Eurographics Workshop on Visual Computing for Biology and Medicine (2015)
K. Bühler, L. Linsen, and N. W. John (Editors)

Interactive Position-dependent Customization of Transfer Function Classification Parameters in Volume Rendering

T. Brix[1], A. Scherzinger[1], A. Völker[1], and K. Hinrichs[1]

[1]University of Münster, Germany

Abstract

In direct volume rendering (DVR) and related techniques a basic operation is the classification of data values by mapping (mostly scalar) intensities to color values using a transfer function. However, in some cases this kind of mapping might not suffice to achieve satisfying rendering results, for instance when intensity homogeneities occur in the volume data due to technical restrictions of the scanner technology. In this case it might be desirable to customize one or more parameters of the visualization depending on the position within the volume.

In this paper we propose a novel approach for an interactive position-dependent customization of arbitrary parameters of the transfer function classification. Our method can easily be integrated into existing volume rendering pipelines by incorporating an additional operation during the classification step. It allows the user to interactively modify the rendering result by specifying reference points within the data set and customizing their associated visualization parameters while receiving direct visual feedback.

Since the additional memory requirements of our method do not depend on the size of the visualized data our approach is applicable to large data sets, for instance in the field of ultra microscopy.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

In recent years the visualization and visual analysis of volumetric data sets (or simply *volumes*) have become standard operations in biomedical research. Technologies such as fluorescence microscopes used by cell biologists or computed tomography (CT) scanners used by physicians allow to acquire volumetric data sets or image stacks from which volumes can ultimately be reconstructed. The application of both two- and three-dimensional rendering techniques can then provide insightful visualizations for domain experts.

The volume rendering process usually includes a classification step which consists of mapping features (often corresponding to scalar intensity values) at given sampling positions within the data set to optical properties, i.e., color values, by applying a *transfer function*. Various transfer functions to realize this mapping have been proposed, ranging from a simple linear transformation from intensity values to grayscale values up to high-dimensional mappings using multiple intensity channels as well as additional input parameters, e.g., the local image gradient.

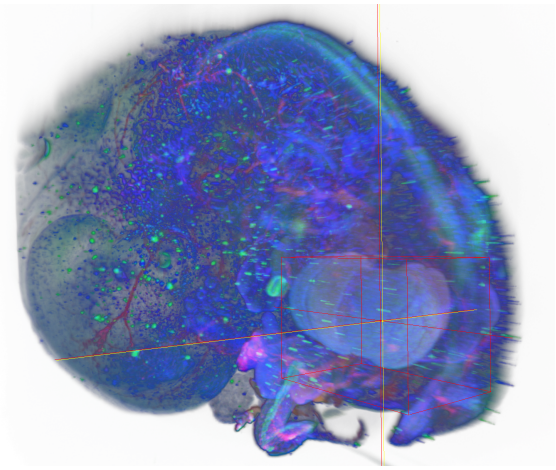


Figure 1: Our proposed customization approach can be integrated into rendering pipelines for large volume data sets. This image shows the maximum intensity projection of a 9.5 GB 3-channel mouse embryo data set where an intensity gamma correction has been applied to the marked region.

DOI: 10.2312/vcbm.20151211

Usually classifications assume constant characteristics of features in the data. An example is the linear mapping of intensity values acquired by a CT scanner to Hounsfield units (HU) which allows physicians to classify anatomical features based on their range of values [Hou80]. Due to this assumption samples in the volume data which correspond to the same feature characteristics are mapped to the same optical properties, similar to a mathematical function mapping a specific set of input parameters always to the same output. However, in some situations this behavior might not be desirable. Instead, incorporating the position of a given sample within the volume as an additional parameter offers a more flexible visualization by allowing *local* adaptations of the transfer function. This might be due to a combination of the scanner technology and the physiological properties of the tissue leading to complex effects such as absorption which can not be easily corrected, or the necessity to modify the classification for highlighting regions of the volumetric data set or choosing different transfer functions for several regions of interest.

We propose a technique which offers domain experts the ability to perform a local on-the-fly customization of arbitrary visualization parameters to achieve insightful visualizations and which can easily be integrated into existing rendering systems. The development of our approach was driven by the following design goals:

1. Interactive position-dependent customization of any parameter related to the transfer function classification
2. Direct visual feedback during user interaction
3. Integration into existing volume rendering techniques
4. Compatibility with existing rendering techniques for large data sets exceeding available GPU memory

2. Related Work

With the emergence of programmable graphics hardware, sophisticated volume rendering techniques such as GPU-based ray-casting [KW03] became applicable to interactive visualizations. In recent years, the rapid development of new and enhanced scanner technologies as well as more complex scientific simulations, e.g., in the field of materials science and geophysics, have led to an increasing size of volume data sets, requiring efficient out-of-core and multi-resolution approaches like the ones proposed by Gobbetti et al. [GMG08] or Crassin et al. [CNLE09] to maintain interactive frame rates. Both of their implementations utilize an octree data structure to organize a dynamic block-wise transfer of required parts of the volume to the GPU memory.

Despite the methodological diversity of available volume rendering techniques, all of the approaches incorporate a classification step relying on a transfer function to map the features of the input data into a color space. A simple way to directly classify intensity values during rendering is given by the use of *opacity functions* [Kin02], which assign an opacity value to each intensity and thus allow to highlight important features of the volume data corresponding to specific ranges of intensity values. By extending the concept from opacity values to additional parameters such as colors, these types of transfer functions offer additional flexibility for creating volume visualizations [LCN98].

Besides one-dimensional transfer functions, which are limited to the scalar intensity values of the volume data, two-dimensional functions, which have been introduced by Levoy [Lev88] in 1988, are commonly used. Such functions use the image gradient at a given sample position as a second input dimension in addition to the intensity value. The specification of simple shapes in the two-dimensional space defined by the gradient magnitude and the scalar intensity allows to visualize the surfaces of structures within the volume data which manifest themselves in the shape of arcs within the two-dimensional histogram [KKH02]. Roettger et al. [RBS05] have introduced *spatialized transfer functions*, which assign colors to connected components within the data set using two-dimensional transfer functions based on the local image gradient.

Besides the image gradient, other features of the data set can be used for specifying two-dimensional transfer functions, including spatial information. Correa and Ma [CM09] compute an occlusion spectrum for each voxel and use it as the second dimension of the classification. Selver and Günzeliş have proposed a method for semi-automatic generation of transfer functions which take into account the individual slices of a medical image stack [SG09]. Another approach is to determine the size of anatomical structures and incorporate it as the second dimension of the transfer function classification. Such methods have been proposed in [CM08] and [WKK10].

Zhou et al. [ZDT04] have proposed a distance-based transfer function, which takes into account the current viewpoint of the volume rendering. In their method, the position of a voxel is used to modify the rendering depending on the distance to the camera or a specific point in space. Tappenbeck et al. [TPD06] have extended this concept by considering distances to anatomical structures which have been determined by segmentation. A similar approach using weighted distance fields has been proposed by Kerwin et al. [KHS*10]. Another system proposed by Correa and Ma [CM11] computes a visibility histogram for the current view point and opacity function, which allows generating visibility-driven transfer functions. Their approach can be applied to one- and two-dimensional transfer functions.

There also exist higher-dimensional transfer functions which consider multiple intensity channels as well as additional input parameters such as curvature or local texture information, e.g., the one proposed by Kim et al. [KSC*10].

Bruckner and Gröller [BG05] have proposed *VolumeShop*, a software package which connects the transfer function classification to segmentation and volume registration approaches. This concept has been extended and refined in [BKW08] and [GY13].

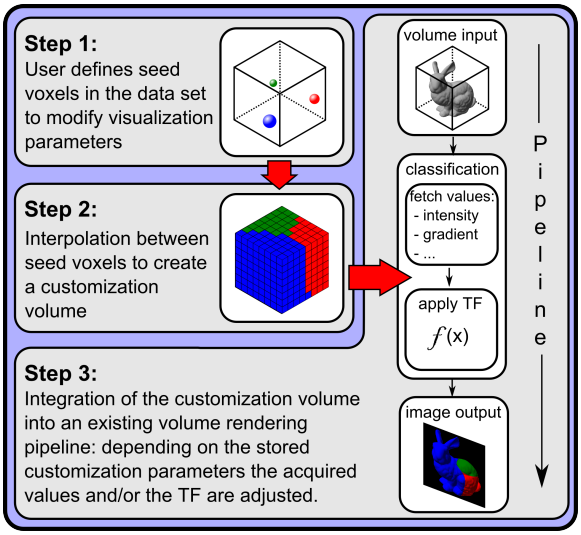


Figure 2: Schematic representation of the three steps required for the classification parameter customization.

Our approach can be combined with all of the above transfer function concepts, extending the corresponding classifications by allowing user-defined position-dependent customizations to arbitrary subsets of the used input parameters and thus addressing a position-dependent extension of the general transfer function concept without the need for preprocessing computations.

3. Overview

In this section we give an overview of the three basic steps required by our method for customizing classification parameters. These steps are also schematically shown in Fig. 2. A detailed description of specific parts of our approach will be given in the following sections.

Step 1: Define seed customization points
For large volumetric data sets with over 1000^3 *voxels* an individual user-driven specification of classification parameters for each individual voxel of the data set is not feasible. We have thus limited the customization to user-defined seed voxels, which we will be referring to as *seed customization points* in the following. A seed customization point p consists of a unique voxel position within the data set as well as a parameter vector $\vec{v}$ containing all customizations of the visualization parameters, i.e., reference values for this voxel position.
The type of the parameters can range from float values, e.g., for an intensity correction, to any complex structure, e.g., an entire color map. The way in which the user can create such seed customization points and define their parameter vectors will be outlined throughout Sec. 6.

Step 2: Calculate customization volume
Since the user modifies the visualization parameters for a few seed points only, it is necessary to interpolate these parameter customizations for all of the other voxels within the volume in a *reasonable* fashion. The term *reasonable* will be defined in detail later on (see Sec. 4).
To reduce the memory requirements and achieve interactive frame rates during the parameter customization, we map the seed customization points into a smaller volume, which will be denoted as the *customization volume* in the following. The resolution of the customization volume can be chosen between 32^3 and 512^3 voxels, depending on the available GPU memory and parameter complexity. The interpolation itself, i.e., the calculation of each interpolated visualization parameter for each voxel of the customization volume, is then performed on the GPU using OpenGL compute shaders (see Sec. 5).

Step 3: Integration into the volume rendering pipeline
The customization volume calculated in the previous step is integrated into the classification step of the volume rendering pipeline. At a specific point in the rendering process all of the required input parameters for the classification are gathered from the volume data set, e.g., the intensity value or image gradient at a sampling position, and passed to the transfer function. The customization step is integrated directly before the application of the transfer function. The parameter modifications for the current sampling position are fetched from the customization volume and can be applied to both the input parameters as well as the transfer function itself, depending on the type of each of the visualization parameters. This modification will be outlined in more detail throughout Sec. 5.

4. Interpolation Method

In this section the interpolation method for the customization values and the calculation of the customization volume will be described. Throughout this section we will assume a given set S of user-defined seed customization points p_i with associated visualization parameter values v_i and a requested point q with $q \notin S$ for which the parameter v_q should be interpolated based on the set S. This section also contains a discussion of alternative interpolation approaches, which we have considered impractical for our system, as they do not fulfill the requirements of a *reasonable* interpolation in our context, which will be defined in the following subsection.

4.1. Requirements of the Interpolation

As outlined above, the customization values are defined by the user for only a small number of seed customization points, making it necessary to derive values for the remaining sampling positions in the volume data set from the seed customization points. Those values are determined by interpolating between the reference values corresponding to the

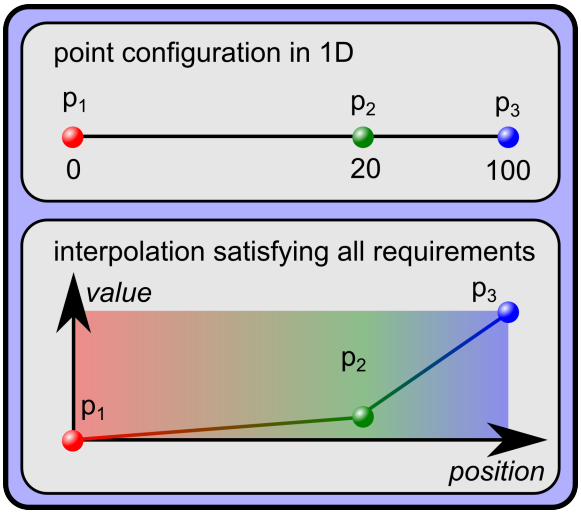


Figure 3: Possible interpolation between seed customization points in 1D satisfying all requirements of Sec. 4.1.

user specified seed customization points. For this purpose three basic requirements for the interpolation were defined:

1. The interpolation should not change the value at the user-defined points.
2. The interpolation between two neighboring seed customization points should be continuous and monotonic.
3. Each seed customization point should only have local influence.

Since the seed customization points are set by the user to customize the visualization in a specific way at the chosen voxel position, the first requirement ensures that the voxel corresponding to the seed customization point itself keeps the specified parameter value independent from changes in the set of the other seed customization points.
The second requirement excludes choosing the value of the nearest seed customization point for each voxel, but allows linear interpolation between two seed customization points. All of the values interpolated between a set of seed customization points should be in the range defined by the minimum and maximum values associated with the seed customization points within the set.
The third requirement means that if a seed customization point is *occluded* by one or more other seed points for a voxel position, it should not influence the value computed for this voxel. We conveniently omit a formal definition of the term *occluded* here and instead settle for the intuitive concept of the one-dimensional example outlined below.
An example of an interpolation for an arbitrary numerical value using seed customization points in a 1D space is shown in Fig. 3. The interpolation shown in the bottom half of the image satisfies all of the above requirements. The first requirement is met, since the values v_i are attained at their corresponding positions p_i. The second requirement is satisfied since the interpolation between each pair of neighboring seed customization points is monotonic and continuous and all of the interpolated values lie within the range of the seed customization points' values. In this example in 1D space the term *occluded* can be intuitively understood, as in the linear interpolation between neighboring points shown in Fig. 3, the rightmost point p_3 with the value $v_3 = 100$ does not influence any of the values left of p_2 with the value $v_2 = 20$. However, it should be noted that the term is more complex to define in 2D or 3D space. Fortunately, our interpolation method already entails a certain form of this property without the necessity of further care (see Sec. 4.3).

4.2. Discussion of Interpolation Approaches

Generally, an interpolation between a set of seed customization points S as declared above can be expressed as the weighted sum of their corresponding values in the form

$$v_q = \sum_{p_i \in S} \omega_i \cdot v_i, \tag{1}$$

where ω_i are the weights defined by the interpolation function, the positions of the seed customization points, and the position of the requested point q.
During the development process various alternative interpolation methods, i.e., definitions of the weights ω_i, have been considered. Below, we will shortly discuss some of the concepts which were not practically applicable in our context, before we outline the interpolation method we have chosen for our system in detail in Sec. 4.3.

Inverse distance weighting (IDW): This approach commonly applied in geostatistics derives the weights ω_i for equation 1 directly from the distance between q and p_i. Shepard [She68] was among the first introducing this approach. However, this method does not comply with the third requirement outlined in Sec. 4.1, as all of the specified seed customization points influence the interpolated voxel and thus have a global influence.
Kriging: The theoretical basis for this approach, which is also frequently applied in geostatistics, was proposed by Krige in his master thesis [Kri51]. The benefit of Kriging compared to IDW is the consideration of local variance which results in better interpolations. However, the method is computationally expensive and (without the additional specification of a local neighborhood) violates the third requirement similar to IDW.
Seed Customization Points as spheres: This approach is based on interpreting the seed customization points as spheres with a user-defined influence radius in which the influence of the seed point decreases with increasing distance to the center. However, difficulties arise when trying to define interpolated values in the intersection between two or even more spheres. Approaches like using the maximum or weighted averages violate at least one of the above requirements.

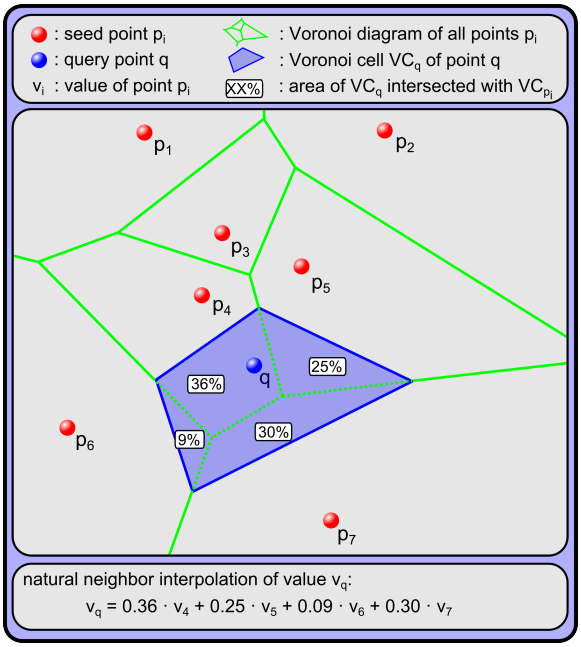


Figure 4: Example of the natural neighbor interpolation in 2D. Given seven seed points p_i with their associated values v_i the value v_q of a query point q is determined by using equation 2.

Tetrahedralization by Delaunay-triangulation: This approach is based on computing a decomposition of the bounding box of the volume data set into a set of tetrahedrons by applying Delaunay-triangulation [Del34] using the set $Q = S \cup E$, where E is the set of the corners of the volume and S is the set of seed customization points. The benefit of this approach is the given natural interpolation inside a tetrahedron by the barycentric coordinate system. However, there occur some degenerate cases where the triangulation results in undesired interpolation results. An example is a set of seed customization points lying on a sphere around the point q, where only four of those seed customization points affect the value of q instead of all of them, which is a common configuration in our approach (see Sec. 6).

4.3. Natural Neighbor Interpolation

To satisfy the requirements outlined in Sec. 4.1 we apply *natural neighbor interpolation*, which has been proposed by Sibson [Sib81] and is also known as Sibson or Voronoi interpolation.
The basic concept of this interpolation method is to calculate a Voronoi diagram containing all seed customization points. A two-dimensional example is shown in Fig. 4 for a set S consisting of seven seed customization points p_i with their associated numerical visualization parameter v_i. Let VC_{p_i} be the Voronoi cell of point p_i, for which its boundaries are shown in green in the image, and let $vol(VC_{p_i})$ be the volume (i.e., the area in 2D) of the Voronoi cell VC_{p_i}. To calculate the interpolated value v_q of a query point q, a new Voronoi diagram is computed for the set $Q = S \cup \{q\}$, i.e., the set of seed customization points and the query point q. The newly created Voronoi cell of q is highlighted in blue in Fig. 4. The interpolated value v_q can then be computed using the equation

$$v_q = \sum_{i=1}^{N} \frac{vol(VC_{p_i} \cap VC_q)}{vol(VC_q)} \cdot v_i, \tag{2}$$

which is a realization of equation 1 in which the value v_q is calculated as the sum of all neighboring v_i, each weighted by the volume of the intersection of VC_q with the original Voronoi cell VC_{p_i}.
It can be shown that this method satisfies the first and second requirement outlined in Sec. 4.1 and at the same time implies a definition of the third requirement and the term *occluded*, where only the seed customization points in a local neighborhood are taken into account for each point q, which are the neighbors of the newly created Voronoi cell of q. Natural neighbor interpolation thus satisfies all of the requirements we have set for our interpolation method.

4.4. Calculation of the customization volume

By applying the interpolation method outlined in the preceding section the required customization volume can be calculated as follows. A natural neighbor interpolation is performed for each voxel in the customization volume not corresponding to a seed customization point. However, a number of special cases have to be taken into account.
A Voronoi cell can have infinite volume, if its associated point is not strictly inside the convex hull of the given set of points. This applies to any requested interpolation point q outside the convex hull of all seed customization points, which makes equation 2 undefined. To overcome this issue we limit the Voronoi diagram to the bounding box of the volume data set so that all Voronoi cells have a finite volume.
To provide a well-suited default setup eight default seed customization points are provided, each of them corresponding to a corner of the volume's bounding box. This prevents the insertion of a single seed customization point for a local customization from having a global influence, which would be the case if computing the Voronoi diagram for the natural neighbor interpolation with a single inserted seed customization point. At the same time these default points allow to apply simple linear gradients over the volume by adjusting the values of the corner seed customization points without the necessity of adding additional seed customization points.

5. Implementation

In this section we describe our implementation and how the performance of the theoretical approach outlined above has been improved. Our method has been integrated

into the rendering pipeline of the visualization framework Voreen (http://voreen.uni-muenster.de), which has been developed at the University of Münster and is capable of rendering large volumetric data sets [MSRMH09, BPH14]. Throughout this section we will be assuming a given set of seed customization points consisting of their position in voxel coordinates and one numeric visualization parameter. This set should also contain the eight corner points mentioned in Sec. 4.4. The user interface for adding or removing seed customization points from the set or modifying the seed customization points will be outlined in Sec. 6.

5.1. Discretization of the customization volume

To apply the natural neighbor interpolation described in Sec. 4.3 it is necessary to compute the Voronoi diagram, i.e., the Voronoi cells of the seed customization points and each query point q, several times. Many accurate algorithms are based on computing the Delaunay-triangulation, which is the dual of the Voronoi diagram. This approach was implemented using the implementation of Pion and Teillaud [PT15] published in the open source library CGAL [cga]. Although this implementation achieves visually satisfying results, the computational cost of the geometric computations results in a very low framerate during user interaction, which prevents immediate visual feedback. Furthermore, calculating the customization volume in the resolution of the original volume data, thus providing interpolated information for each voxel in the data set, might result in problems with the memory management on the GPU during rendering, especially for large volume data sets.

To overcome those issues we use a discretization similar to the one proposed by Park et al. [PLK*06], which can be calculated on the GPU (see Sec. 5.2) and solves both of the above performance issues. We approximate the volume of a Voronoi cell by the number of voxels, which belong to it, by computing an approximate discrete Voronoi diagram on a regular grid. This allows to exploit the massively parallel computing capabilities of modern GPUs.

Furthermore, the required memory was reduced by calculating the Voronoi diagram and the resulting customization volume in a lower resolution. Good results can be achieved even with a resolution of 64^3 voxels. This is due to the fact that samples between the voxels of the low-resolution customization volume are reconstructed using trilinear interpolation when accessing the customization volume during the volume rendering process, which can be performed efficiently on the GPU. This procedure allows to choose the resolution of the customization volume independently from the input volume data size and provides interactive frame rates.

5.2. Calculation on the GPU

The calculation of the customization volume on the GPU is performed in a two-stage process. First, we calculate the Voronoi diagram, and then we compute the customization volume via natural neighbor interpolation. Both steps are performed using compute-shaders which are available in the OpenGL standard since version 4.3 and offer parallel general-purpose computing capabilities in OpenGL, which are similar to OpenCL or the CUDA framework of Nvidia.

A conceptual version of the shader code used to calculate the Voronoi diagram can be seen in Algorithm 1. For every voxel of the customization volume a work item is started, which receives its coordinates and a vector containing all seed customization points as its input. Each work item determines then the seed customization point that has the closest distance to the associated voxel position. The distance measure can be changed between the (squared) Euclidean distance or the Manhattan distance. Since the voxel position of all seed customization points has been specified in the original higher input volume resolution, the position has to be scaled relative to the customization volume resolution before the points are passed to the shader. The index of the nearest point in the input vector is stored in the output Voronoi volume. Thus, the Voronoi volume stores for each voxel the information, which seed customization point, referenced by the index in the input vector, is nearest to it, i.e., to which Voronoi cell the voxel belongs.

In a second shader pass the final customization volume is calculated. Again, we start a work item for each voxel of the customization volume to determine the interpolated value. The simplified shader code can be seen in Algorithm 2. We test for each voxel in the Voronoi volume if it is included

Algorithm 1: PseudoCode of the compute shader calculating the discrete Voronoi diagram.

Input : Buffer of all seed points (*points[]*), each containing their 3D voxel position (*pos*) and their visualization parameter (*value*).
Output: Discrete Voronoi diagram with 3D extent (*resolution*) as linear buffer (*voronoi[]*) storing the index of the point in the input buffer (*points[]*) to which Voronoi-cell this voxel belongs.

```
// get 3D position from current global work item index
vec3 cur3DPos = gl_GlobalInvocationID.xyz;
// initialize temporary variables
int minPoint = 0;
float minDist = ∞;
// determine the nearest point to cur3DPos
for i = 0; i < #points; i++ do
    float d = distance(points[i].pos, cur3DPos);
    if d < minDist then
        minDist = d;
        minPoint = i;
// calculate linear index from cur3DPos and set Voronoi data
int index =                                  cur3DPos.x
          +              resolution.x*cur3DPos.y
          + resolution.y*resolution.x*cur3DPos.z;
voronoi[index] = minPoint;
```

Algorithm 2: Pseudocode of the compute shader calculating the customization volume.

Input : Buffer of all seed points (*points[]*), each containing their 3D voxel position (*pos*) and their visualization parameter (*value*).
Discrete Voronoi diagram with 3D extent (*resolution*) as linear buffer (*voronoi[]*) storing the index of the point in the input buffer (*points[]*) to which Voronoi-cell this voxel belongs.
Output: Customization volume with 3D extent (*resolution*) as linear buffer (*interpolationResult[]*) storing the natural neighbor interpolated value at this voxel.

```
// get 3D position from current global work item index
vec3 cur3DPos = gl_GlobalInvocationID.xyz;
// initialize temporary variables
float result = 0.0;
float divider = 0.0;
// determine for each voxel of the discrete Voronoi diagram if
is included in the new Voronoi cell of cur3DPos.
for x = 0; x < resolution.x; x++ do
    for y = 0; y < resolution.y; y++ do
        for z = 0; z < resolution.z; z++ do
            vec3 curPos = vec3(x, y, z);
            int voronoiIndex =                              x
                                +                 resolution.x*y
                                + resolution.y*resolution.x*z;
            int pointIndex = voronoi[voronoiIndex];
            if distance(cur3DPos, curPos) <=
              distance(curPos, points[pointIndex].pos) then
                result += points[pointIndex].value;
                divider += 1.0;
// since each voxel has the same volume, the result can be
divided by the total number of samples
if divider != 0.0 then
    result = result/divider;
// calculate linear index from cur3DPos and set customization
volume data
int outIndex =                                  cur3DPos.x
             +                  resolution.x*cur3DPos.y
             + resolution.y*resolution.x*cur3DPos.z;
interpolationResult[outIndex] = result;
```

in the newly created Voronoi cell associated with the seed position of the corresponding work item. If the distance (Euclidean or Manhattan) between the tested voxel position and the seed customization point of the work item is smaller than the distance of the tested voxel position to the seed customization point of the stored Voronoi cell in the Voronoi volume, we add the value of the visualization parameter of the stored seed customization point to a temporary sum and count how many values have been added to the sum so far. Since each voxel has the same volume, we get the correct interpolation result by dividing the temporary sum by the count of the terms of the sum. The stored value in the output buffer corresponds to the final customization volume.

Algorithm 3: Pseudocode of a ray-casting loop for volume rendering including the required modifications to support our customization volume. An intensity gamma correction parameter is simulated in this example.

Input : *InputVolume* is the volume which should be visualized.
CustomizationVolume is the previously calculated customization volume.
Helper: *getValue(volume,position)* returns the value stored in the *volume* at the given *position* with linear filtering.
applyTransFunc(intensity) returns the color value mapped from the input *intensity* by applying a transfer function.

```
1 for position ← ray.begin to ray.end do
      // regular code before modification
2     ...
3     float intensity = getValue(InputVolume, position);
      // modification takes place here
4     float gamma = getValue(CustomizationVolume, position);
5     intensity = pow(intensity, gamma);
      // regular code after modification
6     vec4 color = applyTransFunc(intensity);
7     ...
```

5.3. Integration into the Rendering Process

The specific integration depends on the actual visualization parameter that should be adjusted, and thus must be handled individually. An example of how a gamma correction of the intensity values of a volume data set could be realized is shown in Algorithm 3.
Independent from the visualization parameter the customization volume always has to be passed to the shader performing the volume rendering, e.g., ray-casting. In the main ray-casting loop, after fetching the intensity value of the data set at the current sampling position and before the transfer function is applied, a modification has been added, as highlighted in lines 4 and 5. In this example code our modification consists of getting the visualization parameter from the customization volume at the current sampling position (using linear filtering) and applying it to the intensity value of the input volume.
It should be noted that the fetched intensity values in this example are in the range $[0,1]$ to justify the intensity gamma correction in line 5. Overall, this method requires only minor modifications of the existing rendering pipeline.

5.4. Impact on the Rendering Performace

Due to the additional lookups and computations (such as the power function in Algorithm 3) the application of the customization volume reduces the overall performance of the volume rendering pipeline. We have tested the performance of our prototype implementation on a standard PC with 16 GB RAM and an AMD Radeon HD 7970 GPU

by integrating the additional computations into our OpenCL volume ray-caster and measuring the performance in comparison to the original implementation. For our tests, we applied the intensity gamma correction shown in Algorithm 3. We have tested three data sets, the head data set (Fig. 6) with a resolution of $160 \times 160 \times 128$ and a memory size of 6.3 MB, the heart data set (Fig. 5) with a resolution of $512 \times 512 \times 273$ and a memory size of 273 MB, and the mouse embryo volume (Fig. 1) with a resolution of $1004 \times 1002 \times 1611$ and a memory size of 9.5 GB for three intensity channels. For the head data set, the framerate of 70 fps did not suffer from the additional classification step, which is mainly due to the relatively opaque transfer function reducing the number of samples due to early ray termination. For the heart data set, the framerate decreased from 30 to 24 fps. For the mouse embryo data set, the customization volume was applied to each of the three channels separately in maximum intensity projection (MIP), causing a performance reduction from 15 to 9 fps. For the rendering of larger out-of-core volume data sets, the bottleneck of the rendering performance is the memory transfer of the volume data to the GPU and thus the performance impact of the step in which the customization volume is applied becomes negligible. It should be noted that, in addition to the number of samples, the computations necessary for the specific customization have a significant influence on how the customization step impacts the rendering performance. Changing the resolution of the customization volume between 32^3 and 128^3 has no significant impact on the rendering performance, although a higher resolution increases the computational complexity of the Voronoi diagram calculation and the interpolation computations, thus diminishing interactivity in adjusting the seed customization points. This problem, however, can be mitigated by choosing a lower resolution of the customization volume for the interactive customization, and a higher resolution for the actual volume rendering.

6. The User Interface

In our application setup (as shown in Fig. 6) a quad-view consisting of a 3D rendering and three 2D slice views (one for each of the main axes) and a list of properties for user configurations are provided.

Each 2D view shows the position of all of the seed customization points mapped to the current volume slice. A seed customization point is visualized as a circle, if its position is on the currently selected slice, or as a triangle with a small number, indicating the direction and distance to the exact voxel position of the seed customization point in 2D.

The 3D view visualizes each seed customization point at its three-dimensional position. To overcome the problem of occlusion in 3D rendering, e.g., in the direct volume rendering of an opaque object, the user can optionally render the seed customization points in the foreground, despite their actual depth. Since this effect hampers the perception of the exact position of the seed customization point, our interface provides the option to visualize axis aligned orientation lines from the seed customization point to the borders of the bounding box, as depicted in Fig. 6b.

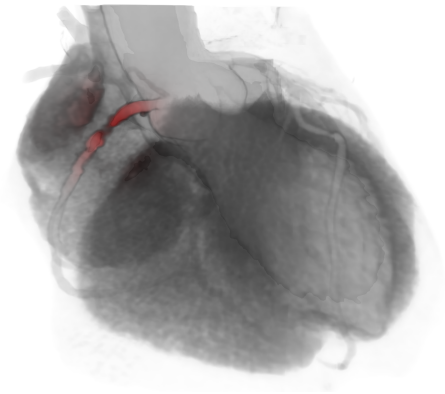
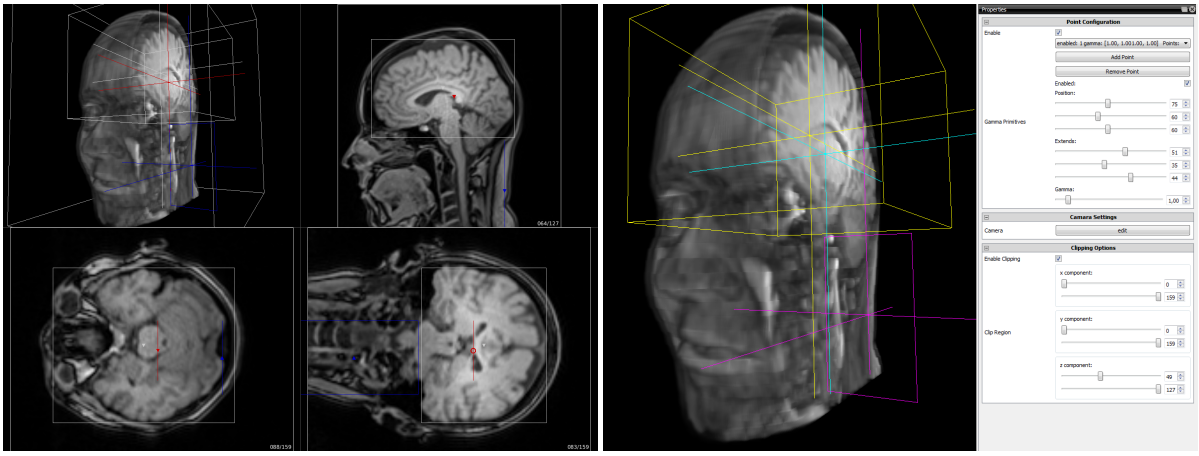


Figure 5: Direct volume rendering of a CT scan of the human heart. We have applied our technique to the opacity values of the transfer function classification to fade out outer regions of the volume, while modifying the color values in the region of interest to highlight the stenosis.

Two color schemes for the visualization of the seed customization points can be selected by the user to accommodate the visualization to the current use case. The first scheme is a mapping of one of the user-defined visualization parameters to a heat map, providing an overview of the customization values. The scheme is shown in Fig. 6a. To alleviate the cognitive load of matching the seed customization points in the 2D and 3D representations, which might especially be difficult with larger numbers of seed customization points, the second scheme colors the points according to their unique identifier which corresponds to the order in which the seed customization points have been created. This scheme is depicted in Fig. 6b. It should be noted that all of the points which belong to the same *primitive* (see Sec. 6.1) are rendered using the same color.

6.1. Primitives as a Shortcut

To allow the quick customization of a visualization parameter in a larger region using multiple points or to confine the influence of a seed customization point (e.g., by surrounding it by multiple points restricting its influence) the possibility to specify simple primitives is provided to the user. Our implementation supports the insertion of points, axis aligned lines (2 points), axis aligned rectangles (4 points), and axis aligned cuboids (8 points). Since all points of such a primitive share the same configuration of visualization parameters, this supports the simple modification of local regions by placing one seed customization point in the center of the region and a cuboid with the default value of the parameter around it to confine the region.



(a) The provided quad-view with one 3D and three 2D views. The color of a seed customization point represents its value.

(b) In this representation, each seed customization point has a unique color. Its properties can be defined on the right side.

Figure 6: Screenshots of the interface provided to define seed customization points in the data set.

6.2. Adding and Adjusting Points

The user can add new primitives and delete or disable existing primitives. Disabled seed customization points are ignored during the computation of the customization volume, but can be activated again. However, it should be noted that this functionality is not provided for the fixed corner points. The configuration is done via a property list (see Fig. 6b) containing information about all of the seed customization points, including the eight corner points. All of the attributes for a selected primitive, such as its position, spatial extent, or visualization parameters, can be adjusted. However, the position and spatial extent of the corner points of the bounding box are fixed. The user can specify a spatial extent for each dimension separately. An extent of zero in each direction corresponds to a point, a non-zero extent in one direction to a line and so on.

Regardless of the number of associated points, the position of a primitive is defined as its center. The position can be specified in voxel coordinates or can be set via direct interaction with the rendering views using the mouse. Due to the difficulties related to the exact placement of points in a 3D view using a mouse as an input device, we have focused our configuration input on the 2D views and use the 3D rendering only for providing additional visual feedback. Each update of the position will immediately trigger an update of the customization volume as well as the visualization of the seed customization points in all of the views, providing direct visual feedback to the user during interaction to support achieving the desired result.

Figure 5 shows an example where our approach has been applied to a CT data set of the human heart. A cuboid has been specified as a region of interest. The additional classification step is used to lower the opacity values of the transfer function outside of this region. Additionally, the color values of the transfer function within the region are modified. This allows to highlight the stenosis in the example, omitting occlusion while still keeping the spatial context of the volume which would be lost when using a simple clipping approach.

7. Conclusion and Future Work

In this paper we have proposed a novel approach for extending the classification of feature characteristics using a transfer function in volume rendering applications by incorporating an additional step to allow an interactive position-dependent customization of visualization parameters related to the classification. We provide a simple user interface with direct visual feedback and have shown examples to illustrate the applicability of our system in practice. Our method can easily be integrated into existing volume rendering pipelines, particularly including the visualization of large volume data, which is due to the configurability of the memory requirements of the additional customization volume. Although incorporating an additional step into the classification process reduces the overall rendering performance, this effect becomes negligible compared to the required memory transfer for rendering large volume data.

In future work, we plan to further refine the user interface of our system to support the specification of non axis-aligned primitives and extend the implementation to include more sophisticated parameters such as color maps for the classification. Furthermore, we plan to evaluate the applicability of our method to specific use cases emerging in medical and biological research in direct collaboration with domain experts, and combine our approach with related methods relying on segmentation techniques for more flexibility. Moreover, we plan to apply our method to other stages of the volume rendering pipeline such as gradient computation or lighting calculations.

8. Acknowledgments

We would like to thank the research group of Friedemann Kiefer at the Max-Planck-Institut in Münster for providing us the biological data sets and Christoph Schülke at the Department of Clinical Radiology at the University Hospital Münster for providing us the medical data sets illustrated in this paper. This work was partly supported by the Deutsche Forschungsgemeinschaft, CRC 656 "Cardiovascular Molecular Imaging" (projects Z1 and Ö).

References

[BG05] BRUCKNER S., GRÖLLER M. E.: VolumeShop: An Interactive System for Direct Volume Illustration. In *Proceedings of IEEE Visualization 2005* (Oct. 2005), C. T. Silva E. Gröller H. R., (Ed.), pp. 671–678. 2

[BKW08] BÜRGER K., KRÜGER J., WESTERMANN R.: Direct Volume Editing. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (Nov. 2008), 1388–1395. 2

[BPH14] BRIX T., PRASSNI J.-S., HINRICHS K. H.: Visualization of Large Volumetric Multi-Channel Microscopy Data Streams on Standard PCs. *CoRR abs/1407.2074* (2014). 6

[cga] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org. 6

[CM08] CORREA C., MA K.-L.: Size-based Transfer Functions: A New Volume Exploration Technique. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (Nov 2008), 1380–1387. 2

[CM09] CORREA C., MA K.-L.: The Occlusion Spectrum for Volume Classification and Visualization. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1465–1472. 2

[CM11] CORREA C., MA K.-L.: Visibility Histograms and Visibility-Driven Transfer Functions. *IEEE Transactions on Visualization and Computer Graphics 17*, 2 (Feb. 2011), 192–204. 2

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)* (Boston, MA, USA, feb 2009), ACM, ACM Press. 2

[Del34] DELAUNAY B.: Sur la Sphere Vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7*, 793-800 (1934), 1–2. 5

[GMG08] GOBBETTI E., MARTON F., GUITIÁN J. A. I.: A Single-pass GPU Ray Casting Framework for Interactive Out-of-core Rendering of Massive Volumetric Datasets. *The Visual Computer 24*, 7-9 (2008), 797–806. 2

[GY13] GUO H., YUAN X.: Local WYSIWYG Volume Visualization. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific* (Feb 2013), pp. 65–72. 2

[Hou80] HOUNSFIELD G. N.: Computed Medical Imaging. *Journal of Computer Assisted Tomography 4*, 5 (1980), 665–674. 2

[KHS*10] KERWIN T., HITTLE B., SHEN H.-W., STREDNEY D., WIET G.: Anatomical Volume Visualization with Weighted Distance Fields. In *Eurographics Workshop on Visual Computing for Biomedicine* (2010), vol. 2010, NIH Public Access, p. 117. 2

[Kin02] KINDLMANN G.: Transfer Functions in Direct Volume Rendering: Design, Interface, Interaction. *Course notes of ACM SIGGRAPH* (2002). 2

[KKH02] KNISS J., KINDLMANN G., HANSEN C.: Multidimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 270–285. 2

[Kri51] KRIGE D. G.: A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa 52*, 6 (Dec. 1951), 119–139. 4

[KSC*10] KIM H. S., SCHULZE J. P., CONE A. C., SOSINSKY G. E., MARTONE M. E.: Multichannel Transfer Function with Dimensionality Reduction. In *IS&T/SPIE Electronic Imaging* (2010), International Society for Optics and Photonics, p. 75300A. 2

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 287–292. 2

[LCN98] LICHTENBELT B., CRANE R., NAQVI S.: *Introduction to Volume Rendering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998. chapter 4. 2

[Lev88] LEVOY M.: Display of Surfaces from Volume Data. *Computer Graphics and Applications, IEEE 8*, 3 (1988), 29–37. 2

[MSRMH09] MEYER-SPRADOW J., ROPINSKI T., MENSMANN J., HINRICHS K.: Voreen: A Rapid-prototyping Environment for Ray-casting-based Volume Visualizations. *Computer Graphics and Applications, IEEE 29*, 6 (2009), 6–13. 6

[PLK*06] PARK S. W., LINSEN L., KREYLOS O., OWENS J. D., HAMANN B.: Discrete Sibson Interpolation. *IEEE Transactions on Visualization and Computer Graphics 12*, 2 (Mar. 2006), 243–253. 6

[PT15] PION S., TEILLAUD M.: 3D Triangulations. In *CGAL User and Reference Manual*, 4.5.2 ed. CGAL Editorial Board, 2015. 6

[RBS05] ROETTGER S., BAUER M., STAMMINGER M.: Spatialized Transfer Functions. In *Proceedings of the Seventh Joint Eurographics / IEEE VGTC Conference on Visualization* (Aire-la-Ville, Switzerland, Switzerland, 2005), EUROVIS'05, Eurographics Association, pp. 271–278. 2

[SG09] SELVER M. A., GÜZELIŞ C.: Semiautomatic Transfer Function Initialization for Abdominal Visualization Using Self-Generating Hierarchical Radial Basis Function Networks. *IEEE Transactions on Visualization and Computer Graphics 15*, 3 (May 2009), 395–409. 2

[She68] SHEPARD D.: A Two-dimensional Interpolation Function for Irregularly-spaced Data. In *Proceedings of the 1968 23rd ACM National Conference* (New York, NY, USA, 1968), ACM '68, ACM, pp. 517–524. 4

[Sib81] SIBSON R.: A Brief Description of Natural Neighbour Interpolation. *Interpreting Multivariate Data 21* (1981). 5

[TPD06] TAPPENBECK A., PREIM B., DICKEN V.: Distance-based Transfer Function Design: Specification Methods and Applications. In *SimVis* (2006), pp. 259–274. 2

[WKK10] WESARG S., KIRSCHNER M., KHAN M.: 2D Histogram-based Volume Visualization: Combining Intensity and Size of Anatomical Structures. *International Journal of Computer Assisted Radiology and Surgery 5*, 6 (2010), 655–666. 2

[ZDT04] ZHOU J., DÖRING A., TÖNNIES K. D.: Distance Transfer Function Based Rendering. In *International Telegraph and Telephone Consultative Committee* (2004). 2