Toward A Multicore Architecture for Real-time Ray-tracing

Venkatraman Govindaraju*, Peter Djeu+, Karu Sankaralingam*, Mary Vernon*, William R. Mark+

Vertical Research Group
* University of Wisconsin-Madison
+ University of Texas at Austin

http://www.cs.wisc.edu/vertical
What is Wrong with this Picture?

Moon

Mercury

Venus

Sun
What is Wrong with this Picture?

Ptolemy's Earth-centric Universe

- Sun
- Venus
- Mercury
- Moon

Ptolemy Earth-centric Universe
Like the Ptoleemic Universe, we claim GPUs have it wrong for future 3D graphics
Ptolemic 3D Graphics Universe

Z-buffer
Ptolemic 3D Graphics Universe

API

Architecture

Z-buffer
Ptolemic 3D Graphics Universe

Effects

API

Architecture

Z-buffer

Shadows

Reflections
Ptoleemic 3D Graphics Universe

Applications

Games

Shadows

Reflections

API

Architecture

Effects

Z-buffer
Center of the Graphics Universe

The Z-buffer

1) Regularly spaced rays
2) Originating from a single point
Center of the Graphics Universe

The Z-buffer

1) Regularly spaced rays
2) Originating from a single point
3) Z-buffer is implemented in H/W
GPUs Have it Wrong!

- Z-buffer centric Ptolemic universe
  - ✓ Model was correct thus far
  - ✗ Scene constraints
  - ✗ Poor productivity
  - ✗ Limited image quality
GPUs Have it Wrong!

- Z-buffer centric Ptolemaic universe
  - ✓ Model was correct thus far
  - ✗ Scene constraints
  - ✗ Poor productivity
  - ✗ Limited image quality

- Not suited for future
  - Graphics algorithms have evolved
  - VLSI technology has changed
GPUs Have it Wrong!

- Z-buffer centric Ptolemic universe
  - Model was correct thus far
  - Scene constraints
  - Poor productivity
  - Limited image quality

- Not suited for future
  - Graphics algorithms have evolved
  - VLSI technology has changed

What is the future?
A Copernican 3D Graphics Universe?
A Copernican 3D Graphics Universe

Ray-tracing
A Copernican 3D Graphics Universe

Multi-core architecture

Ray-tracing
A Copernican 3D Graphics Universe

High-quality images

Multi-core architecture

Ray-tracing
Executive Summary
Executive Summary

- *Ptolemy* GPU model insufficient

- Kopernicus: An entire graphics system stack
  - Algorithms, software, and architecture
  - Leverage programmable multicores
  - Recomputation vs. synchronization
  - Real-time, dynamic scenes, and high quality effects

- Evaluation using analytical models
  - Accurate and extensible
  - Scales to 128 cores
Outline

- Motivation
- Raytracing Background
- Copernicus System
  - Software implementation: Razor
  - Architecture
  - Evaluation and Results
- Summary
Raytracing Overview (1)

Simulate the behavior light rays through 3D scene
Raytracing Overview (1)

- Primary rays
- Secondary rays
Raytracing Overview (1)

Primary rays may hit no object
Raytracing Overview (1)

Primary rays may hit an object
Raytracing Overview (1)

Cast secondary rays to light source(s)
Raytracing Overview (1)

Primary rays
Secondary rays

May hit light source $\Rightarrow$ not in shadow
May hit another object $\Rightarrow$ in shadow
Raytracing Overview (1)

- Primary rays
- Secondary rays
Raytracing Overview (2)

Ray-object intersections is key operation

Acceleration data structure (kd-tree) for efficiency
Raytracing Overview (2)

Ray-object intersections is key operation
Acceleration data structure (kd-tree) for efficiency
Real-time Raytracing is Hard

- Objects move and change in dynamic scenes
  - Per-frame rebuild of acceleration structure

- Irregular data accesses to acceleration structure
  - Pointer-based accesses

- Data structures are large
Outline

- Motivation
- Raytracing Background
- Copernicus System
  - Software implementation: Razor
  - Architecture
  - Evaluation and Results
- Summary
Razor: A Dynamic Multiresolution Raytracer

- Primary rays
- Secondary rays
Razor: A Dynamic Multiresolution Raytracer

- Packet ray-tracer
  - Fine-grained data-level parallelism
Razor: A Dynamic Multiresolution Raytracer

- Packet ray-tracer
  - Fine-grained data-level parallelism
- Multithreaded
  - Coarse-grained parallelism
Razor: A Dynamic Multiresolution Raytracer

- Packet ray-tracer
  - Fine-grained data-level parallelism
- Multithreaded
  - Coarse-grained parallelism
- Per-thread acceleration structure (kd-tree)
  - Lazy build
  - Replicate kd-tree to reduce synchronization
Full System Co-design

Algorithm & Application

Architecture
Full System Co-design

Algorithm & Application

Architecture
Software Implementation

- Designed with future hardware in mind
Software Implementation

- Designed with future hardware in mind
- Algorithm-centric implementation
  - Runs on Intel Clovertown
  - Optimized with SSE instructions
  - 1 FPS on this prototype system

- Architecture-centric flexible implementation
  - Runs on Solaris and simulators
  - SSE “emulation”
Razor Characterization: Parallel Scalability

![Graph showing speedup vs. number of threads for different environments.]

- Courtyard
- Fairyforest
- Forest
- Juarez
- Saloon
Razor Characterization: Parallel Scalability

Good parallel scalability
Razor Characterization: Memory

![Graph showing normalized memory usage vs. number of threads for different environments: Courtyard, Fairyforest, Forest, Juarez, and Saloon.](image)
Razor Characterization: Memory

Normalized Memory Usage vs Number of Threads

- Courtyard
- Fairyforest
- Forest
- Juarez
- Saloon

Memory growth limited
Outline

- Motivation
- Raytracing Background
- Copernicus System
  - Software implementation: Razor
  - Architecture
  - Evaluation and Results
- Summary
Copernicus Architecture
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
  - Large memory footprint
  - Replicate kd-tree
    - Reduce synchronization
  - Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
  - Replicate kd-tree
    - Reduce synchronization
  - Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
  - Replicate kd-tree
    - Reduce synchronization
  - Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
  - Replicate kd-tree
    - Reduce synchronization
  - Packets of rays

But need 80 to 100 cores for real-time!

This organization does not scale
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays

Use tile as building-block
- Private L2-caches
- No global coherence
- Trivially scalable
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays
Copernicus Architecture

- Multithreaded
- Large memory footprint
- Replicate kd-tree
  - Reduce synchronization
- Packets of rays

Fine-grained data-level parallelism
Addressed in core
Copernicus Architecture

- In-order core
- Private L1 Data cache
- Private L1 Inst. cache
Copernicus Architecture

- In-order core
- Private L1 Data cache
- Private L1 Inst. cache
- SMT to hide memory latency
Copernicus Architecture

- In-order core
- Private L1 Data cache
- Private L1 Inst. cache
- SMT to hide memory latency
- SIMD instructions
  - Packets of rays $\Rightarrow$ data-level parallelism
Architecture Summary

- 16 tile chip
  - 8 cores per tile
  - 2MB cache per tile
- 4-wide SIMD unit
- Shared accelerator
S/W Mapping to Architecture

- 16 tile chip
  - 8 cores per tile
  - 2MB cache per tile
- 4-wide SIMD unit
- Shared accelerator
S/W Mapping to Architecture

- 16 tile chip
  - 8 cores per tile
  - 2MB cache per tile

- 4-wide SIMD unit
- Shared accelerator
S/W Mapping to Architecture

- 16 tile chip
  - 8 cores per tile
  - 2MB cache per tile

- 4-wide SIMD unit
- Shared accelerator

Assigned to Tile

Assigned to Core
(128 pixels in block)
Outline

- Motivation
- Raytracing Background
- Copernicus System
  - Software implementation: Razor
  - Architecture
  - Evaluation and Results
- Summary
Benchmark Scenes

- Courtyard
- Fairyforest
- Forest
- Juarez
- Saloon

- 1024x1024
- 25 FPS
- 8 sec-rays per pri-ray
- 2 lights
Evaluation Methodology

- Modified Multifacet/GEMS

- Core simulator
  - SSE instructions
  - In-order SMT support

- Tile simulator
  - Shared L2, up to 8 cores per tile
  - “Validated” against prototype system
    - Pin-based and PAPI-based performance counters

- Full chip simulator
  - Simulating full chip is too slow and little insight
  - Build customized analytic model
1-Core Performance (Single Issue)

IPC
1-Core Performance (Single Issue)

IPC

<table>
<thead>
<tr>
<th>No SMT</th>
<th>2 SMT</th>
<th>4 SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courtyard</td>
<td>Fairyforest</td>
<td>Forest</td>
</tr>
</tbody>
</table>

- Courtyard: 0.44
- Fairyforest: 0.42
- Forest: 0.88
- Juarez: 0.62
- Saloon: 0.70
1-Core Performance (Single Issue)

<table>
<thead>
<tr>
<th></th>
<th>No SMT</th>
<th>2 SMT</th>
<th>4 SMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courtyard</td>
<td>0.40</td>
<td>0.80</td>
<td>1.00</td>
</tr>
<tr>
<td>Fairyforest</td>
<td>0.40</td>
<td>0.70</td>
<td>1.00</td>
</tr>
<tr>
<td>Forest</td>
<td>0.40</td>
<td>0.80</td>
<td>1.00</td>
</tr>
<tr>
<td>Juarez</td>
<td>0.40</td>
<td>0.80</td>
<td>1.00</td>
</tr>
<tr>
<td>Saloon</td>
<td>0.40</td>
<td>0.80</td>
<td>1.00</td>
</tr>
</tbody>
</table>
1-Core Performance (Single Issue)

4-wide SMT achieves close to 100% utilization
Sensitivity Study Insights

- Dual-issue not worth the area overhead

- 8 cores in tile reach close to 100%
  - SMT can hide L2 contention and miss latency

- Custom analytic model
  - Guided recomputation vs. synchronization approach
  - Memory bandwidth is key constraint
Full Chip Performance

- Measured in millions of rays/second
- Increases with # tiles
- Limited by bandwidth
Full Chip Performance

Measured in millions of rays/second
Increases with # tiles
Limited by bandwidth

![Graph showing Full Chip Performance]

- Ideal
- 1 DIMM
- 2 DIMMs
- 3 DIMMs
- 4 DIMMs

Graph indicates performance measured in millions of rays/second increases with the number of tiles and is limited by bandwidth.
Full Chip Performance

- Measured in millions of rays/second
- Increases with # tiles
- Limited by bandwidth

Graph showing performance increases with the number of tiles, measured in millions of rays/second, and limited by bandwidth.
Full Chip Performance

- Measured in millions of rays/second
- Increases with # tiles
- Limited by bandwidth

Graph:
- X-axis: #Tiles
- Y-axis: Million Rays/Seconds
- Lines represent:
  - Ideal
  - 1 DIMM
  - 2 DIMMs
  - 3 DIMMs
  - 4 DIMMs

The performance increases with the number of tiles and is limited by bandwidth.
Full Chip Performance

Measured in millions of rays/second

Increases with # tiles

Limited by bandwidth

- Ideal
- 1 DIMM
- 2 DIMMs
- 3 DIMMs
- 4 DIMMs

Graph showing performance measured in millions of rays/second, increases with the number of tiles, and is limited by bandwidth.
So, are we there yet?
Almost!

- **Quantitative goal**
  - 100 Million rays per second

- **Achieved**
  - ~50 Million rays per second
  - With 16 tiles and 4 DIMMs
  - Fits on 240mm$^2$ chip at 22nm
Almost!

- **Quantitative goal**
  - 100 Million rays per second

- **Achieved**
  - ~50 Million rays per second
  - With 16 tiles and 4 DIMMs
  - Fits on $240\text{mm}^2$ chip at 22nm

- **Insights**
  - Core can be 100% utilized
  - General purpose architecture can support ray-tracing
  - Good parallel scalability
Almost!

- **Quantitative goal**
  - 100 Million rays per second

- **Achieved**
  - ~50 Million rays per second
  - With 16 tiles and 4 DIMMs
  - Fits on 240mm$^2$ chip at 22nm

- **Insights**
  - Core can be 100% utilized
  - General purpose architecture can support ray-tracing
  - Good parallel scalability

- **Potential for optimizations**
  - Memory system, shared accelerator, Wide SIMD bundles
Summary

- *Ptolemic* GPU universe insufficient
- Paradigm shift to a *Copernican* universe

- The Copernicus full system graphics stack
  - Unique design: redundancy vs. synchronization
  - Leverage programmable multi-cores

- Evaluation methodology
  - Co-design of new applications and new architecture
  - Analytic model provides full system performance

- Real time rendering of high quality dynamic scenes
Questions

Some images courtesy:

- [http://www.relativitycalculator.com/images/models_universe/Ptolemaeus.jpg](http://www.relativitycalculator.com/images/models_universe/Ptolemaeus.jpg)
- [http://www.departments.bucknell.edu/History/Carnegie/newton/copernican.html](http://www.departments.bucknell.edu/History/Carnegie/newton/copernican.html)
- Pixar