Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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).
Scalar algorithms

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.
Scalar algorithms

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.

Sampling rate too low

Minimal sampling rate

Department of Computer Science and Engineering
Scalar algorithms

Illumination Models

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


Scalar algorithms

Pure absorption

This “illumination model” assumes that 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 $A = \pi r^2$
Scalar algorithms

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 absorbed by the cylinder is $\rho AE\Delta s/E = \rho A\Delta s$. 
Scalar algorithms

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 $s$. 
Scalar algorithms

The solution for this differential equation is

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

The term

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

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

Example

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

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

$$I(s) = I_0 + \int_0^s g(t) dt$$
Scalar algorithms

Example

Image of a cloud created by an illumination model using pure emission.
Scalar algorithms

Emission and absorption

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

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

Emitted light \quad Absorbed light
Scalar algorithms

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:

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

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.:

$$\alpha = 1 - T(s) = 1 - \exp\left(-\int_{0}^{l} \tau(t) \, dt\right)$$
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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 $[0, \text{#values}-1]$ 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.
Scalar algorithms

Brain of a fruit fly, imaged using laser microscopy

[c]2000 www.amiravis.com

[Data courtesy of Genetikinstitut University of Wuerzburg; software Amira, Konrad-Zuse-Zentrum, Berlin]
Scalar algorithms

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 $T(s)$:

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

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.
Scalar algorithms

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:

$$\sum_{i=1}^{n} g_i \prod_{j=i+1}^{n} t_j$$
Scalar algorithms

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

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

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

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

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

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

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

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

If we no 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 $\tilde{E}_i = C(i\Delta x)\alpha(i\Delta x)$
Scalar algorithms

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

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

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:

\[ \tilde{I}_0 = \tilde{E}_0; \gamma_0 = \alpha_0 \]
\[ \tilde{I}_k = \tilde{I}_{k-1} + (1 - \gamma_{k-1}) \tilde{E}_k \]
\[ \gamma_k = \gamma_{k-1} + (1 - \gamma_{k-1}) \alpha_k \]

Back-to-front compositing

Front-to-back compositing
Scalar algorithms

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 $F \oplus B = (1-\alpha) B + \alpha F = B + \alpha (F-B)$. 
Scalar algorithms

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

\[
G \oplus (F \oplus B) = (1- \beta) F \oplus B + \beta G
= (1- \beta) ((1-\alpha) B+ \alpha F) + \beta G.
\]
Scalar algorithms

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

$$1 - \gamma = (1 - \alpha)(1 - \beta)$$
Scalar algorithms

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.
Scalar algorithms

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:

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

$$\gamma = (1 - \beta) \alpha + \beta$$
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms
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.
Scalar algorithms

Scalar algorithms

Scalar algorithms

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 end emission model and the used model is governed by different parameters, the results are very similar (as well as the equations).
Scalar algorithms

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.
Scalar algorithms

The maximum (upper left), distance to the maximum (upper center), the intensity (upper right), and the combination using the HSV color model
Scalar algorithms

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.
Scalar algorithms

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.

Scalar algorithms

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:

\[ c_{\lambda}(x_i) = c_{p,\lambda} k_{a,\lambda} + \frac{c_{p,\lambda}}{M_1 + M_2 d(x_i)} (k_{d,\lambda} (N(x_i) \cdot L) + k_{s,\lambda} (N(x_i) \cdot H)^n) \]

- \( c_{p,\lambda} \): color component at \( x_i \)
- \( k_{a,\lambda} \): ambient reflection coefficient
- \( k_{d,\lambda} \): diffuse reflection coefficient
- \( k_{s,\lambda} \): specular reflection coefficient
- \( M_1 \) and \( M_2 \): distance to light source and distance to observer, respectively
- \( N(x_i) \cdot L \): dot product of surface normal (normalized gradient) and light vector
- \( N(x_i) \cdot H^\prime \): dot product of surface normal and direction to maximal reflection
- \( d(x_i) \): distance model
- \( \lambda \): wavelength parameter

"surface" normal (normalized gradient)
Scalar algorithms

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 $N(x_i) \cdot L = 0$. 
Scalar algorithms

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.
Scalar algorithms

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:
Scalar algorithms

$$\alpha(x_i) = \alpha_v \begin{cases} 
1 & \text{if } |\nabla f(x_i)| = 0 \text{ and } f(x_i) = f_v \\
1 - \frac{1}{r} \left| \frac{f_v - f(x_i)}{|\nabla f(x_i)|} \right| & \text{if } |\nabla f(x_i)| > 0 \text{ and } f(x_i) - r |\nabla f(x_i)| \leq f_v \\ 
0 & \text{otherwise} 
\end{cases}$$
Scalar algorithms

opacity $\alpha(x_i)$

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

acquired value $f(x_i)$
Scalar algorithms

“Isosurfaces” in the protein cyochrome B5
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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 $f_{vn}, n=1,...,N$, such that $f_{vm} < f_{vm+1}$, $m=1,...,N-1$ 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.
Scalar algorithms

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:

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

opacity $\alpha(x_i)$

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

acquired value $f(x_i)$
Scalar algorithms
Scalar algorithms
Scalar algorithms

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.
Scalar algorithms

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:

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

Control of maximal opacity “contrast”
Scalar algorithms

Volume Illustration Rendering Pipeline

- Volume Rendering
  - Transfer function
  - Volume Illustration
    - color modification
  - Volume Illustration
    - opacity modification
- Final volume sample \([c_\lambda(x_i), \alpha(x_i)]\)
- image pixels \(C_\lambda(u_i)\)

volume values \(f_i(x_i)\)
Scalar algorithms

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.
Scalar algorithms

Image based on original illumination model
Scalar algorithms

Transitions between regions are enhanced
Scalar algorithms

Enhanced transitions between regions and silhouettes
Scalar algorithms

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/
Scalar algorithms

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.
Scalar algorithms

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 Lorenson, “The Visualization Toolkit”]
Scalar algorithms

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.
Scalar algorithms

Often times it makes sense to sub-divide the ray, such that each segment are entirely contained in exactly one cell. For each of these segments, the desired values (intensity, maximum, etc) is 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 though the image is computed for each pixel individually (image-order), the sampling of the ray is done on a per-voxel basis (object-order).
Scalar algorithms

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”]
Scalar algorithms

The result of the modified Bresenham algorithm is a series \( v_1, v_2, ..., v_n \) 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 Lorensen, “The Visualization Toolkit”]
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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 which 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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

Image plane
Scalar algorithms

Image plane

- Front facing
- Back facing
Scalar algorithms

Interpolate depth and value

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

Image plane
Scalar algorithms

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:

$$I_{cell} = \int_{t_i}^{t_o} g(s)T_{cell}(s)ds$$

where

$$T_{cell}(s) = \exp\left(-\int_{s}^{t_o} \tau(x)dx\right)$$
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

$v_y > 0 \Rightarrow$ decreasing

$v_x < 0 \Rightarrow$ increasing

$v_y > 0$

$v_x < 0$
Scalar algorithms

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.
Scalar algorithms

The blocks themselves are traversed starting at the outside and approaching the center.
Scalar algorithms

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.
Scalar algorithms

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.
**Scalar algorithms**

**Splatting**

Splatting is an object-order method for computing of 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:

$$ (f_1 * f_2)(x) = \int_{-\infty}^{+\infty} f_1(t) f_2(x - t) dt $$
Scalar algorithms

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.

Scalar algorithms

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:

$$signal_{3D}(x, y, z) = \sum_{D \in \text{Vol}} h_V(x - D_x, y - D_y, z - D_z) f(D)$$

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

$$contribution_D(x, y, z) = h_V(x - D_x, y - D_y, z - D_z) f(D)$$
Scalar algorithms

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:

$$\text{contribution}_D(x, y) = f(D) \int h_V(x - D_x, y - D_y, \omega) d\omega$$
Scalar algorithms

Density distribution of a three-dimensional Gauss distribution


Department of Computer Science and Engineering
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”
Scalar algorithms

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:

$$\text{footprint}_D(x, y) = \int h_v(x, y, \omega) d\omega$$

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.
Scalar algorithms

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”
Scalar algorithms

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”
Scalar algorithms

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).
Scalar algorithms

Shear-Warp - principle
Scalar algorithms

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 α-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.
Scalar algorithms

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.

Scalar algorithms

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.

Scalar algorithms

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 $\alpha$-blending (compositing) and placing the planes at the correct distances with the according textures containing the color values.
Scalar algorithms

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.
Scalar algorithms

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

Scalar algorithms

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.

Scalar algorithms

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.

Scalar algorithms

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.

Scalar algorithms

Arbitrary plane with the volume

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

Pixel

Texel

Scalar algorithms
Scalar algorithms

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.

Scalar algorithms

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.
Scalar algorithms

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.
Scalar algorithms

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!
Scalar algorithms

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);
```
Scalar algorithms

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);
```
Scalar algorithms

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

\begin{verbatim}
glMatrixMode(GL_TEXTURE);
\end{verbatim}

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

Triangulation of point clouds

Triangulation techniques build topology directly from unstructured points. The points are triangulated to create a topological structure consisting of \( n \)-dimensional simplices that completely bound the points and linear combinations of the points (the so-called convex hull). The result of triangulation is a set of triangles (2-D) or tetrahedra (3-D), depending upon the dimension of the input data.
Modeling algorithms

An $n$-dimensional triangulation of a point set $P = \{p_1, p_2, p_3, \ldots, p_n\}$ is a collection of $n$-dimensional simplices whose defining points lie in $P$. The simplices do not intersect one another and share only boundary features such as edges or faces. The Delaunay triangulation is a particularly important form. It has the property that the circumsphere of any $n$-dimensional simplex contains no other points of $P$ except the $n+1$ defining points of the simplex.
Modeling algorithms

Properties of Delaunay triangulations

The Delaunay triangulation has many interesting properties. In two dimension, the Delaunay triangulation has been shown to be the optimal triangulation. That is, the minimum interior angle of any triangle in a Delaunay triangulation is greater than or equal to the minimum interior angle of any other possible triangulation, i.e. it maximizes the minimal interior angle.
Modeling algorithms

Example:

(a) Delaunay triangulation
Modeling algorithms

Voronoi tessellation

The Delaunay triangulation is the dual of the Dirichlet tessellation. The Dirichlet tessellation, also known as the Voronoi tessellation or Voronoi diagram, is a tiling of space where each tile represents the space closest to a point $p_i$. An $n$-dimensional Delaunay triangulation can be constructed from the Dirichlet tessellation by creating edges between the Voronoi cells (the cells of a Voronoi tessellation) that share common $n-1$ boundaries (e.g. faces in 3-D and edges in 2-D). Conversely, the vertices of the Dirichlet tessellation are located at the circumcenters of the Delaunay circumcircles.
Modeling algorithms

Example:

(b) Dirichlet tessellation
Modeling algorithms

Computing the Delaunay triangulation

The Delaunay triangulation can be computed using a variety of techniques. One elegant technique is based on the circumcircles. The algorithm begins by constructing an initial Delaunay triangulation that strictly bounds the point set $P$, the so-called bounding triangulation. This bounding triangulation can be as simple as a single triangle (2-D) or tetrahedron (3-D). Then, each point of $P$ is injected one by one into the current triangulation. If the injected point lies within the circumcircle of any simplex, then these simplices are deleted and replaced with a local triangulation including the new point.
Modeling algorithms

Example:

Bounding triangulation

Inject point

Create new triangulation

Remove boundary points
Information Visualization

Data types

File systems, internet connections, network of streets, and communications systems are examples of connected structures which can be modeled using a graph of different specializations. The most common types of graphs are:

- Undirected graph: this is a tuple \( G=\{V,E\} \) with a set of nodes \( V=\{v_1,\ldots,v_n\} \) and a set of edges \( E=\{(v_i,v_j)\in\{(v_p,v_q)|i,j=1,\ldots,n; i\neq j\} \). Often times, there are more data elements attached to the edges.

- Directed graph: this is again a tuple \( G=\{V,E\} \) with \( V=\{v_1,\ldots,v_n\} \) and \( E=V\times V \). Again, additional data can be attached to the edges.

- Trees: a tree is a graph without any cycles.

- Multigraph: both directed and undirected graphs allow the presence of multiple edges connecting the same nodes. These graphs are called multigraphs.
Information Visualization

These data types occur frequently. A simple example is the directory hierarchy on a hard drive. Typical questions are:

- Where am I?
- Where is the file I am looking for?

Other typical applications are:

- Organization chart at a hospital
- Taxonomy of biological species
- Evolutionary trees
- Molecular and genetic charts
- Phylogenic trees
- Biochemical reaction paths
Information Visualization

Besides the type of graph (tree, acyclic directed graph, arbitrary directed graph, undirected graph, multigraph) the size of the graph (number of nodes and edges) is a key point in the visualization.

A user is usually not able to comprehend large graphs from a single image. The visualization then only shows the complexity of the graph.

From a certain size on, there are no suitable layout algorithms anymore simply because there is not enough space on the screen. Hence, we need to reduce the complexity first.
Information Visualization

Graph drawing

The basic problem of graph drawing can be described easily:

**Problem:** Let a set of nodes and a set of edges be given. Compute the positions for the nodes and the curves representing the edges.

It is not easily explained what a good layout is. Battista at al. [IEEE TVCG 6(1): 24- , G. Di battista, P. Eades, R. Tamassia, and I. G. Tollis, „Algorithms for Drawing Graphs: An Annotated Bibliograph“, Geometry: Theory and Applications, vol. 4, no. 5, pp. 235-282, 1994] list hundreds of papers devoted to this issue. The difficulty is to define properties of the graph and classify the layouts. A typical property of a graph is being planar. Hence, we need a fast algorithm which can check that property. For an undirected graph a complexity of $O(n)$ can be achieved [J. Hopcroft and R. E. Tarjan, „Efficient Planarity Testing“, J. ACM vol. 21, no. 4, pp. 549-568, 1974]. A possible layout algorithm could be based on a regular grid and use only integer coordinates for the nodes.
Information Visualization

For the visualization, however, the property of being planar is not of that much importance. But the minimization of intersections between the curves representing the edges is a major goal. In addition, the layout influences the perception of the graph.
Information Visualization

Tree drawing

The algorithm of Reingold and Tilford [E. M. Reingold and J. S. Tilford, „Tidier Drawing of Trees“, IEEE Trans. Software Eng., vol7, no2, pp. 223-228, 1981] generates a classic tree representation. All nodes of the same level are located at the same height. The horizontal space is split up according to the number of leaves of the sub-trees.
Information Visualization

It is also possible to use an H-like pattern: [P. Eades, „Drawing Free Trees“, Bulletin of the Inst. For the Combinatorics and Its Applications, pp. 10-36, 1992]
Information Visualization

Another variant uses a radial pattern. The root of the tree is located at the center of the image while the nodes are placed on concentric circles. In addition, the algorithm avoids intersections by choosing fixed sections for the sub-trees. It is possible to weaken the last condition to achieve better results [I.Herman, G. Melancon, M. M. De Ruiter, and M. Delest, „Latour-A Tree Visualization System“, Proc. Symp. Graph Drawing GD'99, pp. 392-399, 1999. A more detailed version in: Reports of the Centre for Math. And Computer Sciens, Report number INS-R9904, available at: http://www.cwi.nl/InfoVisu/papers/LatourOverview.pdf, 1999]
Information Visualization

In a cone tree [J. Carriere and R. Kazman, „Research Report: Interacting with Huge Hierarchies: Bryond Cone Trees“, Proc. IEEE Conf. Information Visualization '95, pp. 74-81, 1995], siblings are located along a circle which, including the parent node, results in a cone.
Information Visualization

A tree map [B. Johnson and B. Schneiderman, „Tree-Maps: A Space-Filling Approach to the Visualzation of Hierarchical Information Structures“, Proc. IEEE Visualization ’91, pp. 275-282, 1991] assigns a rectangle to each sub-tree which is then sub-divided further. Using the size of the rectangles, additional information can be conveyed.

Tree-map: rectangles with color belong to the same level of the (tree) hierarchy. (Adapted from Johnson and Schneiderman [72]).
Information Visualization

The previously discussed methods are predictable, i.e. they give the exact same results for the same input which is an important property in visualization. This is no longer true when using simulated annealing or spring models.
Information Visualization

Layout in the hyperbolic plane

Even though a radial layout of the tree utilizes the space more efficiently compared to classic approaches, it still does not provide sufficient results. The number of nodes grows exponentially per level, while the circle grows linearly within the Euclidian plane. In the hyperbolic plane however, the growth is exponential!

Figure 1. A partial organization chart of Xerox (ca. 1988)
Information Visualization

Figure 2. Original inspiration for the hyperbolic browser. Circle Limit IV (Heaven and Hell), 1963, ©1994 M.C. Escher/Cordon Art—Baarn—Holland. All rights reserved. Printed with permission.
Information Visualization

Basically, the hyperbolic layout is a radial layout where the size gets smaller rapidly when approaching the boundary of the circle. Here, the circular section is determined based on the number of children without considering their grand-children in order to be able to lay out even large trees.
Information Visualization

It is possible to navigate through such a layout by moving the center of projection.

We will see later that we need to avoid a rotation of the origin after several movements of the center.
Information Visualization

The layout is arranged in the hyperbolic plane and then mapped onto a circle within the Euclidian plane. This maps circles onto circles so that it is possible to leave some space for annotations. An ellipse formed by the siblings, parent, and middle child can also be used for annotations.
Information Visualization

Hyperbolic geometry

Euclid postulated five axioms for his geometry:

- There is one line segment that connects two points.
- Every line segment can be extended to an infinite straight line.
- For every line segment, a circle can be constructed that has the segment as its radius and one end point as its center.
- All orthogonal angles are congruent.
- For every straight line and a given point that is located on the line, there is exactly one infinite straight line that does not intersect the existing one.

By replacing the last axiom with one allowing that there are several parallel lines we get the hyperbolic geometry.
Information Visualization

Poincaré model

The model of a hyperbolic plane according to Poincaré consists of a unit disc $C$ (complex)

$$P = \{ z \in C \mid \| z \| < 1 \}$$

“points”

with the locally dependent metric

$$ds^2 = \frac{(dx^2 + dy^2)}{(1 - x^2 - y^2)}$$

“distance”

The “straight lines” are circular arcs orthogonal to the boundary.

Since it is a conform model the angles between the circular arcs determine the angles of straight lines in the hyperbolic plane.
Information Visualization

For the navigation, we need a transformation of the hyperbolic plane. These can be described by

\[ T_{P,\theta} : C \rightarrow C \]

\[ z \rightarrow \frac{\theta z + P}{1 + P \theta z} \]

with \( P, \theta \in C, |P| < 1, |\theta| = 1 \) and \( \bar{p} \) as its conjugated point. This is a rotation of angle \( \theta \) and a subsequent translation of the origin to \( P \).

When combining two of these transformations we get

\[ P' = \frac{\theta_2 P_1 + P_2}{\theta_2 P_1 \bar{P}_2 + 1} \]

\[ \theta' = \frac{\theta_1 \theta_2 + \theta_1 \bar{P}_1 P_2}{\theta_2 P_1 \bar{P}_2 + 1} \]
Information Visualization

Layout

The layout is realized as a recursive procedure with a circular section as its parameter. The circular section is described by a vertex, the end point of the line segment cutting the section in half, and the half the angle.

A simple layout uses the vertex as the location for the parent node and sub-divides the section according to the number of children. Better results are usually achieved when using a logarithmic scale when sub-dividing. The distance to the child is determined by

\[
d = \sqrt{\frac{(1-s)^2 \sin(a)^2}{2s} + 1 - \frac{(1-s^2) \sin(a)}{2s}}
\]

where \(a\) is half of the angle of the child’s circular section and \(s\) is the desired distance between the child and the edge of the circular section. Good results can be achieved using \(s=0.12\).
Information Visualization

Visualization

The visualization of the graph is achieved by drawing the nodes (if desired with annotations) and the edges as circles which correspond to hyperbolic line segments.

The computation of the center of a circle for complex numbers $a, b \in \mathbb{P}$ uses the following formula:

$$d = \text{Re}(a) \text{Im}(b) - \text{Re}(b) \text{Im}(a)$$

$$c = \frac{i}{2} \cdot \frac{(a(1 + \|b^2\|) - b(1 + \|a^2\|))}{d}$$
Information Visualization

Navigation

The layout of the graph in the hyperbolic plane is not changed. Only the mapping of the complex unit disc is manipulated using the previous transformation.

We start with $T_{0,1}$. If the user moves the mouse from $s$ to $e$ and the point $p$ is not supposed to be rotated then we get:

\[
a = T_{-p,1}(s)
\]

\[
b = \frac{\text{Re}((e-a)(1-\overline{ae}))i}{1-(ae)^2}
\]

\[
T = T_{-p,1} \circ T_{b,1}
\]

For a smooth transition it is possible to introduce intermediate steps.
Information Visualization

Literature


Additional hyperbolic approaches:

Information Visualization

Layout of general directed graphs

For these types of graphs, a suitable layering is identified first which assigns a integer number to every node. Most methods are based on extraction of an acyclic sub-graph which contains all nodes. This way, all nodes receive a number and are ordered in rows from top to bottom so that all edges of the acyclic graph point downward. This arrangement is used for minimizing the steps, often only up to the next level. This problem, however, is NP hard.

Information Visualization

Spring-based methods

These methods model nodes and edges as physical entities connected by springs. This results in an optimization problem which can only be solved locally (hence not predictable). In addition, these methods tend to be slow with a complexity of $O(n^3)$ [A. Frick, A. Ludwig, and H. Mehldau, "A Fast Adaptive Layout Algorithm for Undirected Graphs“, Proc. Symp. Graph Drawing GD ’93, pp. 389-403, 1994].
Information Visualization

Layout of undirected graphs

For undirected graphs, we usually start with a spanning tree. This tree is then laid out. The edges can be assigned weights before computing the spanning tree. Good algorithms have complexities $O(N \log N)$ or $O(E \log N)$ ($E$: number of edges, $N$: number of nodes).
Information Visualization

3-D Layout

It is also possible to use 3-D-based layouts. Besides the cone tree representation, two approaches are shown here. However, current input devices are not very suitable for navigation.
Information Visualization

Exploration

For large trees, a fish-eye view can help explore the data.
Information Visualization

In addition, a windowing technique can be used for exploring large graphs.
Information Visualization

Clustering

If the graph is too large to display it can be simplified. Usually, this is done by clustering nodes. Man differentiates different approaches:

- Structural clustering: combine nodes based on the structure
- Content-based clustering: combine semantically similar nodes

Almost all techniques are based on structural clustering, since it is easier to implement and the method can be applied to any graph independently of the application.
Information Visualization

The algorithm generates disjoint clusters. The clusters are then form a new graph by using each cluster as a node and drawing an edge if there is an edge between elements of two different clusters.
Information Visualization

For the clustering, a metric for the nodes is required. This metric can be structural or content-based. Clusters are then formed based on the “distance” between nodes using a pre-defined threshold.
Information Visualization

It is also possible to assign a value of relevance to the nodes and edges. There are three different approaches:

– Ghosting: less relevant nodes and edges are shifted towards the background
– Hiding: less relevant elements are omitted
– Grouping: less relevant elements are combined
Information Visualization

Multidimensional visualization

The treatment of multidimensional data sets is an important data visualization issue. Each point in a data set is described by an $n$-dimensional coordinate, where $n > 3$. Here we assume that each coordinate is an independent variable, and that we wish to visualize a single dependent variable. An application of multidimensional data is financial visualization, where we might want to visualize return on investment as a function of interest rate, initial investment, investment period, and income, to name just a few possibilities.
Information Visualization

There are two fundamental problems that we must address when applying multidimensional visualization. These are the problems of *projection* and *understanding*.

The problem of projection is that in using computer graphics we have two dimensions in which to present our data. Using 3-D graphics we can give the illusion of three dimensions.

The problem of understanding is that humans do not easily comprehend more than three dimensions, or possibly three dimensions plus time.
Information Visualization

Parallel coordinates

One approach to multidimensional visualization is the use of *parallel coordinates*. Instead of plotting points on orthogonal axes, the $i$-th dimensional coordinate of each point is plotted along separate, parallel axes. When using parallel coordinates, points appear as lines. As a result, plots of n-dimensional points appear as sequences of line segments that may intersect or group to form complex fan patterns. Unfortunately, if the number of points becomes large, and the data is not strongly correlated, the resulting plots can become a solid mass of black, and any data trends are drowned in the visual display.
Information Visualization

<table>
<thead>
<tr>
<th>MPG</th>
<th>Cylinders</th>
<th>Horsepower</th>
<th>Weight</th>
<th>Acceleration</th>
<th>Year</th>
<th>Origin</th>
</tr>
</thead>
</table>

Diagram showing relationships between vehicle attributes.
Another multivariable technique is using glyphs. This technique associates a portion of the glyph with each variable. Although glyphs cannot generally be designed for arbitrary $n$-dimensional data, in many applications we can create glyphs to convey the information we are interested in.