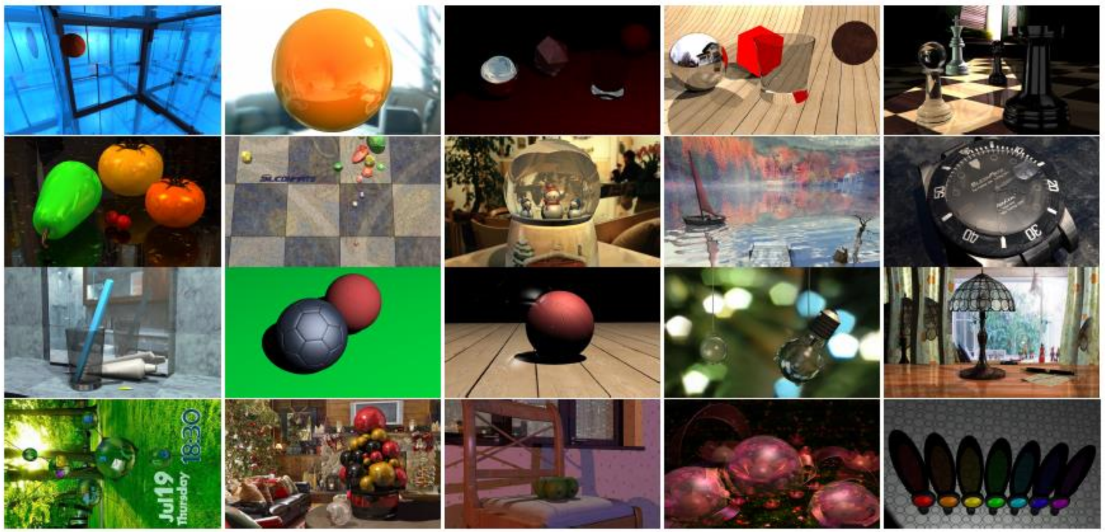
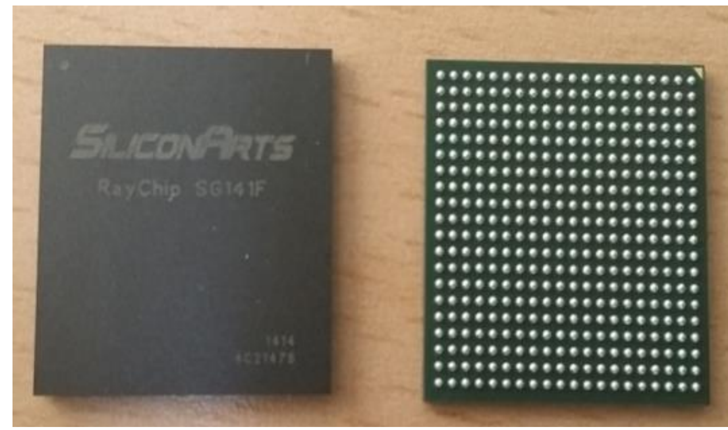
# RayCore: A ray-tracing hardware architecture for mobile devices



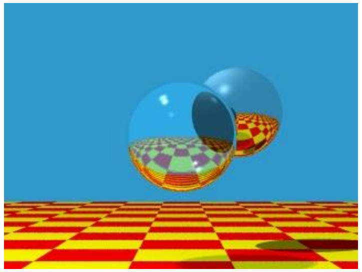



Jae-Ho Nah[1,3,4,5], Hyuck-Joo Kwon[1], Dong-Seok Kim[1],
Cheol-Ho Jeong[2], Jinhong Park[3],
Tack-Don Han[4], Dinesh Manocha[5], and Woo-Chan Park[1]

[1]Sejong University, [2]Siliconarts, [3]LG Electronics,
[4]Yonsei University, [5]University of North Carolina at Chapel Hill

# Motivation

- Ray tracing (RT) [Whitted 1980]
  - Classic rendering algorithm for photo-realistic rendering

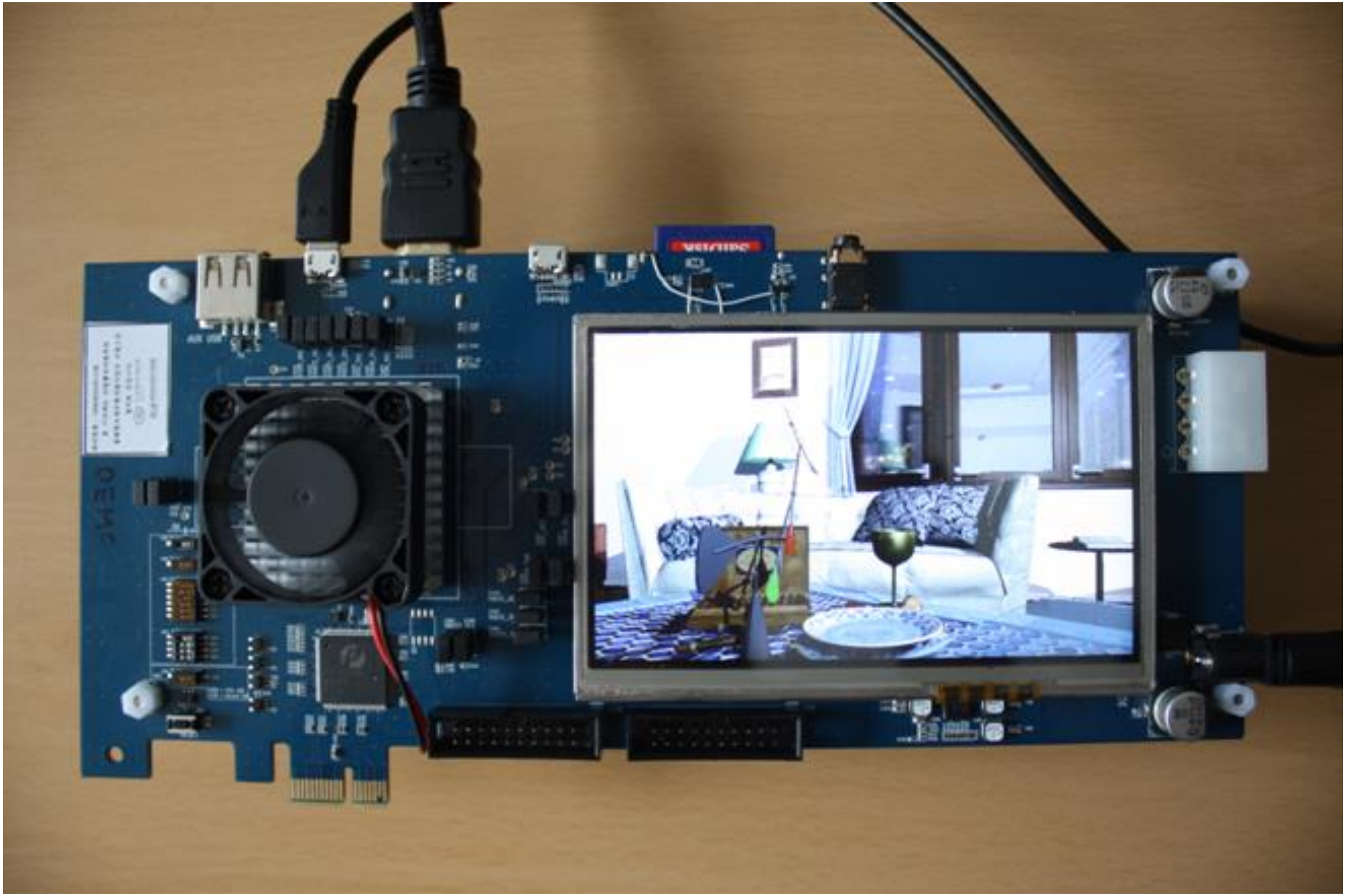


- Our goal
  - "**Real-time**" ray tracing dynamic scenes on "**mobile devices**" for triangulated models


- Why mobile ray tracing?
  - High interest in generating photo-realistic images at low power cost
  - **<u>Ray tracing H/W</u>** can be a solution for mobile graphics

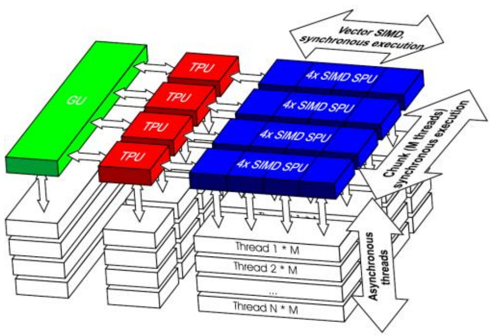
# Demo

Mobile Media Processor Lab.
Sejong Univ.

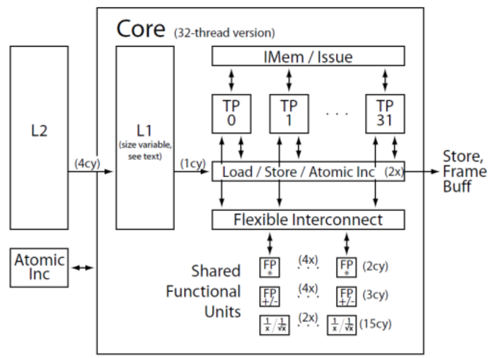


# Related Work: Dedicated RT H/W

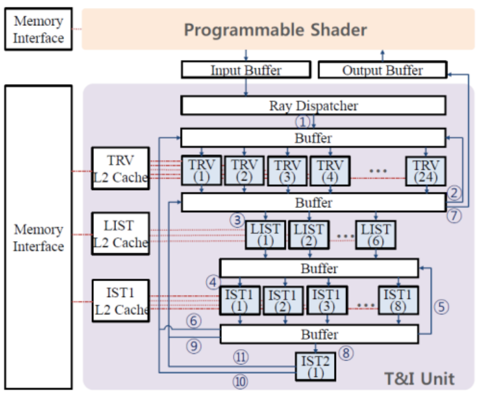
- RPU/D-RPU [Woop et al. 2005/2006]




- TRaX [Spjut et al. 2009]




- T&I Engine [Nah et al. 2011]




- Mainly focus on high-quality rendering on **desktop PCs**
  - A large chip area & high power consumption for high performance

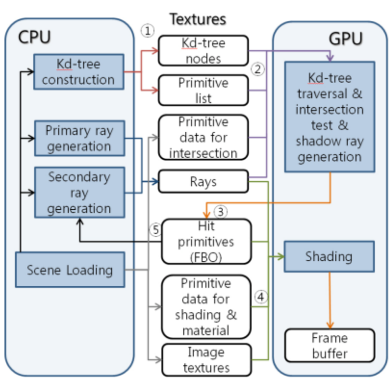
# Related Work: Mobile RT



- Mobile environment
  - Very limited area and power budget (~20 $mm^2$, ~2 W for a GPU)

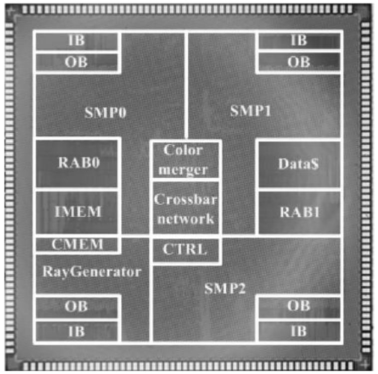
- S/W approach
  - MobiRT

  [Nah et al. 2010]



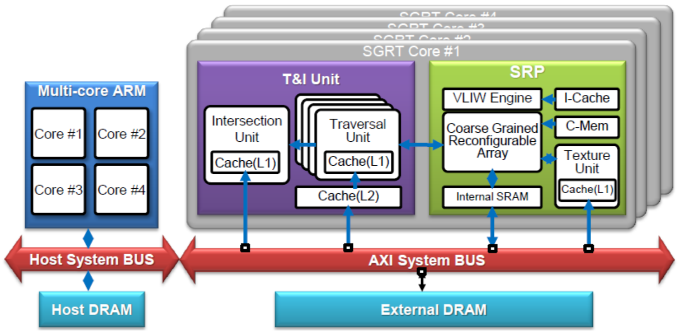

- H/W approaches
  - MRTP

  [Kim et al. 2012, 2013]



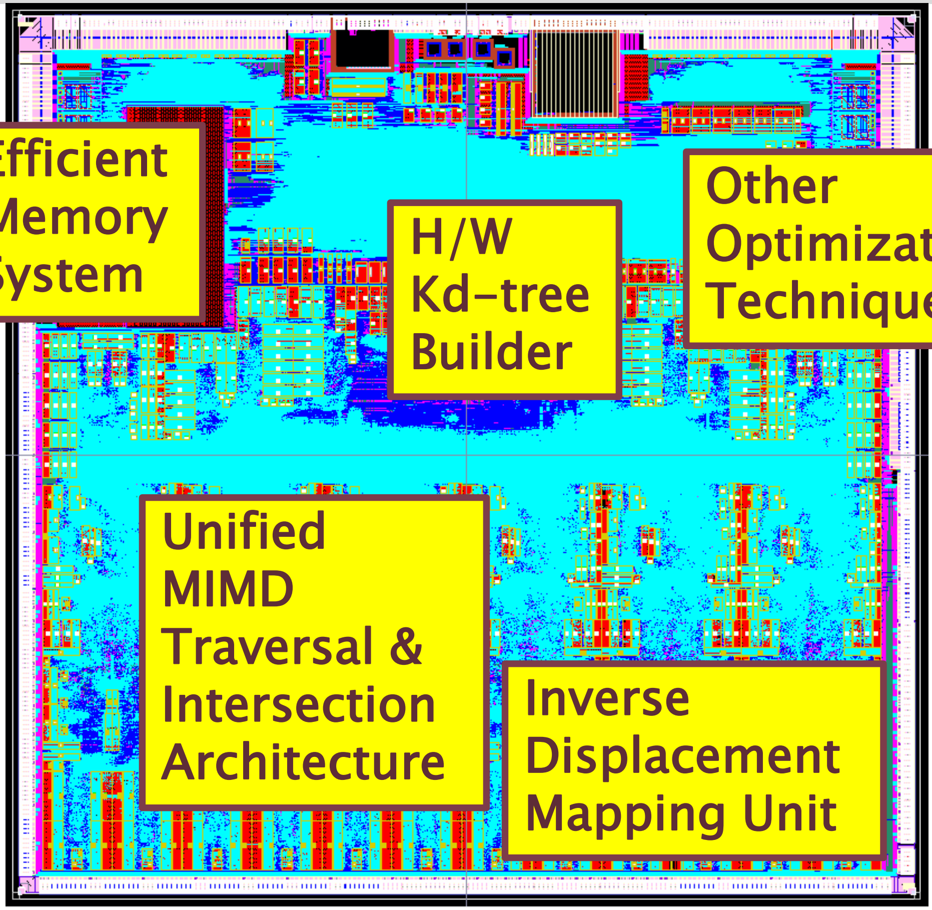

  - SGRT

  [Lee et al. 2012, 2013]




- Either no real-time performance or results based on S/W simulations

# Main Results




**Performance:**
~239M rays/s &
~6M triangles/s

**Area:**
$20mm^2$
@28nm HPL

**Power consumption:**
1W@28nm HPL

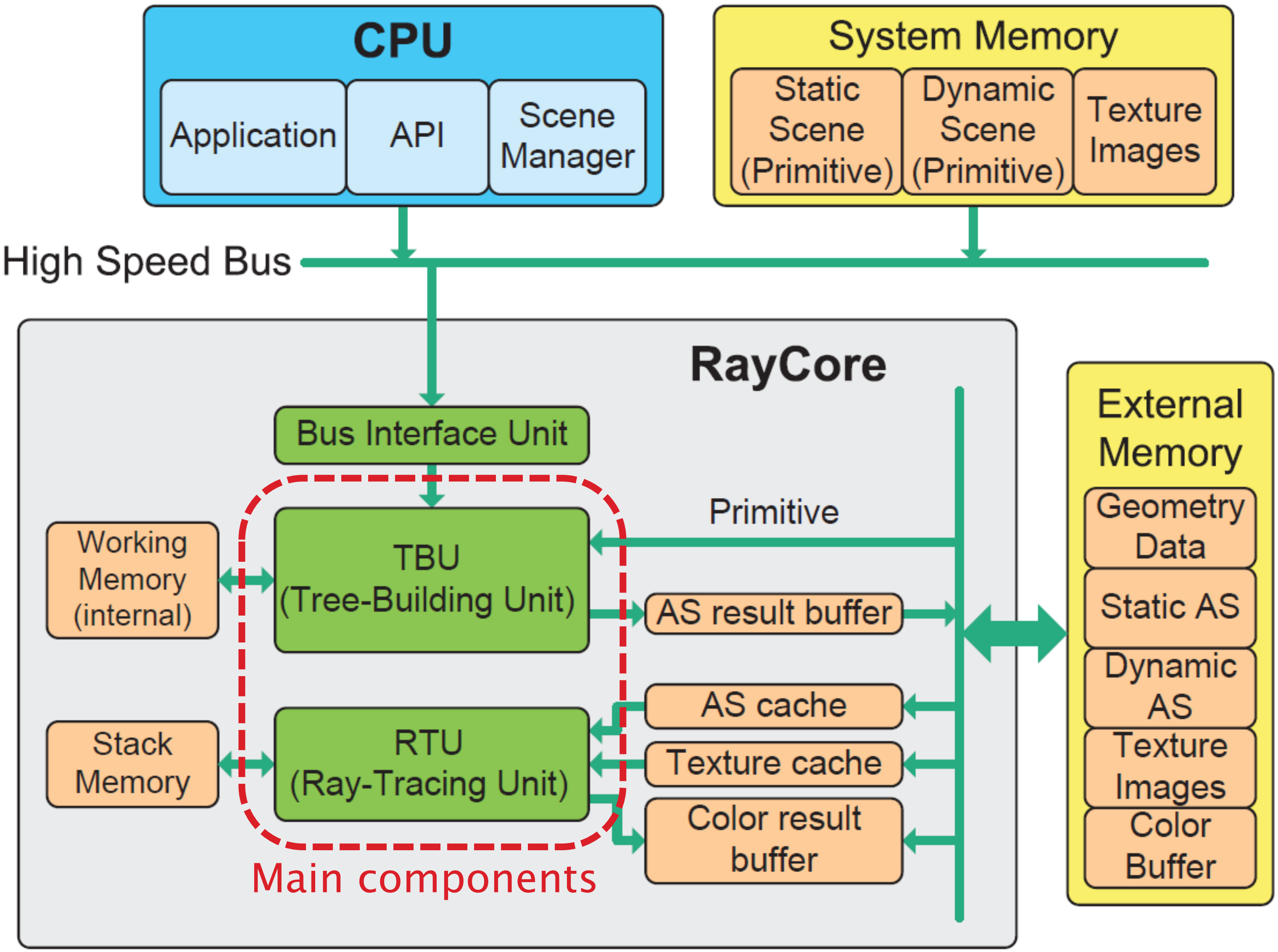
~60X faster than previous mobile RT H/W

Similar to current mobile GPUs

# Overall H/W Architecture & Design Decisions

# RayCore H/W Architecture

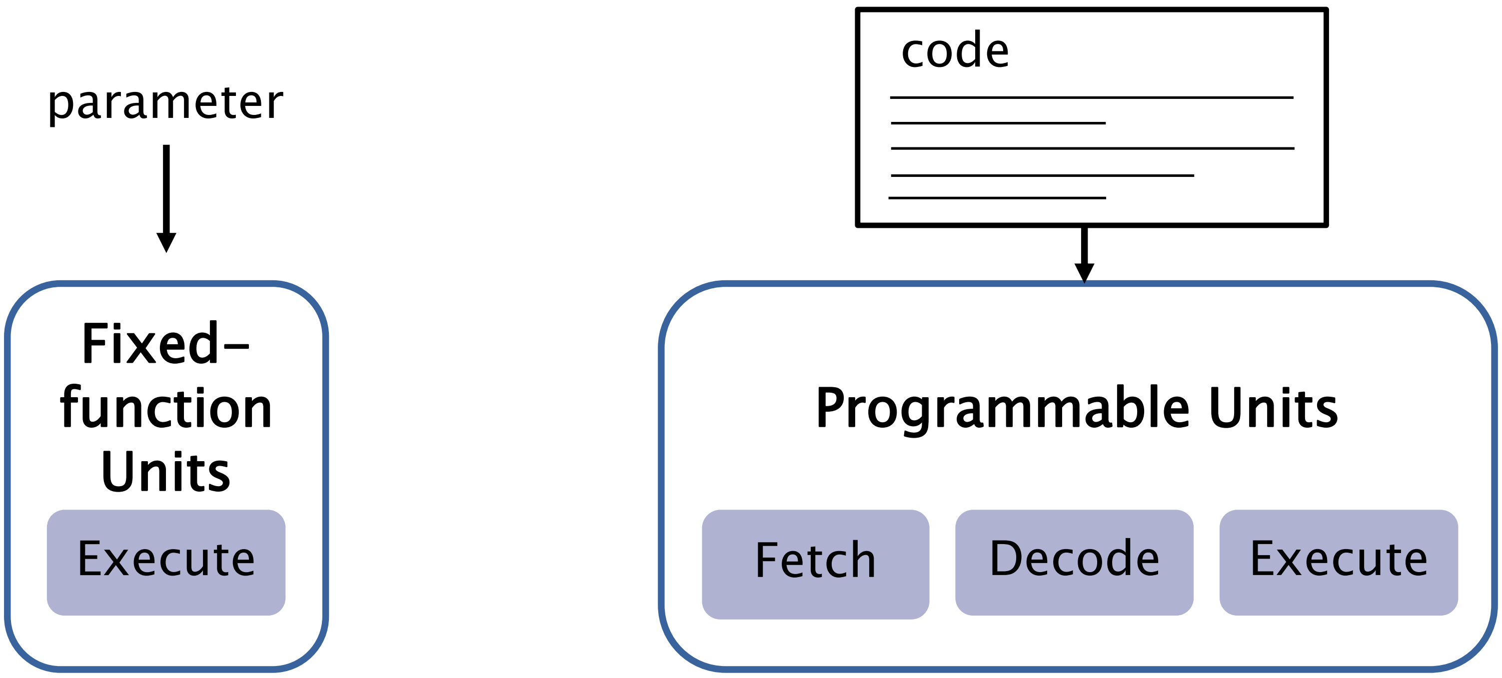
Mobile Media Processor Lab.
Sejong Univ.



# Design Decisions



- Fixed-function units vs programmable units



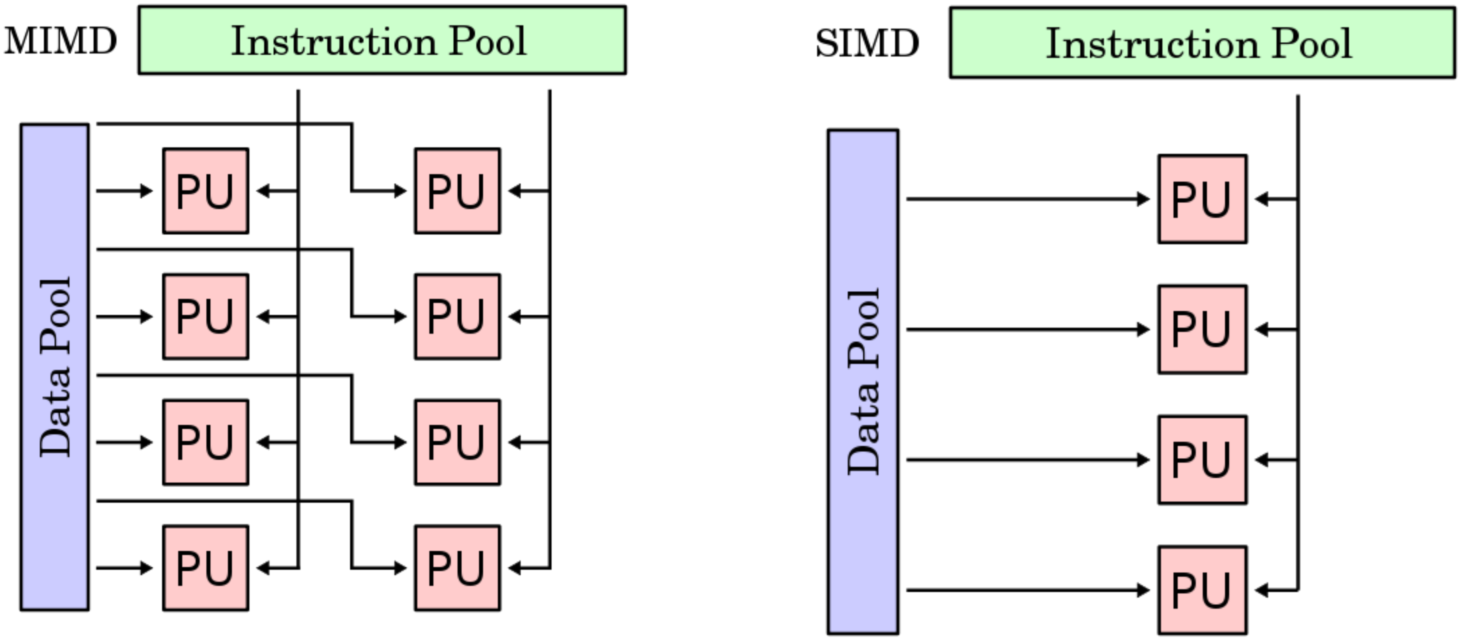

√ Our choice

- High area and power efficiency

# Design Decisions



- MIMD vs SIMD

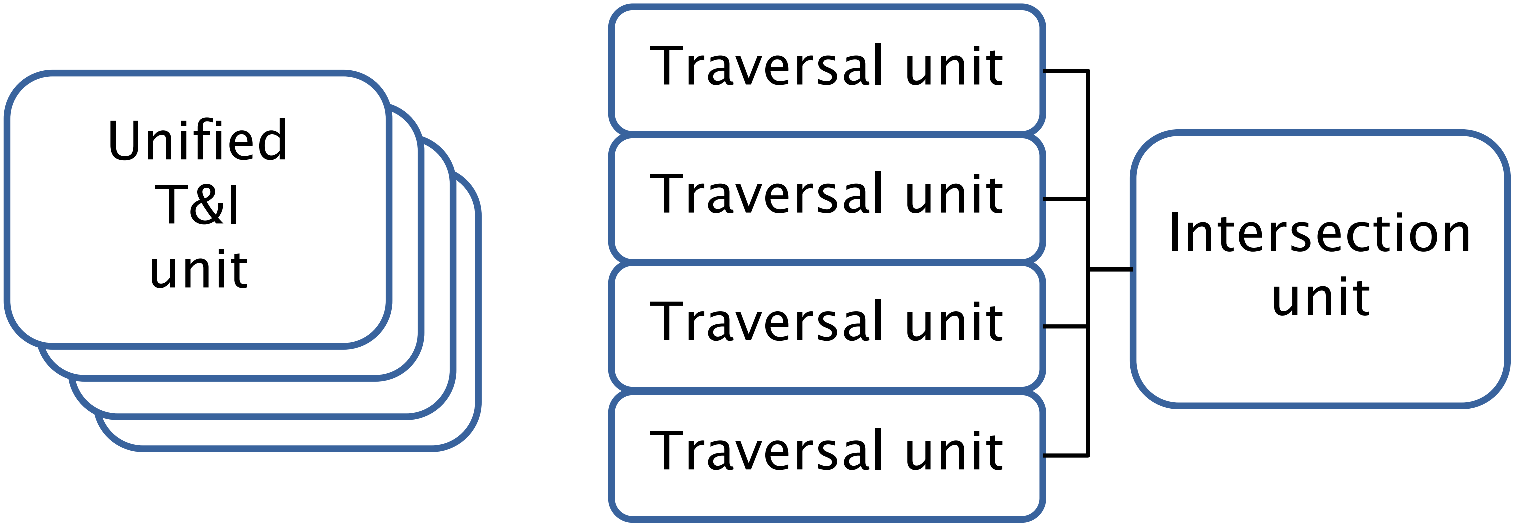



√ Our choice

- High H/W utilization regardless of ray coherence

# Design Decisions

- Unified traversal & intersection (T&I) units vs separate T&I units



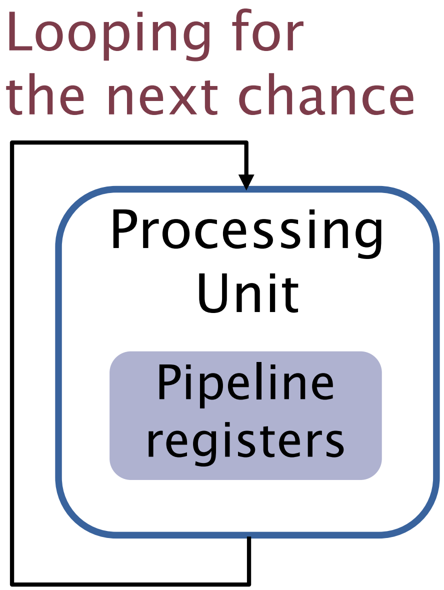

√ Our choice

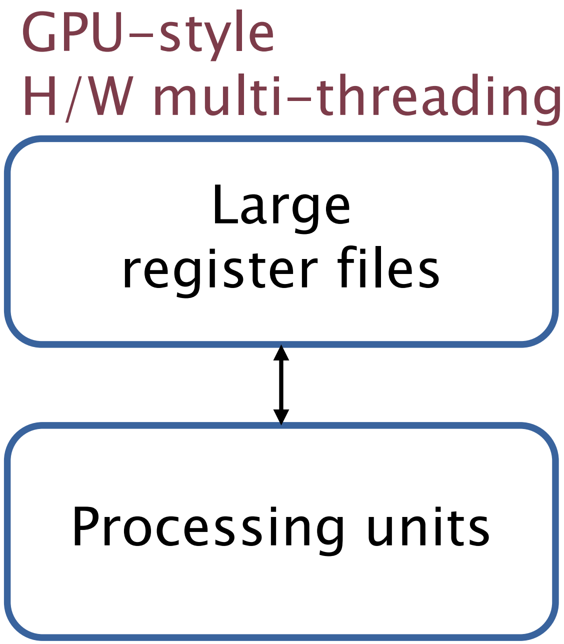
- No load imbalance problem in prior separate T&I units

# Design Decisions



- Multi-threading method



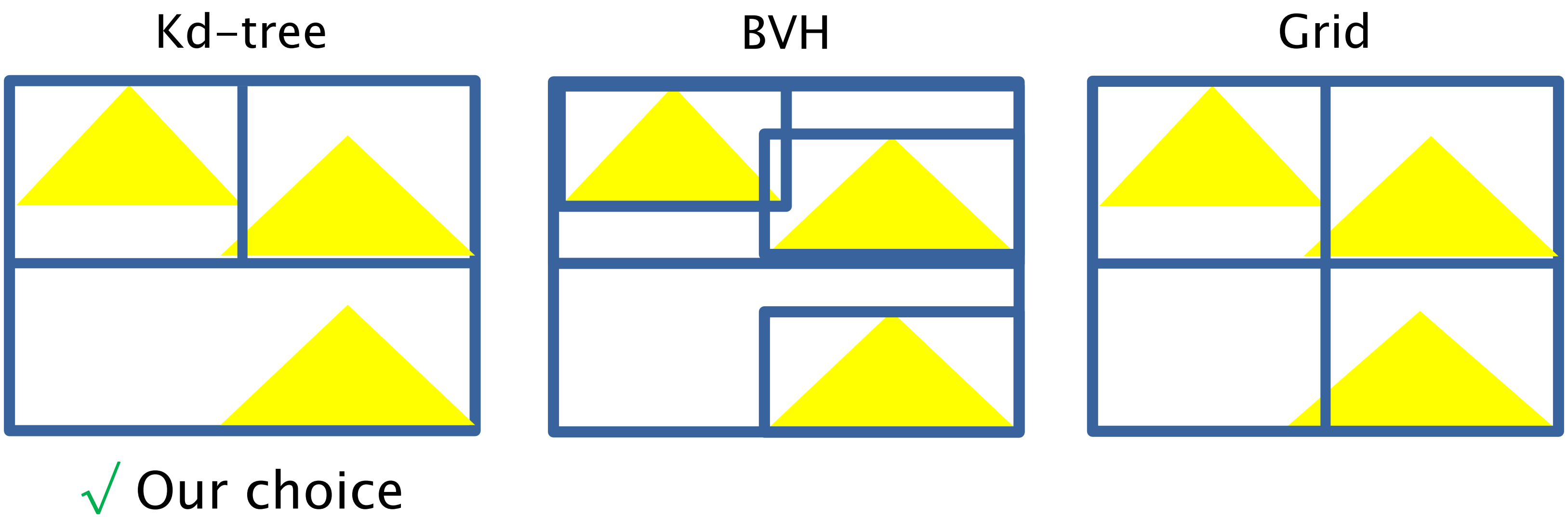

√ Our choice

  - Reuse existing buffers/registers to minimize additional register files

# Design Decisions



- Acceleration structure (AS)




  - Fast traversal with early termination
  - Good cache efficiency with a small node size (8 bytes)
  - Our H/W tree builder solves the tree-build time problem

# Rendering effects

- Full Whitted ray-tracing effects

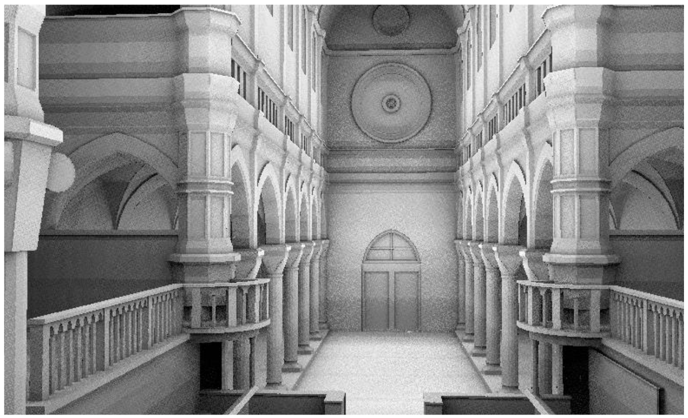
Specular reflection

Refraction

Shadows

# Rendering effects

- Distribution ray-tracing effects



Ambient occlusion (AO)

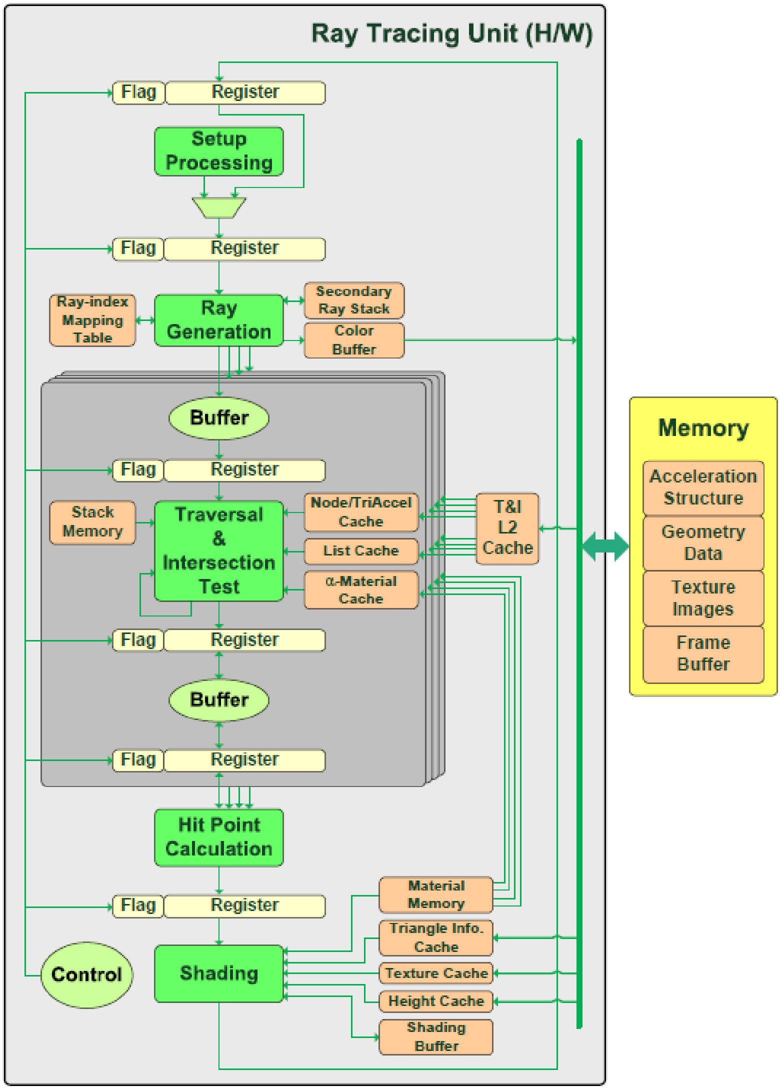
Diffuse inter-reflection

# Rendering effects

- Inverse displacement mapping

# Ray-Tracing Unit (RTU)
# & Tree-Building Unit (TBU)

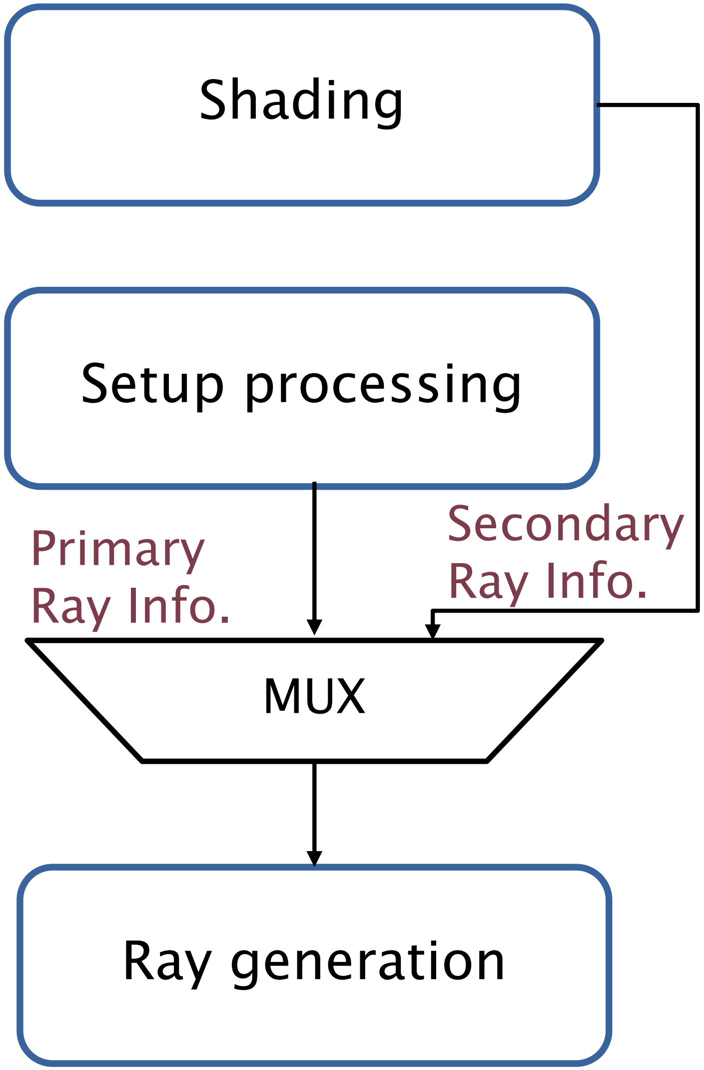
# Ray-Tracing Unit (RTU)




- Setup-processing unit
- Ray-generation unit
- Multiple traversal & intersection (T&I) units
- Hit-point calculation unit
- Shading unit

# Setup Processing & Ray Generation



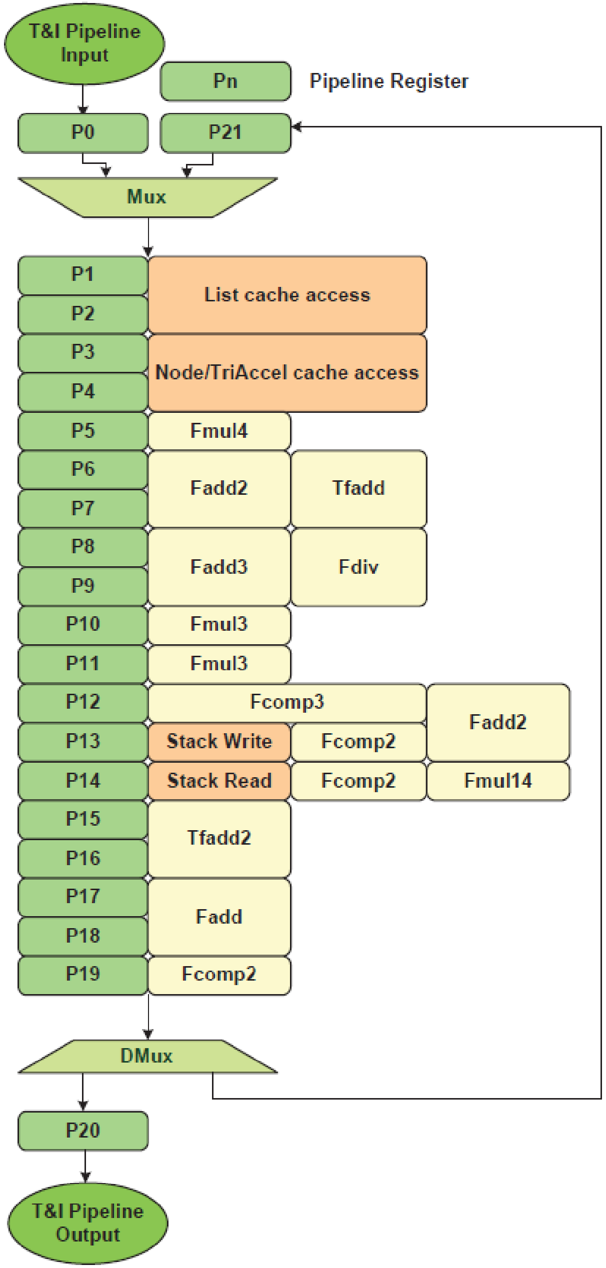



- Initialize primary-ray information (ray type+ray index)
- Pass secondary-ray information defined by the shading unit

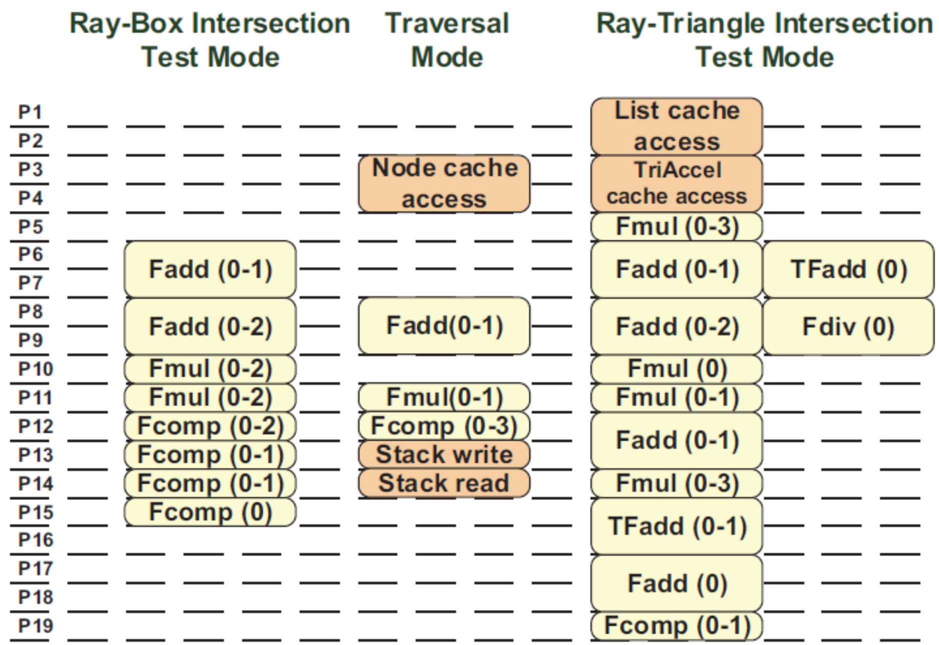
- Generate primary/ secondary/shadow rays

# Unified T&I Pipeline



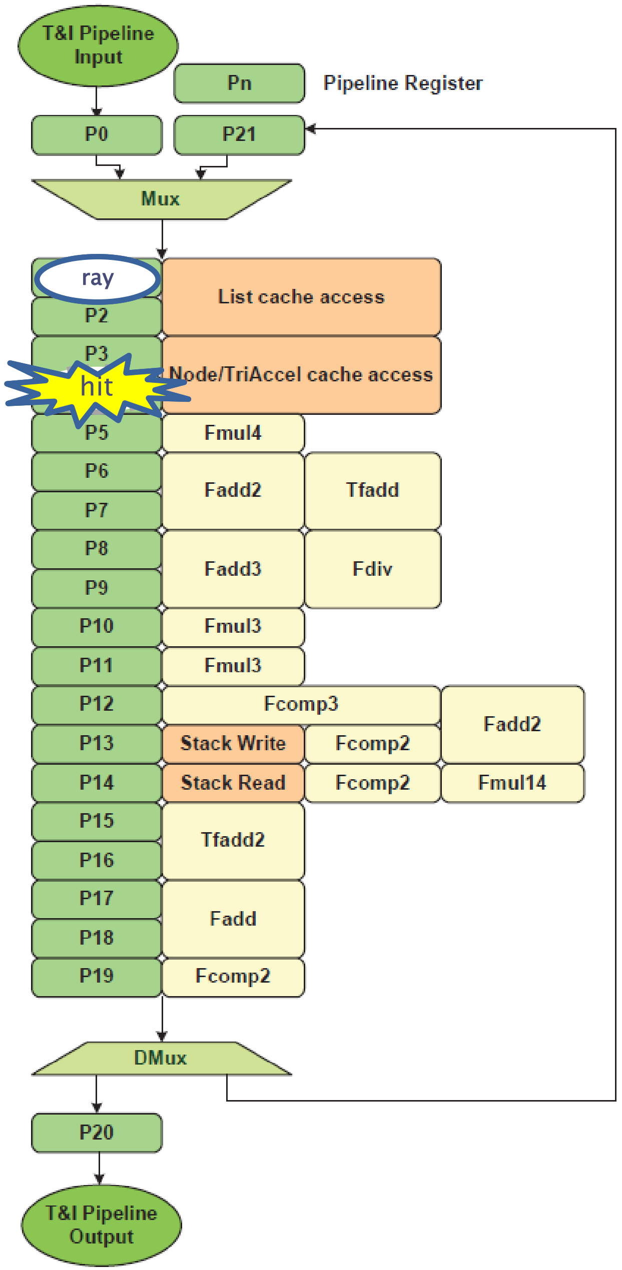



- A single pipeline with three modes




- No load imbalance problem
- Greatly simplified control logic and interfaces between units

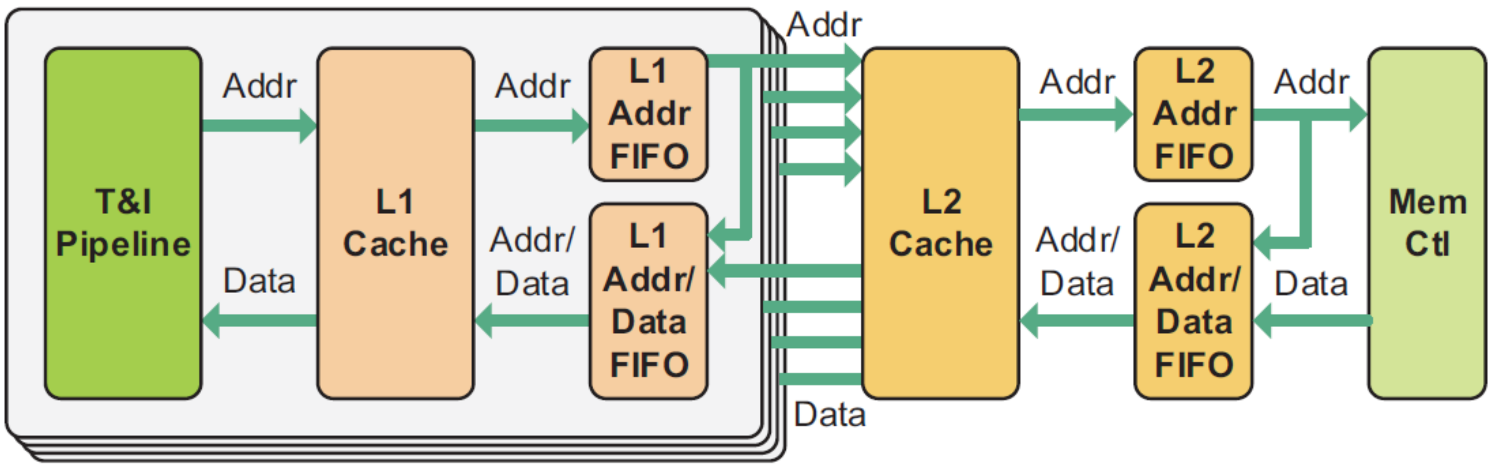
# Looping for the Next Chance




- Simple multi-threading technique
- Operation
  - Cache miss → idle
  - Next loop → reactive
  - A cache miss acts as prefetching data
- Pros
  - Ease of H/W implementation
  - Use of existing internal memory (input/output buffers & pipeline registers)

# Memory System of the T&I unit

- Two-levels of caches (L1/L2)
  - L1/L2 Address FIFO for handling memory requests
  - L1/L2 Address/Data FIFO for delivering address & data to the upper-level cache



# Hit-Point Calculation and Shading



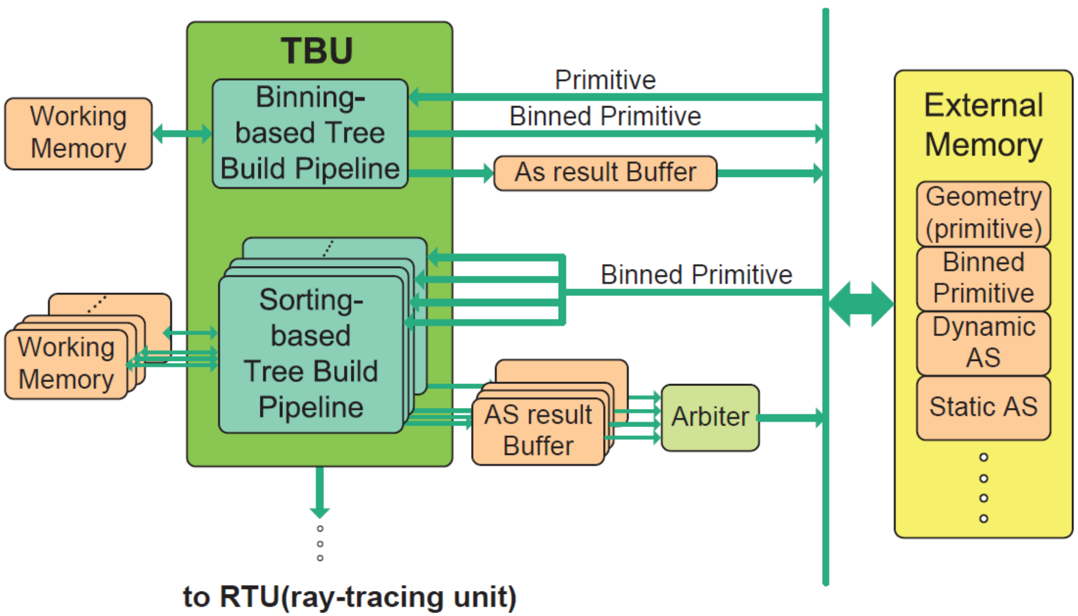
- Hit-point calculation
  - Calculate the final (x,y,z) position

    $p(t) = o + t \cdot d$

- Shading
  - Phong illumination
  - Texture mapping with bilinear filtering & mip-mapping
  - Inverse displacement mapping

# Tree-Building Unit (TBU)

- Two pipeline types
  - Binning for upper-level nodes + sorting for lower-level nodes
  - Fast kd-tree construction without significant tree-quality degradation
- Memory traffic reduction techniques
  - Internal working memory for local tree-building procedure
  - Node scheduling for burst memory accesses



# Ray Tracing API



- OpenGL ES 1.1-like API
  - Provides similar interfaces to OpenGL ES programming
  - Static objects/tree are retained for subsequent frames
  - Dynamic objects/subtrees are updated during each frame

- Complete specification and its programming guide are provided on the Siliconarts homepage (www.siliconarts.com)

```
// set a box
rcStaticSceneBegin();
…
…
rcVertexPointer(3, RC_FLOAT, 0, box);
rcGenMaterials(1, &material_box);
rcBindMaterial(material_box);
rcMaterialf (RC_FRONT_AND_BACK, RC_REFLECTION, 0.0f);
rcMaterialfv(RC_FRONT_AND_BACK, RC_DIFFUSE, &cyan.r);
rcDrawArrays(RC_TRIANGLES, 0, 30);

rcDisableClientState(RC_VERTEX_ARRAY);
rcStaticSceneEnd();
```

# Implementation and Results

# FPGA Prototype

- iNEXT-V6 board
  - 2 Xilinx Virtex-6 LX550 FPGA chips
  - 2 GB of DDR3 RAM & 8 MB of SRAM
  - A TFT LCD board with 800x480 resolution
  - PCI Express interface

- Our implementation
  - 4 FPGA chips for 4 RTUs (with 2 iNEXT-V6 boards)
  - 1 FPGA chip for 1 TBU
  - 84 MHz core and memory clock
  - Total required SRAM: 507KB for an RTU and 218KB for a TBU

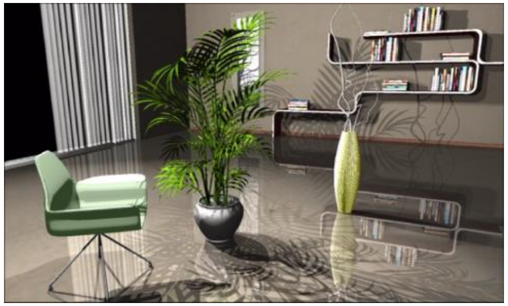
# ASIC Evaluation

- TSMC's 28 nm HPL process and Synopsys design compiler
- Clock frequency: 500MHz@0.9V
- Area: 3mm$^2$ per RTU (18mm$^2$ for 6 RTUs) + 1.6mm$^2$ per TBU
- Internal power consumption : 1W for 6 RTUs and 1 TBU
- <u>Particularly suitable for mobile devices</u>

# RT Performance on the FPGA

- Whitted ray tracing

Kitchen



Moving light

Living room

  - 4 RTUs @ 84MHz
  - 3-5 rays per pixel

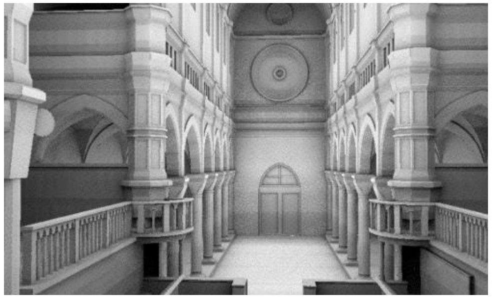
  - Performance: 23-26 Mrays/s
  - Memory traffic: 46-122 MB/s
  - FPS: 13-21 @ 840 X 480 resolution
  - **Interactive** frame rates

# RT Performance on the FPGA

Mobile Media Processor Lab. Sejong Univ.

- Distribution ray tracing

Conference



Sibenik

- 4 RTUs @ 84MHz
- 2 (primary), 16 (AO), 32 (diffuse) rays per pixel

- Performance: 21 (primary), 23 (AO), 18–20 (diffuse) Mrays/s
- Memory traffic: 8-62 (primary / AO), 420-605 (diffuse) MB/s

- **Low** performance degradation when tracing **incoherent** rays

# RT Performance on the FPGA

- Inverse displacement mapping (IDM)

BART Kitchen

(IDM off)

(IDM on)

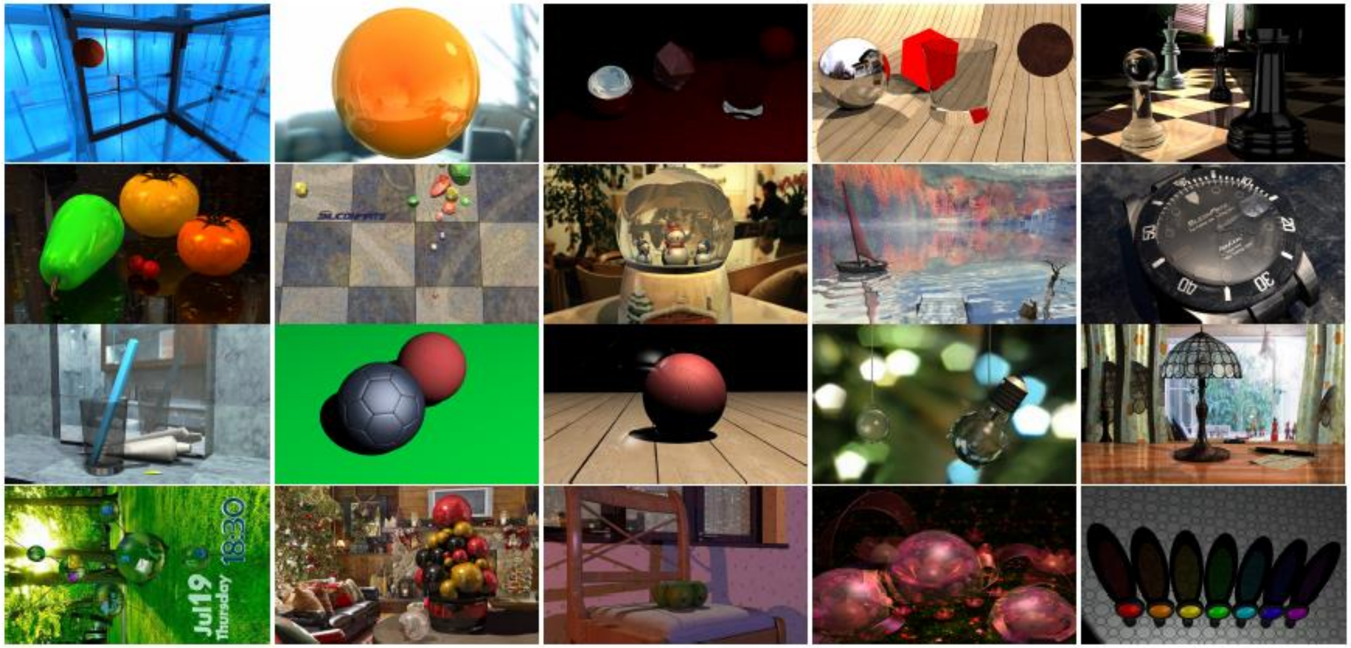
  - 4 RTUs @ 84MHz
  - Performance: 18-20 Mrays/s
  - 30% increase of memory traffic to access the height map
  - Detailed results are included in

Kwon *et al*., Effective traversal algorithms and hardware architecture for pyramidal inverse displacement mapping, *Computers & Graphics*, 2014

# KD-tree Construction Performance

- Performance (1 TBU)
  - 1.6~117.9 ms/frame @ 84MHz for 0.6~64K triangles
  - 0.4~1M triangles/s

- Memory traffic
  - 0.1~36.1 MB/frame



# Performance Evaluation of the ASIC Ver.

- RTU performance
  - 9X faster than the FPGA ver. (84→500MHz & 4→6 RTUs)
  - ~239M rays/s & 56 FPS@720p
- TBU performance
  - 6X faster than the FPGA ver. (84→500MHz & 1 TBU)
  - 2~6M triangles/s
- Required memory bandwidth
  - 1.1GB/s on 6 RTUs for Whitted ray tracing
  - 1.1GB/s on 1 TBU for 30FPS kd-tree construction
  - Much less than the bandwidth of dual LPDDR3 1333MHz (12.8GB/s)

# Preliminary Comparison

- Ray-tracing performance in the Conference scene

| | GPU [Gribble and Naveros 2013] | MIC [Benthin et al. 2012] | Mobile RT H/W [Kim et al. 2013] | RayCore ASIC (ours) |
|---|---|---|---|---|
| **Mrays/s** | 500 | 210 | 4 | **239** |
| **Platform** | NVIDIA GTX690 | Intel MIC | Reconf. SIMT | **RTU** |
| Process(nm) | 28 | 45 | 90 | 28 |
| Clock (MHz) | 915 | 1200 | 50–400 | 500 |
| **Area (mm²)** | 294X2 | – | 16 | **18 (6 RTUs)** |
| **Power consumption(W)** | 300 (TDP) | – | 0.2 @100MHz | **1 (RTU+TBU)** |

- Desktop-level performance
- Mobile-level area and power consumption

# Preliminary Comparison

Mobile Media Processor Lab.
Sejong Univ.

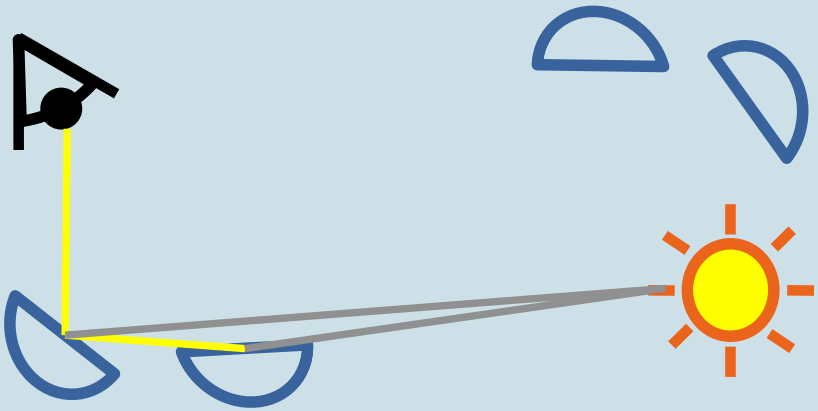
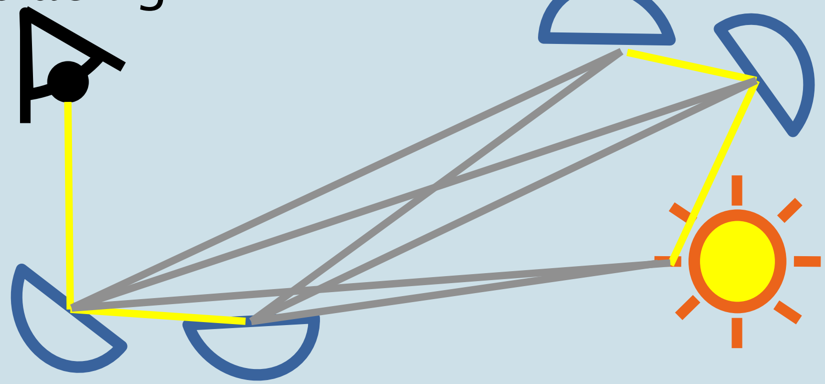
- KD-tree construction performance

| | CPU [Shevtsov et al. 2007] | GPU [Hou et al. 2011] | RayCore ASIC (ours) |
|---|---|---|---|
| **Time to build a kd-tree (ms)** | 27 | 38 | **20** |
| Scene (# of tris) | Bunny (69K) | Robots (71K) | Transparent Shadows (64K) |
| **Platform** | Intel Core2 Duo X 2 | NVIDIA GTX280 | **TBU** |
| Process | 65 | 55 | 28 |
| Clock (MHz) | 3000 | 1476 | 500 |
| **Area ($mm^2$)** | 143 X 2 | 576 | **1.6 (1 TBU)** |
| **Power consumption(W)** | 65 X 2 (TDP) | 236 (TDP) | **1 (RTU+TBU)** |

  - Comparable performance to CPU/GPU approaches & much less H/W resources/power consumption

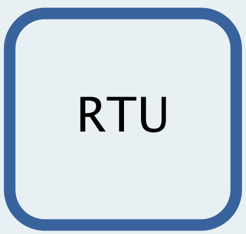
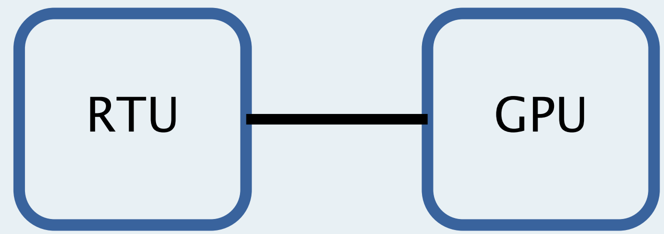
# Conclusions, Limitations, and Future Work

# Limitations and Future Work

| Limitations | Future work |
|---|---|
| Focus on Whitted ray tracing<br> | To accelerate bidirectional path tracing<br> |

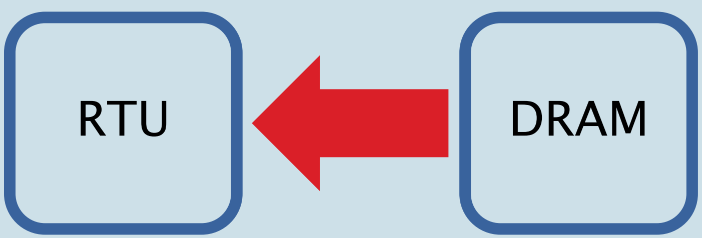
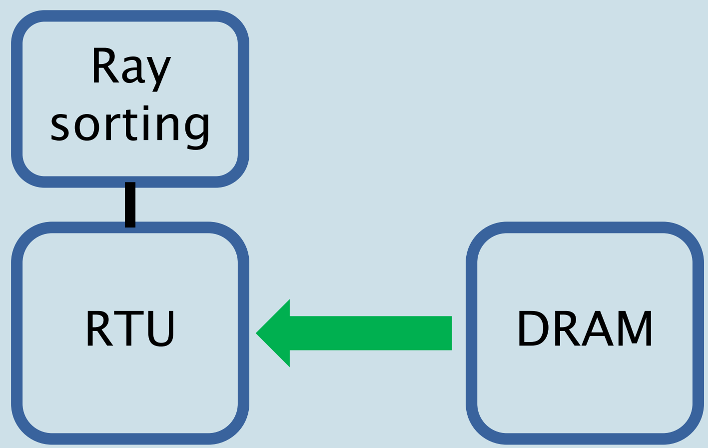
# Limitations and Future Work



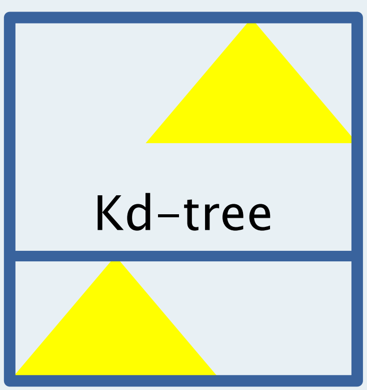
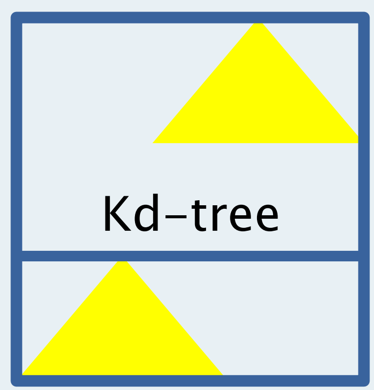
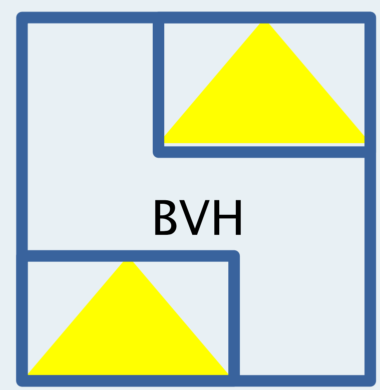
| Limitations | Future work |
|---|---|
| Focus on Whitted ray tracing | To accelerate bidirectional path tracing |
| Fixed pipelines | To combine with programmable shaders |
|  |  |

# Limitations and Future Work



| Limitations | Future work |
|---|---|
| Focus on Whitted ray tracing | To accelerate bidirectional path tracing |
| Fixed pipelines | To combine with programmable shaders |
| High memory traffic on incoherent ray tracing<br> | Additional ray-sorting logic<br> |

# Limitations and Future Work

| Limitations | Future work |
|---|---|
| Focus on Whitted ray tracing | To accelerate bidirectional path tracing |
| Fixed pipelines | To combine with programmable shaders |
| High memory traffic on incoherent ray tracing | Additional ray-sorting logic |
| Only support kd-trees   | To support both kd-trees and BVHs     |

# Conclusions

- A new hardware ray tracer for mobile devices including
  - Unified T&I pipelines
  - H/W kd-tree builder
  - Other various novel techniques

- RayCore can be used for various mobile applications
  - Games, UX, AR, etc.
  - High-quality images & simpler programming

# Acknowledgements

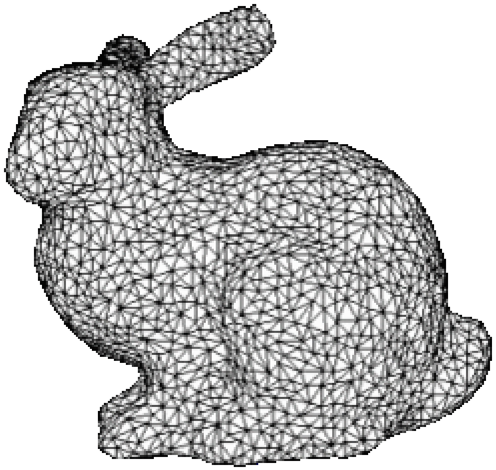
**Siliconarts**, NRF, ARO & NSF

# Q&A

# Backup Slides

# Design Decisions

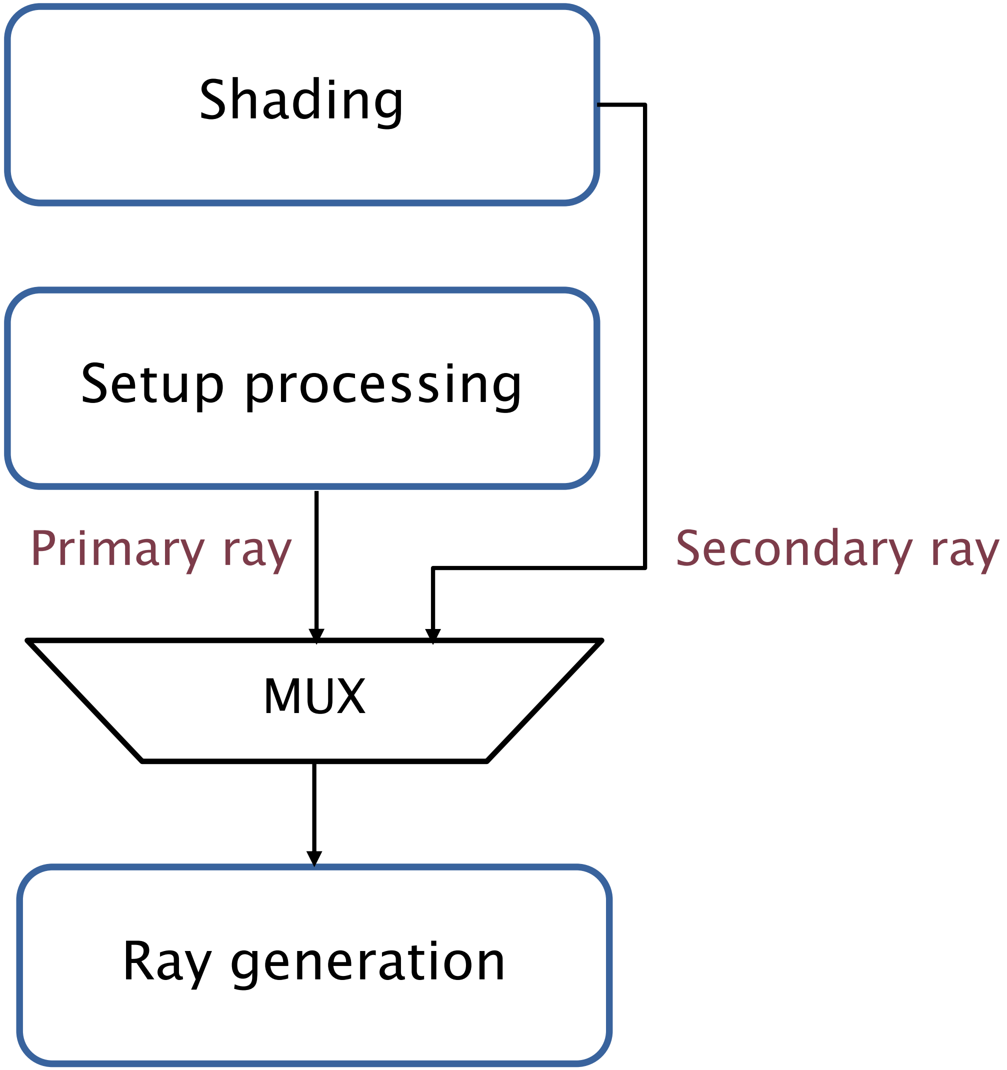
- Primitive type
  - Support only triangles as geometric primitives



  - Improve the performance & simplify the system design
  - Use Wald's ray-triangle intersection algorithm [Wald 2004]

# Setup Processing

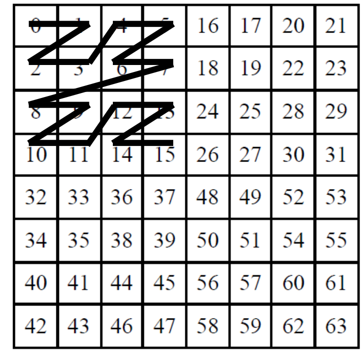
- Initialize primary-ray information (ray type+ray index)
- Pass secondary-ray information defined by the shading unit



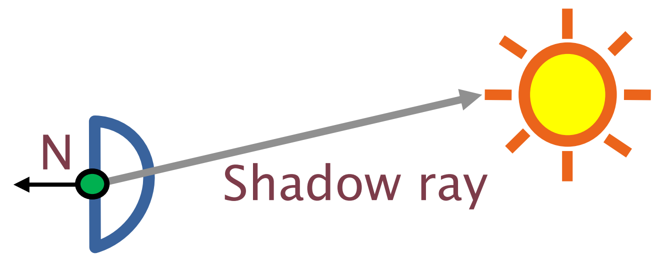
# Ray Generation



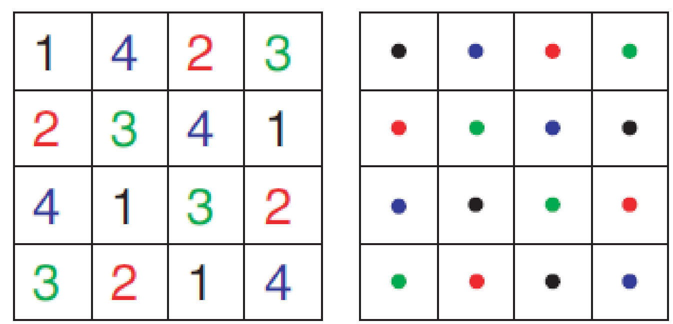
- Primary ray generation using the Morton order [Morton 1966]
  - Increase cache efficiency



- Shadow-ray back-face culling [Suffern 2007]
  - Remove unnecessary shadow-ray traversal




- Sudoku sampling for distribution rays [Boulos et al. 2006]
  - Prevent temporal scintillation



# Hit-Point Calculation and Shading

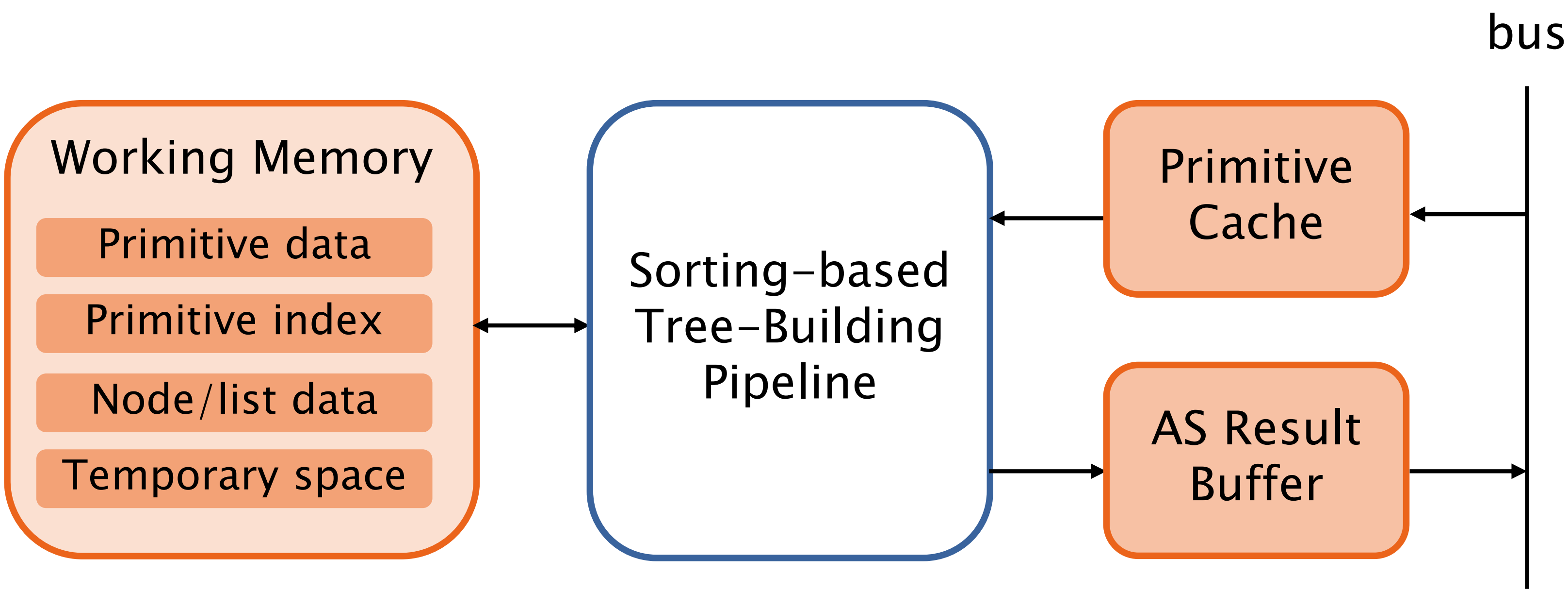
- Hit-point calculation
  - $p(t) = o + t \cdot d$

  where $o$ is the ray's origin and $d$ is the ray's normalized direction

- Shading
  - Phong illumination, texture mapping and bilinear filtering
  - Ray-length-based mip-mapping [Park et al. 2011] + a texture cache → high cache hit rates (~96%)
  - Inverse displacement mapping [Kwon et al. 2014]

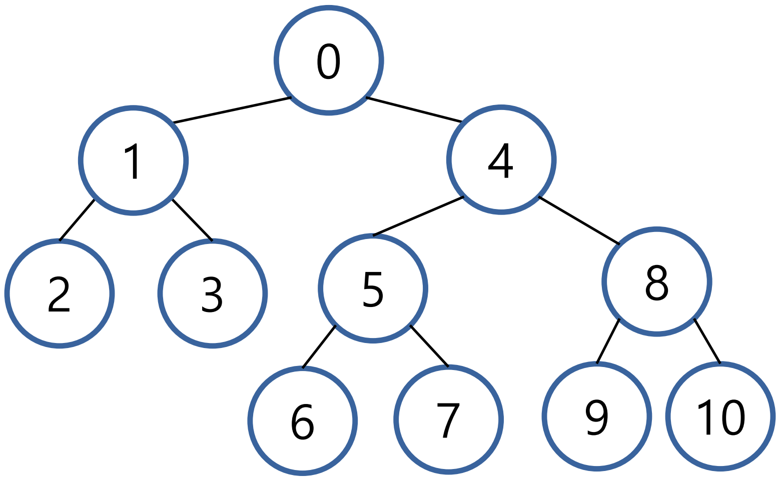
# Tree-Building Unit (TBU)

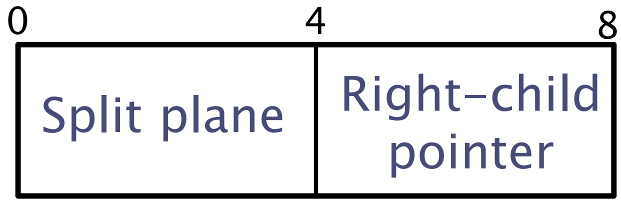
- Internal SRAM for a sorting-based pipeline
  - All data of the tree-building procedure (sorting, split plane selection, and geometry classification) are maintained in the internal working memory
  - Minimize off-chip memory accesses



# Tree-Building Unit (TBU)

- Node scheduling for burst memory accesses
  - We use the 8-byte compact depth-first layout [Pharr and Humphreys 2010]



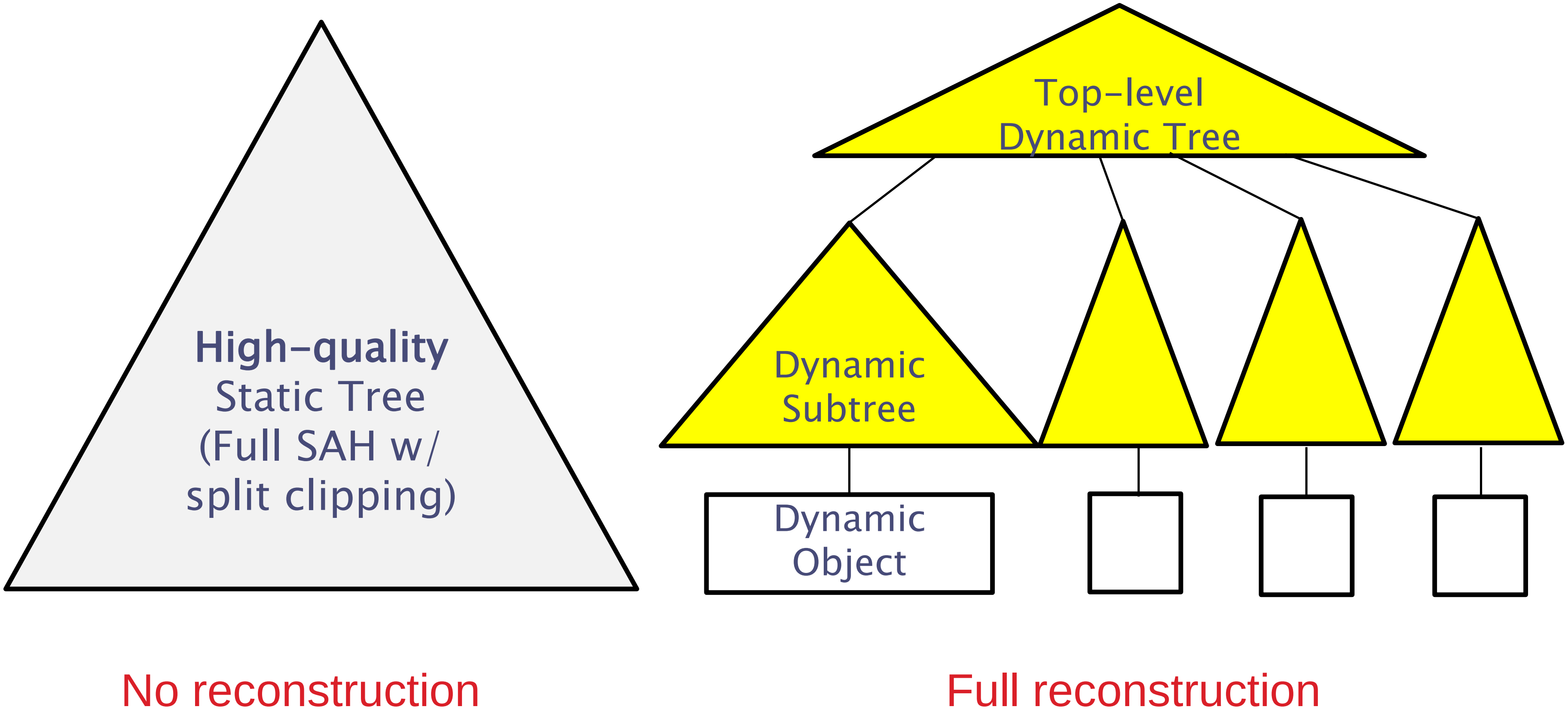




  - Building a depth-first layout is difficult in our parallel tree construction
  - For burst memory accesses, a scheduler reallocates a node construction sequence as the depth-first layout, as far as possible

# Handling Dynamic Trees



- We want to construct subtrees for only dynamic parts
- Combination of two-tree [Bikker 2007] and two-level [Wald et al. 2003] approaches



# Comparison



- Ray-tracing performance in the Conference scene

| | GPU [Gribble and Naveros 2013] | MIC [Benthin et al. 2012] | Desktop RT H/W [ImgTec 2013] | Mobile RT H/W [Kim et al. 2013] | RayCore ASIC (ours) |
|---|---|---|---|---|---|
| **Mrays/s** | 500 | 210 | 50* | 4** | **239** |
| **Platform** | NVIDIA GTX690 | Intel MIC | Caustic R2100 | Reconf. SIMT | **RTU** |
| Process(nm) | 28 | 45 | 90 | 90 | 28 |
| Clock (MHz) | 915 | 1200 | - | 50-400 | 500 |
| **Area ($mm^2$)** | 294X2 | - | - | 16 | **18 (6 RTUs)** |
| **Power consumption(W)** | 300 (TDP) | - | 30 (max) | 0.2 @100MHz | **1 (RTU+TBU)** |
| Normalized Power efficiency | 1.0X | - | 1.0X | 3.0X | 143.4X |

* Average performance in common scenes, ** Bunny scene

  - Comparable performance to desktop GPU/MIC approaches & much less H/W resources/power consumption

# Comparison

Mobile Media Processor Lab.
Sejong Univ.

## KD-tree construction performance

| | CPU [Shevtsov et al. 2007] | GPU [Hou et al. 2011] | RayCore ASIC (ours) |
|---|---|---|---|
| **Time to build a kd-tree (ms)** | 27 | 38 | **20** |
| Scene (# of tris) | Bunny (69K) | Robots (71K) | Transparent Shadows (64K) |
| **Platform** | Intel Core2 Duo X 2 | NVIDIA GTX280 | **TBU** |
| Process | 65 | 55 | 28 |
| Clock (MHz) | 3000 | 1476 | 500 |
| **Area ($mm^2$)** | 143 X 2 | 576 | **1.6 (1 TBU)** |
| **Power consumption(W)** | 65 X 2 (TDP) | 236 (TDP) | **1 (RTU+TBU)** |
| Normalized power efficiency | 1.0X | 0.4X | 162.7X |

- Comparable performance to CPU/GPU approaches & much less H/W resources/power consumption

# Preliminary Comparison

Mobile Media Processor Lab. Sejong Univ.

- Ray-tracing performance in the Conference scene

| | GPU [Gribble and Naveros 2013] | MIC [Benthin et al. 2012] | Desktop RT H/W [ImgTec 2013] | Mobile RT H/W [Kim et al. 2013] | RayCore ASIC (ours) |
|---|---|---|---|---|---|
| **Mrays/s** | 500 | 210 | 50 | 4 | **239** |
| **Platform** | NVIDIA GTX690 | Intel MIC | Caustic R2100 | Reconf. SIMT | **RTU** |
| Process(nm) | 28 | 45 | 90 | 90 | 28 |
| Clock (MHz) | 915 | 1200 | - | 50-400 | 500 |
| **Area ($mm^2$)** | 294X2 | - | - | 16 | **18 (6 RTUs)** |
| **Power consumption(W)** | 300 (TDP) | - | 30 (max) | 0.2 @100MHz | **1 (RTU+TBU)** |

- Comparable performance to desktop GPU/MIC approaches & much less H/W resources/power consumption