CS 5630/6630
Scientific Visualization

Volume Rendering III: Unstructured Grid Techniques
Unstructured Grids

- Image-space techniques
  - Ray-Casting
- Object-space techniques
  - Projected Tetrahedra
- Hybrid
  - Incremental Slicing
  - HAVS
Unstructured Grids

- Algorithms vs. Hardware

![Graph showing Unstructured Volume Rendering Algorithms vs. Hardware](image-url)
Ray-Casting

- Image-space technique
- Similar to regular grids, except:
  - Voxel locations are not implicit
  - Rays may enter/leave the volume multiple times

For each pixel:
  Cast a ray from pixel to volume
For each boundary intersection:
  Current cell = boundary cell
While current cell:
  Classify using transfer function
  Compute volume rendering integral
  Composite
  Current cell = next neighbor

[Garrit 90, Bunyk 00]
Ray-Casting

- Sampling
  - Where do you sample?
    - Cell boundaries
    - Internally to avoid artifacts
  - Which cell do you sample?
    - Use connectivity information
    - Next cell determined by plane intersection tests
      - First positive ray/plane intersection

[Garrity 90]
Ray-Casting

- Sampling
  - Where do you enter the mesh?
    - Intersections with boundary faces
  - Is there a faster way?
Ray-Casting

- Sampling
  - Where do you enter the mesh?
    - Intersections with boundary faces
  - Is there a faster way?
    - Use a grid to partition boundary faces [Garrity 90]
    - Store front boundary fragments [Bunyk 00]
Ray-Casting

- Advantages

- Disadvantages
Ray-Casting

- Advantages
  - It's simple!
  - No hardware constraints
  - Easily parallelized
  - Easily extended for multiple scattering
  - Visibility ordering is implicit

- Disadvantages
  - It's slow!
  - Must sample densely for high quality
Splatting

• Object-space technique
• Renders one cell at a time
• Requires visibility ordering

For each viewpoint
Sort cells in visibility order
For each cell in order:
  Decompose cell into footprints
  Classify footprints using transfer function
  Project footprints to image plane
  Composite
Visibility Order

- Meshed Polyhedra Visibility Order (MPVO)
  - Phase 1: Determine order relations

For each shared triangle:
  \[ d_1 = \text{dot(view normal, surface normal of cell 1)} \]
  \[ d_2 = \text{dot(view normal, surface normal of cell 2)} \]
  if \( d_1 < d_2 \):
    mark cell 1 as outbound
  else:
    mark cell 2 as outbound

For each cell with no outbound faces
mark cell as sink

[Williams 92]
Visibility Order

- Meshed Polyhedra Visibility Order (MPVO)
  - Phase 2: Topological Sort
    - Build a directed acyclic graph
    - Sort via depth first search or breadth-first search

  \(\text{For all sink cells:}\)
  \[\text{DFS(sink cell)}\]

  \(\text{DFS(currentCell)}:\)
  \[\text{currentCell.visited = true}\]
  \[\text{currentCell.cycle = false}\]
  for all neighbor cells \(c\) of \(\text{currentCell}\) not visited:
  \[\text{if } c.\text{cycle } \text{exit with error}\]
  \[\text{DFS}(c)\]
  \[\text{currentCell.cycle = false}\]
  \[\text{output currentCell}\]
Visibility Order

• Meshed Polyhedra Visibility Ordering (MPVO)

• Advantages:
  • Fast (linear)
  • Simple
  • Detects cycles

• Disadvantages:
  • Only works on convex meshes with one connected component

• Extensions: MPVONC, XMPVO, SXMPVO, etc.
Projected Tetrahedra

- Decompose tetrahedra into renderable triangles
- Assign color and opacity to each vertex of triangle
- Render triangles and composite results onto image plane

[Shirley and Tuchman 90]
Projected Tetrahedra

- Cases determined by direction of faces, i.e., dot(view vector, surface normal)
Projected Tetrahedra

- Advantages:

- Disadvantages:
Projected Tetrahedra

• Advantages:
  • Uses graphics hardware for rendering and compositing
  • Per-pixel exact rendering

• Disadvantages:
  • Difficult to achieve more advanced effects
  • Inaccuracies due to object-space sorting (visibility cycles)
Incremental Slicing

- Hybrid technique
- Similar to texture slicing for structured grids
- Sorts the tetrahedra in image space
- Renders slices in object-space

For each viewpoint
  - Project vertices to image-space
  - Determine the z limits, \( z_{\text{max}} \) and \( z_{\text{min}} \), of the vertices
  - \( dz = (z_{\text{max}} - z_{\text{min}})/\text{NumSlices} \)
  - \( z_{\text{slice}} = z_{\text{min}} \)
  - Bucket sort cells by z
  - For each slice in NumSlices:
    - Intersect the transformed cells with the plane \( Z = z_{\text{slice}} \)
    - Render the resulting mesh of 2D polygons
    - Composite into the image plane
    - \( z_{\text{slice}} += dz \)

[Yagel et al. 96]
Incremental Slicing

- Intersections
  - Traverse the cells front-to-back using
  - Keep an active tetrahedra list and an active edge list
  - Use adjacency information to march through intersections
Incremental Slicing

- 2D polygons
  - Each slice of a tetrahedra becomes a triangle or a quadrilateral
Incremental Slicing

- Advantages

- Disadvantages
Incremental Slicing

- Advantages
  - Uses hardware for rendering
  - Adaptive - NumSlices can be easily changed
  - Progressive - Intermediate steps can be shown

- Disadvantages
  - Lower quality - it only approximates the volume due to limited number of slices
HAVS

- Hybrid technique
- Hardware-Assisted Visibility Sorting
- Decomposes tetrahedra into triangles
- Partially sorts triangles in object-space
- Finishes sort on fragments in image-space

Store tetrahedral mesh as a list of triangles
For each viewpoint
  Radix sort triangles by centroid depth
  Render triangles in order
For each fragment \( f = (s, d) \):
  Add \( f \) to k-buffer
  Select closest fragments from k-buffer \( f_1 \) and \( f_2 \)
  Lookup color and opacity \((f_1.s, f_2.s, f_2.d - f_1.d)\)
  Composite
  Remove \( f_1 \) from k-buffer

[Callahan et al. 05]
HAVS

- Object-space sorting
  - Performed on CPU
  - Based on triangle centroids
  - Least significant digit radix sort
HAVS

- Image space sorting
  - Performed on GPU
  - k-buffer: Fixed size sorting buffer
    - Capable of sorting a list with no elements more than k out of place
HAVS

- Advantages

- Disadvantages
HAVS

• Advantages
  • Fast!
  • Dynamic - no connectivity information required
  • Adaptive - easily extended to render subset of triangles
  • Progressive - triangles can be stored out-of-core

• Disadvantages
  • Requires features not supported by all hardware
  • Artifacts may appear
    • Insufficient k
    • Read/modify/write hazards
Acceleration Techniques

- GPU programming:
  - Ray-casting  [Weiler et al. 03]
  - Projected tetrahedra  [Wylie et al. 02]
  - Incremental slicing  [Georgii and Westermann 06]
- Level-of-detail
  - Domain-based  [Cignoni et al. 04]
  - Sample-based  [Callahan et al. 05]

![Images of acceleration techniques](image-url)
Unstructured Grids

- VisTrails demo
Summary

- Unstructured Grids
  - Ray-casting is easiest to add advanced effects
  - Projected tetrahedra is most correct
  - Incremental slicing is faster and most easily made progressive
  - HAVS is fastest and most dynamic