Scalar Field Visualization
Volume Rendering
Iso-surfacing is limited

- Iso-surfacing is "binary"
  - point inside iso-surface?
  - voxel contributes to image?
- Is a hard, distinct boundary necessarily appropriate for the visualization task?
Iso-surfacing is Limited

- Iso-surfacing poor for ...
  - measured, "real-world" (noisy) data
  - amorphous, "soft" objects

virtual angiography  bovine combustion simulation
What is Direct Volume Rendering

• Any rendering process which maps from volume data to an image without introducing binary distinctions / intermediate geometry

• How do you make the data visible? : Color and Opacity

• Directly get a 3D representation of the volume data
  – The data is considered to represent a semi-transparent light-emitting medium
    • Also gaseous phenomena can be simulated
  – Approaches are based on the laws of physics (emission, absorption, scattering)
  – The volume data is used as a whole (look inside, see all interior structures)
Volume Rendering Usefulness

- Measured sources of volume data
  - CT (computed tomography)
  - PET (positron emission tomography)
  - MRI (magnetic resonance imaging)
  - Ultrasound
  - Confocal Microscopy
Volume Rendering Usefulness

• Synthetic sources of volume data
  CFD (computational fluid dynamics)
  Voxelization of discrete geometry
Data Representation

• Volume rendering techniques
  – depend strongly on the grid type
  – exist for structured and unstructured grids
  – are predominantly applied to uniform grids (3D images).
  – for uniform grid, voxels are the basic unit

• Cell-centered data for uniform grids
  – are attributed to cells (pixels, voxels) rather than nodes
  – can also occur in (finite volume) CFD datasets
  – are converted to node data
    • by taking the dual grid (easy for uniform grids, n cells -> n-1 cells!)
    • or by interpolating.
Concepts

• Interpolation
  – trilinear common, others possible
• Color and opacity transfer function
  – Turning scalar value to colors
• Gradient
  – direction of fastest change
• Compositing
  – "over operator"
Color and Opacity Transfer Functions

• \( C(p), \alpha(p) \) – \( p \) is a point in volume

• Functions of input data \( f(p) \)
  – \( C(f), \alpha(f) \) – these are 1D functions
  – Can include lighting affects
    • \( C(f, N(p), L) \) where \( N(p) = \text{grad}(f) \)
  – Derivatives of \( f \)
    • \( C(f, \text{grad}(f)), \alpha(f, \text{grad}(f)) \)
Transfer Functions (TFs)

Map data value $f$ to color and opacity

Shading, Compositing…
Gradient
\n\[ \nabla f = (df/dx, df/dy, df/dz) = \left( \frac{(f(1,0,0) - f(-1,0,0))}{2}, \right. \\
\left. \frac{(f(0,1,0) - f(0,-1,0))}{2}, \right. \\
\left. \frac{(f(0,0,1) - f(0,0,-1))}{2} \right) \]

Approximates "surface normal" (of iso-surface)
Pipelines: Iso vs. Vol Ren

- The standard line - "no intermediate geometric structures"

![Diagram showing the process of isosurface extraction and volume rendering.]
Computational Strategies

• How can the basic ingredients be combined:
  • Image Order
    • Ray casting (many options)
  • Object Order
    • splatting, texture-mapping
  • Combination (neither)
    • Shear-warp, Fourier
Computational Strategies

• How can the basic ingredients be combined:
  • Image Order
    • Ray casting (many options)
  • Object Order
    • splatting, texture-mapping
  • Combination (neither)
    • Shear-warp, Fourier
Image Order

• Render image one pixel at a time

For each pixel ...
- cast ray
- interpolate
- transfer function
- composite
Raycasting

• Raycasting is historically the first volume rendering technique.

• It has common with raytracing:
  – image-space method: main loop is over pixels of output image
  – a view ray per pixel (or per subpixel) is traced backward
  – samples are taken along the ray and composited to a single color

• Differences are:
  – no secondary (reflected, shadow) rays
  – transmitted ray is not refracted
  – more elaborate compositing functions
  – samples are taken at intervals (not at object intersections)
Raycasting

Sampling interval can be fixed or adjusted to voxels:

- Uniform sampling
- Voxel-by-voxel traversal (faster)

Connectedness of "voxelized" rays:

- 6-connected (strongest)
- 18-connected
- 26-connected (weakest)
Ray Templates

A ray template (Yagel 1991) is a voxelized ray which by translating generates all view rays.

Ray templates speed up the sampling process, but are obviously restricted to orthographic views.

**Algorithm:**

- Rename volume axes such that z is the one "most orthogonal" to the image plane.
- Create ray template with 3D version of line pixelized algorithm, giving 26-connected rays which are functional in z coordinate (have exactly one voxel per z-layer)
- Translate ray template in base plane, not in image plane
Ray Templates

Incorrect: translated in image plane

Correct: translated in base plane

base plane

image plane

image plane
Compositing

Two simple compositing functions can be used for previewing:

- **Maximum intensity projection (MIP):**
  - maximum of sampled values
  - result resembles X-ray image

- **Local maximum intensity projection (LMIP):**
  - first local maximum which is above a prescribed threshold
  - approximates occlusion
  - faster & better(!)
Compositing

Comparison of techniques (Y. Sato, dataset of a left kidney): Iso-surface vs. raycasting with MIP, LMIP, α-compositing
α-compositing

Assume that each sample on a view ray has color and opacity:

\((C_0, \alpha_0), \ldots, (C_N, \alpha_N)\) \quad \alpha_i \in [0,1], \alpha_i \in [0,1]

where the 0th sample is next to the camera
and the Nth one is a (fully opaque) background sample:

\(C_N = (r, g, b)_{\text{background}}\)
\(\alpha_N = 1\)

α-compositing can be defined recursively:

Let \(C_f^b\) denote the composite color of samples \(f, f+1, \ldots, b\).

Recursion formula for back-to-front compositing:

\[
C_b = \alpha_b C_b
\]
\[
C_f^b = \alpha_f C_f + (1 - \alpha_f) C_{f+1}^b
\]
\[
\alpha_f^b = \alpha_f + (1 - \alpha_f) \alpha_{f+1}^b
\]
\( \alpha \)-compositing

The first few generations, written with transparency \( T_i = 1 - \alpha_i \)

\[
\begin{align*}
C^b_b &= \alpha_b C_b \\
C^b_{b-1} &= \alpha_{b-1} C_{b-1} + \alpha_b C_b T_{b-1} \\
C^b_{b-2} &= \alpha_{b-2} C_{b-2} + \alpha_{b-1} C_{b-1} T_{b-2} + \alpha_b C_b T_{b-1} T_{b-2} \\
C^b_{b-3} &= \alpha_{b-3} C_{b-3} + \alpha_{b-2} C_{b-2} T_{b-3} + \alpha_{b-1} C_{b-1} T_{b-2} T_{b-3} + \alpha_b C_b T_{b-1} T_{b-2} T_{b-3}
\end{align*}
\]

reveal the closed formula for \( \alpha \)-compositing:

\[
C^b_f = \sum_{i=f}^{b} \alpha_i C_i \prod_{j=f}^{i-1} T_j
\]
α-compositing

front-to-back compositing can be derived from the closed formula:
Let $T_f^b$ denote the composite transparency of samples $f,f+1,\ldots,b$

$$T_f^b = \prod_{j=f}^{b} T_j$$

Then the simultaneous recursion for front-to-back compositing is:

$$C_f^f = \alpha_f C_f$$
$$T_f^f = 1 - \alpha_f$$

$$C_f^{b+1} = C_f^b + \alpha_{b+1} C_{b+1} T_f^b$$
$$T_f^{b+1} = (1 - \alpha_{b+1}) T_f^b$$

Advantage of front-to-back compositing: early ray termination when composite transparency falls below a threshold.
Compositing Example I

\[ c_f = (0,1,0) \]
\[ a_f = 0.4 \]

\[ c_b = (1,0,0) \]
\[ a_b = 0.9 \]

\[ c = a_f \times c_f + (1 - a_f) \times a_b \times c_b \]
\[ a = a_f + (1 - a_f) \times a_b \]

\[ c_{\text{red}} = 0.4 \times 0 + (1-0.4) \times 0.9 \times 1 = 0.6 \times 0.9 = 0.54 \]
\[ c_{\text{green}} = 0.4 \times 1 + (1-0.4) \times 0.9 \times 0 = 0.4 \]
\[ c_{\text{blue}} = 0.4 \times 0 + (1-0.4) \times 0.9 \times 0 = 0 \]
\[ a = 0.4 + (1 - 0.4) \times (0.9) = 0.4 + 0.6 \times 0.9 \]

\[ c = (0.54,0.4,0) \]
\[ a = 0.94 \]
Compositing Example II

\[ c = a_f * c_f + (1 - a_f) * a_b * c_b \]
\[ a = a_f + (1 - a_f) * a_b \]

\[ c_f = (0,1,0) \]
\[ a_f = 0.4 \]
\[ a = 0.4 + (1 - 0.4) * 0.9 = 0.964 \]
\[ c_b = (1,0,0) \]
\[ a_b = 0.9 \]
\[ c_b = (0.54,0.4,0) \]
\[ a = 0.4 + (1 - 0.4) * 0.94 = 0.964 \]

\[ c_{\text{red}} = 0.4 * 0 + (1 - 0.4) * 0.94 * 0.54 = 0.6 * 0.94 * 0.54 = 0.30 \]
\[ c_{\text{green}} = 0.4 * 1 + (1 - 0.4) * 0.94 * 0.4 = 0.6 * 0.94 * 0.4 = 0.23 \]
\[ c_{\text{blue}} = 0.4 * 1 + (1 - 0.4) * 0.94 * 0 = 0.4 \]
\[ a = 0.4 + (1 - 0.4) * 0.94 = 0.964 \]

\[ c = (0.3,0.23,0.4) \]
\[ a = 0.964 \]
Compositing Example II

\[ c = a_f \cdot c_f + (1 - a_f) \cdot a_b \cdot c_b \]
\[ a = a_f + (1 - a_f) \cdot a_b \]

\[ c_f = (0, 1, 1) \]
\[ a_f = 0.4 \]

\[ c_b = (1, 0, 0) \]
\[ a_b = 0.9 \]

\[ c_{red} = 0.4 \cdot 0 + (1 - 0.4) \cdot 0.9 \cdot 1 = 0.6 \cdot 0.9 = 0.54 \]
\[ c_{green} = 0.4 \cdot 1 + (1 - 0.4) \cdot 0.9 \cdot 0 = 0.4 \]
\[ c_{blue} = 0.4 \cdot 1 + (1 - 0.4) \cdot 0.9 \cdot 0 = 0.4 \]
\[ a = 0.4 + (1 - 0.4) \cdot (0.9) = 0.4 + 0.6 \cdot 0.9 \]

\[ c_{red} = 0.4 \cdot 0 + (1 - 0.4) \cdot 0.94 \cdot 0.54 = 0.6 \cdot 0.94 \cdot 0.54 = 0.30 \]
\[ c_{green} = 0.4 \cdot 1 + (1 - 0.4) \cdot 0.94 \cdot 0.4 = 0.6 \cdot 0.94 \cdot 0.4 = 0.23 \]
\[ c_{blue} = 0.4 \cdot 1 + (1 - 0.4) \cdot 0.94 \cdot 0.4 = 0.23 \]
\[ a = 0.4 + (1 - 0.4) \cdot (0.94) = 0.4 + 0.6 \cdot 0.94 \]

\[ c = (0.3, 0.23, 0.23) \]
\[ a = 0.964 \]
Compositing Orders

\[ c = a_f \cdot c_f + (1 - a_f) \cdot a_b \cdot c_b \]
\[ a = a_f + (1 - a_f) \cdot a_b \]

Order Matters!

\[ c = (0.3, 0.23, 0.4) \quad a = 0.964 \]
\[ c = (0.3, 0.23, 0.23) \quad a = 0.964 \]
The Emission-Absorption Model

How realistic is $\alpha$-compositing?

The emission-absorption model (Sabella 1988)

Without absorption all the initial radiant energy would reach the point $s$. 

$\text{Initial intensity at } s_0$  

$I(s) = I(s_0)$ 

Without absorption all the initial radiant energy would reach the point $s$. 

$I(s_0)$ 

$\text{Viewing ray}$ 

$s_0$ 

$s$
The Emission-Absorption Model

How realistic is $\alpha$-compositing?

The emission-absorption model (Sabella 1988)

Initial intensity at $s_0$

$I(s) = I(s_0) e^{-\tau(s,s_0)}$

Absorption along the ray segment $s_0 - s$

Optical depth $\tau$

Absorption $\kappa$

$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s) ds$
The Emission-Absorption Model

How realistic is \( \alpha \)-compositing?

The emission-absorption model (Sabella 1988)

\[
I(s) = I(s_0) e^{-\tau(s,s_0)} + \int_{s_0}^{s} q(\tilde{s}) e^{-\tau(\tilde{s},s)} d\tilde{s}
\]
Numerical Solution

Optical depth: $\tau(0, t) = \int_0^t \kappa(\hat{t}) d\hat{t}$
Numerical Solution

Optical depth: $\tau(0, t) = \int_0^t \kappa(t) \, dt$

Approximate Integral by Riemann sum:

$\tau(0, t) \approx \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t) \Delta t$
Numerical Solution

\[
\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{[t/\Delta t]} \kappa(i \cdot \Delta t) \Delta t
\]

\[
e^{-\tilde{\tau}(0,t)} = e^{-\sum_{i=0}^{[t/\Delta t]} \kappa(i \cdot \Delta t) \Delta t}
\]

\[
e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{[t/\Delta t]} e^{-\kappa(i \cdot \Delta t) \Delta t}
\]
Numerical Solution

Now we introduce opacity

\[ 1 - A_i = e^{-\kappa(i \cdot \Delta t) \Delta t} \]
Numerical Solution

$q(t)$
Numerical Solution

\[ q(t) \approx C_i = c(i \cdot \Delta t) \Delta t \]

\[ e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{[t/\Delta t]} (1 - A_i) \]

\[ \tilde{C} = \sum_{i=0}^{[T/\Delta t]} C_i e^{-\tilde{\tau}(0,t)} \]
Numerical Solution

e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{[t/\Delta t]} (1 - A_i)

q(t) \approx C_i = c(i \cdot \Delta t) \Delta t

\tilde{C} = \sum_{i=0}^{[T/\Delta t]} C_i \prod_{j=0}^{i-1} (1 - A_j)

can be computed recursively/iteratively!
Numerical Solution

Note: we just changed the convention from $i=0$ is at the front of the volume (previous slides) to $i=0$ is at the back of the volume!

can be computed recursively/iteratively:

$$C_i' = C_i + (1 - A_i)C_{i-1}'$$

$$C_i = A_i C_{\text{pure\_color}}$$
Numerical Solution

\[ C_i' = C_i + (1 - A_i)C_{i-1}' \]

Radiant energy observed at position \( i \)
Radiant energy emitted at position \( i \)
Absorption at position \( i \)
Radiant energy observed at position \( i-1 \)
Numerical Solution

Back-to-front compositing

\[ C'_i = C_i + (1 - A_i)C'_{i-1} \]

Iterate from \( i=0 \) (back) to \( i=\text{max} \) (front): \( i \) increases

Front-to-back compositing

\[ C'_i = C'_{i+1} + (1 - A'_{i+1})C_i \]
\[ A'_i = A'_{i+1} + (1 - A'_{i+1})A_i \]

Iterate from \( i=\text{max} \) (front) to \( i=0 \) (back): \( i \) decreases
Other Compositing - Average

Intensity

Average

Depth

Synthetic Reprojection
Computational Strategies

• How can the basic ingredients be combined:
  • Image Order
    • Ray casting (many options)
  • Object Order
    • splatting, texture-mapping
  • Combination (neither)
    • Shear-warp, Fourier
Object Order

- Render image one voxel at a time

for each voxel ...
- transfer function
- determine image contribution
- composite
Splatting

- Lee Westover - Vis 1989; SIGGRAPH 1990
- Object order method
- Front-To-Back or Back-To-Front
- Main idea: Throw voxels to the image

- Original method - fast, poor quality
- Many many improvements since then!
  - Crawfis’93: textured splats
  - Swan’96, Mueller’97: anti-aliasing
  - Mueller’98: image-aligned sheet-based splatting
  - Mueller’99: post-classified splatting
  - Huang’00: new splat primitive: FastSplats
Splatting

Instead of asking which data samples contribute to a pixel value, ask, to which pixel values does a data sample contribute?

- **Ray casting**: pixel value computed from multiple data samples
- **Splatting**: multiple pixel values (partially) computed from a single data sample

**Overview:**
- high-quality
- relatively costly -> relatively slow

**Idea:** contribute every voxel to the image
- projection from voxel: splat
- composite in image space
Splatting - Footprint

- Process from closest voxel to furthest voxel
- The first step is splat. A biggest problem: determination of voxel’s projected area called its footprint
Splatting - Footprint

- Project each sample (voxel) from the volume into the image plane
Splatting - Footprint

Draw each voxel as a cloud of points (footprint) that spreads the voxel contribution across multiple pixels.

A natural way to compute the footprint is to add a filter kernel, which determines how much contribution this voxel makes to those pixels nearby the projected pixel corresponding to the center of the voxel.

Different pixels receive different amount of contribution computed as the multiplication of some weight with the original color or other value.
Splatting - Footprint

- Larger footprint increases blurring and used for high pixel-to-voxel ratio

- Footprint geometry
  - Orthographic projection: footprint is independent of the view point
  - Perspective projection: footprint is elliptical

- Pre-integration of footprint

- For perspective projection: additional computation of the orientation of the ellipse

http://cs.swan.ac.uk/~csbob/teaching/csM07-vis/
Splatting - Footprint

- **Volume** = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
  - Voxel contribution = footprint · (C, opacity)
- Weighted footprints accumulate into image

voxel kernels ⇒ screen footprints = splats

http://cs.swan.ac.uk/~csbob/teaching/csM07-vis/
Splatting - Footprint

- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
  - Voxel contribution = footprint \cdot (C, opacity)
- Weighted footprints accumulate into image

http://cs.swan.ac.uk/~csbob/teaching/csM07-vis/
Splatting - Footprint

• Volume = field of 3D interpolation kernels
  • One kernel at each grid voxel
• Each kernel leaves a 2D footprint on screen
  • Voxel contribution = footprint · (C, opacity)
• Weighted footprints accumulate into image
Splatting - Compositing

- Voxel kernels are added within sheets
- Sheets are composited front-to-back
- Sheets = volume slices most parallel to the image plane
Splatting - Implementation

• Volume
Splatting - Implementation

• Add voxel kernels within first sheet

volume slices

image plane

sheet buffer

compositing buffer
Splatting - Implementation

• Transfer to compositing buffer
Splatting - Implementation

- Add voxel kernels within second sheet
Splatting - Implementation

• Composite sheet with compositing buffer
Splatting - Implementation

- Add voxel kernels within third sheet
Splatting - Implementation

- Composite sheet with compositing buffer
What Doesn’t Work?

• Mathematically, the early splatting methods only work for X-ray type of rendering, where voxel ordering is not important
  – Bad approximation for other types of optical models
• Object ordering is important in volume rendering, front objects hide back objects
  – Need to composite splats in proper order, else we get bleeding of background objects into the image (color bleeding!)
• Axis-aligned approach add all splats that fall within a volume slice most parallel to the image plane, composite these sheets in front-to-back order
  – Incorrect accumulating on axis-aligned face cause popping
• A better approximation with Riemann sum is to use the image-aligned sheet-based approach
Problems Early Implementation – Axis Aligned Splatting

• In-accurate compositing, result in color bleeding and popping artifacts (Demo)!

Part of this voxel gets composited **before** part of this voxel.
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Image-Aligned Sheet-Buffer

- Slicing slab cuts kernels into sections
- Kernel sections are added into sheet-buffer
- Sheet-buffers are composited
Splatting

- Simple extension to volume data without grids
- Scattered data with kernels
- Example: SPH (smooth particle hydrodynamics)
- Needs sorting of sample points
Splatting – Images
Splatting – Conclusion

• Pros:
  • high-quality
  • easy to parallelize
  • works for anisotropic data (dz > dx = dy)
  • perspective projection possible
  • adaptive rendering possible

• Cons:
  • relatively slow
  • yields somewhat blurry images (in original)
Splatting vs Ray Casting

**Splatting:**
- **Object-order:** FOR each voxel \((x,y,z)\) DO
  - sample volume at \((x,y,z)\) using filter kernel
  - project reconstruction result to x-y image plane (leaving footprint)
- FOR each pixel \((x,y)\) DO:
  - composite (color, opacity) result of all footprints

**Ray Casting:**
- **Image-order:** FOR each pixel \((x,y)\) DO
  - cast ray into volume
  - FOR each sample point along ray \((x,y,z)\)
    - Sample volume at \((x,y,z)\) using filter kernel
    - composite (color, opacity) in image space at pixel \((x,y)\)
Computational Strategies

- How can the basic ingredients be combined:
  - Image Order
    - Ray casting (many options)
  - Object Order
    - splatting, texture-mapping
  - Combination (neither)
    - Shear-warp, Fourier
Shear-warp factorization

- General goal - make viewing rays parallel to each other and perpendicular to the image
- This is achieved by a simple shear, followed by a 2D image warp:
Shear-warp factorization

- Algorithm:
  - Shears along the volume slices
  - Projection and compositing
  - Transformation to get correct result
The Shear-Warp Factorization
Texture-based Volume Rendering

- Volume rendering by 2D texture mapping:
  - use planes parallel to base plane (front face of volume which is "most orthogonal" to view ray)
  - draw textured rectangles, using bilinear interpolation filter
  - render back-to-front, using α-blending for the α-compositing

Image credit: H.W. Shen, Ohio State U.
Texture-based Volume Rendering

• Volume rendering by 3D texture mapping:
  – use the voxel data as the 3D texture
  – render an arbitrary number of slices (eg. 100 or 1000) parallel to image plane (3- to 6-sided polygons)
  – back-to-front compositing as in 2D texture method

Limited by size of texture memory.

Image credit: H.W.Shen, Ohio State U.
Slicing

object \((\text{color, opacity})\)

Similar to ray-casting with simultaneous rays
Slicing Volume Data

Graphics Hardware
- Polygons
- Textures
- Blending operations

Volume Data
- Proxy geometry
- Data & interpolation
- Numerical integration
Effect of the Sample Rate

1 slice

5 slices

20 slices

45 slices

85 slices

170 slices
Slice Based Problems?

• Does not perform correct
  – Illumination
  – Accumulation - but can get close

• Can not easily add correct illumination and shadowing
  – See the Van Gelder paper for their addition for illumination
    • Stored in LUT quantized normal vector directions
Unstructured Volume Rendering

• Given an irregular data set that consists of volumetric cells (typically from FEM simulation)

• How can the volume be displayed accurately?

• Numerous approaches:
  – Ray casting
  – Ray tracing
  – Sweep plane algorithms (e.g. ZSWEEP)
  – PT algorithm of Shirley and Tuchman
PT algorithm of Shirley and Tuchman

- Decompose each cell into tetrahedra
- Sort the tetrahedra in a back to front fashion
- Project each tetrahedron and render its decomposition into 3 or 4 triangles

Two different non-degenerate classes of the projected tetrahedra
Composing of the Tetrahedra

- For each rendered pixel the ray integral of the corresponding ray segment has to be computed.

- **Observation**: The ray integral depends only on $S_f$, $S_b$, and $I$ for the *Volume Density Optical Model* of Williams et al.
3D Texturing Approach

- Compute the three-dimensional ray integral by numerical integration and store the integrated chromaticity and opacity in a 3D texture
- Assign appropriate texture coords \((S_f, S_b, l)\) to the projected vertices of each tetrahedron
Additional Reading

For Ray casting


• *Data Visualization, Principles and Practice, Chapter 10 Volume Visualization*, by A. Telea, AK Peters, 2008

For splatting, please see,

• *Data Visualization, Principles and Practice, Chapter 9, Image Visualization*, by A Telea, AK Peters 2008


For shear-warp factorization, please see,

Acknowledgment

Thanks for materials

• Prof. Charles D. Hansen, SCI, University of Utah
• Prof. Ronald Peikert (ETH)
• Prof. Robert Laramee, Swansea University
• Prof. Markus Hadwiger, KAUST
• Prof. Jian Huang, University of Tennessee