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

for each voxel ...
- get color/opacity
- determine image contribution
- composite
Object Order

• Render image one voxel at a time

for each voxel ...
- 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
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
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 & Cons**:
- high-quality
- relatively costly -> relatively slow
Splatting - Footprint

• Process from closest voxel to furthest voxel (front-to-back)

• 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.
Splatting - Footprint

• 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
Splatting - Footprint

- 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

- 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, \text{opacity})
• Weighted footprints accumulate into image

voxel kernels

screen footprints = \textit{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 \times (C, opacity)
- Weighted footprints accumulate into image

![Diagram showing voxel kernels and splats on screen](image-url)
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

![Diagram showing voxel kernels and screen footprints](image)

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 at 30°](image)

![Volume slices at 70°](image)
Splatting - Implementation

- Volume
Splatting - Implementation

• Add voxel kernels within first sheet
Splatting - Implementation

- Transfer to compositing buffer

(image plane) → sheet buffer → compositing buffer

(volume slices) → (Color*opacity)
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

(volume slices)

(sheet buffer)

(image plane)

(Colour*opacity)
Problems Early Implementation – Axis Aligned Splatting

- Inaccurate compositing, result in color bleeding and popping artifacts

Part of this voxel gets composited \textit{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

(image and diagram)
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 \((x,y)\)
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

![Diagram](image.png)

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

Texture mapping can large 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]
In OpenGL

Setting texture environment in the initialization before drawing

```c
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );

glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter );

glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );
```

Wrapping

Filter

Environment

Put this setting before the code you do the texture mapping!
OpenGL Texturing

Define the texture wrapping parameters. This will control what happens when a texture coordinate greater than 1.0 or less than 0.0 is encountered:

```c
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap );
gTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap );
```

where `wrap` is `GL_REPEAT` or `GL_CLAMP`
OpenGL Texturing

Define the texture filter parameters. This will control what happens when a texture is scaled up or down:

```c
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter);
gTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter);
```

where `wrap` is either `GL_NEAREST` or `GL_LINEAR`
OpenGL Texturing

Define the texture environment properties.

```cpp
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode);
```

where `mode` is either

- **GL_REPLACE** specifies that the 3-component texture will be applied as an opaque image on top of the polygon, replacing the polygon’s specified color.

- **GL_MODULATE** specifies that the 3-component texture will be applied as piece of colored plastic on top of the polygon. The polygon’s specified color “shines” through the plastic texture. This is very useful for applying lighting to textures: paint the polygon white with lighting and let it shine up through a texture.
OpenGL Texturing

In your Display() function:

1. If you are using multiple textures, set the current texture using `glTexImage2D(...)` or by binding the correct one

2. Enable texture mapping

   ```
   glEnable(GL_TEXTURE_2D);
   ```

3. Draw your polygons, specifying texture coordinate \((s, t)\) before each vertex

   ```
   glBegin(GL_POLYGON);
   
   glTexCoord2f(s0, t0);
   glNormal3f(nx0, ny0, nz0);
   glVertex3f(x0, y0, z0);
   
   glTexCoord2f(s0, t0);
   glNormal3f(nx0, ny0, nz0);
   glVertex3f(x0, y0, z0);
   
   ........
   
   glEnd();
   ```

4. Disable texture mapping

   ```
   glDisable(GL_TEXTURE_2D);
   ```
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

1 slice

5 slices

View direction

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
Computational Strategies

• How can the basic ingredients be combined:
  • Image Order (in screen coordinate)
    • Ray casting (many options)
  • Object Order (in world coordinate)
    • splatting, texture-mapping

• Combination (neither)
  • Shear-warp, Fourier
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