Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU

by Dietger van Antwerpen
Outline

- Introduction
- Path Tracing
- Bidirectional Path Tracing
- Metropolis Light Transport
- Results
- Demo
Parallel MC Rendering

- Monte Carlo rendering embarrassingly parallel
- Generate many samples in parallel
- Not so trivial for wide SIMD architectures
- Samples have **stochastic sample length**
- Uneven sample workload
- Incoherent execution flow
- Low SIMD efficiency
Random Walk

- PT and BDPT use random walks
- Walk is terminated using Russian roulette
- Stochastic path lengths
- ~33% active threads per GPU warp
- Upper bound on SIMD efficiency
Bidirectional Connections

- BDPT fully connects two random walks

- Number of connections is **quadratic** in average random walk length

- ~17% **active threads** per GPU warp

- Upper bound on SIMD efficiency
Contributions

- Improving average SIMD efficiency
  - Random walk phase:
    Combining stream compaction and sample regeneration
  - Bidirectional connect phase:
    Evaluating all connections from all samples in parallel

- Implement MLT on top of BDPT on the GPU
Outline

- Introduction
- **Path Tracing**
  - Bidirectional Path Tracing
  - Metropolis Light Transport
- Results
- Demo
In-Place Sample Regeneration

- Proposed by Novak et al.
- Regenerate after each extension
- **Restart all terminated samples in-place**
- **Advantage:**
  - Improves SIMD efficiency during sample extension and connection
- **Disadvantage:**
  - Low SIMD efficiency during regeneration
  - ~30% active threads per GPU warp
Stream Compaction + Regeneration

- **Remove terminated samples** from the stream using **stream compaction**
- Short stream length may reduce GPU utilization
- **Regenerate terminated samples** at the end of the sample stream
Stream Compaction + Regeneration

- Initialize sample stream

```
Generate stream

Sample stream
```
Stream Compaction + Regeneration

- Extend all samples with next path vertex
- Some samples terminate
- **Compact output stream**
Stream Compaction + Regeneration

- Output stream becomes next sample stream
- **Regenerate** new **samples** at the end
Advantages

- High SIMD efficiency during extension and connection
- High SIMD efficiency during regeneration
- Fixed size sample stream
- Regenerated samples lie side-by-side
- Primary rays benefit from primary ray coherence
- \(~20\%\) speedup over in-place sample regeneration
Outline

- Introduction
- Path Tracing
- **Bidirectional Path Tracing**
- Metropolis Light Transport
- Results
- Demo
Bidirectional Path Tracing

- Improve SIMD efficiency during random walk
  - Combine stream compaction and regeneration
- Improve SIMD efficiency during connection
  - Evaluate all bidirectional connections in parallel

- Algorithm is divided in random walk and connect phase
- Phases execute repeatedly one after the other
Random Walk Phase

- **Initialize** eye and light path stream

  Eye path stream

  1 2 3 4 5 6 7 8

  Light path stream

  1 2 3 4 5 6 7 8
Random Walk Phase

- **Extend** all paths with one vertex
- Some paths terminate

Eye path stream

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Light path stream

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
Random Walk Phase

- **Compact** path streams

Eye path stream:
1 2 3 4 5 6 7 8

Light path stream:
1 2 3 4 5 6 7 8
Random Walk Phase

- Repeat extend and compact
- Postpone regeneration

**Eye path stream**

1 2 3 4 5 6 7 **8**

1 2 3 4 5 6 7

2 4 6

**Light path stream**

1 2 3 4 5 6 7 8

2 3 4 5 6 8

4 5 6
Random Walk Phase

- Sample terminates when both eye and light path have terminated
Random Walk Phase

- Repeat until 60% of samples terminated
Bidirectional Connect Phase

- Evaluate connections for **terminated** samples
- Generate stream of bidirectional connections

**Eye path stream**

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7

2 4 6

**Light path stream**

1 2 3 4 5 6 7 8

2 3 4 5 6 8

4 5 6

**Bidirectional connection stream**

1 1 3 3 3 3 6 6 6 6 6 6 6 6 6 7 7 8 8
Sample Regeneration

- **Regenerate** terminated samples and resume random walk phase

<table>
<thead>
<tr>
<th>Eye path stream</th>
<th>Light path stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7 8</td>
<td>1 2 3 4 5 6 7 8</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>2 3 4 5 6 8</td>
</tr>
<tr>
<td>2 4 1 3 6 7 8</td>
<td>4 5 1 3 6 7 8</td>
</tr>
</tbody>
</table>
Sample Regeneration

- Sample regeneration keeps path streams long
- Good for GPU utilization
- Total speedup $\sim 15\%$
- Less than for path tracing
- Sample regeneration only improves random walk phase
- BDPT spends only $\sim 55\%$ in random walk phase
Bidirectional Connect Phase

- Evaluate all connection **in parallel**
- Each terminated sample contributes #connections
- Execute thread for each connection
- Threads **figure out** which connection to evaluate using
  - Parallel scan over all samples
  - Binary search for each connection thread
- See paper for details...
Outline

- Introduction
- Path Tracing
- Bidirectional Path Tracing
- Metropolis Light Transport
- Results
- Demo
Metropolis Light Transport

- Run many independent MLT samplers in parallel
- Based on the BDPT implementation
- Use variation on Kelemen mutation
- Only mutate sample dimensions used in both current and mutated sample
- Estimate normalization constant on the fly
Startup Bias

- Each MLT sampler introduces startup bias
- Many parallel samplers means **lots of bias**
- Bias is usually larger for difficult scenes

10 seconds  Reference
Startup Bias

- Each MLT sampler introduces startup bias
- Many parallel samplers means **lots of bias**
- Bias is usually larger for difficult scenes
Startup Bias

- Each MLT sampler introduces startup bias
- Many parallel samplers means **lots of bias**
- Bias is usually larger for difficult scenes

10 minutes  Reference
Outline

- Introduction
- Path Tracing
- Bidirectional Path Tracing
- Metropolis Light Transport
- Results
- Demo
SIMD efficiency

- Algorithms always work on **continuous streams**
- Active threads per GPU warp ~**99%**
- **Upper bound** on actual SIMD efficiency
- Actual SIMD efficiency lower due to divergent shader/traversal code
- Performance improvement less than SIMD efficiency improvement...
Compared with straightforward multicore CPU implementation

- NVIDIA GTX 480 GPU
- Intel Core i7 920 CPU
- Speedup between 8x and 15x
- GPU can do more than path tracing!
Demo
Questions?
Extra
Immediate Stream Compaction

- Stream compaction requires **multi-pass** parallel scan and scatter pass
- Immediate stream compaction in **single pass**
- **Parallel scan** per block in shard memory
- Block **allocates** space in output buffer using one **atomic add**
- Threads write items **directly** into compacted output stream
Immediate Stream Compaction

- **Label** all active threads

 Thread Block

```
1 1 0 1 0 0 1 1 1 1 0
```

Output stream

...
Immediate Stream Compaction

- **Parallel scan** per block in shared memory

```
Thread Block
1 2 2 3 3 3 4 5 6 6
```

```
Output stream
...
```

TU Delft
Immediate Stream Compaction

- Block *allocates* memory in output stream using an *atomic instruction*

[Diagram showing a thread block and an output stream]
Immediate Stream Compaction

- Each active thread **writes** directly in the **output stream**

```
Thread Block
1 2 2 3 3 3 4 5 6 6
```

```
... 1 2 3 4 5 6
```
Parallel Bidirectional Connect

- Each sample **writes #connections** in connection count buffer
- Non-terminated samples write a zero
Parallel Bidirectional Connect

- **Parallel scan** the connection count buffer
- Gives the #connections preceding each sample in the buffer
Parallel Bidirectional Connect

- Start one GPU thread for each connection
Parallel Bidirectional Connect

- **Binary search** thread index for corresponding sample in connection count buffer
Parallel Bidirectional Connect

- Compute **local connection index** from sample connection count
Parallel Bidirectional Connect

- Local connection indices map to an eye-light vertex pair to connection
- Each thread evaluates its connection
Coalesced Vertex Scattering

- Path vertices are stored during random walk
- Vertices are scattered to pre-allocated vertex memory
- Each thread scattering its vertex would result in uncoalesced memory access
- **Threads** in a warp **collaborate** to **efficiently scatter** path **vertices** to memory
- Vertices are scattered through shared memory
- Similar to matrix transpose
Coalesced Vertex Scatter

- Vertex is 128 bytes
- Each thread in warp writes vertex to shared memory
Coalesced Vertex Scatter

- Each thread in warp reads one word from each vertex in shared memory buffer
Coalesced Vertex Scatter

- Each thread scatters one word of each vertex
- Coalesced scatter
Coalesced Vertex Scatter

- Each thread scatters one word of each vertex
- Coalesced scatter
Coalesced Vertex Scatter

- Each thread scatters one word of each vertex
- Coalesced scatter

![Diagram showing coalesced vertex scatter](https://example.com/diagram.png)
Mutation Strategy

- Kelemen et al. proposed to lazily perturb all infinite dimensions
- Leads to uneven workload during mutation
- Instead, perturb only dimensions used in both the current and mutated sample
- Regenerate other dimensions
- Keeps the strategy symmetric
- Reduces memory usage
- Even workload per path vertex