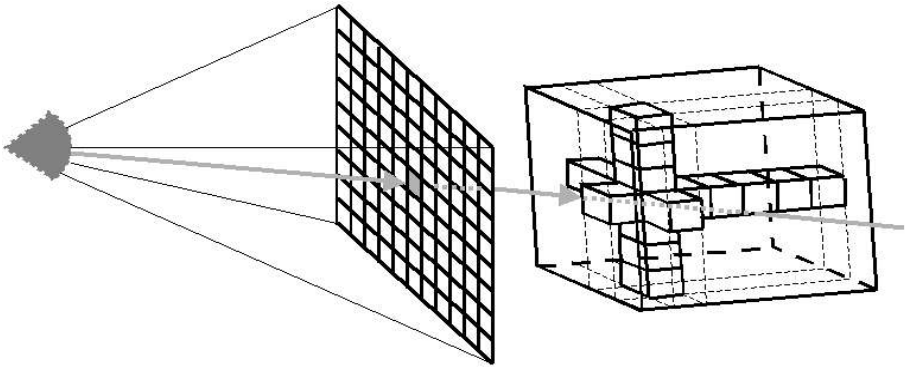
Direct Volume Rendering

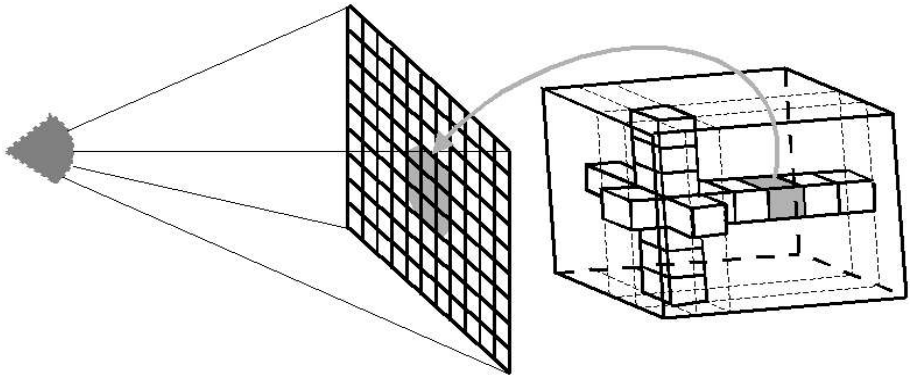
The idea of **Direct Volume Rendering** (**DVR**) is to get a 3D representation of the volume data directly.
The data is considered to represent a semi-transparent light-emitting medium. Therefore also gaseous phenomena can be simulated. The approaches in DVR are based on the laws of physics (emission, absorption, scattering). The volume data is used as a whole (we can look inside the volume and see all interior structures).

In DVR either backward or forward methods can be used.
Backward methods use image space/image order algorithms. They are performed pixel by pixel. An example is *Ray Casting* which well be discussed in detail below.



Backward methods

Forward methods use object space/object order algorithms. These algorithms are performed voxel by voxel and the cells are projected onto the image. Examples for this technique are Slicing, shear-warp and splatting.

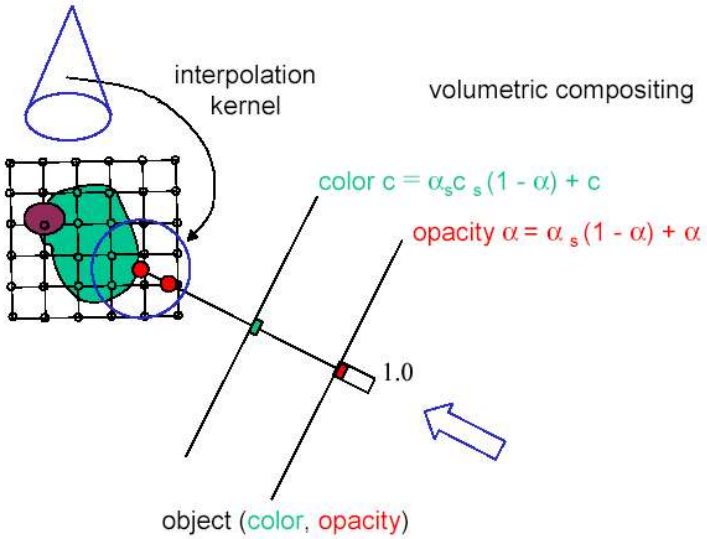


Forward methods

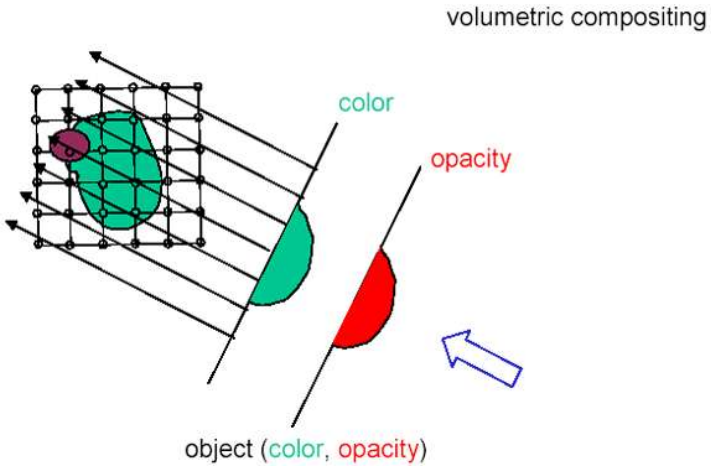
1 Ray Casting

Ray Casting is similar to ray tracing in surface-based computer graphics. In volume rendering we only deal with primary rays, i.e. no secondary rays for shadows, reflection or refraction are considered. Ray Casting is a natural image order technique.
Since we have no surfaces in DVR we have to carefully step through the volume. A ray is cast into the volume, sampling the volume at certain intervals. The sampling intervals are usually equidistant, but they don't have to be (e.g. importance sampling). At each sampling location, a sample is interpolated/reconstructed from the voxel grid. Popular filter

are nearest neighbor (box), trilinear, or more sophisticated (Gaussian, cubic spline).
Volumetric ray integration:
The rays are casted through the volume. Color and opacity are accumulated along each ray (compositing).



Ray Casting

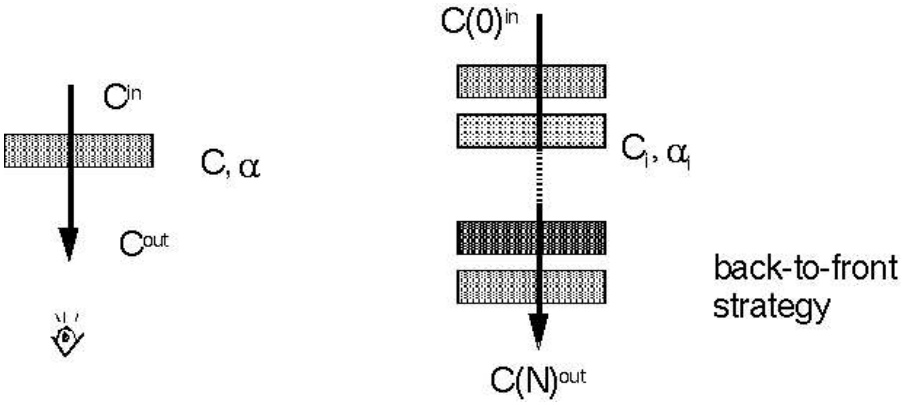


How is color and opacity determined at each integration step ?

- Opacity and (emissive) color in each cell according to classification
- Additional color due to external lighting: according to volumetric shading (e.g. Blinn-Phong)

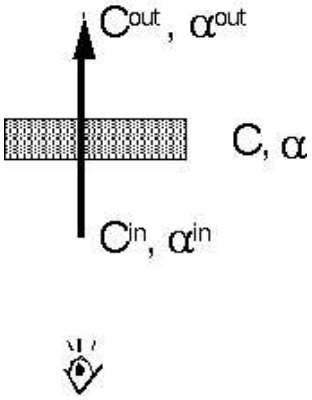
How can the compositing of semi-transparent voxels be done ?

- Physical model: emissive gas with absorption
- Approximation: density-emitter-model (no scattering)
- Over-operator was introduced by Porter *[Porter-1984-CDI]*: $C^{\text{out}} = (1-\alpha) C^{\text{in}} + \alpha C$, $C(i)^{\text{in}} = C(i-1)^{\text{out}}$



Over operator

A front-to-back-strategy is also possible: $C^{out} = C^{in} + (1-\alpha^{in})\alpha C$, $\alpha^{out} = \alpha^{in} + (1-\alpha^{in})\alpha$
This approach causes the need to maintain α .



Front-to-back-strategy

There are several traversal strategies:
- Front-to-back (most often used in ray casting)
- Back-to-front (e.g. in texture-based volume rendering)
- Discrete (Bresenham) or continuous (DDA) traversal of cells

2 Acceleration Techniques for Ray Casting

The problem with ray casting is the fact that it is very time consuming. The basic idea for accelerating the rendering process is to neglect "irrelevant" information and to exploit coherence.

2.1 Early Ray Termination

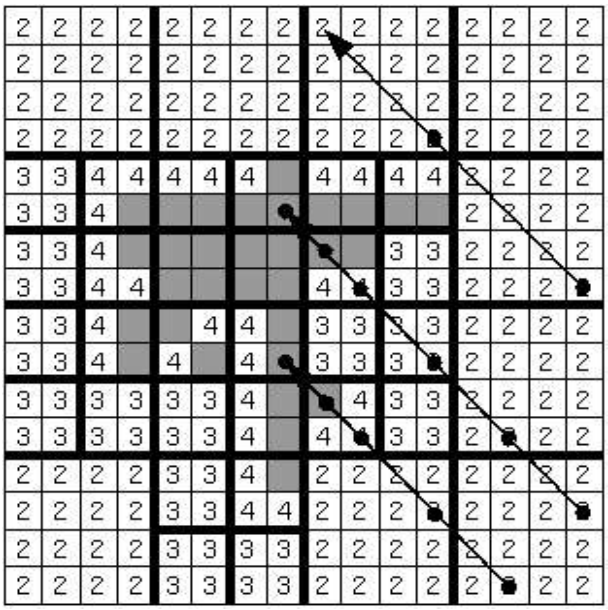
Colors from far away regions do not contribute if the accumulated opacity is too high. The idea of the **early ray termination** approach is to stop the traversal if the contribution of a sample becomes irrelevant. The user can set an opacity level for the termination of a ray. The color value is computed by front-to-back-compositing.

2.2 Space-Leaping

Space-leaping means the fast traversal of regions with homogeneous structure. These regions can be walked through rapidly without loosing relevant information. There are several techniques for this principle.

Original Space-Leaping

In the original space-leaping approach the empty space of the volume is efficiently traversed or even completely skipped. There are several methods for realizing this idea.
Hierarchical spatial data structure: The volume is subdivided into uniform regions which are represented as nodes in a hierarchical data structure (e.g. octree). The empty space can be skipped by traversing the data structure. This technique can be optimized by storing the uniform information of the octree in the empty space of the 3D volume grid. This data structure is called *flat pyramid*.

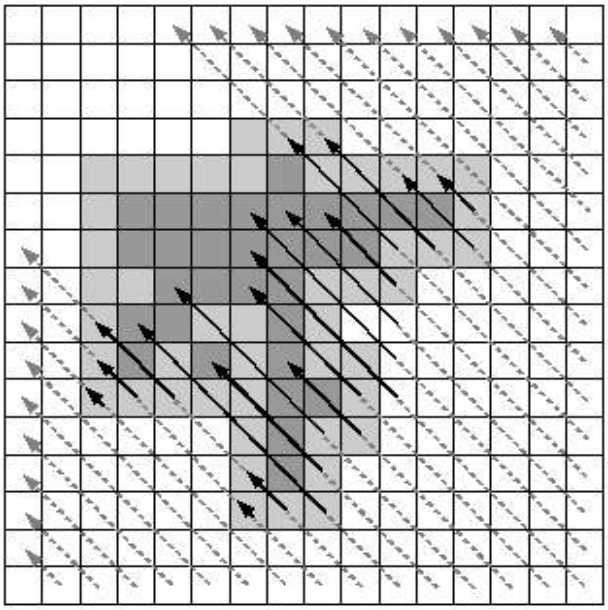


Flat pyramid

The gray squares are the voxels which contain volume data. The white squares denote empty voxels. In these voxels the level of the octree they belong to is stored. When a ray encounters a voxel with uniformity information it jumps to the first voxel outside this uniform region. Voxels with data are treated as usual. The voxels visited by a ray are marked with black dots in the illustration. In this example the rays terminate after two data voxels because a certain opacity level is reached.

Bounding boxes around objects: For volumes which contain only few objects bounding boxes can be used for space-leaping. The PARC (Polygon assisted ray casting) method allows to build a convex polyhedron around the object.

Adaptive ray traversal: The idea of this approach is to assign all empty voxels neighboring a data voxel a so called "vicinity flag". The empty space is traversed rapidly by an algorithm with low precision and when a vicinity voxel occurs it is switched to a more accurate algorithm.

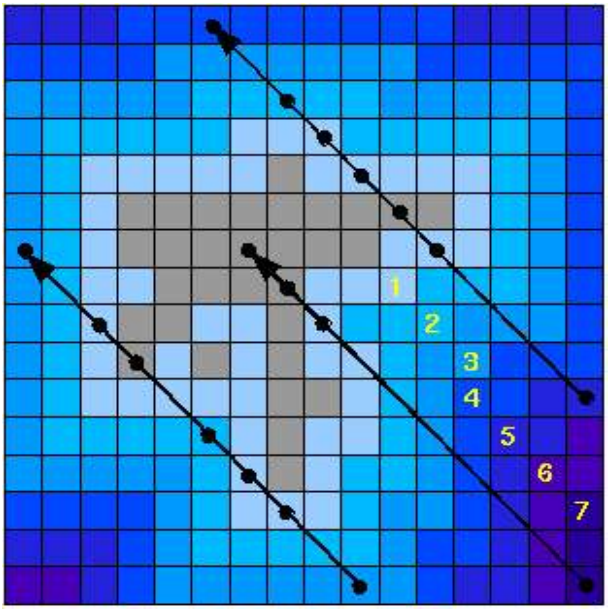


Adaptive ray traversal

The light gray squares denote voxels with vicinity flag turned on. The fast traversal algorithm is shown with the dashed gray lines, the precise algorithm with the solid black lines.

Proximity clouds:

This method is an extension of the approach described above. In a pre-processing step the distance to the nearest data voxel is computed for each empty voxel.



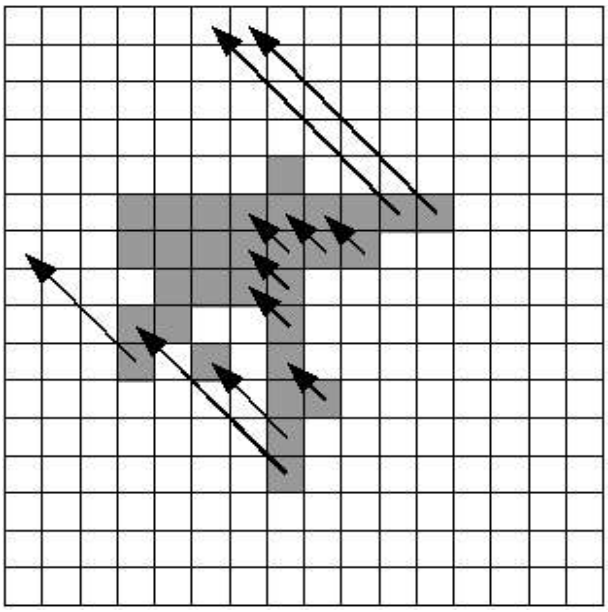
Proximity clouds

In the illustration above voxels with the same distance value are shown by different shades of blue. The sample points are represented by the black dots. When a ray encounters a proximity voxel with value *n* it can skip the next *n* voxels because there are no data voxels in between.

Homogeneity-acceleration: Homogeneous regions are approximated with fewer sample points. Therefore the mean values and variance are stored in the octree.

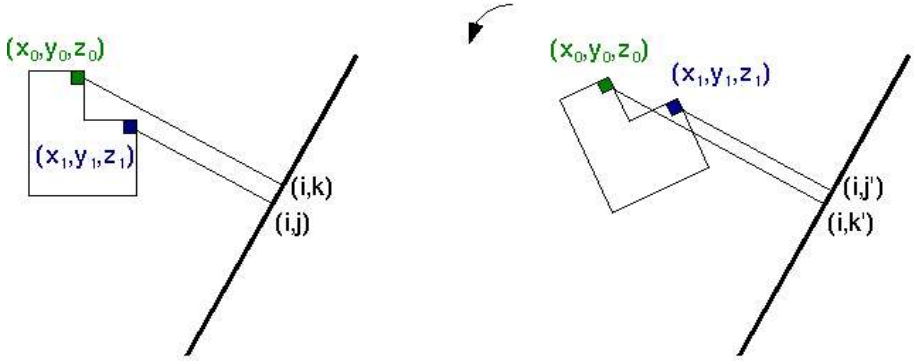
2.3 Exploiting Temporal Coherence in Volume Animations

For each pixel the coordinates of the first non-transparent voxel which is hit by the ray is stored in the so called *C-buffer* (Coordinates-buffer). The coordinates are computed after rendering the first frame and then updated continuously. This data can be used to determine where to start tracing the ray. This is illustrated in the image below.



Using the C-buffer

The C-buffer has to be updated correctly when the world space is rotated. Therefore the C-buffer coordinates are transformed by the rotation, i.e. the voxel coordinates [x,y,z] stored in the C-buffer[i,j] are mapped to C-buffer[i',j']. Now two problems may occur. First it is possible that a C-buffer entry remains empty. In this case the ray starts at the volume boundary as in the normal ray casting approach. The second problem is more complicated. It is possible that several coordinates are mapped to the same C-buffer entry. An example:



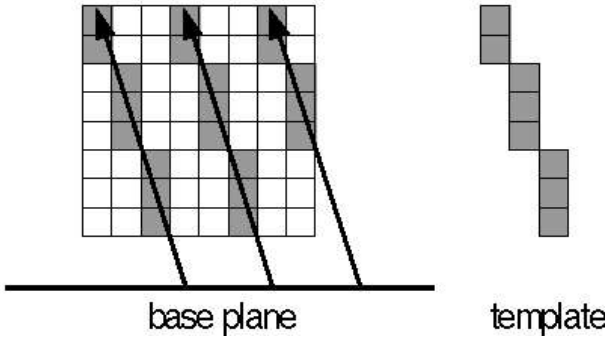
Rotation of the world space.

In the example above the coordinates of the voxel $[x_0, y_0, z_0]$ are stored in C-Buffer[i,j] and those of voxel $[x_1, y_1, z_1]$ in C-Buffer[i,k] with $k > j$. Now the world space is rotated around the X-axis. Therefore the C-Buffer entries remain both in the column i of the buffer, but the row order of the two projected voxels has changed, i.e. $j' \geqslant k'$. This is the indicator for voxels to be potentially hidden after a rotation. These entries possibly have to

be removed.

2.4 Template-Based Volume Viewing

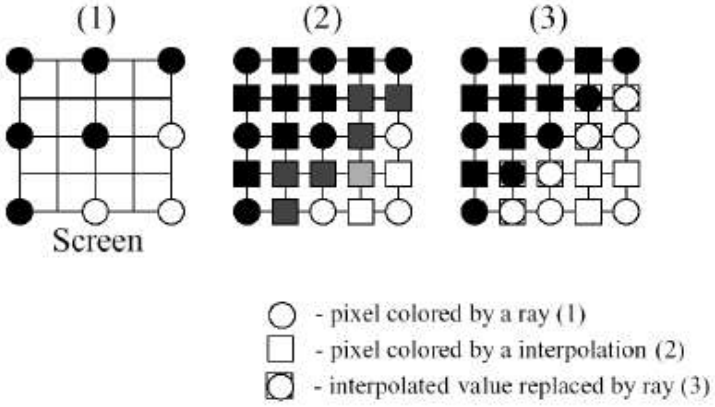
The **Template-Based Volume Viewing** approach by Yagel *[Yagel-1992-DRT]* stores the traversal steps through the volume as a *ray template* to accelerate the rendering process. The template is generated by 3D DDA algorithm and can be used for parallel rays (orthographic projection). The rays have to be traced from a base plane which is oriented parallel to the volume.



Template based volume viewing

2.5 Adaptive screen sampling

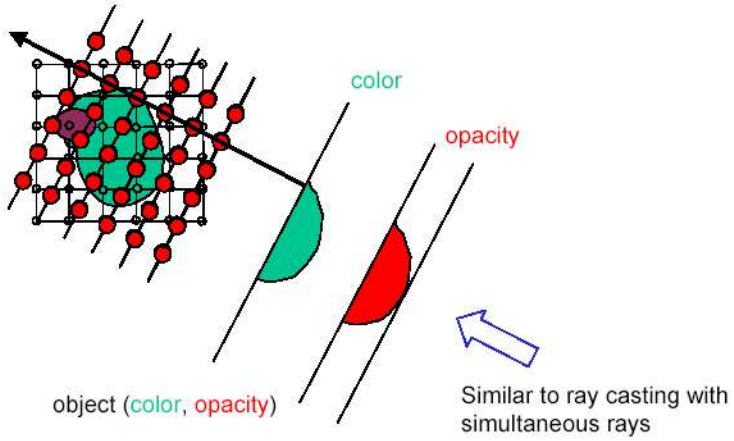
In some applications, rendering is performed by adaptively switching from one rendering method or sampling rate to another. **Adaptive screen sampling** introduced by Levoy *[Levoy-1990-HRT]* is well known in traditional ray tracing, where the number of rays emitted from a given pixel is adapted to the color change in a subset of pixels in a small neighborhood. This means that in areas with high value gradients additional rays are traced. One problem is that the rays occasionally miss the surface, in that case missing values are interpolated.



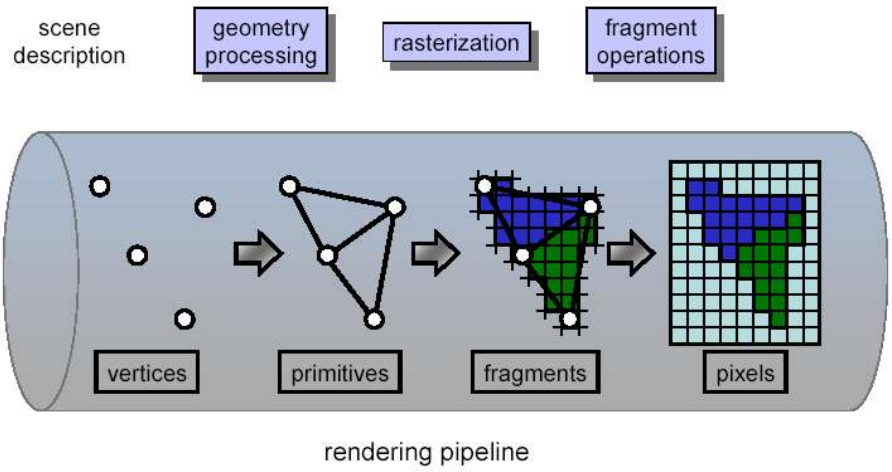
Adaptive screen sampling.

3 Texture-Based Volume Rendering

Texture-based volume rendering is an approach to generate an image viewed with different camera parameters using scene information extracted from a set of reference images. The rendering speed of this approach depends only on image size instead of scene complexity, and complex hand-made geometric models are not required. Further these techniques take advantage of functionality implemented on graphics hardware like Rasterization, Texturing and Blending.
Texture-based volume rendering is an object-space technique, since all calculations are done in object-space. This means that the rendering is accomplished by projecting each element onto the viewing plane such as to approximate the visual stimulus of viewing the element with regard to the chosen optical model. Two principal methods have been shown to be very effective in performing this task: *slicing* and *cell projection*.
We will only discuss the former method: Each element gets projected on a slice-by-slice basis. Therefore, the cross sections of each element and the current slicing plane are computed.



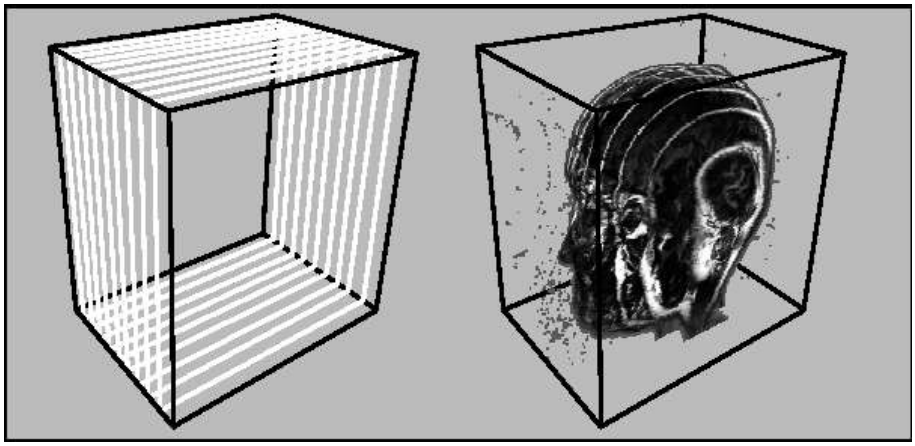
Texture-based volume rendering.



Example of a scene description passing through the rendering pipeline.

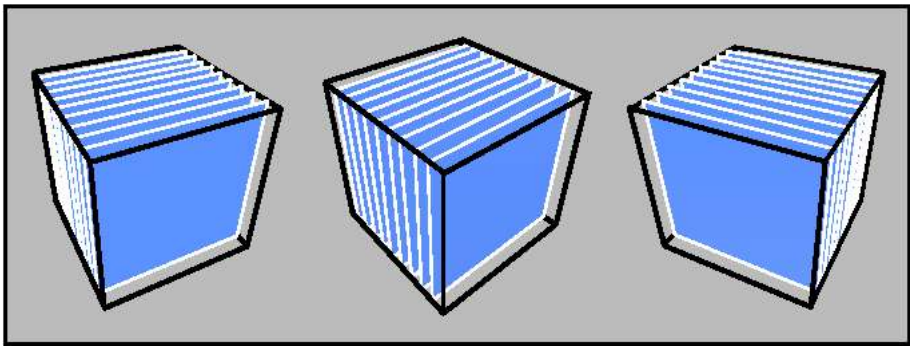
The computed slice has only a proxy geometry because there are no volumetric primitives

in graphics hardware. By sending the computed vertices of each slice down the rendering pipeline, geometric primitives are generated that get fragmented in the rasterization step to finally result in a pixel based texture map. All texture-mapped slices are stored in a stack and are blended back-to-front onto the image plane, which results in a semi-transparent view of the volume.



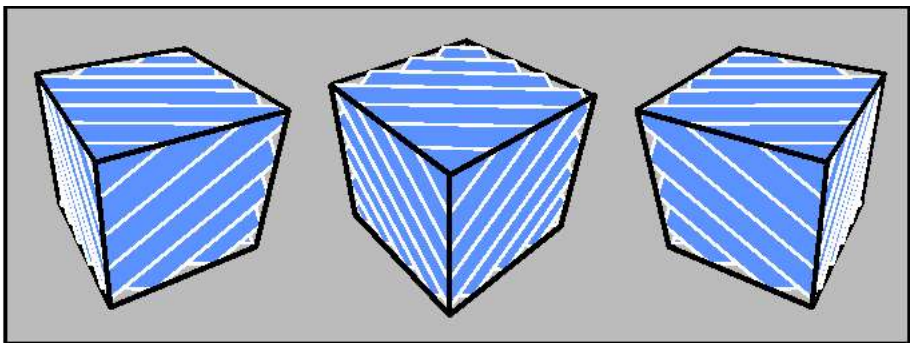
This image illustrates a stack of 2D textured slices and the proxy geometry that could be achieved with it.

The texture mapped 2D slices are object-aligned, because they get computed in object space parallel to the x-, y-, z-axis, unappreciated from which angle the viewer looks at the object. Due to correct visualization of the object from each possible viewing angle three stacks of 2D texture slices have to be computed. By doing this one can switch between the different stacks in case the viewing angle is close to 45 degrees to the slice normal. Otherwise the viewer would look on the slice edges and thus see through the volume. By projecting the voxels of the slice plane on pixels of the viewing plane, a pixel can lie in-between voxels, so its value can be found by bilinear interpolation.

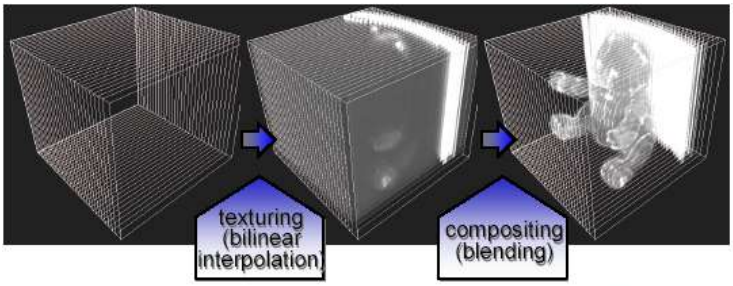


Axis aligned 2D textured slices.

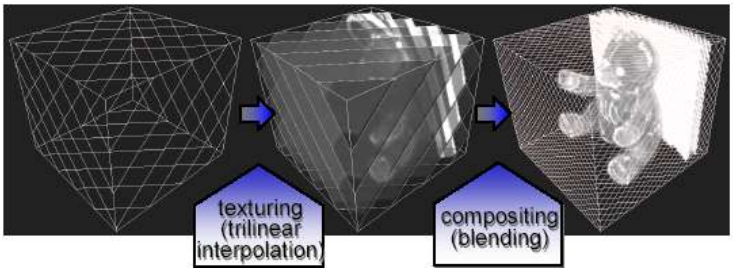
Another method is to compute view-aligned 3D textured slices. In contrast to the object aligned method, these slices are arranged with their surface normals in direction of the viewer. Here we only have to compute one stack of 3D textures, but for projection trilinear interpolation is needed to calculate values in-between. Further, if the object is rotated the slices have to be recomputed for each frame.



View aligned 3D textured slices .



2D textures
axis-aligned



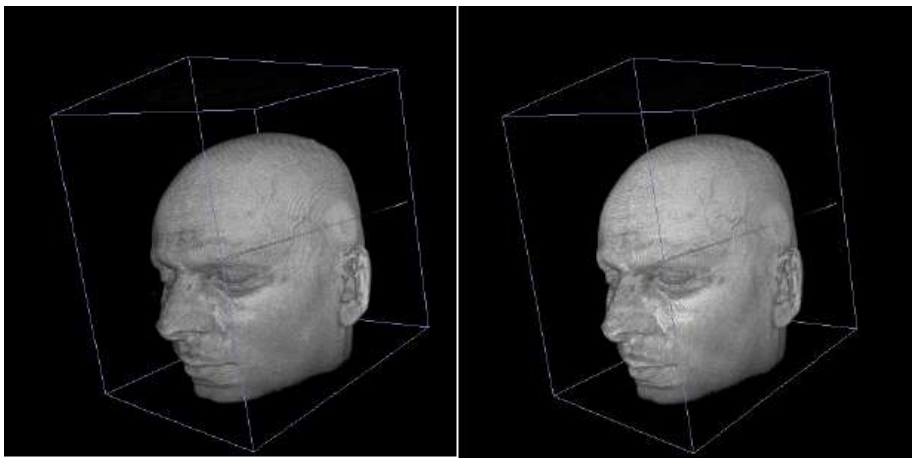
3D texture
view-aligned

Positive attributes of 2D textures:

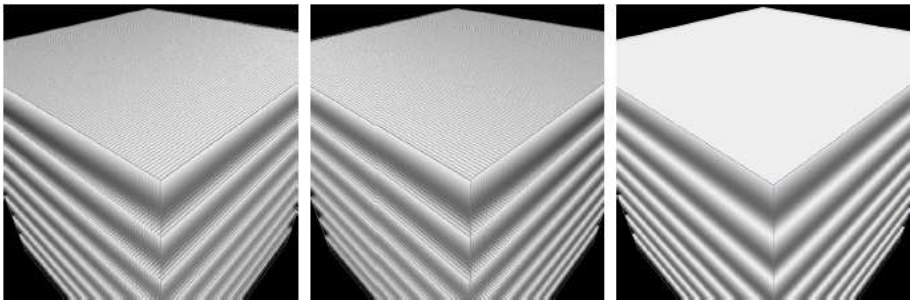
- This method works on older graphics hardware which does not support 3D textures
- Only bilinear interpolation within slices, no trilinear interpolation
    - fast
    - problems with image quality

Problems:

- Euclidean distance between slices along a light ray depends on viewing parameters
    - sampling distance depends on viewing direction
    - apparent brightness changes if opacity is not corrected
- Artifacts when stack is viewed close to 45 degrees



Change of brightness in 2D texture-based rendering.



Artifacts when stack is viewed close to 45 degrees.

Positive attributes of 3D textures:
- Needs graphics hardware support for 3D textures
- Trilinear interpolation within volume
    - slower
    - good image quality
- Constant Euclidean distance between slices along a light ray
- No artifacts due to inappropriate viewing angles

Render components
- Data setup
    - Volume data representation
    - Transfer function representation
- Data download
    - Volume textures
    - Transfer function tables
- Per-volume setup (once per frame)
    - Blending mode configuration
    - Texture unit configuration (3D)
    - (Fragment shader configuration)
- Per-slice setup
    - Texture unit configuration (2D)
    - Generate fragments
    - Proxy geometry rendering

Representation of volume data by textures:
- Stack of 2D textures
- One 3D texture

Typical choices for texture format:
- Luminance and alpha
    - Pre-classified (pre-shaded) gray-scale volume rendering
    - Transfer function is already applied to scalar data
    - Change of transfer function requires complete redefinition of texture data

- RGBA
    - Pre-classified (pre-shaded) colored volume rendering
    - Transfer function is already applied to scalar data

- Paletted texture
    - Index into color and opacity table (= palette)
    - Index size = byte
    - Index is identical to scalar value
    - Pre-classified (pre-shaded) colored volume rendering

- Transfer function applied to scalar data during runtime
- Simple and fast change of transfer functions
- OpenGL code for paletted 3D texture

```
glTexImage3D ( GL_TEXURE_3D, 0, GL_COLOR_INDEX8_EXT,
                    size_x, size_y, GL_COLOR_INDEX,
                    GL_UNSIGNED_BYTE, voldata);
```

- Representation of transfer function:
  - For paletted texture only
  - 1D transfer function texture = lookup table
  - OpenGL extensions required
  - OpenGL code

```
glColorTableEXT (GL_SHARED_TEXTURE_PALETTE_EXT,
                    GL_RGBA8, 256*4, GL_RGBA,
                    GL_UNSIGNED_BYTE, palette);
```

Compositing:
- Works on fragments
- Per-fragment operations
- After rasterization
- Blending of fragments via over operator
- OpenGL code for over operator

```
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Generate fragments:
- Render proxy geometry
- Slice
- Simple implementation: quadrilateral
- More sophisticated: triangulated intersection surface between slice plane and boundary of the volume data set

Advantages of texture-based rendering:
- Supported by consumer graphics hardware
- Fast for moderately sized data sets
- Interactive explorations
- Surface-based and volumetric representations can easily be combined
  - mixture with opaque geometries

Disadvantages of texture-based rendering:
- Limited by texture memory
  - Solution: bricking at the cost of additional texture downloads to the graphics board
- Brute force: complete volume is represented by slices
- No acceleration techniques like early-ray termination or space leaping
- Rasterization speed and memory access can be problematic

Outlook on more advanced texture-based volume rendering techniques:
- Exploit quite flexible per-fragment operations on modern GPUs (nVidia GeForce 3/4 or ATI Radeon 8500)
- Post-classification (post-shading) possible
- So-called pre-integration for very high-quality rendering
- Decompression of compressed volumetric data sets on GPU to save texture memory

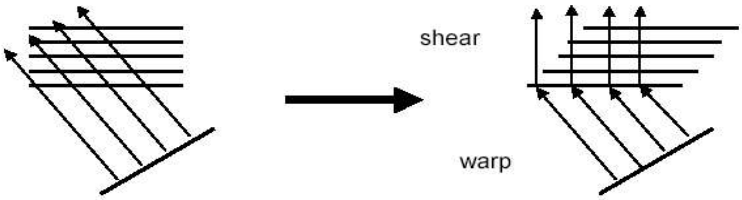
- Object-space method
- Slice-based technique
- Fast object-order rendering
- Accelerated volume visualization via shear-warp factorization [Lacroute & Levoy 1994]
- Software-based implementation

4 Shear-Warp Factorization

As described in the last chapter, algorithms that use spatial data structures can be divided into two categories according to the order in which the data structures are traversed: image-order or object-order.
Image-order algorithms operate by casting rays from each image pixel and processing the voxels along each ray (e.g. Ray Casting). This processing order has the disadvantage that the spatial data structure must be traversed once for every ray, resulting in redundant computation (e.g. Multiple descents of an octree). In contrast, object-order algorithms operate by splatting voxels onto the image while streaming through the volume data in storage order. We will deal with this technique more deeply in this chapter. However, this processing order makes it difficult to implement early ray termination, an effective optimization in ray-casting algorithms.

Lacroute ***[Lacroute-1994-FVR]*** presented an accelerated volume visualization algorithm via **Shear-Warp** factorization that combines the advantages of image-order and object-order algorithms.

The algorithm is software-based and works on the same slice-based technique as texture based volume rendering. Therefore it also needs 3 stacks of the volume along 3 principal axes. They are computed in object-space. The general goal is to make viewing rays parallel to each other and perpendicular to the image, which is achieved by a simple shear.



Shear along the volume slices.

Mathematical description of the shear-warp factorization:
Splitting the viewing transformation into separate parts

$$M_{view} = P \cdot S \cdot M_{warp}$$

where

M_{view} = general viewing matrix

P = permutation matrix: transpose the coordinate system in order to make the z-axis the principal viewing axis. (Note that P stands not for the Projection)

S = transforms volume into sheared object space

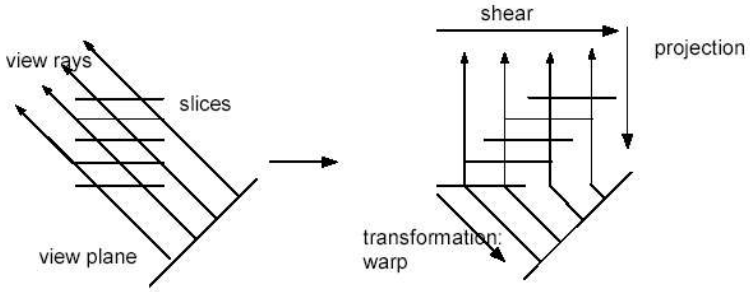
M_{warp} =warps sheared object coordinates into image coordinates

Shear S for parallel and perspective projections:

$$S_{par} = \begin{pmatrix} 1 & 0 & s_x & 0 \\ 0 & 1 & s_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S_{persp} = \begin{pmatrix} 1 & 0 & s'_x & 0 \\ 0 & 1 & s'_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & s'_w & 1 \end{pmatrix}$$

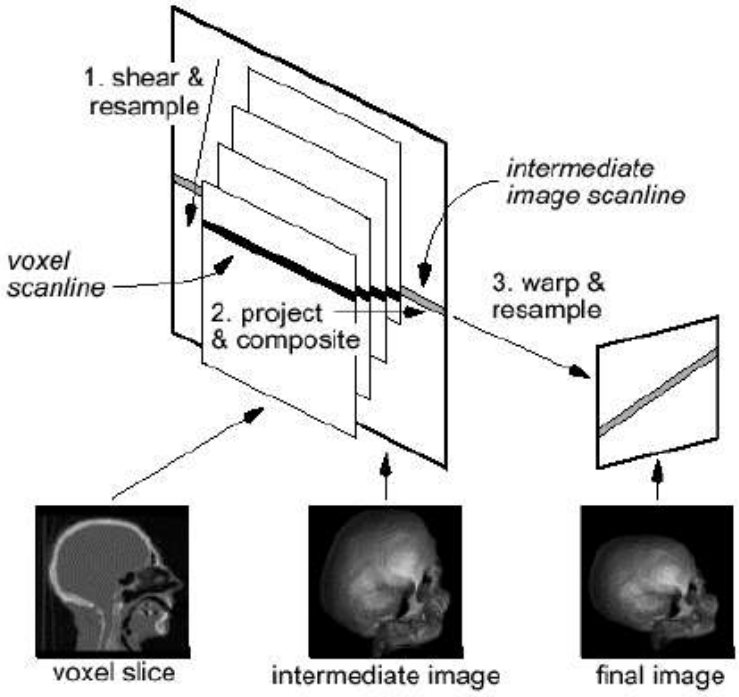
Algorithm:

1. Transform the volume data to sheared object space by translating and resampling each slice according to S . For perspective transformations, also scale each slice. P specifies which of the three possible slicing directions to use. For resampling bilinear interpolation is used.
2. Composite the resampled slices together in front-to-back order using the over operator. This step projects the volume into a 2D intermediate image in sheared object space. Here we can use early ray termination to accelerate the algorithm.
3. Transform the intermediate image to image space by warping it according to M_{warp} This second resampling step produces the correct final image.



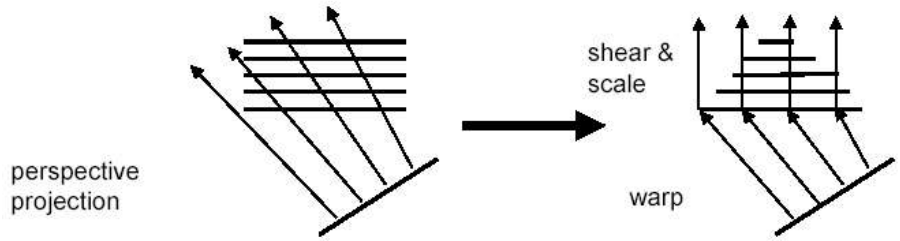
Shear along the volume slices, composite the resampled slices to an intermediate image and warp it to image space.

Example for one scan line:



After shearing slices, you have to resample them. Then the sample ray hits voxel center

after voxel center and no interpolation is needed anymore.
Most of the work in volume rendering has focused on parallel projections. However, perspective projections provide additional cues for resolving depth ambiguities. Perspective projections present a problem because the viewing rays diverge so it is difficult to sample the volume uniformly.



For a perspective projection each image slice has to be scaled after shearing and warping.

The Algorithm in a nutshell:

- Shear (3D)
- Project (3D → 2D)
- Warp (2D)

The Projection in sheared object space has several geometric properties that simplify the compositing step of the algorithm:

1. Scan lines of pixels in the intermediate image are parallel to scan lines of voxels in the volume data.
2. All voxels in a given voxel slice are scaled by the same factor. In different slices by different factors.
3. Parallel projections only: Every voxel slice has the same scale factor. The scale factor for parallel projections can be chosen arbitrary. But we choose a unity scale factor so that for a given voxel scan line there is a one-to-one mapping between voxels and intermediate image pixels.

Highly optimized algorithm for:

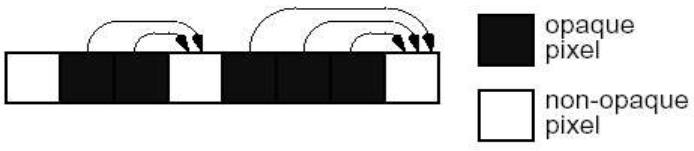
- Parallel projection and
- Fixed opacity transfer function (pre-shading), but if you change the transfer function interactively, all three slice stacks have to be recomputed.

Optimization of volume data (voxel scan lines):

- Run-length encoding of voxel scan lines
- Skip runs of transparent voxels
- Transparency and opaqueness determined by user-defined opacity threshold

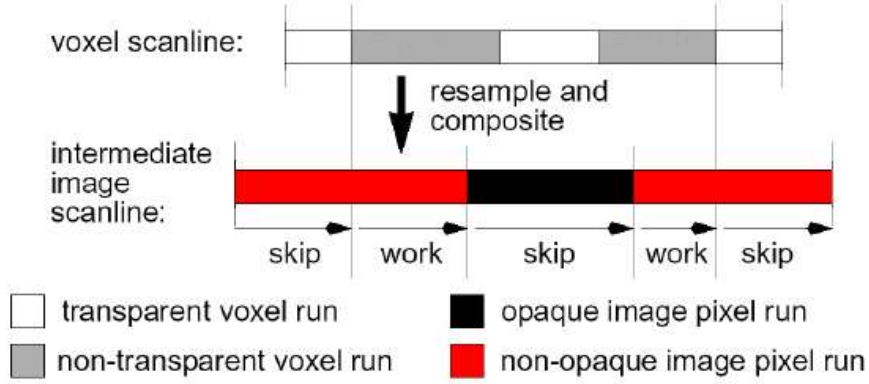
Optimization in intermediate image:

- Skip opaque pixels in intermediate image (analogously to early-ray termination)
- Store (in each pixel) offset to next non-opaque pixel (as shown in graphic below).



Offsets stored with opaque pixels in the intermediate image allow occluded voxels to be skipped efficiently.

By combining the both ideas of volume data and intermediate image optimization, we can exploit coherencies and make the algorithm very fast. As first property parallel scan lines are used for pixels and voxels. That means that voxel scan lines in sheared volume are aligned with pixel scan lines in intermediate image and both can be traversed in scan line order simultaneously. By using the run-length encoding of the voxel data to skip voxels which are transparent and the run-length encoding of the image to skip voxels which are occluded, we perform work only for voxels which are both non-transparent and visible.



Resampling and compositing are performed by streaming through both the voxels and the intermediate image in scanline order, skipping over voxels which are transparent and pixels which are opaque.

Coherence in voxel space:

- Each slice of the volume is only translated
- Fixed weights for bilinear interpolation within voxel slices
- Computation of weights only once per frame

Final warping:

- Works on composited intermediate image
- Warp: affine image warper with bilinear filter
- Often done in hardware: render a quadrilateral with intermediate 2D image being attached as 2D texture

Parallel projection:

- Efficient reconstruction
- Lookup table for shading
- Lookup table for opacity correction (thickness)
- Three RLE of the actual volume (in x, y, z)

Perspective projection:

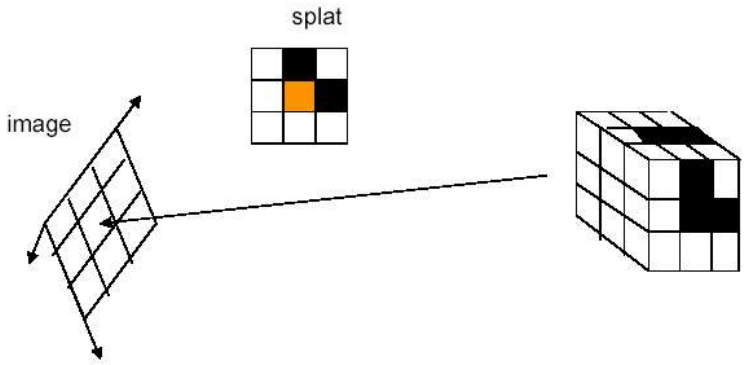
- Similar to parallel projection
- Viewing rays diverge in infinity
- Difference: voxels need to be scaled
- Hence more then two voxel scan lines needed for one image scan line
- Many-to-one mapping from voxels to pixels makes it slower than parallel projection.

5 Splatting

> **Splatting**
>
> Splatting is a volume rendering method that distributes volume data values across a region on the rendered image in terms of a distribution function.

Splatting is an object-order method described by [Westover, 1990], this method accumulates data points by "throwing" kernels for each voxel to the drawing plane. The footprints on the drawing plane represent the visualization. The kernel shape and size is critical for the quality of the result. The accumulation, or compositing process can be done back-to-front, which guarantees correct visibility. Front-to-back compositing is faster, because the process can be stopped when pixels are fully opaque. The original algorithm is fast but poor in quality, however since the first publication many improvements have been implemented. Note that good splatting results need as much CPU computing time as ray casting.

The basic idea of splatting is to project each sample (voxel) from the volume onto the image plane, which can be understood in such a way that each voxel is thrown on a plane and leaves it's footprint on it. For each thrown voxel we have to compute, which part of the image plane could be included.



The Splatting approach throws voxels on the image plane.

You can think of the volume (cloud) as each of the voxels is a measurement of a continuous scalar function. Ideally we want to reconstruct the continuous volume and not the discrete values. That's why we use the interpolation kernel w which should be spherically symmetric, because we don't want different results when looking from different directions. A 3D Gaussian convolution kernel would be a good choice.
The reconstruction can be done through:

$f_r(v)=\sum_k w(v-v_k)f(v_k)$ where

$f(v_k)$ is the density value at sample k and
$w(v-v_k)$ is the reconstruction kernel such as the Gaussian function.

Analytic integral along a ray *r* for intensity (emission):
$I(p)=\int f_r(p+r)dr=\int \sum_k w(p+r-v_k)f(v_k)dr$ where

$I(p)$ is the intensity for pixel p,
f_r is the scale intensity,
r is the ray and
$\sum_k \ldots$ is the sum of all voxels weighted by the kernel.

We can rewrite the formula by switching the summation and integral:

$$I(p)=\sum_k f(v_k)\cdot\int w(p+r-v_k)\,dr$$

with $\int w(p+r-v_k)\,dr$ as splatting kernel (= "splat") or footprint.

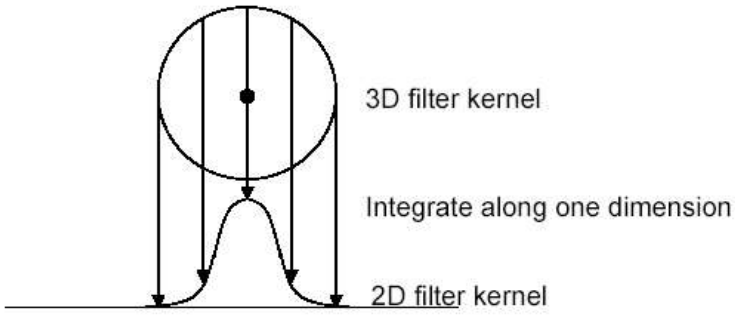
Discretization via 2D splats, where ray is in z-axis:

$$Splat(x,y)=\int w(x,y,z)\,dz$$

from the original 3D kernel. Often w is chosen as Gaussian

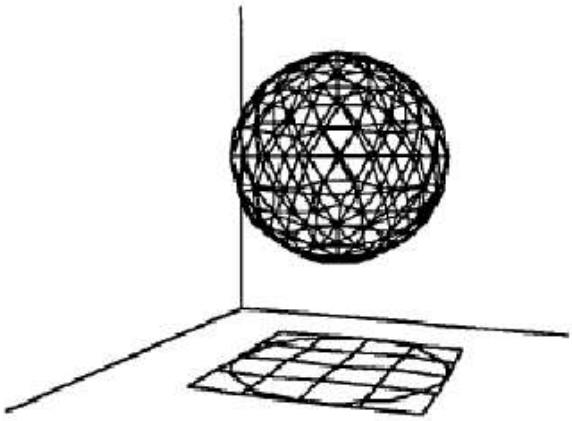
$$w(x,y,z)=d\exp(-x^2/a^2-y^2/b^2-z^2/c^2)$$

The 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel



The 2D filter kernel has the highest intensity at its center.

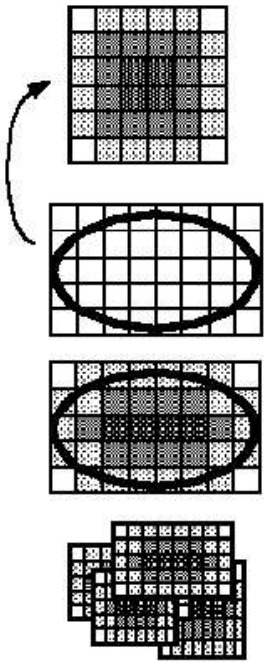
- Draw each voxel as a cloud of points (footprint) that spreads the voxel contribution across multiple pixels
- Footprint: splatted (integrated) kernel
- Approximate the 3D kernel *h*(*x,y,z*) extent by a sphere



Approximated 3D convolution kernel.

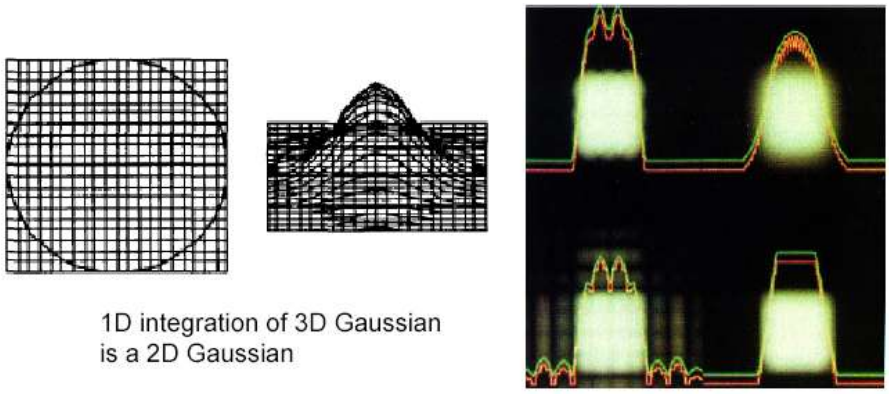
- Larger footprint increases blurring and is used for high pixel-to-voxel ratio.
- Smaller size of footprint generates gap.
- Footprint geometry:

    - Orthographic projection: footprint is independent of the view point.
    - Perspective projection: footprint is elliptical.
- The splat footprint, a function of x and y can be pre-computed and stored. Once this is done, the algorithm just needs to add many of the footprints for projection on image plane.
- For perspective projection: additional computation of the orientation of the ellipse.



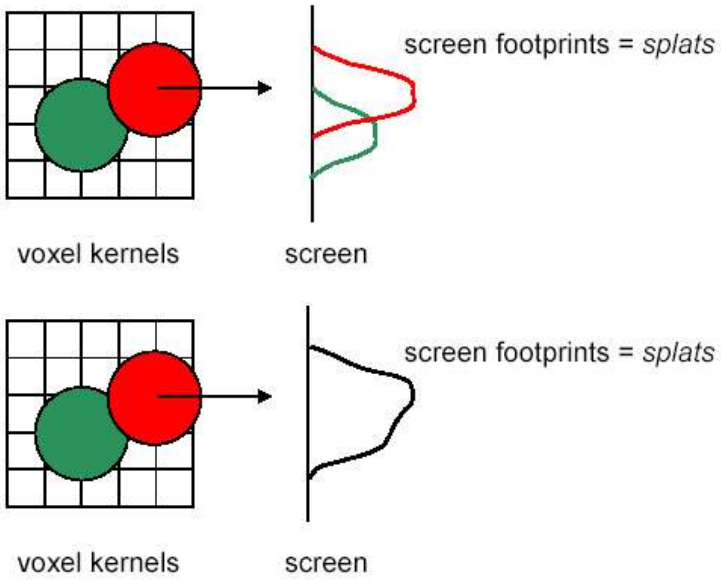
2D convolution kernel.

The choice of kernel can affect the quality of the image as you can see in the picture below. Examples are cone, Gaussian, sinc, and bilinear function.



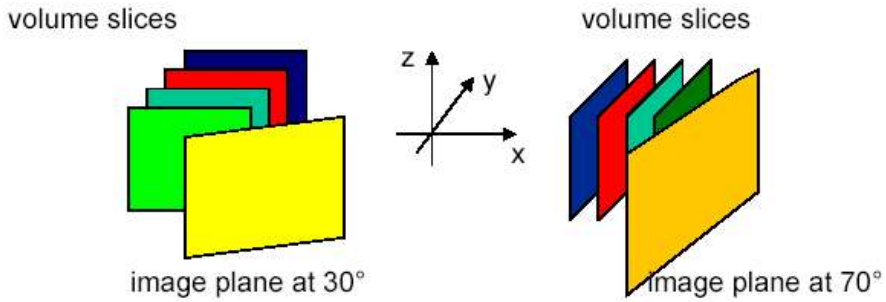
Example of the 2D Gaussian convolution kernel.

For splatting, each slice of the volume is filled with a field of 3D interpolation kernels, one kernel at each grid voxel. The reconstruction kernels might overlap and each of them leaves a 2D footprint on the screen, which is weighted and accumulated into the final image.



Splatts on the viewing plane.

Voxel kernels are added within each 2D sheet of the volume. The weighted footprints of the sheets are composited in front-to-back order. For projection of all sheets in the sheet-buffer we use volume slices most parallel to the image plane (analogously to stack of slices).



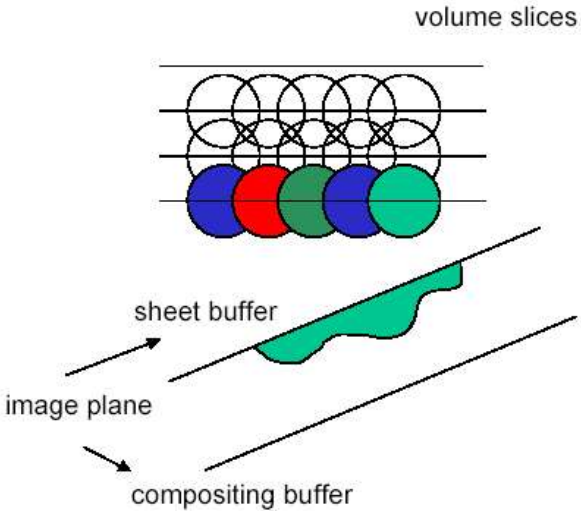
Which stack of volume slices is used for projection onto the image plane depends of the viewing angle.

Core algorithm for splatting:
Volume:

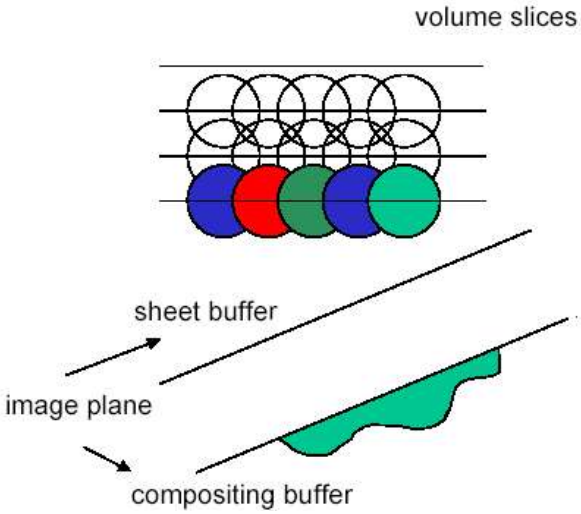
- Represented by voxels
- Slicing

Image plane:

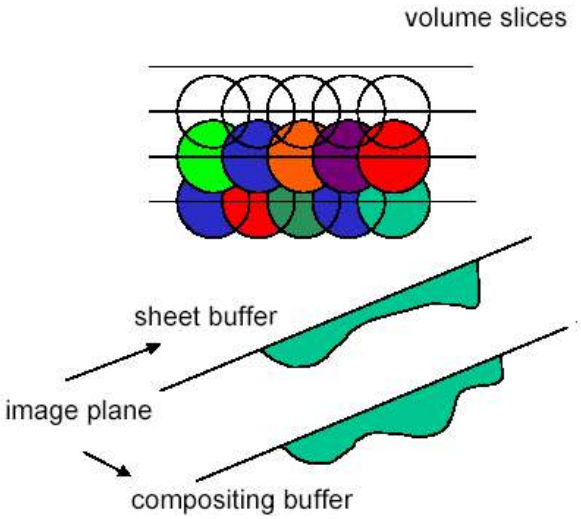
- Sheet buffer
- Compositing buffer



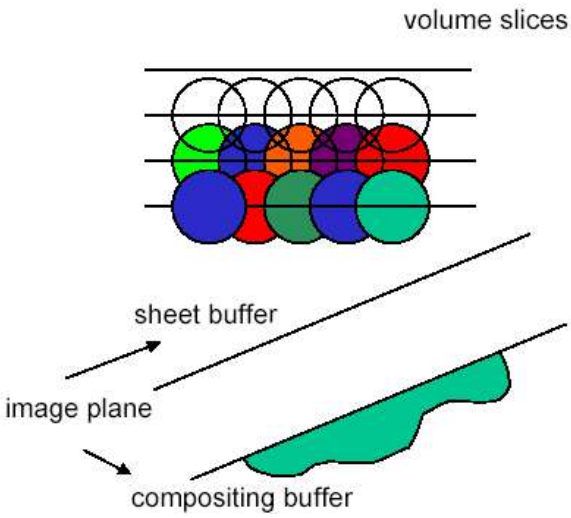
Add voxel kernels within first sheet to the sheet-buffer.



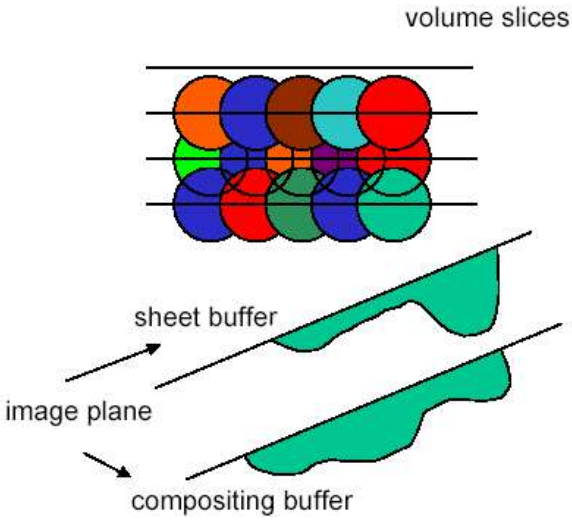
Transfer sheet buffer to compositing buffer.



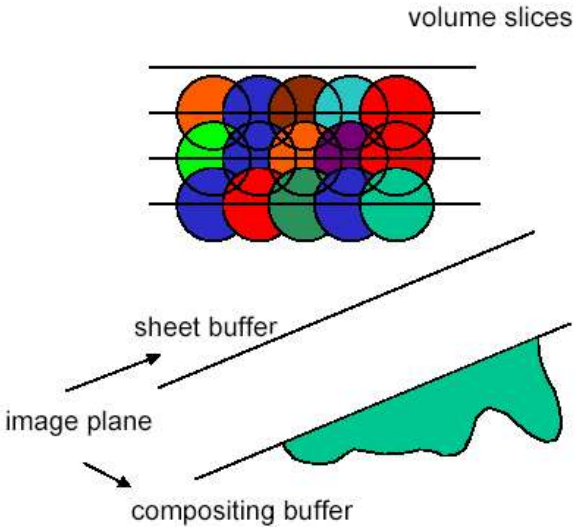
Add voxel kernels within second sheet to the sheet-buffer.



Composite sheet buffer with compositing buffer.

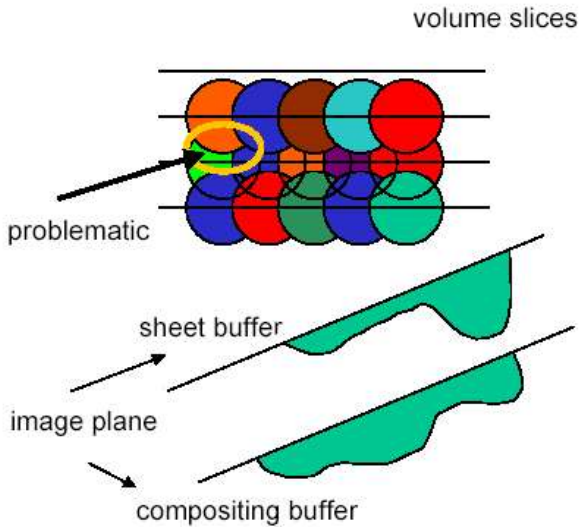


Add voxel kernels within third sheet to the sheet-buffer



Composite sheet buffer with compositing buffer.

A problem that leads to inaccurate compositing is when two or more splats overlap. In these regions we will have an incorrect mixture of integration (3D kernel to 2D splat) and compositing.



Overlapping kernels lead to aliasing.

Advantages:

- Footprints can be pre-integrated.
- Quite fast: voxel interpolation is in 2D on screen.
- Simple static parallel decomposition.
- Acceleration approach: only relevant voxels must be projected, air voxels for example can be disregarded.

Disadvantages:

- Blurry images for zoomed views, because Gaussian cuts away high frequencies.
- High fill-rate for zoomed splats.
- Reconstruction and integration to be performed on a per-splat basis.
- Dilemma when splats overlap

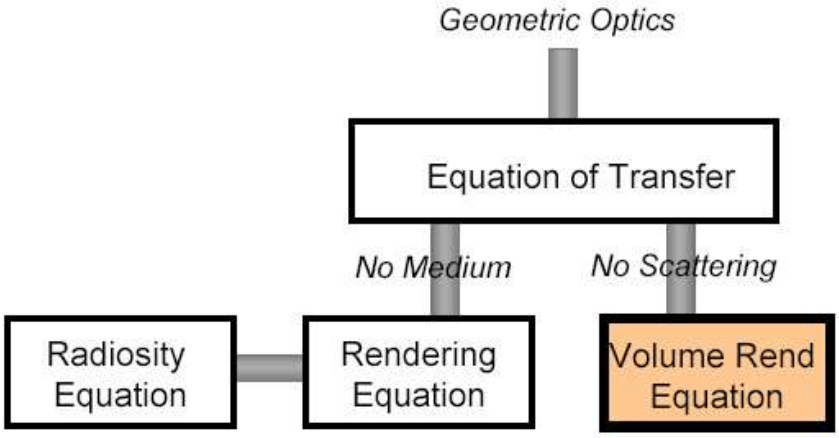
Simple extension to volume data without grids:

- Scattered data with kernels.
- Example: SPH (smooth particle hydrodynamics).
- Needs sorting of sample points.

6 Equation of Transfer for Light

All rendering models that have been described before are physically not correct. The goal is, to find a physical model for volume rendering. There are two main approaches, the **emission-absorption** model by Hege *[Hege-1993-MMA]* and the **density-emitter** model by Sabella *[Sabella-1988-RAV]*. Still there exist other, more general approaches like the **linear transport theory** or the **equation of transfer for light** by Kajiya *[Kajiya-1986-TRE]* which is the basis for all rendering methods. The equation tells you what happens to the radiation of the ray, when it moves through the volume. Some more important aspects are absorption, emission, scattering and participating mediums like gas, fire or semitransparent materials.

The Grand Scheme



Two ways to make the equation of transfer simpler:

1. *Changes can only appear by interaction with surfaces, if there is no medium (vacuum) it does not effect the ray and we can use the standard rendering equation. The radiance resides constant.*
2. *If there is a medium, we have absorbtion along the ray passing through the volume. But therefore with the emission-absorbtion model we have no scattering, which means no change of direction and no computation of secundary rays. In this case we can use the standard volume rendering equation.*

The basic idea of the equation of transfer for light is very much in the spirit of the radiosity equation, simply balancing the energy flows from one point of a surface to another. The equation states that the transport intensity of light from one surface point to another is simply the sum of the emitted light and the total light intensity which is scattered toward x from all other surface points. Further on, the more detailed explanations are based on the equation model by Hege *[Hege-1993-MMA]* .

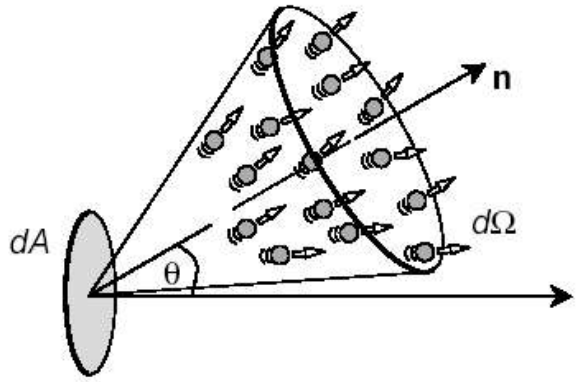
Assumptions:
- Based on a physical model for radiation
- Geometrical optics

Neglect:
- Diffraction
- Interference
- Wave-character
- Polarization

Interaction of light with matter at the macroscopic scale:
- Describes the changes of specific intensity due to absorption, emission, and scattering

Based on energy conservation on microscopic level (not on photon and electron level).

- Basic quantity of light is the radiance, sometimes also called specific intensity I .



Photons reflected on a surface.

dA n θ $d\Omega$

The equation of radiative energy tells us, if we have a surface element and radiation with specific direction n with solid angle, what the amount of energy moving through this area will be. Therefore the equation looks like:

$\delta E = I(x, n, \nu)\cos\theta \; dA \; d\Omega \; d\nu \; dt$ with

δE = amount of radiation energy, traveling in time dt with a frequency interval $d\nu$ around ν through a surface element dA into an element of solid angle $d\Omega$ in direction n .

I = radiance or specific intensity (6 dimensional: position: 2 dimensions (angles), direction: 3 dimensions, frequency: 1 dimension).

x = position

n = direction

ν = frequency

θ = angle between n and the normal on dA

$\cos\theta \; dA$ = projected area element

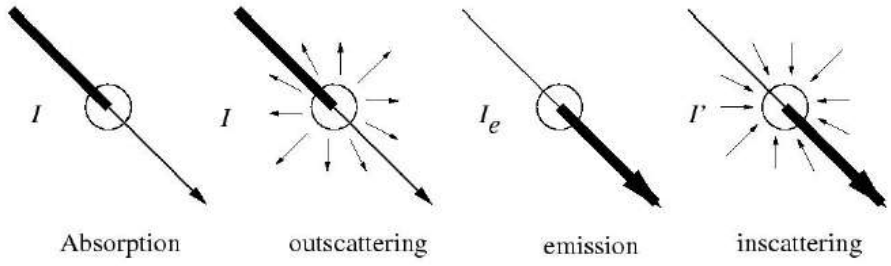
$d\Omega$ = solid angle

$d\nu$ = frequency

dt = time

There exist 4 responsible effects if there is a difference between incoming and outgoing radiance at a single position:

- *Absorption*: specific intensity comes in, less intensity goes out.
- *Emission*: less energy comes in, but more energy goes out.
- *Outscattering*: energy scatters out in all directions and can scatter in at another part.
- *Inscattering*: energy scatters in from other parts.



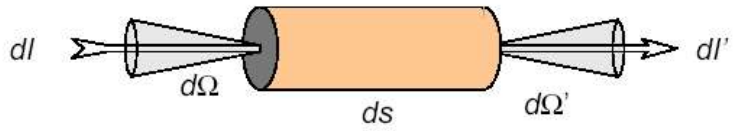
Contributions to radiation at a single position.

Absorption:
When radiation passes through material, energy is generally removed from the beam. This loss is described in terms of an *extinction coefficient* or *total absorption coefficient* $\chi(x,n,\nu)$.
The amount of energy removed from a beam with specific intensity $I(x,n,\nu)$, when passing through a cylindrical volume element of length ds and cross section da , is given by:

$\delta E^{(ab)} = \chi(x,n,\nu) I(x,n,\nu)\, ds\, dA\, d\Omega\, d\nu\, dt$ with

$\delta E^{(ab)}$	= absorption
$\chi(x,n,\nu)$	= absorption coefficient
χ	= scalar value or function

Notice that no cosine term appears in this expression. This is because the absorbing volume does not depend on the incident angle. The absorption coefficient generally is a function of x , n and z . The absorption coefficient is measured in units of m^{-1} . The term $\frac{1}{\chi}$ is also known as the *mean free path* of photons of frequency ν in material.

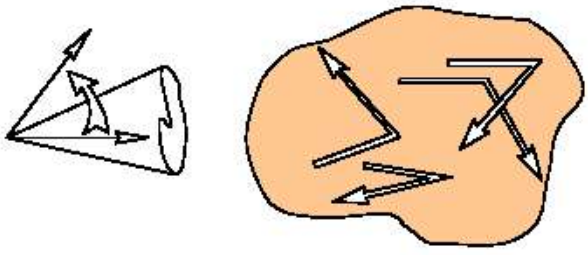


Removal of radiative energy by true absorption (conversion to thermal energy).

It is important to distinguish between true or thermal absorption and emission processes, and the process of scattering. In the former case, energy removed from the beam is converted into material thermal energy, and energy is emitted into the beam at the expense of material energy respectively. In contrast, in a scattering process a photon interacts with a scattering center and emerges from the event moving in a different direction, in general with a different frequency too. Therefore the total absorption coefficient consists of two parts:

- True absorption coefficient $\kappa(x,n,\nu)$
- Scattering coefficient $\sigma(x,n,\nu)$

total absorption coefficient; $\chi = \kappa + \sigma$
The ratio of scattering coefficient and total absorption coefficients albedo. An albedo of 1 means, that there will be no true absorption at all. This is the case of perfect scattering.



Scattering out of solid angle $d\Omega$.

Emission:
The emission coefficient $\eta(x,n,\nu)$ is defined in such a way, that the amount of radiant

energy within a frequency interval $d\nu$ emitted in time dt by a cylindrical volume element of length ds and cross section dA into a solid angle $d\Omega$ in a direction n is:

$$\delta E^{em} = \eta(x, n, \nu)\, ds\, dA\, d\Omega\, d\nu\, dt$$

In an analogous way as we did for absobtion, we break the total emission coefficient into two parts:

- Thermal part or source term $q(x, n, \nu)$
- Scattering part $j(x, n, \nu)$

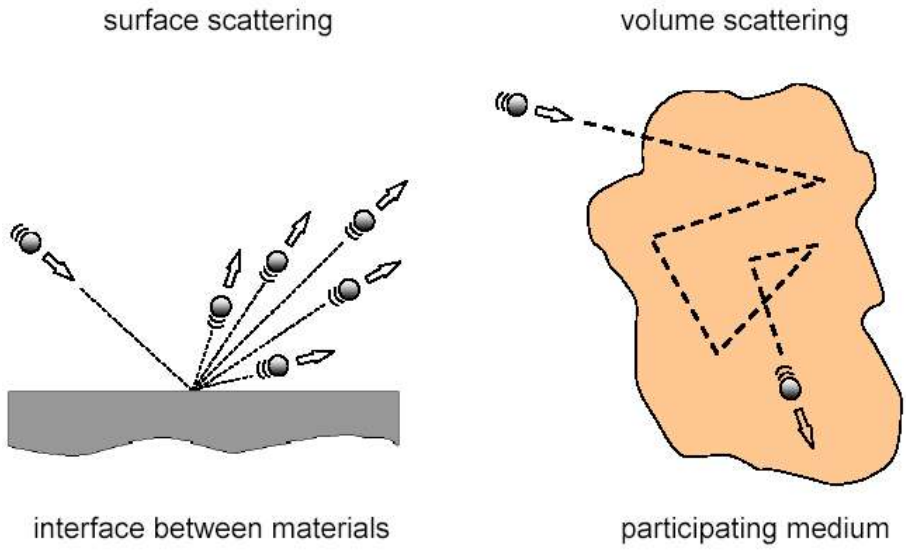
total emission coefficient: $\eta = q + j$

Scattering:
In order to take into account the angle dependence of scattering, we introduce a phase function $p(x, n, n', \nu, \nu')$. The amount of radiant energy scattered from frequency ν to frequency ν' and from direction n to direction n' , is given by:

$$\delta E^{(scat)} = \sigma I\, ds\, dA\, d\Omega\, d\nu\, dt \cdot \frac{1}{4\pi} p(x, n, n', \nu, \nu')\, d\Omega'\, d\nu' \quad \text{with}$$

σI = inscattered amount

It should be mentioned that there is no deeper meaning behind the factor $\frac{1}{4\pi}$. It simply cancels the factor 4π resulting from integrating a unity phase function over the sphere. Further in a scattering process a photon interacts with a scattering center and emerges from the event moving in a different direction, in general with a different frequency, too. If frequency doesn't change, one speaks of elastic scattering.



Scattering can appear, when photons hit a surface (rendering) or if they enter a participating medium (volume rendering).

Derivation of the Equation of Transfer:
The equation of transfer describes the change of specific intensity due to absorption, emission, and scattering. With all material constants given, the radiation field can be

calculated from this equation. Consider a cylindrical volume element. The difference between the amount of energy emerging at position $x+dx$ and the amount of energy incident at x must be equal to the difference between the energy created by emission and the energy removed by absorption. Thus we have:

$\{I(x,n,\nu)-I(x+dx,n,\nu)\}dA\ d\Omega\ d\nu\ dt$ (the difference can be one of the four effects)
$=\{-\chi(x,n,\nu)I(x,n,\nu)+\eta(x,n,\nu)\}ds\ dA\ d\Omega\ d\nu\ dt$ with

$-\chi(x,n,\nu)$ = loss of energy
$I(x,n,\nu)$ = radiance
$\eta(x,n,\nu)$ = emission

If we write $dx=n\,ds$ there follows the time-independent equation of transfer. Note that the emission coefficient in general contains a scattering part and thus an integral over I :

$n\cdot\nabla I=-\chi I+\eta$ with

$n\cdot\nabla I$ = with $\cdot$ as scalar product and ∇ as package of differentials
$-\chi I$ = losses
η = gains

This means that radiance along a ray in direction n changes due to absorption plus the emission. In vacuum for example $-\chi I=0$ and $\eta=0$, so we have no absorption and no emission which means the radiance does not change.

Writing out the equation of transfer:

$$n\cdot\nabla I=-(\kappa+\sigma)I+q+\frac{1}{4\pi}\iint\sigma(x,n',\nu')p(x,n',n,\nu',\nu)I(x,n',\nu')d\Omega'\,d\nu'$$ with

κ = true absorption
σ = scattering
q = true emission
$\iint\ldots$ = we integrate over all incoming scattering photons

Without frequency-dependency and without inelastic scattering:

$$n\cdot\nabla I=-(\kappa+\sigma)I+q+\frac{1}{4\pi}\int\sigma(x,n')p(x,n',n)I(x,n')d\Omega'$$

In this case the integro-differential equation is a time-independent equation of transfer. Integro-differential in general means that the equation contains at least one integral and a differential part. Here we have ∇I as differential on the one side and the integration over I itself from the inscattering part.

Boundary conditions for equation of transfer:
The equation of transfer alone does not describe the radiation field completely, it is only valid away from surfaces. So we have to specify some boundary conditions to completely describe what happens. This is also necessary in order to eliminate the constant terms arising from the integration of the gradient operator in the time-independent equation. In the most simple case we have *explicit boundary conditions* they stand for *emission of light* at boundaries and are independent of I itself. The other case is called *implicit boundary*

condition and stands for *reflection of light* at surfaces. A combination of explicit and implicit boundary conditions on boundary surface can be written:

$$I(x,n,\nu)=E(x,n,\nu)+\iint k(x,n',n,\nu',\nu)I^{(inc)}(x,n',\nu')d\Omega' d\nu' \quad \textit{with}$$

$E(x,n,\nu)$ = *emission*
$\iint k(x,n',n,\nu',\nu)$ = *surface scattering kernel*
$I^{(inc)}(x,n',\nu')$ = *incoming radiation*

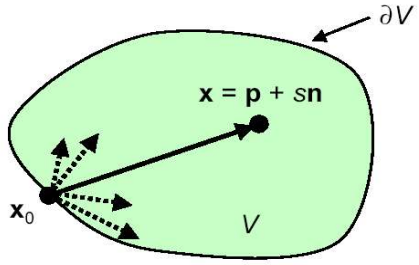
Integral form of the Equation of Transfer:
Instead of an integro-differential equation we can rewrite the equation of transfer to a pure integral equation:

$$n\cdot\nabla I(x,n,\nu)=-\chi(x,n,\nu)I(x,n,\nu)+\eta(x,n,\nu)$$

yields

$$\frac{\partial}{\partial s}I(x,n,\nu)=-\chi(x,n,\nu)I(x,n,\nu)+\eta(x,n,\nu)$$

Notice that the operator $n\cdot\nabla$ is the directional derivative along a line, $x=p+sn$ with p being some arbitrary reference point.



Derivative along a line with arbitrary reference point.
The optical depth between 2 points $x_1=p+s_1n$ and $x_2=p+s_2n$ is defined as:

$$\tau_\nu(x_1,x_2)=\int_{s_1}^{s_2}\chi(p+s'n,n,\nu)ds'$$

Recall that $\frac{1}{\chi}$ is the mean free path of a photon with frequency ν. Therefore we can interpret optical depth as the number of mean free paths between two points x_1 and x_2. Further optical depth serves as integrating factor $e^{\tau_\nu(x_0,x)}$ for equation of transfer and thus we can write:

$$\frac{\partial}{\partial s}I(x,n,\nu)\cdot e^{\tau_\nu(x_0,x)}=\eta(x,n,\nu)\cdot e^{\tau_\nu(x_0,x)}$$

This equation can be integrated and we obtain:

$$I(x,n,\nu)e^{\tau_\nu(x_0,x)}-I(x_0,n,\nu)=\int_{s_0}^{s}\eta(x',n,\nu)e^{\tau_\nu(x_0,x')}ds' \quad \text{with}$$

$\int_{s_0}^{s} \eta(x',n,\nu)$ = integral of total emission

$e^{\tau_\nu(x_0,x')}$ = integrating factor

x_0 = on the boundary surface

Making use of the fact that optical depth can be decomposed into $\tau_\nu(x_0,x)=\tau_\nu(x_0,x')+\tau_\nu(x',x)$ the final integral form of the equation of transfer can be written as follows:

$$I(x,n,\nu)=I(x_0,n,\nu)e^{-\tau_\nu(x_0,x)}+\int_{s_0}^{s}\eta(x',n,\nu)e^{-\tau_\nu(x',x)}ds'$$ with

$I(x_0,n,\nu)e^{-\tau_\nu(x_0,x)}$ = at any point of the volume there can be emission.

$\int_{s_0}^{s}\eta(x',n,\nu)e^{-\tau_\nu(x',x)}$ = if there is emission at a point in the volume it will be absorbed on its further way.

This equation can be viewed as the general formal solution of the time-independent equation of transfer. It states that the intensity of radiation traveling along n at point x is the sum of photons emitted from all points along the line segment $x'=p+s'n$, attenuated by the integrated absorptivity of the intervening material, plus an attenuated contribution from photons entering the boundary surface when it is pierced by that line segment. Generally the emission coefficient η will contain I itself.
Vacuum condition is an important special case of the transport equation, because there is no absorption, emission or scattering at all, except on surfaces. Further the frequency dependence can be ignored and the equation of transfer (inside the volume) is greatly simplified:

$$I(x,n)=I(x_0,n)$$

Rays incident on surface at x are traced back to some other surface element at x':

$$I^{(inc)}(x,n')=I(x',n')$$

If we substitute this into the generic boundary condition:

$$I(x,n,\nu)=E(x,n,\nu)+\iint k(x,n',n,\nu',\nu)I^{(inc)}(x,n',\nu')d\Omega'\,d\nu'$$

we obtain the famous rendering equation published by Kajiya *[Kajiya-1986-TRE]*:

$$I(x,n)=E(x,n)+\int k(x,n',n)I^{(inc)}(x,n')d\Omega' \quad , \quad x\in S$$ with

S = surface

Rewriting the element of solid angle $d\Omega'$ into a more familiar form

$$d\Omega'=\frac{\cos\theta'_\tau}{|x-x'|}dA'$$

leads to the standard form via BRDF (bidirectional reflection distribution function):

$$k(x, n', n) = f_r(x, n', n)\cos\theta_i$$

Special case for most volume rendering approaches:

- Emission-absorption model
- Density-emitter model *[Sabella-1988-RAV]*
- Volume filled with light-emitting particles
- Particles described by density function

If we ignore scattering in volume rendering equation we are left with the so called **emission-absorption model** or **density-emitter model**. One can imagine the volume to be filled with small light emitting particles. Rather than being modeled individually, these particles are described by some density function.
This leads to some **Simplifications:**

- No scattering
- Emission coefficient consists of source term only: $\eta = q$ (true emission)
- Absorption coefficient consists of true absorption only: $\chi = \kappa$
- No mixing between frequencies (no inelastic effects), so we can ignore any ν

Let us consider a ray of light traveling along a direction n, parametrized by a variable s. The ray enters the volume at position s_0. Suppressing the argument n, the equation of transfer in integral form reads:

$$I(s) = I(s_0)e^{-\tau(s_0, s)} + \int_{s_0}^{s} q(s')e^{-\tau(s', s)}ds' \quad \text{with}$$

$I(s_0)$ = incoming radiance is usually given by the boundary conditions.
$e^{-\tau(s_0, s)}$ = absorption
$q(s')$ = emission inside
$e^{-\tau(s', s)}$ = absorption on the way out

with optical depth

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)ds$$

This equation is much simpler because we are only interested in the radiance along the ray without any scattering. In order to solve this equation one has to discretize along the ray. There is no need to use equidistant steps, though this is the most common used procedure. If we divide the range of integration into n intervals , then the specific intensity at position s_k is related to the specific intensity at position s_{k-1} by the identity. So the **discretization of volume rendering equation** can be written:

$$I(s_k) = I(s_{k-1})e^{-\tau(s_{k-1}, s_k)} + \int_{s_{k-1}}^{s_k} q(s)e^{-\tau(s, s_k)}ds \quad \text{with}$$

$e^{-\tau(s_{k-1}, s_k)}$ = attenuation between s_{k-1} and s_k
$q(s)$ = emission term with q, the color which is emitted

For abbreviation we define two parts:

- Transparency part: $\theta_k = e^{-\tau(s_{k-1}, s_k)}$
- Emission part: $b_k = \int_{s_{k-1}}^{s_k} q(s) e^{-\tau(s, s_k)}$

So specific intensity at s_n can now be written as:

$$I(s_n) = I(s_{n-1})\theta_n + b_n - \sum_{k=0}^{n} (b_k \prod_{j=k+1}^{n} \theta_j) = I(S_{n-1}) \cdot (1 - \alpha_n) + b_n$$

This formula describes the entire physics of radiation transport, with the sum of all things that get emitted and the product of all transparencies (this is the over operator from ray casting). More exact the quantity θ_k is called the transparency of the material between s_{k-1} and s_k. Transparency is a number between 0 and 1. An infinitely optical depth $\tau(s_{k-1}, s_k)$ corresponds to a vanishing transparency $\tau_k = 0$. On the other hand a vanishing optical depth corresponds to full transparency $\tau_k = 1$. Alternatively one often uses opacity, defined as $\alpha_k = 1 - \tau_k$.

7 Compositing Schemes

As a matter of course there exist variations of **composition schemes** other then the over operator:

- First
- Average
- Maximum intensity projection
- Accumulate

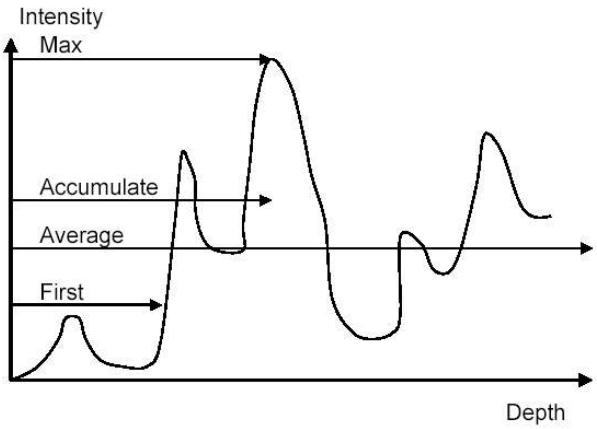
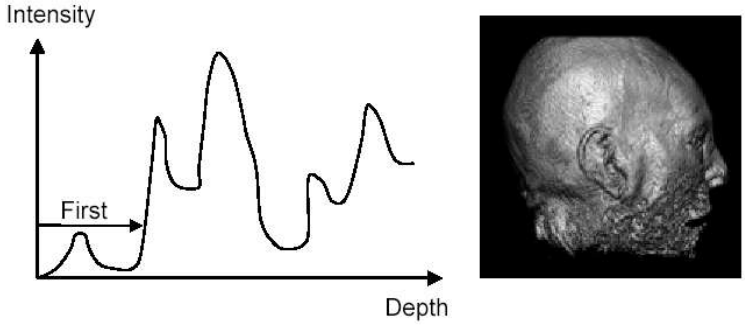


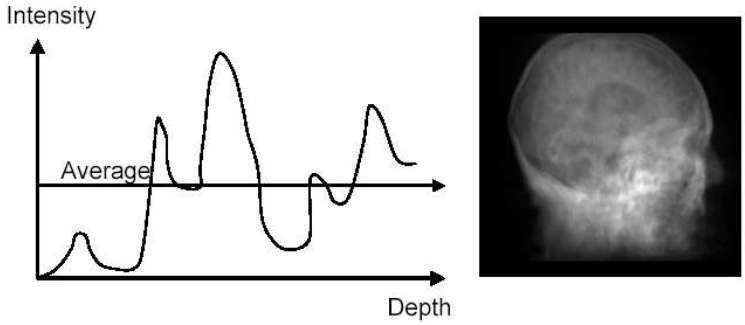
Illustration of different styles of compositing schemes.

The composite first hit works as follows: Send a ray into a volume, if the scalar value at a point is higher then the threshold use this one to extract isosurfaces and render them.



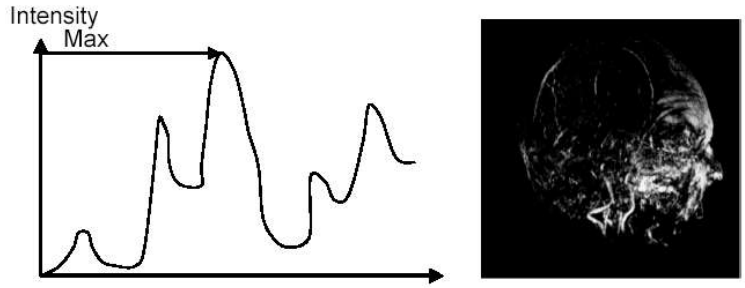
For the first value above a specified threshold is used for rendering.

Average or linear compositing produces basically an X-ray picture and takes no opacity into account.



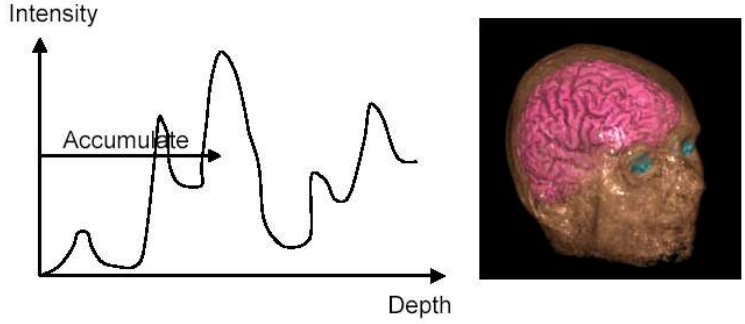
The result of average compositing looks like an X-ray picture.

Maximum Intensity Projection (MIP) uses the greatest value for rendering. It is often used for magnetic resonance angiograms or to extract vessel structures from an CT or MRT scan.



MIP emphasises structures with high intensities.

Accumulate compositing is similar to the over operator and the emission-absorption model where we have early ray termination. Intensity colors are accumulated until enough opacity is collected. This makes transparent layers visible (see volume classification).



Accumulation stops if enough opacity is collected.

8 Non-uniform Grids

- Resampling approaches, adaptive mesh refinement (AMR)
- Cell projection for unstructured (tetrahedral) grids
- Shirley-Tuchman *[Shirley-1990-PAD]*