

Direct Volume Rendering

R. Daniel Bergeron
Department of Computer Science
University of New Hampshire
Durham, NH 03824

Based on:

Brodie and Wood, *Recent Advances in Visualization of Volumetric Data*, Eurographics 2000
State of the Art Report.

Drebin et al., *Volume Rendering*, **Siggraph 88.**

Sabella, *A Rendering Algorithm for Visualizing 3D Scalar Fields*, **Siggraph 88.**

Levoy, Westover, et al., *Introduction to Volume Rendering*, **Siggraph 90 Course Notes**

05/03/17

1

R. Daniel Bergeron

Overview

- ♦ Model data as a translucent gas or gel
 - need to assign material properties to data values
- ♦ **Classification** – assign *color* / *opacity* to data values
 - *Opacity transfer function* – maps data value and other parameters (such as gradient) to opacity value
 - *Color transfer function* – same for color
- ♦ **Segmentation** – applic-dependent “labeling” of data values, typically *a priori*.
 - gradient often used as *ad hoc* effort to segment

2

R. Daniel Bergeron

Volume Rendering Integral

- ♦ Treat volume as particles with density μ .
- ♦ Send ray through each pixel in image plane; for each wavelength λ , the light reaching pixel is

$$I_{\lambda} = \int_0^L C_{\lambda}(s) \mu(s) e^{-\int_0^s \mu(t) dt} ds$$

- ♦ where L is ray length, $C_{\lambda}(s)$ is light reflected at s in ray direction.
 $\mu(s)$ is a weight based on density – larger density means more reflected light. Integral accumulates intensity, but attenuates it (the exponential) as it passes through material.
 μ defines rate at which light is occluded per unit length due to scattering or extinction

3

R. Daniel Bergeron

Volume Rendering Integral Approximation

- ♦ Using Riemann sum approximation and using n as # samples
$$I_{\lambda} = \sum_{i=0..n} C_{\lambda}(i\Delta s) \mu(i\Delta s) \Delta s \prod_{j=0..i-1} \exp(-\mu(j\Delta s) \Delta s)$$
- ♦ Now replace exponential term with 2 terms of Taylor expans.
$$\exp(-\mu(j\Delta s) \Delta s) = 1 - \mu(j\Delta s) \Delta s$$

and define *transparency* $t(j\Delta s)$ as
$$t(j\Delta s) = \exp(-\mu(j\Delta s) \Delta s)$$

and *opacity*, $\alpha(j\Delta s) = 1 - t(j\Delta s) = \mu(j\Delta s) \Delta s$
and:
$$I_{\lambda} = \sum_{i=0..n} C_{\lambda}(i\Delta s) \alpha(i\Delta s) \prod_{j=0..i-1} (1 - \alpha(j\Delta s))$$

for $\Delta s=1$, we get:
$$I_{\lambda} = \sum_{i=0..n} C_{\lambda}(i) \alpha(i) \prod_{j=0..i-1} (1 - \alpha(j))$$
- ♦ Do this for R,G,B: summing intensities of individual samples, each of which is attenuated by the product of transparencies accumulated as light passes from sample to pixel.

4

R. Daniel Bergeron

Recursive Approximation

- ♦ Dropping λ , and expanding we get

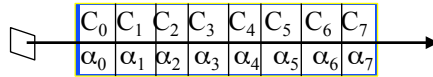
$$C = C_0\alpha_0 + C_1\alpha_1(1-\alpha_0) + C_2\alpha_2(1-\alpha_1)(1-\alpha_0) + \dots$$

- ♦ Can compute recursively using

$$C_{out} = C_{in} + (1-\alpha_{in}) \alpha_i C_i$$

$$\alpha_{out} = \alpha_{in} + (1-\alpha_{in}) \alpha_i$$

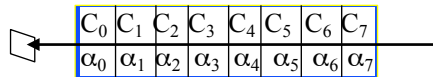
This is *front-to-back* image composition (Duff's *over* operator).



back-to-front ordering only needs to recursively compute color component

$$C_{out} = \alpha_i C_i + C_{in} (1-\alpha_i)$$

Note: compositing is associative, but
not commutative: order matters



5

R. Daniel Bergeron

DVR Approaches

- ♦ Image order approach: process from the image plane to the object
 - also called *backward rendering*
 - *ray casting* is classic image order algorithm
- ♦ Object order approach: process from the object to the image plane
 - also called *forward rendering*
 - *splatting* is the classic object order algorithm

6

R. Daniel Bergeron

Image Order Issues

- ♦ Volume rendering equation approximation
 - improve accuracy and/or speed
- ♦ Interpolation
 - calculating data values between grid points is vital
- ♦ Curvilinear and unstructured grids
 - basic approaches map nicely to rectilinear grids, others are more difficult to handle
- ♦ Faster ray traversal
- ♦ Hardware designed for volume rendering

7

R. Daniel Bergeron

Volume Rendering Eqn. 2

- ♦ Has been much work to make integration faster and more accurate
- ♦ Alternative is to dramatically simplify the approximation at the cost of accuracy:
 - *Maximum Intensity Projection* (MIP): simply find the maximum data value along the ray and project its “color”.
 - works well for angiography (highlight blood vessels)

8

R. Daniel Bergeron

Drebin et al., Siggraph '88

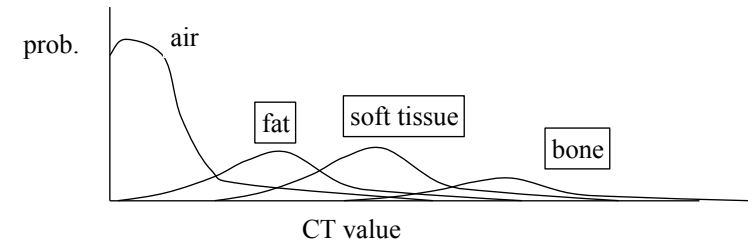
- ♦ CT data
- ♦ Basic *segmentation* based on probabilities
 - from segmentation, produced *density*, *color* and *opacity*
- ♦ Estimated gradient by simple forward differencing
 - Used gradient to infer surfaces for reflections

9

R. Daniel Bergeron

Segmentation

- ♦ Segmentation is often *ad hoc*, but *shouldn't* make binary decisions
 - for CT, X-ray absorption of materials is known *a priori* as a probability distribution function (pdf)

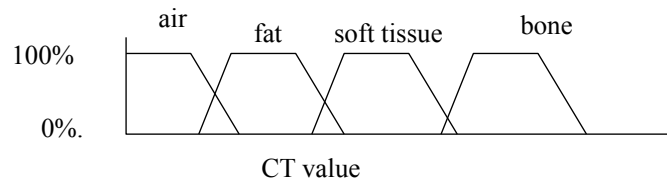


10

R. Daniel Bergeron

Segmentation 2

- ♦ Given a voxel has the value I ,
 probability of getting I , $P(I) = \sum_i p_i P_i(I)$
 where p_i is the probability of getting material i and
 $P_i(I)$ is probability that material i has value I
 Using Bayesian estimation,
 $p_i(I) = P_i(I) / (\sum_j P_j(I))$ which can be implemented as lookup
- ♦ Only 2 materials overlap: get simple relationship:



11

R. Daniel Bergeron

Density, color, opacity

- ♦ “density”, D , computed as
 $D(I) = \sum_i \rho_i p_i(I)$ where ρ_i is density of material i
- ♦ color and opacity ($\text{rgb}\alpha$)
 - $C(I) = \sum_i p_i(I) \alpha_i (R_i, G_i, B_i)$
- ♦ For each x, y, z , estimate by forward differences
 - gradient: $N(x, y, z) = (D_{x+1} - D_x, D_{y+1} - D_y, D_{z+1} - D_z)$
 - normalized gradient: $n(x, y, z) = N(x, y, z) / \|N(x, y, z)\|$
 - strength: $\|N(x, y, z)\|$
- ♦ $n(x, y, z)$ is used in lighting model for reflected light from a light source.

12

R. Daniel Bergeron

Ray Tracing Volume Data

(Notes from Levoy in *Introduction to Volume Rendering*, Siggraph 91 tutorial.)

- ◆ Data assumed to be samples of a continuous scalar function (voxel as point not volume)
- ◆ Sampling lattice is rectilinear and uniformly spaced
- ◆ Pixel spacing < voxel spacing
- ◆ Other typical simplifications
 - one ray per pixel (no supersampling)
 - parallel projection

13

R. Daniel Bergeron

View Specification

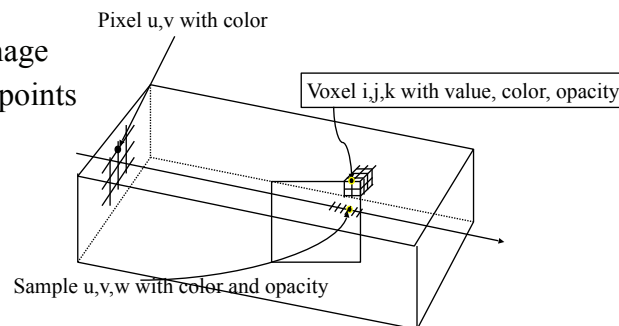
- ◆ Need view specification, image plane, volume location
 - Parallel projection along major axis
 - » integral mapping of voxel address space to pixel address space: 1-1 is easiest; usually have projection of a voxel map to $k \times k$ pixels
 - » arbitrary mapping requires interpolation
 - Arbitrary parallel projection
 - » need view direction and size of image space
 - » usually voxel address space as “world coordinates”

14

R. Daniel Bergeron

Coordinate Systems

- ◆ Object space
 - coordinate axes correspond to volume array indices
 - typically $N \times N \times N$
- ◆ Image space
 - $P \times P$ pixels in image
 - $P \times P \times W$ sample points

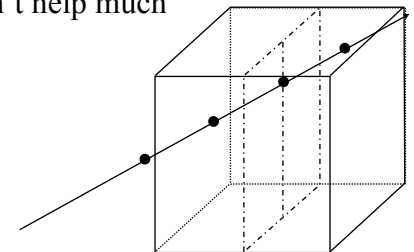


15

R. Daniel Bergeron

Resampling

- ◆ Calculating color/opacity inside a voxel is resampling the functions
- ◆ Sample at even spacing along ray
- ◆ Sampling rate (for typical CT and MR data)
 - less than voxel spacing introduces artifacts
 - more than twice per voxel doesn't help much
- ◆ Use trilinear interpolation



16

R. Daniel Bergeron

Trilinear Interpolation

From *Graphics Gems V*, p. 521

- ◆ Linear interpolation between 2 sample values:

$$v_x = (1-f_x)v_0 + f_x v_1 \text{ where } 0 \leq f_x \leq 1, \text{ also written as}$$

$$v_x = v_0 + f_x (v_1 - v_0)$$
- ◆ In 2-dimensions, interpolate from 4 points

$$v_{xy} = (1-f_x)(1-f_y)v_{00} + (1-f_x)f_y v_{01} + f_x(1-f_y)v_{10} + f_x f_y v_{11}$$
- ◆ But, more efficient (3 mults) to do 2 linear steps:

$$v_{x0} = v_{00} + f_x (v_{10} - v_{00})$$

$$v_{x1} = v_{01} + f_x (v_{11} - v_{01})$$

$$v_{xy} = v_{x0} + f_y (v_{x1} - v_{x0})$$

17

R. Daniel Bergeron

Trilinear Interpolation – 2

- ◆ And in 3D, interpolate from 8 points
 Use 3 linear steps (7 mults)

$$v_{x00} = v_{000} + f_x (v_{100} - v_{000})$$

$$v_{x01} = v_{001} + f_x (v_{101} - v_{001})$$

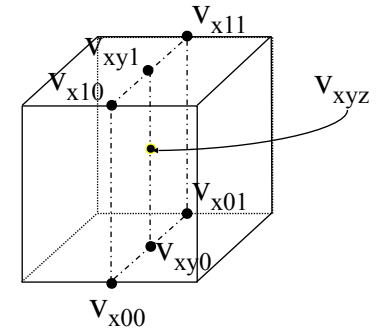
$$v_{x10} = v_{010} + f_x (v_{110} - v_{010})$$

$$v_{x11} = v_{011} + f_x (v_{111} - v_{011})$$

$$v_{xy0} = v_{x00} + f_y (v_{x10} - v_{x00})$$

$$v_{xy1} = v_{x01} + f_y (v_{x11} - v_{x01})$$

$$v_{xyz} = v_{xy0} + f_z (v_{xy1} - v_{xy0})$$



18

R. Daniel Bergeron

Splatting

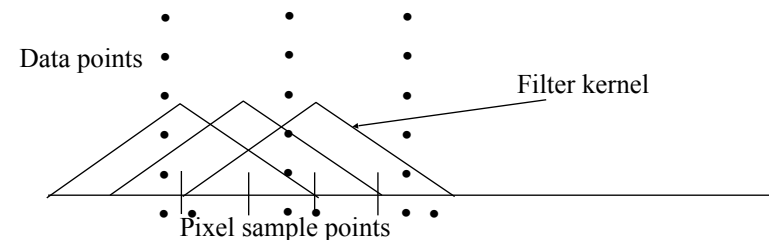
- ◆ Westover, *VolVis Symposium 89* and *Siggraph 90*
- ◆ Each voxel drawn on image plane as a cloud of points (footprint), covering many pixels
- ◆ Voxel treated as a single value “thrown at the screen”
- ◆ Example of *feed forward convolution* as opposed to a *feed backward convolution*

19

R. Daniel Bergeron

Feed Backward Convolution

- ◆ Output (pixel value) is weighted average of input data
- ◆ Center a convolution kernel at the output (pixel) location and gather data points that project onto kernel
- ◆ Touch each output sample once
- ◆ Touch each input data point many times

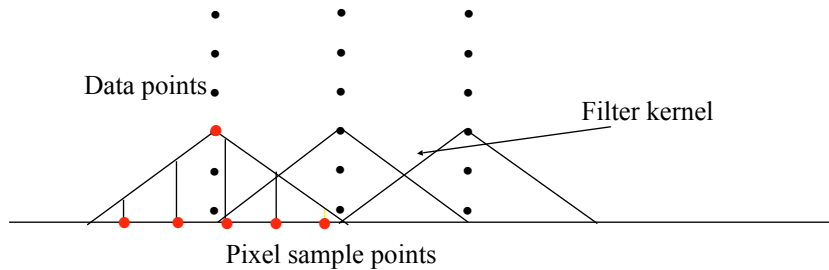


20

R. Daniel Bergeron

Feed Forward Convolution

- ◆ Input energy spread to many outputs (pixels)
- ◆ Center kernel at data point and distribute to output pixels (really a 3D convolution)
- ◆ Touch each input data point once
- ◆ Touch each output often



21

R. Daniel Bergeron

Splatting: Ideal

- ◆ Feed forward and incremental reconstruction
- ◆ Ideal splatting
 - center kernel at D
 - evaluate kernel
 - multiply by input value at D

$$\text{contribution}_D(x,y,z) = h(x-x_D, y-y_D, z-z_D)\rho(D)$$
 where h evaluates the convolution function
 - of course, this is terribly expensive

22

R. Daniel Bergeron

Splatting: Dimension Reduction

- ◆ Want 2D image from 3D data
 - given pixel at (x,y) , want the contribution for each point, D
 - center kernel at D
 - project weighted kernel onto (x,y) plane (assumes parallel projection along the z-axis)

$$\text{contribution}(x,y) = \rho(D) \int h(x-x_D, y-y_D, w)dw$$
 - Note integral is independent of the density (ρ); it depends only on (x,y) projected location; leads to *footprint function*:

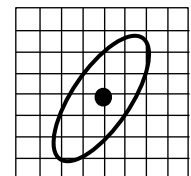
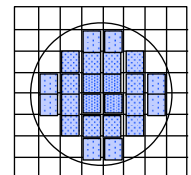
$$\text{footprint}(x,y) = \int h(x, y, w)dw$$
 where (x,y) is the displacement from projected sample point

23

R. Daniel Bergeron

Footprint Function Tables

- ◆ Can integrate the kernel function into a generic footprint table
- ◆ for each voxel
 - transform to screen space
 - for each pixel in the extent of the footprint
 - map back to precomputed table
 - composite the weighted contribution



24

R. Daniel Bergeron