Local Shading Coherence Extraction for SIMD-Efficient Path Tracing on CPUs

Attila Áfra, Carsten Benthin, Ingo Wald, Jacob Munkberg

Intel Corporation
Path tracing

- Standard method for production rendering

- Main steps:
  - Ray traversal
  - Shading
    - Usually more than half of the rendering time
Path tracing

- Single-ray tracing
  - SIMD single-ray traversal
  - Scalar shading

- Packet tracing
  - SIMD single-ray traversal (or packet traversal)
  - SoA-based SIMD packet shading
Path tracing

- Random rays $\to$ incoherent traversal and shading 😞
  - Low utilization of vector units
    - The wide (8-32) vector units of modern CPUs and GPUs are wasted
  - Incoherent memory accesses
Incoherent shading

- SIMD divergence
  - Many different shaders are evaluated within a SIMD batch
  - Low SIMD utilization

- Incoherent texture access
  - Non-cached reads form memory, disk, or network
Coherence extraction

- Performance can be improved by extracting coherence
  - Find batches of similar rays and process them together

- Most previous research focused on traversal

- Stream shading
  - Trace streams of rays and sort them on various criteria (e.g., material)

- Previous methods operate on a single large, global stream (millions of rays):
  - Wavefront path tracing on GPUs [Laine et al. 2013]
  - Sorted deferred shading for production path tracing [Eisenacher et al. 2013]
Our algorithm

- Traces and sorts small local streams independently on each CPU thread
  - 2K-8K rays per stream
- Enables efficient SIMD shading with low overhead

Why local?
- Has much lower overhead than global!
- Cache-friendly: the streams fit into the CPU’s last-level cache (LLC)
- Avoids expensive cross-core communication
- Very fast (and simple) ray sorting
- Sufficient for high (> 90%) SIMD utilization
Path tracing integrator

- Unidirectional path tracer with next event estimation
- Cast a ray from the camera
- Evaluate the material at the hit point
  - Material ID
  - Material shader which constructs a BSDF
- Cast a shadow ray toward a light source
- Cast an extension ray and repeat
Stream tracing

- Two ray streams:
  - Extension ray stream
  - Shadow ray stream
- SoA memory layout
  - SIMD-friendly
- Compact
  - No gaps (inactive rays)

- Algorithm consists of stages
  - Each stage involves a stream iteration
Stream tracing

- Ray generation
  - Generate primary rays from an image tile
    - e.g., 16x16 pixels, 8 samples per pixel
Stream tracing

- Ray generation
  - Generate primary rays from an image tile
    - e.g., 16x16 pixels, 8 samples per pixel

- Ray intersection
  - Intersect all extension rays in the stream
  - Single-ray traversal
  - Stream traversal
    - DRST [Barringer & Akenine-Möller 2014]
    - ORST [Fuetterling et al. 2015]
Stream tracing

- **Sorting**
  - Sort ray IDs by material ID
  - Counting sort → fast!
Stream tracing

- **Sorting**
  - Sort ray IDs by material ID
  - Counting sort → fast!

- **Material evaluation**
  - Iterate over the sorted ray IDs
  - Execute shaders for **coherent SIMD batches**
  - Generate extension and shadow rays
  - Append to new streams using **pack-stores**
    - Filter out terminated paths
    - Double buffering
Stream tracing

- Shadow ray intersection
  - Test all shadow rays for occlusion
Stream tracing

- **Shadow ray intersection**
  - Test all shadow rays for occlusion

- **Accumulation**
  - For unoccluded shadow rays, add direct light
  - For terminated paths, accumulate to image
Stream tracing

- Shadow ray intersection
  - Test all shadow rays for occlusion

- Accumulation
  - For unoccluded shadow rays, add direct light
  - For terminated paths, accumulate to image

- Path regeneration (optional)
  - Append new primary rays to the stream
    - Replace terminated paths
SIMD stream shading example

Sorted ray ID array: 0 4 11 1 2 6 9 15 5 10 12 13 3 7 8 14

Input array: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Output array: 0 4 11

Double buffers
SIMD stream shading example

Sorted ray ID array: 0 4 11 1 2 6 9 15 5 10 12 13 3 7 8 14

Input array: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Output array: 0 4 11

Double buffers
SIMD stream shading example

Sorted ray ID array: 0 4 11 1 2 6 9 15 5 10 12 13 3 7 8 14

Input array: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Output array: 0 4 11

Double buffers
SIMD stream shading example

Sorted ray ID array:

Input array:

SIMD register:

Output array:

Double buffers
SIMD stream shading example

Sorted ray ID array: 0 4 11 1 2 6 9 15 5 10 12 13 3 7 8 14

Input array: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

SIMD register: 1 2 6 9

Output array: 0 4 11

Gather

Double buffers
SIMD stream shading example

Sorted ray ID array:

Input array:

SIMD register:

Gather

Shading

SIMD register:

Output array:

Double buffers
SIMD stream shading example

Sorted ray ID array:

Input array:

SIMD register:

Gather

Shading

Pack-store

Output array:

Double buffers
Results

Stream tracing (Our)
- Stream size: 2K rays (376 KB/thread)
- Single-ray tracing w/ scalar shading
- Packet tracing w/ SIMD shading

- Same SIMD single-ray traversal kernel
- 8-wide SIMD, AVX2 instruction set
- Hardware: dual-socket Xeon E5-2699 v3
  - 36 cores, 72 threads, 90 MB LLC (30% used for streams)
Test scenes

Art Deco / 111 materials

Mazda / 76 materials

Villa / 97 materials

Conference / 36 materials
complex procedural shaders

Dragon / 5 materials
simple shaders
Path tracing performance (Mray/s)

- **Art Deco**
- **Mazda**
- **Villa**
- **Conference**
- **Dragon**

- Single
- Packet
- Our
Path tracing performance (Mray/s)

- Art Deco
- Mazda
- Villa
- Conference
- Dragon

3× speedup
SIMD utilization for shading (%)
Rendering time breakdown

![Bar chart showing rendering time breakdown for different models and scenes.

- Art Deco
- Mazda
- Villa
- Conference
- Dragon

The chart illustrates the normalized time spent on different processes:

- Unaccounted
- Sorting
- Shading
- Traversal

Each bar represents a model or scene, with segments indicating the time spent on each process.]
SIMD utilization vs. stream size

- Stream size: 8, 32, 128, 512, 2048, 8192, 32768
- SIMD utilization (%): 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100

Graph showing SIMD utilization (%) versus stream size for different streams:
- Art Deco
- Mazda
- Villa
- Conference
- Dragon
Conclusion

- Achieves much higher SIMD utilization than single-ray and packet shading
- Reduces shading time by 2-3× for complex scenes with hundreds of shaders
- Could perform even better with production-quality shaders
- Scales well to hundreds of CPU cores, wider SIMD (16), and bigger caches

Future work:
- Additional sorting steps (e.g., textures)
- Bidirectional path tracing
Questions?
SIMD utilization vs. number of materials

- **Packet**
- **Stream 2K**
- **Stream 8K**
- **Stream 32K**