Parallel Rendering

Why Parallelism

Applications need:
- High frame rates
- High resolution
- Large geometric models
- Stereo
- Antialiasing
- etc.

Performance implications:
- Hundreds of MFLOPS compute power
- Gigabytes per second memory bandwidth
Stages of Parallelism (for object-order rendering)

- Graphics database traversal
- Geometry processing
- Rasterization
- Display
Processing Tasks

Geometry Processors

- Each processor gets a subset of primitives
- Transformation
- (Lighting)
- Set-up for Rasterization

Rasterization Processors

- Each processor gets a subset of pixels
- Visibility computation
- Shading
Rendering as Sorting

- Primitives may lie anywhere on or off screen
- Determine effect of each primitive on each pixel
- Primitives are “sorted” onto screen
- Sorting affects distribution of data on geometry and rasterization processors
Primitives in Screen-space Regions
Where to sort

Sort Middle

• Sort between geometry processing and rasterization

Sort First

• Sort during geometry processing

Sort Last

• Sort during rasterization

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen
Sort Middle

Graphics database
(arbitrarily partitioned)

Geometry processing

Redistribute screen-space primitives

Rasterization

Display

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen
Sort Middle: Data Arrangement

Geometry processors
• Arbitrary (random) distribution of primitives
• Good for load balancing

Rasterization processors
• Screen-space distribution of primitives
• Load balancing difficult
Sort Middle: Communications

All geometry transformed primitives must be communicated every frame

All geometry processors must communicate with all rasterization processors

• $O(n^2)$ communications paths
Sort First

Graphics database
(arbitrarily partitioned)

Redistribute "raw"
primitives

(Pre-transform)

Geometry
processing

Rasterization

Display
Sort First: Data Arrangement

Distribute both geometry and rasterization work according to position of primitives on screen

Load balancing difficult

- Different screen regions of equal sizes may contain different numbers of primitives
- May need dynamic region sizes
Sort First: Communications

Must determine primitive screen coverage before full transformation

Exploit frame-to-frame coherence

- Shuffle primitives between geometry processors only if screen coverage changes

Possibly employ primitive clustering and bounding volumes

- Pre-transform bounding volumes for small groups of primitives
Sort Last

Graphics database
(arbitrarily partitioned)

Geometry processing

Rasterization

Redistribute pixels,
samples, or fragments

(Compositing)

Display
Sort Last: Data Arrangement

Arbitrary (random) arrangement of data on both geometry and rasterization processors

Great for load balancing

Each rasterization processor makes image of entire screen, with subset of primitives
Sort Last: Communications

Rasterization processors must communicate final pixel data

Composition of pixel data may take place along linear or tree-shaped network

Requires high bandwidth, assuming pixel data is much larger than primitive data
## Advantages and Disadvantages

<table>
<thead>
<tr>
<th>SF</th>
<th><strong>Advantages</strong></th>
<th><strong>Disadvantages</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Low communications when good coherence</td>
<td>• Susceptible to load imbalance</td>
</tr>
<tr>
<td></td>
<td>• Each processor implements entire pipeline</td>
<td>• Retained mode and complex data handling</td>
</tr>
<tr>
<td>SM</td>
<td>• General and straightforward</td>
<td>• High communication cost</td>
</tr>
<tr>
<td></td>
<td>• Natural communications placement</td>
<td>• Rasterizer load imbalance</td>
</tr>
<tr>
<td>SL</td>
<td>• Each processor implements entire pipeline</td>
<td>• Large communication cost, especially for high resolution or multisampling</td>
</tr>
<tr>
<td></td>
<td>• Easier load balancing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Linear scalability</td>
<td></td>
</tr>
</tbody>
</table>

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen