Visualization Techniques

Volume Rendering

Isosurface Generation

Cláudio T. Silva
Outline

• Introduction to Volume Rendering
• Out-Of-Core Techniques
  • Introduction to External Memory Algorithms
  • Isosurface Generation
  • Direct Volume Rendering
• Hardware-Assisted Techniques
  • Introduction to Programmable Hardware
  • Direct Volume Rendering
  • Isosurface Generation (remarks)
Introduction to Volume Rendering
Grid Types

- Regular
- Curvilinear
- Rectilinear
- Irregular
Volume Rendering vs Isosurfaces

Direct Volume Rendering
- Volume data $\rightarrow$ Image
- Looks “inside” the data

Isosurfaces
- Volume data $\rightarrow$ Polygon model
- Slice of the data
Isosurfaces

For a query value $q$, find and display the isosurface of $q$: $C(q) = \{ p \mid F(p) = q \}$
Isosurface Extraction Pipeline

**Search Phase**
- \(O(N)\) Volume Data
  - Search Phase: \(N\) cells

**Generation Phase**
- \(O(K), K = N^{(2/3)}\) Triangles
  - Generation Phase: \(K\) active cells
  - Marching Cubes
  - Triangles
  - Triangles
  - simplification
  - stripping

**Display**
- Triangle Strips
  - Display

**Notes**
- \(O(N)\) complexity for search phase
- \(O(K), K = N^{(2/3)}\) complexity for generation phase
Scalar Intervals and Active Cells

Produce Interval $I = [f_{\text{min}}, f_{\text{max}}]$ for each cell

Active cells
= cells intersected by the isosurface
= cells with $f_{\text{min}} \leq \text{isovalue } q \leq f_{\text{max}}$

[Cignoni et al 96]
Interval Tree [Edelsbrunner 81]

Intervals: crossing slab boundary -- u, else -- child
left list: left(u) = (2, 0, 1)
right list: right(u) = (1, 2, 0)
Isosurface Acceleration

Span Space [Livnat et al]
Seed Cells [Bajaj et al, and others]
Optimal Contour Trees [Chiang et al, Carr et al]
Lots of other work…
Direct Volume Rendering: Optical Models

\[ I(s + \Delta s) = I(s) + g(s)\Delta s - I(s)\Omega(s)\Delta s \]

\[ I(s) = \int_{0}^{x} g(x)e^{-\int_{y}^{\Omega(y)dy} \frac{x}{0} \ dx} \]
Volume Rendering at a High Level

(a) Sampling Phase
(b) Sorting Phase
Unstructured Grid Volume Rendering

Ray Tracing

Viewing Direction

Screen

Scientific Computing and Imaging Institute, University of Utah
Outline

• Introduction to Volume Rendering
• Out-Of-Core Techniques
  • Introduction to External Memory Algorithms
  • Isosurface Generation
  • Direct Volume Rendering
• Hardware-Assisted Techniques
  • Introduction to Programmable Hardware
  • Direct Volume Rendering
  • Isosurface Generation (remarks)
Principles of External Memory Algorithms

I/O Computational Model

Algorithmic Techniques

• Caching & Prefetching
  - [Cox-Ellsworth 1997; Sutton-Hansen 2000; Bajaj et al 1999; etc]
• External Merge Sort
• Out-Of-Core Pointer De-Referencing
• Meta-Cell Technique [Chiang and Silva 1998]
• Indexing (B-Tree-Like Data Structures)
I/O Computational Model

- **B**: # items fitting in one disk block
- One I/O reads/writes one block (B items)
- Virtual memory vs Out-of-core techniques
- Batched computations (e.g. external sorting)
- On-line computations (e.g. B-tree)
Caching & Prefetching

node state
- hit
- missed
- prefetched
- replaced
Meta-Cell Technique [Chiang and Silva 1998]

Out-of-core technique for creating spatially coherence geometry for unstructured grids

216 meta-cells

1000 meta-cells
Preprocessing Pipeline

- Volume dataset
- Meta-cell Computation
- Meta-intervals
- Tree Construction
- Isosurface Extraction
- Binary-blocked I/O interval tree
- Meta-cells
Isosurface Query Pipeline

- iso-value \( q \) → interval searching → full dataset
- active meta-cell ID’s → disk read → meta-cells
- active meta-cells → generation phase (use Vtk) → isosurface
Out-Of-Core Direct Volume Rendering

- Sampling Phase does not change
- The part that changes is the **sorting phase**
Sorting

Application

Object Space

Rasterization

Image-Space Sorting

Display

i.e., let’s sort the pixes!
Image-Order Sorting: Carpenter’s A-Buffer

Idea: Keep a list of intersections for each pixel
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer
Cell-Projection With An A-Buffer

Not sorted!
Cell-Projection With An A-Buffer

Sorted!
A Simple Memory-Insensitive Technique

Data

Dereferenced

Sampling

Sorted

Compose

\[ \begin{align*}
X_0, Y_0, Z_0 \\
X_1, Y_1, Z_1 \\
\vdots \\
X_n, Y_n, Z_n \\
\end{align*} \]

\( n \) vertices

\[ \begin{align*}
I_{00}, I_{01}, I_{02}, I_{03} \\
I_{10}, I_{11}, I_{12}, I_{13} \\
\vdots \\
I_{t0}, I_{t1}, I_{t2}, I_{t3} \\
\end{align*} \]

\( t \) tetrahedra
Data

Dereferenced

Sampling

Sorted

Compose

XYZ

Y

Z

00 01 02 03

10 11 12 13

t0 t1 t2 t3

XYZ

Y

Z

00 01 02 03

10 11 12 13

...
Data → Dereferenced → Sampling → Sorted → Compose

Pix Z Alpha

Pix Z Alpha

Pix Z Alpha

Pix Z Alpha
Features of Memory-Insensitive Algorithm

- Dataset can be arbitrarily large
- Easily to implement
  - Cell intersection code
  - Sample Integration code
  - External Memory sort
- Transfer function modifications for same viewpoint are fast! (also for time-varying data)
Complexity Analysis

Number of Intersections: $O(c N^2)$

Problems:
1. Time: sorting intersections takes too long!
2. Memory: storage too high!
Sorting

- Application
- Object-Space Sorting
- Rasterization
- Image-Space Sorting
- Display
Approximate Object-Space Sorting
A Solution: Use an insertion-sort A-buffer!
What about the space problem?

→ Use a conservative bound on the intersections
ZSWEEP: Sort in Image and Object Space

Do an approximate sorting of the cells (e.g., based on the order of the vertices), and use an insertion sort for the intersections – solves the sorting overhead.

Propose a conservative technique for limiting the memory usage, by performing compositing on the intersections, anytime the lists become “too large” – solves the memory overhead.
Blocking for Out-Of-Core Rendering

Chiang, Silva, Schroeder 1998
Out-Of-Core ZSWEEP: Data Management

[Farias and Silva 2001a]

- fetch B
- dump B
- Meta-cell
- Sweep Plane
- Event: fetch A
- dump A
Tile-Based ZSWEEP

SGI R10K  L2 – 2MB
ZSWEEP:   56% hit rate
TB-ZSWEEP: 94% hit rate
## Out-Of-Core ZSWEEP Results

### Generating 2048x2048 Image (sec)

<table>
<thead>
<tr>
<th></th>
<th>Original ZSWEEP</th>
<th>OOC-ZSWEEP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory</td>
<td>Time</td>
</tr>
<tr>
<td><strong>Blunt Fin</strong></td>
<td>330</td>
<td>386</td>
</tr>
<tr>
<td><strong>Combustion Chamber</strong></td>
<td>330</td>
<td>407</td>
</tr>
<tr>
<td><strong>Oxygen Post</strong></td>
<td>350</td>
<td>775</td>
</tr>
<tr>
<td><strong>Delta Wing</strong></td>
<td>380</td>
<td>639</td>
</tr>
</tbody>
</table>
Video

QuickTime™ and a
Photo Decompressor
are needed to see this picture.
Outline

• Introduction to Volume Rendering
• Out-Of-Core Techniques
  • Introduction to External Memory Algorithms
  • Isosurface Generation
  • Direct Volume Rendering
• Hardware-Assisted Techniques
  • Introduction to Programmable Hardware
  • Direct Volume Rendering
  • Isosurface Generation (remarks)
Hardware-Assisted Volume Rendering

- Introduction to Programmable Hardware
- Texture-Based Volume Rendering
- Rendering Unstructured Grids
Introduction to Programmable Hardware

- Graphics Pipeline
- Programmable Graphics Pipeline
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Colored Fragments

Pixel Updates

Raster Operations

FB
### Graphics Pipeline

**3D Application or Game**

3D API commands:

```c
glBegin(GL_TRIANGLES);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(0.5, 1.0, 0.0);
...
glEnd();
```

**3D API:**
- OpenGL or
- Direct 3D

**Vertices**

- **Vertex Transformation**
- **Primitive Assembly and Rasterization**
- **Fragment Texturing and Coloring**
- **Raster Operations**

**Colored Fragments**

**Pixel Updates**

**FB**
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Assembly and Rasterization

Fragment Texturing and Coloring

Colored Fragments

Pixel Updates

Raster Operations

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

1 2 3 1

1 2 3

1

2

3

4

Fragments

Colored Fragments

Pixel Updates

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API: OpenGL or Direct 3D

4 Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragments

Fragment Texturing and Coloring

Colored Fragments

Pixel Updates

Raster Operations

FB

Joao Comba, UFRGS
### Graphics Pipeline

1. **3D Application or Game**
   - 3D API commands

2. **3D API:**
   - OpenGL or Direct 3D

3. **Vertices**
   - Transformed Vertices

4. **Vertex Transformation**

5. **Primitive Assembly and Rasterization**

6. **Fragments**

7. **Fragment Texturing and Coloring**

8. **Colored Fragments**

9. **Raster Operations**

10. **Pixel Updates**

11. **FB**

---

Joao Comba, UFRGS

---

Scientific Computing and Imaging Institute, University of Utah
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

3' 2' 1'

Fragments

Fragment Texturing and Coloring

Colored Fragments

Raster Operations

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game -> 3D API commands

3D API: OpenGL or Direct 3D

Vertices -> Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Colored Fragments

Pixel Updates

Raster Operations

FB

Joao Comba, UFRGS

Scientific Computing and Imaging Institute, University of Utah
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

3 4

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

3’ 1’

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API: OpenGL or Direct 3D

3D API commands

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

Fragments

Transformed Vertices

Vertices

3D Application or Game

3D API commands
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Fragments

Colored Fragments

Pixel Updates

FB

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Fragments

Colored Fragments

Pixel Updates

FB

Joao Comba, UFRGS
Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB
Graphics Pipeline: Parallelization

- 3D Application or Game
- 3D API: OpenGL or Direct 3D
- Vertices
- Transformed Vertices
- Fragment Assembly and Rasterization
- Fragment Texturing and Coloring
- Raster Operations
- Pixel Updates
- Colored Fragments
- FB

Joao Comba, UFRGS
Graphics Pipeline: Parallelization

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Vertex Transformation

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

Colored Fragments

Pixel Updates

FB
Graphics Pipeline: Parallelization

3D Application or Game

3D API: OpenGL or Direct 3D

3D API commands

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Colored Fragments Updates

Raster Operations

FB

Joao Comba, UFRGS
Graphics Pipeline: Parallelization

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Transformation

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Fragment Texturing and Coloring

Colored Pixels

Raster Operations

FB

Joao Comba, UFRGS
Programmable Graphics Pipeline

- 3D Application or Game
  - 3D API commands
  - 3D API: OpenGL or Direct 3D
- Vertices
  - Transformed Vertices
  - Programmed Vertices
  - Programmable Vertex Processor
- CPU - GPU Boundary
  - Fragments
  - Colored Fragments
  - Pixel Updates
  - Raster Operations
  - FB
- 3D Application or Game
  - 3D API commands
  - 3D API: OpenGL or Direct 3D
- Joao Comba, UFRGS
Programmable Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Fractals

Fragments

Colored Fragments

Pixel Updates

CPU - GPU Boundary

3D API commands

Verte Processor

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

FB

Program

Joao Comba, UFRGS
Programmable Graphics Pipeline

3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D

Vertices
Transformed Vertices

Vertex Processor
Primitive Assembly and Rasterization

Fragment Processor
Fragment Texturing and Coloring

Programmable Fragment Processor
Raster Operations

Colored Fragments
Pixel Updates

CPU - GPU Boundary

3D Application or Game
3D API commands
3D API: OpenGL or Direct 3D

Vertices
Transformed Vertices

Vertex Processor
Primitive Assembly and Rasterization

Fragment Processor
Fragment Texturing and Coloring

Programmable Fragment Processor
Raster Operations

Colored Fragments
Pixel Updates

CPU - GPU Boundary
Programmable Graphics Pipeline

3D Application or Game → 3D API commands → 3D API: OpenGL or Direct 3D → Vertices → Transformed Vertices → Primitive Assembly and Rasterization

3D API commands

CPU - GPU Boundary

Vertices → Transformed Vertices

Fragment Processor

Fragment Processor

Fragment Processor

3D Application or Game

3D API: OpenGL or Direct 3D

Transformed Vertices

Primary Assembly and Rasterization

Colored Fragments

Pixel Updates

Raster Operations

FB

Program

Joao Comba, UFRGS
Programmable Graphics Pipeline

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

Transformed Vertices

Vertex Processor

Primitive Assembly and Rasterization

Fragment Processors

Fragment Processor

Fragment Processor

Fragment Processor

Fragment Processor

Colored Fragments

Pixel Updates

CPU - GPU Boundary

Raster Operations

FB

Joao Comba, UFRGS
Graphics Pipeline: The Big Picture

3D Application or Game

3D API commands

3D API: OpenGL or Direct 3D

Vertices

CPU - GPU Boundary

Transformed Vertices

Fragment Processor

Primitive Assembly and Rasterization

Fragment Processor

Colored Fragments

Pixel Updates

Raster Operations

FB

Vertex Processor
Graphics Pipeline: The Big Picture

3D Application or Game

3D API: OpenGL or Direct 3D

3D API commands

Vertices

Transformed Vertices

Vertex Processor

Primitive Assembly and Rasterization

CPU - GPU Boundary

Fragment Processor

Colored Fragments

Pixel Updates

Raster Operations

FB

GPU is a stream processor

- Multiple programmable processing units
- Connected by data flows

Joao Comba, UFRGS
Programmable Graphics Pipeline

3D Application or Game

3D API: OpenGL or Direct 3D

Transformed Vertices

Primitive Assembly and Rasterization

Fragment Texturing and Coloring

Raster Operations

CPU - GPU Boundary

Pixel Updates

Colored Fragments

Fragments

Transformed Vertices

Program

3D API commands

Joao Comba, UFRGS
Texture-Based Volume Rendering

Slice Decomposition (Proxy Geometry)  Rendering of textures slices  Final Image

Trilinear Hardware Interpolation  Compositing (Blending)

Rezk-Salama Visualization 2002 Tutorial
Rendering Unstructured Grids

Brian Wylie, Sandia

model with millions of cells

Visibility Sort

Software

for each cell in order

decompose to triangles

compute each triangle's parameters

find thickest cell distance

compute cell's screen projection

PC (CPU)

graphics card

final image of model

GPU

Cell Contribution

Programmable Hardware

Scientific Computing and Imaging Institute, University of Utah
Visibility Sorting
Williams’ MPVO

Idea: Define ordering relations by looking at shared faces.

- $B < A$
- $A < C$
- $B < E$
- $C < E$
- $C < D$
- $E < F$
- $D < F$

Viewing direction
Shirley-Tuchman (ST) Algorithm

Class 1

Class 2

Class 3

Class 4

Brian Wylie, Sandia
Wylie et al’s GPU-based ST

Moves all of the following functions from the CPU to the GPU:

- Transform to screen space
- Determine projection class
- Calculate thick vertex location
- Determine depth at thick vertex
- Compute color and opacity for thick vertex
- Apply exponential attenuation texture
GPU Limitations

- Each instance of a *vertex shader* program works independently on a single vertex in SIMD fashion.
- No support for dynamic vertex creation or topology modification within the vertex program.
- No branching.
- No knowledge of neighboring vertices.
- Cannot change execution based on past information.
Idea: Morph a Canonical Graph

Basis Graph

Isomorphic to all projection cases

Example later…
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Brian Wylie, Sandia
PT algorithm in Vertex Program

- Transform to screen space. (Trivial)
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Brian Wylie, Sandia
Projection Classes

Class 1

Class 2

Class 3

Class 4

Brian Wylie, Sandia
Projection Permutations

Permutation Determination
14 cases need at least 4 Boolean tests

Definitions
vec1 = v1-v0
vec2 = v2-v0
vec3 = v3-v0
cross1 = vec1 x vec2
cross2 = vec1 x vec3
cross3 = vec2 x vec3

Tests
test1 = (cross1*cross2 < 0)
test2 = (cross1*cross3 > 0)
test3 = (distance from v0 to middle vertex – distance from v0 to Intersection) > 0
test4 = (cross1 > 0)
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Brian Wylie, Sandia
Isomorphic Property of Basis Graph

Object Space

Screen Space (Case 5 projection)

Triangle Output

Map to Basis Graph (V₀ maps to both V₀' and V₃')

(Coincident points V₀' and V₃' create degenerate triangle)
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture).
- Multiplex the result to correct output vertex.

Brian Wylie, Sandia
In all cases the coordinates of the intersection point I are computed. (Intersection of lines computed ala Graphics Gems III p. 199-202). This intersection calculation gives us $\alpha$ and $\beta$ terms that are used for interpolation (depth, alpha, and color) later on.

Class 2:

```
// Compute thick vertex "thickness"
float thickness = fabs(z1-z2);
```

Class 1:

```
// Extra computation for class 1
if (!test3) thickness /= alpha;
```
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture).
- Multiplex the result to correct output vertex.

Brian Wylie, Sandia
Color and Opacity Calculation

- Use same $\alpha$ and $\beta$ terms to interpolate color and opacity along the line segments to give the front and back face intersection terms $C_F$, $C_B$ and $\tau_F$, $\tau_B$.

- Thick vertex color $(C_F + C_B) / 2$ *

- The extinction coefficient $\tau$ is $(\tau_F + \tau_B) / 2$.

- $\tau$ and the thickness $l$, are then used as lookups into a 2D texture map defined as $1 - \exp (-\tau l)$. [Stein et al. 1994].

* approximate color from Shirley and Tuchman.

Brian Wylie, Sandia
PT algorithm in Vertex Program

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the basis graph.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.
Multiplex input to output

Use a lookup table (loaded in the parameter registers) and an index based on the 4 tests to determine the output vertex.

```c
// Which vertex to copy to output (using lookup table)
lookup_index = test1*8 + test2*4 + test3*2 + test4;
output_vertex = lookup_table[call_index][lookup_index];
```
`Feeding’ the Vertex Program

Brian Wylie, Sandia

```c
// Load up the 4 vertices
glVertexAttrib3fvNV(1, nodes[0]->getXYZ());
glVertexAttrib3fvNV(2, nodes[1]->getXYZ());
glVertexAttrib3fvNV(3, nodes[2]->getXYZ());
glVertexAttrib3fvNV(4, nodes[3]->getXYZ());

// Load up color for the vertices
glVertexAttrib4fvNV(5, colorvectors[0]);
glVertexAttrib4fvNV(6, colorvectors[1]);
glVertexAttrib4fvNV(7, colorvectors[2]);
glVertexAttrib4fvNV(8, colorvectors[3]);

// Writing to v[0] here invokes the vertex program.
glBegin(GL_TRIANGLE_FAN);
    glVertexAttrib3sNV(0, 0, 1, 0);
    glVertexAttrib3sNV(0, 1, 0, 1);
    glVertexAttrib3sNV(0, 2, 0, 1);
    glVertexAttrib3sNV(0, 3, 0, 1);
    glVertexAttrib3sNV(0, 1, 0, 1);
    glVertexAttrib3sNV(0, 4, 0, 1);
    glVertexAttrib3sNV(0, 1, 0, 1);
    glVertexAttrib3sNV(0, 1, 0, 1);
glEnd();

These calls could easily be wrapped up into a glTetraExt() call.
```
Remarks

Wylie et al’s technique can be easily extended to other computations, e.g., isosurface or isoline generation (this was a homework exercise in my graphics class last Spring)

For isosurfaces, one can send two triangles (four vertices in a strip), since for a tetrahedral cell, the isosurface going through it has at most two triangles

One student (Francis Chang) found a clever solution with 30 instructions!
Acknowledgements

• DOE VIEWS, DOE MICS, SNL, and LLNL

• National Science Foundation
  • CCR-0306530 and EIA-0323604

• Joao Comba, Ricardo Farias, Brian Wylie, and others for slides and help with presentation