Hardware-Driven Ray Tracing

Jae-Sung Yoon

KAIST (Korea Advanced Institute of Science and Technology)
Contents

● 1. Introduction
● 2. Acceleration Structure
● 3. Techniques for Hardware
● 4. Previous Hardware Implementations
● 5. Hardware Issues and Problems
● 6. Future Work
Introduction

- One of the most popular and simple way of doing global illumination.

[Porcell 2004]
Why Ray Tracing Hardware?

- CPU can accelerate only single-threaded application
- CPU Cannot optimally exploit the inherent parallelism in ray tracing.
Basic Algorithm

For BSP structure, Traversal is typically 2-3 times as costly as intersection test  [Wald 2001]
Acceleration Structure (AS)

Uniform Grid
[Martin 2005]

KD-tree

BVH
(bounding volume hierarchy)
Spatial Subdivision Schemes

- Uniform Grid, KD-tree
- Commonalities
  - 1. Multiple Reference
    - triangles can straddles a voxel boundary
  - 2. Can stop traversal at first intersection
    - visit the voxels in order of increasing distance from the ray origin
  - 3. Stack
1. Uniform Grid (Construction)

- 1. Decide the resolution of the grid
  - High resolution: fewer triangle intersection
  - Low resolution: fewer voxel traversal
  - Triangle/voxel = 20 works well [Woo]
  - $3^{\frac{3}{N}}$ voxels along the longest axis [Pharr]
- 2. Add a reference to the triangle in each voxel
1. Uniform Grid (Traversal)

- 3D DDA (digital differential analyzer) algorithm [Amanatides & Woo]

\[
\begin{align*}
\text{if } t_{max_x} < t_{max_y} \text{ then} & \\
X & \leftarrow X + step_x \\
t_{max_x} & \leftarrow t_{max_x} + \delta_{max_x} \\
\text{else} & \\
Y & \leftarrow Y + step_y \\
t_{max_y} & \leftarrow t_{max_y} + \delta_{max_y}
\end{align*}
\]
2. **KD-tree** *(Construction)*

- Top-down in a recursive fashion
- Choose an axis-perpendicular splitting plane
- If a primitive straddles the split plane then both child get a reference
2. KD-tree (Construction)

- Where to place the splitting plane?
- Surface area heuristic [MacDonald & Booth 1990]

[MacDonald 1990]

[Fig. 1. Surface area heuristic data]

[Wald 04]
2. KD-tree (Construction)

\[ C_t = \sum_{i=1}^{N_i} SA(i) + C_t \cdot \sum_{l=1}^{N_l} SA(l) + C_o \cdot \sum_{l=1}^{N_l} SA(l) \cdot N(l) \]

\[ \frac{SA(root)}{SA(root)} \]

- \( C_t \) = cost of traversing an interior node
- \( C_l \) = cost of traversing a leaf
- \( C_o \) = cost of testing an object for intersection

- \( N_i \) = no. of interior nodes
- \( N_l \) = no. of leaves
- \( N(l) \) = no. of objects stored in leaf \( l \)
- \( SA(i) \) = surface area of interior node \( i \)
- \( SA(l) \) = surface area of leaf node \( l \)

**Diagram:**
- Straddling triangle
- Split axis
- Left child
- Right child
- Nodes a, b, c, d, e, f, g, h, i, j
2. KD-tree (Traversal)

- Given node N

- If N is leaf, intersection test for triangles
- If N is internal, determine which child node is first hit by ray
- Stack is needed (stack per ray)

But, GPU don’t have Stack
3. BVH (Construction)

- There is no known algorithm for constructing optimal BVH
- Cost function - surface area heuristic [Goldsmith & Salmon]
- AABB is most widely used.
- There are two major paper for BVH construction
  - [Kay & Kajiya] top-down
  - [Goldsmith & Salmon] bottom-up
3. BVH (Construction)

- Sphere
- Axis-aligned bounding box: Very cheap to compute
- Oriented Bounding Box
- K-dops
3. BVH (Construction) Kay/Kajiya

```
BVNODE BuildTree(triangles)

if we were passed just one triangle then
    return leaf holding the triangle
else
    Calculate best splitting axis and where to split it
    BVNODE result
    result.leftChild ← BuildTree(triangles left of split)
    result.rightChild ← BuildTree(triangles right of split)
    result.boundingBoxBox ← bounding box of all given triangles
    return result
end if
```
3. **BVH (Construction) Goldsmith/Salmon**

- Assign first triangle as the root
- For other objects, best position in the tree is found by cost function
3. BVH (Traversal)

- Unlike the kd-tree, all the children have to be visited
- Stack is needed

- Main issue: order in which a node’s children are traversed
Techniques for Acceleration & Hardware

- 1. Mailbox [Amanatides & Woo]
- 2. Stackless KD-tree traversal [Foley 2005]
- 3. Distributed Interactive Ray Tracing of Dynamic Scenes [Wald 2003]
1. Mailbox (spatial structure)

- **Problem**
  - If triangle is split into two child voxel, each voxel has reference to the original triangle.
  - We could test intersection against the same ray more than once.

- **Solution**
  - Store the ID of ray with the triangle to skip the test the second time.
2. Stackless KD-tree Traversal

- **KD-Restart**
  - If leaf intersection fail, update (tmin, tmax)
  - Then, restart the search at the root

- **KD-Backtrack**
  - Store the parent link in the nodes
  - If leaf intersection fail, update (tmin, tmax)
  - Then, move up the tree to find parent
3. Distributed Interactive Ray Tracing of Dynamic Scenes

- **Motivation**: In dynamic scene, large parts of a scene is static.

- **Objects can be separated into three classes**
  - Static objects
  - Objects undergoing affine transformation
    - \( x \rightarrow Ax + B \)
  - Objects with unstructured motion
For affine transformation,

- \((\text{Ray} - \text{transformed Object}) \text{ intersection} = = (\text{inverse transformed Ray} - \text{Object}) \text{ intersection}\)

Can remove the reconstruction cost for transforming objects

Only top-level structure has to be rebuilt
Hardware Implementations

- 1. Streaming Ray Tracing [Purcell 2002]
- 2. Static SaarCOR [Schmittler 2002]
- 2. Dynamic SaarCOR [Schmittler 2004]
  - Upgrade version of Static SaarCOR
- 3. RPU [Woop 2005]
  - Programmable version of SaarCOR
- 4. Ray Tracing in FPGA [Lee 2007]
1. Streaming Ray Tracing [Purcell 2002]

- Used a streaming processor
- All geometry is stored in texture memory before rendering
- Used a uniform grid acceleration structure
- Used static scene
Grid list contains a pointer to a list of triangles
Triangle list contains a pointers to vertex data
2. Dynamic SaarCOR [Schmittler 2004]

- Special purpose hardware for ray tracing
- Concerned about AS reconstruction using [Wald 2003] method
- Using cache [Schmittler 2003], load scene data from host
- Used a KD-tree
- Used unit triangle intersection method

![Diagram of coordinate spaces](image-url)
Architecture

Dynamic SaarCOR

Memory Interface (MI)

Display

Bus-Mem

Simple
Routing

Mem-Ctrl

RGS

Trav-Cache

List-Cache

Matrix-Cache

Transformation

Intersection

Dynamic ray tracing pipeline

Scene data & camera setting upload

Geometry & AS stored

Host upload: camera settings and control

Ray-Generation Controller (RGC)

DynRTP-1

DynRTP-2

DynRTP-n

(...)

Memory Chips

Host System Bus

KAIST
1. **RGS**: ray generation
2. **a**: ray & T(matrix) are sent
3. **Transformation**
4. **b & Traversal**: top-level traversal
5. **c**: ray & objects are sent
6. **Mailbox**
7. **d & Transformation**: ray transform into object space
8. **recursive traversal**
Operation Details

9. **Transformation**: ray transform to unit triangle space
10. **e & Intersection**
11. **f & recursive traversal**
12. **g**: final result sent
13. **RGS**: shading
3. RPU [Woop 2005]

- Programmable hardware for ray tracing
- Traversal needs dedicated unit
- Same algorithm with Dynamic SaarCOR
- Upgrade version of Dynamic SaarCOR

4D-vector dual-issue system
Architecture

Shader Processing Unit
Traversal Processing Unit
Mailbox List Processing Unit
Dedicated Traversal Unit

- K-D tree traversal is very expensive scalar floating point operation.
- So, programmable hardware is inefficient
- Stack is in the SPU register

SPU is quite similar with current GPU

Communicate with main memory because on-chip stack can be not sufficient
Intersection Test Program

Used unit triangle intersection method
Program quite fit into GPU

1 load4x A.y,0 ; load triangle
    ; transformation
2 dp3_rcp R7.z,I2,R3 ; transform ray dir to
    ; unit triangle space
3 dp3 R7.y,I1,R3
4 dp3 R7.x,I0,R3
5 dph3 R6.x,I0,R2 ; transform ray origin to
    ; unit triangle space
6 dph3 R6.y,I1,R2
7 dph3 R6.z,I2,R2
8 mul R8.z,-R6.z,S.z ; compute hit distance d
    ; and exit if negative
+ if z <0 return
9 mad R8.xy,R8.z,R7,R6 ; compute barycentric
    ; coordinates u and v
    ; and return if
+ if or xy
    ; hit is outside
    ; the bounding square
    ; compute u+v and test
+ if w >=1 return ; against triangle diagonal
10 add R8.w,R8.x,R8.y
    ; terminate if last hit
    ; distance in R4.z is
    ; closer than the new one
+ if w >=0 return
12 mov STD,I3.x ; set shader ID
    ; and update MAX value
+ mov MAX,R8.z
13 mov R4.xyz,R8 ; overwrite old hit data
    ; and return
4. Ray Tracing in FPGA [Lee 2007]

1. Dedicated Hardware
2. Used KD-backtrack