Direct Volume Rendering

- Sampled data of a continuous 3D field
- Semi-transparent cloud of points on a discrete grid (no explicit notion of surfaces)
- View all layers of surfaces at once
- A gel-type material that reflects and attenuates light, controlled by transfer functions (radiative transport theory)
The Volume Data Structure

3D Regular Rectilinear Grid

vtkStructuredPoints:
Dimensions = \((D_x, D_y, D_z)\)
Spacing = \((S_x, S_y, S_z)\)
How to visualize?

- **Slicing**: display the volume data, mapped to colors, along a slice plane
- **Iso-surfacing**: generate opaque and semi-opaque surfaces on the fly
- **Transparency effects**: volume material attenuates reflected or emitted light

![Semi-transparent material](image)
![Iso-surface](image)
Semi-Transparent - How?

- Rendering Integral
  \[ I(t) = \int_{t_0}^{t} c(s) e^{-\alpha(s)} ds \]
  \[ \alpha(s) = k \int_{t_0}^{s} \rho(u) du \]

- Discretize Integral

C(t): intensity
\(\alpha(t)\): opacity (accumulated)
\(\rho(t)\): "density"
Evaluation = Compositing

“over” operator - Porter & Duff 1984

\[
C_{out} = C_{in} \cdot (1 - \alpha) + C \cdot \alpha \quad \quad \quad C_{(i)}_{in} = C_{(i-1)}_{out}
\]
Direct Rendering Pipeline

- Transformation
- Interpolation
- Classification
- Composition
- Shading
- Validate
Back-To-Front

- A viewing algorithm that traverses and renders the scene objects in order of **decreasing** distance from the observer.
- Maybe derived from a standard - "Painters Algorithm"
2D
- Start traversal at point farthest from the observer,
- 2 orders
- Either x or y can be innermost loop
- If x is innermost, display order will be A, C, B, D
- If y is innermost, display order will be C, A, D, B
- Both result in the correct image!
- If voxel (x,y) is (partially) obscured by voxel (x',y'), then x ≤ x' and y ≤ y'. So project (x,y) before (x',y') and the image will be correct.
Back-To-Front

✓ 3D
- Axis traversal can still be done arbitrarily, 8 orders
- Data can be read and rendered as slices
- Note: voxel projection is NOT in order of strictly decreasing distance, so this is not the painter’s algorithm.
- Composition
  \[ I = (1 - \alpha) I + \alpha I_{\text{new}} \]
- Pros: easy composition
  cons: no early termination
### Projection order

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Display order</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
<td>&gt;=0</td>
<td>&gt;=0</td>
<td>7,6,5,4,3,2,1,0</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&gt;=0</td>
<td>&gt;=0</td>
<td>6,7,4,5,2,3,0,1</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>&lt;0</td>
<td>&gt;=0</td>
<td>5,4,7,6,1,0,3,2</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>&gt;=0</td>
<td>&lt;0</td>
<td>3,2,1,0,7,6,5,4</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&lt;0</td>
<td>&gt;=0</td>
<td>4,5,6,7,0,1,2,3</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&gt;=0</td>
<td>&lt;0</td>
<td>2,3,0,1,6,7,4,5</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>&lt;0</td>
<td>&lt;0</td>
<td>1,0,3,2,5,4,7,6</td>
</tr>
<tr>
<td>&lt;0</td>
<td>&lt;0</td>
<td>&lt;0</td>
<td>0,1,2,3,4,5,6,7</td>
</tr>
</tbody>
</table>
**Front-To-Back**

✓ A viewing algorithm that traverses and renders the scene objects in order of *increasing* distance from the observer.

\[
C_{out} = C_{in} + C \cdot \alpha \cdot (1 - \alpha_{in})
\]

\[
\alpha_{out} = \alpha_{in} + \alpha \cdot (1 - \alpha_{in})
\]
Volume Rendering Strategies

Object-Order Approach: Traverse the volume, and project to the image plane.

Splatting cell-by-cell

Texture Mapping plane-by-plane
Volume Rendering Strategies

Image-Order Approach: Traverse the image pixel-by-pixel and sample the volume.

Ray Casting
Ray Tracing

✓ “another” typical method from traditional graphics
✓ Typically we only deal with primary rays - hence: ray-casting
✓ a natural image-order technique
✓ as opposed to surface graphics - how do we calculate the ray/surface intersection???
✓ Since we have no surfaces - we need to carefully step through the volume
Ray Casting

✓ A ray is cast into the volume, sampling the volume at certain intervals
✓ The sampling intervals are usually equi-distant
✓ At each sampling location, a sample is interpolated / reconstructed from the grid voxels
✓ popular filters are: nearest neighbor (box), trilinear (tent), Gaussian, cubic spline
✓ Along the ray - what are we looking for?
Ray Cast Functions

A Ray Function examines the scalar values encountered along a ray, and produces a final pixel value according to the volume properties, and the specific function.
Ray Traversal Schemes

Intensity
- Max
- Average
- Accumulate
- First

Depth
Ray Traversal - First

Intensity

First

Depth
Ray Traversal - Average

Intensity

Average

Depth

✓ **Average**: produces basically an X-ray picture
Ray Traversal - MIP

- Max: Maximum Intensity Projection used for Magnetic Resonance Angiogram
- Faster to render
- Show the most important features
Maximum Intensity Function

Scalar Value

Ray Distance

Opacity

Scalar Value

Opacity

Gradient Magnitude

Maximum Value

Maximize Scalar Value
Ray Traversal - Accumulate

Accumulate: make transparent layers visible!
Levoy '88
Composite Function

Use $\alpha$-blending along the ray to produce final RGBA value for each pixel.
Stop ray traversal at isosurface value. Use cubic equation solver if interpolation is trilinear.
Raycasting

Interpolation kernel

volumetric compositing

object (color, opacity)

1.0
Raycasting

Interpolation kernel

volumetric compositing

color \( c = c_s \alpha_s (1 - \alpha) + c \)

opacity \( \alpha = \alpha_s (1 - \alpha) + \alpha \)

object (color, opacity)
Raycasting

volumetric compositing

object (color, opacity)
Raycasting

volumetric compositing

object (color, opacity)
Interpolation

- Eye
- Image pixel
- Viewing ray
- Voxel
- Trilinear interpolation
- Sample point
Interpolation (2)

- **Binary**
  - Closest value

- **Smooth**
  - Weighted average
Scalar Value Interpolation

\[ v = (1-x)(1-y)(1-z)S(0,0,0) + \]
\[ (x)(1-y)(1-z)S(1,0,0) + \]
\[ (1-x)(y)(1-z)S(0,1,0) + \]
\[ (x)(y)(1-z)S(1,1,0) + \]
\[ (1-x)(1-y)(z)S(0,0,1) + \]
\[ (x)(1-y)(z)S(1,0,1) + \]
\[ (1-x)(y)(z)S(0,1,1) + \]
\[ (x)(y)(z)S(1,1,1) \]

\[ v = S(\text{rnd}(x), \text{rnd}(y), \text{rnd}(z)) \]

Nearest Neighbor \hspace{1cm} Trilinear
Scalar Value Interpolation

Nearest Neighbor Interpolation

Trilinear Interpolation
Sampling Distance

0.1 Unit Step Size

1.0 Unit Step Size

2.0 Unit Step Size
Derivatives

✓ Central difference

\[
G_x = \frac{v_{i+1,j,k} - v_{i-1,j,k}}{2}
\]

\[
G_y = \frac{v_{i,j+1,k} - v_{i,j-1,k}}{2}
\]

\[
G_z = \frac{v_{i,j,k+1} - v_{i,j,k-1}}{2}
\]

Or

\[
G_x = \frac{v(p + \tilde{n}_x) - v(p - \tilde{n}_x)}{2}
\]

\[
G_y = \frac{v(p + \tilde{n}_y) - v(p - \tilde{n}_y)}{2}
\]

\[
G_z = \frac{v(p + \tilde{n}_z) - v(p - \tilde{n}_z)}{2}
\]
Shading

✓ Phong Shading + Depth Cueing

\[
C(x) = C_p k_a + \frac{C_p}{k_1 + k_2d(x)} (k_d (N(x) \cdot L) + k_s (N(x) \cdot H)^n)
\]

✓ \(C_p\) = color of light source
✓ \(k_a / k_d / k_s\) = ambient / diffuse / specular light coefficient
✓ \(k_1, k_2\) = fall-off constants
✓ \(d(x)\) = distance to picture plane
✓ \(L\) = normalized vector to light
✓ \(H\) = normalized vector for maximum highlight (half vector)
✓ \(n\) = shininess coefficient
✓ \(N(x_i)\) = surface normal at voxel \(x_i\)
Classification of Scalar Data

✓ Maps raw voxel value into presentable entities: color, intensity, opacity, etc.
1. Raw-data $\rightarrow$ material ($R, G, B, \alpha, K_a, K_d, K_s, ...$).
2. Material $\rightarrow$ shaded material.

✓ Step 1: may require probabilistic methods. Derive material volume from input. Estimate % of each material (e.g. fat, tissue, bone, air) in all voxels. Pre-computed.

✓ Step 2: High gradient in the data values detects surface and is used as a measure of its orientation.
Material Segmentation

Scalar value can be classified into color and opacity (RGBA)

Gradient magnitude can be classified into opacity

Final opacity is obtained by multiplying scalar value opacity by gradient magnitude opacity
Material Segmentation
Classification example (Levoy)

\[ \alpha(x_i) = \alpha_v \left( 1 - \frac{1}{r} \left| \frac{f_v - f(x_i)}{|f'(x_i)|} \right| \right) \]

\[ r = \frac{\max(f_v - f(x_i))}{\max(|f'(x_i)|)} \]

Opacity \( \alpha(x_i) \)

Gradient magnitude \( |\nabla f(x_i)| \)

Acquired value \( f(x_i) \)
Classification example (Levoy)

\[ \alpha(x_i) = f'(x_i) \left( \alpha_{v_{n+1}} \left( \frac{f(x_i) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}} \right) + \alpha_{v_n} \left( \frac{f_{v_{n+1}} - f(x_i)}{f_{v_{n+1}} - f_{v_n}} \right) \right) \]

Opacity \( \alpha(x_i) \) vs. Gradient magnitude \( |\nabla f(x_i)| \)

Acquired value \( f(x_i) \)
Volume Properties

- Color
- ScalarOpacity
- GradientOpacity
- GradientOpacityScale
- GradientOpacityBias
- InterpolationType
- Shading
- Ambient
- Diffuse
- Specular
- SpecularPower
- vtkColorTransferFunction
  or
  vtkPiecewiseFunction
- vtkPiecewiseFunction
- vtkPiecewiseFunction
- float
- float
- Nearest or Linear
- On or Off
- float
- float
- float
- float
Material Percentage Volume --
Drebin, Carpenter, Hanrahan (DCH 1988)

✓ Probabilistic classifier
✓ probability that a voxel has intensity I:

\[ P(I) = \sum_{i=1}^{n} p_i P_i(I) \]

✓ \( p_i \) - percentage of material
✓ \( P_i(I) \) - prob. that material i has value I, \( P_i(I) \) given through statistics/physics

✓ \( p_i \) (percentage of material i in a voxel with intensity I) is then given by:

\[ p_i(I) = \frac{P_i(I)}{\sum_{j=1}^{n} P_j(I)} \]

✓ A lookup table is used
Classification (DCH)

✓ Assumes only two materials per voxel that will lead to material percentage volumes from them we conclude color/opacity:

\[ C = \sum_{i=1}^{n} p_i C_i \]

where \( C_i = (\alpha_i R_i, \alpha_i G_i, \alpha_i B_i, \alpha_i) \)

✓ Matting operations: adding or reducing certain materials.
Classification - Probabilistic Methods

- **Material Assignment**
  - Air
  - Fat
  - Tissue
  - Bone

- **Constituent’s Distributions**
  - CT Number
  - Air
  - Fat
  - Tissue
  - Bone

- **Histogram**
Shading (DCH)

- Using the material percentage volumes, a density volume is extracted:

\[ D = \sum_{i=1}^{n} p_i \rho_i \]

- A surface normal is computed through forward differences.
Three layers need to be combined

\[ I' = C_F \text{ over } C_S \text{ over } C_B \text{ over } I \]

\[ A \text{ over } B = A \alpha_A + (1 - \alpha_A) B \]

\[
C_s = k_d (\vec{N} \cdot \vec{L}) C_{diff} + k_s (\vec{N} \cdot \vec{H}) C_L
\]

\[ C_{diff} = C_F + C_B \]

or

\[ C_{diff} = C_B \]

(need to examine sign of gradient)
Shear Warp Algorithm

- General goal - make viewing rays parallel to each other and perpendicular to the image
- This is achieved by a simple shear:

![Shear Warp Algorithm Diagram]
Shear Warp: transformation matrix decomposition

✓ General algorithm:

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

Where

- $M_{\text{view}} = \text{general viewing matrix}$
- $P = \text{permutation matrix, which transposes the coordinate system in order to make the z-axis the principal viewing axis}$
- $S = \text{transforms volume into a sheared object with a slices parallel to the viewing plan.}$
- $M_{\text{warp}} = \text{sheared object coordinates into image coordinates} \quad M_{\text{warp}} = S^{-1} \cdot P^{-1} \cdot M_{\text{view}}$
Shear Warp: Shearing matrix

✓ Parallel vs. Perspective

\[
S_{\text{par}} = \begin{pmatrix}
1 & 0 & s_x & 0 \\
0 & 1 & s_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

\[
S_{\text{persp}} = \begin{pmatrix}
1 & 0 & s'_x & 0 \\
0 & 1 & s'_y & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & s'_w & 1 \\
\end{pmatrix}
\]
Shear Warp properties

1. scanlines of pixels in the intermediate image are parallel to scanlines of voxels in the volume data: *image copy, 2D resampling, scanline-by-scanline data access*

2. all voxels in a given voxel slice are scaled by the same factor: *pre-computed resampling coefficients*
Shear Warp: Run-Length Encoding (RLE) data structure

- Need to scan two or more scanlines together for resampling.
- Pre-classified data: no change of transfer function.
Shear Warp : implementations

- Parallel projection:
  - efficient reconstruction: *front-to-back composition, early termination by skipping opaque pixel spans, 2D resampling, pre-computed interpolation coefficients.*
  - lookup table for shading: *using normals on a sphere*
  - three RLE of the actual volume (in x, y, z): *memory cost is high.*

- Perspective Projection:
  - similar to parallel projection, except - voxels also need to be scaled.
  - more then two voxel scan-lines may be needed for each image scan-line
Shear Warp: dynamic classification

- Opacity based density and gradient: \( \alpha = f(p, q) \)
- A Min-max octree is constructed for each volume
- A Summed Area Table (SAT) is build for each transfer function
- Transparent blocks - lookup through summed area table
(ideally) we would reconstruct the continuous volume (cloud) using the interpolation kernel $w$:

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

then compute the analytic integral along a ray $r$:

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

This can only be approximated by discretization.

Integration normally also involve alpha channel composition.
Fig. 14.27 Sampling and reconstruction: Adequate sampling rate. (a) Original signal.
(b) Sampled signal. (c) Sampled signal ready to be reconstructed with sinc. (d) Signal reconstructed with sinc. (e) Sampled signal ready to be reconstructed with triangle.
(f) Signal reconstructed with triangle. (Courtesy of George Wolberg, Columbia University.)
Splatting - principal idea

This last equation

\[ I(p) = \int \sum_k w(p + r - v_k)f(v_k)dr \]

can be rewritten in the following way:

\[ I(p) = \sum_k f(v_k)\int w(p + r - v_k)dr \]

Splatting Kernel or “Splat”

Which can be computed \textit{analytically}:

\[ \text{Splat}(x, y) = \int w(x, y, z)dz \]
Splatting - principal idea

- Draw each voxel as a cloud of points that spreads the voxel.
- Its projection provides contributions across multiple pixels (footprint):
  - Integration of interpolation kernel
  - Computed in pixel resolution
- Larger footprint increases blurring and used for high pixel-to-voxel ratio.
Splatting - efficiency

✓ "footprint" - splatted (integrated) kernel
  - Computed as a 2D field in X-Y (pixel) coordinates
  - Sample over (x,y), and integral over z
    weight (x,y) = footprint (x-dx, y-dy);
  - Pre-compute footprint in a lookup table
  - Pixel displacement: interpolate within footprint

✓ Parallel projection: kernel is spherical, and its footprint is independent of the view point

✓ Perspective projection: footprint is an ellipse, needs to compute the orientation of the ellipse for each view point.
Splatting

- Volume = field of 3D interpolation kernel
  - 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

![Diagram of voxel kernels and screen footprints](image)
Splatting - Highlights

✔ Footprints can be pre-integrated
  - fast voxel projection

✔ Advantages over raycasting:
  - Fast: voxel interpolation is in 2D on screen
  - More accurate integration (analytic for X-ray)
  - More accurate reconstruction (afford better kernels)
  - Only relevant voxels must be projected (read one voxel at a time)
Splatting - Implementation

- Voxel kernels are added within sheets (still a voxel-by-voxel operation)
- Sheets are composited front-to-back
- Sheets = volume slices most perpendicular to the viewing direction.

volume slices

image plane at 30°

volume slices

image plane at 70°
Splatting - Implementation
Splatting - Drawbacks

✓ In-accurate compositing

Part of this pixel gets composited **before** part of this pixel

Reason: Integral and summation cannot switch order when alpha blending is used.