Computer Graphics
- Ray-Tracing III -

Philipp Slusallek

Ray Tracing Dynamic Scenes

- **Problem: Changing geometry requires updating indices**
  - While maintaining interactivity
- **Very little research except for 2006**
  - Efficient dynamic data structures: so far not in realtime
    - From computational geometry (i.e. kinetic data structures)
  - Animation with predefined motion [Glassner'88, Gröller'91, …]
  - Exclude dynamic primitives [Parker'99]
  - Constant time rebuild [Reinhard'00]
  - Divide and conquer [Lext'00, Wald02]
- **In 2006:**
  - Motion Compensation [Guenther06, Guenther06]
  - Updated BVH [Wächter06, Lauterbach06, Woop06, Wald06]
  - Fast rebuild [Wald06, Popov06, Ize06, Wald06, Havran06, Hunt06]

Ray Tracing Dynamic Scenes

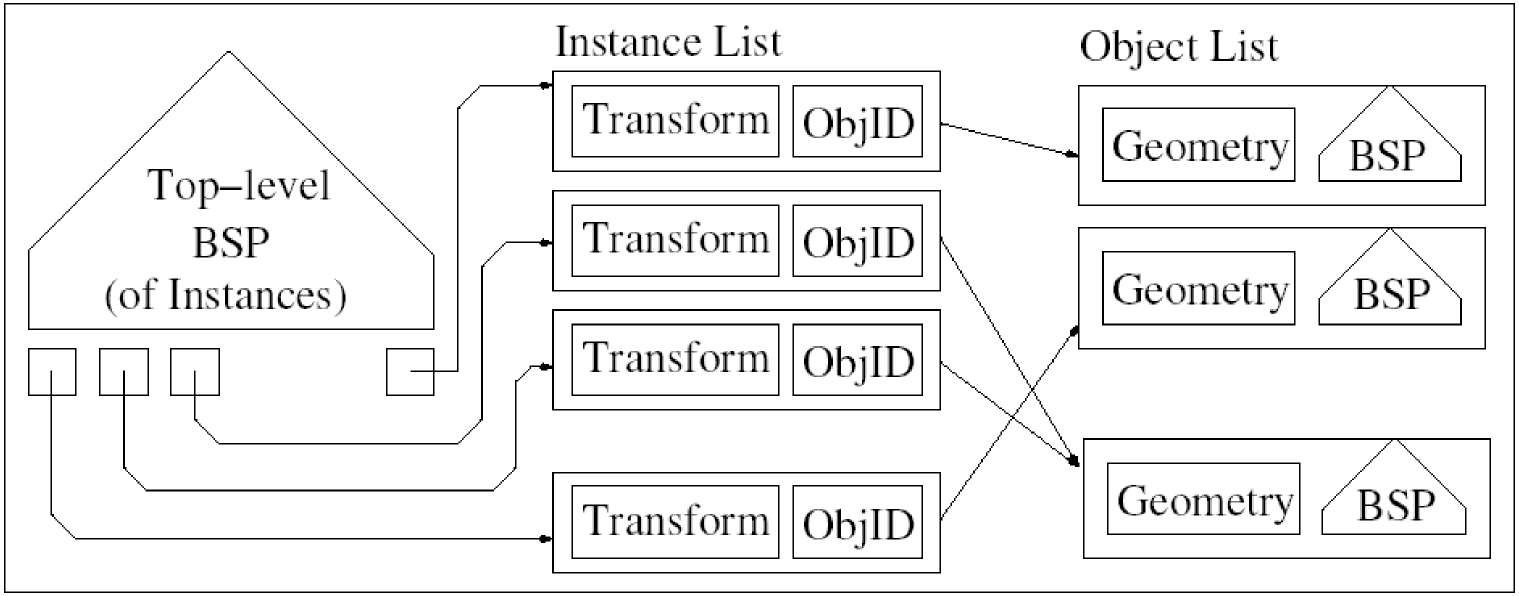
- **Different Types of Motion**
  - Static: No changes
  - Structured: Affine transformations for groups of primitives
  - Continuous: Adjacent geometry stays adjacent during animation
  - Unstructured: Arbitrary or random movements of primitives
- **General Framework**
  - What does the application know about the motion?
  - How do we communicate that to the renderer? (➔ API)
  - How to efficiently and effectively use this information?
- **General Approaches**
  - Partition scene depending on type of motion
  - Build index valid for longer time (fuzzy indicies)
  - Rebuild entire index each frame
  - Lazy building of tree (on-the-fly only where needed)
  - Update index each frame

Partitioning: Divide & Conquer

- **Observation**
  - 80/20 rule: Very often a simple approach is sufficient
  - Building hierarchical index structures requires O(n logn)
    - Divide and conquer reduces complexity
- **Categorize primitives into independent groups/objects**
  - Static parts of a scene (often large parts of a scene)
  - Structured motion (affine transformations for groups of primitives)
  - Anything else
- **Select suitable approach for each group**
  - Do nothing
  - Transform rays instead of primitives
  - Only update index structure for remaining part of the scene

Divide & Conquer Approach

- **Two-level index structure**
  - Find relevant object along the ray
  - Transform ray (efficient SSE code)
  - Find primitives within object
  - Same kd-tree traversal algorithms in both cases
- **Results in some run-time overhead**



Implementation

- **KD-tree building algorithms**
  - Static & structured motion
    - Build once with sophisticated and slow algorithm [Havran'01]
    - Optimize for traversal (as low as 1.5 intersection per ray)
  - Unstructured Motion
    - Will be used for single or few frames
    - Balance construction and traversal time
      - E.g. allow more primitives in leaf nodes
  - Top-Level tree over objects:
    - Significantly more efficient than for primitives
    - Possible splitting planes for kd-tree are already given (Bbbox)

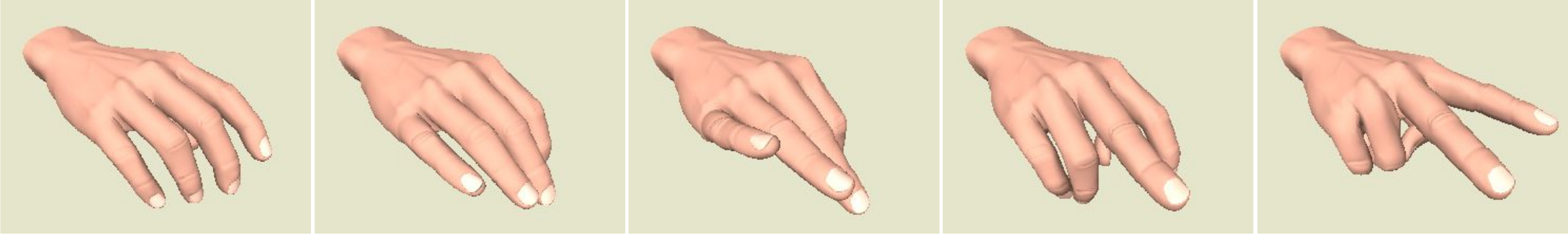
Implementation

- **Index Structure Updates**
  - Static: Done
  - Structured Motion
    - Update transformation
    - Update of top-level index with transformed bounding boxes
  - Unstructured Motion
    - Rebuild local index
    - Schedule top-level update, iff bounding box changed

Updating the Index Structure

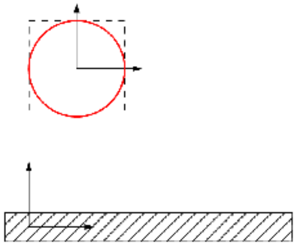
- **IDEA**
  - „Make dynamic scenes static”

- **Assumptions:**
  - Deformation of a base mesh (constant connectivity)
  - All frames of animation known in advance
  - Continuous motion

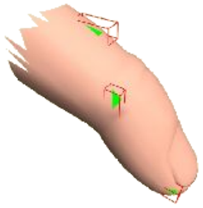


Method Overview

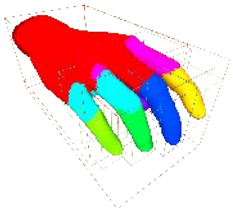
- **Motion decomposition**
  - Affine transformations
  - + residual motion



- **Fuzzy kd-tree**
  - Handles residual motion

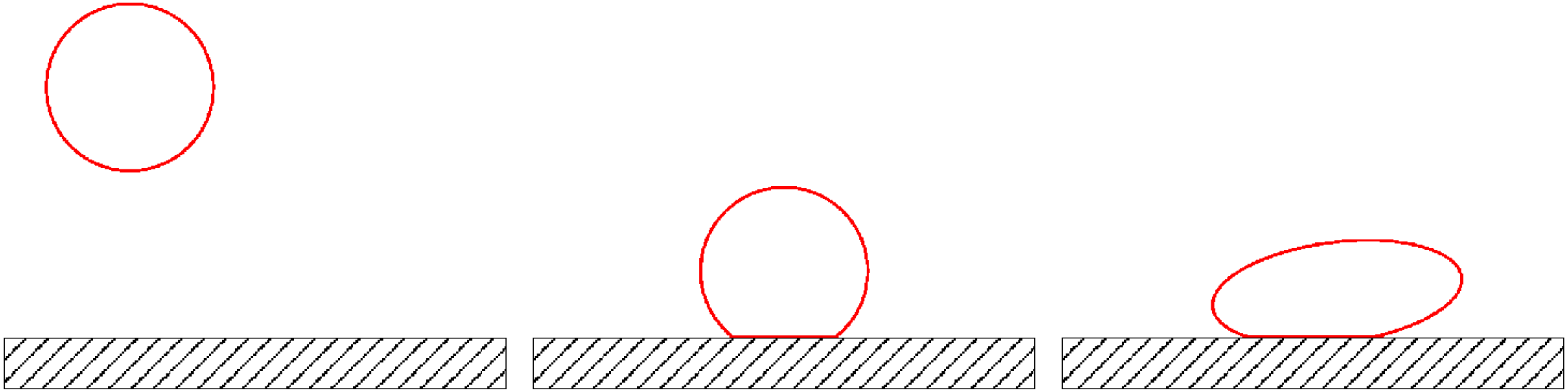


- **Clustering**
  - Exploit local coherent motion



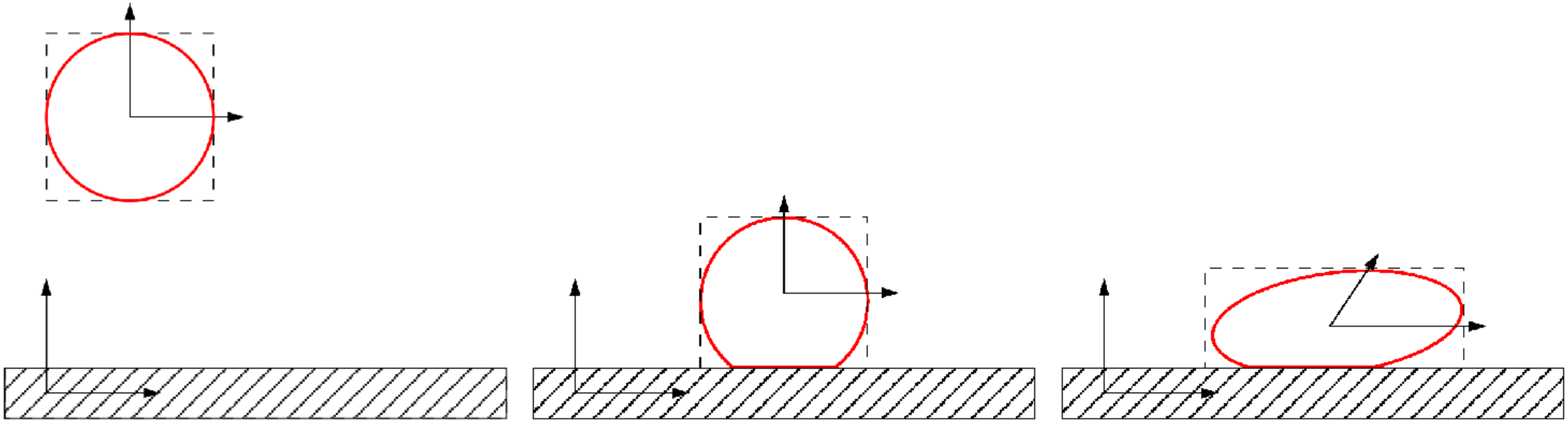
Motion Decomposition

- **Dynamic scene: ball thrown onto floor**

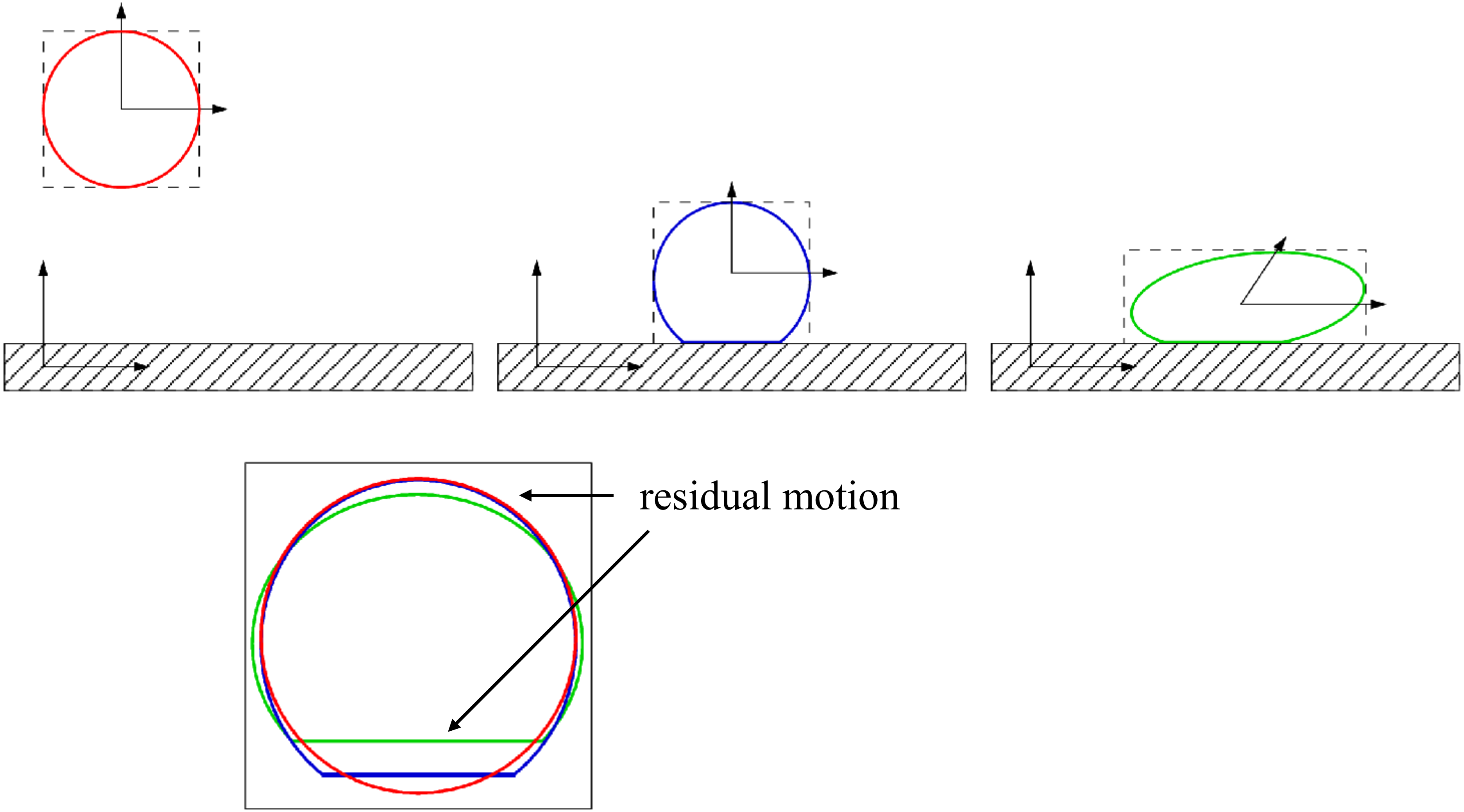


Motion Decomposition

- **Affine transformations**
  - Approximate deformations
  - Include shearing (3rd frame)

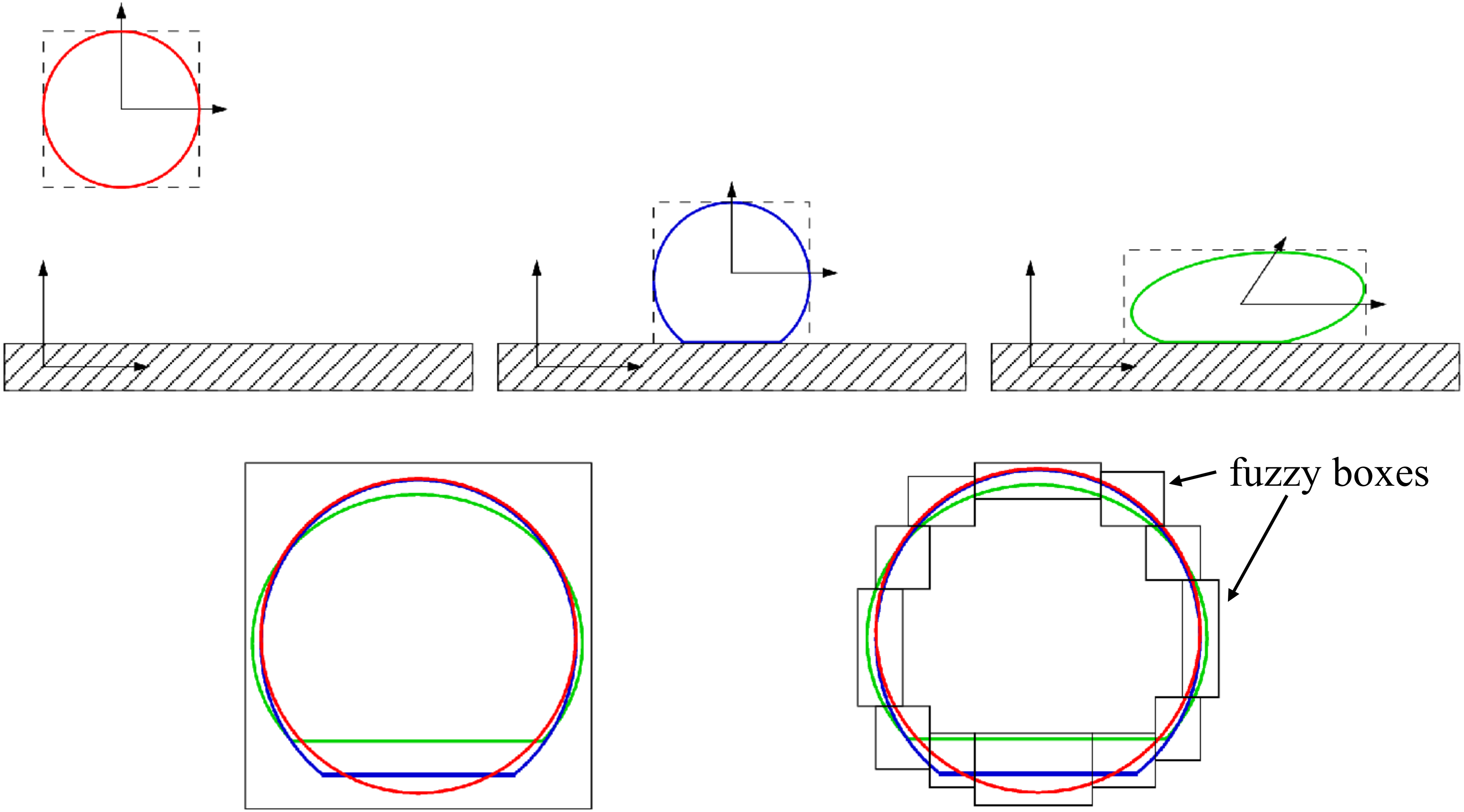


Motion Decomposition



residual motion

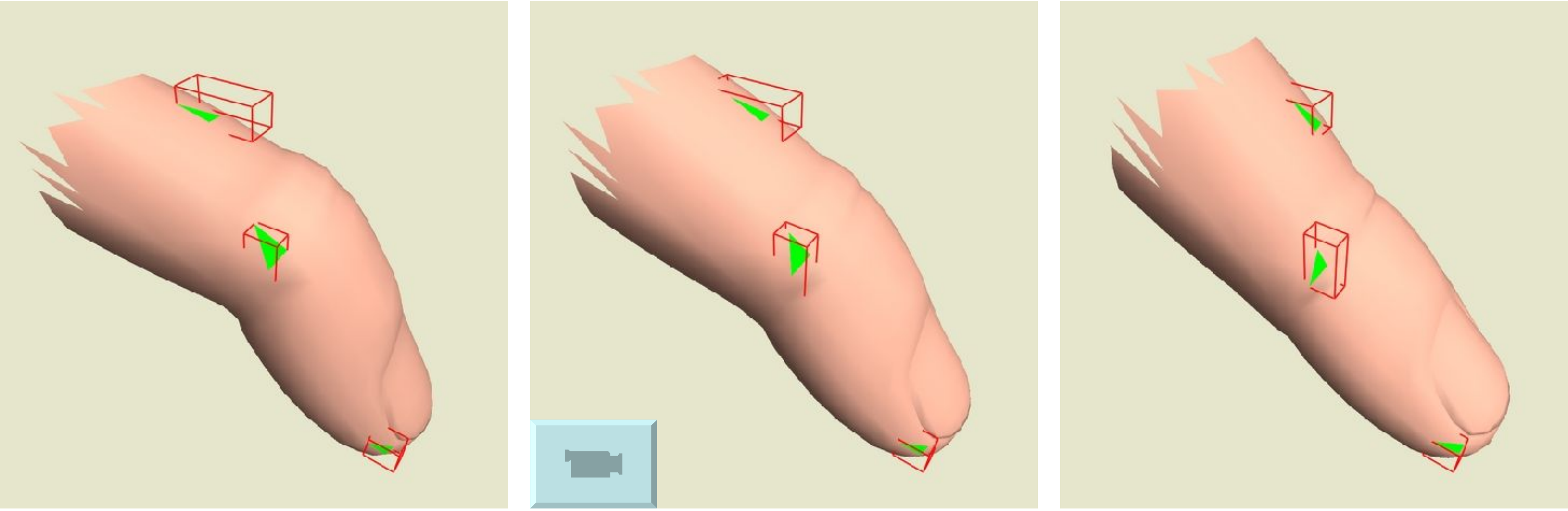
Motion Decomposition



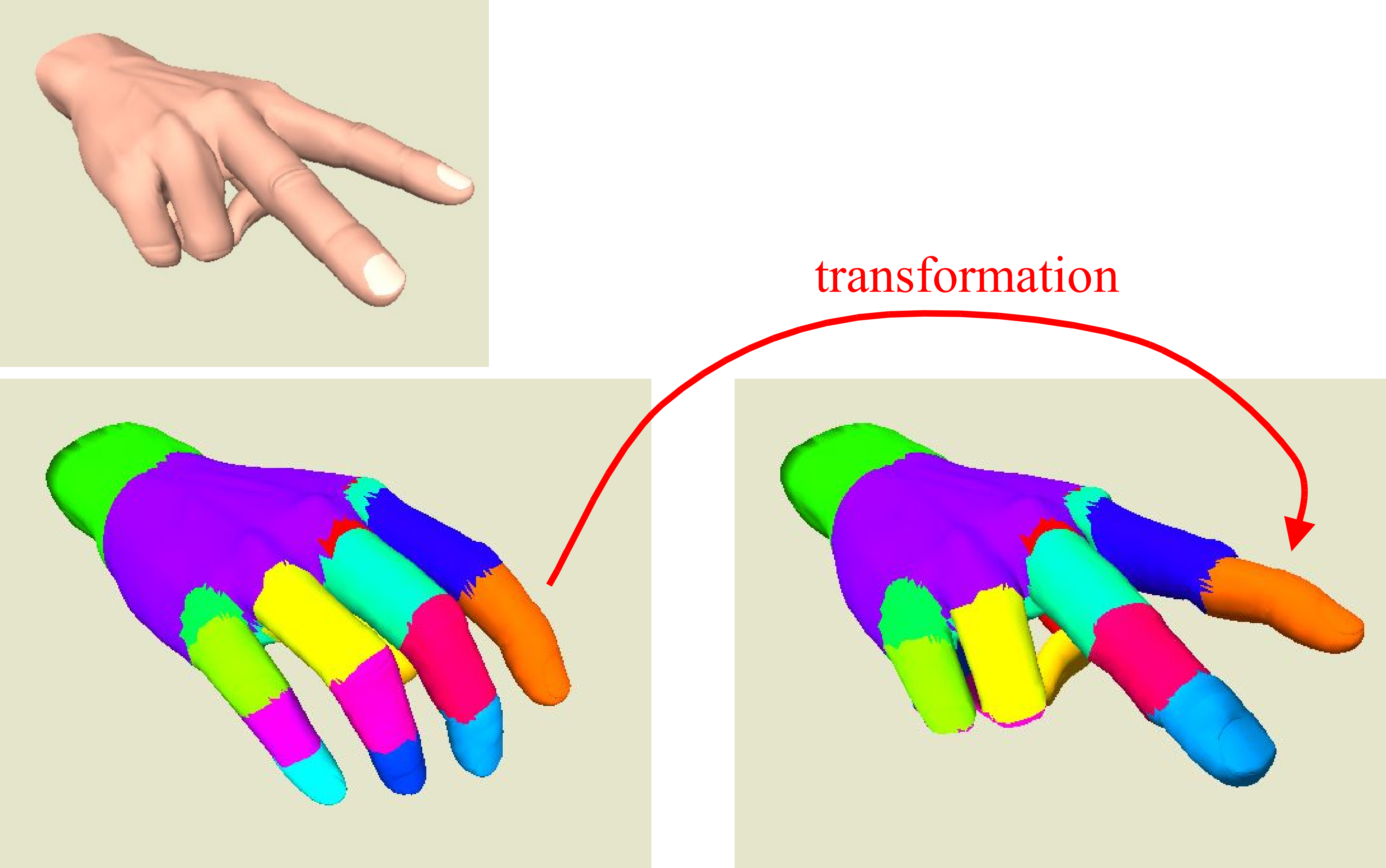
Fuzzy KD-Tree

- **Handles residual motion**

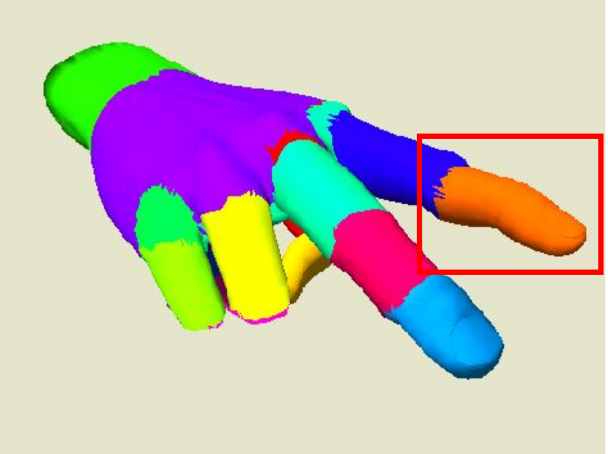
- **KD-Tree over the fuzzy boxes of triangles**
  - Valid over complete animation

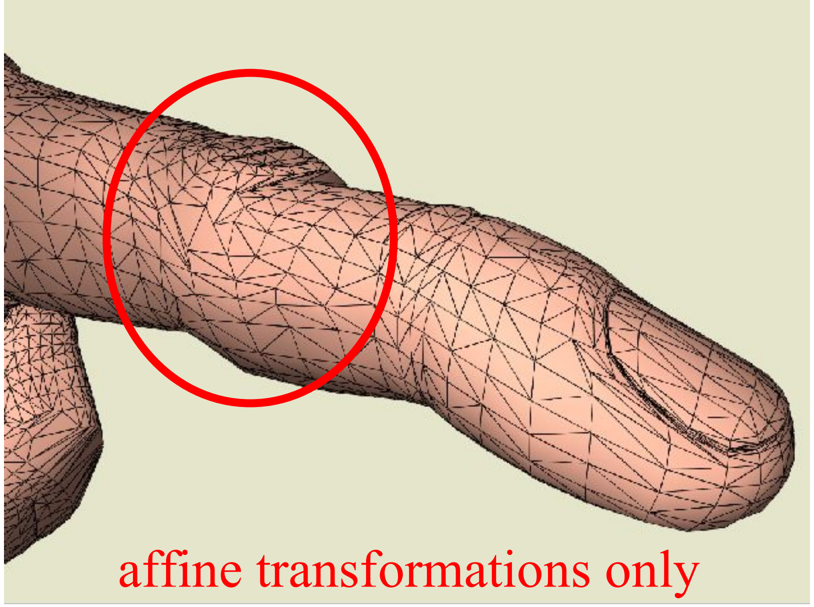


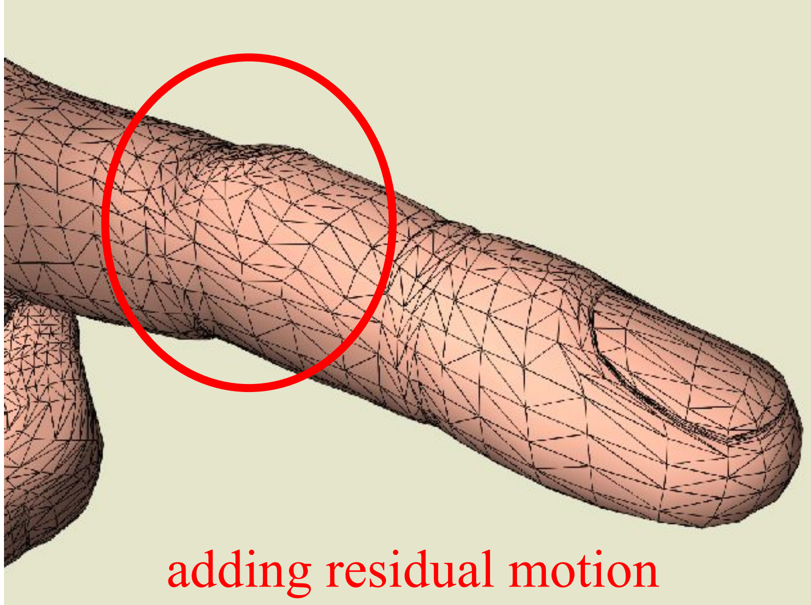
Illustration



Illustration





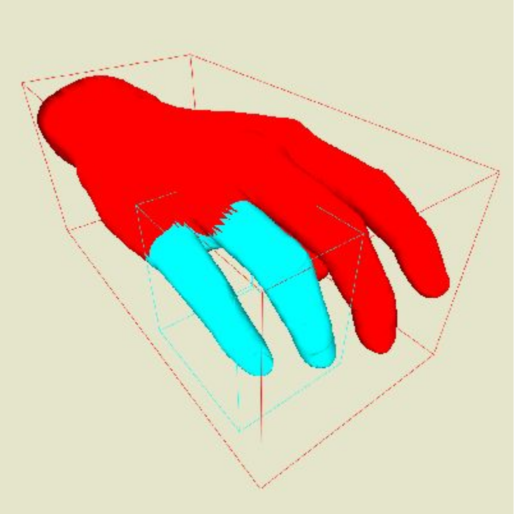


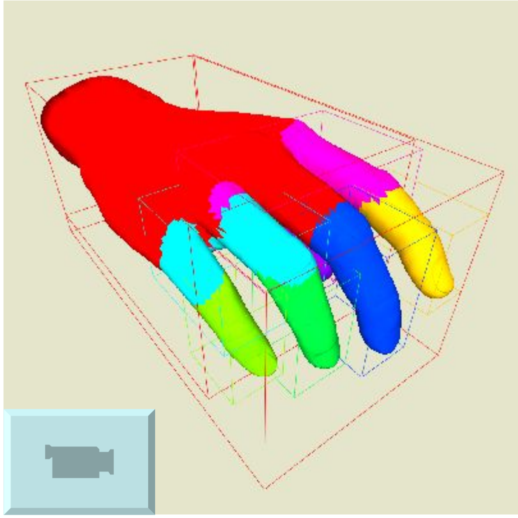
Details: Clustering

- **Efficient ray tracing:**
  - Requires small fuzzy boxes
  - Must minimize residual motion
  - Should cluster coherently moving triangles
    - Results in few object
- **Many clustering algorithms**
  - But mostly for static meshes
  - Not designed for ray tracing
- **Develop new one**
  - Based on Lloyd relaxation

Clustering Algorithm

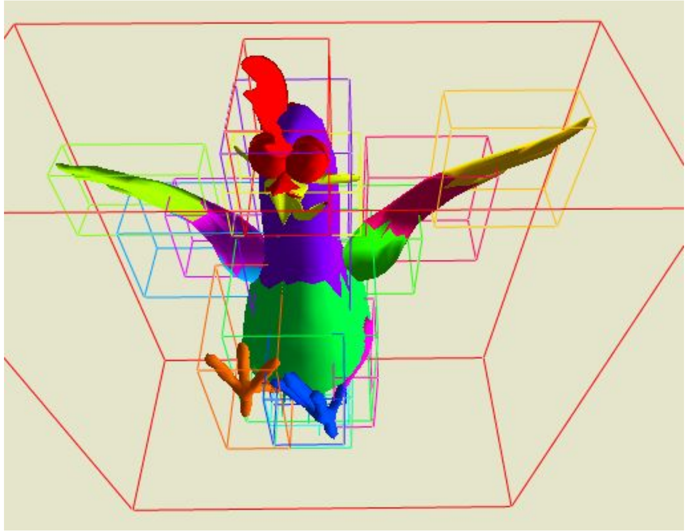
- **Start with one cluster (all triangles)**
- **Lloyd relaxation:**
  - Find transformations for clusters
    - Linear least squares problem
  - Recluster triangles
    - By choosing transformation with minimal error
  - Until convergence
    - No triangles move between clusters
- **Insert new cluster**
  - Seeded by triangle with highest residual motion
- **Until improvement below threshold**
  - Measured by summing all error terms

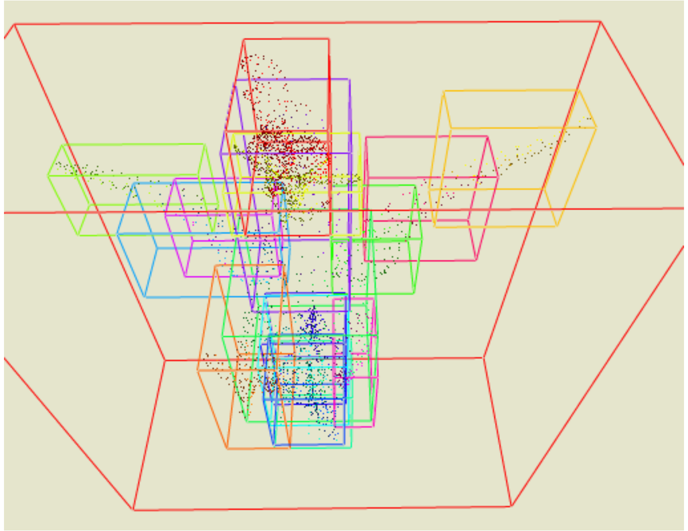




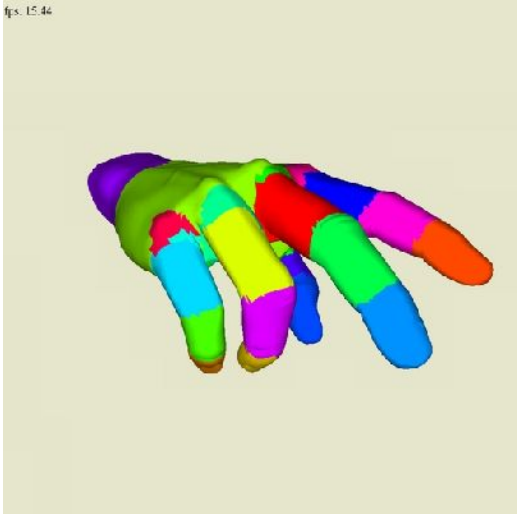
Ray Tracing: Two-level Approach

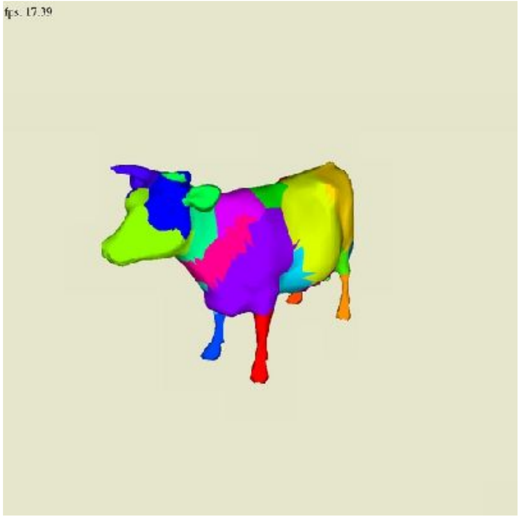
- **Build top-level kd-tree over current cluster bounds**
- **Transform rays into local coordinate system**
  - Inverse affine transformation of cluster
  - from motion decomposition
- **Traverse fuzzy kd-tree of cluster**

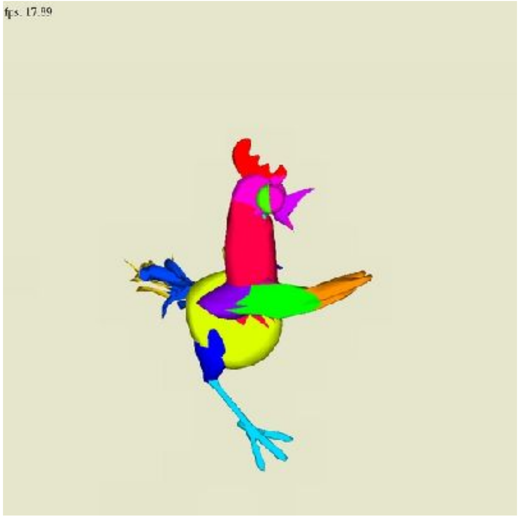




Video

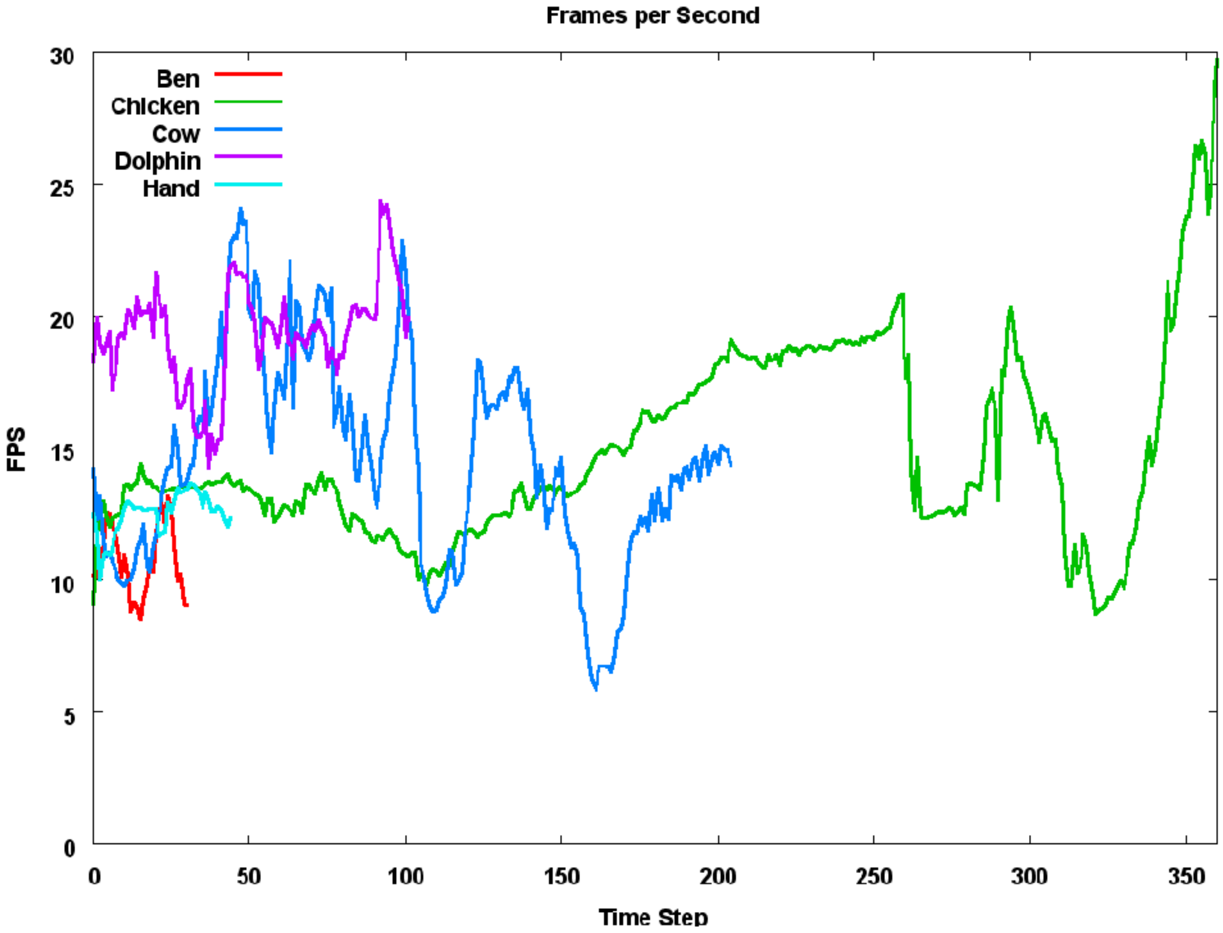






Ray Tracing Performance

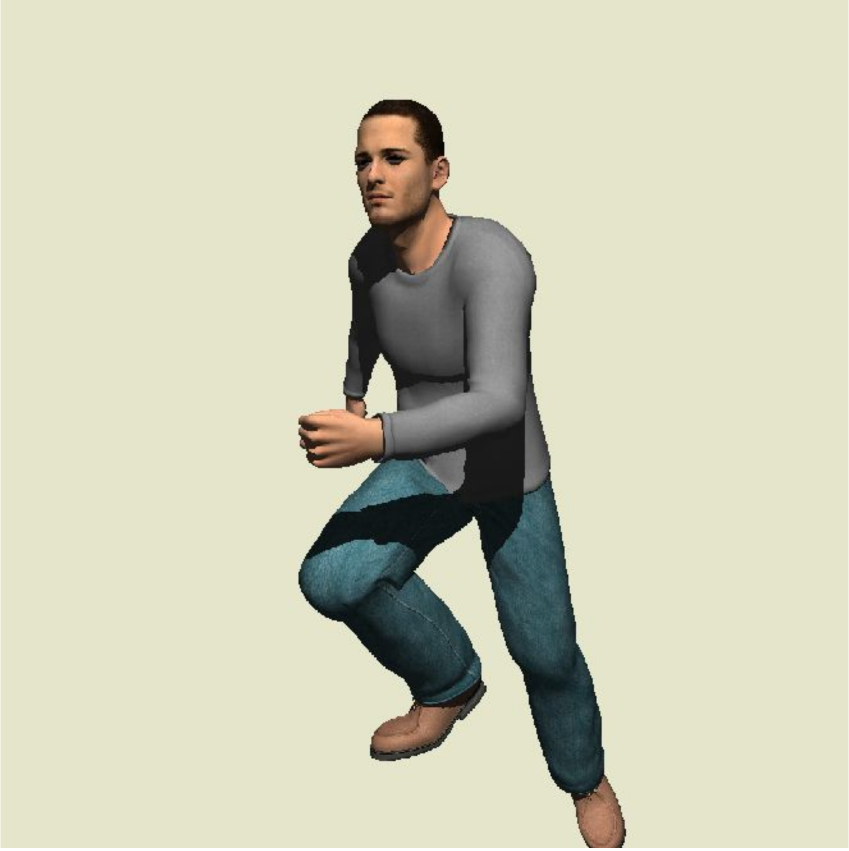
- **Single CPU**
  - (Opteron 2.8 GHz)
- **1024×1024 px**
- **Incl. shading**

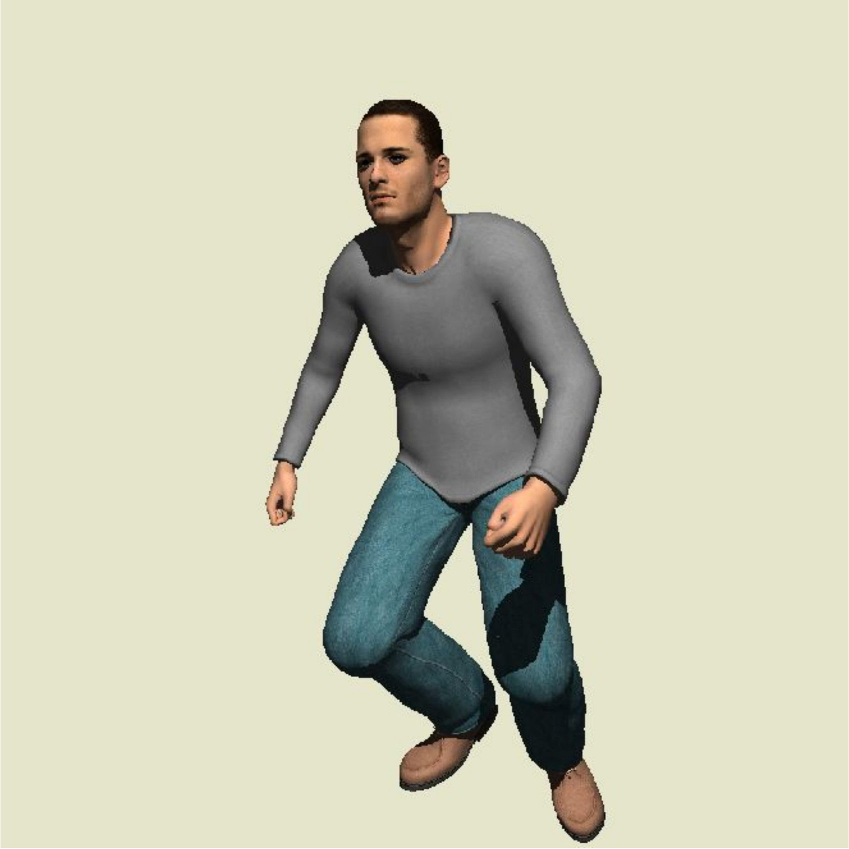


More Complex Shading

- **With texturing, lighting, shadows: 2.2 fps**

  (static kd-tree: 4.1 fps)



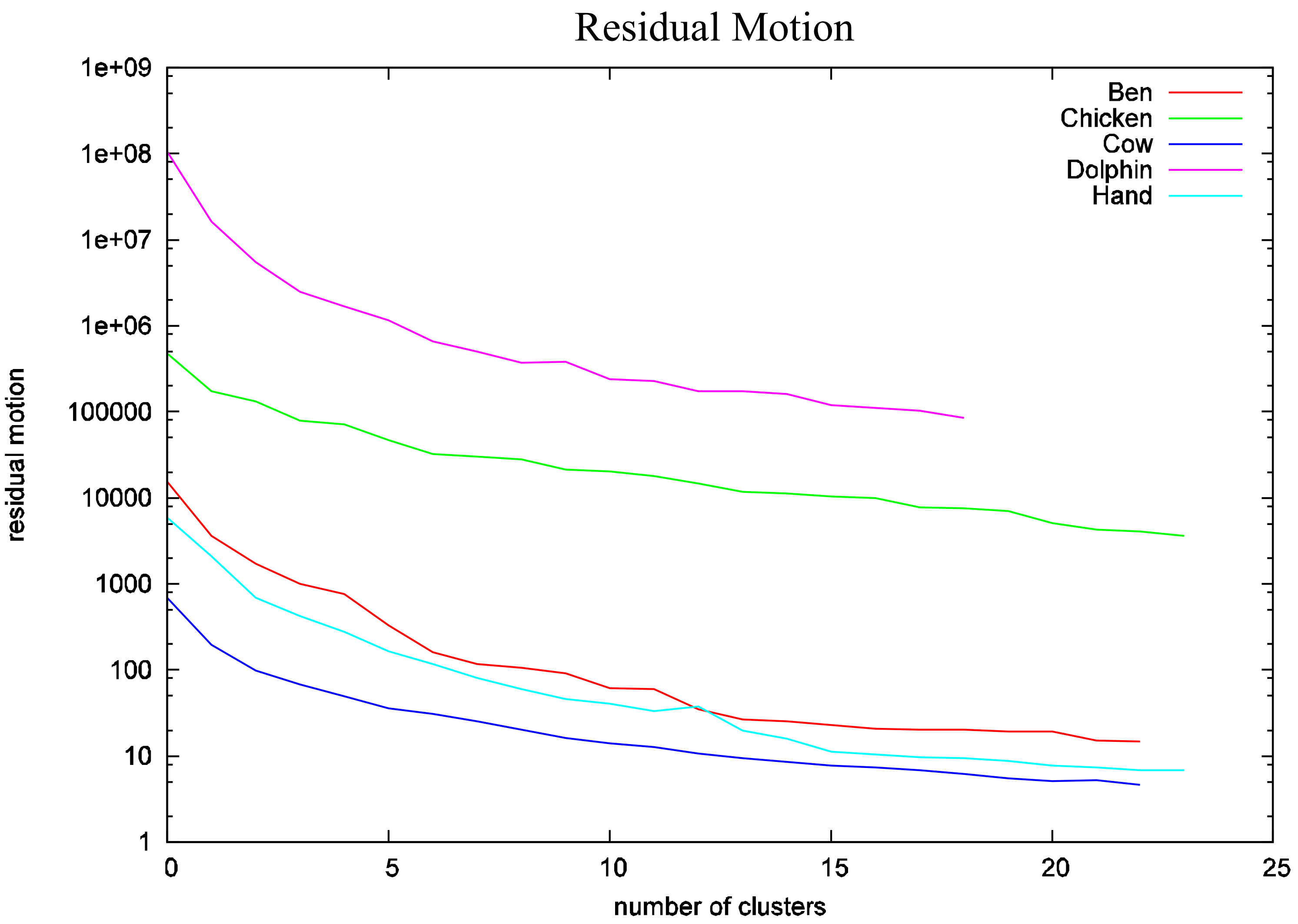


Comparison to Static KD-Tree

- **Baseline: separate static kd-tree per frame**

- **Traversal steps**
  - Factor 1.5 - 2
- **Intersections**
  - Factor 1.2 – 2, Cow 4, Chicken 6
- **Average fps**
  - Factor 1.2 – 2.6, alone two-level kd-tree costs ca. 30%
- **Memory**
  - only one fuzzy kd-tree (+ transformation matrices)
  - vs. #frames static kd-trees

Clustering Process



Future Work

- **Clustering also in time domain**
  - Better adaptation to separated animation sequences
  - E.g. with the chicken: walking, being scared, flying
    -
- **Handle interpolation between key frames**
  - By interpolation of the computed transformations?
    -
- **Interaction with dynamic scenes**
  - So far all poses known in advance
    - Harnessing more information from application
  - Skinning operators, skeleton, joint angles with limits
  - See also [TVC/PG06]

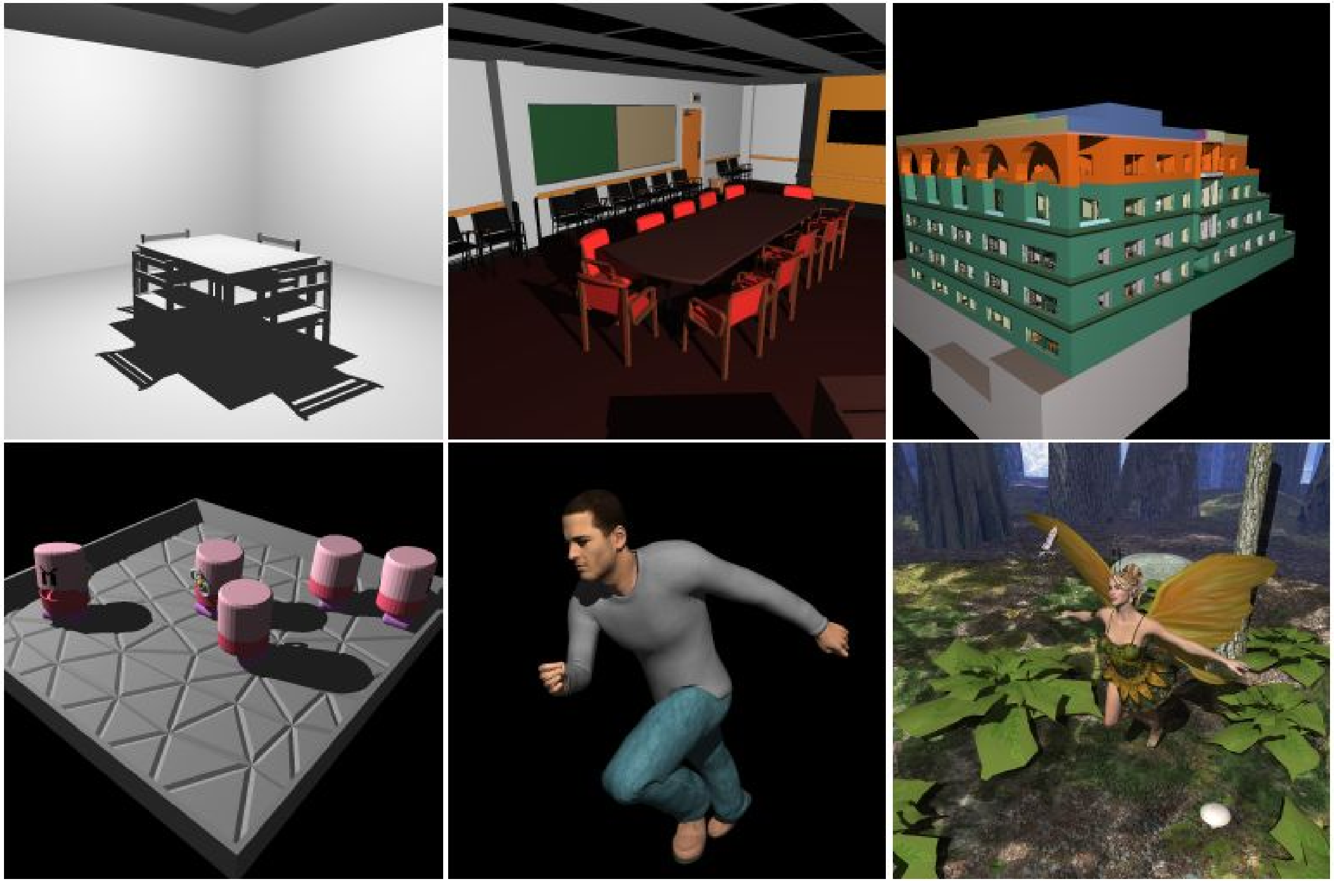
Curing Mad Cow Desease ?!?

Updating Spatial Index Structures

Bounding Volume Hierarchies

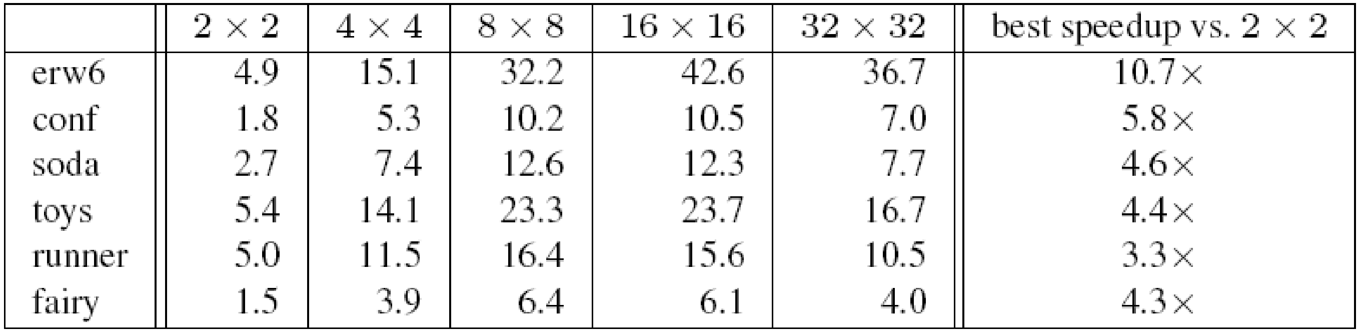
- **Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies [Wald, TOG06]**
- **Build binary BVH hierarchy**
  - Using variation of SAH algorithm
- **Fast packet traversal**
  - Test first ray against box of child node
    - If hit, immediately traverse child node
  - Test frustum of rays against box of child node
    - If miss, do not traverse child node
  - Otherwise, test all ray until hit is found
    - Do not traverse in case of no hit
  - Optimization, store order of child nodes for ever axis
    - Allows for more effective early ray termination
  - Can use fast SIMD computation for packets of rays

Test Scenes



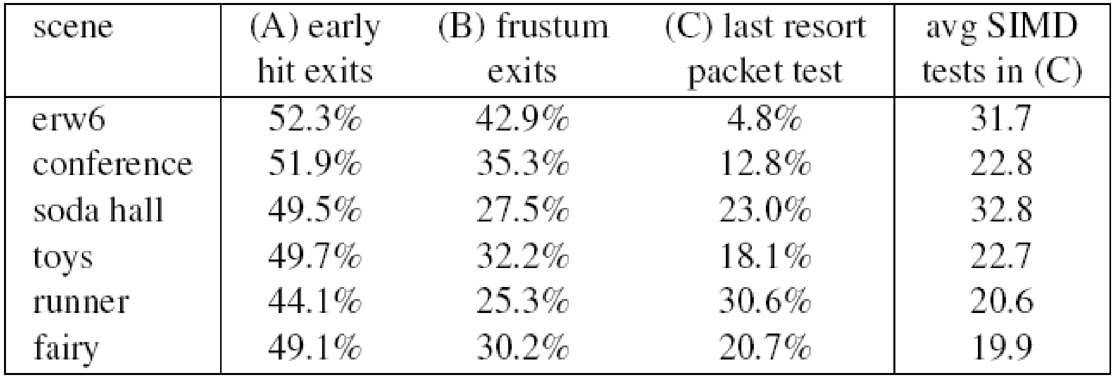
Test Results

- **Performance with Packet Size**
  - 2.6 GHz Opteron



	2 × 2	4 × 4	8 × 8	16 × 16	32 × 32	best speedup vs. 2 × 2
erw6	4.9	15.1	32.2	42.6	36.7	10.7×
conf	1.8	5.3	10.2	10.5	7.0	5.8×
soda	2.7	7.4	12.6	12.3	7.7	4.6×
toys	5.4	14.1	23.3	23.7	16.7	4.4×
runner	5.0	11.5	16.4	15.6	10.5	3.3×
fairy	1.5	3.9	6.4	6.1	4.0	4.3×

- **Effectiveness of Early Ray Tests**



scene	(A) early hit exits	(B) frustum exits	(C) last resort packet test	avg SIMD tests in (C)
erw6	52.3%	42.9%	4.8%	31.7
conference	51.9%	35.3%	12.8%	22.8
soda hall	49.5%	27.5%	23.0%	32.8
toys	49.7%	32.2%	18.1%	22.7
runner	44.1%	25.3%	30.6%	20.6
fairy	49.1%	30.2%	20.7%	19.9

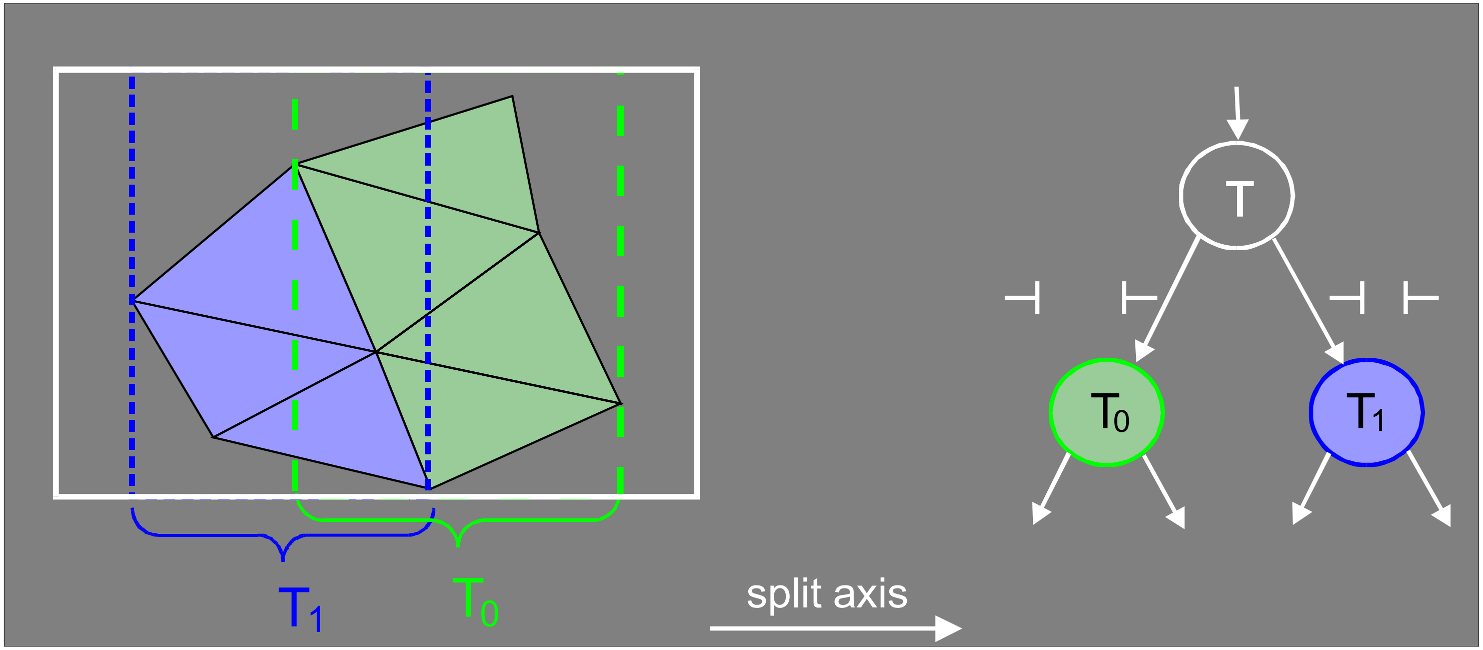
BHV Updates

- **Choose a Good Initial Pose**
  - Avoids accidental closeness of triangles that separate later
  - Usually not a big issue
- **Good simple approach**
  - Test various poses from (known) animation
- **Build BHV over entire animation**
  - Build BHV hierarchy with respect to best partitioning over all animation frames
  - Usually not much improvement
    - Shows that hierarchies stay good during animations

Bounding KD-Trees:
Mixing Bounding Volumes and KD-trees

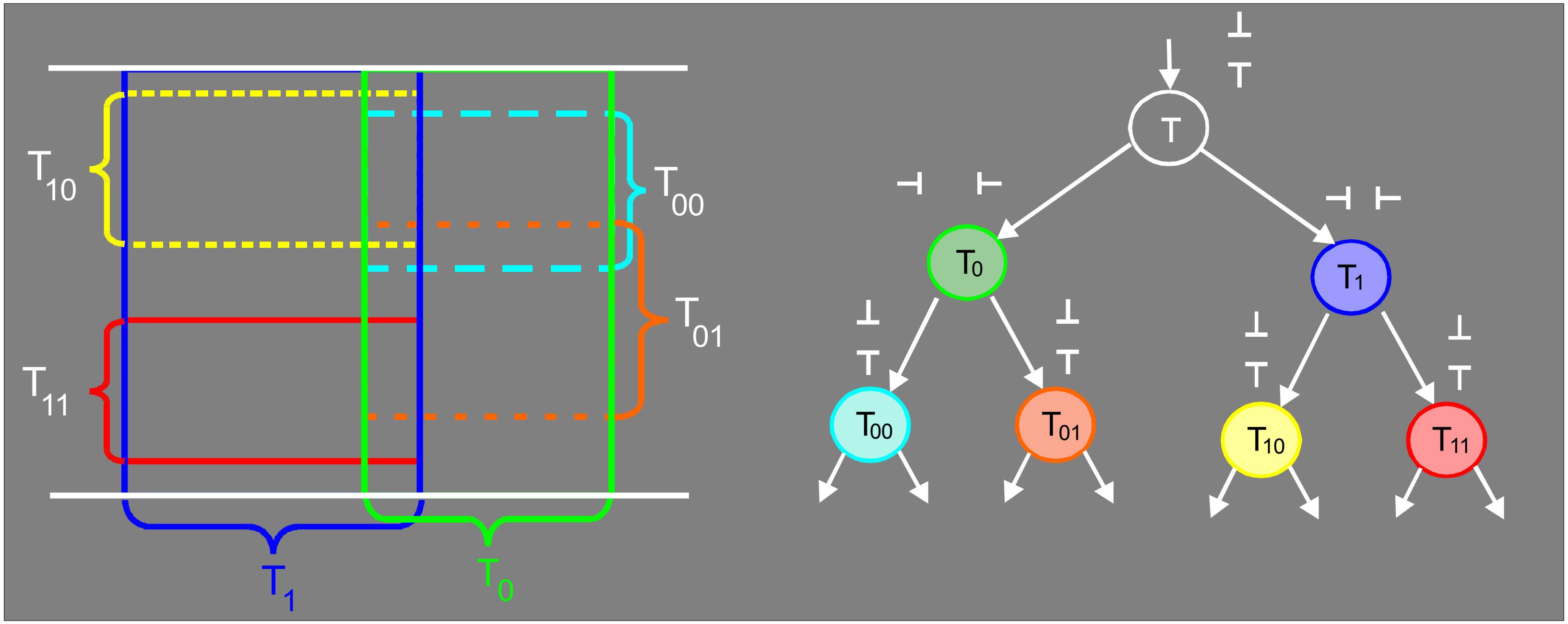
Definition of B-KD Trees

- **B-KD Tree (Bounded KD-Tree)**
  - Binary Tree
  - 1D bounding intervals for each child
  - Leaf nodes point to a single primitive



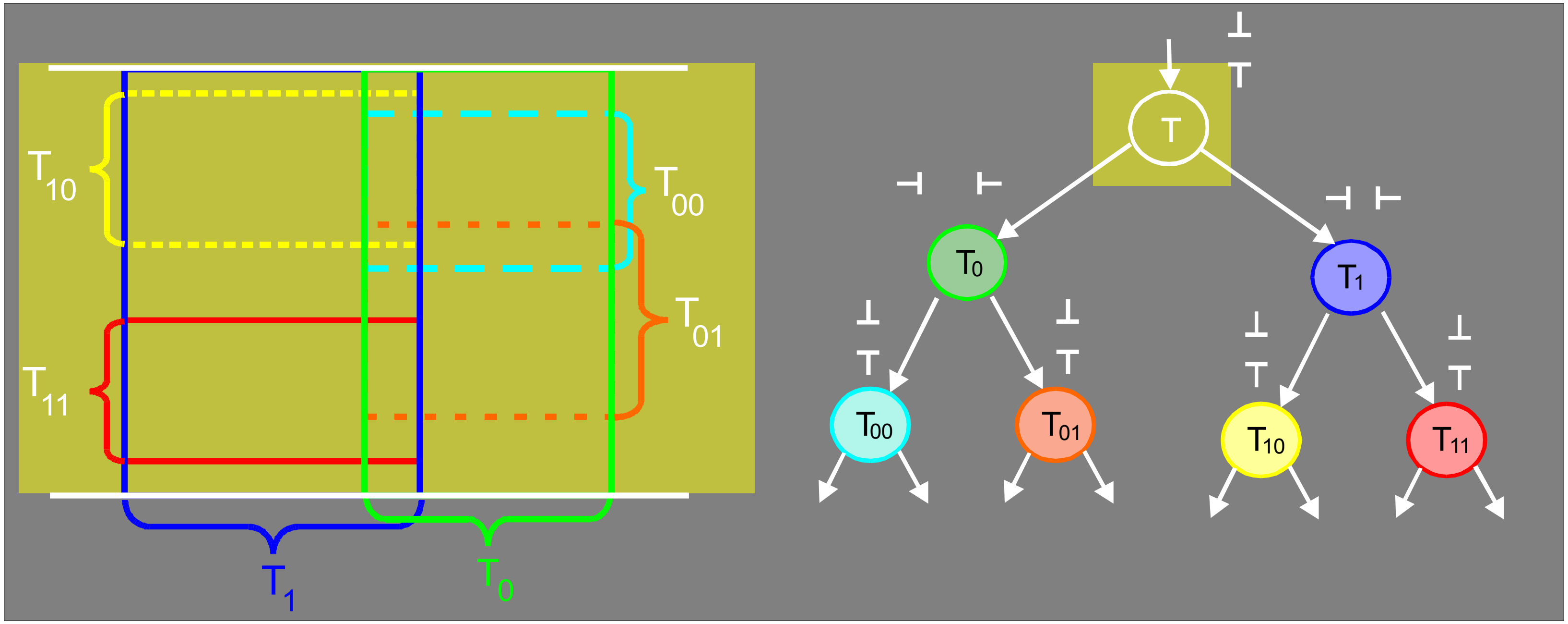
B-KD Tree Subdivision

- **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



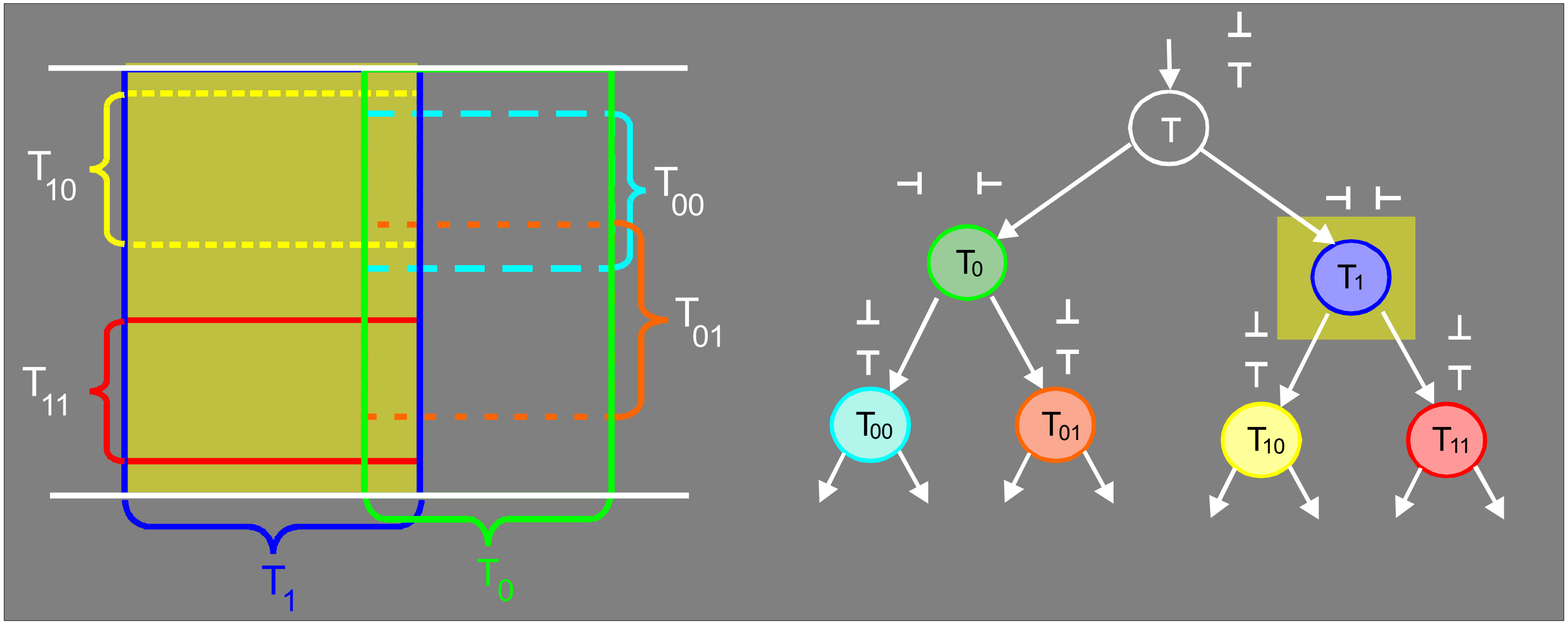
B-KD Tree Subdivision

- **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



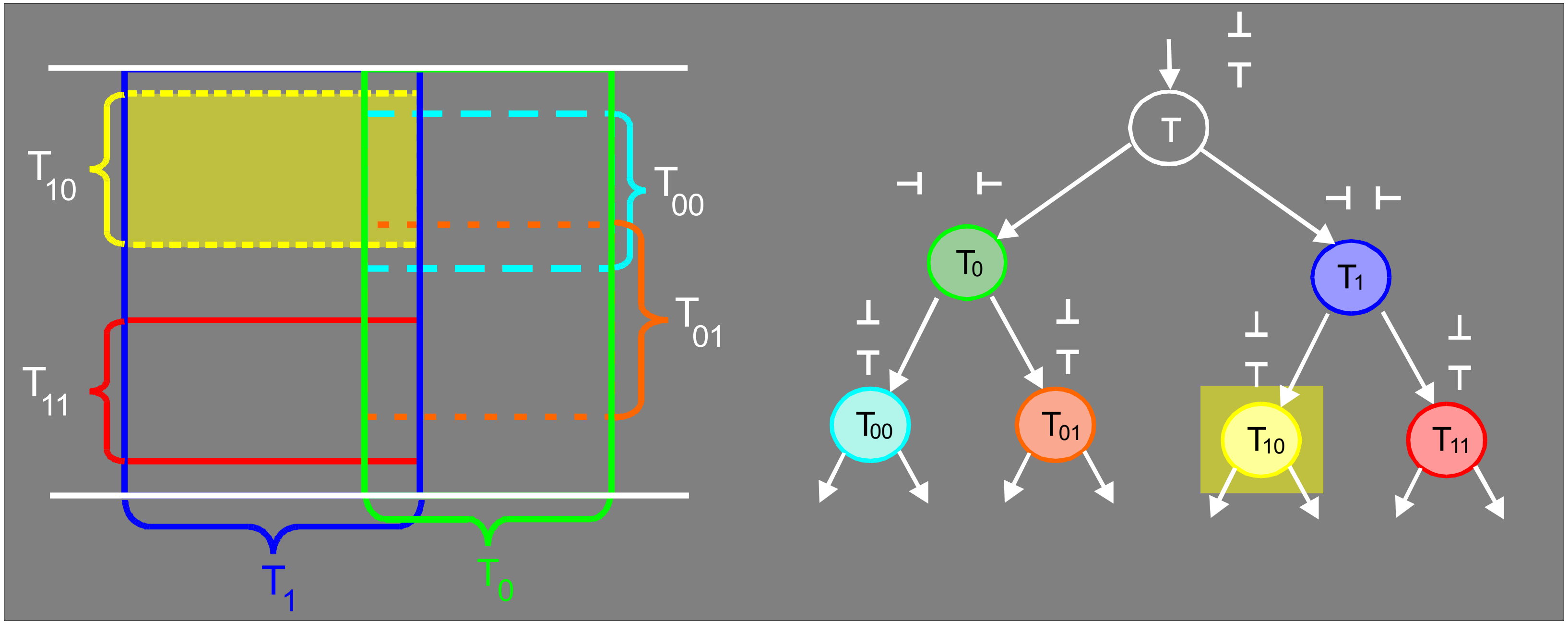
B-KD Tree Subdivision

- **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



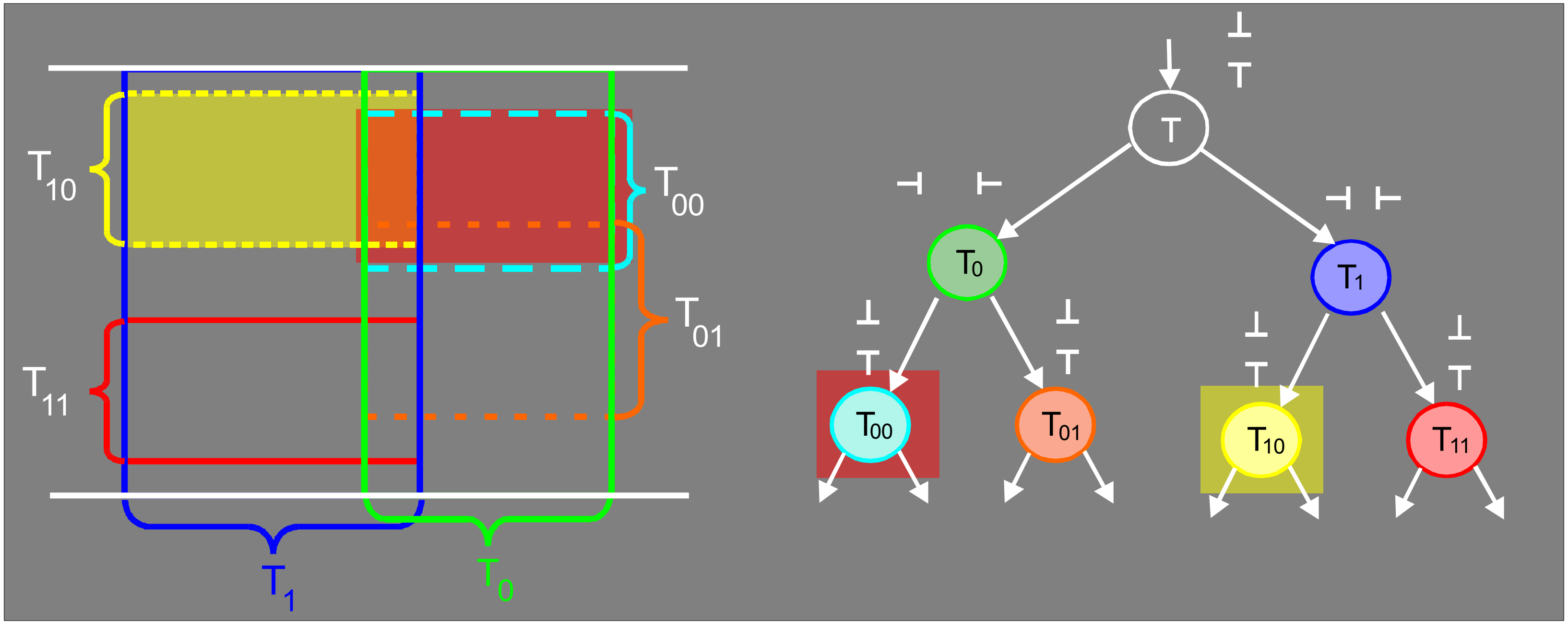
B-KD Tree Subdivision

- **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



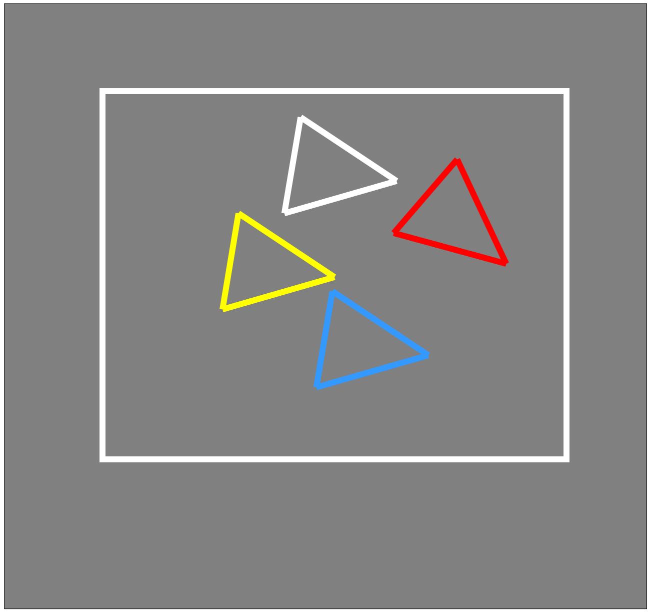
B-KD Tree Subdivision

- **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



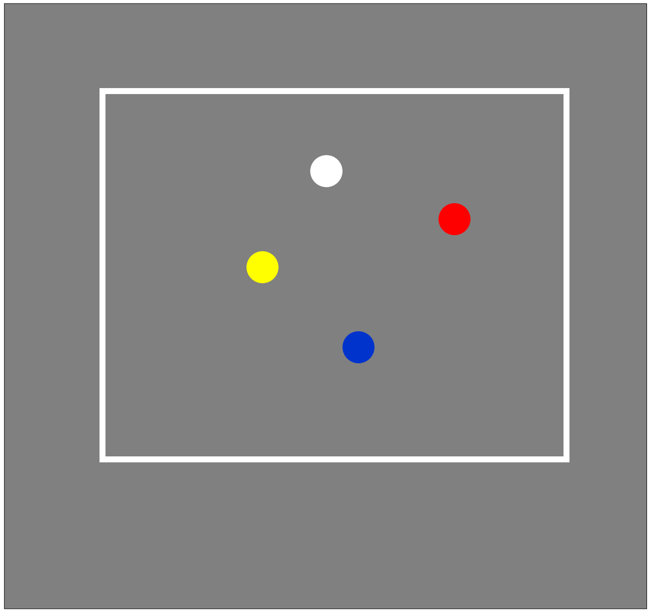
B-KD Tree Construction

- **If #primitives > 1 then**



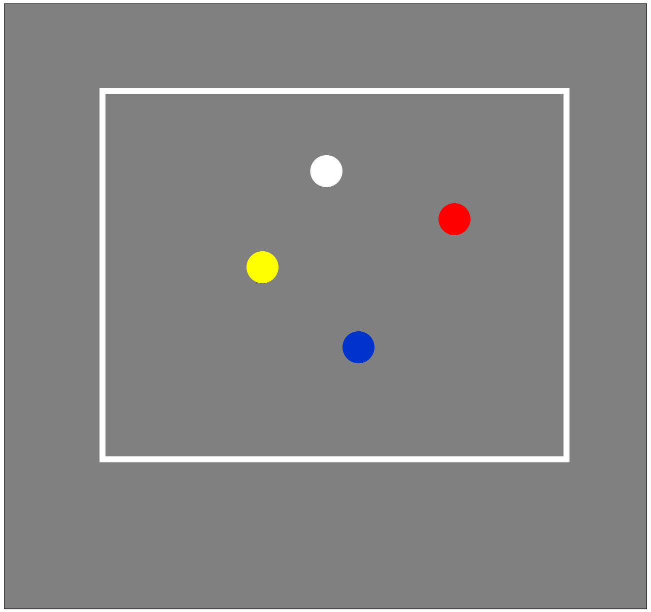
B-KD Tree Construction

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



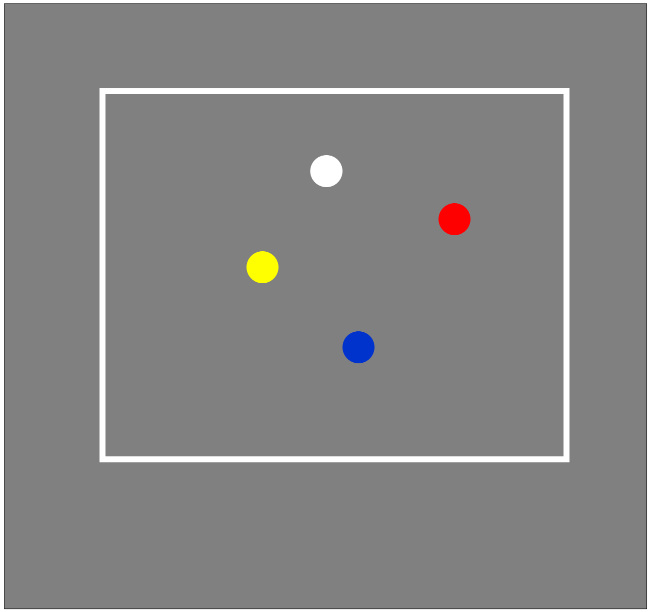
B-KD Tree Construction

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



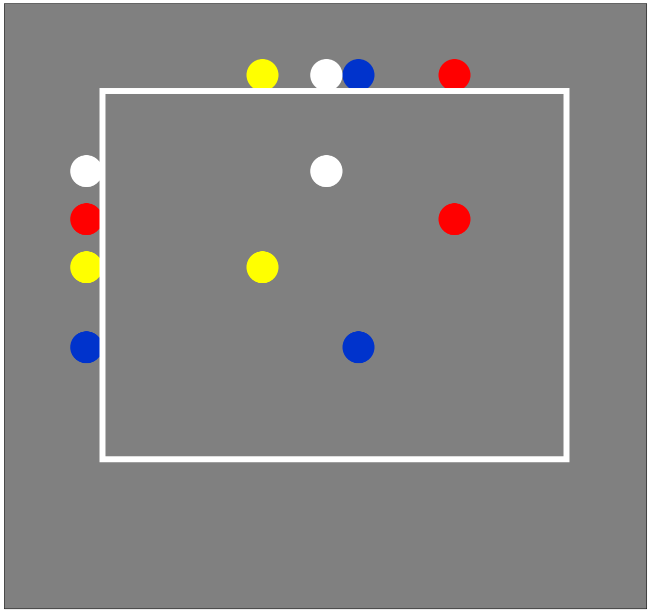
B-KD Tree Construction

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



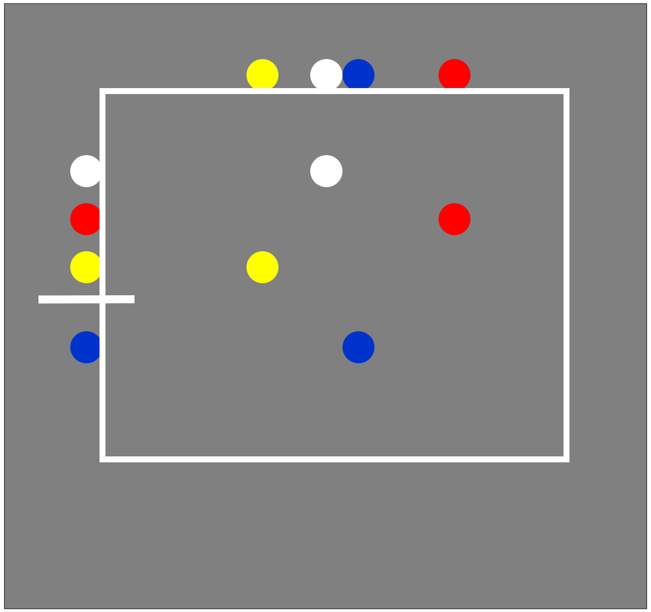
B-KD Tree Construction

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



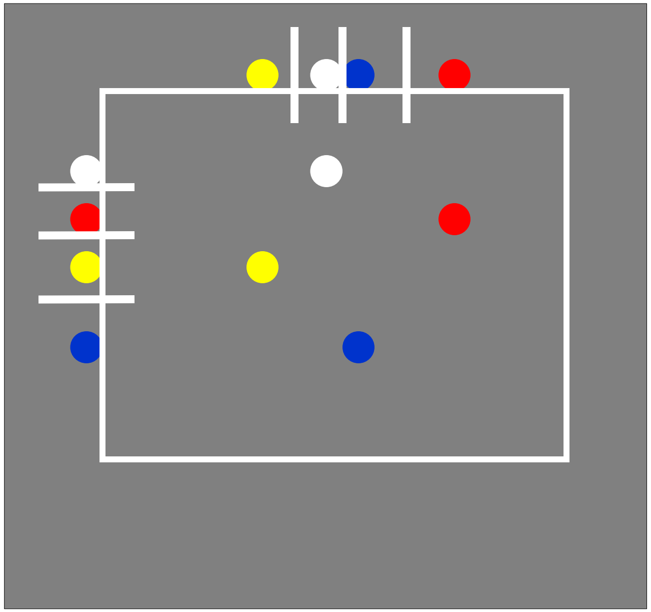
B-KD Tree Construction

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



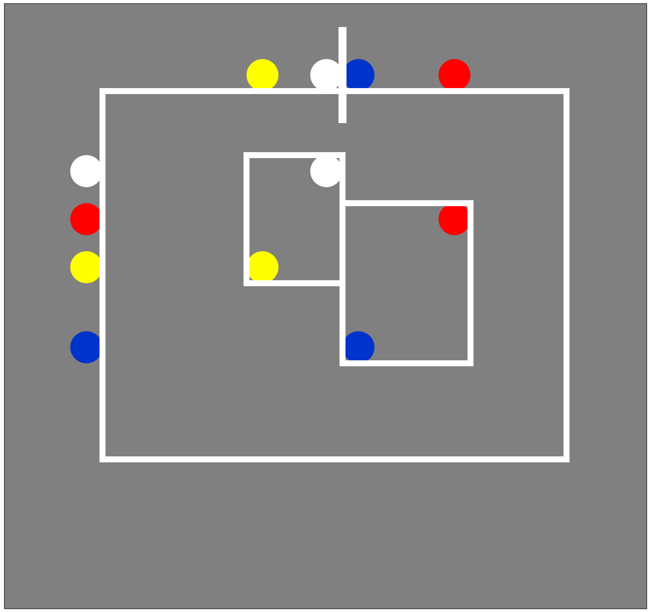
B-KD Tree Construction

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



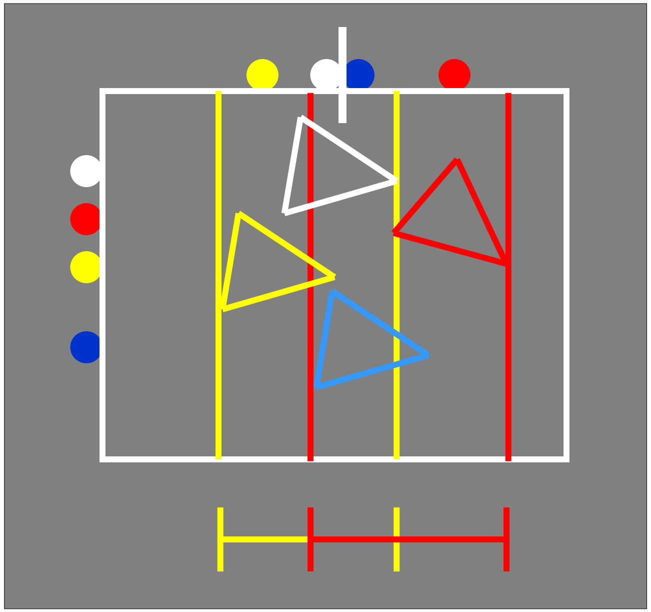
B-KD Tree Construction

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

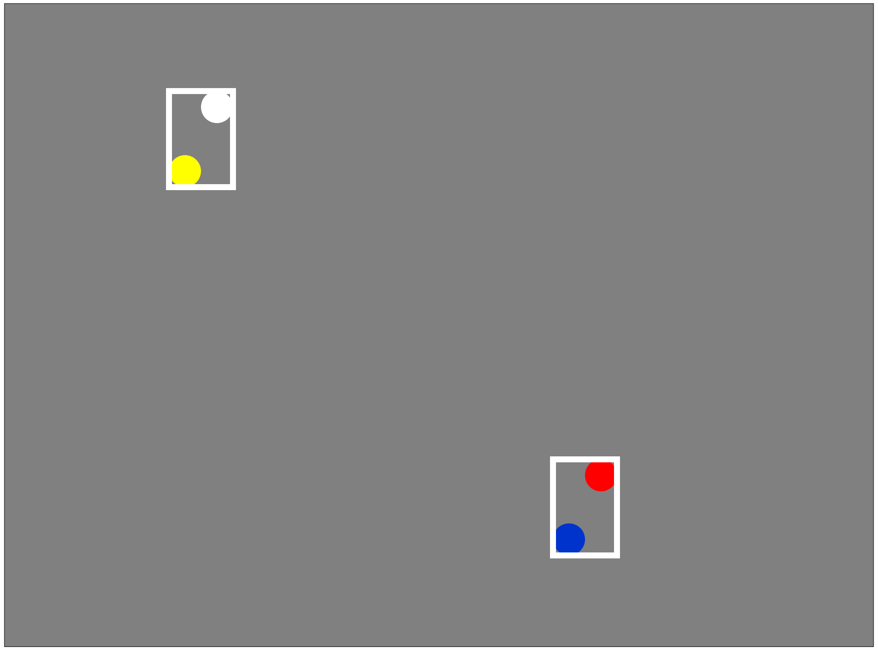
B-KD Tree Construction

- **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

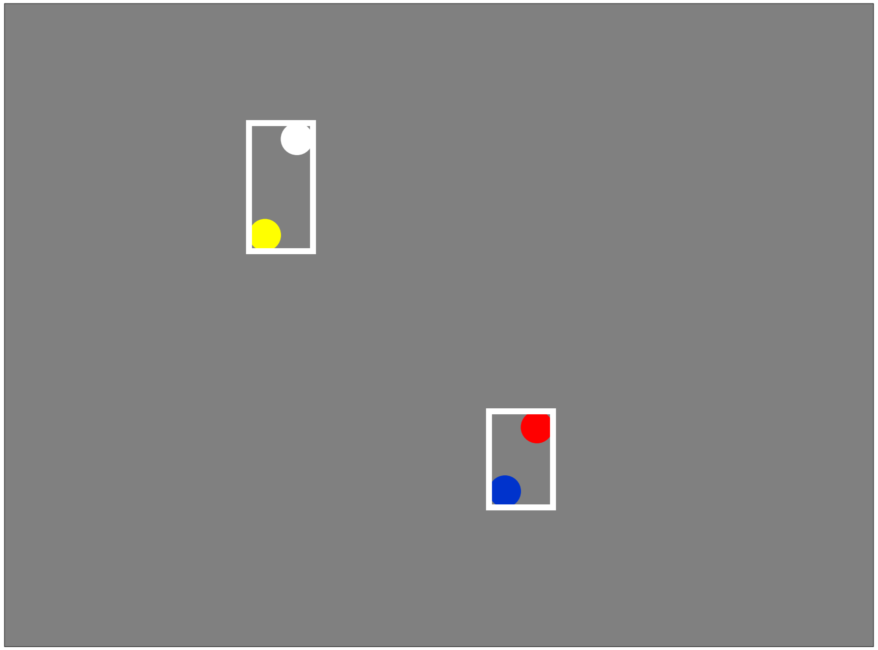
- **On changed object geometry**
  - B-KD tree bounds are updated from bottom up
  - B-KD tree structure remains constant
  - Linear updating complexity

Examples



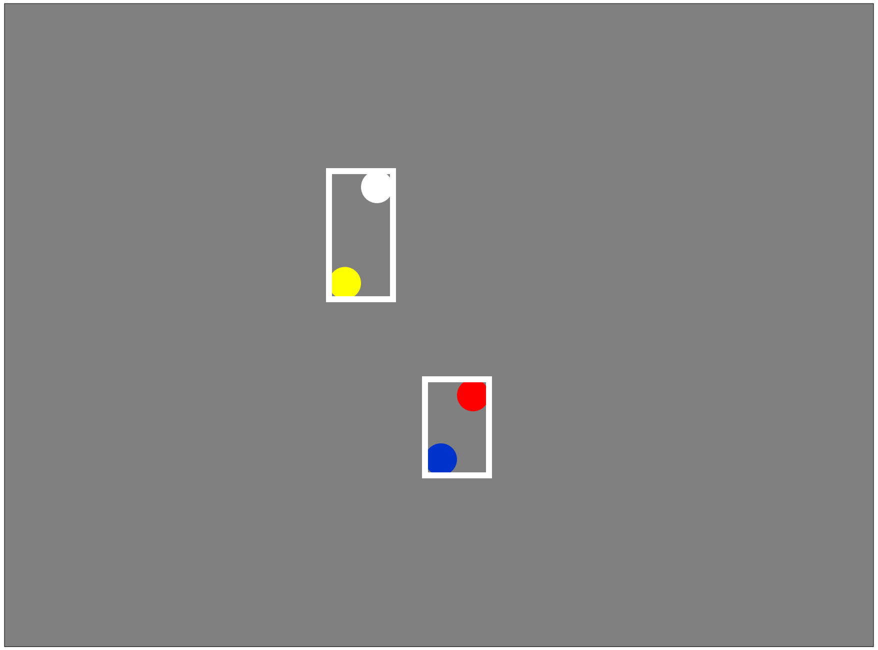
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



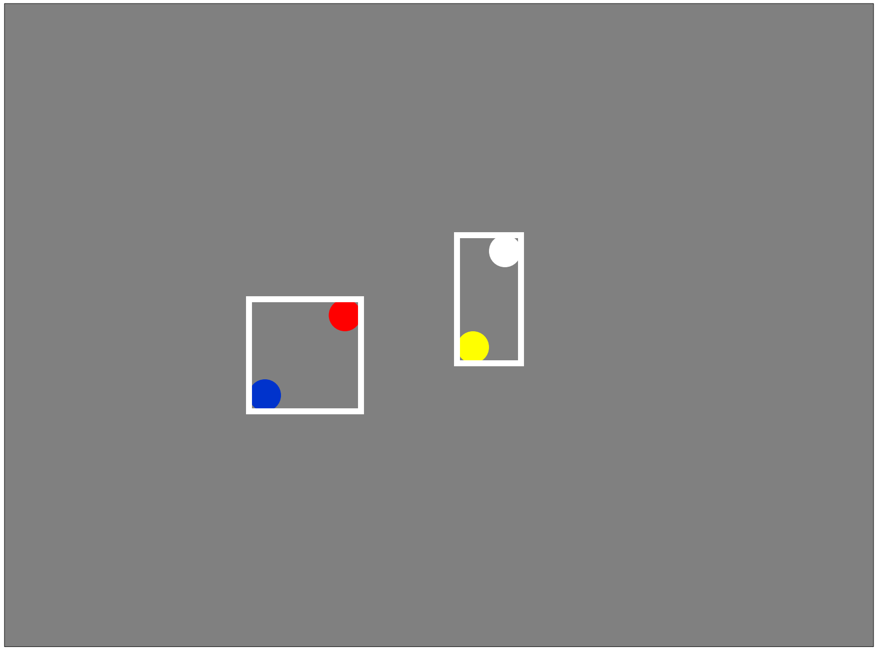
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



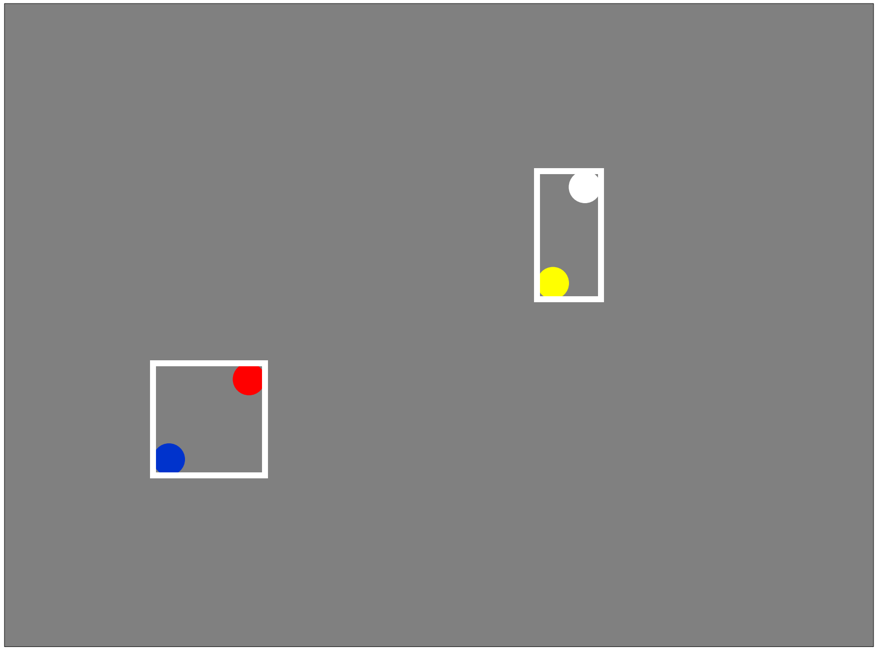
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



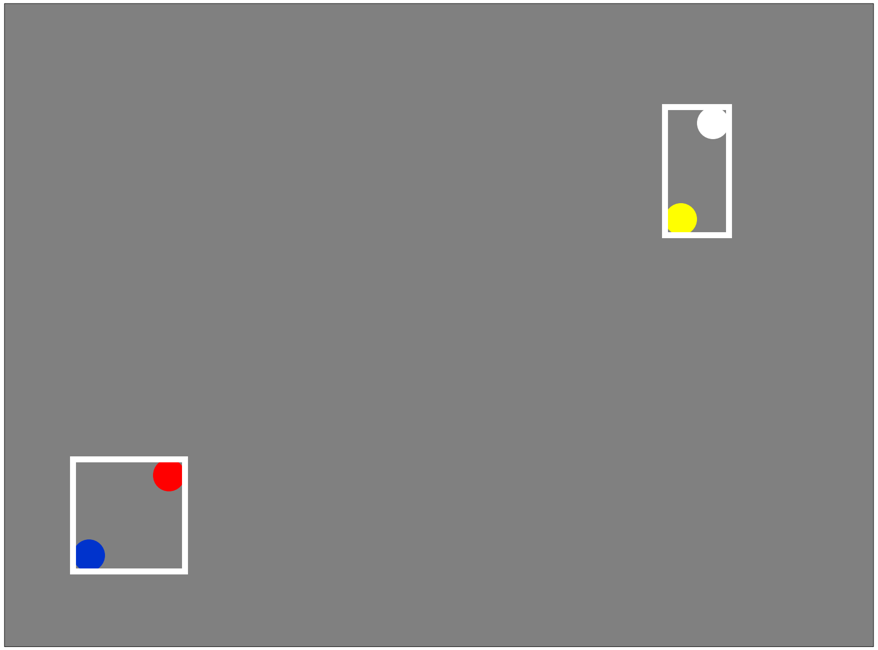
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



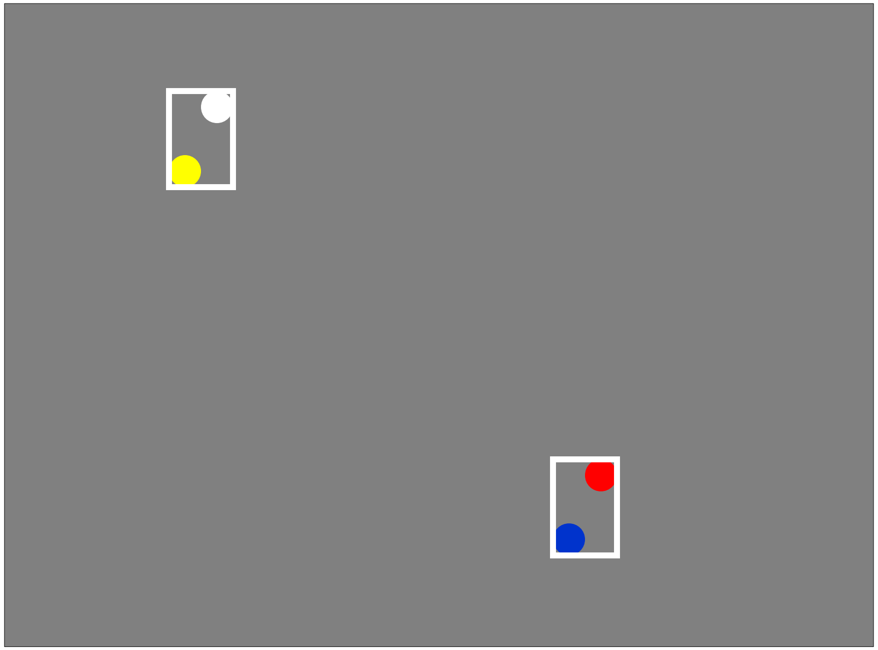
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



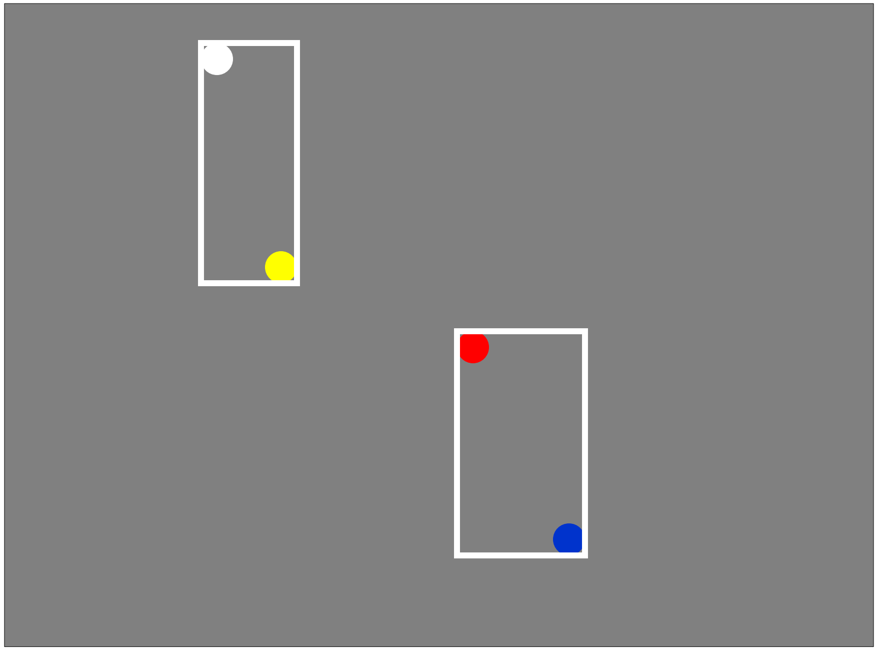
- **Bounding approaches perform well for**
  - Continuos motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



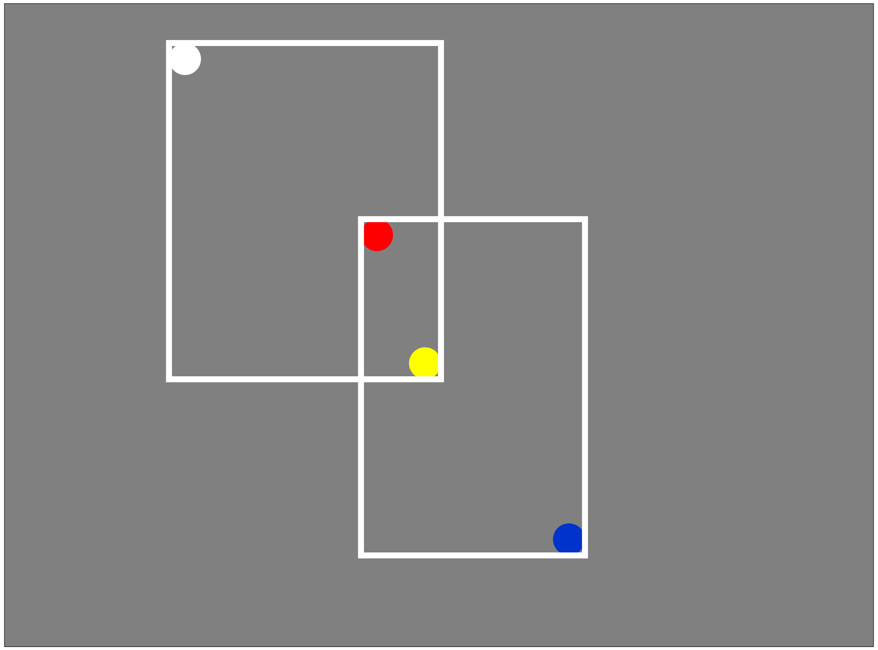
- **Bounding volume approaches are less efficient for**
  - Non-continuos motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

Examples



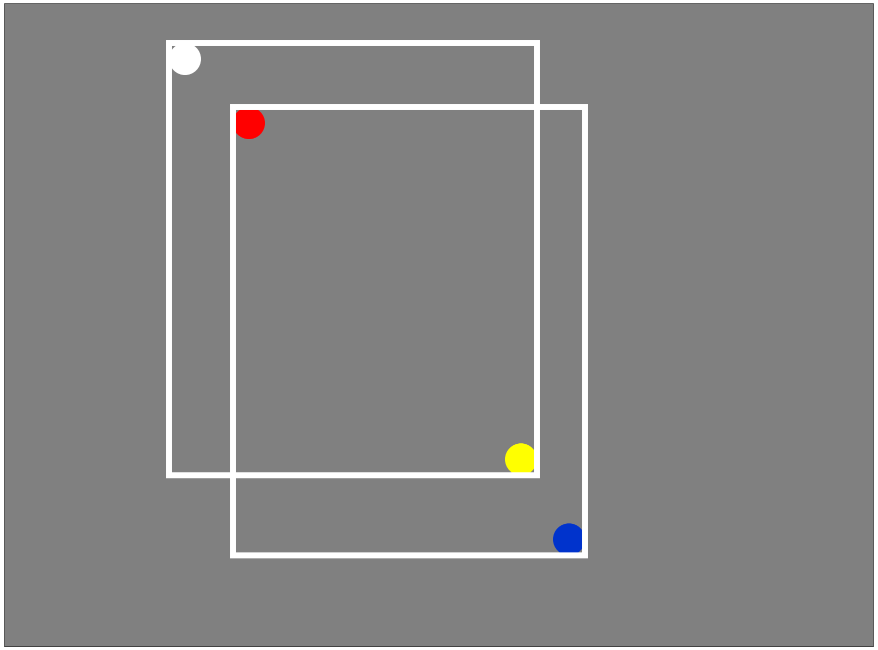
- **Bounding volume approaches fail for**
  - Non-continuos motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

Examples



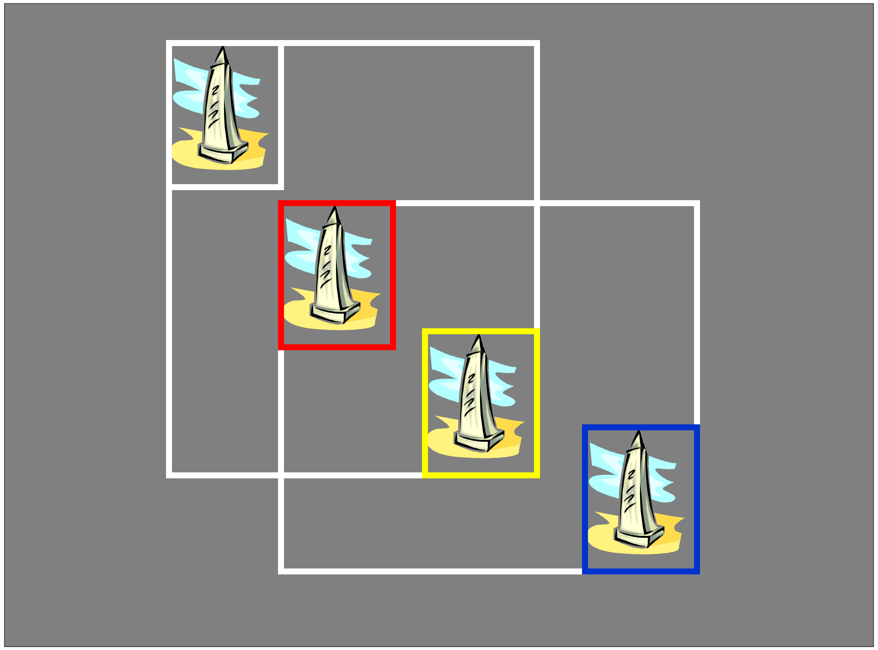
- **Bounding volume approaches fail for**
  - Non-continuos motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

Examples



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

Examples



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

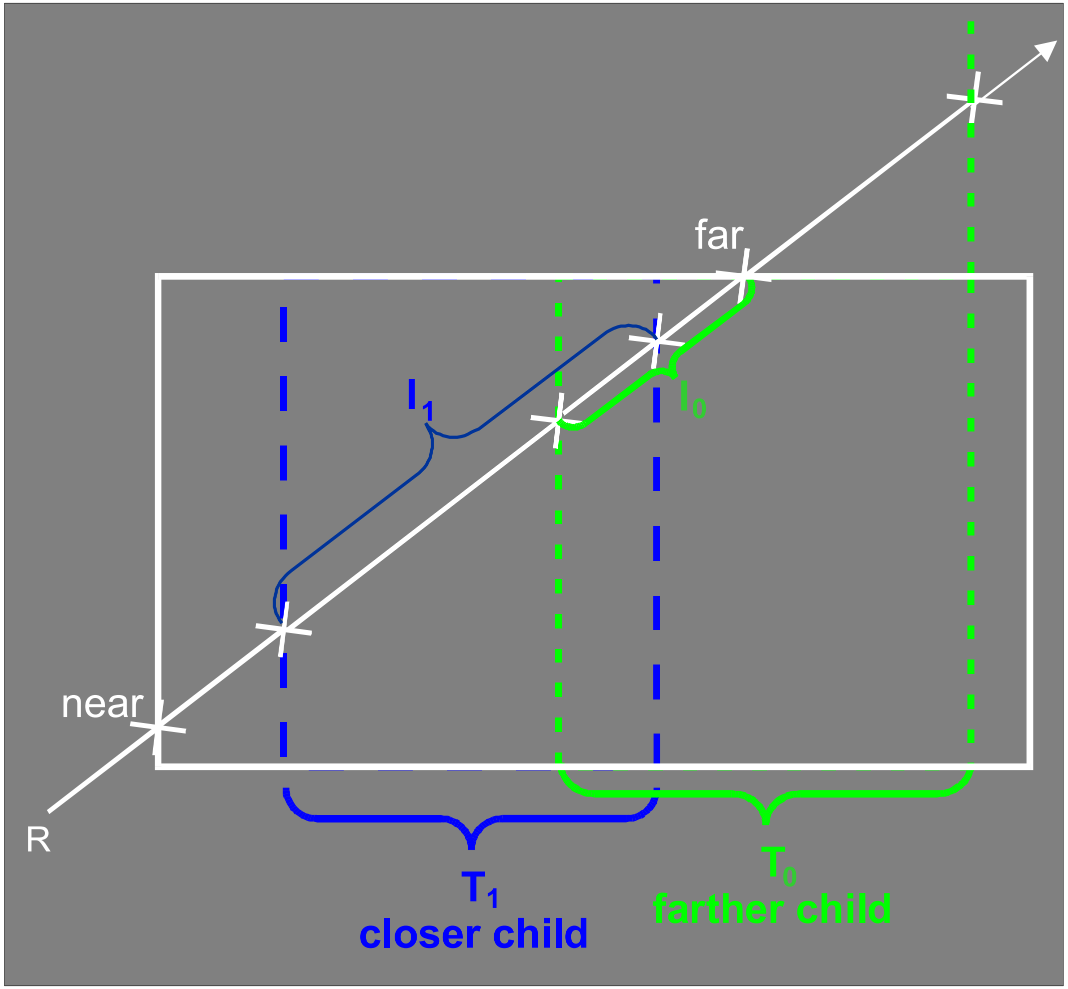
Traversal of B-KD Trees

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

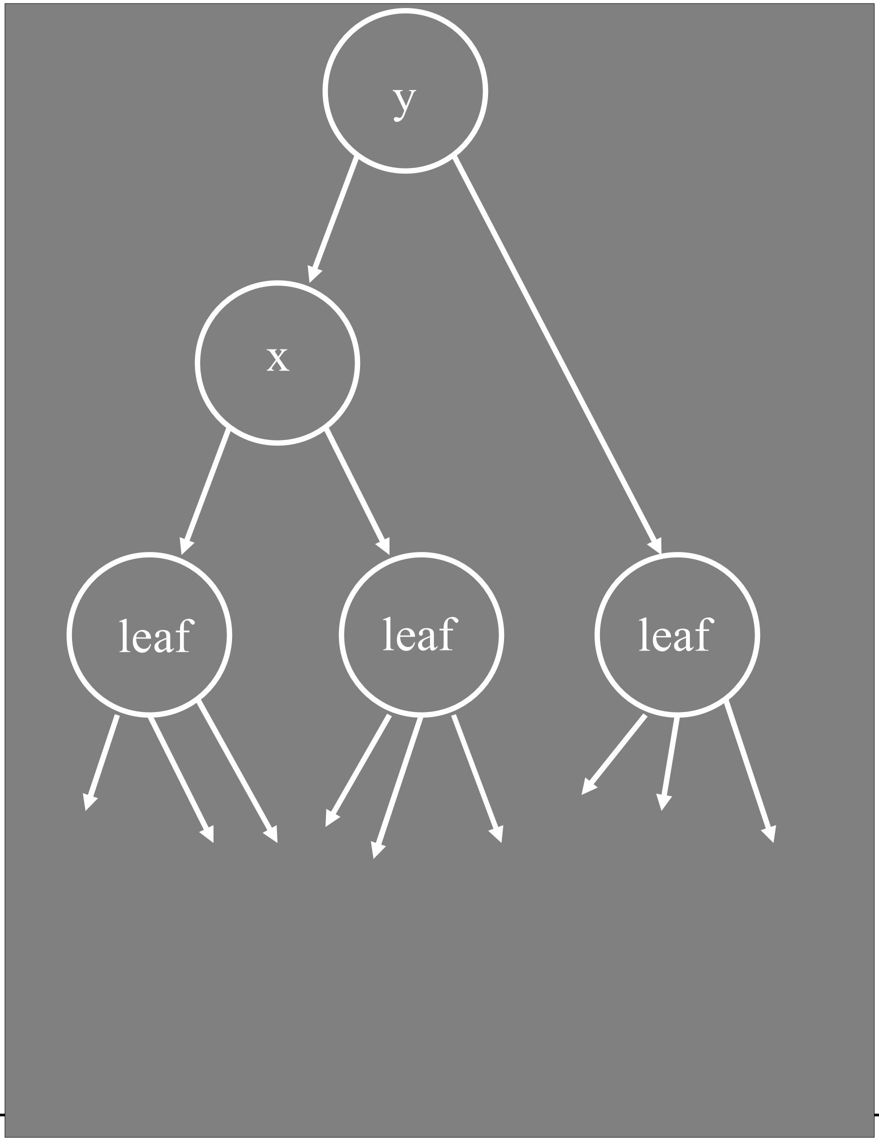
Complexity

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



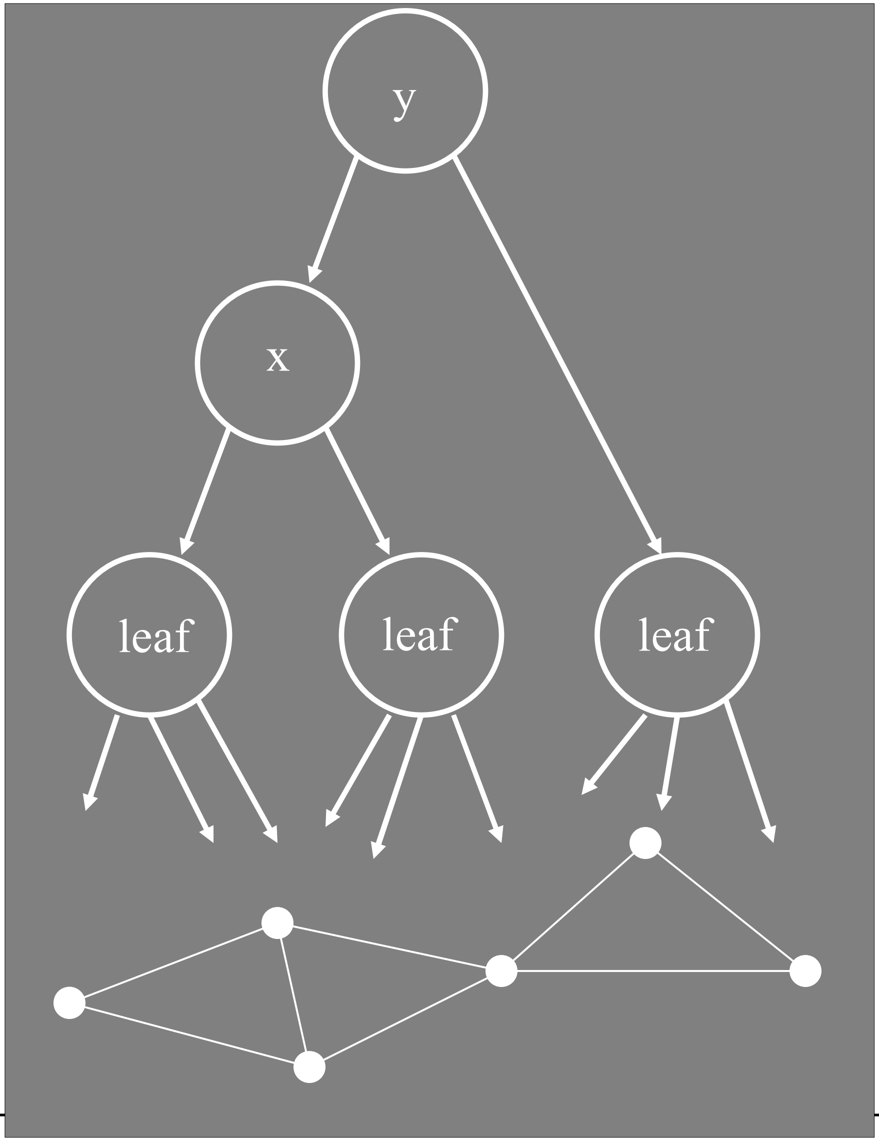
Update of B-KD Trees

- **Leaf Node**



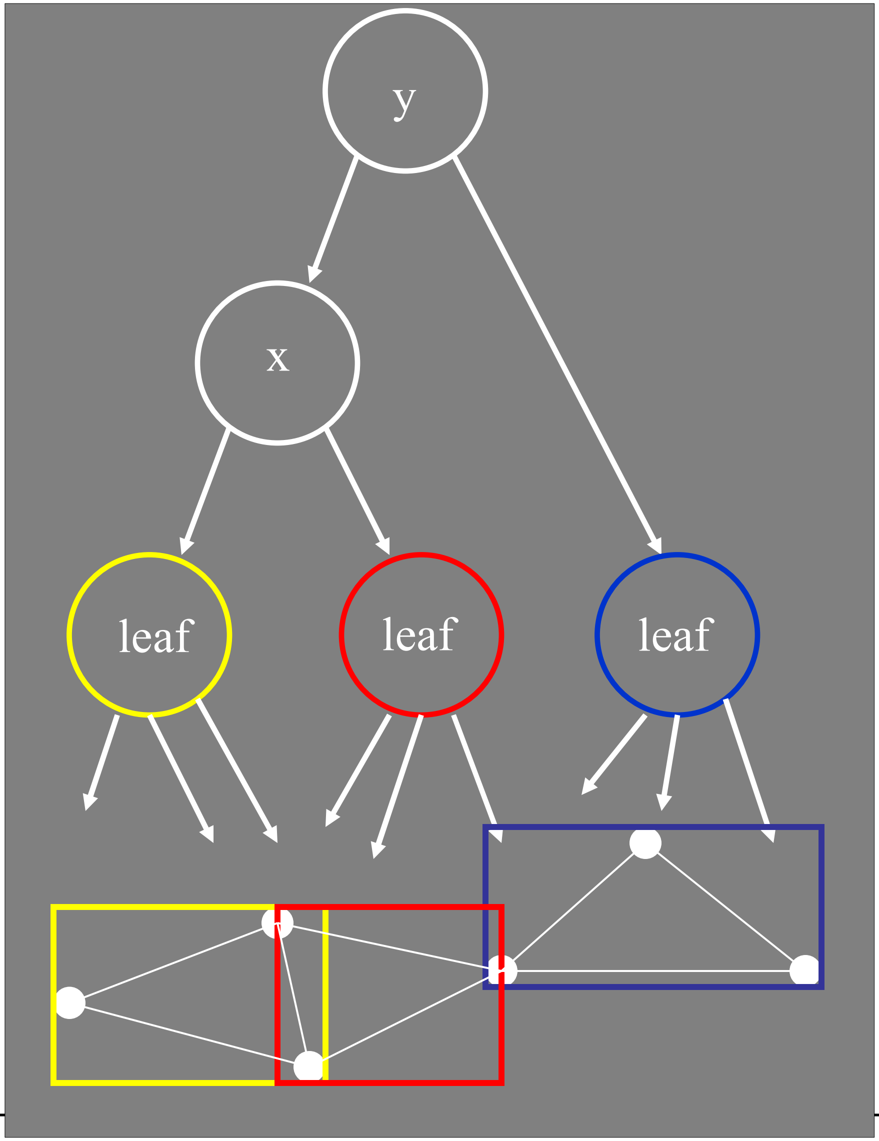
Update of B-KD Trees

- **Leaf Node**
  - Fetch vertices



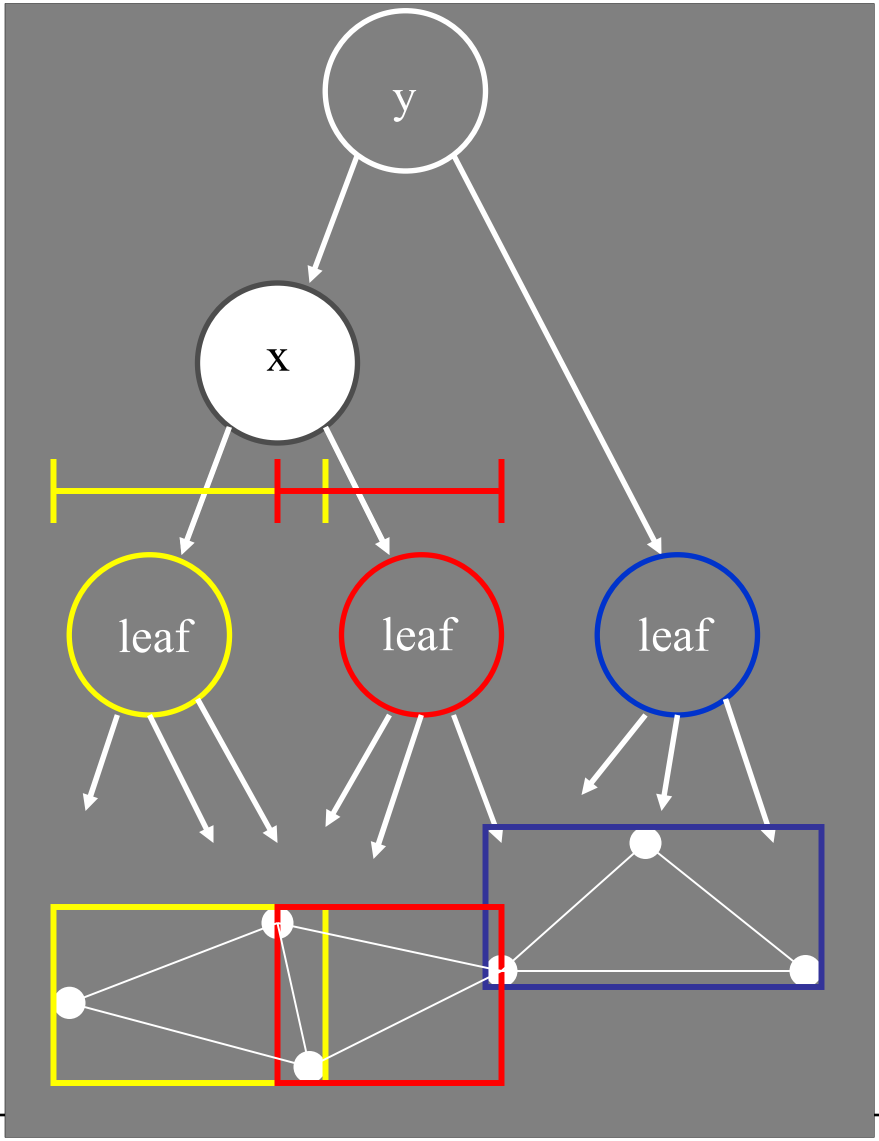
Update of B-KD Trees

- **Leaf Node**
  - Fetch vertices
  - Compute leaf boxes



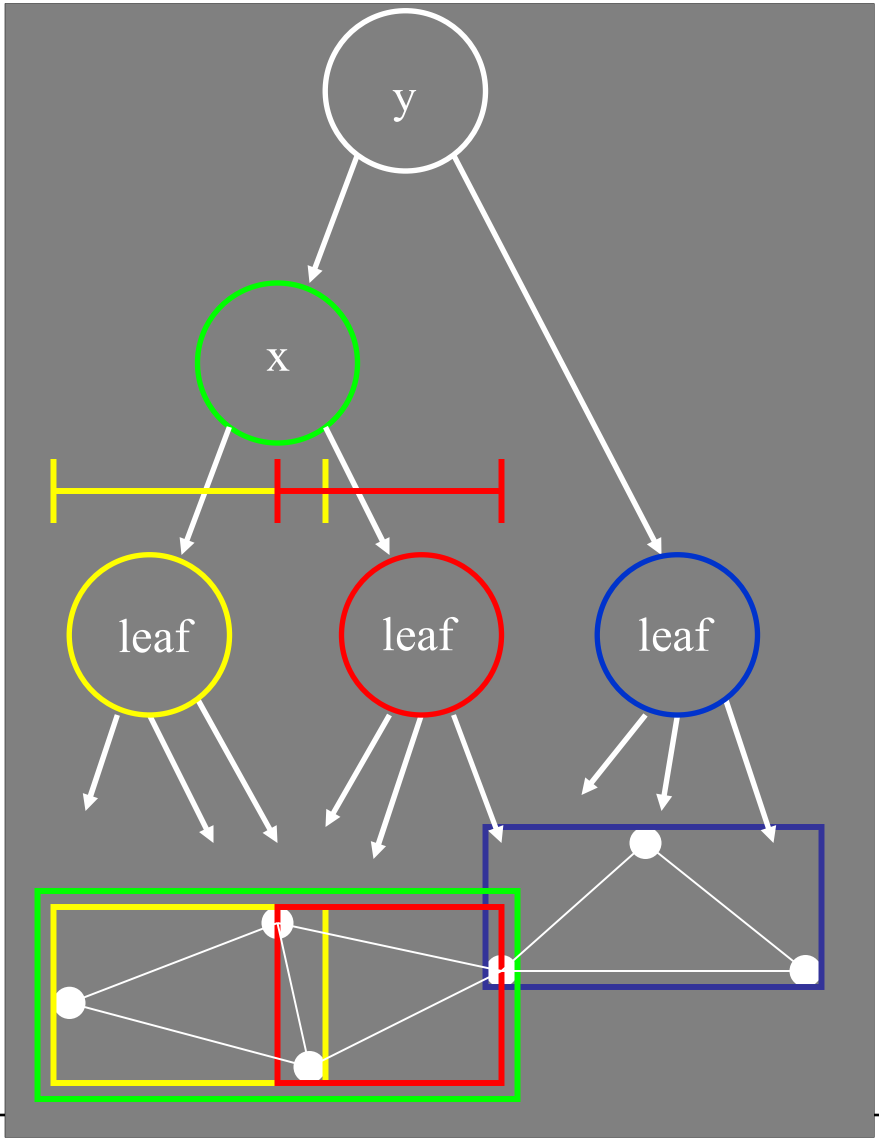
Update of B-KD Trees

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



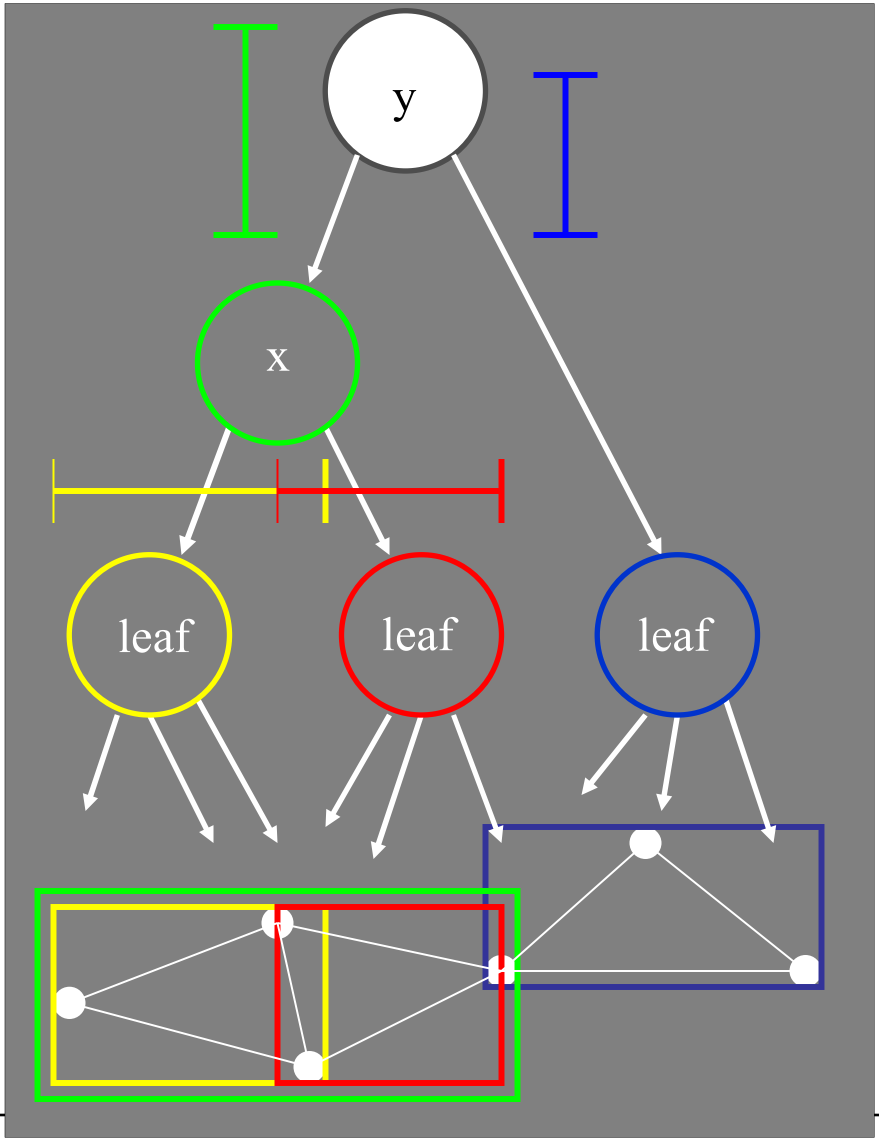
Update of B-KD Trees

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



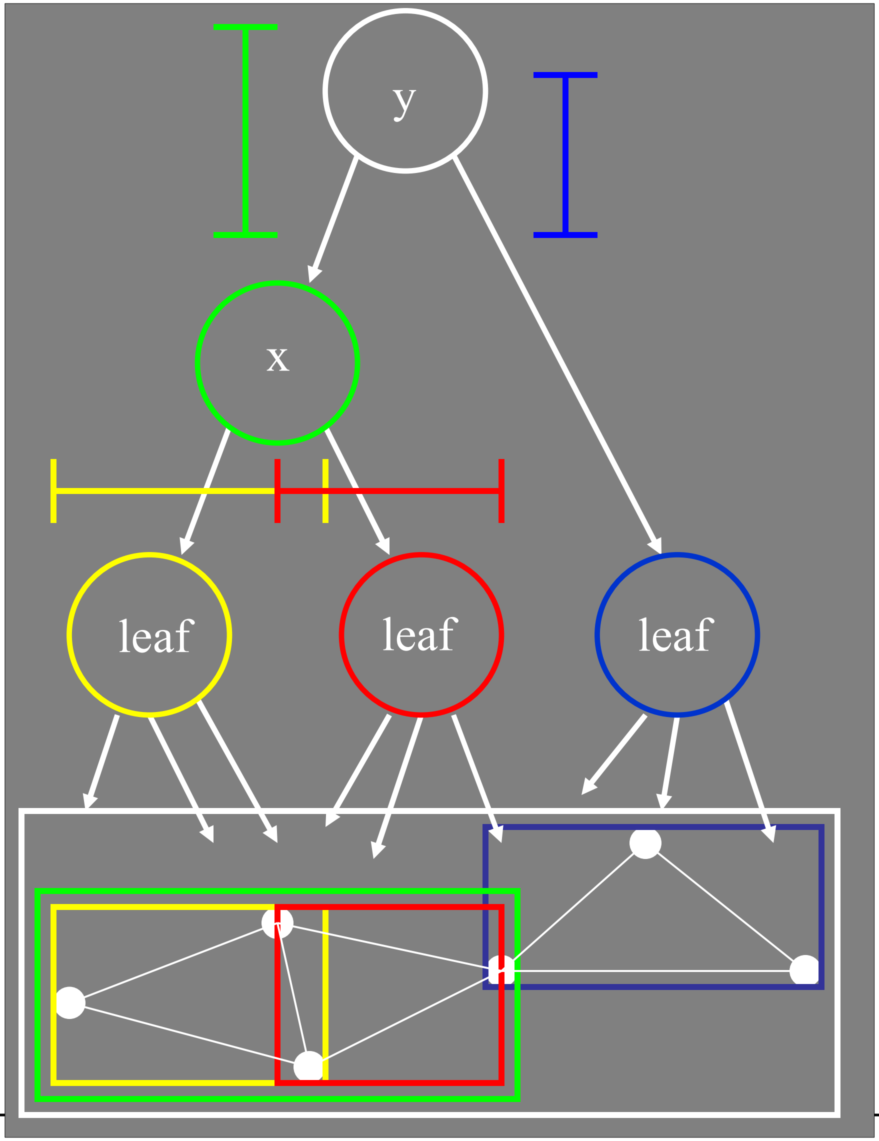
Update of B-KD Trees

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



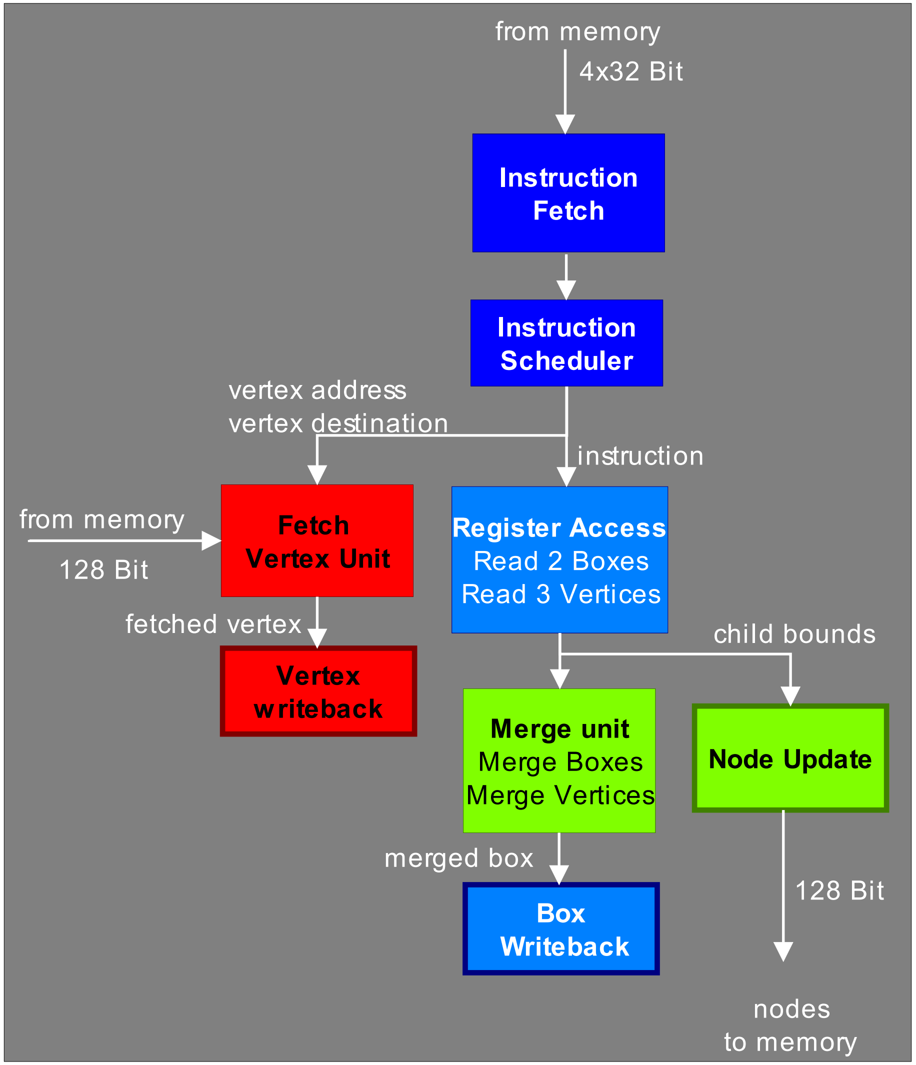
Update of B-KD Trees

- **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 stream
  - Vertices are accessed
    as sequential as possible



Prototype Implementation

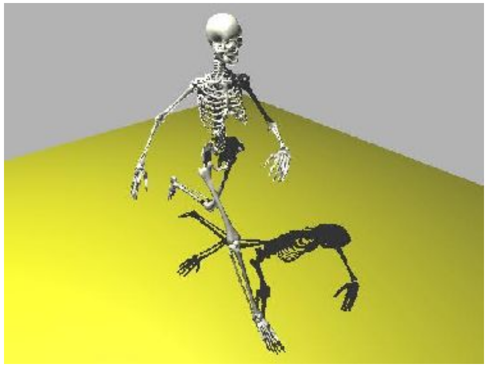
- **Hardware**
  - FPGA board from Alpha Data
  - Xilinx Virtex4 LX160
  - 128 MB DDR Memory

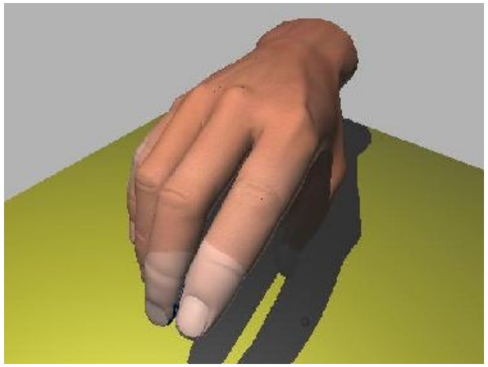
- **Implementation**
  - Packets of 4 rays
  - 32 packets of rays
  - 24 bit floating point
  - 66 MHz

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

- **80-90% of the dynamic scene problem has been solved**
  - But still much more work required
- **Partitioning is always a good idea**
- **Bounding volume approaches are useful for updating**
  - Avoids costly changes of tree structure
- **Open questions:**
  - What about random motion?
  - Will fast complete rebuilds be the ultimate solution?
  - How well will lazy building work?
- **All of this probably depends on the scenes!**