Volume Visualization
Chapter 6

Direct Volume Rendering

Direct Volume Rendering is based on techniques which were originally developed for simulating 3-D phenomena in the realm of photorealistic rendering:


Originally, Direct Volume Rendering was developed for rendering phenomena such as clouds or smoke where a surface description is not available. This was achieved by generalizing the ray-tracing method to 3-D volumetric objects.
Chapter 6

Application in visualization

In order to use this method for visualization, we assume that a volume is filled with a medium that has certain optical properties described by the scalar values. These optical properties can be directly derived from the scalar values if, for example, the scalar values describe the concentration of particles which absorb or emit light. This is particularly useful, if the scalar values describe the density within the volume. This is, for example, the case in medical data sets. In the volumetric image resulting from a CT scan, the scalar values describe the absorption of x-rays.
Chapter 6

Transfer Functions

Additional flexibility can be achieved by deriving the optical properties not directly from the scalar values but by using a so called transfer function instead. These transfer functions determine an optical property for every scalar value. This way, it is possible to assign more than one optical property to a single scalar value. For example, a scalar value can be assigned different colors by using different transfer functions for the three color channels red, green, and blue. In addition, we gain another degree of freedom in the visualization. By changing the transfer function, certain areas of scalar values can be emphasized, therefore adapting the visualization.
Chapter 6

Generating images

Most techniques trace a ray through the volume. There are basically two different approaches:

- **Object Order Methods**: these methods work on the cells/voxels of the volume as basic primitives and project these onto the image plane (forward mapping). The visualization is computed on a per cell/voxel basis.

- **Image Order Methods**: these techniques use pixels as basic primitives and cast rays through the pixel intersecting the image plane (reverse-mapping). The visualization is computed on a per pixel basis.

Also, combinations (hybrid methods) between these two can be used.
Chapter 6

Ray casting

Ray casting is an image order method. Starting with an image plane, rays are cast through the volume. This is done in the same way as for classic ray tracing/ray casting (see Computer Graphics II).

Parallel projection   Perspective projection
Chapter 6

Sampling

While ray casting only traces the ray until it hits the first object, the Direct Volume Rendering version has to follow the ray all the way through the entire volume. Since there are no surfaces in the volume that could be intersecting with the ray, samples are used along the ray. At each sampling location, the illumination model is used to determine a color value. The simplest version uses equidistant sampling.
Chapter 6

The choice of the sampling rate has to be done with special care in order to ensure that every cell is considered by at least one sample. The number of samples per cell depends on the interpolation and illumination model used.
Chapter 6

Illumination Models

We will now discuss different illumination models. Further information can be found in the following papers:


Chapter 6

Pure absorption

This “illumination model” assumes a clear medium filled with small particles that absorb light. Hereby we assume that all the particles are perfectly black, i.e. absorb all incoming light. In addition, the particles do not emit any light themselves. The scalar field describes the density of particles, i.e. it determines how many particles exist in a unit volume.

To make things simpler, we assume that the particles are all the same objects represented by a sphere with radius \( r \) and a projected surface of area \( A = \pi r^2 \).

Projection: circle with radius \( r \) and surface of area \( A = \pi r^2 \)
Considering a cylindrical section of thickness $\Delta s$ and surface area $E$, the volume of this section is $E\Delta s$ and contains $\rho E\Delta s$ particles, where $\rho$ is the particle density within the volume. Choosing $\Delta s$ small enough ensures that the probability of intersecting particles on the base area of the cylindrical sections is minimal and the portion of particles covering the base area is $NA = \rho AE\Delta s$. Then, the amount of light per unit area absorbed by the cylinder is $\rho AE\Delta s/E = \rho A\Delta s$. 
For $\Delta s$ approaching 0, we get the differential equation

$$\frac{dI}{ds} = - \rho(s) AI(s) = - \tau(s) I(s)$$

where $s$ is the parameter value in direction of the light and $I(s)$ the light intensity at distance $s$. 
Chapter 6

The solution for this differential equation is

\[ I(\tau) = I_0 \exp \left\{ -\int_0^\tau \tau(t) \right\} \]

The term

\[ T(\tau) = \exp \left\{ -\int_0^\tau \tau(t) \right\} \]

describes the “transparency” of the medium, which determines the amount of light that is not yet absorbed after traveling from 0 through \( s \).
Example

Image of a smoke cloud on top of a city simulated by an illumination model using pure absorption.
Chapter 6

Pure emission

If we assume that the particles are transparent (i.e. do not absorb any light) but emit light with intensity \( C \) per unit projected surface area, then the projected surface with area \( \rho A E \Delta s \) contributes an amount of light of \( C \rho A E \Delta s \) to the light passing through the base area \( E \). Hence, the amount of light per unit surface area is \( C \rho A \Delta s \) which results in the differential equation

\[
\frac{dI}{ds} = C(s) \rho(s) A = C(s) \tau(s) = g(s)
\]

The solution is

\[
I(s) = I_0 + \int_0^s g(t) \, dt
\]
Example

Image of a cloud created by an illumination model using pure emission.
In visualization it is common to use an illumination model that uses a combination of emission and absorption. This then assumes that the medium contains particles that absorb light as well as particles that emit light. Of course, this has to take into account that light which is emitted at a point along the ray is itself being absorbed on its way to the camera/observer. The differential equation for this model is

\[
\frac{dl}{ds} = g(s) - \tau(s)I(s)
\]

- Emitted light
- Absorbed light
Chapter 6

Assuming that the light enters the volume at \( s=0 \) and exits at \( s=D \) and then reaches the camera(observer) without any changes the solution for the previous differential equation is:

\[
I(D) = I_0 T(0) + \int_0^D g(s)T(s)\,ds \quad \text{where} \quad T(s) = \exp\left\{ - \int_s^D \tau(x)\,dx \right\}
\]
Chapter 6

Alternative specification of the absorption using opacity

Instead of describing the absorption using the absorption coefficient \( \tau(s) = \rho(s)A \), we can also use the opacity \( \alpha \). The opacity describes the amount of light that is absorbed for a certain distance \( l \), i.e.:

\[
\alpha = 1 - T(s) = 1 - \exp \left( - \int_0^l \tau(t) \, dt \right)
\]
Chapter 6

It makes sense to specify the opacity per unit length. Instead of specifying the absorption coefficient for every point within the volume, a value is given which describes the amount of light that would be absorbed after traveling a distance of length 1 through the medium. The relation between the opacity per unit length and the absorption coefficient is as follows:

\[ \alpha(s) = 1 - \exp(-\tau(s)) \]

Unfortunately, some papers use the term opacity instead of absorption coefficient which can be confusing.
Chapter 6
Colored emission

We can further improve the absorption and emission model by assuming that the particles in the medium emit light at different parts of the color spectrum. This can be achieved by splitting $C(s)$ and $g(s)$ into different components (for example for red, green, and blue) and specifying a transfer function which assigns different emissions to each component. This means that the transfer function assigns each scalar value four optical properties: an opacity and an amount of emission for each fundamental color.

This does not change the way the transparency $T(s)$ is computed; the integral $I(D)$ has to be computed for each component separately.
Chapter 6

Specification of transfer functions

There are different ways to specify a transfer function. Basically, we have to define a function $F : \mathbb{R} \rightarrow \mathbb{R}^3$. Most common are tables with a pre-defined number of entries, for example 256. The range of values of the scalar field is then mapped onto the interval $[0, \#values-1]$ and a specific color value is chosen by rounding off the scalar value of the data set. Also useful are piecewise linear transfer functions. A piecewise linear transfer function can allow us to split the previous integral and compute them analytically.

It is also possible to realize the transfer function as a spline.
Chapter 6

Transfer Functions

Transfer function editors often utilize the histogram on top of which the transfer function for the opacity and color values are designed. In this example, colors (bottom) are based on intensity ranges.
Chapter 6

Typically, transfer functions are specified for the fundamental colors red green and blue and opacity:

Image courtesy of Alexandru Telea
Brain of a fruit fly, imaged using laser microscopy

[Data courtesy of Genetikinstitut University of Wuerzburg; software Amira, Konrad-Zuse-Zentrum, Berlin]
Chapter 6
Solving the integrals numerically

The easiest approximation of an integral is to use the Riemann sum:

\[
\int_0^D h(x) \, dx \approx \sum_{i=1}^n h(x_i) \Delta x
\]

If we use ray casting with equidistant sampling then the position of the \( i \)-th sample with respect to the ray is \( x_i = i \Delta x \). Then, we can approximate the integral \( T(s) \):

\[
\exp \left\{ -\int_0^D \tau(x) \, dx \right\} \approx \exp \left\{ -\sum_{i=1}^n \tau(i \Delta x) \Delta x \right\} = \prod_{i=1}^n \exp \left( \tau(i \Delta x) \Delta x \right) = \prod_{i=1}^n t_i
\]

This approximation subdivides the ray within the interval \([0,D]\) in \( n \) segments of equal length and assumes constant absorption within each segment. Here, we can interpret \( t_i = \exp(-\tau(i \Delta x) \Delta x) \) as transparency of the \( i \)-th segment.
Chapter 6

If we use the same positions $i \Delta x$ for solving the entire integral, i.e. emission and absorption, therefore using the exact same ray segments, we can compute the $i$-th sample for the emission $g_i = g(i \Delta x)$. The transparency along the ray up to this sample, i.e. the part of the light that gets to the observer, then is:

$$ T (i \Delta x) = \exp \left\{ - \int_{i \Delta x}^D \tau (x) \, dx \right\} \approx \prod_{j = i + 1}^n t_j $$

Hence, we can approximate the amount of light that gets to the observer:

$$ \sum_{i=1}^n g_i \prod_{j = i + 1}^n t_j $$
Chapter 6

The approximation of the entire integral is then (if we assume that $g_0 = I_0$):

$$I(D) \approx I_0 \prod_{i=0}^{n} t_i + \sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j$$

$$= g_n + t_n (g_{n-1} + t_{n-1} (g_{n-2} + t_{n-2} (g_{n-3} + \cdots + (g_1 + t_1 I_0))))$$

$$I(D) \approx \sum_{i=0}^{n} g_i \prod_{j=i+1}^{n} t_j$$
Chapter 6

Since $\tau(s) = -\ln(1 - \alpha(s))$, we can compute $t_i$ using the opacity at location $s$:

$$t_i = \exp \left( \ln \left( 1 - \alpha \left( i \Delta x \right) \right) \Delta x \right) = \exp \left( \ln \left( 1 - \alpha \left( i \Delta x \right) \right) \right)^{\Delta x} = (1 - \alpha \left( i \Delta x \right))^{\Delta x}$$

The opacity of the $i$-th ray segment $\alpha_i$ then is

$$\alpha_i = 1 - t_i = 1 - \left( 1 - \alpha \left( i \Delta x \right) \right)^{\Delta x}$$

If we now use $\tilde{E}(s) = C(s)\alpha(s)$ instead of $g(s) = C(s)\tau(s)$ we get the following approximation for our integral:

$$I(D) \approx \sum_{i=0}^{n} \tilde{E}_i \prod_{i=i+1}^{n} (1 - \alpha_j)$$

with

$$\tilde{E}_i = C \left( i \Delta x \right) \alpha \left( i \Delta x \right)$$
Chapter 6

If we now add up the ray segments starting at the observer instead of the light source, then we get:

\[ I(D) \approx \sum_{k=0}^{n} \tilde{E}_{k} \prod_{j=1}^{k} (1 - \alpha_{j}) \]

Where \( \tilde{E}_{n} \) is the brightness of the background

These equations can also be written recursively:

\[ \tilde{I}_{n} = \tilde{E}_{n} \]

\[ \tilde{I}_{k} = \tilde{E}_{k} + (1 - \alpha_{k}) \tilde{I}_{k+1} \]

Or as:

\[ I_{0} = E_{0}; \gamma_{0} = \alpha_{0} \]

\[ I_{k} = I_{k-1} + (1 - \gamma_{k-1}) E_{k} \]

In Back-to-front compositing

\[ \gamma_{k} = \gamma_{k-1} + (1 - \gamma_{k-1}) \alpha_{k} \]

In Front-to-back compositing
Chapter 6

Compositing

In its simplest form, *compositing* describes the overlaying of images. In order for this to make sense, we specify an opacity value for each pixel in addition to the color values. The opacity is somewhere between zero and one. If the opacity is zero, then the pixel of the image behind is completely visible. In case of a value of one, the image behind is entirely occluded.

The images are added on a per-pixel basis. If $F$ and $B$ are color values of two pixels, i.e. three-dimensional vectors (R, G, and B component) and $\alpha$ the opacity of $F$, then the color value of the combination $F \oplus B$ of the two images can be computed as $F \oplus B = (1-\alpha) B + \alpha F = B + \alpha (F-B)$. 
Chapter 6

If we go further and add another image with color value $G$ and opacity $\beta$ we would get the following result:

$$G \oplus (F \oplus B) = (1 - \beta) F \oplus B + \beta G$$

$$= (1 - \beta) ((1 - \alpha) B + \alpha F) + \beta G.$$
Chapter 6

If we want to be able to combine the images in arbitrary order, i.e. combine F and G first in our example, and then overlay the result on top of B, then would we calculate \((G \oplus F) \oplus B\). Therefore, we also need the opacity \(\gamma\) of the combination of the two images. Based on the condition that the combination of images is supposed to be associative, i.e. \(G \oplus (F \oplus B) = (G \oplus F) \oplus B\), and therefore independent of the order we evaluate, we can derive:

\[
\gamma = \alpha + \beta - \alpha\beta
\]

This also implies that

\[
1 - \gamma = (1 - \alpha)(1 - \beta)
\]
Chapter 6

In order to see that this is correct, we can look at the transparency of the images. A pixel with opacity $\alpha$ has the transparency $(1 - \alpha)$, i.e. a pixel with opacity $\alpha$ lets a fraction of $(1 - \alpha)$ of the underlying image pass through. If we now put an image with transparency of $(1 - \beta)$ on top of an image with transparency $(1 - \alpha)$, the combined transparency is $(1 - \alpha) \cdot (1 - \beta)$ which is exactly the result of the previous slide.
Chapter 6

Associated color

When looking at the previous formulae we can notice that the computation of the opacity is different from the computation of the color values. If we now use the so-called associated color values $\tilde{F} = \alpha F$, $\tilde{G} = \beta G$, and $\tilde{H} = \gamma H$ instead of the color values $F$, $G$, and $H$ and the opacities $\alpha$, $\beta$, and $\gamma$ separately we can treat all components in the same way:

$$H = (1 - \beta ) \tilde{F} + \tilde{G}$$

$$\gamma = (1 - \beta ) \alpha + \beta$$
Chapter 6

This means that the with associated colors the color components are already multiplied with the opacity values. This is equivalent to compositing with a black background and results in an image with correct color components. Images where the color component is not multiplied with the opacity values can only be shown correctly in front of a black background.
Chapter 6

Compositing and ray casting

If we convert transparencies into opacities, then compositing is equivalent to the combination of the ray segments used for the ray casting. Since compositing is nowadays supported by the graphics hardware, this can be used to accelerate the computation of the ray casting algorithm.
Chapter 6

Interpolation of color values

When interpolating color values with associated opacity values, the associated colors need to be used for the interpolation. Then it is ensured that we can apply the operations interpolation and combination with the background in arbitrary order.

This is important for volume rendering as well. Often times, colors are specified instead of scalar values at the cell vertices. Instead of computing a scalar value and then transforming into color/opacity, this should be done only at the cell vertices. Inside the cell, the color values and opacities are then interpolated. It is important here to use the associated colors for the interpolation. Otherwise, artifacts can occur at the transitions to media with an opacity of zero.
Chapter 6

Chapter 6

Sabella illumination model

Sabella [Paolo Sabella, “A Rendering Algorithm for Visualizing 3D Scalar Fields”. ACM Computer Graphics, Vol. 22, No.4 (SIGGRAPH 1988 Proceedings, 51-58] was one of the first authors who used ray tracing/ray casting for visualizing scalar data sets. In this paper, the illumination model uses the so called “varying density emitters”. This illumination model simulates light emitting particles embedded in a transparent gel. Even though the derivation of the equations differs from the absorption and emission model and the used model is governed by different parameters, the results are very similar (as well as the equations).
Chapter 6

Sabella, however, uses achromatic white light for the illumination model so that every scalar value has only one transparency value. The color is used to represent additional information about the scalar field along the ray.

For this, Sabella uses the HSV color model and determines the color components as follows:

- The brightness (V component) is equal to the intensity after evaluating the illumination model
- The hue (H component) is chosen according to the maximum along the ray (see maximum intensity projection)
- The saturation (S component) is set according to the distance to that maximum. This corresponds to the effect of fog: the further the object, the less colorized it appears.
Chapter 6

The maximum (upper left), distance to the maximum (upper center), the intensity (upper right), and the combination using the HSV color model.
Chapter 6

Levoy illumination model

In addition to the pure scalar values other properties can be taken into consideration by the illumination model. Levoy [Marc E. Levoy, “Display of Surfaces from Volume Data”, IEEE Computer Graphics and Applications, Vol.8, No. 3, 1988, 29-37] uses image gradients as well. This gradient is used in two different ways. Based on the Phong illumination model, the gradient is used to create the effect of surfaces within the volume. Additionally, the opacity is influenced by the gradient.

Levoy separates the determination of color and opacities based on the scalar values. A function $c(x_i)$ calculates colors while another function $\alpha(x_i)$ computes the opacities.

Levoy describes this using a diagram for the “volume rendering pipeline”, which most algorithms are based on today. The diagram in the original paper, however, contains an error that suggests that the color values and opacities can be interpolated independently.
Chapter 6

Traditional Volume Rendering Pipeline

![Diagram of the traditional volume rendering pipeline]

Corrected version of the “volume rendering pipeline” according to Levoy. This version takes into account that associated colors have to be used for interpolation of color values.

Levoy model – shading

Levoy applies the transfer function to the original data values. In order to determine a color value for the sample on the ray, the colors are interpolated. At every sample point, a color value is calculated according to the Phong illumination model:

\[
c_\lambda(x_i) = c_p \cdot \lambda k_{a,\lambda} + \frac{c_p}{M} + M \cdot M \cdot d(x_i)
\]

where:
- \( c_\lambda(x_i) \) is the color component at \( x_i \)
- \( c_p \) is the intensity of the light source
- \( k_{a,\lambda} \) is the ambient reflection coefficient
- \( M \) is the distance model
- \( d(x_i) \) is the distance to the observer
- \( k_{d,\lambda} \) is the diffuse reflection coefficient
- \( k_{s,\lambda} \) is the specular reflection coefficient
- \( N(x_i) \) is the "surface" normal (normalized gradient)
- \( H \) is the normalized vector pointing maximal reflection
- \( \lambda \) is the material property

\( c_p \cdot \lambda k_{a,\lambda} \) is the color component due to ambient light.

\( \frac{c_p}{M} \) is the color component due to the distance to the observer.

\( M \cdot M \cdot d(x_i) \) is the color component due to the distance model.
Chapter 6

When implementing his algorithm we have to consider the same special cases as with the Phong model for surface rendering itself. For example:

If $N(x_i) \cdot L < 0$ we have to set $N(x_i) \cdot L = 0$. 
Chapter 6

Levoy model – classification

The mapping of opacity values according to the scalar values is chosen by the user based on significant surfaces within the data. Levoy calls this step classification. Two options are presented in Levoy’s paper:

- Iso-surfaces: the transfer function for the opacity is chosen in such a way that a specific iso-value appears opaque.
- Transition between regions: the data set is considered a set of regions with homogeneous density and the transfer function is chosen in such a way that the transitional surfaces between two regions appear transparent.
Chapter 6

Levoy model – iso-surface representation

Setting the opacity to $\alpha_v$ for a specific iso-value $f_v$ and all other opacities to zero results in aliasing artifacts if only one ray is cast for every pixel. Hence, a transfer function should be chosen that smoothly blends over between the $f_v$ and its surrounding. The best results are achieved if the thickness of the layer around the iso-surface is more or less equal throughout the entire volume. Therefore, the length of the image gradient is used as well so that the slope of the transfer function is reciprocally proportional to the image gradient:
\[
\alpha(x_t) = \alpha_v \begin{cases} 
1 & \text{if } |\nabla f(x_t)| = 0 \text{ and } f(x_t) = f_v \\
1 - \frac{1}{r} \left| f_v - f(x_t) \right| \frac{1}{|\nabla f(x_t)|} & \text{if } |\nabla f(x_t)| > 0 \text{ and } f(x_t) - r |\nabla f(x_t)| \leq f_v \leq f(x_t) + r |\nabla f(x_t)| \\
0 & \text{otherwise}
\end{cases}
\]
Chapter 6

opacity $\alpha(x_1)$

gradient magnitude $|\nabla f(x_1)|$

acquired value $f(x_1)$
Chapter 6

“Isosurfaces” in the protein cyochrome B5
Chapter 6

Several iso-surfaces

It is also possible to display more than one iso-surface using this technique. By applying the classification step more than once for every iso-value and then combining the resulting transfer functions a transfer function can be generated so that several iso-surfaces are visualized.
Chapter 6

Transition between regions

The visualization of plain iso-surfaces can be problematic when applied to medical data sets. Assume an anatomical data set with two different types of tissue A and B with scalar values $f_{vA}$ and $f_{vB}$ where $f_{vA} < f_{vB}$. At the transition from A to B, the data set will contain values $f(x_i)$ with $f_{vA} < f(x_i) < f_{vB}$. Hence, there is no value larger than $f_{vA}$ that ensures that thin regions of tissue B are not displayed. On the other hand, a value close to $f_{vA}$ can generate a “noisy” image.
Levoy solves this problem by making the following assumptions:

- There exist an arbitrary number of types of tissue with CT values (scalar values) close together.
- There are only transitions between one type of tissue and other, different types of tissue.
- There is an order of CT values $f_{vn}$, $n=1,\ldots,N$, such that $f_{vm} < f_{vm+1}$, $m=1,\ldots,N-1$ and no tissue of CT value $f_{vn1}$ touches another tissue of CT value $f_{vn2}$ if $|n1-n2| > 1$.

These assumptions are true for many medical data sets. Levoy picks an opacity $\alpha_{vn}$ for every CT value $f_{vn}$. For every value in between those $f_{vn}$, the opacities are interpolated linearly. This results in thin tissue areas appearing as a light glowing.
Chapter 6

The types of tissue are displayed as an overlay of half-transparent regions. In order to emphasize the transitions between these regions, the opacity is chosen small within these regions, while the transitional areas are assigned a large opacity. This can be achieved by scaling the opacities using the gradient:

\[
\alpha(x_i) = |\nabla f(x_i)| \begin{cases} 
\alpha_{n+1} \frac{f(x_i) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}} + \\
\alpha_n \frac{f_{v_{n+1}} - f(x_i)}{f_{v_{n+1}} - f_{v_n}} & \text{if } f_{v_n} \leq f(x_i) \leq f_{v_{n+1}} \\
0 & \text{otherwise}
\end{cases}
\]
Chapter 6

Department of Computer Science and Engineering
Chapter 6

Department of Computer Science and Engineering
Chapter 7

Volumetric Shading

Shading can provide additional cues that can significantly improve the quality of volume renderings. Shading can be easily combined with the volume illumination integral. Instead of directly using colors, we can use an illumination function instead:

\[ I(t) = c_{amb} + c_{diff}(t) \max (-L \cdot n(t), 0) + c_{spec}(t) \max (-r \cdot v, 0)^{\alpha} \]

This is nothing more than the application of the Phong illumination model. To approximate normal vector \( n \), we can use the image gradient.
Chapter 7

Volumetric lighting

No lighting  Diffuse lighting  Specular lighting

Images courtesy of Alexandru Telea
Chapter 6

Volume illustration

The previously introduced illumination models were not necessarily physically correct, but motivated by physical phenomena. It is also possible to achieve meaningful images by not following any physical model or modifying them in order to add additional information to the visualization. This was done, for example, in the Sabella model by using the HSV color model in order to visualize the maximal value along the ray and its distance to that value.
Chapter 6

David Ebert and Penny Rheingans [David Ebert, Penny Rheingans, “Volume Illustration: Non Photorealistic Rendering of Volume Models”, Proceedings of IEEE Visualization 2000] developed methods for that. The traditional volume rendering pipeline is modified and the colors and opacities are chosen based on an illumination model. The basis for this illumination model is the transfer function:

\[ \alpha (v) = (k_{os} v)^{k_{oc}} \]

Control of maximal opacity

“contrast”
Chapter 6

Volume Illustration Rendering Pipeline

- Volume values $f_i(x_i)$
- Transfer function
- Volume Rendering
  - Volume Illustration
    - color modification
  - Volume Illustration
    - opacity modification
- Final volume sample $[c_\lambda(x_i), \alpha(x_i)]$
- Image pixels $C_\lambda(u_i)$
Chapter 6

Volume illustration – feature enhancement

Ebert and Rheingans enhance the “features” present in the data set by using the gradient for modifying the opacity of the basic transfer function:

- The transition between regions is emphasized similar to Levoy’s approach by scaling the opacities using the length of the gradient.
- Silhouettes are amplified by projecting the normalized gradient onto the unit vector in direction pointing to the viewer/camera. The length of this vector is somewhere between zero and one. By subtracting the resulting value from one and using this for scaling the opacity, transitions between regions which are perpendicular to the view direction are enhanced.
Image based on original illumination model
Chapter 6

Transitions between regions are enhanced

Image courtesy of David Ebert
Chapter 6

Enhanced transitions between regions and silhouettes

Image courtesy of David Ebert
Volume-illustration – depth and orientation cues

Ebert and Rheingans also modify the color values in order to convey more information to the user. For example, the blue component is amplified with increasing distance to the viewer/camera. This way, a better depth perception is achieved. More methods can be found in the original paper. The paper and more examples can be found using the following URL:

http://www.csee.umbc.edu/~ebert/npr/
Chapter 6

Image computation using direct volume rendering

Ray-casting is an image-order technique for generating an image based on a scalar data set. Rays are cast through the volume and samples at specific locations along the rays are evaluated. The problem usually is to find an appropriate distance between these samples.

A small sample distance increases the computational effort significantly, since the number of samples increases for every ray. If the sampling rate is too small details and features may be missed or aliasing artifacts occur.

Often, the sample distance is defined by the user. For regular grids, this distance can be given in relation to the size of the cells. This makes this parameter “more independent” from the given data set.
Image of a vase given as a volumetric data set. The scalar values are given on a regular grid with distance 1.0. The image on the left shows artifacts due to the low sampling rate. The image on the right takes ten times as long to compute.

[Image courtesy of Will Schroeder, Ken Martin, Bill Lorensen, “The Visualization Toolkit”]
Chapter 6

The sample distance can be given in dependence of the cell size as a multiple of the diameter of the inscribed sphere of a cell. This inscribed sphere is the largest sphere which completely fits into a cell. For quad-shaped cells, i.e. not necessarily cubed, the smaller dimension should be used as diameter.
Chapter 6

Often times it makes sense to sub-divide the ray, such that each segment is entirely contained in exactly one cell. For each of these segments, the desired values (intensity, maximum, etc) are computed and combined according to the segments.

If the interpolation is known, it is often possible to compute the maximum of the ray segment. For constant interpolation, the integral can be calculated analytically, i.e. without approximation (the assumption of constant opacity/emission is true for every cell for approximating the integral). When using other interpolations, it may make sense to use a sub-division that makes it easier to approximate the integral if the function that is to be integrated is given as a closed form.

By sampling the ray on a per-voxel basis, the ray-casting algorithm turns into a hybrid method. Even thought the image is computed for each pixel individually (image-order), the sampling of the ray is done on a per-voxel basis (object-order).
Chapter 6

Left: the ray is sampled equidistantly; right: the series of all voxels is computed along the ray. This enumeration can be determined by using a more general version of the Bresenham algorithm.

[image courtesy of Will Schroeder, Ken Martin, Bill Lorenson, “The Visualization Toolkit”]
The result of the modified Bresenham algorithm is a series \( v_1, v_2, \ldots, v_n \) of cells. This series is called 6-, 18-, or 26-connected depending on two subsequent cells sharing just surfaces, surfaces or edges, or edges or vertices. A 6-connected series results in a larger computational time, while details might be lost when using a 26-connected series.

[Image courtesy of Will Schroeder, Ken Martin, Bill Lorenson, “The Visualization Toolkit”]
When using orthogonal projection, a “template” can be pre-computed for a ray. The volume is then sampled according to this template displaced according to the current ray. However, this has to be done from the base plane of the volume, since otherwise not all cells are covered (left). When computing the image this way, it has to be undistorted since it was computed from the base plane of the volume.

[Image courtesy of Will Schroeder, Ken Martin, Bill Lorenson, “The Visualization Toolkit”]
Cell projection

Cell projection is an object-order technique for visualizing images using direct volume rendering. Cell projection was originally developed for displaying data defined on unstructured grids. Since it is very costly to determine the cell for a given location within an unstructured grid it is not efficient to evaluate samples along a ray since for each sample the cell that contains the location for that sample has to be determined. Also, the generalized Bresenham algorithm works only with regular grids so that it cannot be used for determining a series of cells in this case.
Chapter 6

Cell projection solves this problem by computing the resulting image on a per-cell basis. For each cell, the boundary polygons are rastered. At the same time, ray segments are generated and then combined with the already existing ray segments.
Chapter 6

Computation of the ray segments

In order to compute the ray segments associated with a cell, the boundary surfaces of the cell are divided into two groups. The group “front-facing” contains all surfaces which are visible by the viewer, i.e. the front side of the cell. Since virtually all cell types are convex, these are exactly the surfaces whose normals are facing towards the observer. The surfaces on the backside of the cells are associated with the “back-facing” group, i.e. those surfaces with normals pointing away from the viewer.
Chapter 6

Now, the boundary surfaces of the front- and backside of the cells are rastered into two different buffers (if possible using the graphics hardware). I.e., the corresponding polygons are mapped onto the image plane and for each pixel a depth value and an interpolated value (color or scalar value) are computed (see Computer Graphics I). For each pixel that is covered by a cell using the current projection, the two buffers are evaluated and ray segments are generated.
Chapter 6

Here, the depth values of the buffer that resulted from projecting the front faces determines the point of entrance $t_{in}$ and the depth value of the buffer that resulted from projecting the back faces gives us the point of exit $t_{out}$. Based on these parameters, the limits of the ray segment and the interpolated color or scalar values can now be computed. In case of an illumination model, this means that the opacities and color intensities can now be calculated.
Chapter 6

One-buffer solution

It is also possible to use a two-step approach, therefore avoiding to compute two buffers. In the first step, the back faces are rastered. In the second step, the front faces are rastered. Instead of writing the results to a buffer, the value is read from the buffer resulting from the first step and the resulting ray segment is generated directly for every pixel.
Image plane
Chapter 6

Image plane

- Front facing
- Back facing
Chapter 6

Interpolate depth and value

Image plane

Department of Computer Science and Engineering
Chapter 6

Image plane
Chapter 6

Image plane
Chapter 6

Image plane
Chapter 6

Image plane
Chapter 6

Image plane
Chapter 6

Image plane
Chapter 6

Image plane
Generating ray segments within the cells

For every pixel that is covered by a cell, a ray segment has to be computed within the cell. Let \( t_i \) and \( t_o \) be the point of entrance and exit, respectively. In case of tetrahedral cells, linear interpolation can be used. Hence, the scalar values are interpolated linearly as well. Along the ray within a cell, we can interpolate the scalar values linearly resulting in correct interpolation values. For example, an absorption and emission illumination model results in the following integrals:

\[
I_{\text{cell}} = \int_{t_i}^{t_o} g(s) T_{\text{cell}}(s) \, ds \quad \text{where} \quad T_{\text{cell}}(s) = \exp \left( - \int_{s}^{t_o} \tau(x) \, dx \right)
\]
Chapter 6

On certain conditions, this can be done analytically (see ray casting). For quad-shaped or other non-linear cells, the interpolation of the two scalar values on the surfaces does not result in correct values. In that case, we have to interpolate within the cell using the cell’s local coordinate system.
Chapter 6

Combining the ray segments

There are two approaches for combining the resulting ray segments: either the cells are sorted before projecting them or the resulting ray segments are sorted.

If the cells are sorted before the projection in such a way that each cell is located completely in front of an already processed cell, then the rays are generated in the correct order and can be combined using a simple back-to-front approach. Using graphics hardware, this can be done very efficiently.
Chapter 6

It is also possible to process the cells “front-to-back”. Then, every cell has to be located completely behind all processed cells. For this approach, we also need to compute and store the opacities of the so far combined ray segments (since we add at the end of the ray). Due to the additional effort necessary (separate $\alpha$-buffer required), this approach is chosen only rarely.
Chapter 6

Sorting the cells

Max [Nelson L. Max, “Sorting for Polyhedron Compositing”, Focus on Scientific Visualization, 259-268, Springer, 1993] describes sorting methods that can be used depending on the requirements. If the grid is convex, a topological sorting algorithm can be used as described by Knuth [Donald E. Knuth, “The Art of Computer Programming”, Volume 1: Fundamental Algorithms (second edition), Addison Wesley, 1973]. The grid is transformed into a directed graph:

- For every cell A there exist a node in the graph
- For every face F between two cells A and B there exist a directed edge in the graph connecting the corresponding nodes
- The edge associated to F is pointing from A to B if the view point V is on the same side of F as A, i.e. B has to be processed before A.
Chapter 6

The orientation of these edges can be determined based on the sign of the scalar product between the surface normal and the vector pointing from a vertex of the surface to the viewer. The nodes of the graph (and therefore the cells in the grid) are then sorted by removing nodes that do not have any incoming edges and adding this to the front of a separate list. All edges connecting to this node are removed by this step as well. After all nodes are removed from the graph, the list contains all grid cells in the correct order suitable for back-to-front compositing.
Chapter 6

The sorting gets easier if cell projection is applied to regular grids. For orthographic projection, the cells can be sorted using three nested loops. In each of these loops, only the order of the cells (increasing or decreasing) has to be set according to the corresponding component of the view direction.
Chapter 6

\[ v_y > 0 \implies \text{decreasing} \]

\[ v_x < 0 \implies \text{increasing} \]
Chapter 6

When using perspective projection, the view point is computed with respect to the grid’s coordinate system. Then, the volume is divided into several blocks separated by layers containing the view point. Each of these blocks can now be sorted using three nested loops just like in the orthogonal case. Only those blocks in view direction are considered.
The blocks themselves are traversed starting at the outside and approaching the center.
Chapter 6

Alternative: Sorting of the ray segments

As an alternative approach, the cells can also be processed without sorting. Then, the ray segments need to be sorted instead. Since the ray segments are generated in an arbitrary order, it is possible that a ray segment that is just generated cannot be combined with already existing ray segments. In order to avoid this, a queue is used for each pixel where new ray segments are inserted with regard to the correct order. At the same time, it is checked if the newly generated ray segment can be combined with ray segments already present in the queue or if it connects two of those ray segments.
Chapter 6

For example, in the figure below the ray segments 1 and 2 can only be combined after ray segment 3 is added to the queue.
Splatting

Splatting is an object-order method for computing an image based on scalar data. Here, we do not cast rays through the volume but construct a visualization directly based on the volume data.

Considering a 2-D image, we have values given at discrete locations. These can be interpreted as discrete samples of a 2-D signal. When displaying the image using a CRT monitor, the discrete samples are turned into a continuous signal. To describe this mathematically, we need a reconstruction kernel $f_2$ and the compute the folding between the signal $f_1$ and the reconstruction kernel:

\[
(f_1 * f_2)(x) = \int_{-\infty}^{+\infty} f_1(t) f_2(x - t) \, dt
\]
Chapter 6

For a discrete signal, we can illustrate this as follows: for every discrete value, the kernel is replicated and displaced in such a way that the origin is located at the sample location. Then the kernel is scaled according to the signal. The resulting value at a specific location of the signal then is equal to the sum of all the scaled folding kernels.

Chapter 6

Splatting [Lee Westover, “Footprint Evaluation for Volume Rendering, ACM Computer Graphics (SIGGRAPH 1990 Proceedings), Vol. 24, No. 4, 367-376, 1990] considers direct volume rendering a reconstruction problem. If a 3-D reconstruction kernel $h_V(x,y,z)$ is given, then the “signal” defined by the volume can be reconstructed as follows:

$$\text{signal } D (x, y, z) = \sum_{D \in \text{Vol}} h_V(x - D_x, y - D_y, z - D_z) f(D)$$

$D$ traverses the domain of the reconstruction kernel. The contribution of a data value $D$ at a specific location within the volume is:

$$\text{contribution } on_D (x, y, z) = h_V(x - D_x, y - D_y, z - D_z) f(D)$$
Chapter 6

If we cast a ray through the volume and project orthogonally onto a position defined by $x$ and $y$ then the contribution of a data value $D$ is described by the following integral:

$$\text{contributi} \quad on \quad p \quad ( \quad x \quad , \quad y \quad ) \quad = \quad f \quad ( \quad D \quad ) \int \quad h \quad v \quad ( \quad x \quad - \quad D \quad x \quad , \quad y \quad - \quad D \quad y \quad , \quad \omega \quad ) \quad d \quad \omega \quad$$
Density distribution of a three-dimensional Gauss distribution

Usually, reconstruction kernels are spherical. For orthogonal projections, we get circles with identical density distribution from all viewing directions.

Image courtesy of Richard S. Gallagher, “Computer Visualization: Graphics Techniques for Scientific Engineering and Analysis"
Chapter 6

The contribution of a single scalar value for the integral of a pixel is independent of the original scalar value. In addition, we can assume a canonical contribution function which is then shifted according to the location of the scalar value:

\[
footprint_d (x, y) = \int h_v (x, y, \omega) d\omega
\]

This “footprint” only has to be computed once at a sufficiently high resolution (which can, for example, be stored on disk). In order to compute an image from a volumetric data set, the “footprint” is scaled according to the view point. Then, the “footprint” is shifted to match with the location of the scalar value. The result is combined by a back-to-front compositing.
Example of a “footprint” function given as a raster image. This can be pre-computed and stored on disk since it only depends on the reconstruction kernel.

Image courtesy of Richard S. Gallagher, “Computer Visualization: Graphics Techniques for Scientific Engineering and Analysis”
Splatting is also possible for perspective projection. Here, the “footprint” function has to be transformed independently for every scalar value, since the projection of the kernel depends on the location within the volume.

Image courtesy of Richard S. Gallagher, “Computer Visualization: Graphics Techniques for Scientific Engineering and Analysis”
Chapter 6

Shear-Warp

The Shear-Warp method [Phillipe Lacroute, Marc Levoy, “Fast Volume Rendering Using A Shear-Warp Factorization of the Viewing Transformation”, SIGGRAPH 1994 Proceedings, 451-458] is a hybrid method. The orthogonal projection is divided into a shear and a warp. The planes of the volume are sheared in such a way that the view rays are parallel to one of the axis of the volume and can therefore be cast accordingly (object-order). The resulting image then needs to be warped to get the correct visualization (image-order).
Chapter 6

Shear-Warp - principle
Chapter 6

Direct volume rendering using 2-D textures

It is possible to simulate ray casting for regular grids with quad-shaped cells using the graphics hardware. The graphics hardware takes over the interpolation within the grid cells, combining the ray segments, and – if possible – mapping of the scalar values to colors. Here, we can exploit that direct volume rendering – as well as texture mapping – uses bi-linear and tri-linear interpolation.

Combining the ray segments is equivalent to compositing (also known as $\alpha$-blending) and is supported by the graphics hardware directly.

For hardware assisted direct volume rendering, most often the orthogonal projection is used. The rays traverse through the volume in a parallel fashion. First, we assume that the view direction is parallel to one of the coordinate axis.
If we look at the equidistant samples along a ray we can notice that all samples are located on planes orthogonal to the view direction. We now do not process the rays sequentially, i.e. compute all samples along a ray entirely first before continuing with the next one.

Instead, all rays are processed in parallel by evaluating all samples that are located at the same distance from the viewer before continuing with the next ones. This is equivalent to ray casting using planes orthogonal to the view direction and compositing of the resulting images.

Chapter 6

Within every plane, the tri-linear interpolation is equivalent to a bi-linear interpolation which is realized using the 2-D texture graphics hardware. The distance between the samples is chosen in such a way that the planes the samples are located at are identical with those planes of the data set where scalar values are available. For every scalar value, the illumination model is evaluated at the vertices of the grid and the results (including the opacity values) stored in a 2-D texture. The complete image then results from activating α-blending (compositing) and placing the planes at the correct distances with the according textures containing the color values.
Chapter 6

For the view directions parallel to the coordinate axes, we need three sets of 2-D textures with the sampled color values. The order of the 2-D textures in which they are mapped onto the planes depends on the view direction, i.e. if we look at the data from a positive or negative direction with respect to the coordinate axis.
Chapter 6

Polygon placed at a certain distance from the viewer resembling one of the planes

Pre-computed values derived from the illumination model stored in a texture

Arbitrary view directions (which are not parallel to any of the coordinate axes) cannot be achieved correctly using 2-D textures. Usually, the coordinate axis is picked that is closest to the view direction. Then, the planes are placed along that direction. This, however, results in visible artifacts.

Chapter 6

Direct volume rendering using 3-D textures

For a correct visualization, the planes on which the samples are located need to be orthogonal to the view direction. This is not possible to achieve with 2-D textures since a tri-linear interpolation for arbitrary planes cannot be reduced to a bi-linear interpolation.

Chapter 6

Current off-the-shelf graphics cards support 3-D textures. For a 3-D texture, color values and opacities are specified on a regular grid. When displaying geometry represented by polygons, a location within the volume can be specified for every vertex of the polygons. For displaying the polygon, a location within the polygon is interpolated based on the vertices while the color value is determined using the 3-D texture. Here, tri-linear interpolation is used by the hardware.

Since the tri-linear interpolation within the volume is computed entirely by the graphics hardware, the planes can be placed at arbitrary angles. Instead of several sets as with 2-D textures, a single pre-computed 3-D texture is sufficient.

Chapter 6

Arbitrary plane with the volume

Pre-computed 3-D texture with color values and opacities

Chapter 6

Volume Visualization

a) 

b) 

c) 

d) 

e) 

f)
Chapter 6

Perspective projection

Since interpolated values can be computed at arbitrary locations within the volume using 3-D textures it is also possible to realize perspective projections. In this case, the samples are located along concentric spheres surrounding the viewer. By rendering these spheres and applying the 3-D texture the performance of ray casting can be improved.

Chapter 6

Disadvantages of hardware assisted methods

Even though the interpolation and $\alpha$-blending is exactly what is required for direct volume rendering, the result of hardware assisted methods may not be exact. The main reason for this is the lower precision of many graphics cards. The software implementation uses precise floating point arithmetic, many graphics cards only allow 8-bit fixed point calculations for each color component. Rounding errors occur which may be amplified by subsequent steps. Modern graphics cards, such as NVIDIA GeForce series 6 and 7 however, support 64-bit floating point values for textures.
Chapter 6

Problems with the Levoy illumination model

Another problem concerns the Levoy illumination model and other models that try to achieve the appearance of surfaces. If these models are only evaluated at the grid vertices followed by an interpolation of the resulting color values, this is no longer Phong shading but equivalent to Gouraud shading within the volume. Furthermore, since the illumination effects depend on the view direction (the direction to the viewer is included in the illumination model), the texture needs to be re-computed for a different direction.
Chapter 6

Hardware assisted volume rendering using OpenGL

OpenGL supports 3-D texture mapping directly. First, the color values have to be computed and stored in a texture properly (including opacity values). Since the texture needs to be mapped on some geometry, several planes need to be generated which are all orthogonal (for orthogonal projection) to the view direction and usually placed equidistantly.

**Beware:** the number of planes influences both image quality and computational time directly!
Chapter 6

Creating a 3-D texture using OpenGL

A 3-D texture is created using the command `glTexImage3D`:

```c
void glTexImage3D (GLenum target,
                  Glint level,
                  Glint internalFormat,
                  GLsizei width,
                  GLsizei height,
                  GLsizei depth,
                  Glint border,
                  GLenum format,
                  GLenum type,
                  const GLvoid *texels);
```
Chapter 6

Example:

```c
glTexImage3D(GL_TEXTURE_3D,
0,
GL_RGBA,
tex.size_x,
tex.size_y,
tex.size_z,
0,
GL_RGBA,
GL_UNSIGNED_BYTE,
tex.pix);
```
Chapter 6

Creating geometry to map texture onto

max = 32;

for (int i=0; i<=max; i++) {
    glBegin (GL_QUADS)
    glTexCoord3f (0.0, 0.0, (float)i/(float)max);
    glVertex3f (0.0, 0.0, (float)i/(float)max);
    glTexCoord3f (1.0, 0.0, (float)i/(float)max);
    glVertex3f (1.0, 0.0, (float)i/(float)max);
    glTexCoord3f (1.0, 1.0, (float)i/(float)max);
    glVertex3f (1.0, 1.0, (float)i/(float)max);
    glTexCoord3f (0.0, 1.0, (float)i/(float)max);
    glVertex3f (0.0, 1.0, (float)i/(float)max);
    glEnd ()
}
Chapter 6

Enable 3D texture mapping

```c
 glEnable (GL_TEXTURE_3D);
```

Also, you need to request a visual that includes transparencies. In GLUT, this is done via the function `glutInitDisplayMode`:

```c
 glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
```

and define blending

```c
 glEnable (GL_BLEND);
 glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```
View manipulations

The texture can be scaled, rotated, and panned in OpenGL using the matrix \texttt{GL\_TEXTURE}. Hence, by switching to this matrix using the command

```
glMatrixMode(GL_TEXTURE);
```

the current view can be changed; e.g. by zooming, panning, or rotating; and OpenGL adapts the visualization automatically.
Chapter 6

Multi-dimensional transfer functions

Designing appropriate transfer functions is very difficult for three main reasons:

– There is an enormous number of degrees of freedom
– Usual interfaces are not constrained or guided by the underlying data set
– Separating different tissue types based on intensity value alone may be difficult or even impossible

To address the third issue, multi-dimensional transfer functions can be used.
Chapter 6

Multi-dimensional transfer functions

Traditionally, transfer functions were simply based on the intensity values provided by the volumetric data set. Hence, the intensity values were directly mapped to color and opacity values in a one-dimensional fashion. However, this already presents a challenging problem to the user to design the transfer function. For example, basing the transfer function on a few simple ramp functions around intensity values of interest already provides several degrees of freedom.
Chapter 6

Multi-dimensional transfer functions

In order to provide some guidance to the user, histograms are often used. The occurrences of the intensity values are simply counted and then plotted as a one-dimensional function. Since the axis for this histogram plot are similar to the ones for color and opacity values, the transfer function is often overlaid on top of the histogram:
Chapter 6

Multi-dimensional transfer functions

By adding more dimensions to the transfer function, the complexity of designing the transfer function becomes even more complex. However, it provides more options at the same time. Usually, the image gradient as the basis for the second dimension. As a result, the transfer function is defined on a two-dimensional area to map to color and opacity values.
Chapter 6

Multi-dimensional transfer functions

The image gradient for a volumetric data set can simply be computed as the difference between neighboring intensity values in each dimension:

\[ x_{i+1} - x_{i-1} \]
\[ y_{i+1} - y_{i-1} \]
\[ z_{i+1} - z_{i-1} \]

Or as filter matrix:

\[-1 \ 0 \ 1 \]
\[-1 \ 0 \ 1 \]
\[-1 \ 0 \ 1 \]
\[-1 \ 0 \ 1 \]
Chapter 6

Multi-dimensional transfer functions

Hence, we get a 3D vector as a result indicating the direction of the fastest change.

However, better results are typically achieved by using a derivative of Gaussian filter. Similar to Gaussian filtering, this incorporates some smoothing at the same time when the gradient is computed.
Chapter 6

Multi-dimensional transfer functions

The basis of the derivative of Gaussian filter is simple the first derivative of the Gaussian function that is applied in all three dimensions.
Chapter 6

Multi-dimensional transfer functions

Since the volumetric data is discrete in nature in that data is only present at discrete locations, i.e. the vertices of the regular grid structure, the derivative of Gaussian needs to be discretized so that it can be applied to the volumetric data.
Chapter 6

Multi-dimensional transfer functions

Since the gradient vector indicates the direction of the fastest change, it is usually used as indicator for a transition between different types of tissues. If we assume that these gradient vectors are located in a surface along this transition, it is only natural to use these gradient vectors as normal vectors to compute correct lighting to improve the resulting visualization and add a more “natural” appearance.
Chapter 6

Multi-dimensional transfer functions

Typically, the Phong illumination model is used in computer graphics. This model is not physically accurate, but it can be computed fast and is therefore a good model for interactive rendering. It computes the overall lighting based on three components:

• Diffuse reflection
• Specular reflection
• Ambient light
Chapter 6

Multi-dimensional transfer functions

The diffuse reflection assumes that light is scattered in all directions after it hits a surface. The amount of reflected light depends on the incoming angle of the light. If the light hits the surface in an orthogonal fashion most light is reflected. If the direction of the incoming light is almost parallel to the surface almost no light is reflected.
Chapter 6

Multi-dimensional transfer functions

We can therefore simply use the cosine between light direction and normal vector to compute the amount of reflected light. Assuming all involved vectors are normalized, i.e. have the length 1, this results in:

\[ I = I_{\text{source}} \cdot k_d (N \cdot L) \]
Chapter 6

Multi-dimensional transfer functions

The specular part of the Phong illumination model considers light that is directly reflected similar to a mirror. Hence, the light will only be visible from the viewers point if it is close to the reflected direction. The closer the vector pointing toward the viewer is to the reflected direction, the more light will arrive at the viewer’s location.
Chapter 6

Multi-dimensional transfer functions

Hence, we can use the cosine again; this time using the reflected direction and the vector pointing toward the viewer:

\[ I = I_{\text{source}} k_s (R \cdot N)^n \]

The exponent \( n \) is just a design parameter indicating the “perfectness” of the reflection.
Chapter 6

Multi-dimensional transfer functions

The Phong illumination model is not physically accurate as it does not include any reflection of light bouncing off of other surfaces; it only considers light coming directly from light sources. To make up for this deficiency, an ambient background illumination is added that evenly brightens up everything.

\[ I = I_{\text{source}} k_a \]
Chapter 6

Multi-dimensional transfer functions

By adding all the components, we get the equation describing the Phong illumination model:

\[ I = I_{\text{source}} k_a + I_{\text{source}} k_s (R \cdot N)^n + I_{\text{source}} k_d (N \cdot L) \]

The coefficients \( k_a \), \( k_s \), and \( k_d \) are design parameters that can be used to describe different surface characteristics by identifying how reflective a surface is with respect to the diffuse and specular components. This then provides a large variety of options for defining various types of surface.
Chapter 6

Multi-dimensional transfer functions

Similar to the gradient direction indicating the direction of fastest change, the length of the gradient vector indicates the amount of difference between the tissue types with respect to their intensity values. The longer the gradient vector, the larger the difference in intensity. As a result, the gradient length can be computed for all gradient vectors and then be used for further processing. Particularly, the gradient lengths then describe the second dimension the transfer function is defined on in addition to the intensity values.
Chapter 6
Multi-dimensional transfer functions
With two dimensions available, the histogram can now be plotted accordingly:

(a) A 1D histogram. The black region represents the number of data value occurrences on a linear scale, the grey is on a log scale. The colored regions (A,B,C) identify basic materials.

(b) A log-scale 2D joint histogram. The lower image shows the location of materials (A,B,C), and material boundaries (D,E,F).

Image courtesy of Joe Kniss
Chapter 6

Multi-dimensional transfer functions

Volume rendering based on a 2D transfer function with the areas marked according to the plotted histogram from the previous slide.

Image courtesy of Joe Kniss
Chapter 6

Multi-dimensional transfer functions

For the frontal and maxillary sinuses of the Visible Male CT, a 1D transfer function can show the sinuses along with the skin, it cannot capture them in isolation. Only a higher-dimensional transfer function, in this case 2D using data value and gradient length, can uniquely classify them.

Image courtesy of Joe Kniss