What is direct volume rendering?
What is direct volume rendering?

Any rendering process which maps from volume data to an image \textbf{without} introducing \textit{binary distinctions} / \textit{intermediate} geometry, i.e., using color and opacity.

What is the difference between iso-surfacing and volume rendering?
What important concepts/techniques are needed for volume rendering?
What important concepts/techniques are needed for volume rendering?

- Interpolation
- Color/opacity transfer functions
- Color/opacity composition
- Gradient (optional for transfer function design and enhancing rendering quality)
What is the process of Raycasting?
What is the process of Raycasting?

For each pixel ...  
- cast ray  
- sampling along ray  
- interpolate  
- get colors/opacity  
- composite
What color and opacity compositions strategies are there?
What color and opacity compositions strategies are there?

- Maximum intensity projection (MIP)
- Local maximum intensity projection (LMIP)
- Average
- $\alpha$-composition
How does $\alpha$-composition work?
How does $\alpha$-composition work?

Recursively compose/blend colors and opacities in order (either back-to-front or front-to-back) in a linear fashion.

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

What physical model is $\alpha$-composition built on?
How does \( \alpha\)-composition work?

Recursively compose/blend colors and opacities in order (either back-to-front or front-to-back) in a linear fashion.

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

What physical model is \( \alpha\)-composition built on?

Emission-absorption
Direct Volume Rendering: Splatting & Texture-based
Computational Strategies

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

• Render image **one voxel at a time**

  - get color/opacity
  - determine image contribution
  - composite
Object Order

- Render image **one voxel at a time**

  - get color/opacity
  - determine image contribution
  - composite
Splatting-literature

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

• 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
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

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

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

**Props**
- high-quality, why?

**Cons**
- relatively costly -> relatively slow, why?
Splatting - Footprint

• Typically, process from closest voxel to furthest voxel (front-to-back)

• **The important step is splat.** A biggest problem: determination of voxel’s projected area called its **footprint**

```plaintext
for each voxel ...
- get color/opacity
- determine image contribution
- composite
```
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.
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

• Footprint geometry
  • **Orthographic projection**: footprint is independent of the viewpoint
  • Pre integration of footprint (like a template)
Splatting - Footprint

• **Footprint geometry**
  • *Orthographic projection*: footprint is independent of the viewpoint
  • Pre integration of footprint (like a template)

• **Perspective projection**: footprint is elliptical
  • additional computation of the orientation of the ellipse
Splatting - Footprint

• Footprint geometry
  • **Orthographic projection**: footprint is independent of the viewpoint
  • Pre integration of footprint (like a template)

• **Perspective projection**: footprint is elliptical
  • additional computation of the orientation of the ellipse

• **Importance of choosing footprint size!**
  • Larger footprint increases blurring and used for high pixel-to-voxel ratio
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
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

voxel kernels

screen footprints = splats

screen
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 \( \cdot (C, \text{opacity}) \)
- Weighted footprints accumulate into image

voxel kernels

screen footprints = splats
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

voxel kernels

screen footprints = splats
Splatting - Compositing

- Voxel kernels are added within sheets
- Sheets are composited front-to-back
- Sheets = volume slices most parallel to the image plane (i.e., base plane!)

volume slices

image plane at 30°

volume slices

image plane at 70°
Splatting - Implementation

• Volume

![Diagram of volume slices, sheet buffer, and compositing buffer]
Splatting - Implementation

• Add voxel kernels within first sheet
Splatting - Implementation

• Transfer to compositing buffer

![Diagram showing volume slices, sheet buffer, and compositing buffer]
Splatting - Implementation

• Add voxel kernels within second sheet

Image plane

Sheet buffer

Compositing buffer

Volume slices
Splatting - Implementation

- Composite sheet with compositing buffer

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

• Add voxel kernels within third sheet
Splatting - Implementation

• Composite sheet with compositing buffer

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

\[ a = a_f + (1 - a_f) \cdot a_b \]
Problems Early Implementation – Axis Aligned Splatting

• Inaccurate compositing, result in color bleeding and popping artifacts

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 plane ➔ sheet buffer

compositing buffer
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 (e.g., front to back)
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)

What parameters control the DVR quality for each method?
Direct Volume Rendering: Texture-based

Polygon Slices → 2D Textures → Final Image

Image credit: H.W. Shen, Ohio State U.
Texture in Graphics

Texture mapping can largely enhance the reality of the 3D objects.

How does it work?

For each fragment: interpolate the texture coordinates (barycentric)

Texture-Lookup: interpolate the texture color (bilinear)

Image source: Google image

[EuroGraphics 2006 Tutorial]
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). This is an axis-aligned approach!

Image credit: H.W. Shen Ohio State Univ.
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). **This is an axis-aligned approach!**
  - draw textured rectangles, using **bilinear** interpolation filter

**Image credit: H.W. Shen Ohio State Univ.**
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). **This is an axis-aligned approach!**
  - 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 Univ.
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). This is an axis-aligned approach!
  - draw textured rectangles, using bilinear interpolation filter
  - render back-to-front, using $\alpha$-blending for the $\alpha$-compositing

Is texture-based volume rendering an object-order or image-order approach? Why?
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). This is an axis-aligned approach!
  - draw textured rectangles, using bilinear interpolation filter
  - render back-to-front, using α-blending for the α-compositing

Essentially, a simplified version of splatting without splatting!
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
Effect of the Sample Rate

View direction

Slices

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
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 from

• 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
• Prof. Mike Bailey, Oregon State University