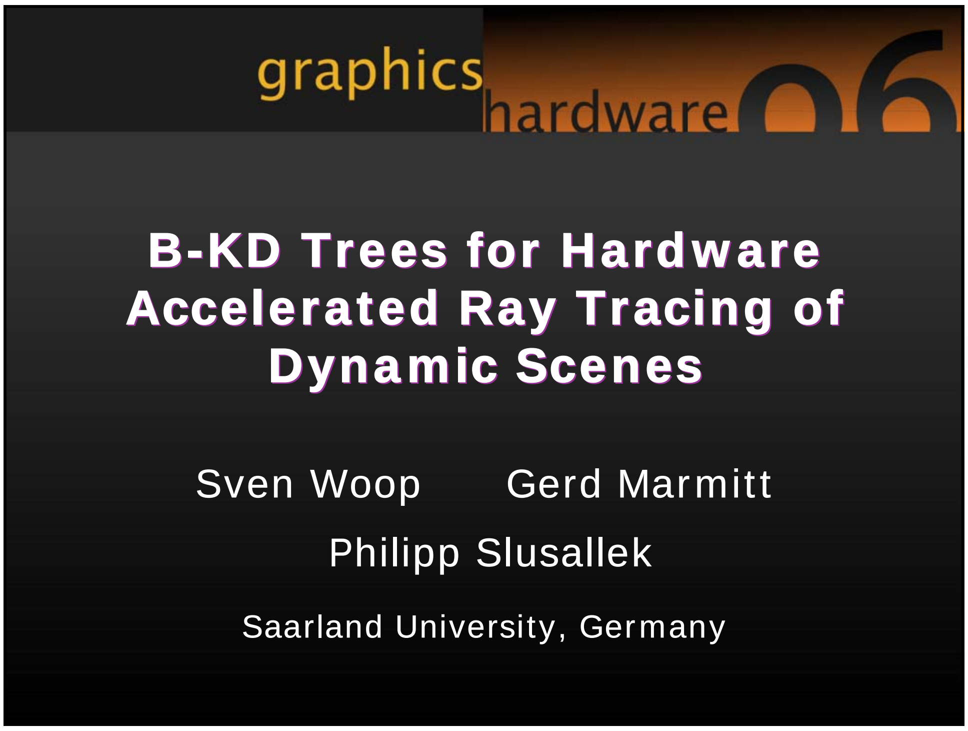
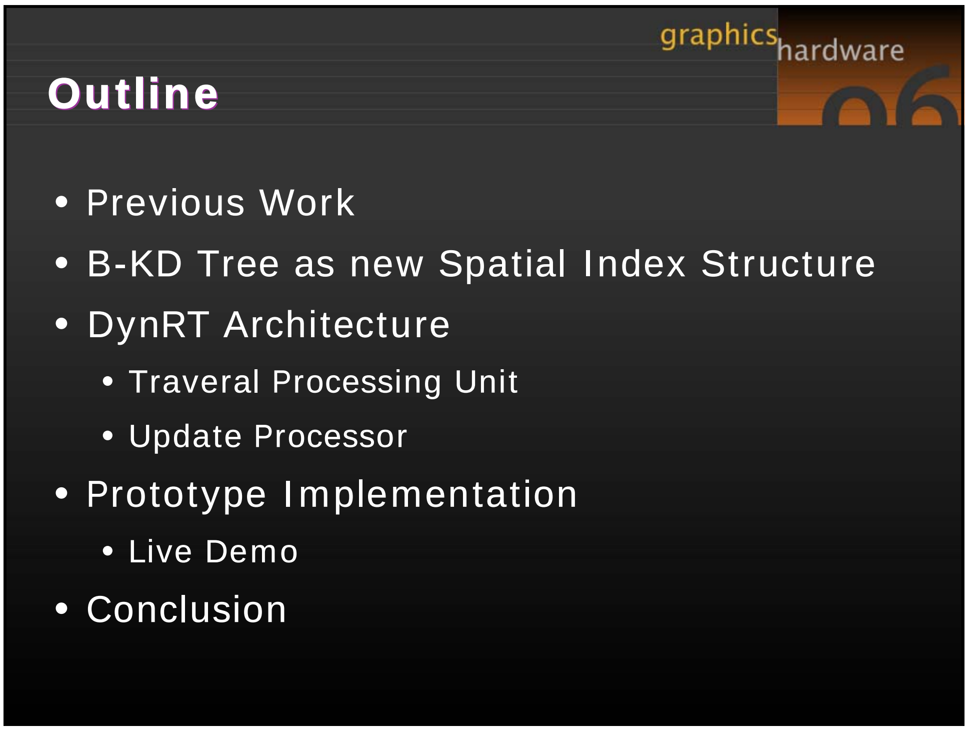

B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes

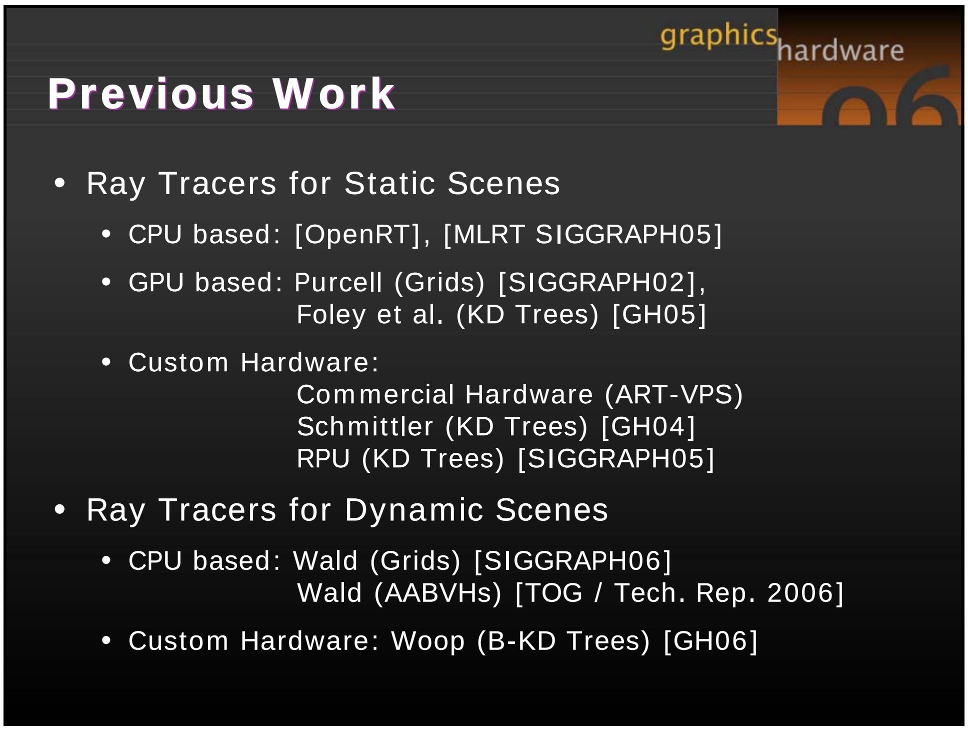
Sven Woop Gerd Marmitt

Philipp Slusallek

Saarland University, Germany

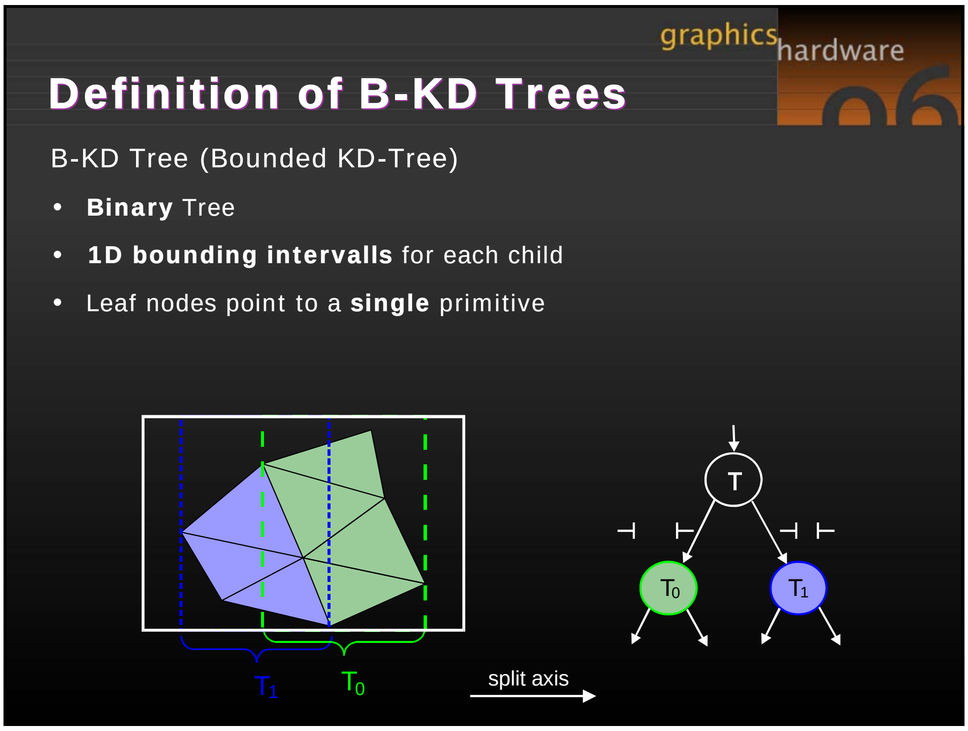

Outline

- Previous Work
- B-KD Tree as new Spatial Index Structure
- DynRT Architecture
  - Traveral Processing Unit
  - Update Processor
- Prototype Implementation
  - Live Demo
- Conclusion



Previous Work

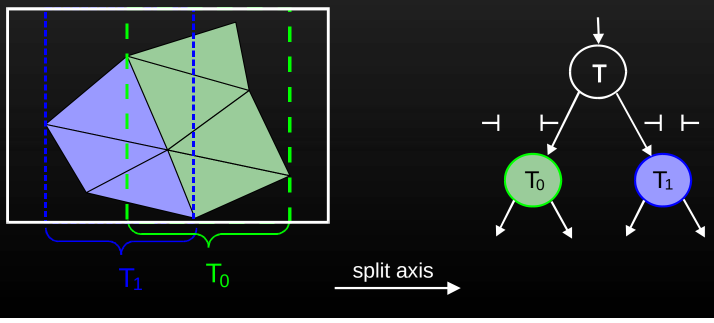
- Ray Tracers for Static Scenes
  - CPU based: [OpenRT], [MLRT SIGGRAPH05]
  - GPU based: Purcell (Grids) [SIGGRAPH02],
    Foley et al. (KD Trees) [GH05]
  - Custom Hardware:
    Commercial Hardware (ART-VPS)
    Schmittler (KD Trees) [GH04]
    RPU (KD Trees) [SIGGRAPH05]
- Ray Tracers for Dynamic Scenes
  - CPU based: Wald (Grids) [SIGGRAPH06]
    Wald (AABVHs) [TOG / Tech. Rep. 2006]
  - Custom Hardware: Woop (B-KD Trees) [GH06]

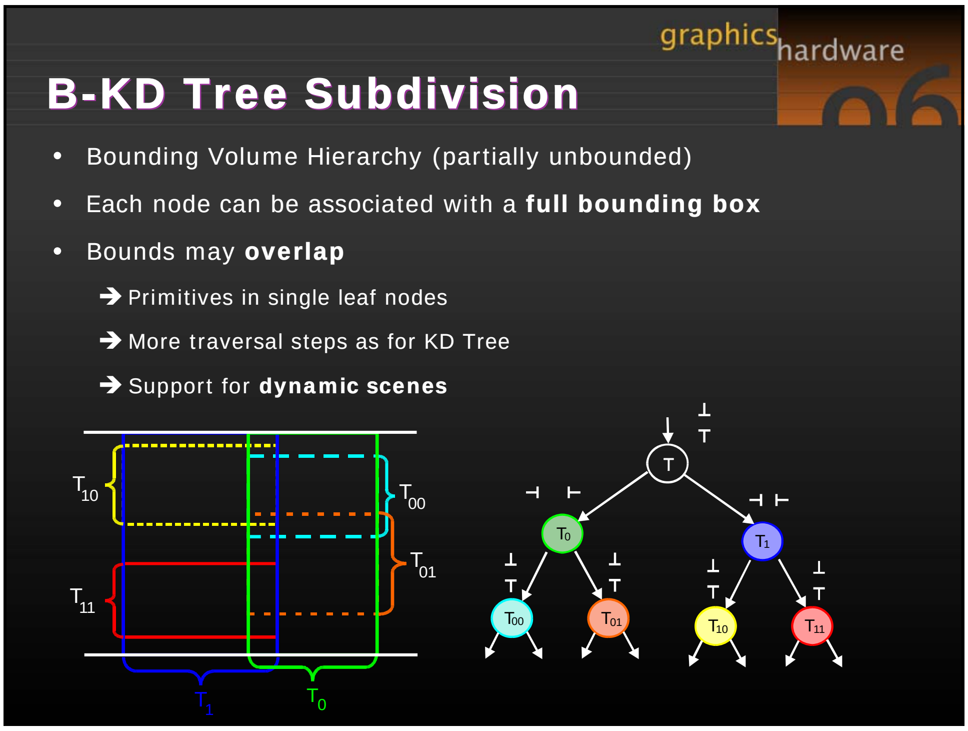


Definition of B-KD Trees

B-KD Tree (Bounded KD-Tree)

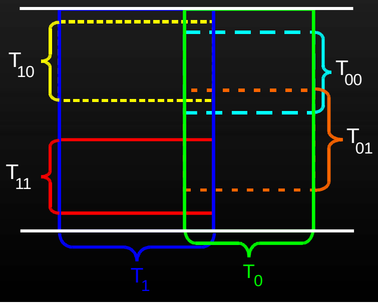
- **Binary** Tree
- **1D bounding intervalls** for each child
- Leaf nodes point to a **single** primitive

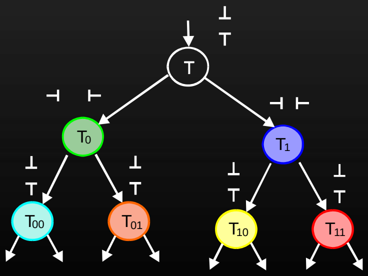


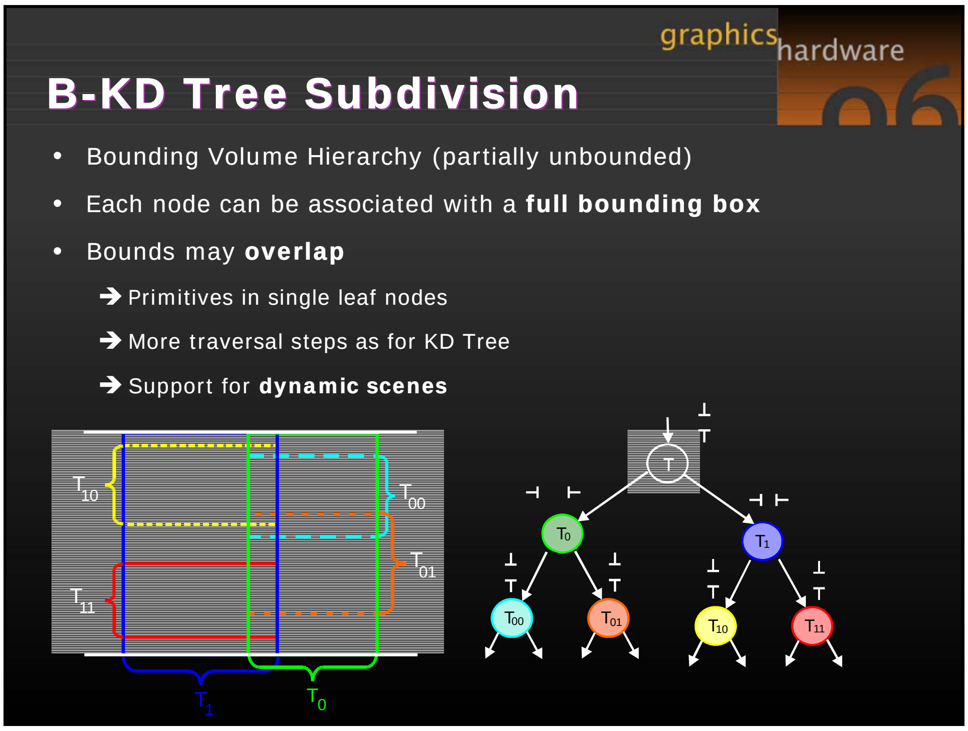


B-KD Tree Subdivision

- Bounding Volume Hierarchy (partially unbounded)
- Each node can be associated with a **full bounding box**
- Bounds may **overlap**
  - ➔ Primitives in single leaf nodes
  - ➔ More traversal steps as for KD Tree
  - ➔ Support for **dynamic scenes**

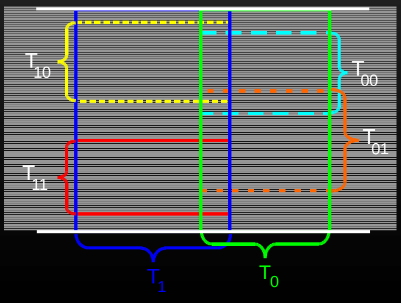


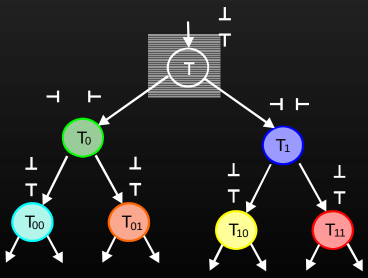




B-KD Tree Subdivision

- Bounding Volume Hierarchy (partially unbounded)
- Each node can be associated with a **full bounding box**
- Bounds may **overlap**
  - ➔ Primitives in single leaf nodes
  - ➔ More traversal steps as for KD Tree
  - ➔ Support for **dynamic scenes**

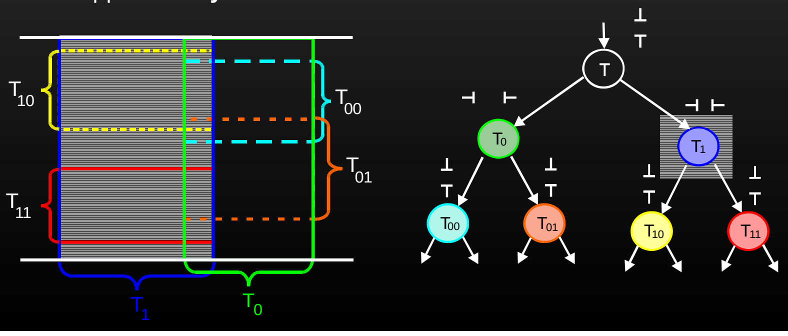




graphics hardware 06

B-KD Tree Subdivision

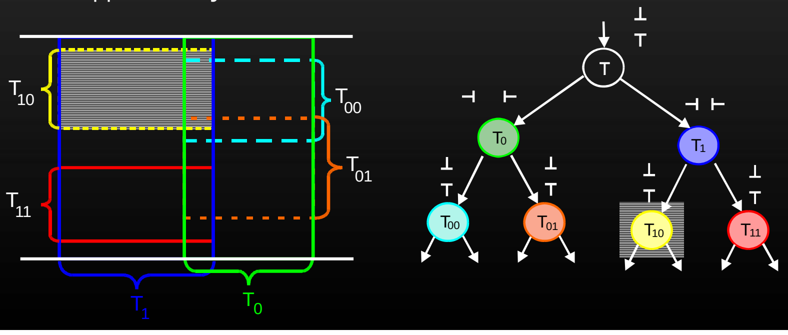
- Bounding Volume Hierarchy (partially unbounded)
- Each node can be associated with a **full bounding box**
- Bounds may **overlap**
  - ➔ Primitives in single leaf nodes
  - ➔ More traversal steps as for KD Tree
  - ➔ Support for **dynamic scenes**

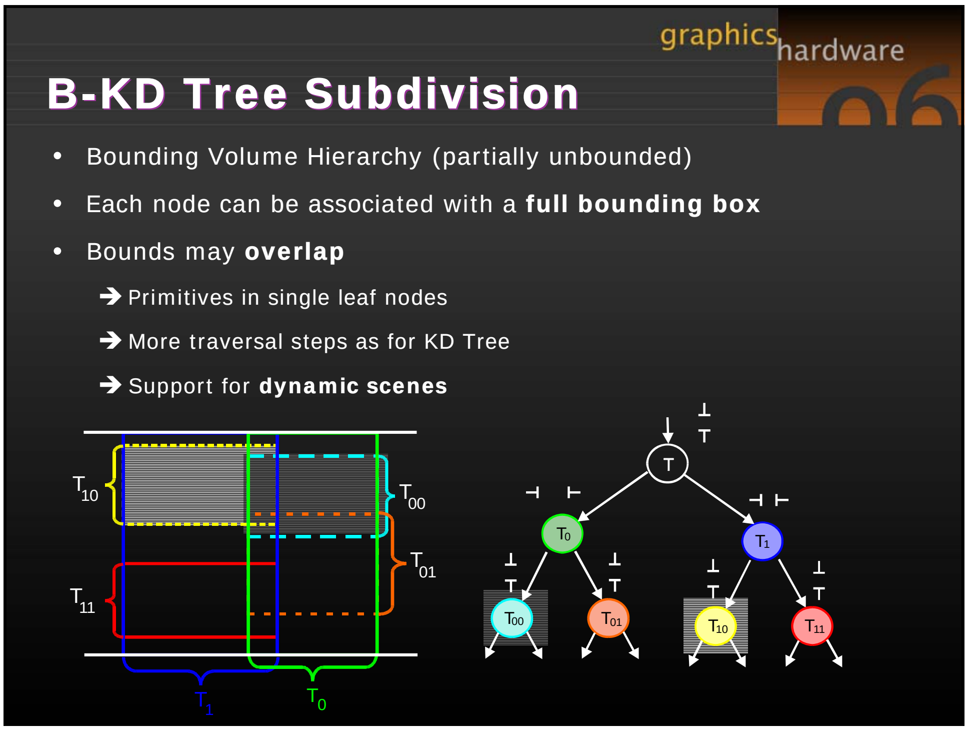


graphics hardware 06

B-KD Tree Subdivision

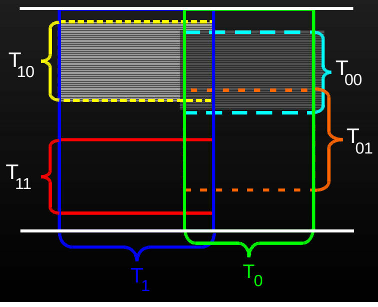
- Bounding Volume Hierarchy (partially unbounded)
- Each node can be associated with a **full bounding box**
- Bounds may **overlap**
  - ➔ Primitives in single leaf nodes
  - ➔ More traversal steps as for KD Tree
  - ➔ Support for **dynamic scenes**

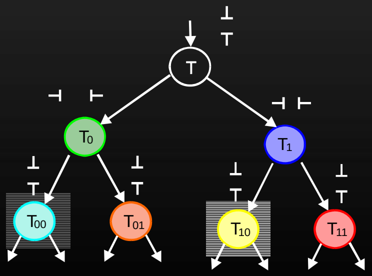


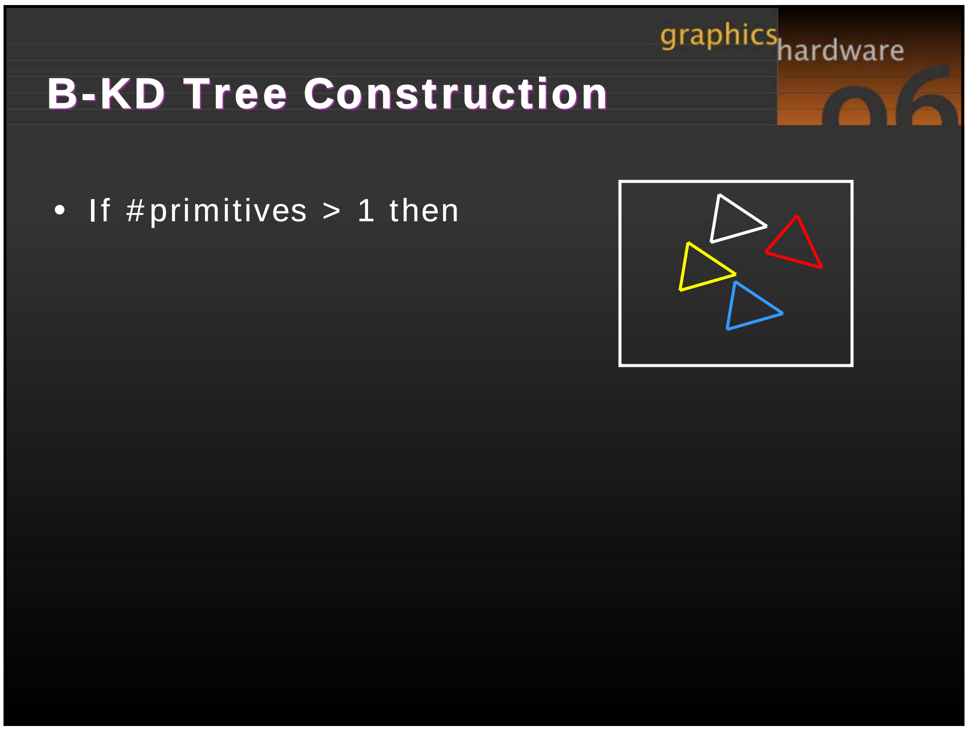


B-KD Tree Subdivision

- Bounding Volume Hierarchy (partially unbounded)
- Each node can be associated with a **full bounding box**
- Bounds may **overlap**
  - ➔ Primitives in single leaf nodes
  - ➔ More traversal steps as for KD Tree
  - ➔ Support for **dynamic scenes**

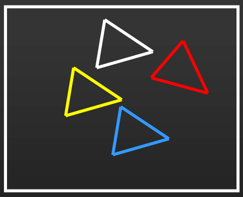


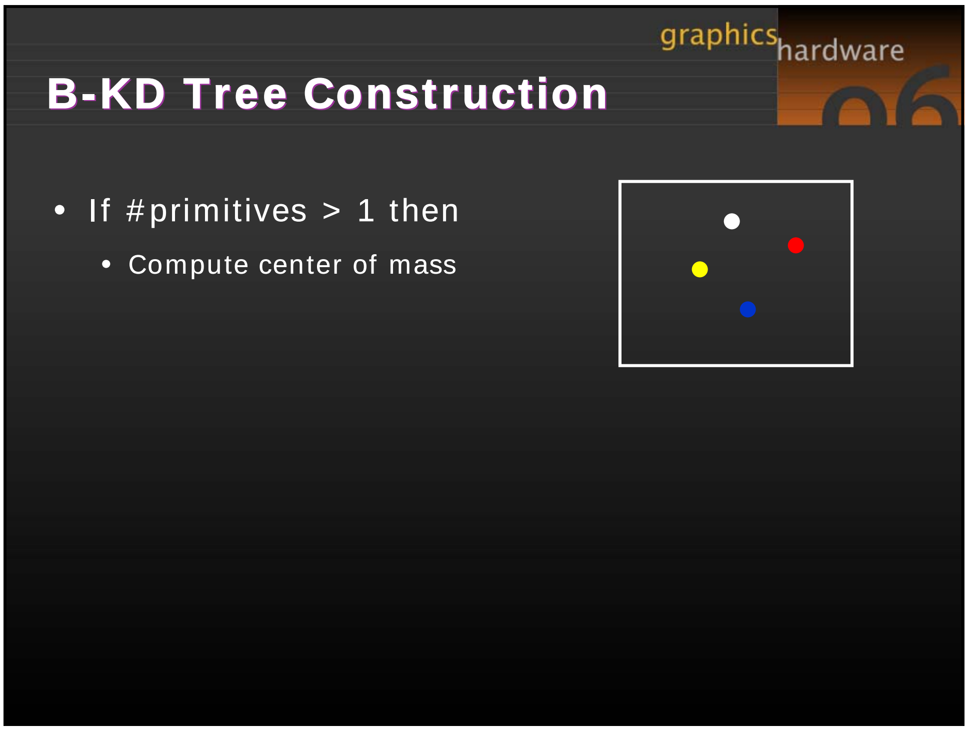




B-KD Tree Construction

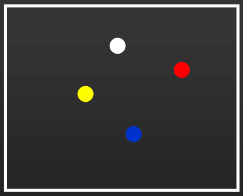
- If #primitives > 1 then

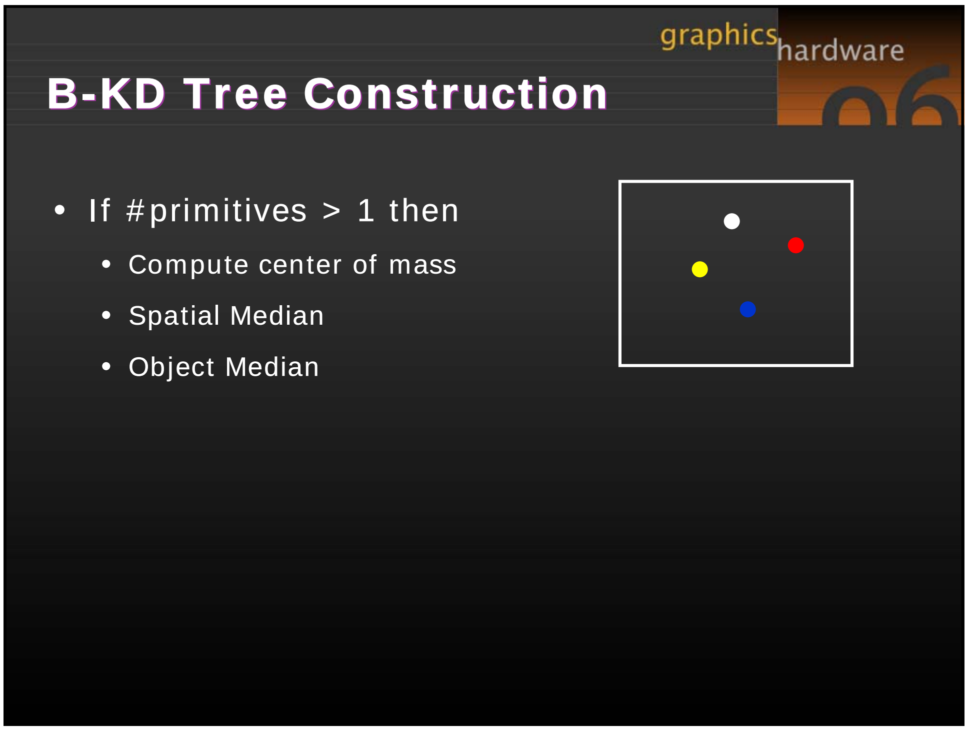




B-KD Tree Construction

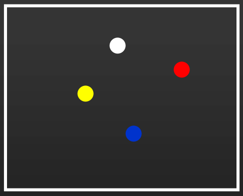
- If #primitives > 1 then
  - Compute center of mass

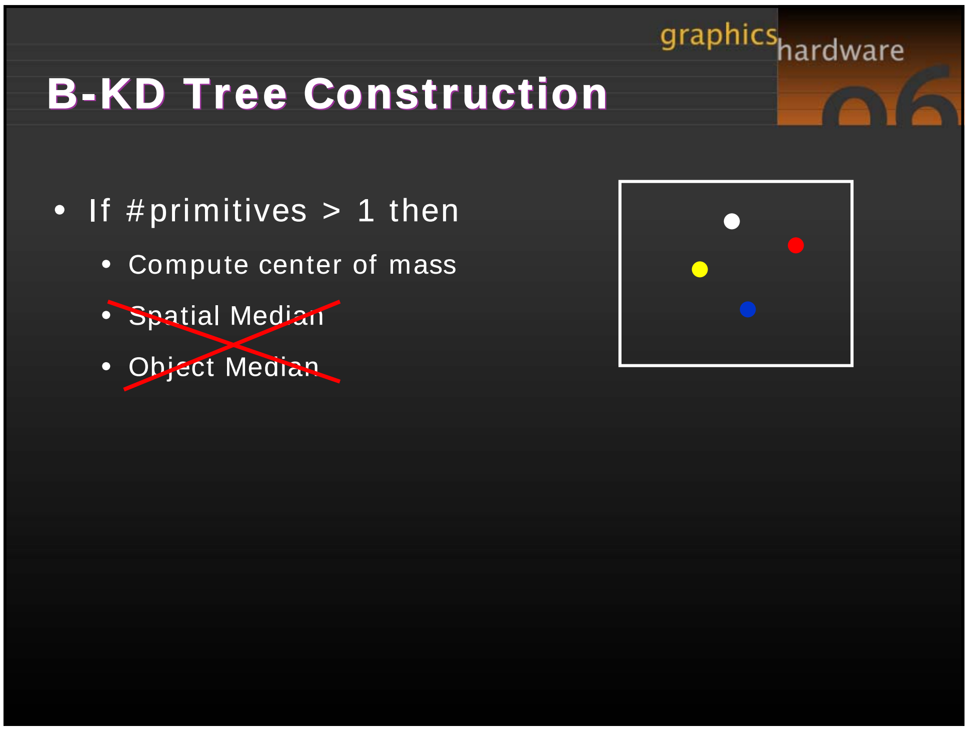




B-KD Tree Construction

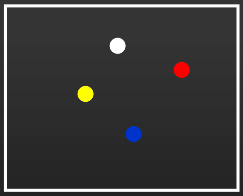
- If #primitives > 1 then
  - Compute center of mass
  - Spatial Median
  - Object Median

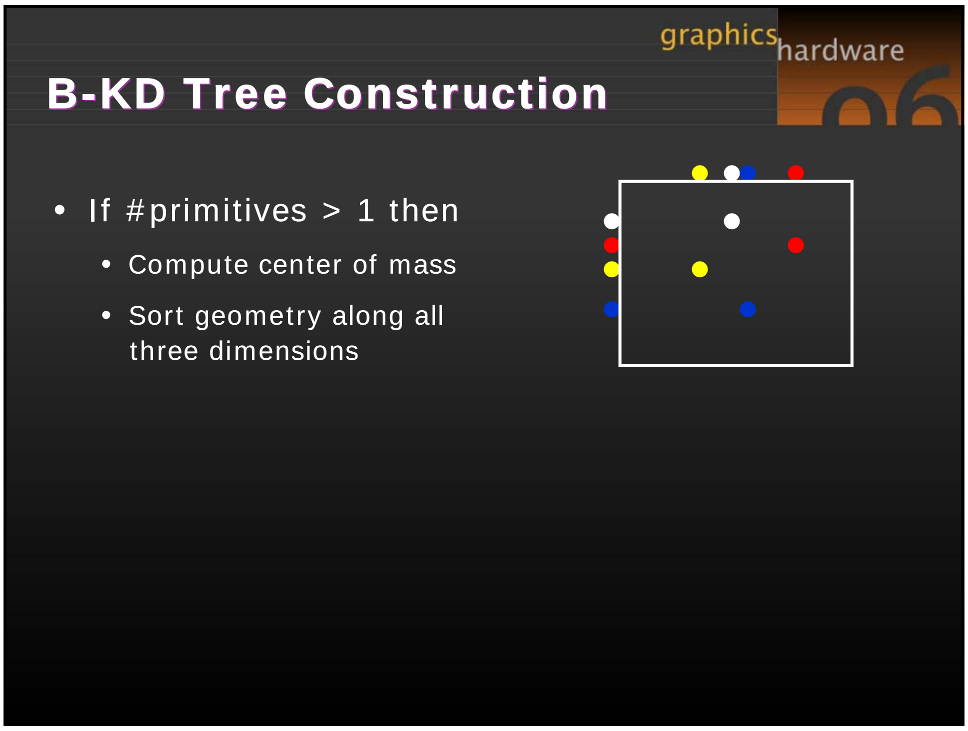




B-KD Tree Construction

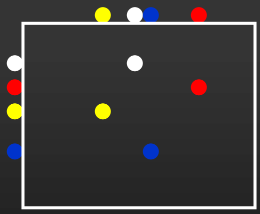
- If #primitives > 1 then
  - Compute center of mass
  - ~~Spatial Median~~
  - ~~Object Median~~

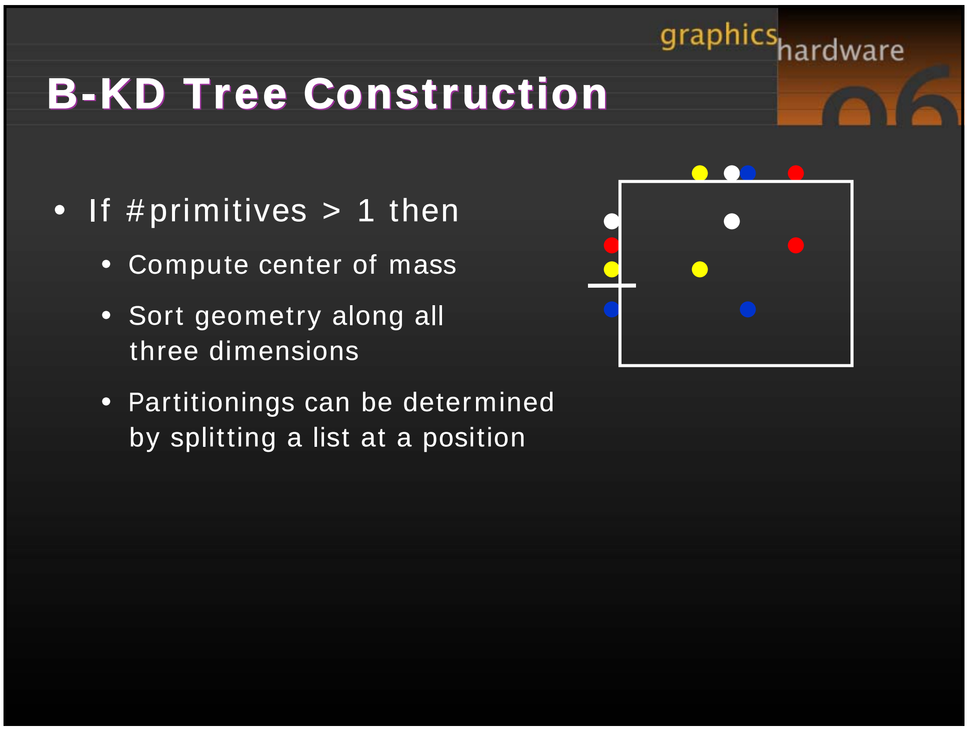




B-KD Tree Construction

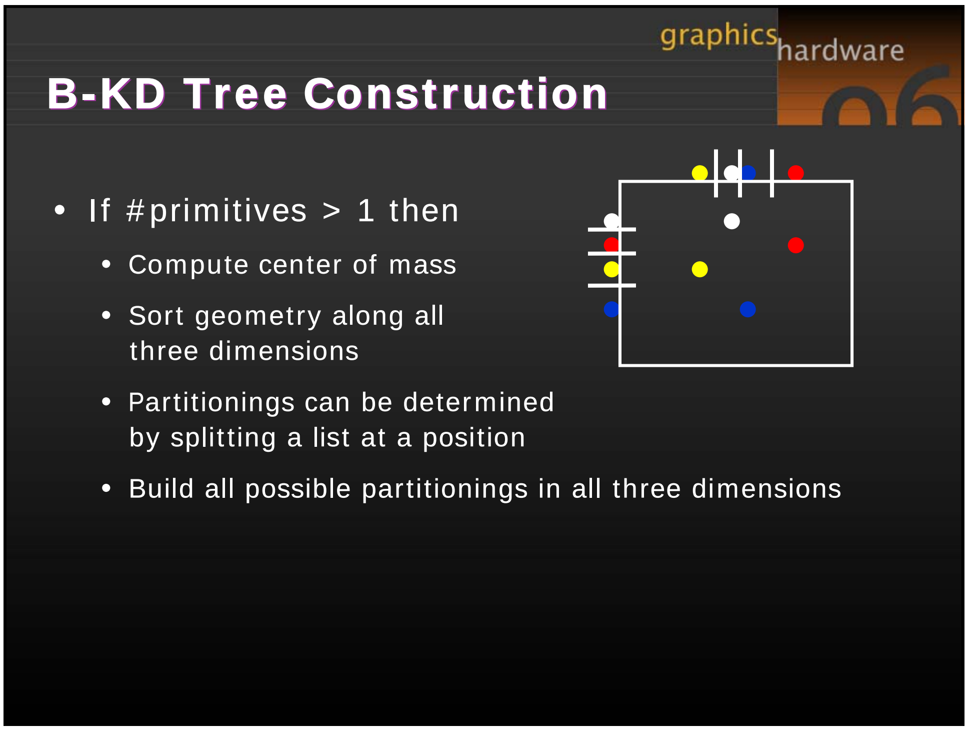
- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions





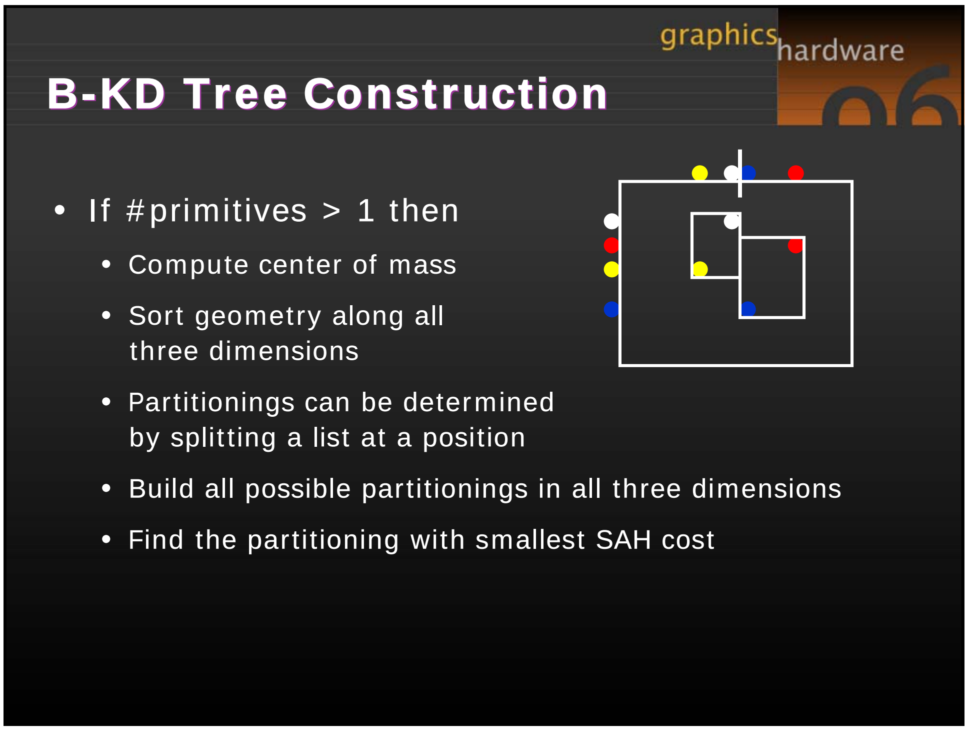
B-KD Tree Construction

- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions
  - Partitionings can be determined by splitting a list at a position



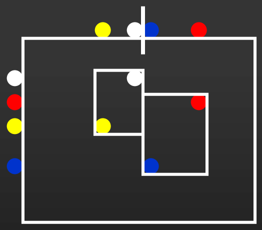
B-KD Tree Construction

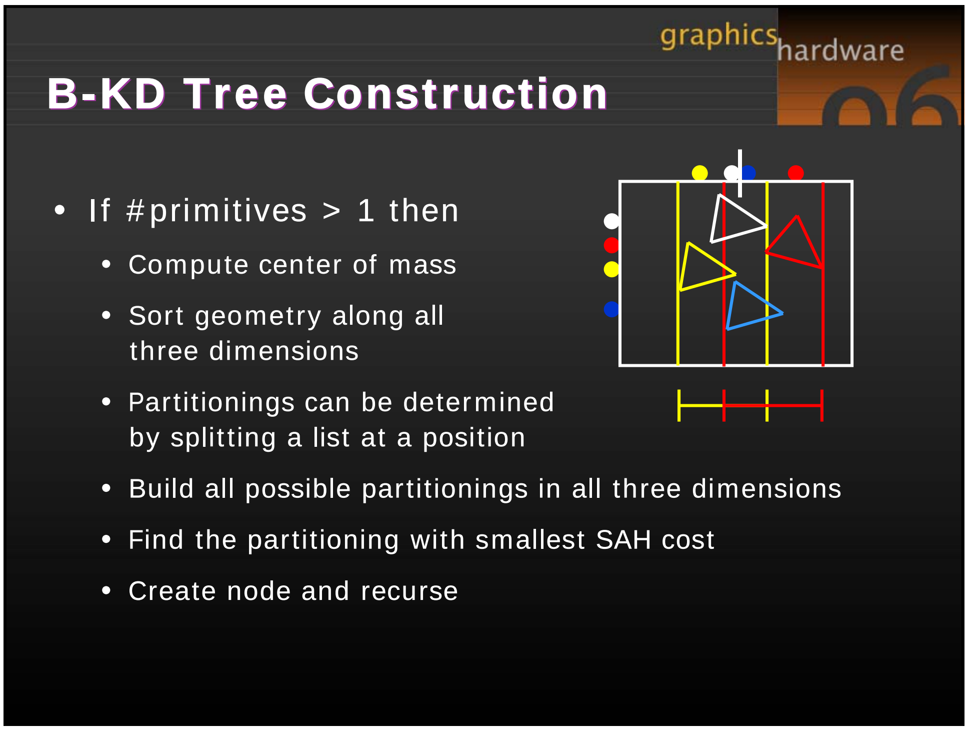
- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions
  - Partitionings can be determined by splitting a list at a position
  - Build all possible partitionings in all three dimensions



B-KD Tree Construction

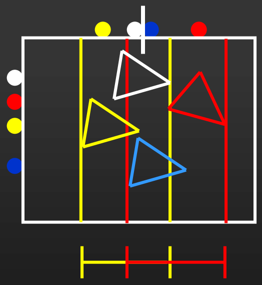
- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions
  - Partitionings can be determined by splitting a list at a position
  - Build all possible partitionings in all three dimensions
  - Find the partitioning with smallest SAH cost

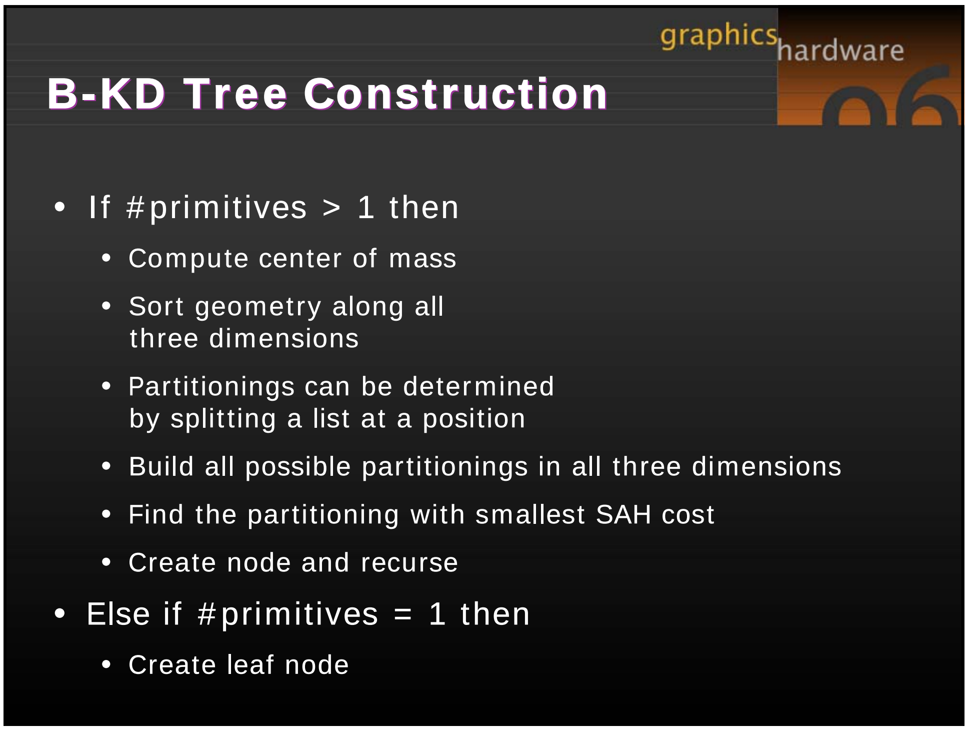




B-KD Tree Construction

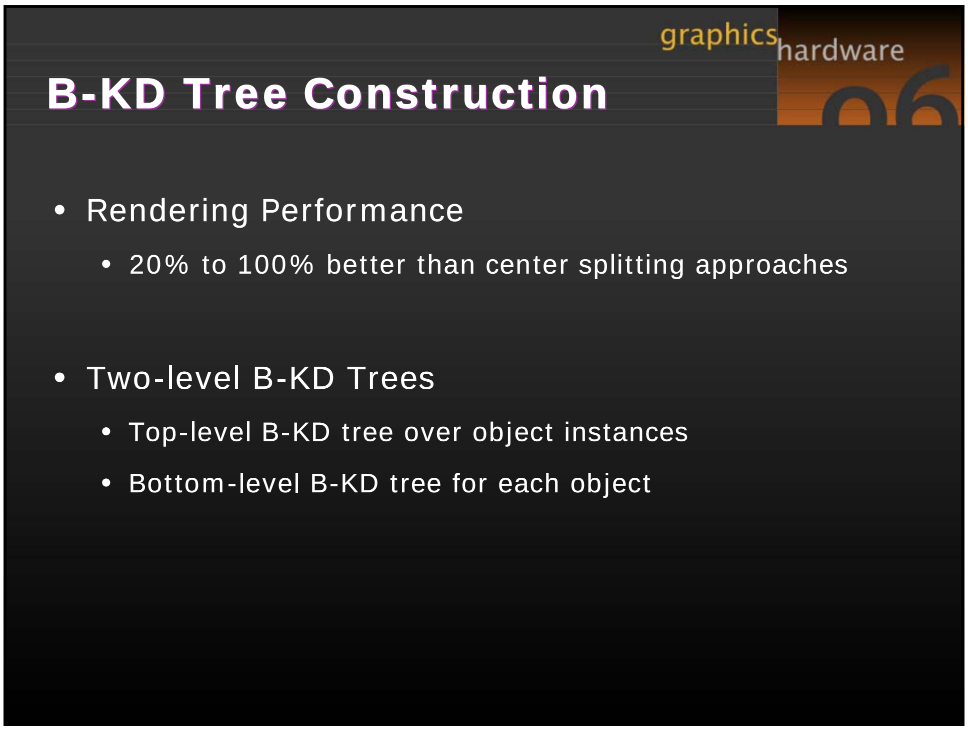
- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions
  - Partitionings can be determined by splitting a list at a position
  - Build all possible partitionings in all three dimensions
  - Find the partitioning with smallest SAH cost
  - Create node and recurse





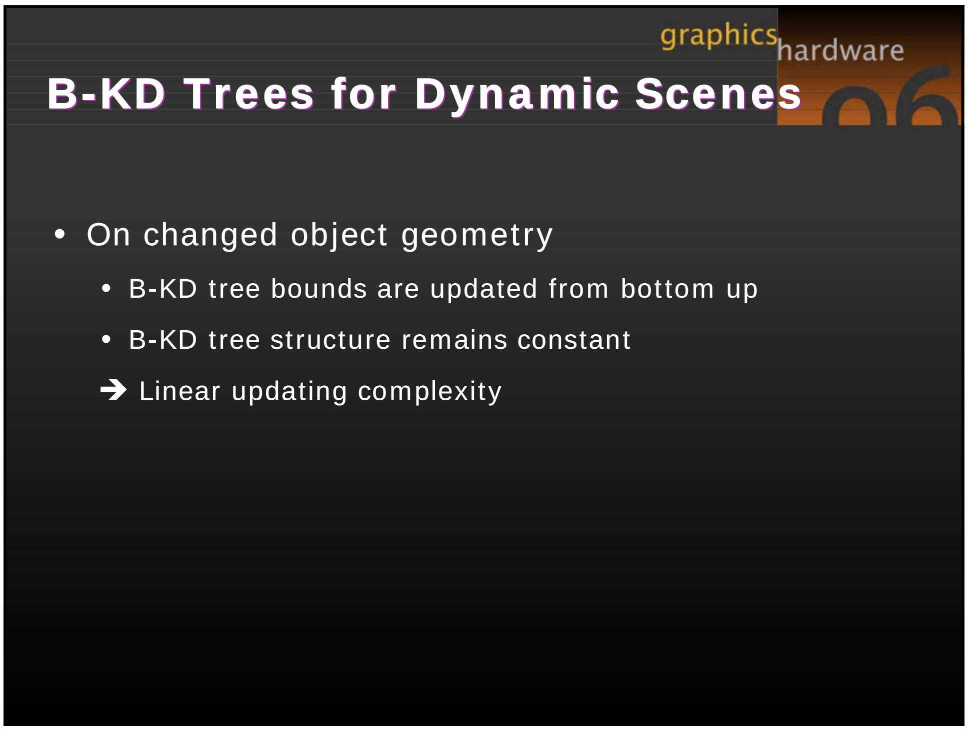
B-KD Tree Construction

- If #primitives > 1 then
  - Compute center of mass
  - Sort geometry along all three dimensions
  - Partitionings can be determined by splitting a list at a position
  - Build all possible partitionings in all three dimensions
  - Find the partitioning with smallest SAH cost
  - Create node and recurse
- Else if #primitives = 1 then
  - Create leaf node



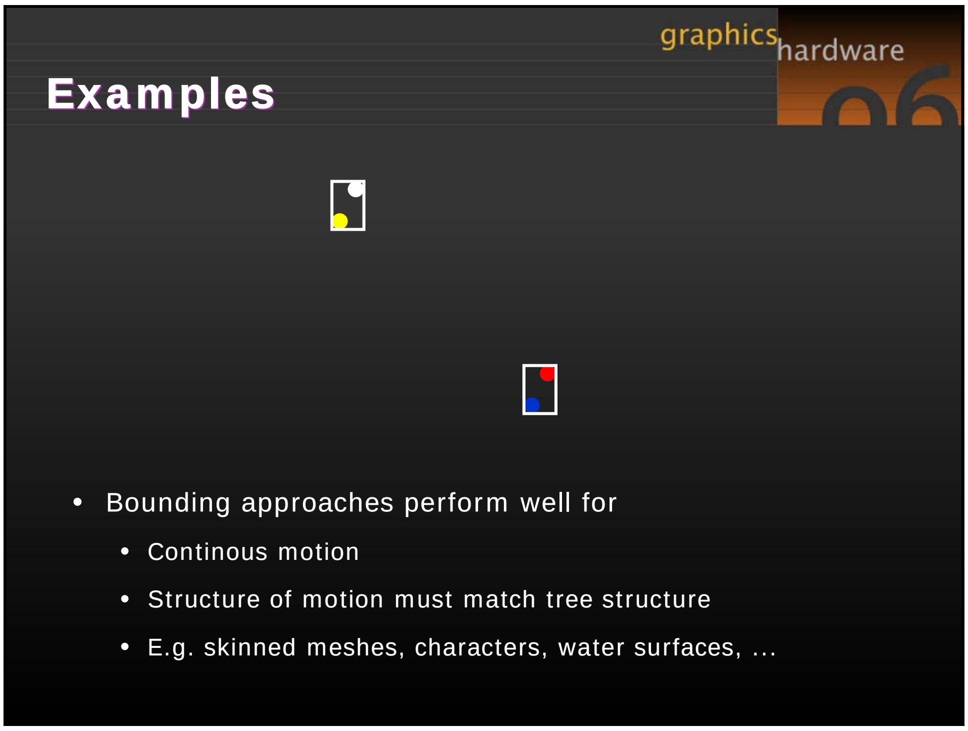
B-KD Tree Construction

- Rendering Performance
  - 20% to 100% better than center splitting approaches
- Two-level B-KD Trees
  - Top-level B-KD tree over object instances
  - Bottom-level B-KD tree for each object

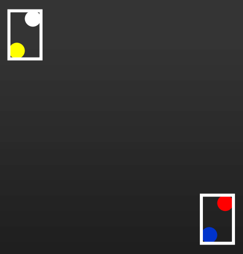


B-KD Trees for Dynamic Scenes

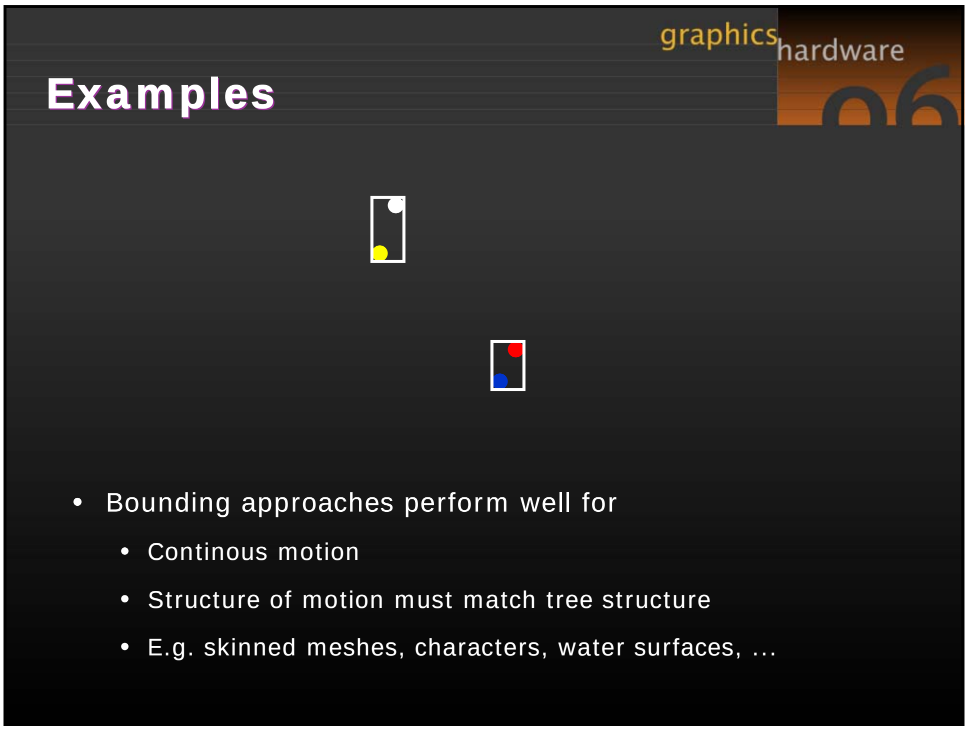
- On changed object geometry
  - B-KD tree bounds are updated from bottom up
  - B-KD tree structure remains constant
  - ➔ Linear updating complexity



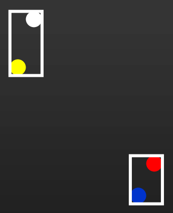
Examples



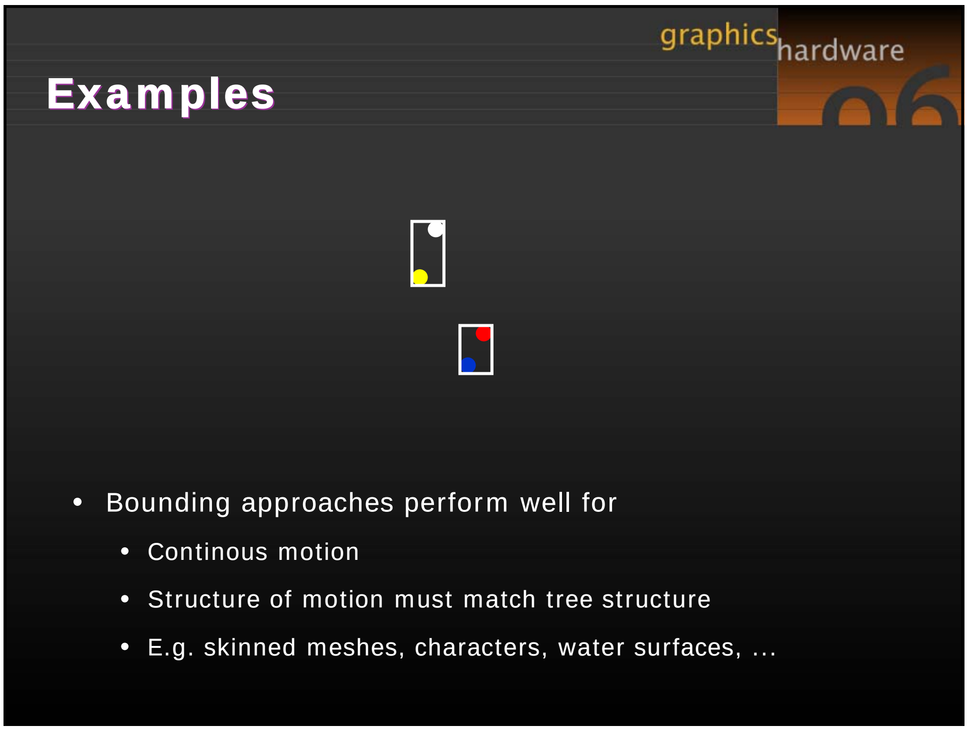
- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, …


graphics hardware 06

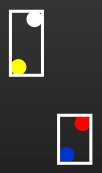
Examples



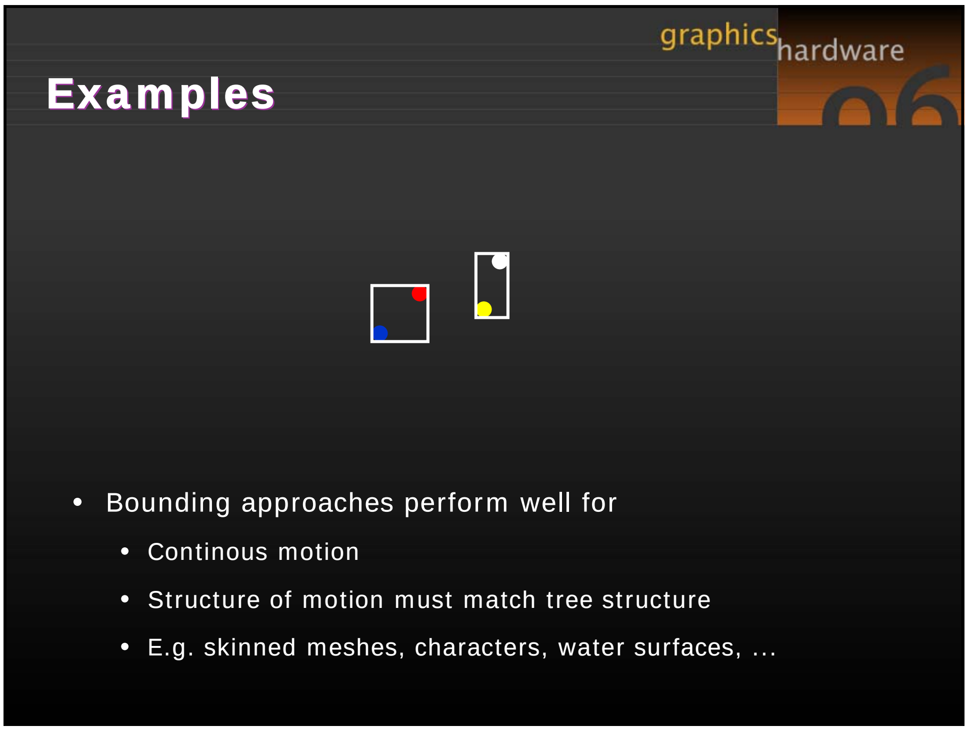
- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...


graphics hardware 06

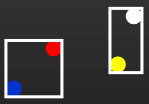
Examples



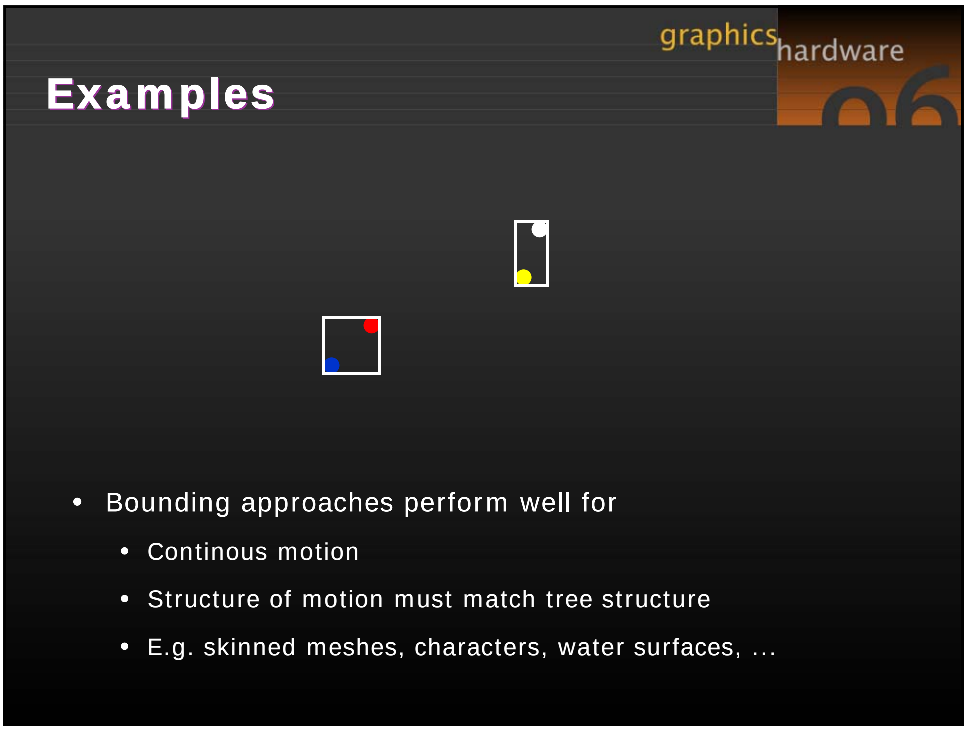
- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...



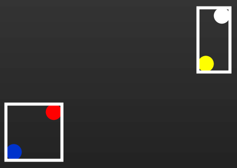
Examples



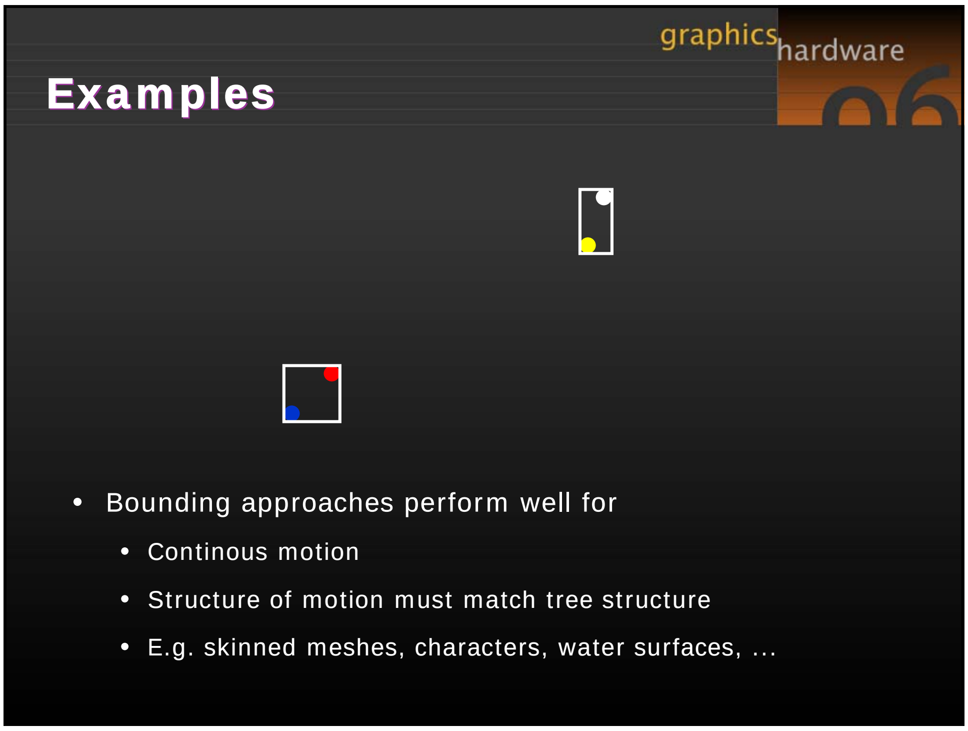
- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, …



Examples

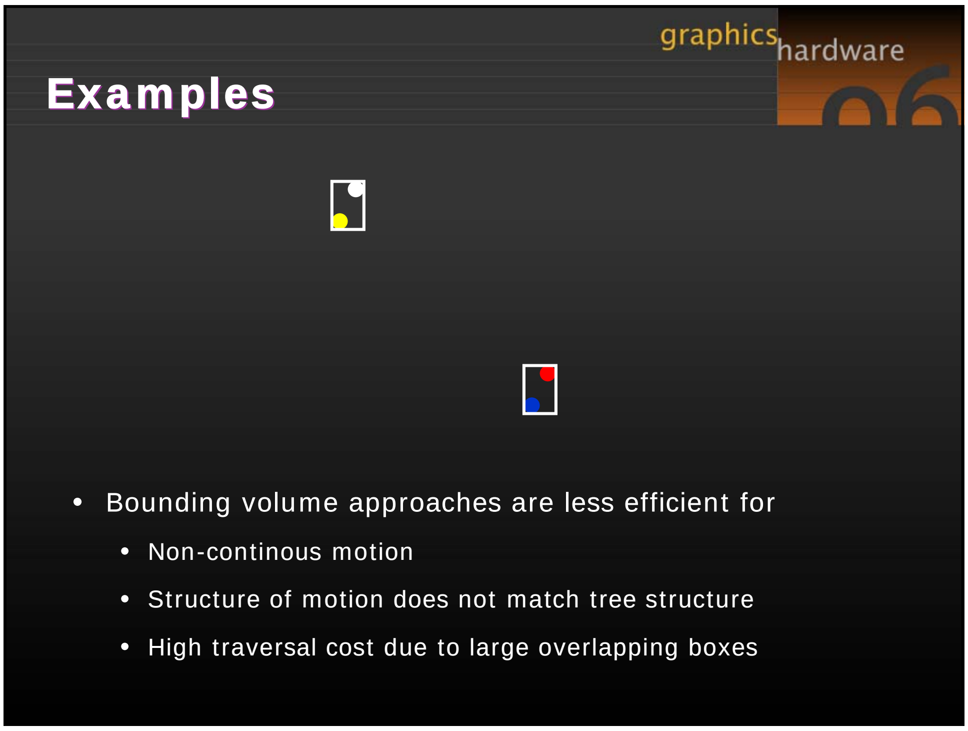


- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, …



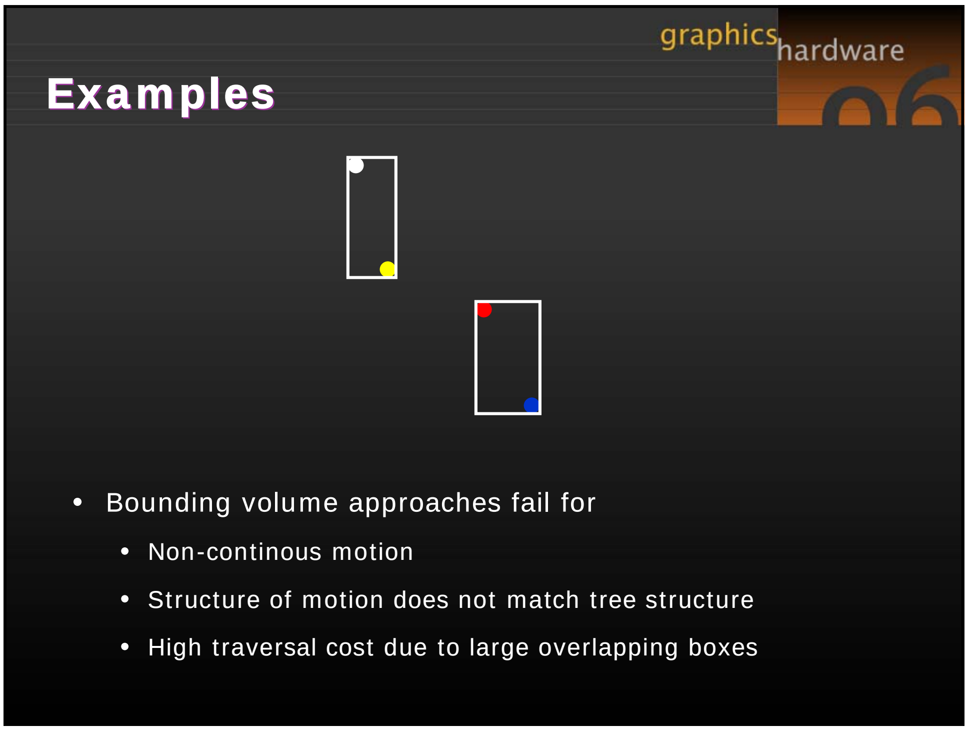
Examples

- Bounding approaches perform well for
  - Continous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...



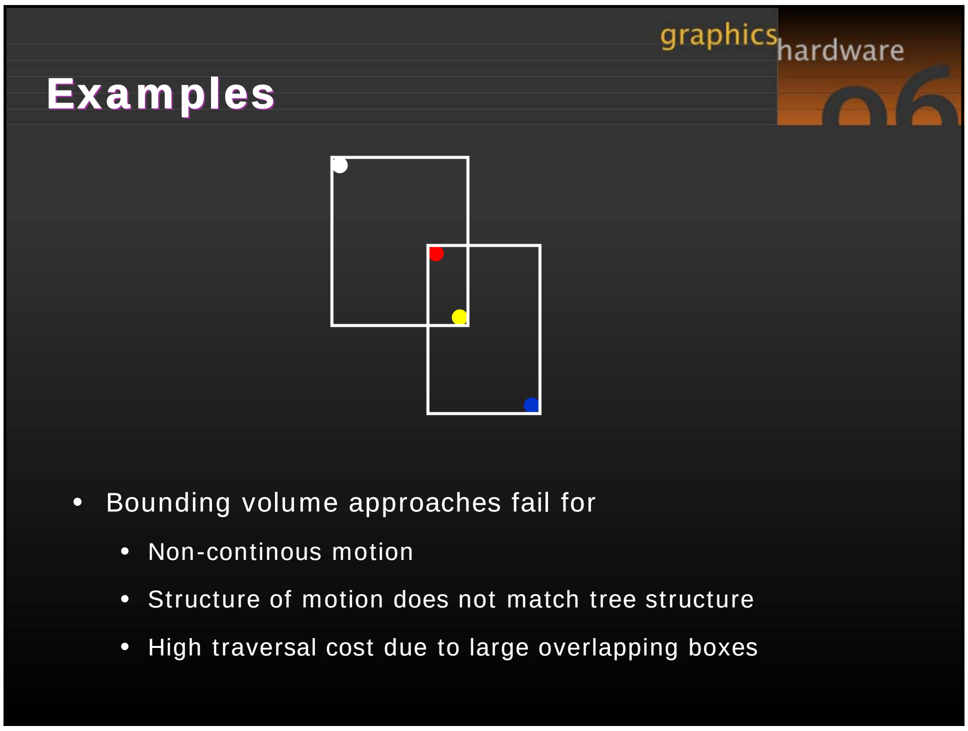
Examples

- Bounding volume approaches are less efficient for
  - Non-continous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes



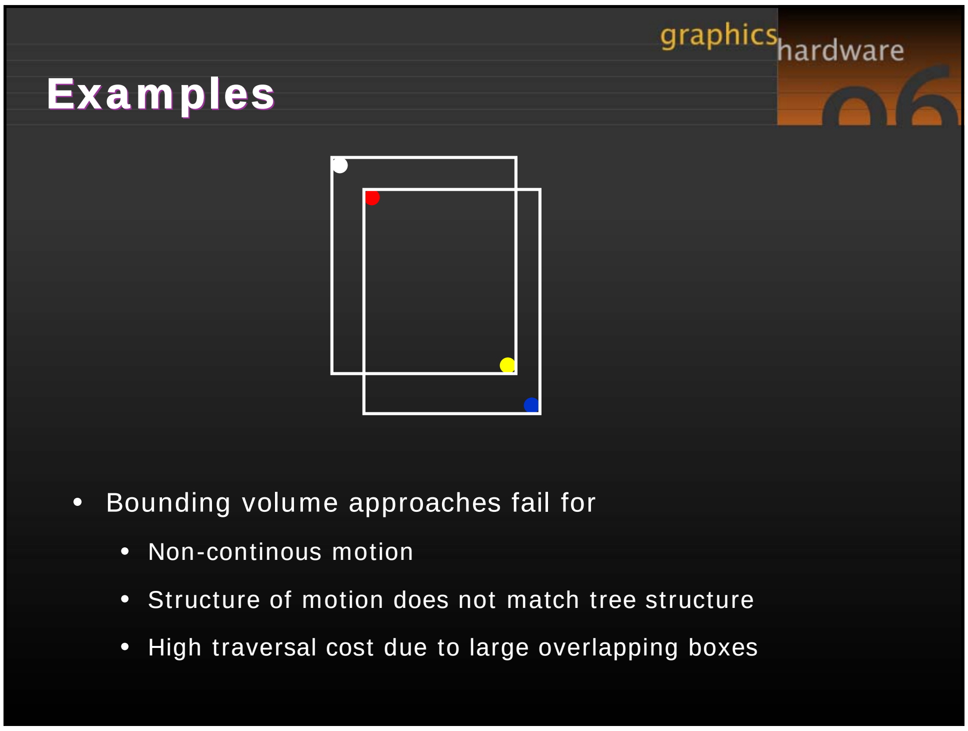
Examples

- Bounding volume approaches fail for
  - Non-continous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

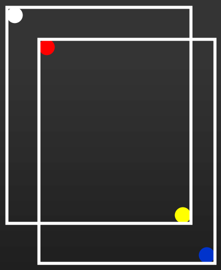


Examples

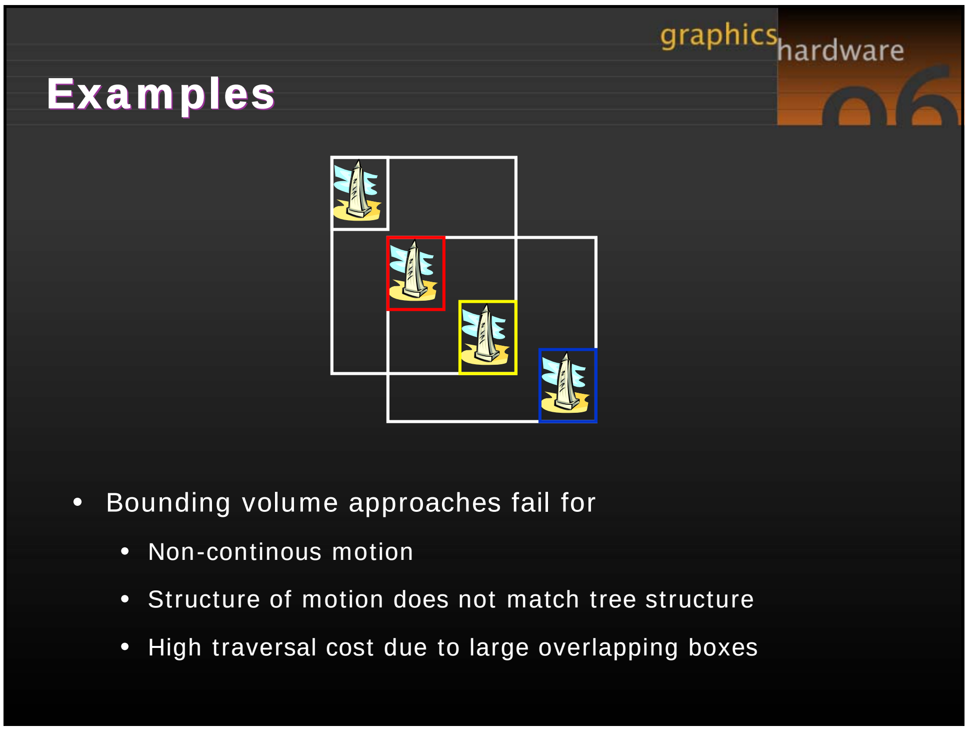
- Bounding volume approaches fail for
  - Non-continous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes



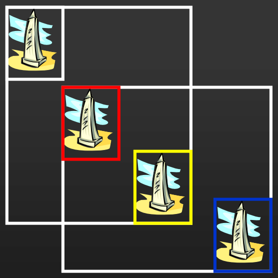
Examples



- Bounding volume approaches fail for
  - Non-continous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes



Examples

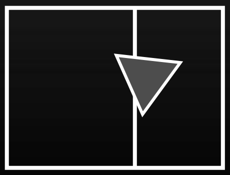


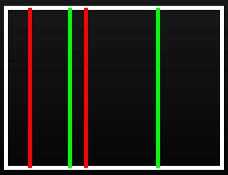
- Bounding volume approaches fail for
  - Non-continous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

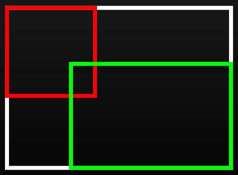
Comparison for Gael Scene

Index type	Index size	# trav-cost	# tri-ints
KD	1.4 MB	31	4.8
B-KD	1.1 MB	116	6.8
AABVH	2.2 MB	253	5.3

52k triangles

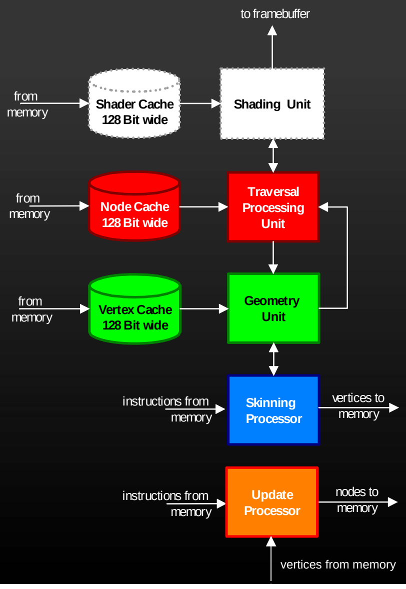

KD tree


B-KD tree


AABVH

DynRT Architecture

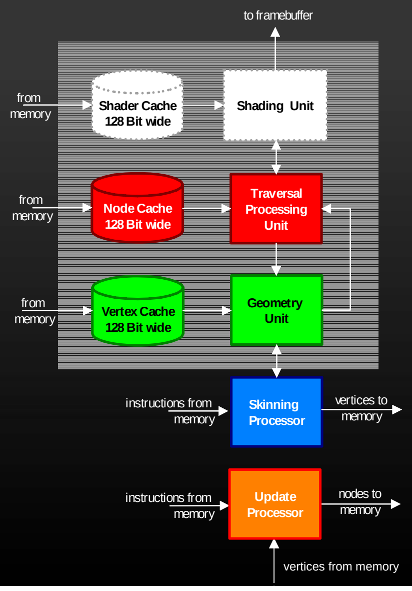
- Extension of RPU approach



graphicshardware 06

DynRT Architecture

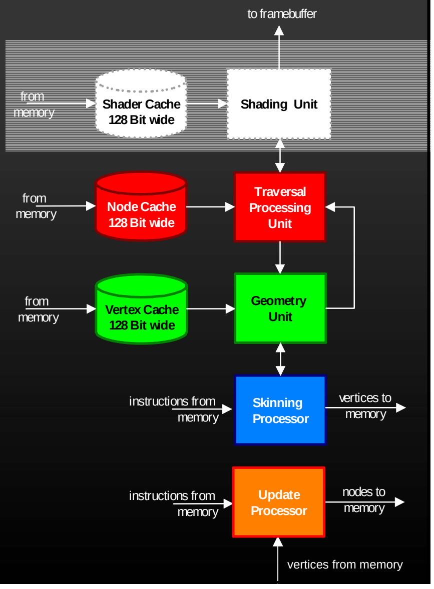
- Rendering Units
  - Highly multi-threaded
    - Higher hardware usage
  - Synchronous execution of packets of 4 rays
    - Memory bandwidth reduction
  - First level caches
    - Memory bandwidth reduction

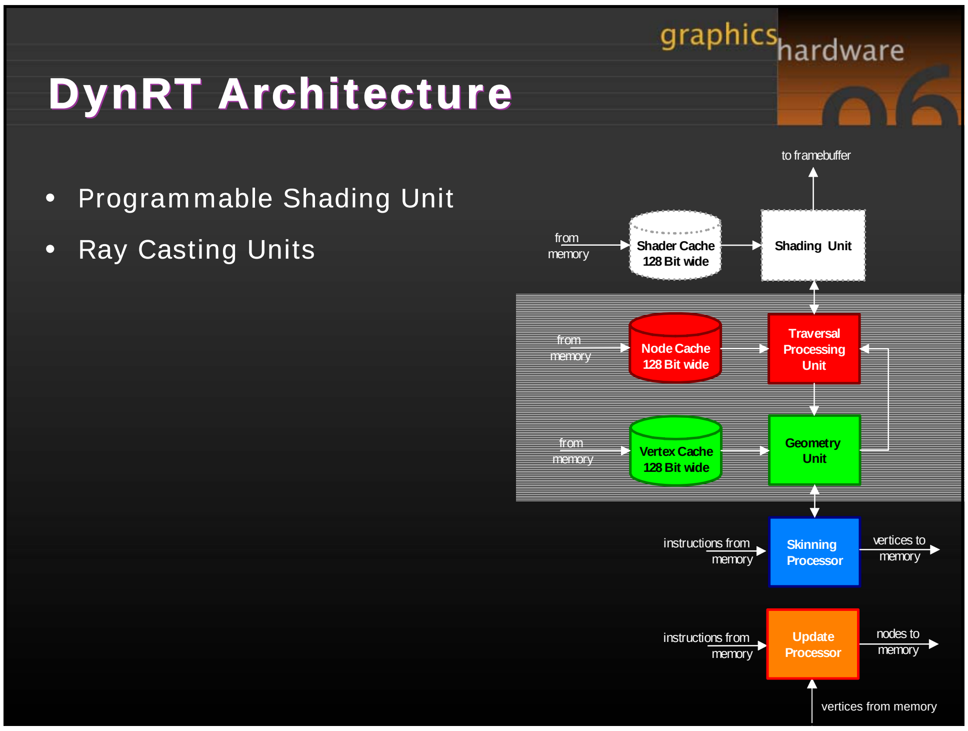


graphicshardware 06

DynRT Architecture

- Programmable Shading Unit
  - Similar to RPU shading processor
  - Ray generation tasks
  - Material shading
  - Calls Ray Casting Units to cast rays

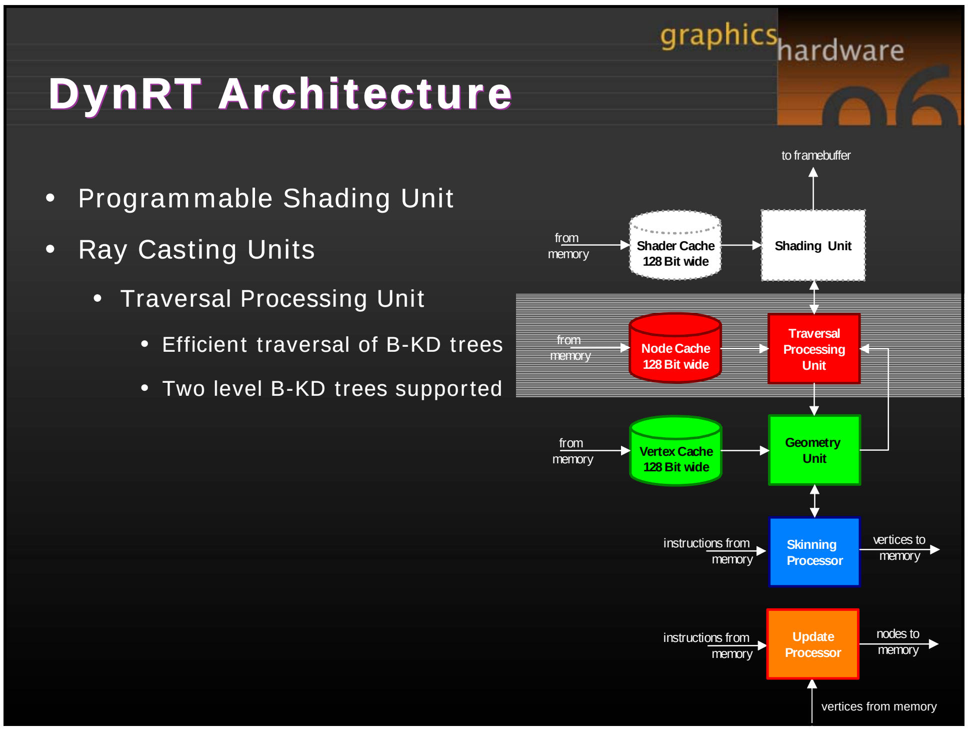




DynRT Architecture

- Programmable Shading Unit
- Ray Casting Units

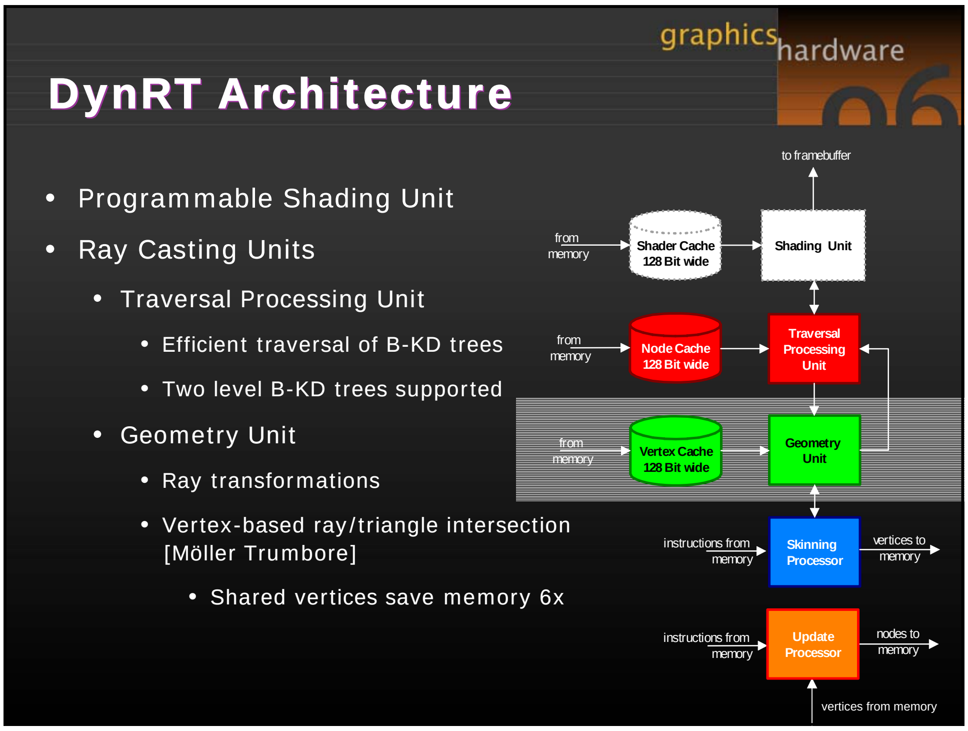
to framebuffer
from memory
Shader Cache 128 Bit wide
Shading Unit
from memory
Node Cache 128 Bit wide
Traversal Processing Unit
from memory
Vertex Cache 128 Bit wide
Geometry Unit
instructions from memory
Skinning Processor
vertices to memory
instructions from memory
Update Processor
nodes to memory
vertices from memory



DynRT Architecture

- Programmable Shading Unit
- Ray Casting Units
  - Traversal Processing Unit
    - Efficient traversal of B-KD trees
    - Two level B-KD trees supported

to framebuffer
from memory
Shader Cache 128 Bit wide
Shading Unit
from memory
Node Cache 128 Bit wide
Traversal Processing Unit
from memory
Vertex Cache 128 Bit wide
Geometry Unit
instructions from memory
Skinning Processor
vertices to memory
instructions from memory
Update Processor
nodes to memory
vertices from memory

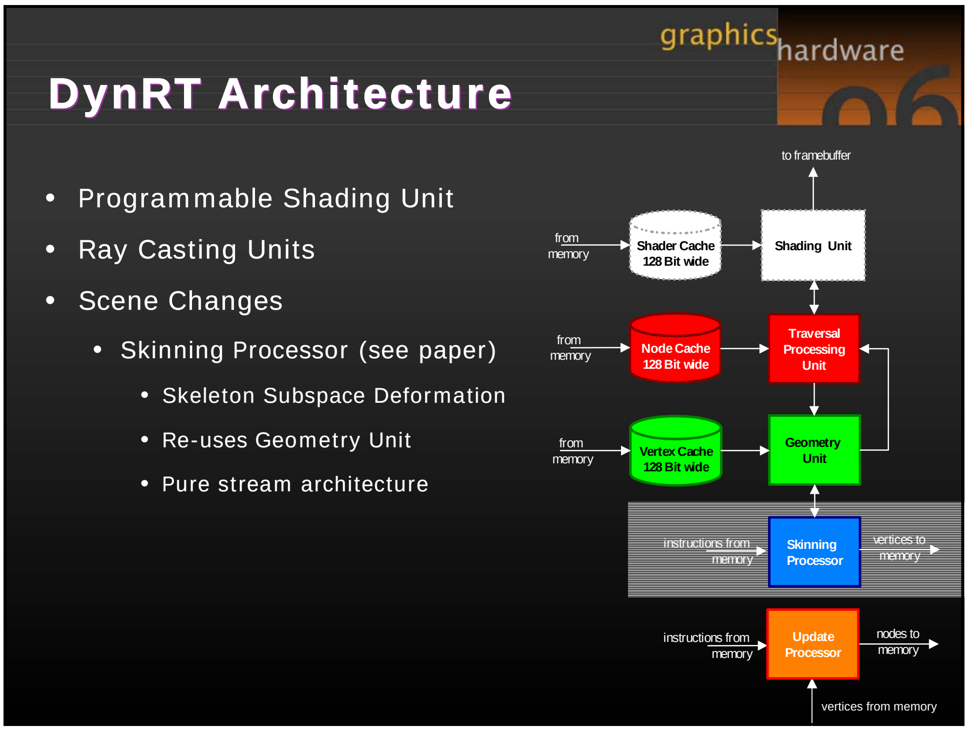


graphics hardware 06

DynRT Architecture

- Programmable Shading Unit
- Ray Casting Units
  - Traversal Processing Unit
    - Efficient traversal of B-KD trees
    - Two level B-KD trees supported
  - Geometry Unit
    - Ray transformations
    - Vertex-based ray/triangle intersection [Möller Trumbore]
      - Shared vertices save memory 6x

to framebuffer
from memory
Shader Cache 128 Bit wide
Shading Unit
from memory
Node Cache 128 Bit wide
Traversal Processing Unit
from memory
Vertex Cache 128 Bit wide
Geometry Unit
instructions from memory
Skinning Processor
vertices to memory
instructions from memory
Update Processor
nodes to memory
vertices from memory



graphics hardware 06

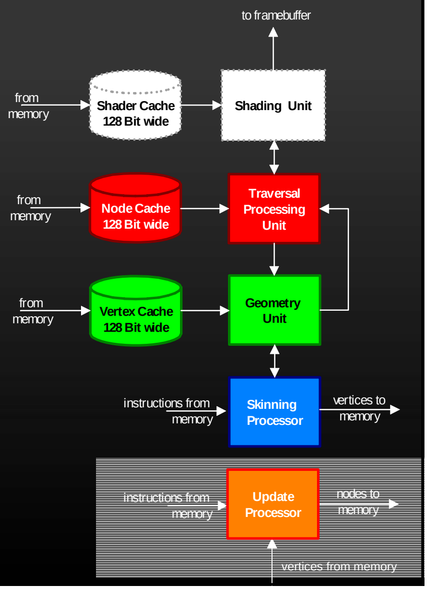
DynRT Architecture

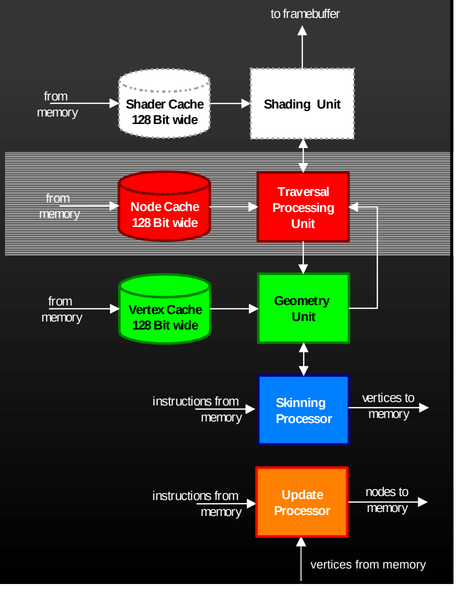
- Programmable Shading Unit
- Ray Casting Units
- Scene Changes
  - Skinning Processor (see paper)
    - Skeleton Subspace Deformation
    - Re-uses Geometry Unit
    - Pure stream architecture

to framebuffer
from memory
Shader Cache 128 Bit wide
Shading Unit
from memory
Node Cache 128 Bit wide
Traversal Processing Unit
from memory
Vertex Cache 128 Bit wide
Geometry Unit
instructions from memory
Skinning Processor
vertices to memory
instructions from memory
Update Processor
nodes to memory
vertices from memory

DynRT Architecture

- Programmable Shading Unit
- Ray Casting Units
- Scene Changes
  - Skinning Processor (see paper)
    - Skeleton Subspace Deformation
    - Re-uses Geometry Unit
    - Pure stream architecture
  - Update Processor
    - Stream-like architecture
    - Partial breadth-first execution
    - One B-KD node update per clock cycle peak





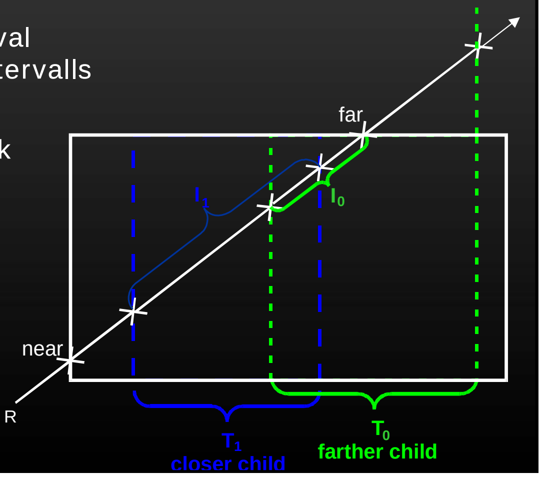
Traversal of B-KD Trees

Traversal of B-KD Trees

- Early ray termination
- Clipping of near/far interval against both bounding intervalls
- Take closer child, push farther child to stack
- Traversal order does not affect correctness

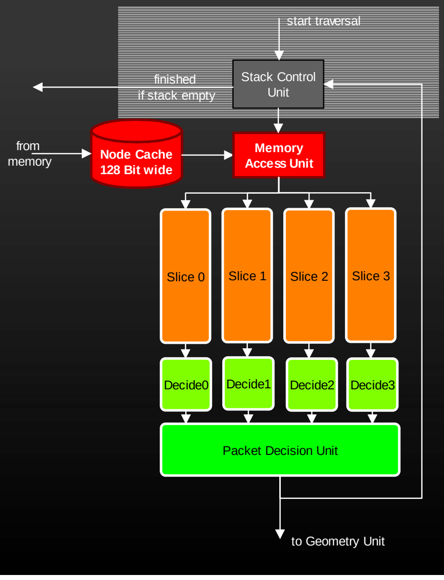
Complexity

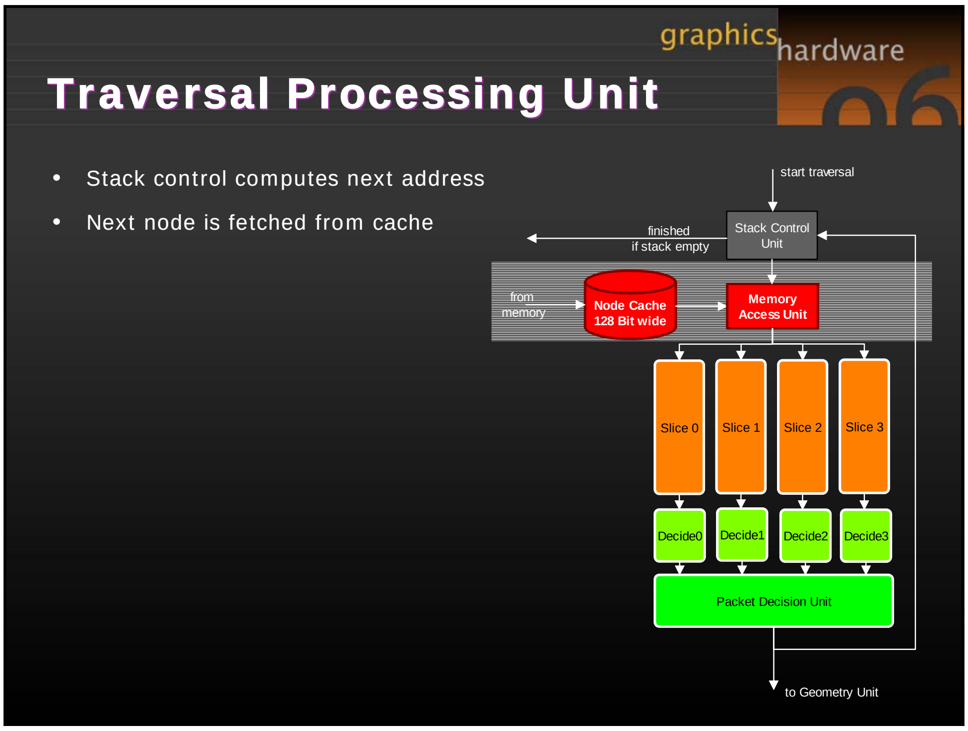
- 4x computational cost of KD tree traversal step
- 2x stack memory



Traversal Processing Unit

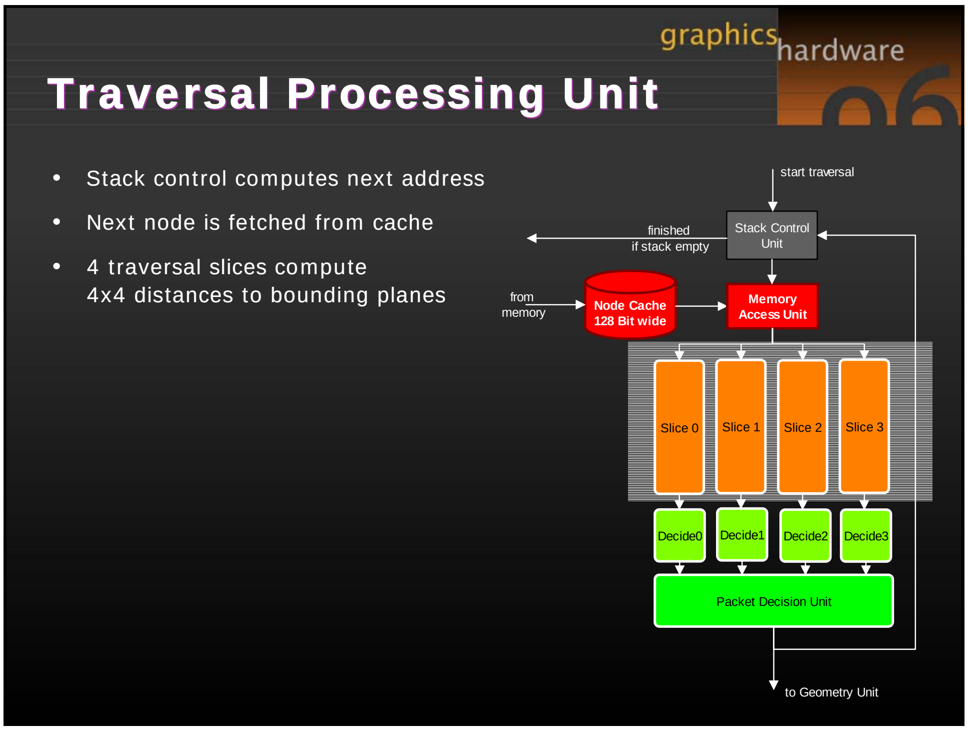
- Stack control computes next address





Traversal Processing Unit

- Stack control computes next address
- Next node is fetched from cache

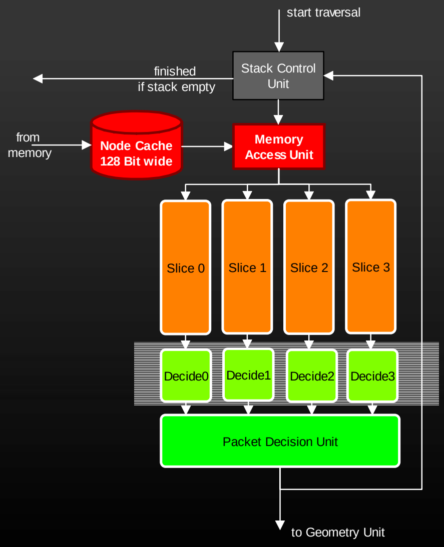


Traversal Processing Unit

- Stack control computes next address
- Next node is fetched from cache
- 4 traversal slices compute 4x4 distances to bounding planes

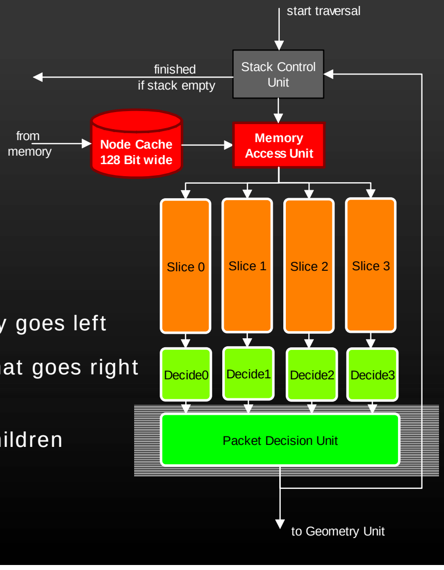
Traversal Processing Unit

- Stack control computes next address
- Next node is fetched from cache
- 4 traversal slices compute
  4x4 distances to bounding planes
- 4 Decision Units compute
  per ray traversal decision



Traversal Processing Unit

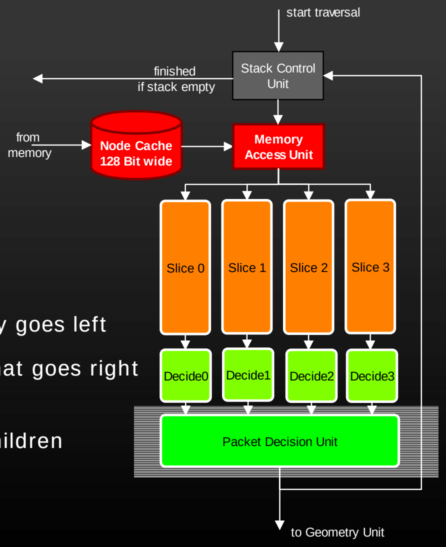
- Stack control computes next address
- Next node is fetched from cache
- 4 traversal slices compute
  4x4 distances to bounding planes
- 4 Decision Units compute
  per ray traversal decision
- Packet Decision Unit computes
  packet traversal decision
  - Packet goes left if exists a that ray goes left
  - Packet goes right if exists a ray that goes right
  - Packet goes from left to right if
    exists a ray that goes into both children
    from left to right

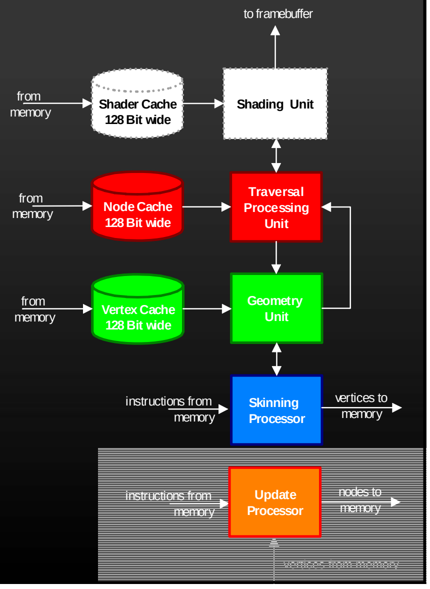


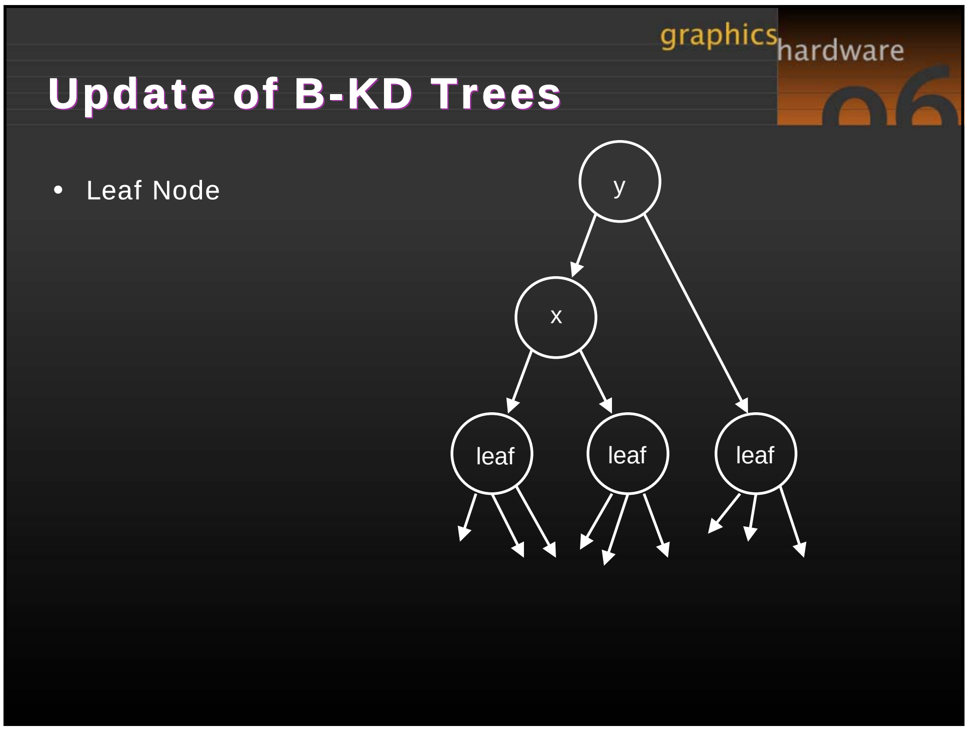
Traversal Processing Unit

- Stack control computes next address
- Next node is fetched from cache
- 4 traversal slices compute
  4x4 distances to bounding planes
- 4 Decision Units compute
  per ray traversal decision
- Packet Decision Unit computes
  packet traversal decision
  - Packet goes left if exists a that ray goes left
  - Packet goes right if exists a ray that goes right
  - Packet goes from left to right if
    exists a ray that goes into both children
    from left to right

  ➔ Incoherent packets possible

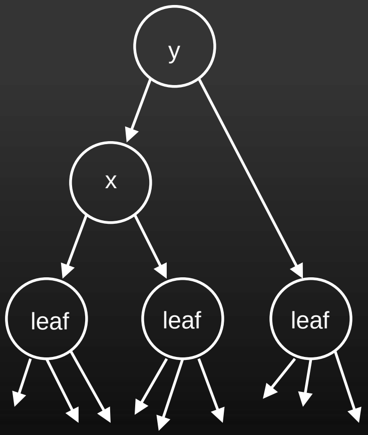


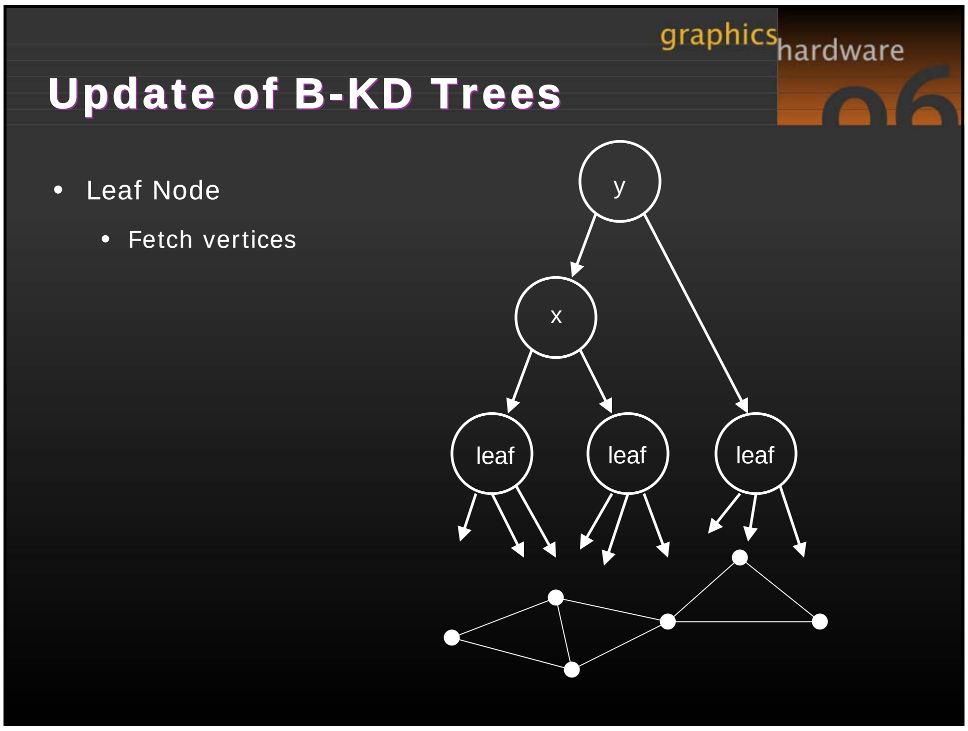




Update of B-KD Trees

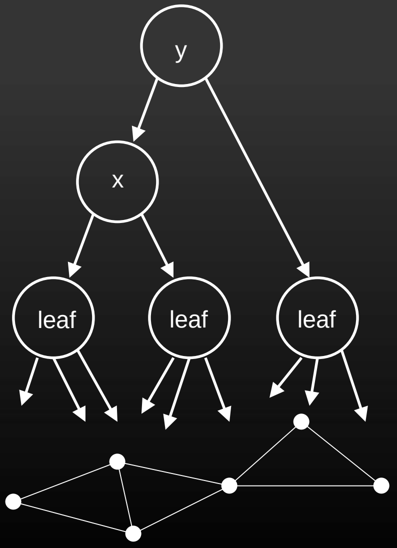
- Leaf Node

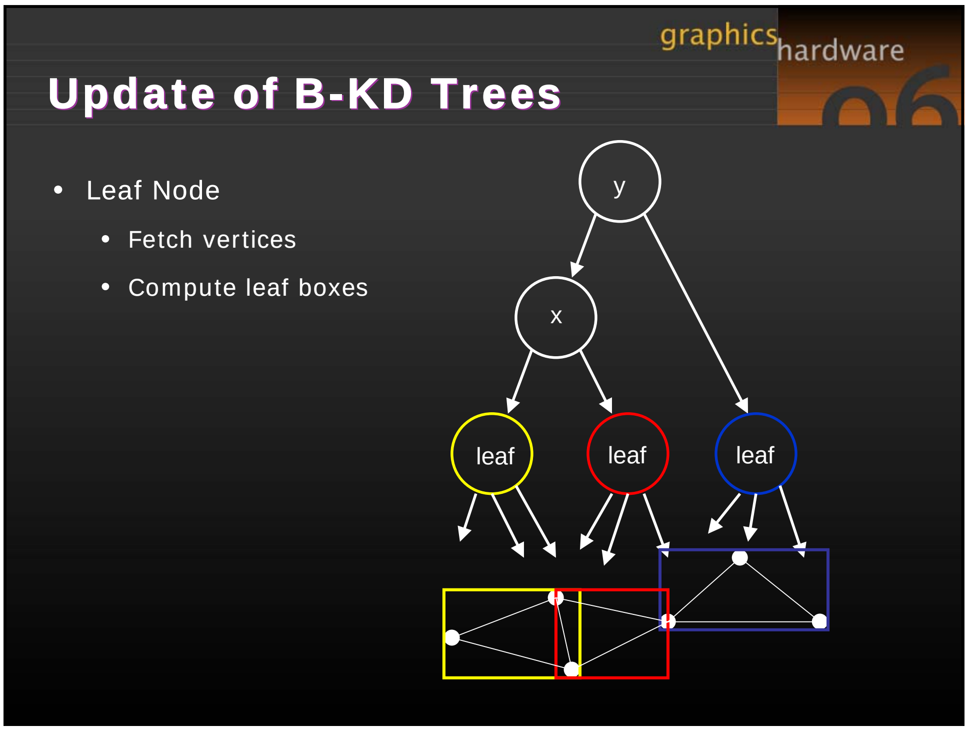




Update of B-KD Trees

- Leaf Node
  - Fetch vertices

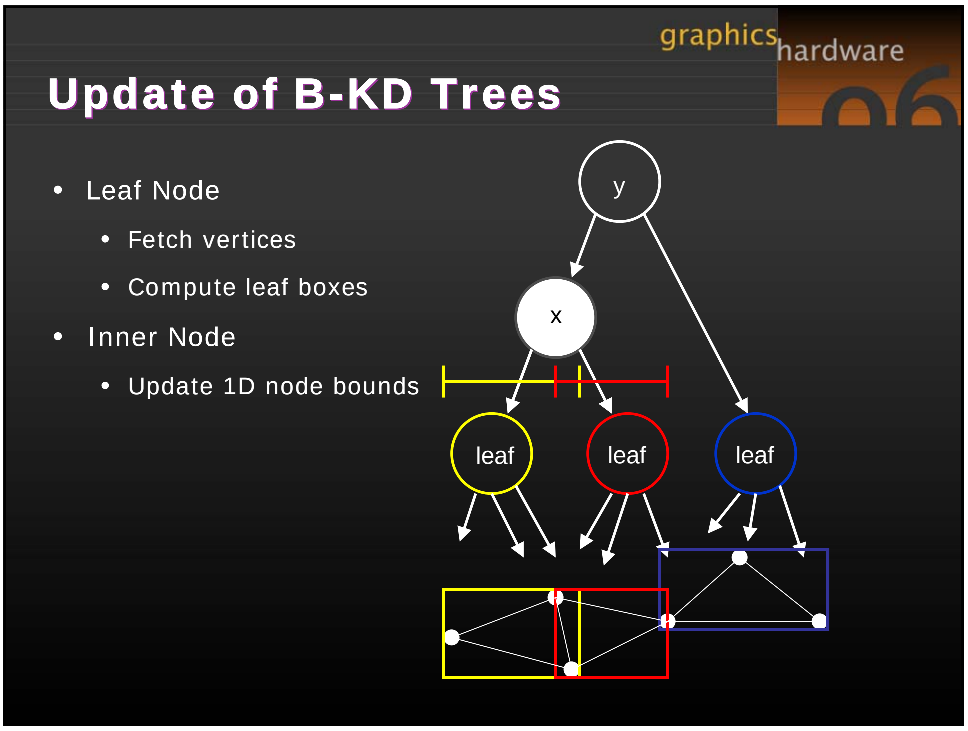




Update of B-KD Trees

- Leaf Node
  - Fetch vertices
  - Compute leaf boxes

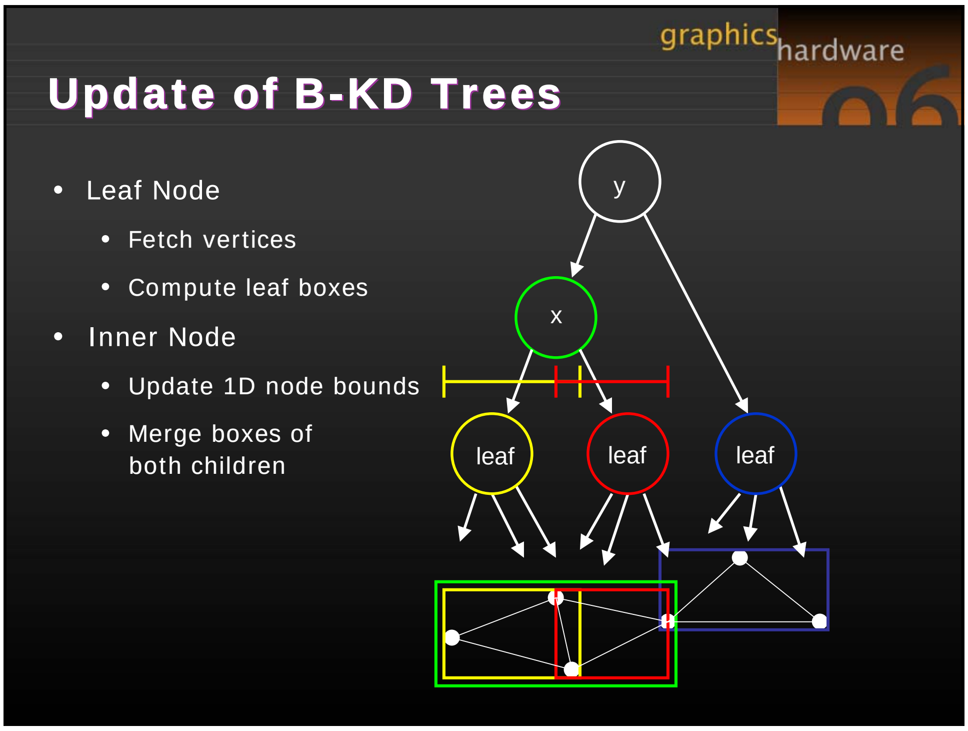
graphics hardware 06
y
x
leaf
leaf
leaf



Update of B-KD Trees

- Leaf Node
  - Fetch vertices
  - Compute leaf boxes
- Inner Node
  - Update 1D node bounds

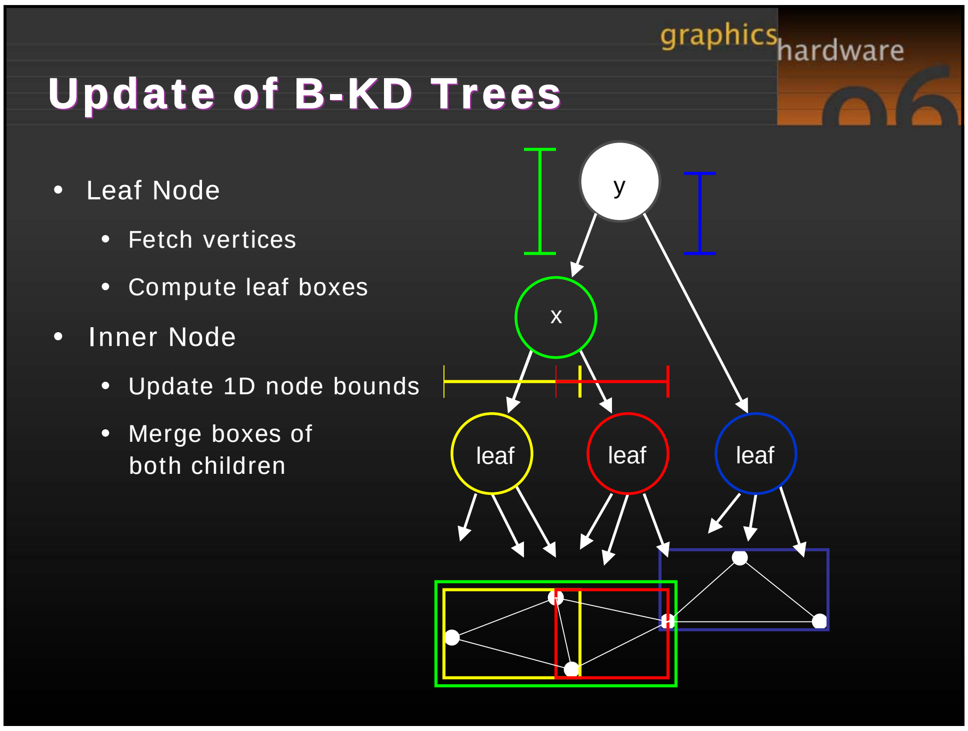
graphics hardware 06
y
x
leaf
leaf
leaf



Update of B-KD Trees

- Leaf Node
  - Fetch vertices
  - Compute leaf boxes
- Inner Node
  - Update 1D node bounds
  - Merge boxes of both children

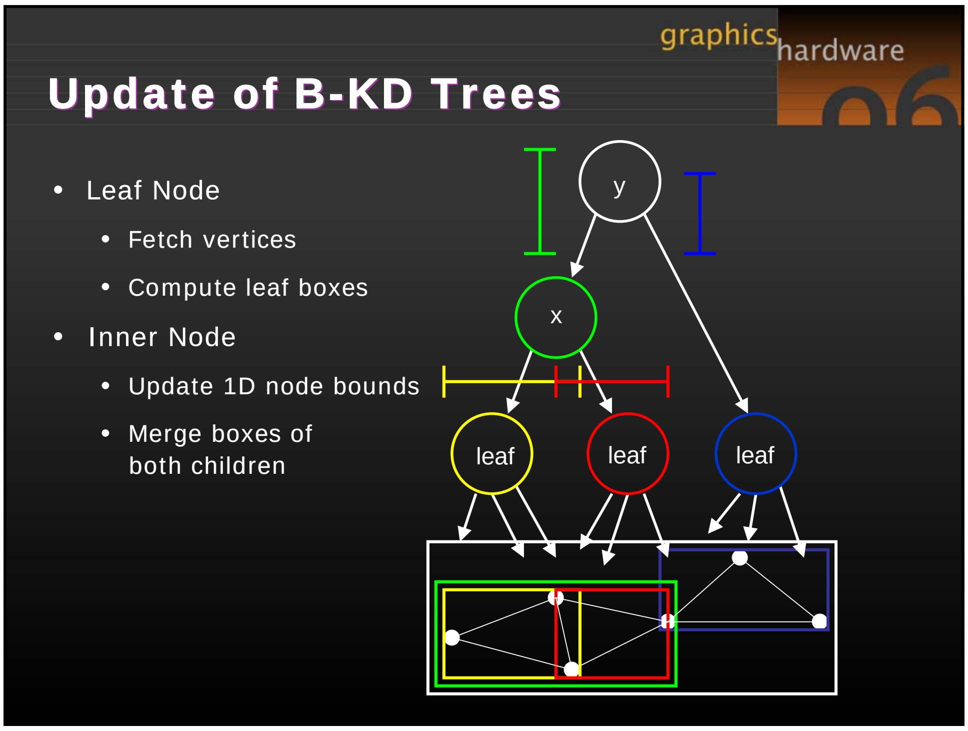
y
x
leaf
leaf
leaf



Update of B-KD Trees

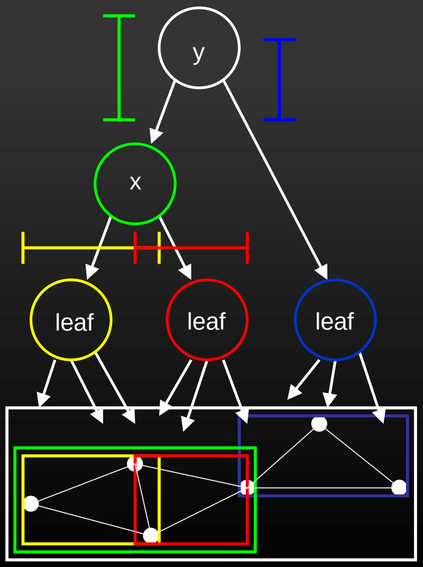
- Leaf Node
  - Fetch vertices
  - Compute leaf boxes
- Inner Node
  - Update 1D node bounds
  - Merge boxes of both children

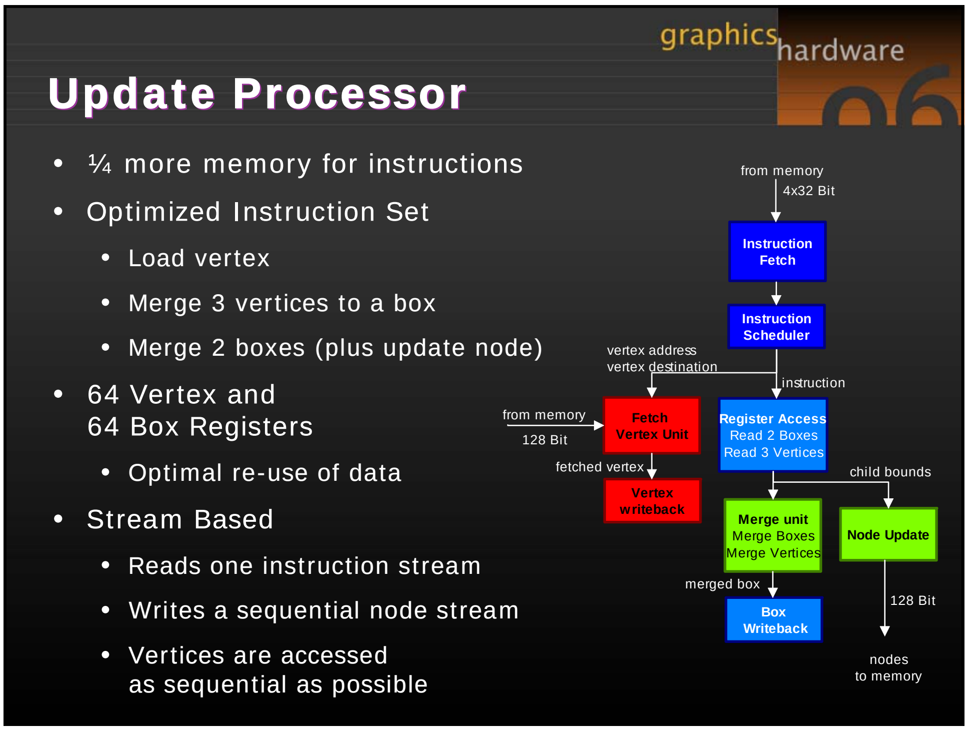
y
x
leaf
leaf
leaf



Update of B-KD Trees

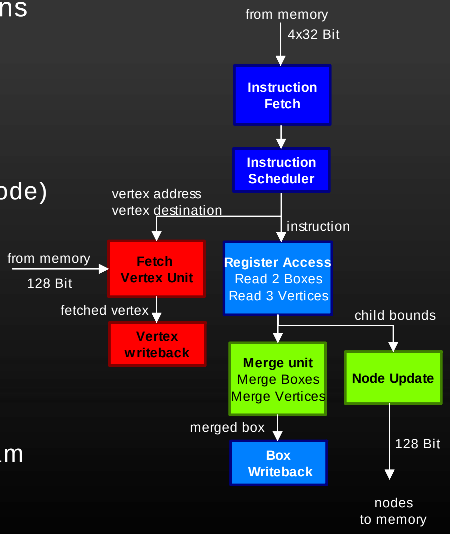
- Leaf Node
  - Fetch vertices
  - Compute leaf boxes
- Inner Node
  - Update 1D node bounds
  - Merge boxes of both children

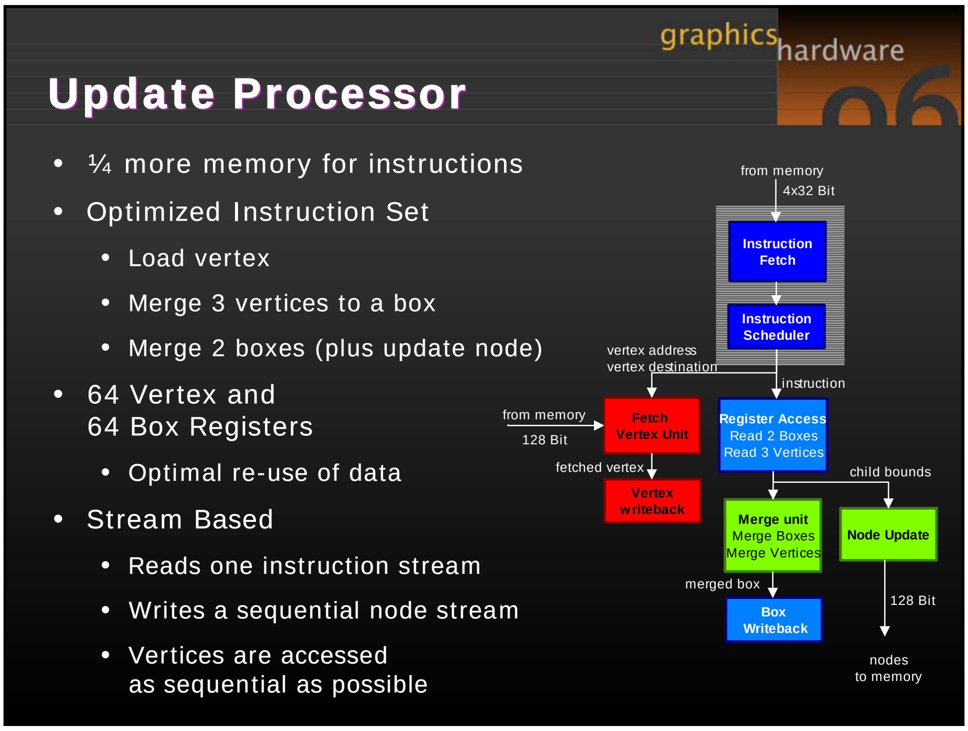




Update Processor

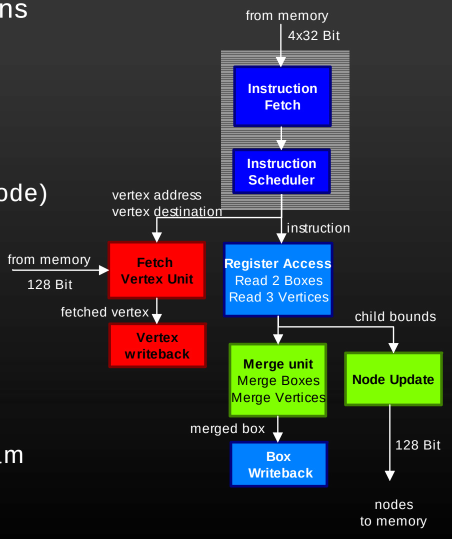
- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible

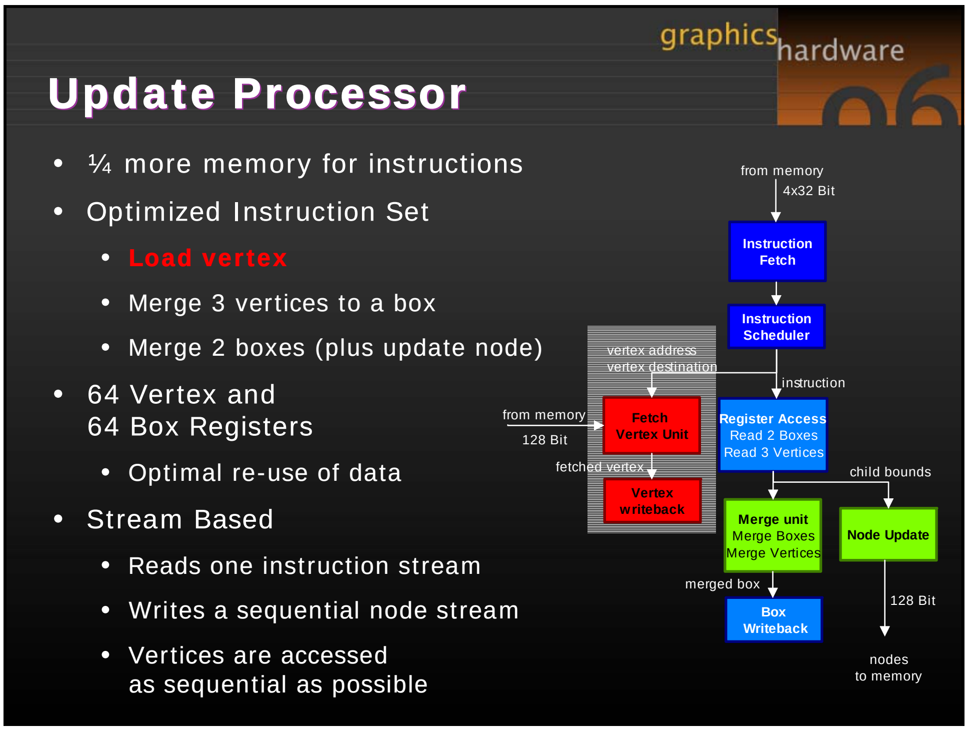




Update Processor

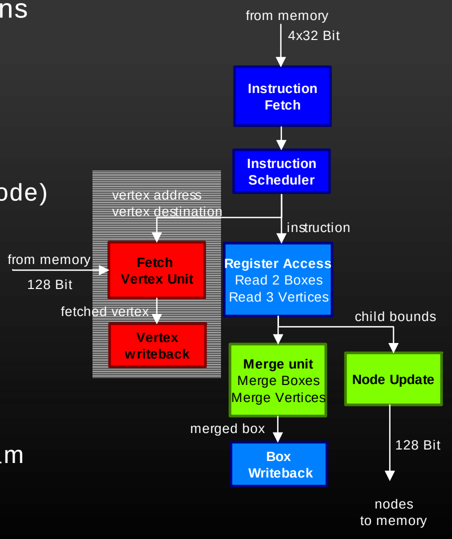
- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible





Update Processor

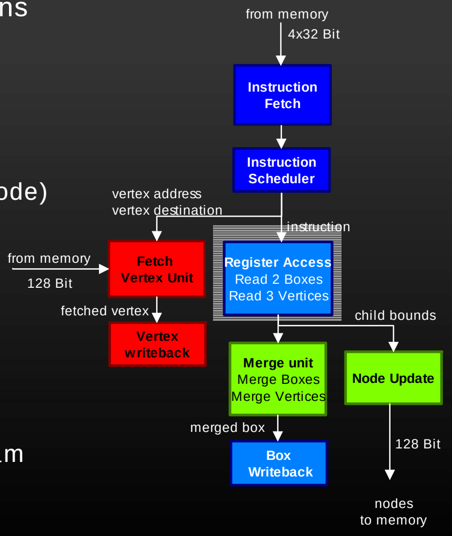
- ¼ more memory for instructions
- Optimized Instruction Set
  - **Load vertex**
  - Merge 3 vertices to a box
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible



graphics hardware 06

Update Processor

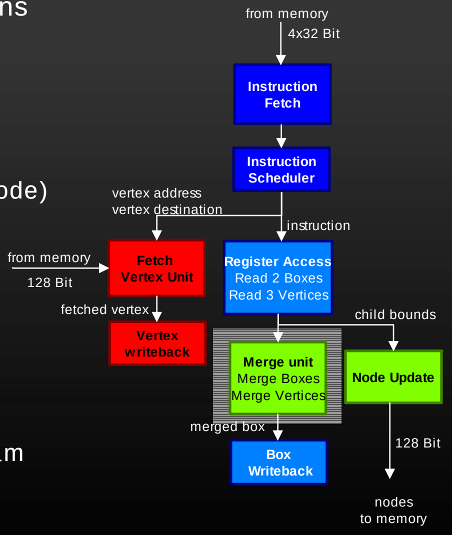
- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - **Merge 3 vertices to a box**
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible



graphics hardware 06

Update Processor

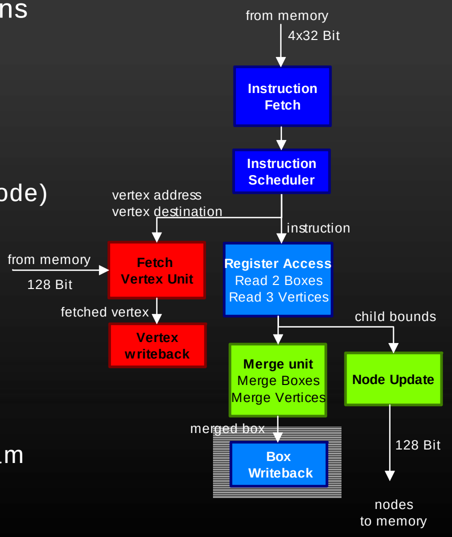
- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - **Merge 3 vertices to a box**
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible



graphics hardware 06

Update Processor

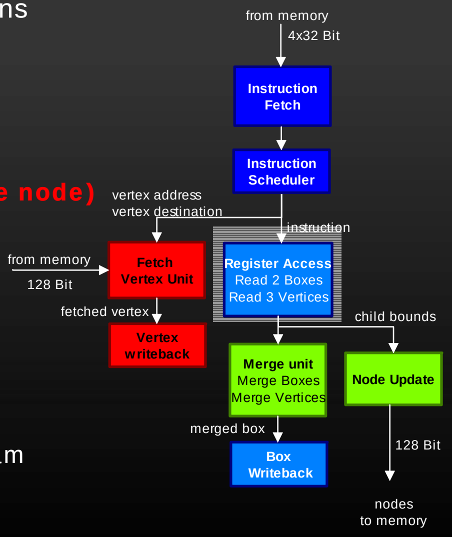
- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - **Merge 3 vertices to a box**
  - Merge 2 boxes (plus update node)
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible

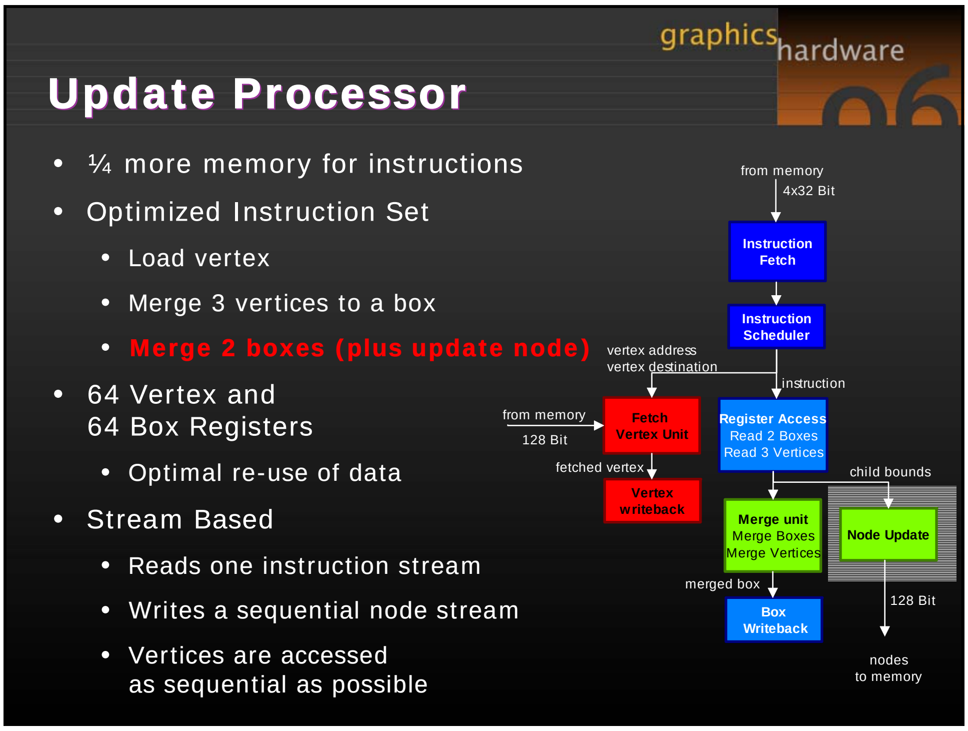


graphics hardware 06

Update Processor

- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - **Merge 2 boxes (plus update node)**
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible

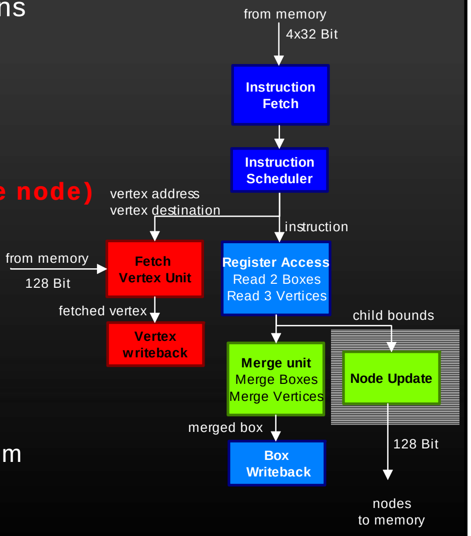


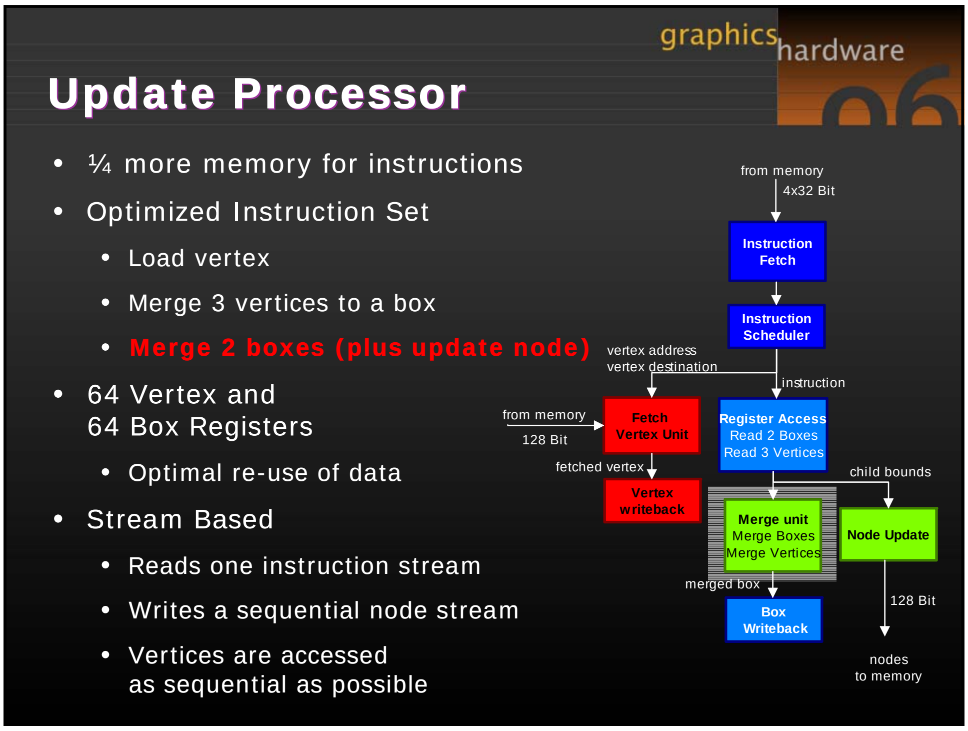


graphics hardware 06

Update Processor

- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - **Merge 2 boxes (plus update node)**
- 64 Vertex and
  64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed
    as sequential as possible

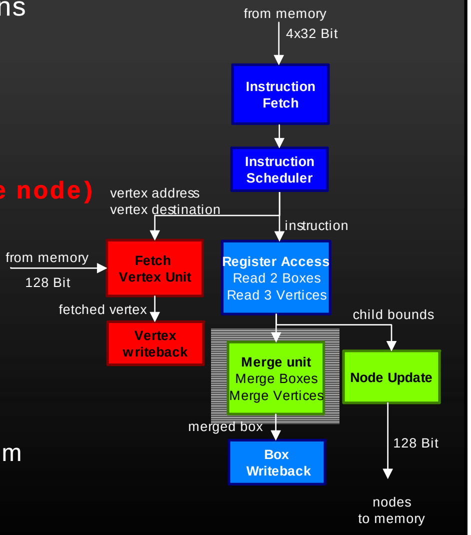


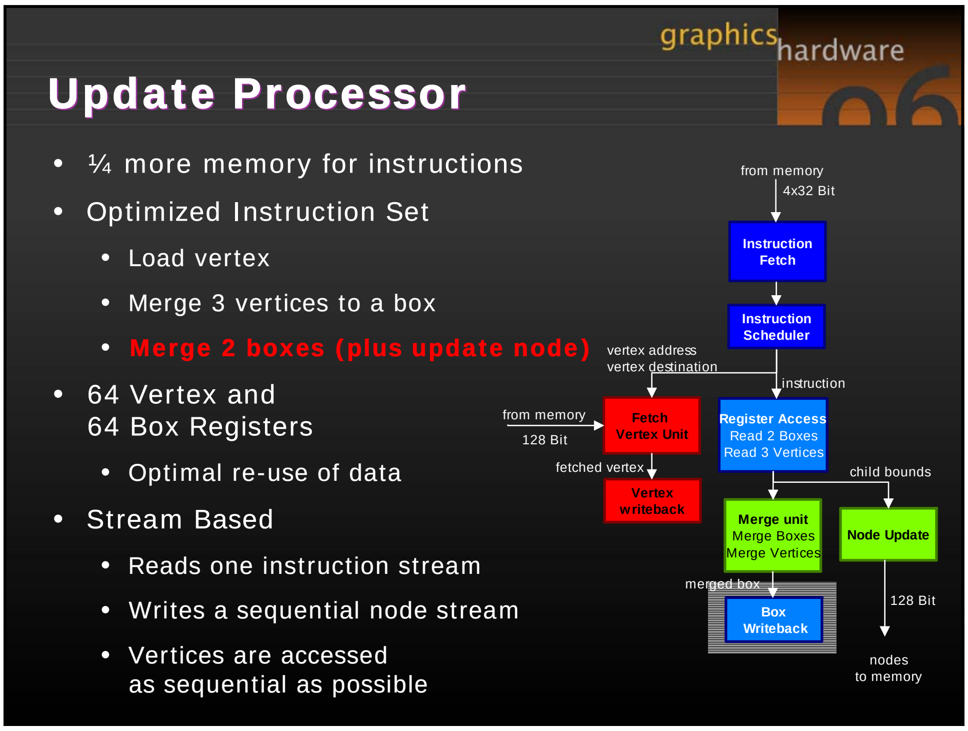


graphics hardware 06

Update Processor

- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - **Merge 2 boxes (plus update node)**
- 64 Vertex and
  64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed
    as sequential as possible

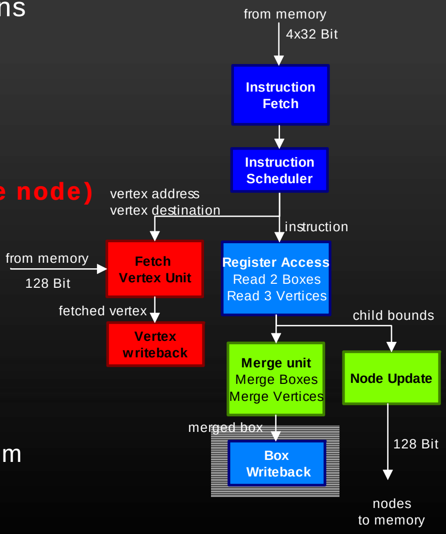


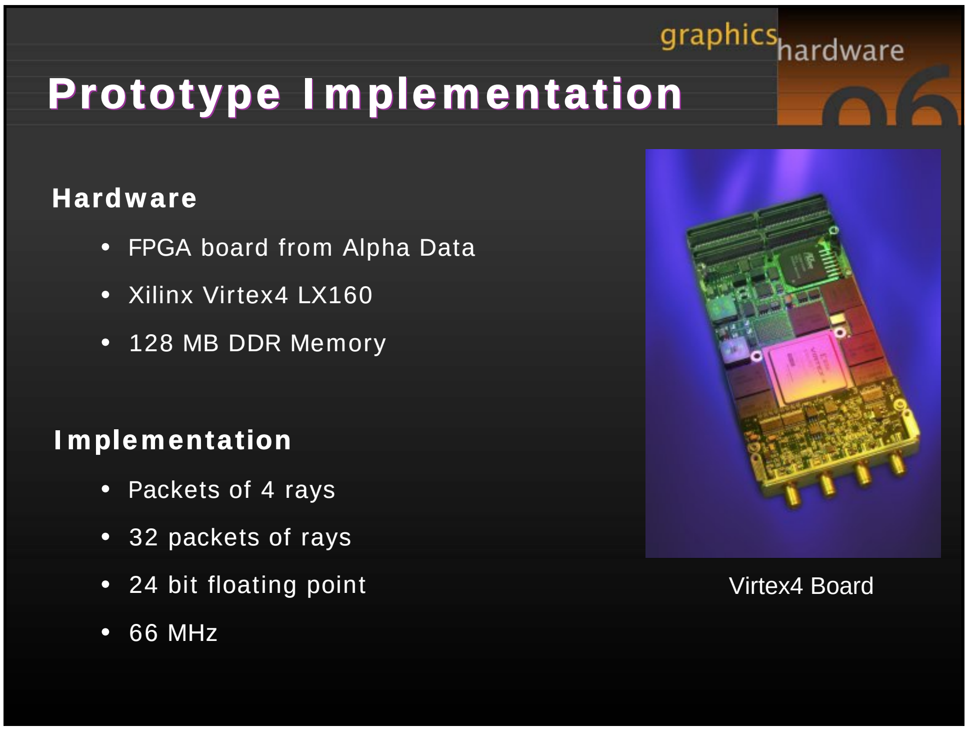


graphics hardware 06

Update Processor

- ¼ more memory for instructions
- Optimized Instruction Set
  - Load vertex
  - Merge 3 vertices to a box
  - **Merge 2 boxes (plus update node)**
- 64 Vertex and 64 Box Registers
  - Optimal re-use of data
- Stream Based
  - Reads one instruction stream
  - Writes a sequential node stream
  - Vertices are accessed as sequential as possible





graphics hardware 06

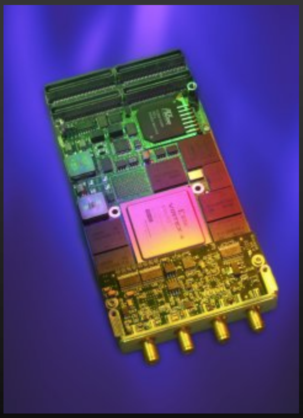
Prototype Implementation

Hardware

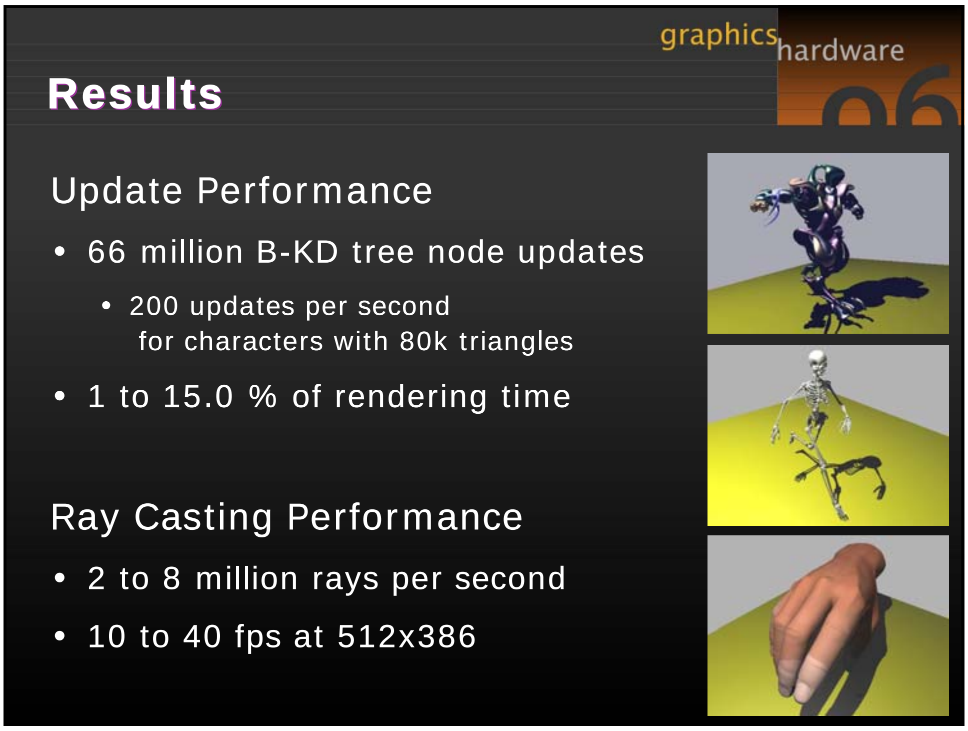
- FPGA board from Alpha Data
- Xilinx Virtex4 LX160
- 128 MB DDR Memory

Implementation

- Packets of 4 rays
- 32 packets of rays
- 24 bit floating point
- 66 MHz



Virtex4 Board



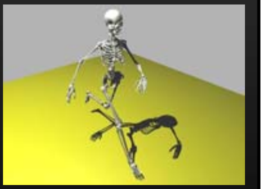
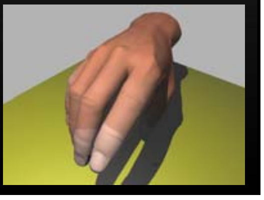
Results

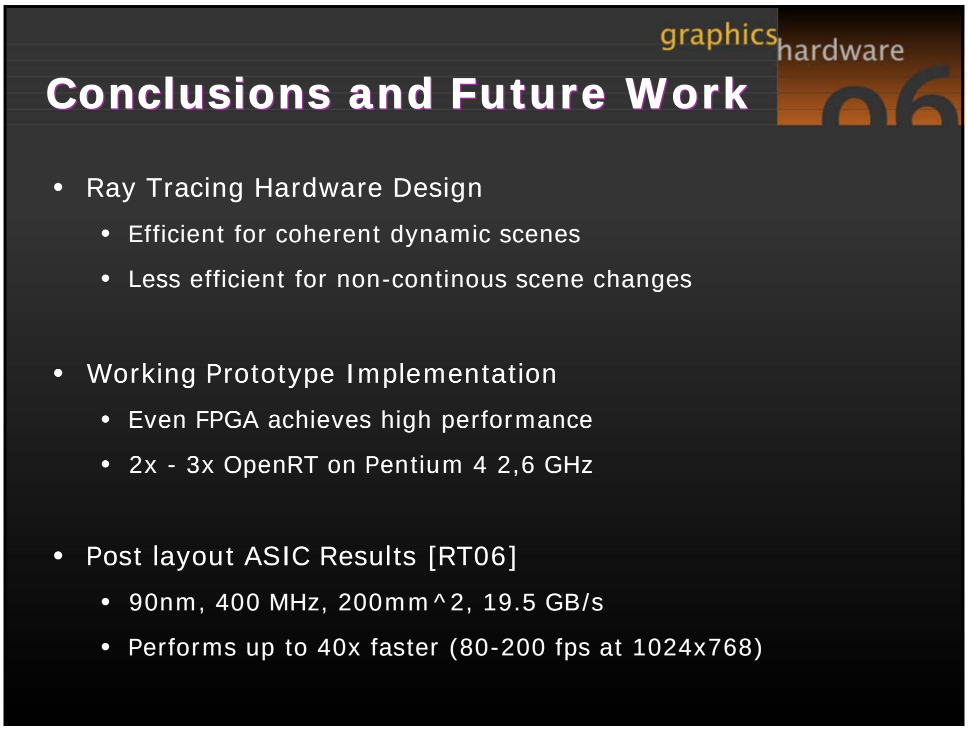
Update Performance

- 66 million B-KD tree node updates
  - 200 updates per second
    for characters with 80k triangles
- 1 to 15.0 % of rendering time

Ray Casting Performance

- 2 to 8 million rays per second
- 10 to 40 fps at 512x386



Conclusions and Future Work

- Ray Tracing Hardware Design
  - Efficient for coherent dynamic scenes
  - Less efficient for non-continous scene changes
- Working Prototype Implementation
  - Even FPGA achieves high performance
  - 2x - 3x OpenRT on Pentium 4 2,6 GHz
- Post layout ASIC Results [RT06]
  - 90nm, 400 MHz, 200mm^2, 19.5 GB/s
  - Performs up to 40x faster (80-200 fps at 1024x768)

graphics hardware 06

Live Demo

graphics hardware 06

Questions?

- Project Homepage:
  http://www.saarcor.de
- Computer Graphics Lab at Saarland University:
  http://graphics.cs.uni-sb.de