EFFICIENT DATA-PARALLEL TREE-TRAVERSAL FOR SOLID MODELING

Herbert Grasberger
University of Victoria

Jean-Luc Duprat
Intel

Brian Wyvill
University of Victoria

Paul Lalonde
Intel
MOTIVATION

• Evaluate BlobTrees on the GPU
• Use OpenCL to traverse tree data-structure
• Evaluate thousands of field-values in parallel
OVERVIEW

• Problem statement & Introduction
• Tree traversal
• Implementation
• Test cases
• Results

Asia 2012
History: The Blobtree: skeletal primitives

\[ f(p) = g(d_L(p)) \]

\[ g: \mathbb{R} \rightarrow \mathbb{R} \]
The Blobtree: skeletal primitives
Resulting shapes
SOLID MODELING
BLOB-TREE TRAVERSAL

Traversing The BlobTree

‘N’ - indicates a node in the BlobTree
L (‘N’) - left child R (‘N’) - right child
function F returns the field value for the node ‘N’ at the point M

function F(‘N’, M)
1. Primitive: F( M)
2. Warp: F( L (‘N’) , w( M)) (warp is a unary operator)
3. Blend: F( L (‘N’) , M)+ F( R (‘N’) , M))
4. Union: max( F( L (‘N’) , M), F( R (‘N’) , M))
5. Intersection: min( F( L (‘N’) , M), F( R (‘N’) , M))
6. Difference: min( F( L (‘N’) , M), -F( R (‘N’) , M))

end
NATURAL PHENOMENAE
Sketch Based Interactive Modelling

Shape Shop - Ryan Schmidt

Loic Barthe
Marie-Paule Cani

Using field value and gradient.
SKETCH BASED PROTOTYPING
ENGINEERING MODELS

Student: Herbert Grasberger
Two very similar algorithms published independently

1986  (Implicit Surfaces)
*Data Structure for Soft Objects*
Geoff Wyvill, Craig McPheeters, Brian Wyvill
Hash table, efficient memory use, many optimizations,
took care of ambiguous cases, animation support etc.
case statement

1987  (Volume Data)
*Marching Cubes*
Lorenson and Cline
ACM SIGGRAPH 1987
Stored every cube, missed ambiguous cases, no animation support.
table . Hybrid of these for parallel implementation!
Polygonization of Implicit Surfaces on Multi-Core Architectures with SIMD

Pourya Shirazian, Brian Wyvill, Jean-Luc Duprat

1.pouryash@cs.uvic.ca
2.blob@cs.uvic.ca
3.jld@acm.org

EPGV 2012
THE PROBLEM

• No recursion in OpenCL

• Iterative traversal with stack to store immediate values

• Can we do better (faster)?

• Reduce storage size of model information
SPMD PROGRAMMING

- Run multiple tasks in parallel
- all tasks work on the same tree data (traversed for thousands of points).
- different results through different input variables
- Ray-tracing: different input rays -> different intersections
- BlobTree: different input positions -> different field-values

thousands of points traversing same tree – optimize type of result different but traversing same tree – rays and for fields
Memory access important for performance

- Different threads should access different input (memory access in parallel with same offset). Neighbouring threads access neighbouring nodes - memory blocks.

- Different threads should access the same memory location (multicast memory access) (same node of the tree).

- Random memory access is expensive -> avoid as much as possible.

You want neighbouring threads to access neighbouring memory blocks in parallel or all threads read the same memory block (node) of the tree at the same time (fetched once).

Avoid each thread accessing different blocks.
TREE EVALUATION

• Solid model trees need to be evaluated by visiting every tree node
  • acceleration approaches discard subtrees
  • in worst case, all nodes need to be visited anyway
• Information needed on how to access child (/parent/ neighbour) nodes for traversal (how to go down or up)
top down approach traditional – repeatedly visiting nodes back and forth in memory
some access not to next memory (unpredictable)
we store tree nodes – start at leaves postorder – memory in order predictable
2 results build next
<table>
<thead>
<tr>
<th></th>
<th>Top-down</th>
<th>Bottom-Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree Structure</td>
<td>store tree structure and node information</td>
<td>only node information required</td>
</tr>
<tr>
<td></td>
<td>memory reads are done non-linearly</td>
<td>only linear memory reads</td>
</tr>
<tr>
<td></td>
<td>stack or additional information needed to traverse tree</td>
<td>no stack or additional information needed for traversal</td>
</tr>
</tbody>
</table>
IMPLEMENTATION

- OpenCL on ATI Radeon HD5870 under MacOSX 10.7.3
- for (immediate) results (field-value and CSG intersect) we store:
  - 1 float: field-value or intersection t
  - 2 float: gradient x,y + sign of z in 1 byte
  - 3 bytes: for RGB color
- Local GPU memory too small for many problem instances

2 normal calculated so just need sign of saves memory global memory used
TEST CASES

• Field-value calculations for:
  • Chain of cylinders with expensive combination operator
    • variable length, increasing space needs, several tree structures for same model
  • Models of different size
• Ray-tracing on recursively generated CSG models

balanced left and right centric trees tested
RESULTS

• The bigger the model, the better the speed-up

• One order of magnitude difference for large models

• Unoptimized bottom-up faster than BVH accelerated top-down (even though 2/3 tree discarded in some test cases).

• kD - tree can be used to accelerate bottom-up

• Same performance characteristics for CSG ray-intersection

bottom up better than optimized top down which is discarding perhaps 2/3 of tree in test cases. BVH needs top down but can use KD tree with bottom up.
1. top down traversal and TDT + BVH
2. bottom up + BU+Kd tree
at 256 nodes balanced top down becomes faster than unaccelerated BU
order of magnitude difference between accelerated approaches. No
difference between best and worst case with accelerated BU.
BVH discards stuff in tree requires top down
KD store only nodes of interest
in engine (green) most objects are not access aligned and so KD is actually slower.
engine has 140 prim. robot ~ 100 donkey ~15 monkey ~30
no accel. structures (Menger 3 had too many nodes for GPU)
CONCLUSION

• Changed the way tree nodes are stored in memory and how memory is read to traverse the tree

• Less memory reads, less memory needed to traverse the tree

• Memory reads are linear and done in one direction only

• One order of magnitude performance gain for very large models
FUTURE WORK

• Investigate performance characteristics on CPUs
  • better branch prediction
  • SIMD instructions
• Find ways of reducing memory needs even more, to make it work on even bigger models

similar on a CPU in first tests
branch prediction : on newer GPUs next step
CPU is best branch prediction and it shows that will work on GPU
THANK YOU