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
Splatting

• 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

**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**

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.

[Image: Diagram of 2D filter kernel and voxel contributions]

http://cs.swan.ac.uk/~csbob/teaching/csM07-vis/
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 \cdot (C, 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
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

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

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

volume slices

image plane at 30°

volume slices

image plane at 70°
Splatting - Implementation

• Volume

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

• Add voxel kernels within first sheet
Splatting - Implementation

• Transfer to compositing buffer

(volume slices)

(Color*opacity)

image plane

sheet buffer

compositing buffer
Splatting - Implementation

• Add voxel kernels within second sheet
Splatting - Implementation

• Composite sheet with compositing buffer

volume slices

(Color*opacity)

image plane

sheet buffer

compositing buffer
Splatting - Implementation

• Add voxel kernels within third sheet
Splatting - Implementation

• Composite sheet with compositing buffer

volume slices

(Color*opacity)

image plane

sheet buffer

compositing buffer
Problems Early Implementation – Axis Aligned Splatting

• Inaccurate compositing, result in color bleeding and popping artifacts (Demo)!

Part of this voxel gets composited before part of this voxel.

Problem:
“popping” of brightness when the image plane becomes more parallel to a different volume face.
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 \((x,y)\)