Wide BVH traversal with a short stack

Karthik Vaidyanathan, Sven Woop, Carsten Benthin
Motivation

Ray tracing hardware acceleration easily constrained by memory bandwidth

- Compressed wide BVHs can reduce node bandwidth
  [Wald et al. 14, Ylitie et al. 17]
- Lossless meshlet compression can reduce geometry bandwidth
- Full stack can add additional bandwidth when stored to memory when procedural geometry is intersected

➔ Need stack algorithm for wide BVHs with compact state
Previous Work

Stackless traversal with skip pointers [Smits 98]
- Enforces a fixed traversal order

Stackless traversal with backtracking to parent [Hapala et al. 11, Afra et al. 14]
- Allows front to back order
- Needs to re-intersect parent node

Binary BVH traversal with a restart trail [Laine 10]
- Maintain a restart trail and short stack of topmost entries
- Restart from root on an short stack underflow
BVH Traversal with a Restart Trail

A restart trail with short stack is well suited for fixed function ray tracing

- Short stack entries and the restart trail small
- Low overhead (10%) as short stack avoids most redundant traversal steps

Our Wide-BVHs restart trail algorithm builds on binary BVH traversal of [Laine 10]
Binary BVH Traversal with a Restart Trail

Assuming fixed front to back ordering of child nodes along the ray

Restart trail is bit vector (and depth) that encodes processed part of BVH

- RestartTrail[level] == 0 indicates that the first child is being traversed
- RestartTrail[level] == 1 indicates that the last child at the given level is being traversed

➔ Pop operations can easily skip over finished levels
Binary Restart Trail

LEVEL = 3

Restart Trail

1
0
0
0

Increment

Find closest level < 1

LEVEL = 1

clear
0
0
0
N-Wide BVH Traversal with a Restart Trail

Assuming fixed front to back ordering of child nodes along the ray

Restart trail is an **integer vector** (and depth) that encodes processed part of BVH

- RestartTrail[level] indicates the n’th child currently traversed
- RestartTrail[level] set to N-1 indicates the last child subtree being traversed

⇒ Pop operations can easily skip over finished levels
Detecting the Last Child

During down traversal the last child at a given level is detected when RestartTrail[level] == (number of intersections – 1)

But we do not know if an entry popped from the short stack is the last child

• Therefore we use **one additional bit** per short stack entry to mark the last child when it is pushed onto the stack.

• Not required for binary BVH as popped nodes are always the last node.
Example 4-Wide BVH Traversal

LEVEL = 0

Restart Trail
0
0
0
0

Short Stack (3 Entries)
Example 4-Wide BVH Traversal

LEVEL = 0

Restart Trail

Short Stack (3 Entries )

Last Child
Example 4-Wide BVH Traversal

Restart Trail

LEVEL = 1

Short Stack
(3 Entries)

B
C
Example 4-Wide BVH Traversal

LEVEL = 1

Last Child

C dropped
Example 4-Wide BVH Traversal

Find closest level < 3

LEVEL = 2

Restart Trail

Increment

E Popped

Short Stack (3 Entries)

E
F
B
Example 4-Wide BVH Traversal

Find closest level < 3

Set to 3 last child

LEVEL = 2

Restart Trail

F Popped

Short Stack
(3 Entries )

B

F
Example 4-Wide BVH Traversal

Restart Trail

0

3

3

0

0

LEVEL = 3

Set to 3 last child

Short Stack (3 Entries )

B
Example 4-Wide BVH Traversal

LEVEL = 3

Restart Trail

Increment

Find closest level < 3

Short Stack (3 Entries)

A
B
C
D
E
F
G
Example 4-Wide BVH Traversal

Find closest level < 3

LEVEL = 1

Restart Trail

Increment

Short Stack
(3 Entries)

B Popped
Example 4-Wide BVH Traversal

Stack empty: restart

LEVEL = 0

Restart Trail

Short Stack (3 Entries)
Example 4-Wide BVH Traversal

LEVEL = 0

Restart Trail

Short Stack
(3 Entries)
Example 4-Wide BVH Traversal

Restart Trail

<table>
<thead>
<tr>
<th>3</th>
<th>Set to 3 last child</th>
</tr>
</thead>
</table>

Short Stack

(3 Entries )

LEVEL = 1

D E F

G
Example 4-Wide BVH Traversal

Restart Trail

Find closest level < 3

LEVEL = 1

Short Stack (3 Entries)
Results for 6-wide BVH
Stack culling

- Storing near distance on the short stack would allow culling of popped nodes when a closer hit was found
  ➔ memory overhead

- Alternatively one can process closest child and push the parent node onto the short stack for later re-intersection
  ➔ small culling overhead but often pays off
Conclusions

• Presented short stack with restart trail for wide BVHs
• Allows very compact stack storage at minimal overhead
• Suitable for fixed function ray tracing implementations