EUROGRAPHICS 2007 / D. Cohen-Or and P. Slavík
(Guest Editors)

Volume 26 (2007), Number 3

Style Transfer Functions for Illustrative Volume Rendering

S. Bruckner and M. E. Gröller†

Insitute of Computer Graphics and Algorithms
Vienna University of Technology, Austria

Abstract

Illustrative volume visualization frequently employs non-photorealistic rendering techniques to enhance important features or to suppress unwanted details. However, it is difficult to integrate multiple non-photorealistic rendering approaches into a single framework due to great differences in the individual methods and their parameters. In this paper, we present the concept of style transfer functions. Our approach enables flexible data-driven illumination which goes beyond using the transfer function to just assign colors and opacities. An image-based lighting model uses sphere maps to represent non-photorealistic rendering styles. Style transfer functions allow us to combine a multitude of different shading styles in a single rendering. We extend this concept with a technique for curvature-controlled style contours and an illustrative transparency model. Our implementation of the presented methods allows interactive generation of high-quality volumetric illustrations.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Volume rendering is a well established method for the visualization of scientific data, such as tomographic scans. Historically, most volume rendering techniques are based on an approximation of a realistic physical model. It was noticed, however, that traditional depictions of the same types of data – as found in medical textbooks, for example – deliberately use non-realistic techniques in order to focus the viewer's attention to important aspects [ER00, RE01]. Using abstraction, visual overload is prevented leading to a more effective visualization. Recent approaches have considered this fact, leading to an increased interest in *illustrative volume visualization*.

Approaches for illustrative volume visualization frequently employ non-photorealistic rendering techniques to mimic the style of traditional illustrations. They take advantage of the illustrators' century-long experience in depicting complex structures in an easily comprehensible way. Many of these techniques require tedious tuning of various parameters to achieve the desired result. We aim to circumvent this issue by presenting the user with a gallery of styles extracted from actual illustrations.

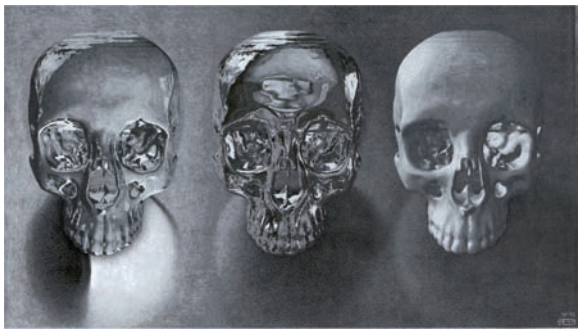
For this purpose, we introduce the concept of *style transfer functions*. Instead of defining a color transfer function which is augmented with various parameters for controlling the non-photorealistic rendering, our approach allows the user to directly specify styles captured from existing artwork in the transfer function. Style transfer functions allow the generation of volumetric illustrations in the style of a given work of art (see Figure 1).

Contours are an important concept to enhance the stylized depiction of volume data. We present a simple measure to calculate the curvature along the viewing direction which is used to control the thickness of style-based contours. Our approximation is computationally very efficient since it does not require explicit reconstruction of second-order partial derivatives. Using this curvature measure, we also introduce a new transparency model based on techniques commonly found in traditional illustrations. Our implementation is capable of producing high-quality illustrations of volume data using a wide variety of different styles at interactive frame rates.

This paper is structured as follows: In Section 2, we dis-

† E-mail: {bruckner|groeller}@cg.tuwien.ac.at

(a)



(b)

Figure 1: *Using lit sphere maps from existing artwork. (a) "Three Spheres II" (lithograph, 1946) by Dutch artist M. C. Escher. (b) Direct volume renderings of a human skull using the respective spheres as style, Escher's painting is used as background.*

cuss related work. Section 3 presents the concept of style transfer functions. Our implementation is detailed in Section 4. Section 5 discusses the results of our approach. The paper is concluded in Section 6.

2. Related Work

In computer graphics, many techniques have been developed to capture lighting effects in order to plausibly embed objects in photographs or video or to create new scenes under the same environmental conditions [Deb98, SSI99, DHT*00]. For non-photorealistic rendering, approaches have been presented to reproduce numerous artistic techniques, such as tone shading [GGSC98], pencil drawing [SB99], hatching [PHWF01], or ink drawing [SFWS03]. While these are specialized algorithms which aim to accurately simulate a particular technique, Sloan et al. [SMGG01] employ a simple method to approximately capture non-photorealistic shading from existing artwork. Their approach forms one building block of style transfer functions which we introduce in this paper (see Section 3.1).

In the context of volume visualization, the combination of different rendering styles is of particular interest, as it allows to put emphasis on features of interest. Ebert and Rheingans [ER00, RE01] present several illustrative techniques which enhance structures and add depth and orientation cues. They also propose to locally apply these methods for regional enhancement. Lu et al. [LME*02, LMT*03] developed an interactive direct volume illustration system that simulates traditional stipple drawing. Csébfalvi et al. [CMH*01] visualize object contours based on the magnitude of local gradients as well as on the angle between viewing direction and gradient vector using depth-shaded maximum intensity projection. The concept of two-level volume rendering proposed by Hauser et al. [HMBG01], allows focus+context visualization of volume data by combining maximum intensity projection and direct volume rendering. Viola et al. [VKG05], inspired by cutaway views which are commonly used in technical illustrations, apply different compositing strategies to prevent an object from being occluded by less important structures. Nagy et al. [NSW02] combine line drawings and direct volume rendering techniques. Yuan and Chen [YC04] enhance surfaces in volume rendered images with silhouettes, ridge and valleys lines, and hatching strokes. Tietjen et al. [TIP05] use a combination of illustrative surface and volume rendering for visualization in surgery education and planning. Salah et al. [SBS05] employ point-based rendering for non-photorealistic depiction of segmented volume data. Techniques by Lu and Ebert [LE05] as well as Dong and Clapworthy [DC05] employ texture synthesis to apply different styles to volume data. Their approaches, however, do not deal with shading. Krüger et al. [KSW06] use interactive magic lenses based on traditional illustration techniques for focus+context visualization of iso-surfaces.

Multi-dimensional transfer functions have been proposed to extend the classification space and to allow better selection of features. Kniss et al. [KKH01, KKH02] use a two-dimensional transfer function based on scalar value and gradient magnitude to effectively extract specific material boundaries and to convey subtle surface properties. Hladuvka et al. [HKG00] propose the concept of curvature-based transfer functions. Kindlmann et al. [KWTM03] employ curvature information to achieve illustrative effects, such as ridge and valley enhancement. Lum and Ma [LM04] assign colors and opacities as well as parameters of the illumination model through a transfer function lookup. They apply a two-dimensional transfer function to emphasize material boundaries using illumination.

3. Style Transfer Functions for Illustrative Volume Rendering

In this section, we present the concept of style transfer functions as well as additional techniques to enhance the illustrative depiction of volume data.

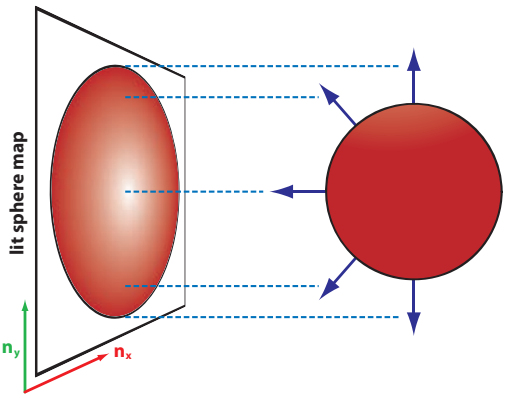


Figure 2: *Lit sphere shading. The shading of an object is represented as a function of eye-space normal orientation.*

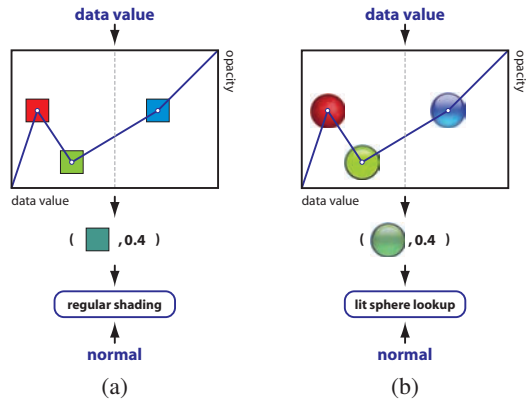


Figure 3: *Basic concept of style transfer functions. (a) Regular transfer function. (b) Style transfer function.*

3.1. Lit Sphere Shading

Sloan et al. [SMGG01] presented a simple yet effective method for capturing artistic lighting by using an image of a sphere shaded in the desired style. They employed this approach for non-photorealistic rendering of polygonal models. The basic idea is to capture color variations of an object as a function of normal direction. As a sphere provides coverage of the complete set of unit normals, an image of a sphere under orthographic projection will capture all such variations on one hemisphere (see Figure 2). This image is then used as a sphere map indexed by the eye-space normals to shade another object. Essentially, the sphere acts as a proxy object for the illumination. In their work, Sloan et al. also describe a method for extracting lit sphere maps from non-spherical regions in a piece of artwork. They present an interactive tool which allows rapid extraction of shading styles from existing images.

The lit sphere map itself is a square texture where texels outside an inscribed disk are never accessed. Normal vectors parallel to the viewing direction map to the center of the disk and normal vectors orthogonal to the viewing direction map to the circumference of the disk. The lit sphere map is indexed by simply converting the n_x and n_y components of the eye-space normal $n = (n_x, n_y, n_z)$ which are in the range $[-1..1]$ to texture coordinate range (usually $[0..1]$). As the n_z component is ignored, lighting does not distinguish between front and back faces. This is desired as the gradient direction in the volume which serves as the normal might be flipped depending on the data values at a material boundary.

While lit sphere shading fails to capture complex aspects of realistic illumination, it is well-suited to represent the general shading style of an object. Images of an illuminated sphere are relatively easy to obtain and the extraction process described by Sloan et al. allows to build up a large database of styles with little effort. Another advantage is the view-dependency of this technique. All lighting effects will appear as if the light source was a headlight, i.e. as if it were rotating with the camera. Generally, this is the desired setup in volume visualization. For these reasons, we employ lit sphere maps as the basic components of our style transfer functions.

3.2. Style Transfer Functions

We assume a continuous volumetric scalar field $f(P)$. A sample value at an arbitrary position P is denoted by $s = f(P)$. We denote the gradient at position P by $g = \nabla f(P)$. For the purpose of shading, the normalized gradient $n = \frac{g}{|g|}$ serves as the normal. Conventionally, the transfer function assigns color and opacity to each scalar value. There are approaches that use multi-dimensional transfer functions which employ derivatives of the volumetric function, such as the gradient magnitude or the curvature magnitudes. For simplicity we will restrict our discussion to one-dimensional transfer functions at this point. Our technique equally applies to multi-dimensional transfer functions (see Section 4.2 for a discussion of this matter).

During rendering, at each sample point the scalar value and the gradient are reconstructed. The transfer function defines the color and opacity contribution of this sample, while the gradient is used to compute the illumination. The illumination model and its parameters are usually fixed, i.e. they are not dependent on the transfer function. Lum et al. [LM04] presented an approach were the parameters of the Phong illumination model are specified by an additional lighting transfer function. We extend this idea of data-dependent lighting to enable a wide variety of non-photorealistic shading styles. In our approach, we integrate color and shading information in a combined style transfer function. Mathematically, this is equivalent to extending the transfer function domain by including normal direction. A one-dimensional transfer function based on the scalar value becomes three-dimensional, a two-dimensional transfer function becomes four-dimensional, etc.

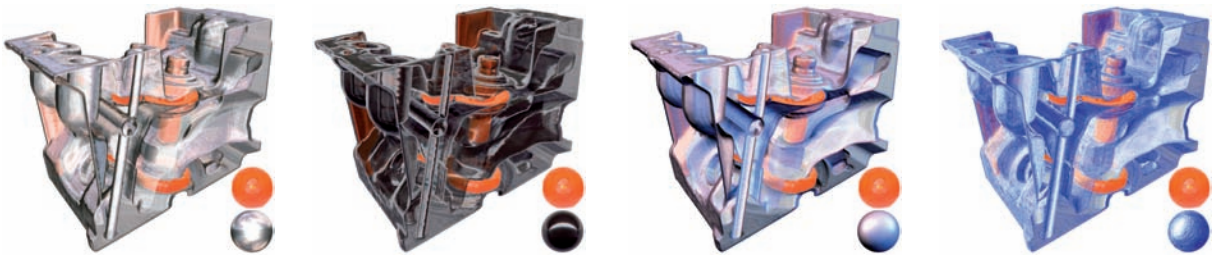


Figure 4: *Engine block rendered using different style transfer functions. The lit sphere maps used in the transfer function are depicted at the bottom right corner of each image.*

Transfer functions are usually implemented as lookup tables. The memory requirements for storing a complete style transfer function lookup table would be prohibitively high due to the increase in dimensionality. However, as there is only a discrete number of styles it is not necessary to store the whole function. We can store the set of styles separately. The transfer function lookup table now contains references to these styles instead of colors. The only restriction necessary is that when interpolating between two styles, the interpolation is performed uniformly for all normal directions, i.e. transitions only occur between whole styles. If a non-uniform transition is desired, this can easily be accomplished by adding one or multiple intermediate styles. Conceptually, this can be illustrated by replacing the single color of a transfer function entry by a lit sphere map (see Figure 3). When performing a style transfer function lookup, styles are first interpolated according to the specified transfer function. Using this interpolated style, the eye-space normal direction then determines the final sample color.

From a user's point of view, the transfer function now not only specifies the color over the range of data values, but also the shading as a function of eye-space normal direction. The complexity of specifying a transfer function, however, is not increased. Instead of assigning a single color to a certain value range, a pre-defined shading style represented by a lit sphere map is chosen. In this context, one advantage of sphere maps as opposed to other mappings is that they can be directly presented to the user as an intuitive preview image for the style.

Style transfer functions allow for a flexible combination of different shading styles in a single transfer function. Unshaded volume rendering (a constant color sphere), tone shading, cartoon shading, metallic shading, painterly rendering, and many other styles can be used in the same rendering. Style transfer functions also enable inconsistent lighting of different structures in a single data set as a means of accentuating features [LHV04]. Figure 4 shows examples of different styles applied to a data set.

3.3. Style Contours

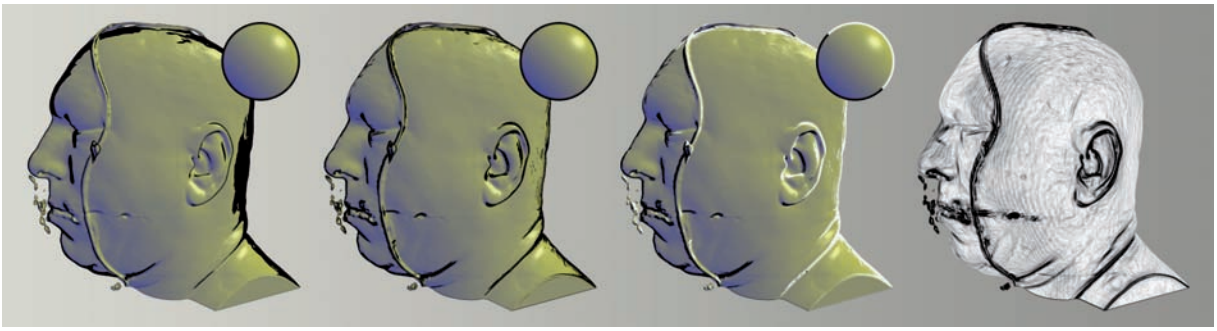
Illustrators frequently employ contours to enhance the depiction of objects. Contours help to clearly delineate object shape and resolve ambiguities due to occlusion by emphasizing the transition between front-facing and back-facing surface locations [ST90]. In volume rendering, contours are generally produced using the dot product between the view vector v and the normal n. The sample color is darkened if v is approximately orthogonal to n, i.e., $v \cdot n$ is close to zero. The drawback of this method is an uncontrolled variation in the apparent contour thickness. Where the surface is nearly flat, a large region of surface normals is nearly perpendicular to the view vector, making the contours too thick. Conversely, in fine structures, where the emphasis provided by contours could be especially helpful, they appear to be too thin. These deficiencies are illustrated in Figure 5 (a).

To remedy this problem, Kindlmann et al. [KWTM03] proposed to regulate contours based on the normal curvature along the view direction κ_v. A sample is defined to be on a contour if the following condition is true:

$$|n \cdot v| \leq \sqrt{T\kappa_v(2 - T\kappa_v)} \tag{1}$$

where T is a user-defined thickness value. While this method is effective in depicting contours of constant thickness, it requires the expensive reconstruction of second-order derivatives of the volumetric function. Specifically, the curvature measure κ_v is based on the geometry tensor. The geometry tensor is constructed from the Hessian matrix. Computing the geometry tensor in a fragment program is very expensive and would not allow for interactive performance. On the other hand, pre-computation would require two additional 3D textures (the geometry tensor is symmetric and can be stored in six values per voxel). Hadwiger et al. [HSS*05] circumvent this problem by restricting themselves to iso-surfaces, but for direct volume rendering no viable solution has been presented so far.

We propose a simple approximation of κ_v which does not suffer from these drawbacks. We are interested in the rate of



(a) (b) (c) (d)

Figure 5: *Style contours. (a) Contours without curvature-controlled thickness. (b) Curvature-controlled contours with constant color. (c) Curvature-controlled contours with varying colors. (d) Our curvature measure (darker regions corresponds to higher curvature).*

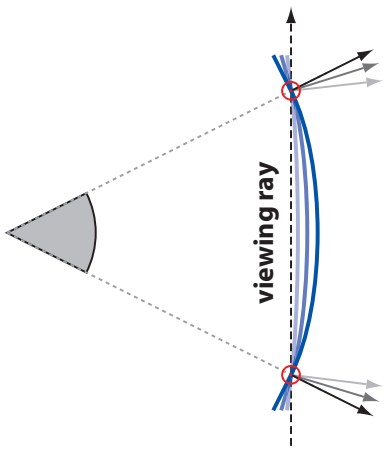


Figure 6: *Using the angle between the normals of two subsequent points along a viewing ray as an approximate measure for the curvature along the view direction* κ_v.

change in normal direction of the iso-surface corresponding to the current sample value along the viewing direction. When performing volume ray casting, we step along the ray direction and evaluate the normal at every sample point. The angle between two subsequent normals along the ray taken at a sufficiently small distance gives us information about the curvature along the viewing direction (see Figure 6). When performing ray casting, we can therefore use the angle between the normal at the current sample point and the previous normal divided by the step size as an estimate for κ_v. This is of course not accurate, as we are not stepping along the iso-surface. However, due to the finite resolution of the volume this coarse approximation has proven to be sufficient for our purposes. The advantage of this approach is that it introduces almost no additional costs as the normal is evaluated at every sample point anyway. Since we now have a measure for the curvature along the viewing direction, we can employ the criterion proposed by Kindlmann et al. to determine whether a sample is located on a contour (Equation 1). Using a fixed contour color, however, would be potentially inconsistent with the selected styles. Instead, the contour color should be determined by the style transfer function. For this reason we adjust the coordinates for the lit sphere map lookup based on our curvature measure: if a sample falls below the contour threshold, we simply push the coordinates outwards along the radius of the sphere in the following way: If $r = |n_{x,y}|$, i.e., the length of the eye-space normal n projected onto the lit sphere map, we adjust the length of $n_{x,y}$ to $r' = \min(1, \frac{r}{\delta})$ with $\delta = 1 - \min(1, \frac{\sqrt{T\kappa_v(2-T\kappa_v)}-|n\cdot v|}{\sqrt{T\kappa_v(2-T\kappa_v)}})$. This not only allows for varying contour appearance between different styles, but also for a variation based on the normal direction. Contours in a highlight region, for example, may be brighter than in a dark region. Figure 5 (b) uses a style with constant contour color while Figure 5 (c) employs varying contour colors. Our curvature measure is depicted in Figure 5 (d).

3.4. Illustrative Transparency

In volume visualization transparency is frequently used in order to depict complex three-dimensional structures. Our approach provides two basic ways to control opacity:

Uniform opacity α_u**.** The opacity value in the transfer function controls the overall opacity of a sample independent of normal direction.

Directional opacity α_d**.** Each entry in a lit sphere map is an (r, g, b, α) tuple. This allows for varying opacity based on the normal direction.

While α_u allows to control opacity independent of style, α_d is a function of the style. For the overall opacity we want to apply the following two constraints in order to maintain the semantics of opacity control in the transfer function:

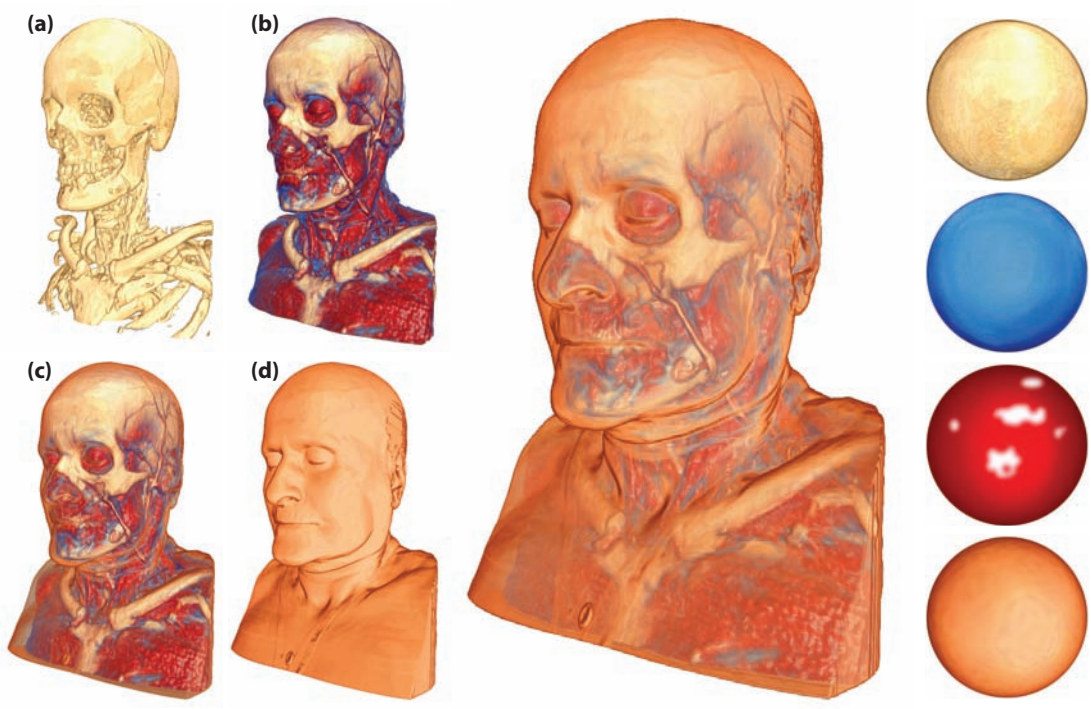


Figure 7: *Illustrative volume rendering using a style transfer function. Images (a)-(d) depict different opacity settings.*

- If the value of α_u is one, the opacity of a sample should be solely determined by α_d.
- If the value of α_u is zero, the sample should be completely transparent.

Transparency in illustrations frequently employs the 100-percent-rule where transparency falls off close to the edges of transparent objects and increases with the distance to edges [DWE02]. Since this technique non-uniformly decreases the opacity of an object, it results in a clearer depiction of transparent structures while still enabling the viewer to see through them. In order to achieve an effect similar to the 100-percent-rule, we employ a modulation of α_u with the curvature measure proposed in the previous section and the gradient magnitude to compute the overall opacity α of a sample:

$$\alpha = \alpha_d \cdot \alpha_u^{0.5+\max(0,|n\cdot v|-\sqrt{T\kappa_v(2-T\kappa_v)})\cdot(1-|g|)} \quad (2)$$

If the exponent is lower than one, the opacity of a sample is enhanced, if it is greater than one the opacity is reduced. The term $\max(0, |n \cdot v| - \sqrt{T\kappa_v(2 - T\kappa_v)})$ is zero when the sample point is on the contour, and increases as points are farther away from the contour. The term $1 - |g|$ ranges from zero to one and is included to prevent enhancement of nearly homogeneous regions, where noise causes the gradient direction to vary rapidly. When decreasing α_u from one to zero, flat and homogeneous regions become more transparent first. As α_u drops further, the remaining contour regions also begin to become more transparent. The constant of 0.5 restricts the maximum opacity enhancement. This value was empirically determined and has proven to be effective for all our test data sets. The overall effect is weighted by α_d. An example for our transparency model is shown in Figure 7.

4. Implementation

In this section we describe our implementation of style transfer functions for a GPU-based ray casting approach. Our renderer makes use of conditional loops and dynamic branching available in Shader Model 3.0 GPUs. It was implemented in C++ and OpenGL/GLSL. The presented techniques can be integrated into an existing renderer using regular transfer functions with little effort.

4.1. Style Transfer Function Lookup

A transfer function is usually implemented as a lookup table which corresponds to a 1D texture on the GPU. For conventional transfer functions, this texture stores an (r, g, b, α) tuple for every data value. At each sample point, the interpolated data value is used to perform a texture lookup in this 1D texture to retrieve the color and opacity of the sample.

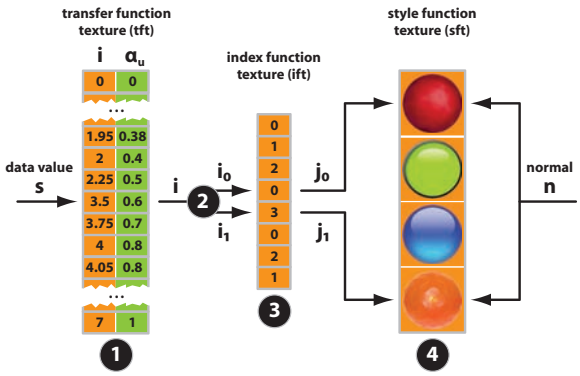


Figure 8: *Style transfer function lookup for data value s and normal n.*

Within the transfer function texture, linear interpolation is performed. A naive implementation of a style transfer function would simply replace the 1D texture by a 3D texture which stores an (r,g,b,α) tuple for every data value and normal direction. This approach requires only one texture lookup and exploits native trilinear interpolation. As discussed in Section 3.2 this is not practical due to high storage requirements. Thus, we use an alternative approach which does not suffer from this problem. Our implementation uses three different textures:

Transfer function texture tft**.** This 1D texture stores the uniform opacities α_u and index values i for each data value. The index values in the transfer function texture range from zero to $N-1$, where N is the number of styles specified in the style transfer function. For example, an index value of one corresponds to the second style, two corresponds to the third style, etc. Fractional values indicate that an interpolation between two styles has to be performed.

Index function texture ift**.** As one style might be used multiple times for different value ranges, we define M as the number of distinct styles in the style transfer function. The one-dimensional index function texture maps the index values i (ranging from zero to $N-1$) retrieved from the transfer function texture to locations j in the style function texture (ranging from zero to $M-1$). As this mapping is discrete, no interpolation is performed for index function texture lookups. If no style is used multiple times, the index function texture lookup can be skipped.

Style function texture sft**.** This texture contains the discrete set of M distinct styles specified in the current style transfer function. As each style is a two-dimensional image, an intuitive representation for this function would be a 3D texture. Since this can lead to problems with mipmapping, an alternative way of storage may be more appropriate (see Section 4.3).

Using these three textures, the complete lookup proceeds as follows (see Figure 8):

1. Using the sample value s, retrieve the index value i and the uniform opacity α_u from the transfer function texture tft: $(i,\alpha_u) = tft(s)$.
2. Compute the indices to be used in the index function texture lookup $i_0 = \lfloor i \rfloor$, $i_1 = i_0 + 1$ and the interpolation weight $w = i - i_0$.
3. Retrieve the style indices j_0 and j_1 using two lookups into the index function texture ift: $j_0 = ift(i_0)$, $j_1 = ift(i_1)$. If no style occurs multiple times in the style transfer function, these lookups can be skipped.
4. Using the n_x and n_y components of the eye-space normal and the style indices j_0 and j_1, perform two lookups into the style function texture sft and linearly interpolate between them: $(r,g,b,\alpha_d) = sft(n_x,n_y,j_0)\cdot(1-w) + sft(n_x,n_y,j_1)\cdot w$.

4.2. Multi-dimensional Transfer Functions

So far, we have restricted our discussion to extending one-dimensional transfer functions based on the data value to style transfer functions. However, the presented techniques also apply to multi-dimensional domains. To illustrate the generality of our approach, this section describes the changes necessary to employ two-dimensional transfer functions:

The transfer function texture becomes a 2D texture and stores two indices ix and iy instead of i. The first index ix increases along the horizontal axis and the second index iy increases along the vertical axis. The index function texture also becomes two-dimensional. Its width is now the maximum number of horizontal style nodes in the two-dimensional transfer function, its height is the maximum number of vertical nodes. Analogous to the one-dimensional case, the indices for the index function texture lookup are: $ix_0 = \lfloor ix \rfloor$, $iy_0 = \lfloor iy \rfloor$, $ix_1 = ix_0 + 1$, $iy_1 = iy_0 + 1$. The two interpolation weights are also computed accordingly: $wx = ix - ix_0$, $wy = iy - iy_0$. Four lookups into the index function texture ift are performed to retrieve the four style indices: $j_{00} = ift(ix_0, iy_0)$, $j_{10} = ift(ix_1, iy_0)$, $j_{01} = ift(ix_0, iy_1)$, and $j_{11} = ift(ix_1, iy_1)$. Finally, these four style indices and the n_x and n_y components of the eye-space normal are used to perform four lookups into the style function texture sft. The final color is computed by bilinear interpolation: $(r,g,b,\alpha_d) = (sft(n_x,n_y,j_{00})\cdot(1-wx) + sft(n_x,n_y,j_{10})\cdot wx) \cdot (1-wy) + (sft(n_x,n_y,j_{01})\cdot(1-wx) + sft(n_x,n_y,j_{11})\cdot wx)\cdot wy$.

This procedure is independent of the actual quantities mapped to each axis. While data value and gradient magnitude are common choices [KKH01, KKH02], many other attributes are useful in the context of specific applications. Rendering of segmented data, for example, is frequently realized through a two-dimensional lookup using the data

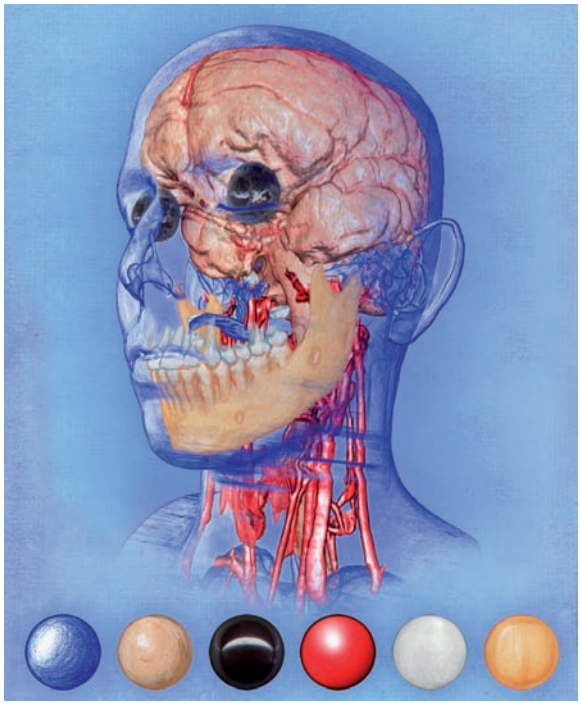


Figure 9: *Rendering of segmented volume data using a multi-dimensional style transfer function based on data value and object membership.*

value and an object identifier [HBH03]. Figure 9 shows an example of such a multi-dimensional style transfer function. In this way other measures such as distance [ZDT04], saliency [KV06], or importance [VKG05], either predefined or computed on-the-fly, could be mapped to visual styles easily. Further exploration of these possibilities will be part of our future work.

4.3. Mip-Mapping

Current graphics hardware uses mip-mapping to avoid aliasing in texture mapping. To take advantage of the GPU's mip-mapping capabilities for style lookups, certain considerations have to be made. First, for 3D textures, each dimension is halved for every subsequent mip-map level. If styles are stored as slices of a 3D texture, undesired mixing between styles occurs at higher mip-map levels. Thus, if the style function texture is implemented as as a 3D texture, mip-mapping has to be disabled. One solution to this problem is the EXT_texture_array OpenGL extension. A texture array is a collection of two-dimensional images arranged in layers. Mip-mapping is performed separately for each layer. This extension is currently only available on GeForce 8 series graphics hardware. A third possibility is to arrange the styles in a single 2D texture. Although this slightly complicates indexing, it is the option of our choice as it is supported

Figure	Regular TF	Style TF
Figure 7 (a)	11.7 fps	11.9 fps
Figure 7 (b)	10.5 fps	9.6 fps
Figure 7 (c)	10.1 fps	8.1 fps
Figure 7 (d)	12.5 fps	12.8 fps

Table 1: *Performance comparison of style transfer functions and regular transfer functions measured on a system equipped with an AMD Athlon 64 X2 Dual 4600+ CPU and an NVidia GeForce 7900 GTX GPU. Performance numbers are given in frames per second. Data dimensions:* $256 \times 256 \times 230$. *Viewport size:* 512×512. *Object sample distance:* 0.5.

on a wider range of hardware. We perform custom mip-map generation to avoid mixing between styles at their borders. Conventionally, when performing a texture lookup the appropriate mip-map level is determined by the hardware using the screen-space derivatives of the texture coordinates. These derivatives are undefined when the texture fetch takes place within conditionals or loops. Thus, as our algorithm uses raycasting, we cannot exploit the standard mechanism. We therefore manually compute the size of a projected voxel at each sample point to determine the mip-map level. In areas of high curvature, the gradient direction varies more quickly, which can lead to artifacts. Additionally, when its magnitude approaches zero, the gradient vector becomes a less reliable predictor for the normal direction. We therefore also bias the determined mip-map level using a function of curvature and gradient magnitude.

4.4. Performance

In comparison to a regular one-dimensional transfer function, a style transfer function lookup requires a maximum of four additional texture fetches (two for the index function texture and two for the style function texture). The index function texture does not require filtering and is rather small. It therefore heavily benefits from texture caching. On GeForce 8 series hardware it could also be implemented as a buffer texture using the EXT_texture_buffer_object OpenGL extension for additional performance gains. Although the additional texture fetches incur an overhead, the cost for evaluating the illumination model is saved when using style transfer functions.

To evaluate the performance of our approach, we compared the use of a style transfer function for classification and shading to a regular one-dimensional transfer function with simple Phong shading. The same opacities were used for both transfer functions. Both implementations use empty-space skipping and early-ray termination. The viewport size was 512×512 and the object sample distance was set to 0.5. We used the data set depicted in Figure 7 (dimensions: $256 \times 256 \times 230$). Our test system was equipped with

an AMD Athlon 64 X2 Dual 4600+ CPU and an NVidia GeForce 7900 GTX GPU. The results of this comparison are shown in Table 1. If only one style is visible, the performance is approximately equal (style transfer functions are even slightly faster) as all lighting computations are replaced by texture fetches which benefit from coherent access. If multiple styles are visible, the style transfer function performs slightly worse due to texture caching effects. In total, however, the overhead of employing a style transfer function is only minor but greatly increases the flexibility.

5. Discussion

In our experiments, style transfer functions have proven to be a simple method for generating images and animations in a wide variety of different appearances. Lit sphere maps are particularly effective in representing the styles typically used in medical illustrations. Our approach is well-suited for this application, as illustrations frequently rely on certain shading conventions. This means that a database of styles can potentially be reused for a large number of data sets. Figures 7 and 9, for example, use styles obtained from medical illustrations. Another advantage is that the theme of an image can quickly be changed by simply replacing one set of styles with another one. This is illustrated in Figure 10, where two very different results are achieved by a simple exchange of styles.

While the representation of styles as lit sphere maps has proven to be effective and efficient, it has drawbacks. One problem already discussed by Sloan et al. [SMGG01] occurs when the sphere contains prominent texture features. When the camera is rotated, they will appear to follow the eye leading to an undesired metallic impression. To solve this problem, texture and lighting information have to be separated. The texture information could then be aligned to the object, for example based on curvature directions. This might be an interesting direction for future research.

6. Conclusions

In this paper, we presented a new technique for data-dependent image-based shading of volumetric data. We introduced the concept of style transfer functions which define the color at a sample point as a function of the data value and the eye-space normal. This allows for flexible combination of different rendering styles. We extended our approach to handle thickness-controlled style-based contours using an efficient approximation of the normal curvature along the viewing direction. Furthermore, we introduced a new transparency model designed to emulate techniques employed by illustrators. Our framework is able to generate high-quality images resembling traditional illustrations at interactive frame-rates.

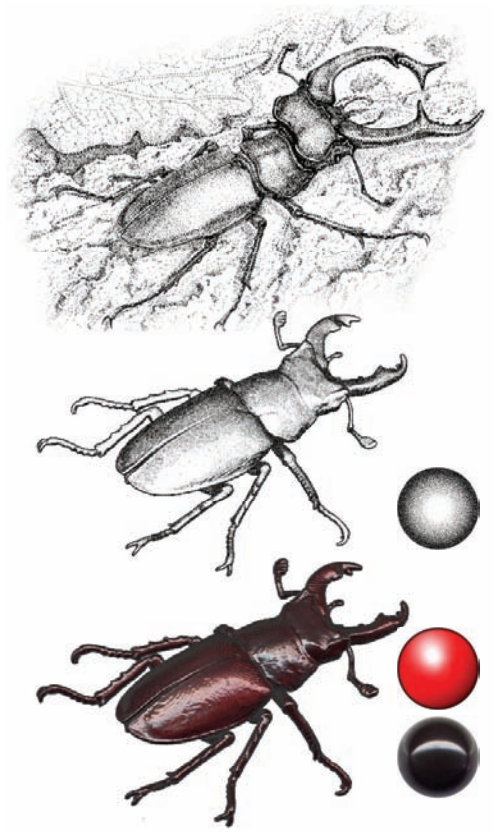


Figure 10: *Changing the theme of an image by replacing styles. Top: Drawing of a staghorn beetle by A. E. Brinev. Middle: Volume rendering of a staghorn beetle using a similar style. Bottom: Staghorn beetle rendered using a more realistic style.*

Acknowledgements

We thank the anonymous reviewers for their valuable comments. The work presented in this publication was carried out as part of the exvisation project (http://www.cg.tuwien.ac.at/research/vis/exvisation) supported by the Austrian Science Fund (FWF) grant no. P18322.

References

[CMH*01] CSÉBFALVI B., MROZ L., HAUSER H., KÖNIG A., GRÖLLER M. E.: Fast visualization of object contours by non-photorealistic volume rendering. *Computer Graphics Forum 20*, 3 (2001), 452–460.

[DC05] DONG F., CLAPWORTHY G. J.: Volumetric texture synthesis for non-photorealistic volume rendering of medical data. *The Visual Computer 21*, 7 (2005), 463–473.

[Deb98] DEBEVEC P.: Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with

global illumination and high dynamic range photography. In *Proceedings of ACM SIGGRAPH 1998* (1998), pp. 189–198.

[DHT*00] DEBEVEC P., HAWKINS T., TCHOU C., DUIKER H.-P., SAROKIN W., SAGAR M.: Acquiring the reflectance field of a human face. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 145–156.

[DWE02] DIEPSTRATEN J., WEISKOPF D., ERTL T.: Transparency in interactive technical illustrations. *Computer Graphics Forum 21*, 3 (2002).

[ER00] EBERT D. S., RHEINGANS P.: Volume illustration: non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization 2000* (2000), pp. 195–202.

[GGSC98] GOOCH A., GOOCH B., SHIRLEY P., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of ACM SIGGRAPH 1998* (1998), pp. 447–452.

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of IEEE Visualization 2003* (2003), pp. 301–308.

[HKG00] HLADUVKA J., KÖNIG A., GRÖLLER M. E.: Curvature-based transfer functions for direct volume rendering. In *Proceedings of the Spring Conference on Computer Graphics 2000* (2000), pp. 58–65.

[HMBG01] HAUSER H., MROZ L., BISCHI G.-I., GRÖLLER M. E.: Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 242–252.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum 24*, 3 (2005), 303–312.

[KKH01] KNISS J., KINDLMANN G., HANSEN C.: Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proceedings of IEEE Visualization 2001* (2001), pp. 255–262.

[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.

[KSW06] KRÜGER J., SCHNEIDER J., WESTERMANN R.: Clearview: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 941–948.

[KV06] KIM Y., VARSHNEY A.: Saliency-guided enhancement for volume visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 925–932.

[KWTM03] KINDLMANN G., WHITAKER R., TASDIZEN T., MÖLLER T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization 2003* (2003), pp. 513–520.

[LE05] LU A., EBERT D.: Example-based volume illustrations. In *Proceedings of IEEE Visualization 2005* (2005), pp. 655–662.

[LHV04] LEE C. H., HAO X., VARSHNEY A.: Light collages: Lighting design for effective visualization. In *Proceedings of IEEE Visualization 2004* (2004), pp. 281–288.

[LM04] LUM E. B., MA K.-L.: Lighting transfer functions using gradient aligned sampling. In *Proceedings of IEEE Visualization 2004* (2004), pp. 289–296.

[LME*02] LU A., MORRIS C. J., EBERT D. S., RHEINGANS P., HANSEN C.: Non-photorealistic volume rendering using stippling techniques. In *Proceedings of IEEE Visualization 2002* (2002), pp. 211–218.

[LMT*03] LU A., MORRIS C. J., TAYLOR J., EBERT D. S., HANSEN C., RHEINGANS P., HARTNER M.: Illustrative interactive stipple rendering. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (2003), 127–138.

[NSW02] NAGY Z., SCHNEIDER J., WESTERMANN R.: Interactive volume illustration. In *Proceedings of Vision, Modeling, and Visualization 2002* (2002), pp. 497–504.

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. In *Proceedings of ACM SIGGRAPH 2001* (2001), pp. 581–586.

[RE01] RHEINGANS P., EBERT D. S.: Volume illustration: Non-photorealistic rendering of volume models. *IEEE Transactions on Visualization and Computer Graphics 7*, 3 (2001), 253–264.

[SB99] SOUSA M. C., BUCHANAN J.: Computer-generated graphite pencil rendering of 3D polygonal models. *Computer Graphics Forum 18*, 3 (1999), 195–207.

[SBS05] SALAH Z., BARTZ D., STRASSER W.: Illustrative rendering of segmented anatomical data. In *Proceedings of SimVis 2005* (2005), pp. 175–184.

[SFWS03] SOUSA M. C., FOSTER K., WYVILL B., SAMAVATI F.: Precise ink drawing of 3D models. *Computer Graphics Forum 22*, 3 (2003), 369–379.

[SMGG01] SLOAN P.-P., MARTIN W., GOOCH A., GOOCH B.: The lit sphere: A model for capturing NPR shading from art. In *Proceedings of Graphics Interface 2001* (2001), pp. 143–150.

[SSI99] SATO I., SATO Y., IKEUCHI K.: Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 1–12.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. In *Proceedings of ACM SIGGRAPH 1990* (1990), pp. 197–206.

[TIP05] TIETJEN C., ISENBERG T., PREIM B.: Combining silhouettes, surface, and volume rendering for surgery education and planning. In *Proceedings of EuroVis 2005* (2005), pp. 303–310.

[VKG05] VIOLA I., KANITSAR A., GRÖLLER M. E.: Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 408–418.

[YC04] YUAN X., CHEN B.: Illustrating surfaces in volume. In *Proceedings of Joint IEEE/EG Symposium on Visualization 2004* (2004), pp. 9–16.

[ZDT04] ZHOU J., DÖRING A., TÖNNIES K. D.: Distance based enhancement for focal region based volume rendering. In *Proceedings of Bildverarbeitung für die Medizin 2004* (2004), pp. 199–203.