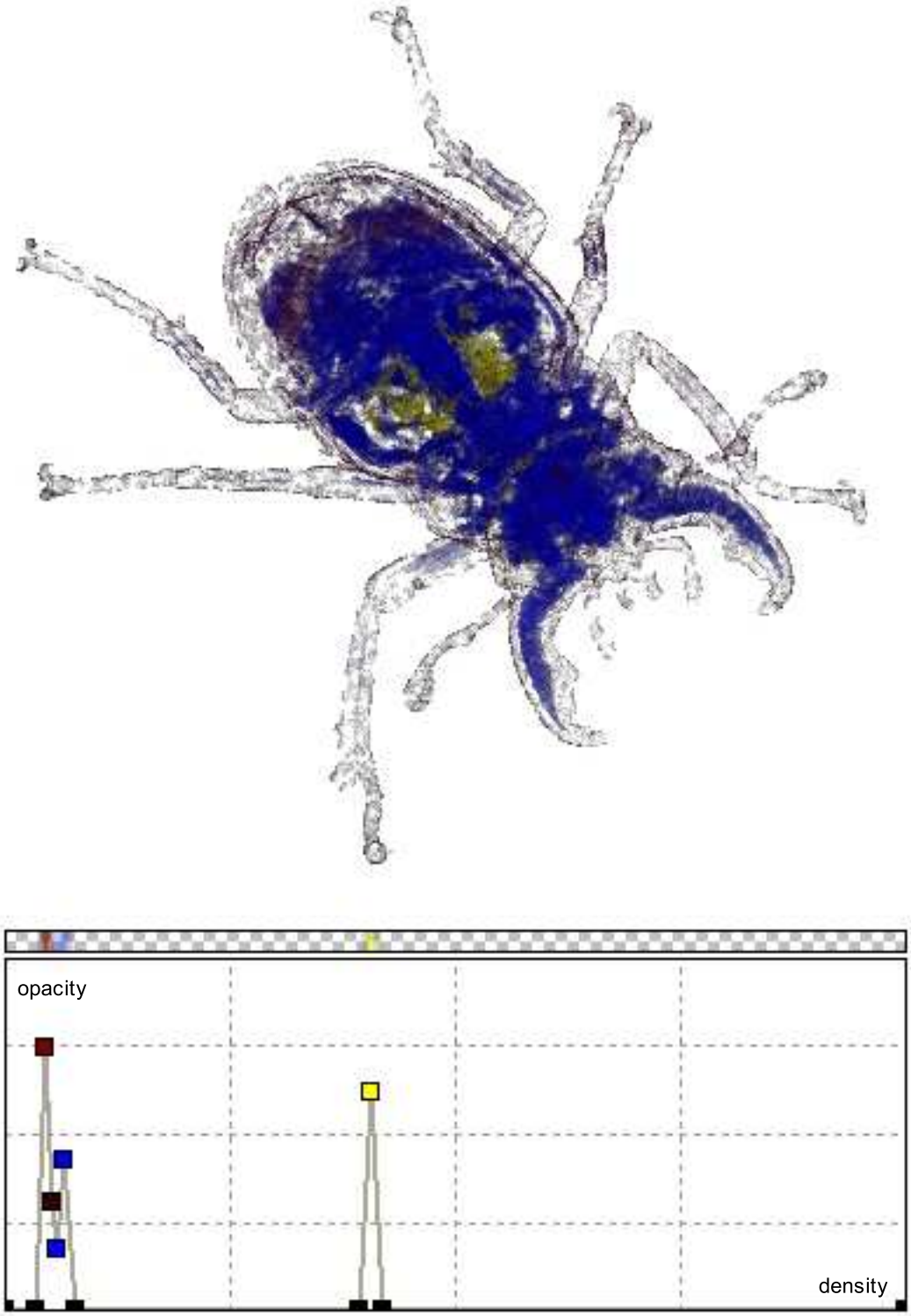


Figure 5.2: Stag beetle Visualization and final Transfer Function

Chapter 6

Summary

I may not have gone where I intended to go, but I think I have ended up where I needed to be.

Douglas Adams

6.1 Introduction

In volume visualization the definition of transfer functions is a critical task for the generation of informative images. Even more as we have a growing complexity in the images, because it is possible nowadays to interact with multiple volumes in one image. This is on one side a benefit for creating interesting visualizations, but on the other hand this will increase the problems we have already in designing proper transfer functions in mono-volume visualization.

The problem of defining a transfer function consists of two main subproblems:

1. The parameter-space of all possible transfer functions is large, so that finding the transfer function which visualizes the information we need, is hard to find in a Trial-and-Error approach.

2. There exists no bijective relation between data properties and the objects in the data we want to view.

There are several solutions to handle this problem and design transfer functions:

1. The Trial-and-Error approach: It is an iterative process between changing the transfer function and looking which effect this change produces in the image. As everybody can imagine this is a rather time-consuming procedure.

2. The data-centric approach without an underlying mathematical model: This solution is based on the analysis of a dataset (e.g., topology analysis) and creation of a model of the data on which the definition of the transfer function is based. Examples of such an approach were presented before [9, 1, 35].

3. The data-centric approach with an underlying mathematical model: In the work of Kindlmann and Durkin [16] such a solution was based on the mathematical model of boundaries between materials. It is less time consuming but a lot of parameters have to be set to use this solution efficiently.

4. The image-centric approach usually uses multiple preview images for different settings to help the user in the creation of transfer functions. This solution is rather simple, but needs a lot of computational power, to render the preview images first. Examples of this kind of solution were already presented [21, 18].

This work presents a solution which belongs to the group of image centric approaches and focuses mainly on the generation of informative galleries for different datasets. We will give a short description of the gallery framework we have programmed in the next section. Then the algorithms, which were designed to create efficient galleries, will be presented in the section “Gallery Generation Algorithms”. In the last section we will present a short conclusion and the results.

6.2 The Gallery Framework

The definition of a transfer function with the help of the gallery framework is done in three steps:

1. Selecting previously rendered items which represent different opacity functions. These items can be from different galleries and datasets.

2. Define the colors to the opacity items.

3. Do a final selection between the previously selected and colored items to get the final result.

With this three step strategy we can handle complex settings, like multiple transfer functions and functions for viewing different objects in one volume.

The crucial task in this process is how to generate galleries with representative items in it. This issue will be discussed next.

6.3 Gallery Generation Algorithms

The first method for creating galleries is defining the collections of transfer functions in such a way that in every function a pattern like a box or tent or ramp is placed at a specific point along the data axis. The transfer functions in the sequence are generated by shifting this pattern along the data axis with a constant offset.

We can see that these patterns have different properties [18]. The ramp can be used to get images with good representation of the surfaces in the datasets, the box and tent pattern on the other hand show more of the interior, whereas the box pattern produces because of its steep ascends and descends reconstruction artifacts (pixelization at hard boundaries between materials). So we should avoid using this pattern.

We can improve the method of transfer function generation by regular sampling by further processing the data or the final images.

The image based methods work rather simple. Here the predefined transfer functions are used in context of a specific dataset to render the images

of the gallery. Then based on these images different image measures can be applied. For instance we can calculate the entropy of an image, which is a measure for the information content, and sort out all images which have a too low value. The same can be done for the magnitude of image gradients and curvature information. The difference is that in the methods based on curvature the edges in the image are more emphasised. We have built methods for computing the average and the maximum magnitude of the curvature information and the gradient respectively. In general the maximum methods prefer images with finer detail than the average methods, which results in different selected images. The other variant of image based methods compares images to each other with an image difference calculation. So we can eliminate images which are nearly equal and so reduce the gallery size.

The data based methods can be divided into two groups:

1. Algorithms based on data
2. Algorithms based on the histogram of the data

In the first group there we have two algorithms, which are working nearly equally. One algorithm is based on data gradients and the other on the curvature. In both algorithms we can apply a high and a low threshold value. If the measure lies between these delimiters, we are saving the data value of the voxel which corresponds to the gradient or curvature. After processing the whole dataset we can place at the saved positions the pattern of our choice. During testing this method it became quite clear that this methods produce a lot of transfer functions even with small threshold ranges and therefore the galleries will not be easy to handle, because of their size. Perhaps a solution with a combination of multiple data properties will be better here.

The second group are the algorithms based on the data histogram. The simplest approach here is a thresholding based on the histogram value, but this will not lead to good results. In commonly used dataset there are a lot of voxels with low density values, which are not of interest, so these will not create a small and representative gallery of the data. Another algorithm which

works on histogram differences can be created if we again take a threshold value and check if the difference of two neighbouring histogram values lies above this value. In this case we create a transfer function with one of the patterns at this position for our gallery. This method produces large galleries, because a lot of items in these galleries lie in the unwanted low data value range, which represents the air in the datasets. So these galleries will still be very big and if we are changing to a bigger threshold value we are usually losing the interesting items before the uninteresting ones are removed. In the unwanted data range are there are a great number of items, which also leads to big differences between the histogram values there. So they will be removed only at the end. One more algorithm we created was based on the extraction of data values where the histogram shows a local maximum, but the histogram values are rapidly changing and so we get a lot of local maxima, which are producing too big galleries.

These methods produce rather simple transfer functions. The last attempt was to use a genetic algorithm to produce more complex transfer functions. A genetic algorithm usually has the following steps (see Algorithm 4).

The population consists of chromosomes, which are in our case the transfer functions. The genes of the chromosomes are the controlpoints of the transfer function. The phenotype are the rendered images which are based on the transfer functions (chromosomes). For the evaluation of a population we have to evaluate the images, this calculation is based on a fitness function. In our implementation we have realized two different fitness functions:

- Fitness function based on the average magnitude of the image gradient
- Fitness function based on the entropy of the image

The selection process of the chromosomes for recombination is a fitness proportional selection method. This means that the possibility of selecting chromosomes is proportional to the fitness, but has still some random factor in it.

For the recombination we have designed two crossover operators:

- Operator 1 divides the two selected transfer function in two parts at a special point and exchanges the second part of both functions with each other
- Operator 2 chooses two points and exchanges in both functions the part which lies between these two points

In the testing we have found out that the second operator leads to a better convergence behaviour than the first operator.

For the mutation we have also created two methods:

- Method 1 inserts a random controlpoint in the transfer functions which were selected for mutation
- Method 2 randomly chooses new opacity values for all existing controlpoints in the transfer function

We have seen in the tests that the second method has a worse convergency behaviour. This is a result of the bigger changes in the transfer function, which leads to a more random behaviour of the search process defined by the genetic algorithm.

The runtime of such genetic algorithms is rather long. They are designed to find an optimal solution corresponding to maximizing the fitness function in our case, this will not lead to images in the galleries which represent all different object in a dataset. So this will in general not provide a good gallery if we are comparing the expense of the runtime to the other methods.

6.4 Conclusion

We can see from the description before that there is no single method which produces representative galleries in general of every dataset, but if we combine two or more methods we can produce an informative selections of transfer functions. A good initial attempt for a gallery of a dataset will be first creating a gallery with the method of transfer function generation through regular sampling with a small sampling interval, which will lead to a big

gallery. Then we can apply some of the image based algorithms to reduce the gallery size. It is also a good idea to vary the patterns and pattern sizes.

Chapter 7

Acknowledgements

I would maintain that thanks
are the highest form of
thought, and that gratitude is
happiness doubled by wonder.

G. K. Chesterton

I want to thank Stefan Bruckner first for his suggestions and feedback when designing the gallery methods. I also want to thank Sören Grimm for his patience and great supervision in the programming and designing process of the raycasting framework, also want a big "thanks"to both of them for programming such a marvelous raycaster which made this work possible. I further want to thank Eduard Gröller who made it possible that I could do this interesting work at the Institute of Computer Graphics and Algorithms of the Vienna University of Technology.

Bibliography

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

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

[3] J. F. Blinn. Simulation of wrinkled surfaces. *Computer Graphics*, 12(3):286–292, 1978.

[4] E. Catmull. Computer display of curved surfaces. *PhD Thesis*, 1974.

[5] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in reeb graphs of 2-manifolds. *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 344–350, 2003.

[6] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-quality volume rendering using texture mapping hardware. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 69–ff., 1998.

[7] W. E.Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–168, 1987.

[8] T. Elvins. A survey of algorithms for volume visualization. *SIGGRAPH Comput. Graph.*, 26(3):194–201, 1992.

[9] I. Fujishiro, T. Azuma, and Y. Takeshima. Automating transfer function design for comprehensible volume rendering based on 3d field topology analysis. *Proceedings of IEEE Visualization*, pages 319–326, 1998.

[10] I. Fujishiro, Y. Takeshima, T. Azuma, and S. Takahashi. Volume data mining using 3d field topology analysis. *IEEE Comput. Graph. Appl.*, 20(5):46–51, 2000.

[11] S. Grimm, S. Bruckner, A. Kanitsar, and E. Gröller. Flexible direct multi-volume rendering in interactive scenes. *Vision, Modeling, and Visualization(VMV)*, pages 379–386, October 2004.

[12] T. He, L. Hong, and H.-P. Pfister. Generation of transfer functions with stochastic search techniques. *Proceedings of IEEE Visualization*, pages 227–234, 1996.

[13] 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.

[14] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

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

[16] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. *Proceedings Volume Visualization Symp. 1998*, pages 79–86, 1998.

[17] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* MIT Press, Cambridge, MA, USA, 1992.

[18] A. H. König and E. M. Gröller. Mastering transfer function specification by using volume pro technology. *Spring Conference on Computer Graphics 2001*, 17:279–286, 2001.

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

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

[21] 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. *Proceedings of ACM SIGGRAPH*, pages 389–400, 1997.

[22] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[23] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. *Proceedings of the IEEE Symposium on Volume Visualization*, pages 81–90, 2000.

[24] J. O'Connor and E. Robertson. Harold Calvin Marston Morse. http://www-history.mcs.st-andrews.ac.uk/history/Mathematicians/Morse.html. [Online; accessed 8-12-2005].

[25] H.-P. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volume pro real-time ray-casting system. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 251–260, 1999.

[26] H.-P. Pfister, B. Lorensen, C. Bajaj, G. Kindlmann, W. Schroeder, L. Sobierajski Avila, K. Martin, R. Machiraju, and J. Lee. The transfer function bake-off. *IEEE Comput. Graph. Appl.*, 21(3):16–22, 2001.

[27] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-textures and multi-stage rasterization. *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, 2000.

[28] L. Schmitt. Theory of genetic algorithms. *Theor. Comput. Sci.*, 259:1–61, 2001.

[29] L. Schmitt. Theory of genetic algorithms ii: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling. *Theor. Comput. Sci.*, 310(1-3):181–231, 2004.

[30] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, pages 623–656, 1948.

[31] J. Shapiro. Statistical mechanics theory of genetic algorithms. *Theoretical aspects of evolutionary computing*, pages 87–108, 2001.

[32] Y. Shinagawa and T. Kunii. Constructing a reeb graph automatically from cross sections. *IEEE Comput. Graph. Appl.*, 11(6):44–51, 1991.

[33] P. Shirley and A. Tuchman. A polygonal approximation to direct scalar volume rendering. *Computer Graphics*, 24(5):63–70, 1990.

[34] S. Süsstrunk, R. Buckley, and S. Sven. Standard rgb color space. *Proceedings IS&T/SID's 7th Color Imaging Conference*, pages 127–134, 1999.

[35] G. H. Weber and G. Scheuermann. Topology-based transfer function design. *Proceedings of the Second IASTED International (VIIP 2002) Conference on Visualization, Imaging, and Image Processing 2002*, pages 527–532, 2002.

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

[37] Kitware Public WIKI. ITK hounsfield units. http://www.itk.org /Wiki/ITK_Hounsfield_Units. [Online; accessed 8-12-2005].

[38] Wikipedia. Color theory. http://en.wikipedia.org/wiki/Color_theory. [Online; accessed 8-12-2005].

[39] Wikipedia. Hounsfield scale. http://en.wikipedia.org/wiki/Hounsfield_scale. [Online; accessed 8-12-2005].

[40] Wikipedia. Luminance. http://en.wikipedia.org/wiki/Luminance. [Online; accessed 8-12-2005].

[41] Wikipedia. Morse theory. http://en.wikipedia.org/wiki/Morse_theory. [Online; accessed 8-12-2005].

List of Figures

List of Tables

List of Algorithms