

# coherent ray tracing via stream filtering

---



christiaan gribble  
karthik ramani

ieee/eurographics symposium on  
interactive ray tracing

august 2008

# acknowledgements

---

- early implementation
  - andrew kensler (utah)
  - ingo wald (intel)
  - solomon boulos (stanford)
- other contributors
  - steve parker & pete shirley (nvidia)
  - al davis & erik brunvand (utah)

# wide SIMD environments

---

- ray packets → SIMD processing
- increasing SIMD widths
  - current GPUs
  - intel's larrabee
  - future processors

**how to exploit wide SIMD units for  
fast ray tracing?**

# stream filtering

---

- recast ray tracing algorithm
  - series of filter operations
  - applied to arbitrarily-sized groups of rays
- apply filters throughout rendering
  - eliminate inactive rays
  - improve SIMD efficiency
  - achieve interactive performance

# core concepts

---

- ray streams
  - groups of rays
  - arbitrary size
  - arbitrary order
- stream filters
  - set of conditional statements
  - executed across stream elements
  - extract only rays with certain properties

# core concepts

---

input stream



← conditional statement(s)

↑ stream element

```
out_stream filter<test>(in_stream)
{
    foreach e in in_stream
        if (test(e) == true)
            out_stream.push(e)
    return out_stream
}
```

# SIMD filtering

---

- process stream in groups of  $N$  elements
- two steps
  - $N$ -wide groups  $\rightarrow$  boolean mask
  - boolean mask  $\rightarrow$  partitioned stream

# SIMD filtering

---

input stream



boolean mask



step one

# SIMD filtering

---

input stream



boolean mask

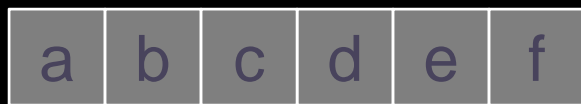


step one

# SIMD filtering

---

input stream



boolean mask



step one

# SIMD filtering

---

input stream

a	b	c	d	e	f
---	---	---	---	---	---

test

boolean mask

t	t	f	t	f	t
---	---	---	---	---	---

# SIMD filtering

---

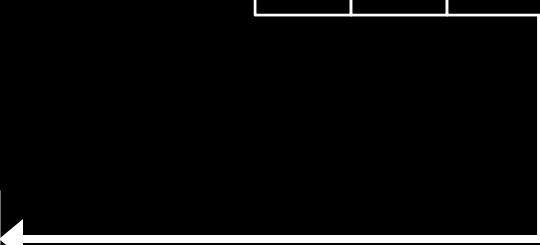
input stream



boolean mask



output stream



# hardware requirements

---

- wide SIMD ops ( $N > 4$ )
- scatter/gather memory ops
- partition op

# key characteristics

---

- all rays requiring same sequence of ops will always perform those ops together
- coherence defined by ensuing ops

# key characteristics

---

- all rays requiring same sequence of ops will always perform those ops together

**independent of execution path**

**independent of order within stream**

- coherence defined by ensuing ops

# key characteristics

---

- all rays requiring same sequence of ops will always perform those ops together

**independent of execution path**

**independent of order within stream**

- coherence defined by ensuing ops

**no guessing with heuristics**

**adapts to geometry, etc.**

# application to ray tracing

- recast ray tracing algorithm as a sequence of filter operations
- possible to use filters in all three major stages of ray tracing
  - traversal
  - intersection
  - shading

# filter stacks

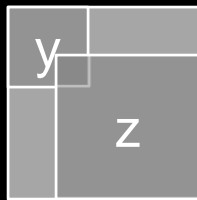
---

- sequence of stream filters
  - extract certain rays for processing
  - ignore others, process later
  - implicit or explicit
- traversal → implicit filter stack
- shading → explicit filter stack

# traversal

---

input stream



current node  $\rightarrow$  x

stack



output stream

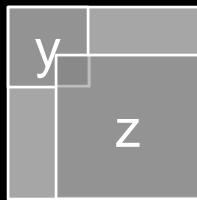


filter ~~top~~ ~~active~~ ~~range~~

# traversal

---

input stream



current node  $\rightarrow$  x

stack

y	(0, 3)
w	(0, 5)
...	

output stream

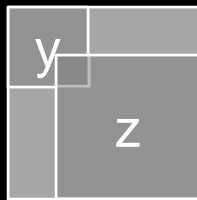


push back child

# traversal

---

input stream



current node  $\rightarrow$  x

output stream



stack

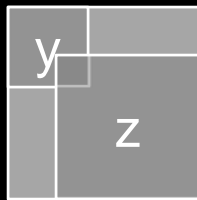
z	(0, 3)
y	(0, 3)
w	(0, 5)
...	

push front child

# traversal

---

input stream



current node  $\rightarrow$  x

output stream



stack

z	(0, 3)
y	(0, 3)
w	(0, 5)
...	

continue to next traversal step

# intersection

---

- explicit filter stacks
  - decompose test into sequence of filters
    - sequence of barycentric coordinate tests
    - ...
  - too little coherence to necessitate additional filter ops
- simply apply test in  $N$ -wide SIMD fashion

# shading

---

- explicit filter stacks
  - extract & process elements
    - shadow rays for explicit direct lighting
    - rays that miss geometry
    - rays whose children sample direct illumination
    - ...
  - streams are quite long
  - filter stacks are used to good effect
- shading achieves highest utilization

# algorithm – summary

- general & flexible
- supports parallel execution
  - process only active elements
  - yields highest possible efficiency
  - adapts to geometry, etc.
- incurs low overhead

# hardware simulation

---

- why a custom core?
  - skeptical that algorithm could perform interactively
  - provides upper bound on expected performance
  - explore parameter space more easily
- if successful, implement for available architectures

# simulator highlights

---

- cycle-accurate
  - models stalls & data dependencies
  - models contention for components
- conservative
  - could be synthesized at 1 GHz @ 135 nm
  - we assume 500 MHz @ 90 nm
- additional details available in companion papers

# key questions

---

- does sufficient coherence exist to use wide SIMD units efficiently?
- is interactive performance achievable with a custom core?

# key questions

---

- does sufficient coherence exist to use wide SIMD units efficiently?

## **focus on SIMD utilization**

- is interactive performance achievable with a custom core?

# key questions

---

- does sufficient coherence exist to use wide SIMD units efficiently?

**focus on SIMD utilization**

- is interactive performance achievable with a custom core?

**initial exploration of design space**

# rendering

---

- monte carlo path tracing
  - explicit direct lighting
  - glossy, dielectric, & lambertian materials
  - depth-of-field effects
- tile-based, breadth-first rendering

# experimental setup

---

- 1024x1024 images
- stream size → 1K or 4K rays
  - 1 spp → 32x32 or 64x64 pixels/tile
  - 64 spp → 4x4 or 8x8 pixels/tile
- per-frame stats
  - O(100s millions) rays/frame
  - O(100s millions) traversal ops
  - O(10s millions) intersection ops

# test scenes

---



*rtrt*



*conf*

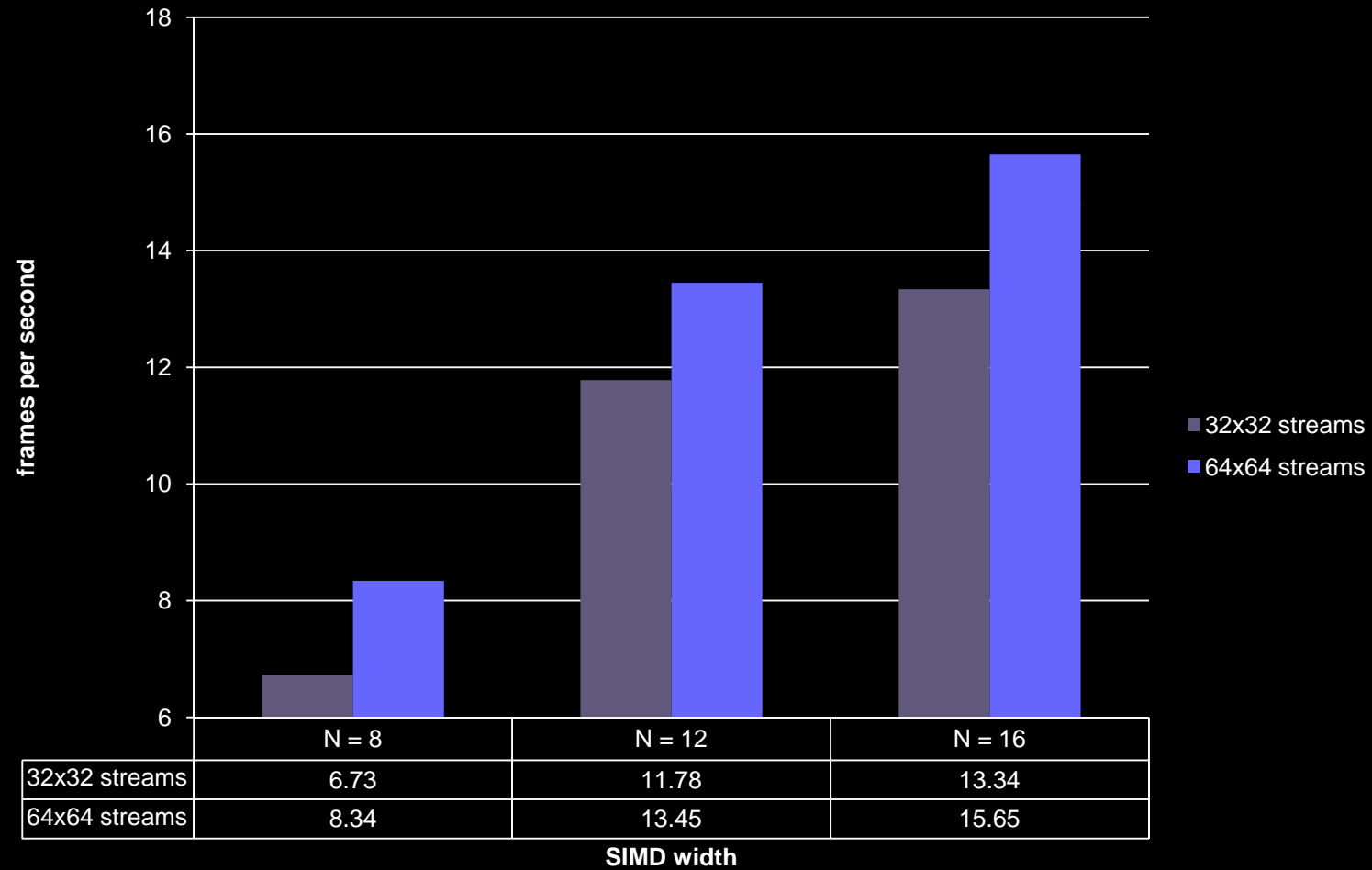


*kala*

- high geometric & illumination complexity
- representative of common scenarios

# predicted performance

*kala* – frame rate



# results – summary

---

- achieve high utilization
  - as high as 97%
  - SIMD widths of up to 16 elements
  - utilization increases with stream size
- achieve interactive performance
  - 15-25 fps
  - performance increases with stream size
  - currently requires custom core

# limitations – parallelism

---

- too few common ops → no improvement in utilization
- possible remedies
  - longer ray streams
  - parallel traversal

# limitations – hw support

- conventional cpus
  - narrow SIMD (4-wide SSE & altivec)
  - limited support for scatter/gather ops
  - partition op → software implementation
- possible remedies
  - custom core
  - current GPUs
  - time

# conclusions

---

- new approach to coherent ray tracing
  - process arbitrarily-sized groups of rays in SIMD fashion with high utilization
  - eliminates inactive elements, process only active rays
- stream filtering provides
  - sufficient coherence for wider-than-four SIMD processing
  - interactive performance with custom core

# future work

---

- additional hw simulation
  - parameter tuning
  - homogeneous multicore
  - heterogeneous multicore
  - ...
- improved GPU-based implementation
- implementations for future processors

# (more) acknowledgements

- temple of kalabsha
  - veronica sundstedt
  - patrick ledda
  - other members of the university of bristol computer graphics group
- financial support
  - swezey scientific instrumentation fund
  - utah graduate research fellowship
  - nsf grants 0541009 & 0430063

