EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

FACULTY OF
SCIENCE
Computer Graphics

Computer Graphics (Graphische Datenverarbeitung)

- Ray Tracing III -

WS 2021/2022

Corona

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Regular random lookup of the 3G certificates

- Contact tracing: We need to know who is in the class room
  - New ILIAS group for every lecture slot
  - Register via ILIAS or this QR code (only if you are present in this room)

Overview

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Last lecture
  - spatial acceleration structures
  - Grids, Octrees, BVH, KD-Tree
  - static scenes only

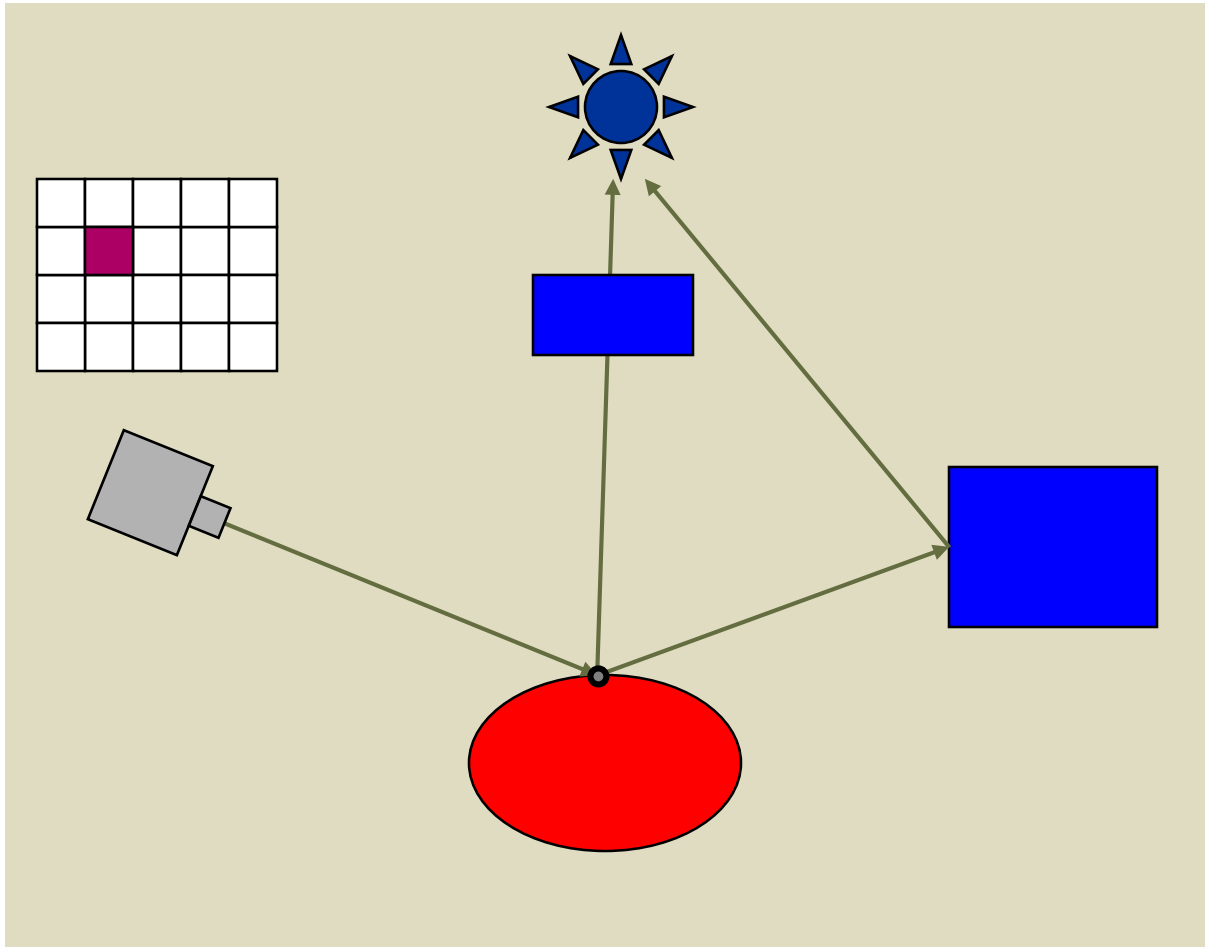
- Today
  - large scenes
  - updates for dynamic scenes

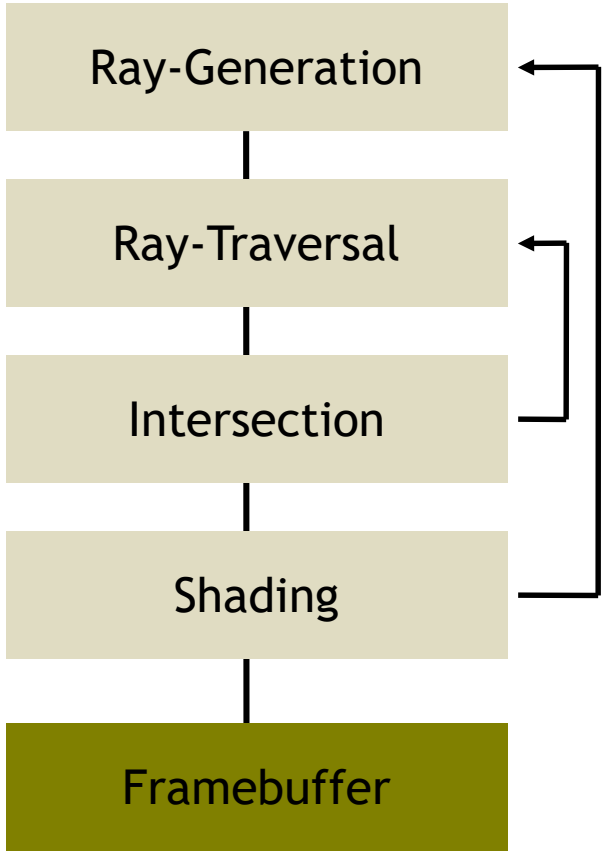
Ray Tracing Steps (repetition)

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Generation of primary rays
  - Rays from viewpoint along viewing directions into 3D scene
  - (At least) one ray per picture element (pixel)
- Ray tracing
  - Traversal of spatial index structures
  - Intersection of ray with scene geometry
- Shading
  - From intersection, determine “light color” sent along primary ray
  - Determines “pixel color”
  - Needed
    - Local material color and reflection properties
      - Object texture
    - Local illumination of intersection point
      - Can be hard to determine correctly

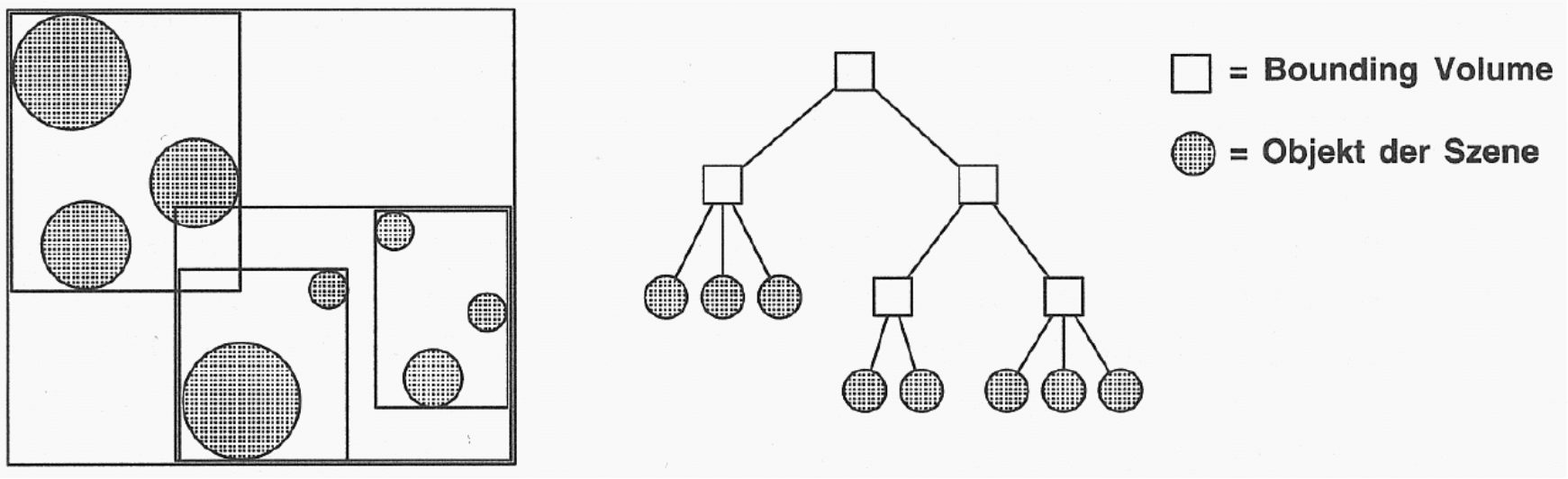
Ray Tracing Pipeline





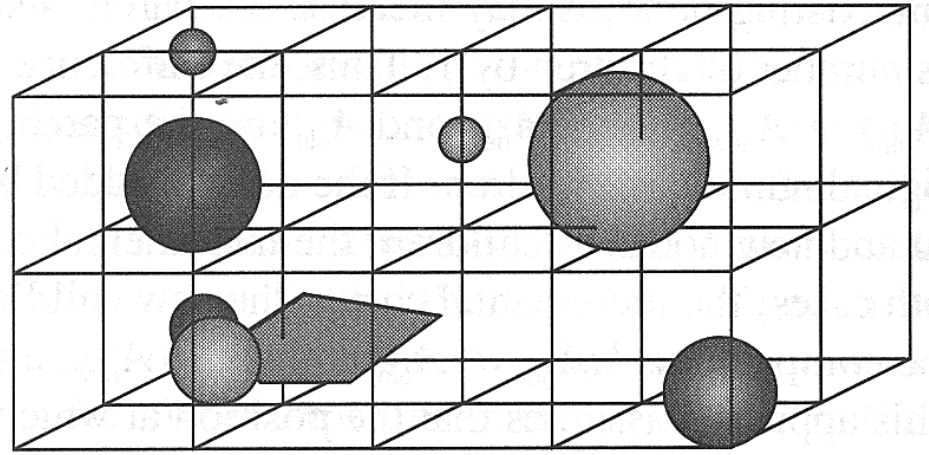
Bounding Volume Hierarchies

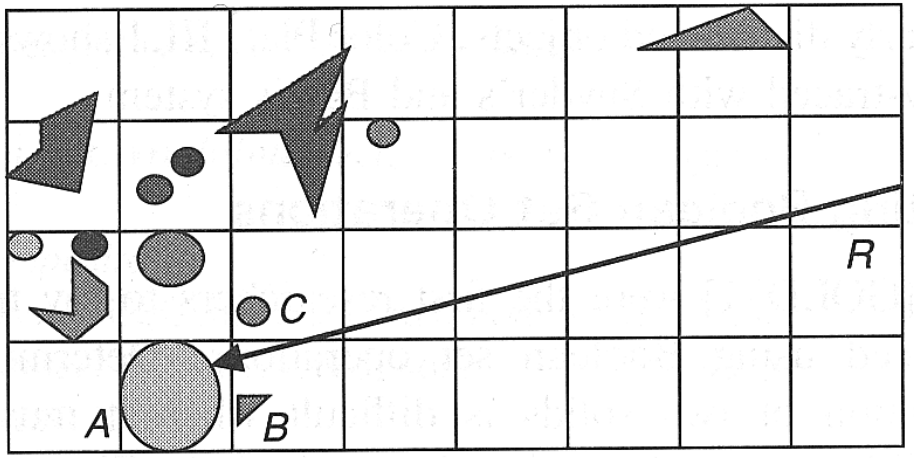
- Idea:
  - Organize bounding volumes hierarchically into new BVs
- Advantages:
  - Very good adaptivity
  - Efficient traversal O(log N)
  - Often used in ray tracing systems
- Problems
  - How to arrange BVs?



Grid

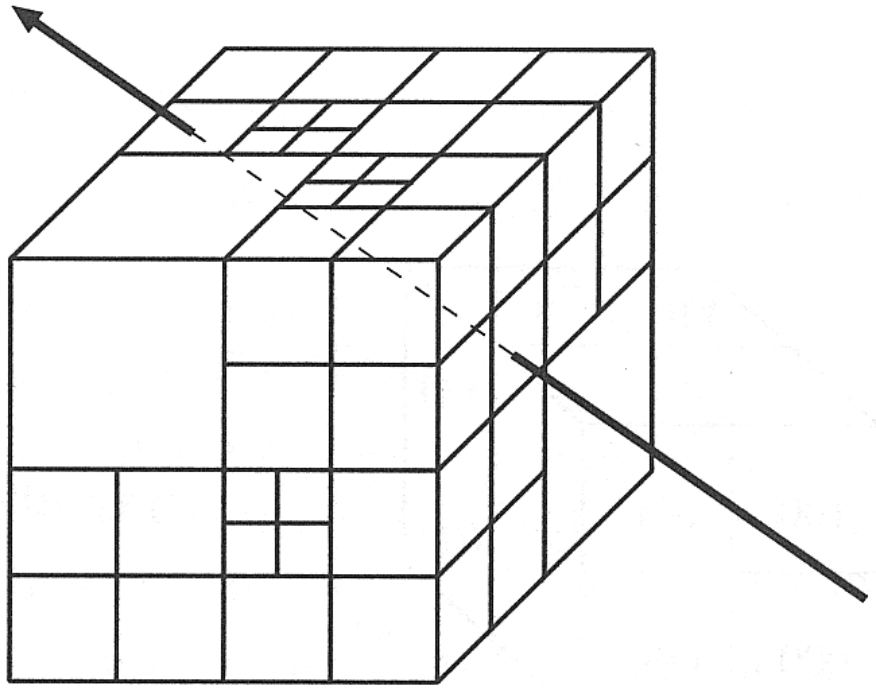
- Grid
  - Partitioning with equal, fixed sized „voxels“
- Building a grid structure
  - Partition the bounding box (bb)
  - Resolution: often $\sqrt[3]{n}$
  - Inserting objects
    - Trivial: insert into all voxels overlapping objects bounding box
    - Easily optimized
- Traversal
  - Iterate through all voxels in order as pierced by the ray
  - Compute intersection with objects in each voxel
  - Stop if intersection found in current voxel





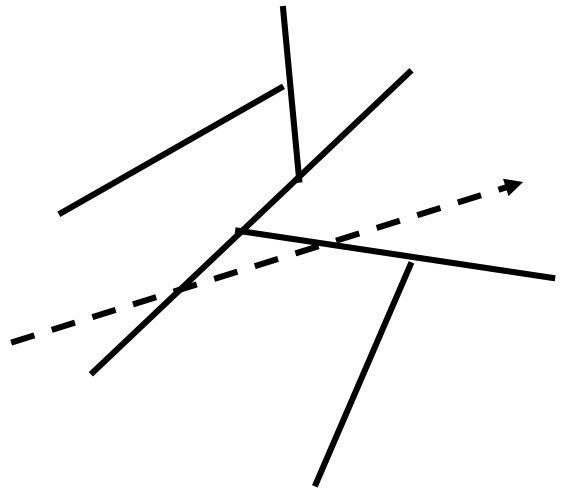
Octree

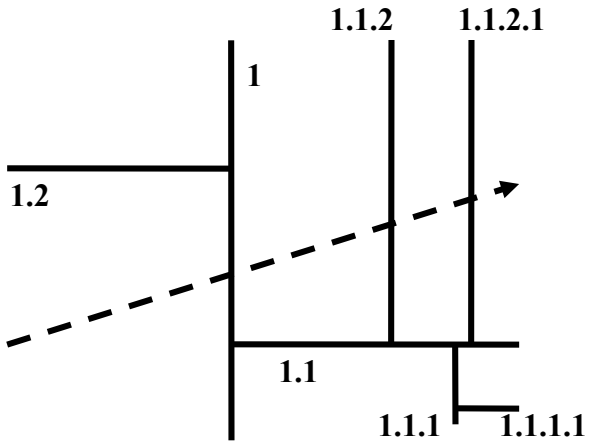
- Hierarchical space partitioning
  - Start with bounding box of entire scene
  - Recursively subdivide voxels into 8 equal sub-voxels
  - Subdivision criteria:
    - Number of remaining primitives and maximum depth
  - Result in adaptive subdivision
    - Allows for large traversal steps in empty regions
- Problems
  - Pretty complex traversal algorithms
  - Slow to refine complex regions
- Traversal algorithms
  - HERO, SMART, ...
  - Or use kd-tree algorithm …



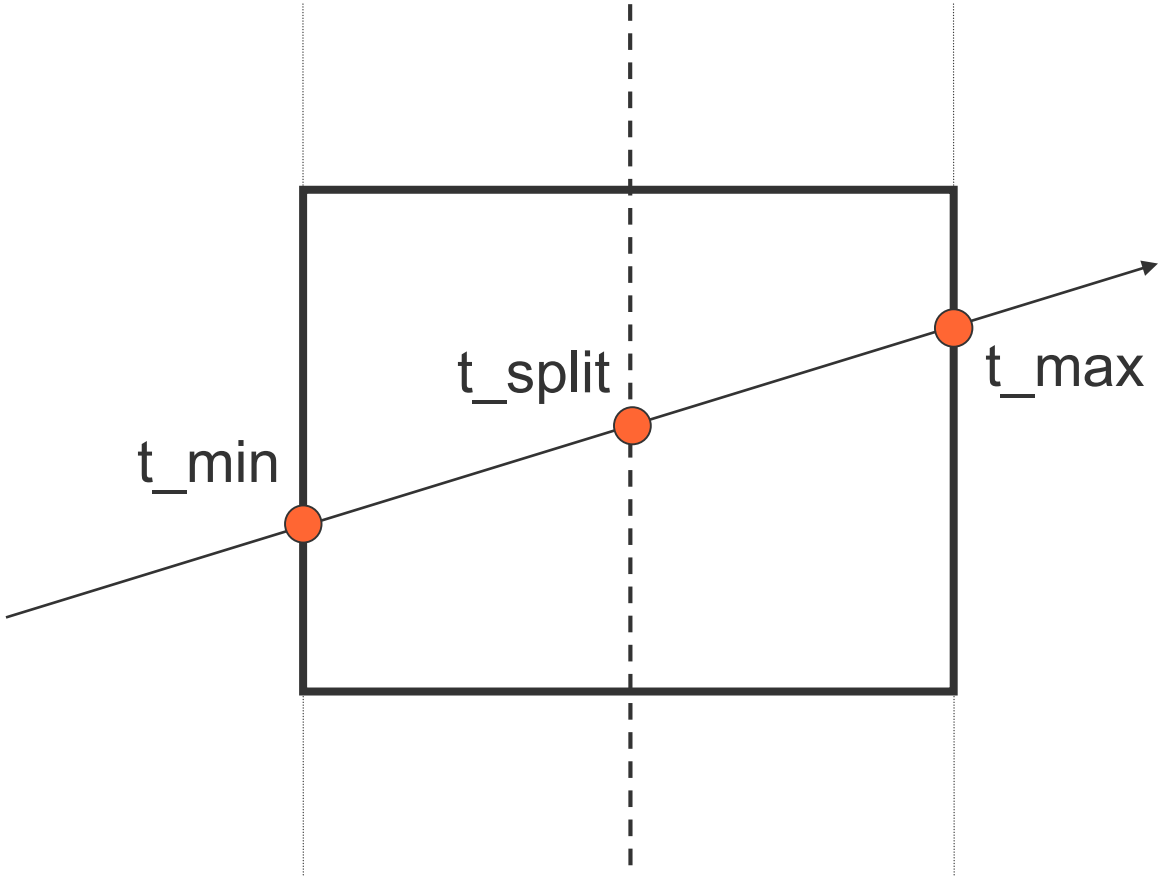
BSP- and kD-Trees

- Recursive space partitioning with half-spaces
- Binary Space Partition (BSP):
  - Recursively split space into halves
  - Splitting with half-spaces in arbitrary position
    - Often defined by existing polygons
  - Often used for visibility in games (→ Doom)
    - Traverse binary tree from front to back
- kD-Tree
  - Special case of BSP
    - Splitting with axis-aligned half-spaces
  - Defined recursively through nodes with
    - Axis-flag
    - Split location (1D)
    - Child pointer(s)
  - See following slides for details

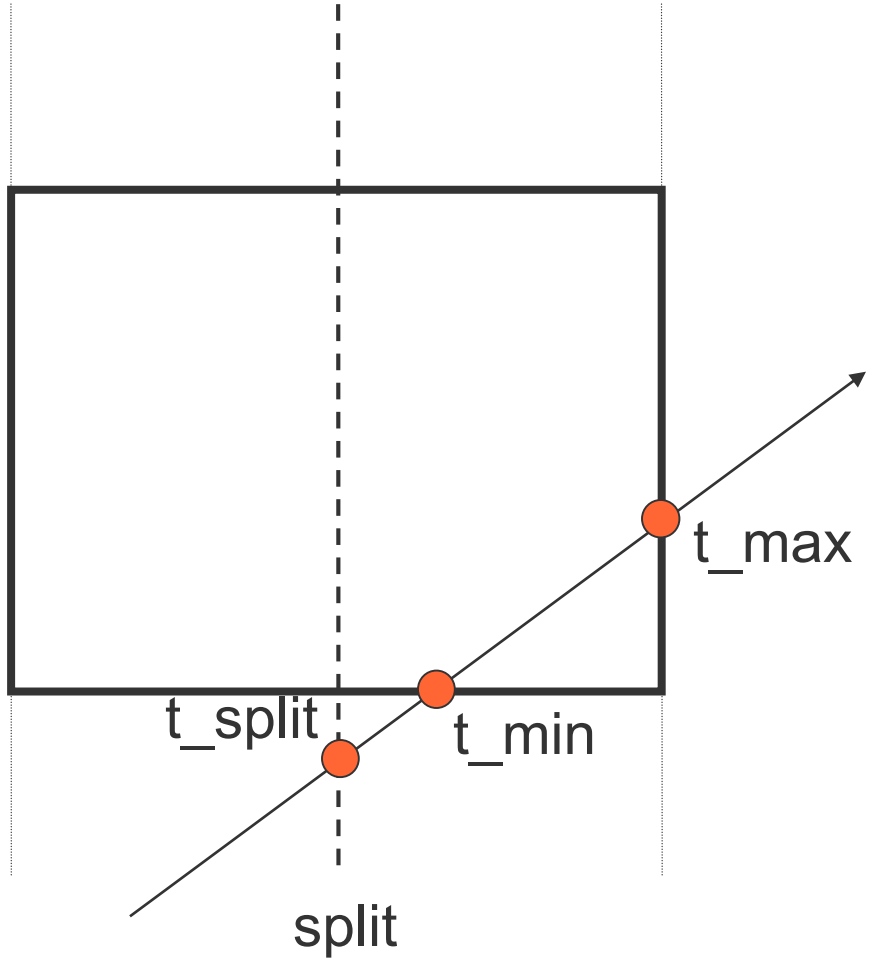




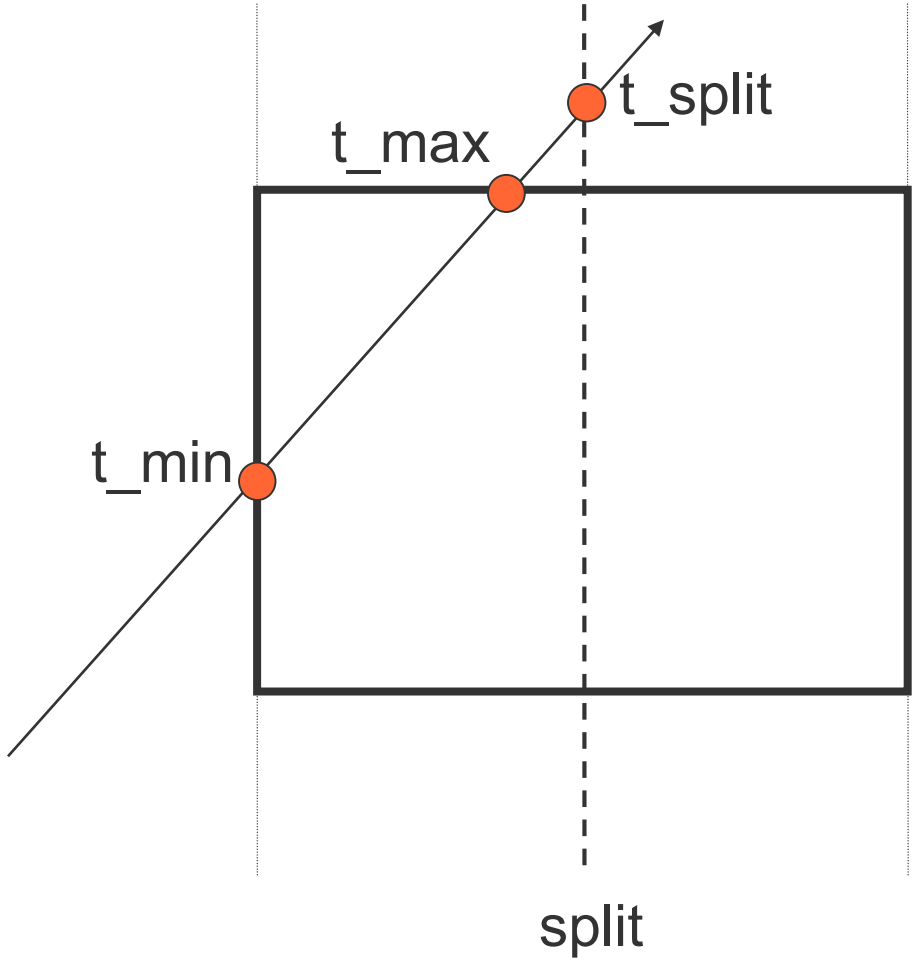
kD-Tree Traversal Step



kD-Tree Traversal Step



kD-Tree Traversal Step



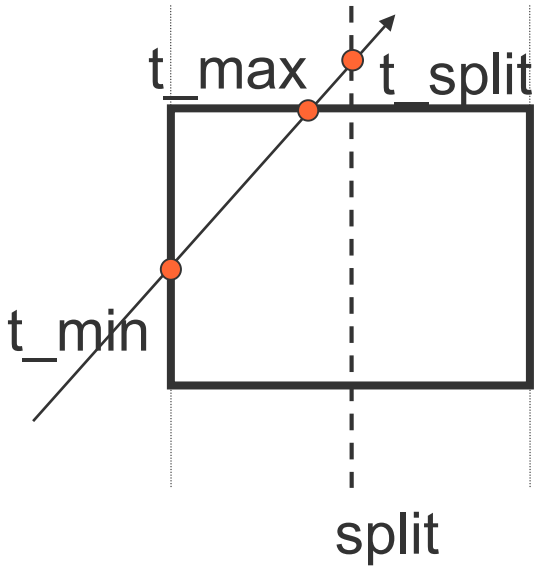
kD-Tree Traversal Step

```
ray: origin P,  direction V (normalized)
t_min, t_max (e.g. from bounding box of the entire scene)

Given:  ray P & V, t_min, t_max, split_location, split_axis

t_split = ( split_location - ray->P[split_axis] ) /
          ray->V[split_axis]

if t_split > t_min
 need to test against near child
If t_split < t_max
 need to test against far child
```

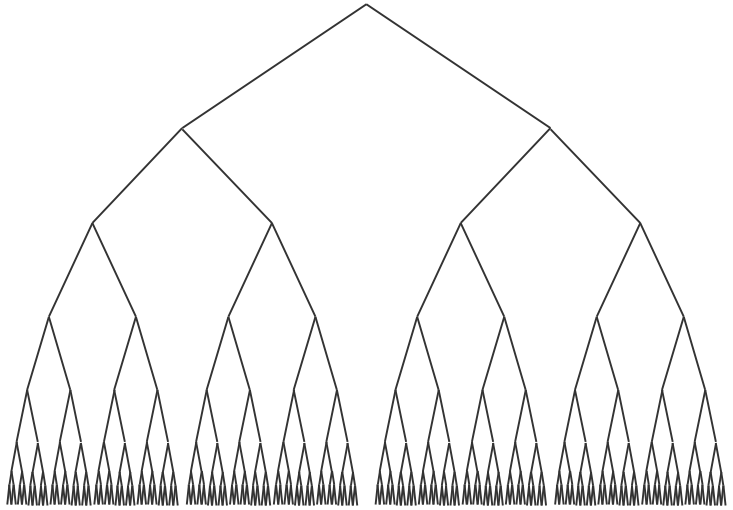


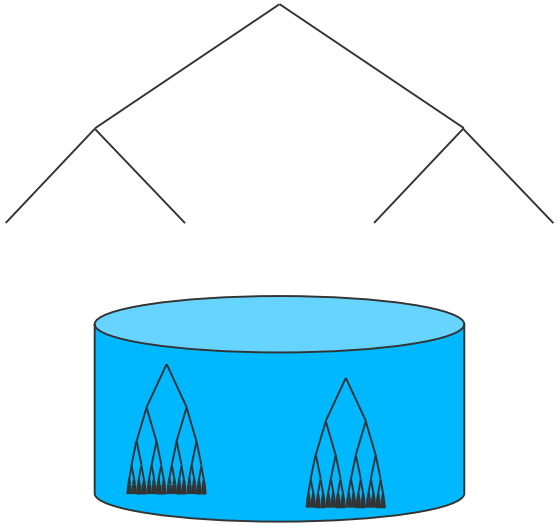
Large Scenes

What if the scene and the spatial data structure does not fit into memory?

Large Scenes

- What to do if the scene does not fit into memory?
- .. or should be rendered while efficiently reading from disc?





- Approaches:
  - Lazy build
  - Caching
  - Ray-reordering
  - Multi-level hierarchy

```
generateGeometry(ray, node) {
    triangleSoup = manyTriangles();
    return triangleSoup;
}
```

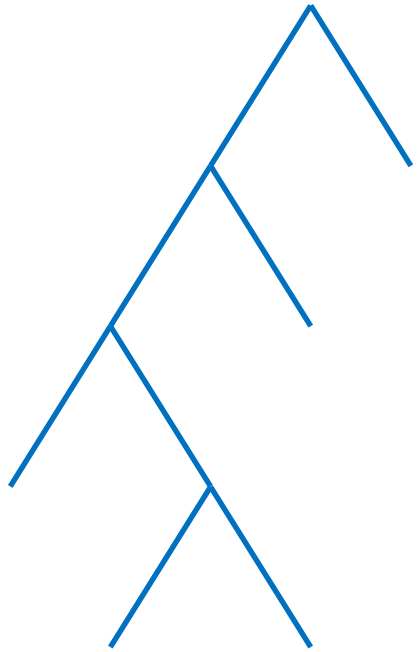
Lazy Building of Trees

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Build a sub-tree only when some ray may intersect it

- Naïve Algorithm:

```
bool intersect( ray, node ){
  if (hit( ray, node )) {
   split( node )
   return ( intersect( ray, child.left) ||
            intersect( ray, child.right) )
   }
  else return false
}
```



- Pros/Cons
  - no memory wasted
  - can be slow if node is re-built over and over again

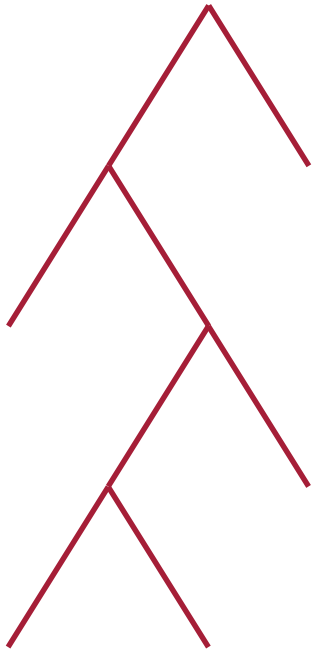
Lazy Building of Trees

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Build a sub-tree only when some ray may intersect it

- Naïve Algorithm:

```
bool intersect( ray, node ){
  if (hit( ray, node )) {
   split( node )
   return ( intersect( ray, child.left) ||
            intersect( ray, child.right) )
   }
  else return false
}
```

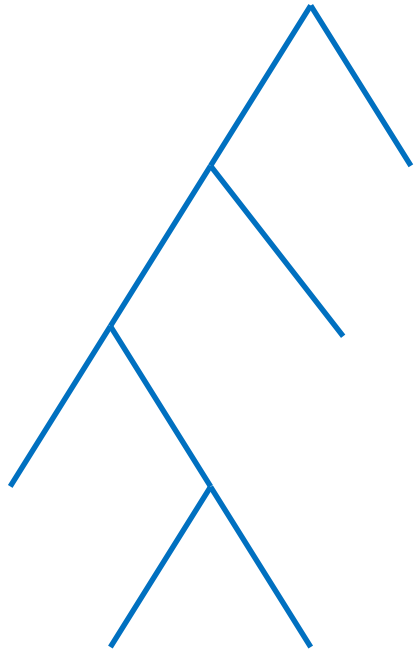


- Pros/Cons
  - no memory wasted
  - can be slow if node is re-built over and over again

Lazy Building and Caching

- Cached Algorithm:

```
nodesCache.add( root );
bool intersect( ray, node ) {
 if (hit( ray, node ))  {
   if (!in(nodesCache(node)) {
    split( node )
     nodesCache.add( children )
   }
   return ( intersect( ray, child.left) ||
            intersect( ray, child.right) )
 }
 else return false
}
```

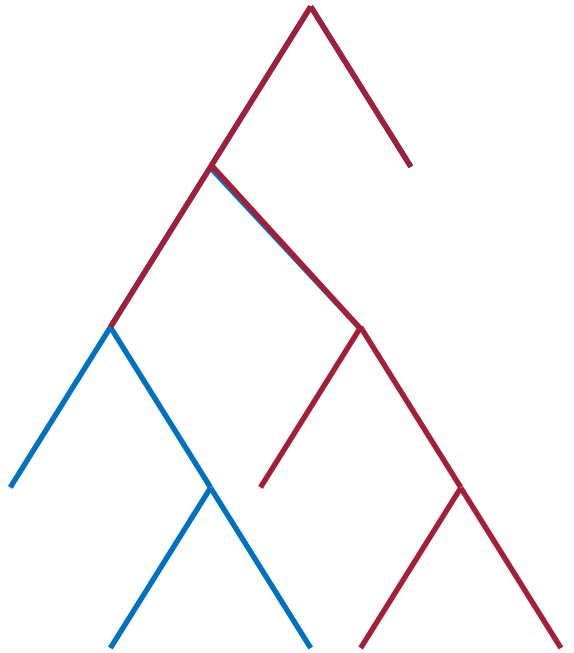


- Problems:
  - Lots of tests, especially in the top tree
  - Cache might not be large enough

Lazy Building and Caching

- Cached Algorithm:

```
nodesCache.add( root );
bool intersect( ray, node ) {
 if (hit( ray, node ))  {
   if (!in(nodesCache(node)) {
    split( node )
     nodesCache.add( children )
   }
   return ( intersect( ray, child.left) ||
            intersect( ray, child.right) )
 }
 else return false
}
```



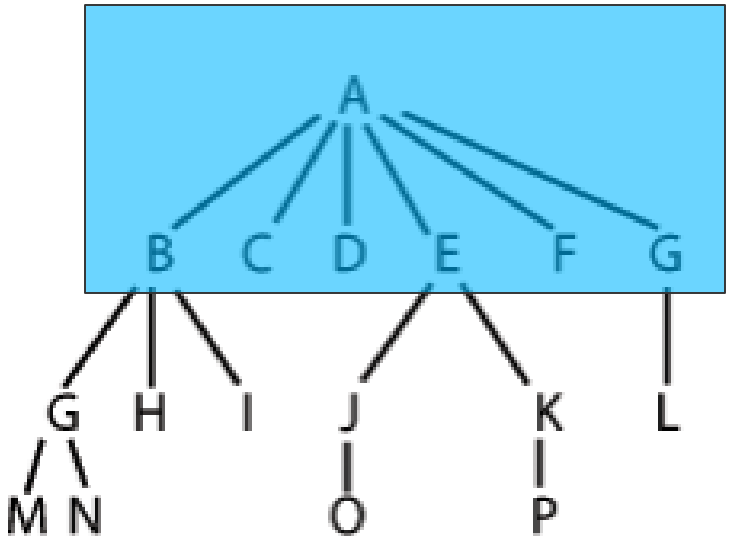
- Problems:
  - Lots of tests, especially in the top tree
  - Cache might not be large enough

Multi-Level Hierarchy

- E.g. coarse shape plus procedural geometry detail

- Fixed spatial data structure for coarse shape
  - small enough to fit in memory
  - will be visited over and over again
  - fast access

- Lazy build for fine scale geometry
  - cached or un-cached



Ray Re-ordering

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Avoid re-building detailed children over and over
  - rays might be incoherent
  - cache might be too small
  - don‘t waist memory

- Compute intersection off all rays with the top part of the hierarchy
  - first, second and …. n-th intersection
- Sort rays to children
- Process each requested child once:
  - split
  - intersect child with all potentially intersecting rays
  - remove child


paper_1234
Two-Level Ray Tracing with Reordering
for Highly Complex Scenes

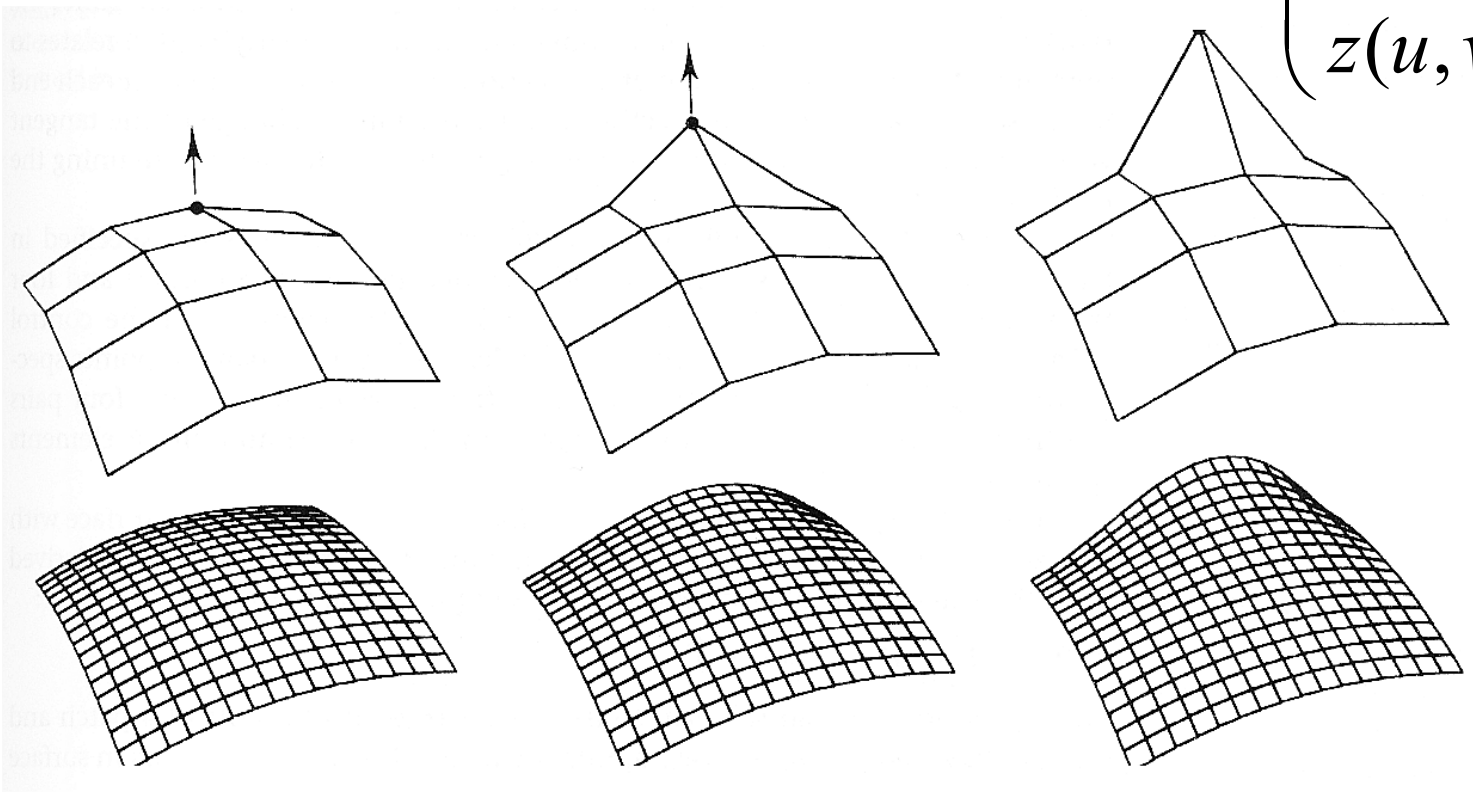
Rendering Free-Form Surfaces

How to ray trace smooth surfaces, which can be tessellated into triangles?

Free-Form Surfaces

- Parametric surfaces could be tesselated into lots and lots of triangles
  - E.g. tensor product Beziér patches

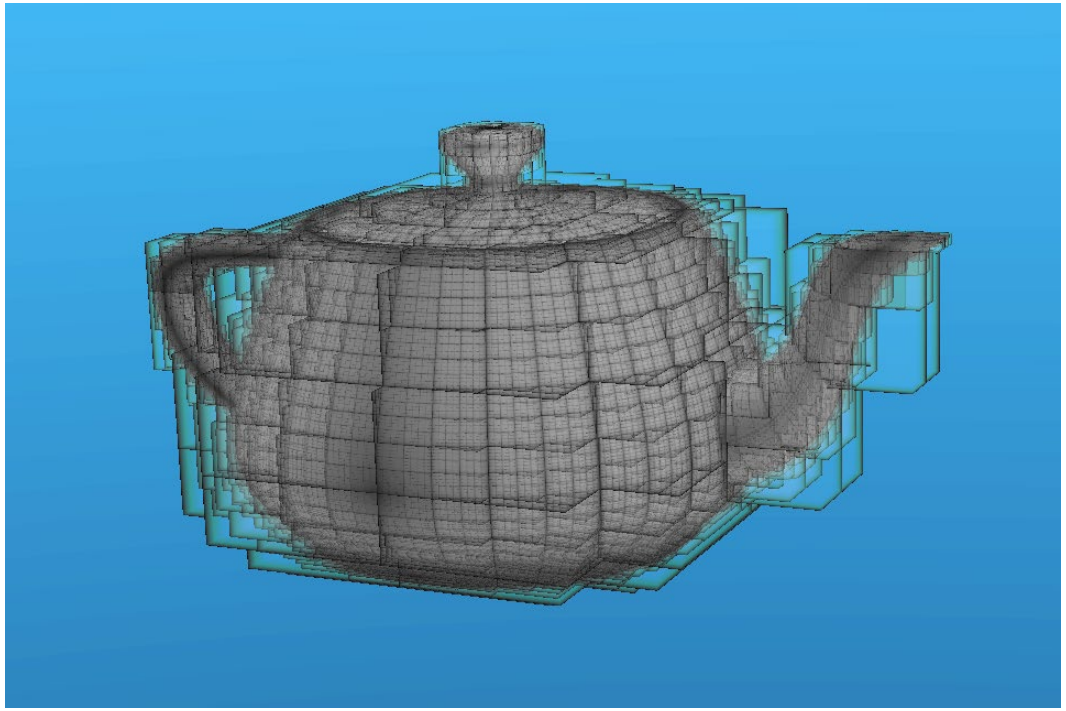
$$\vec{s}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$



Free-Form Surfaces

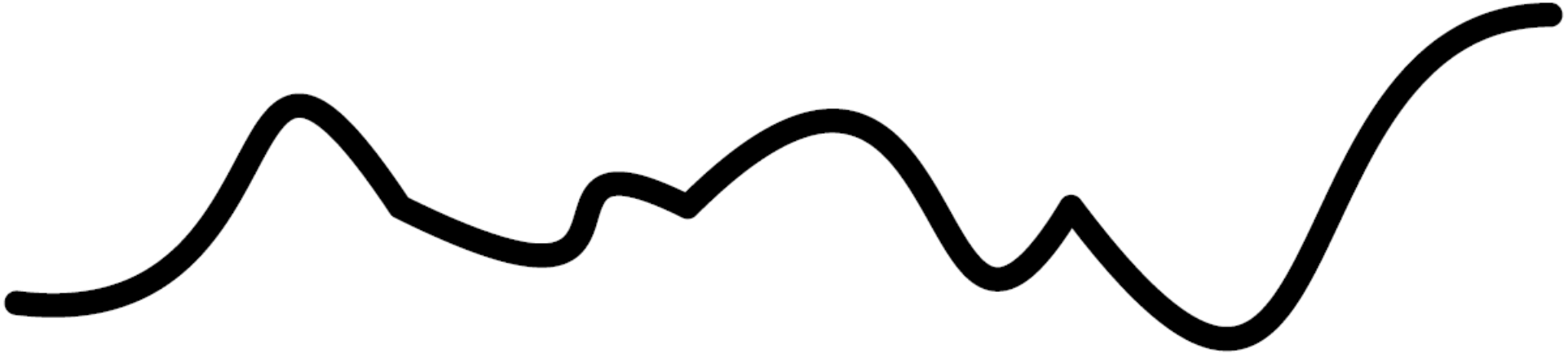
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Preparing the surface (given as function over the surface parameters) for rendering
  - Explicit: transform into set of triangles -> tessellation
  - Implicit: by hierarchy of bounding volumes

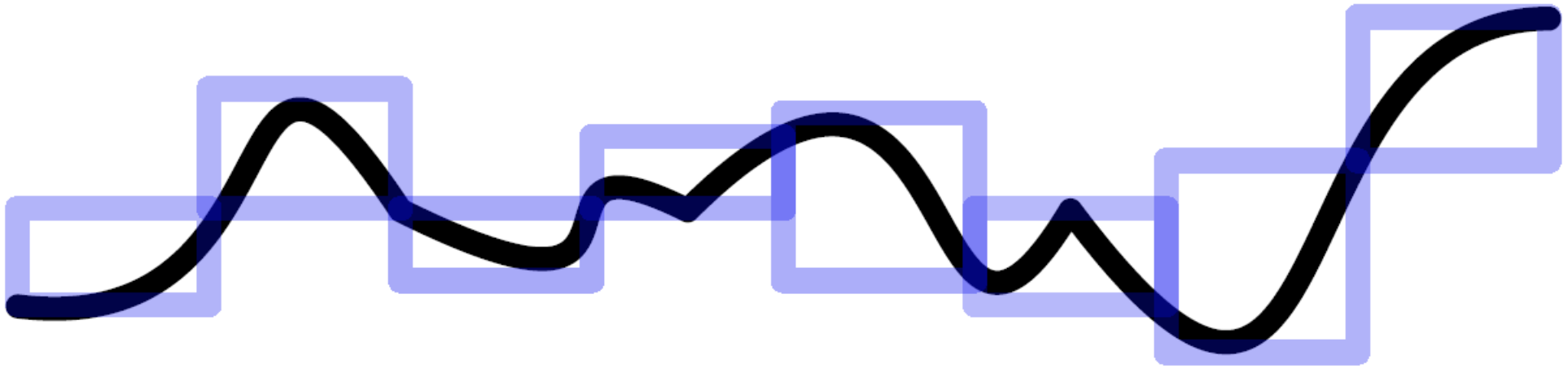


Free-Form Surface

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

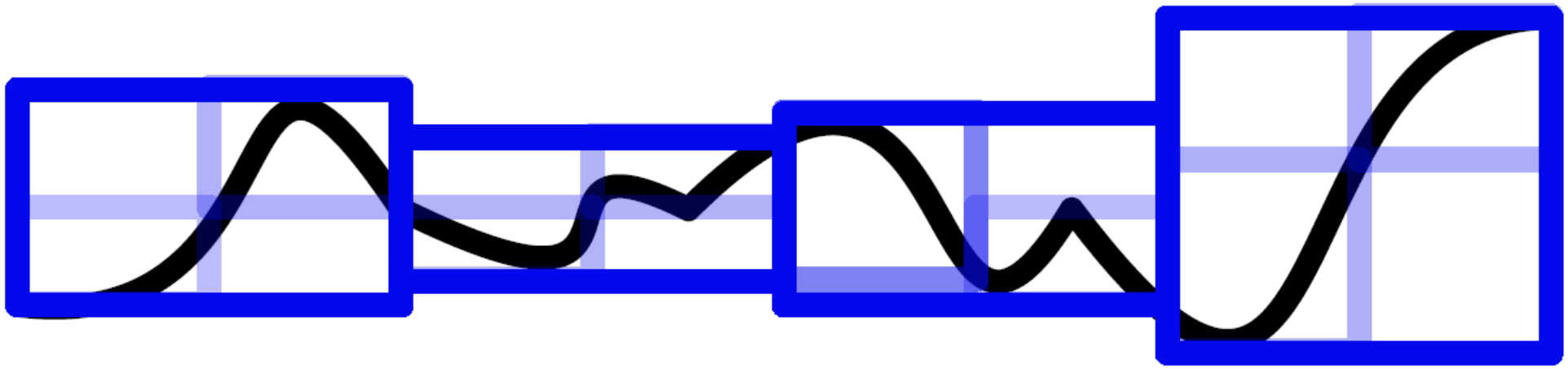


Approximation by Bounding Boxes

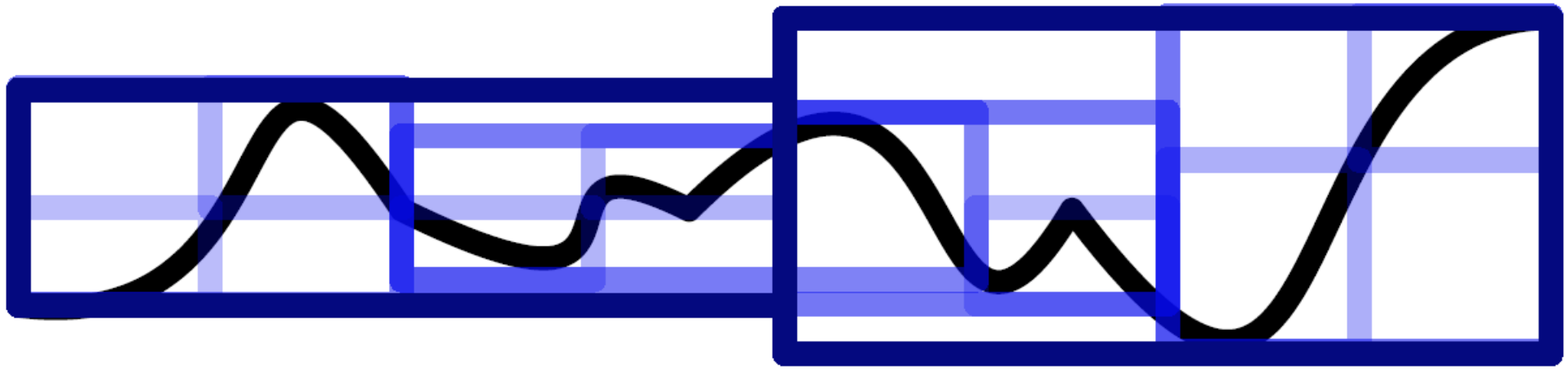


Bounding Volume Hierarchy

- Hierarchy of Bounding Boxes

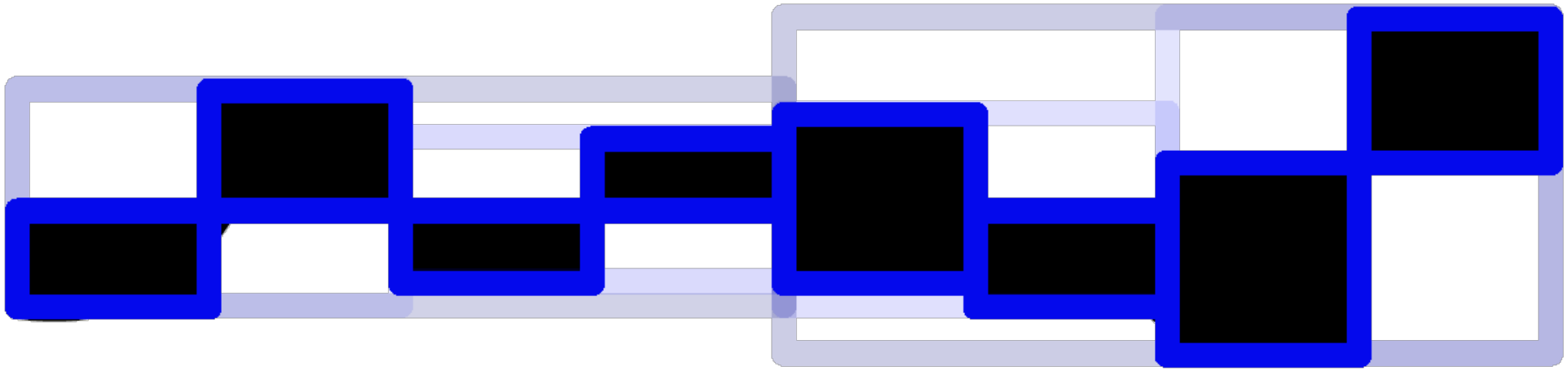


Bounding Volume Hierarchy



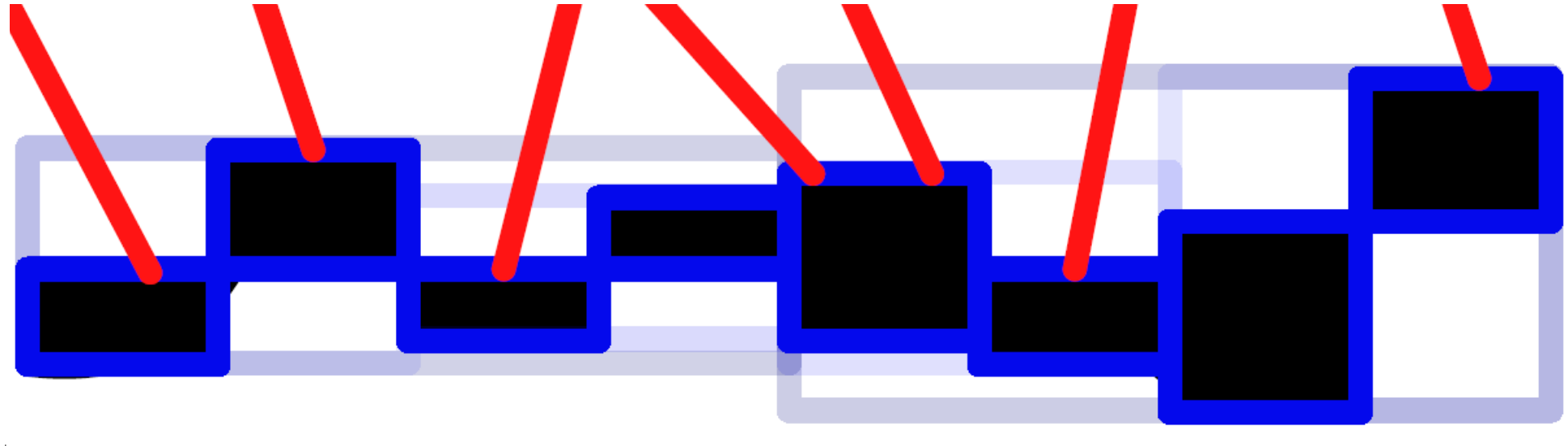
Bounding Volume Hierarchy

- On the finest level it can be shown that there will never be any holes.



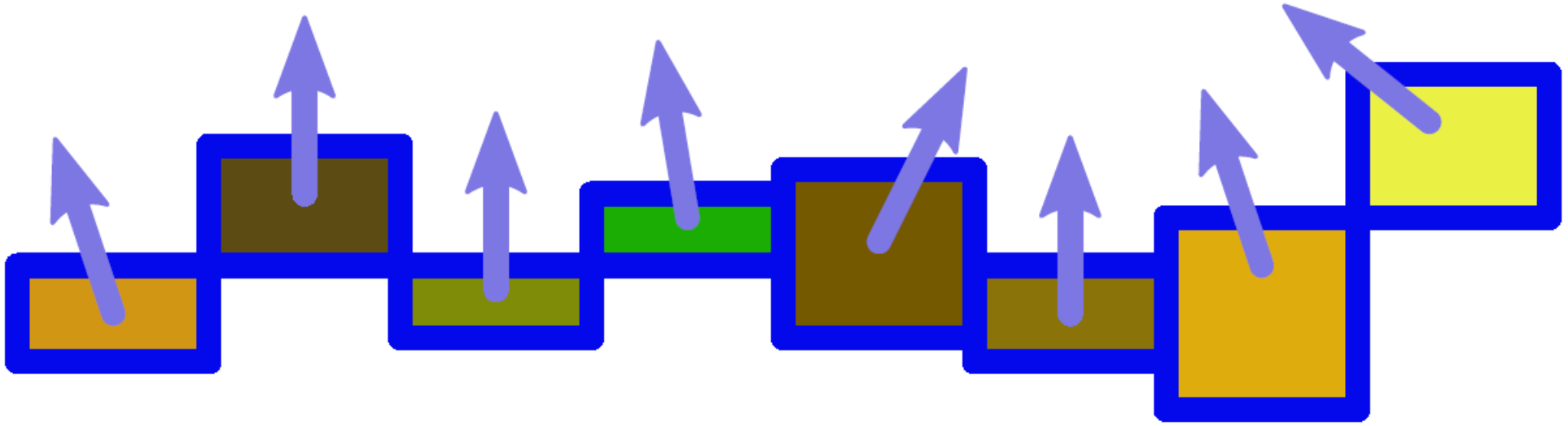
Implicit Representation

- Finest level reached when box smaller than a pixel.



Norman and Color

- To provide a reasonable (smooth) surface the normal and the color will be interpolated for each cell.



Ray Tracing Dynamic Scenes

Ray Tracing Dynamic Scenes

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- 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
  - Interactive: Motion defined by user
- 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

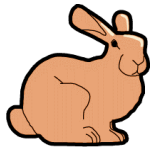
Partitioning: Divide & Conquer

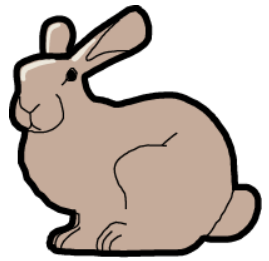
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

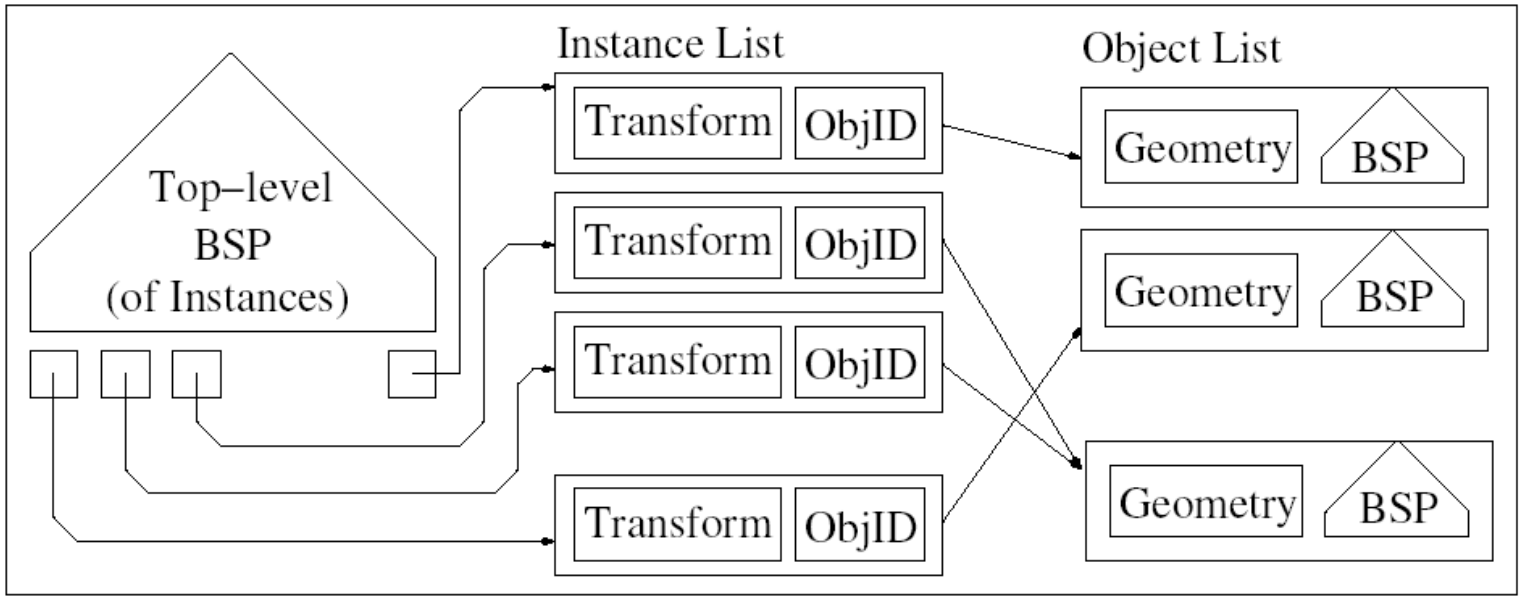
- Observation
  - Very often a simple approach is sufficient
  - Building hierarchical index structures requires O(n logn)
    - Divide and conquer reduces complexity
- Categorize primitives into independent groups/objects
  - Static parts of a scene (often large parts of a scene)
  - Structured motion (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: k times O(log n)

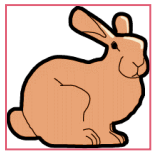


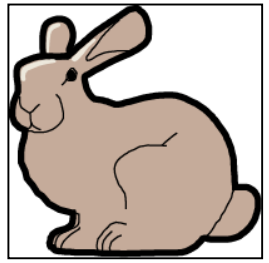


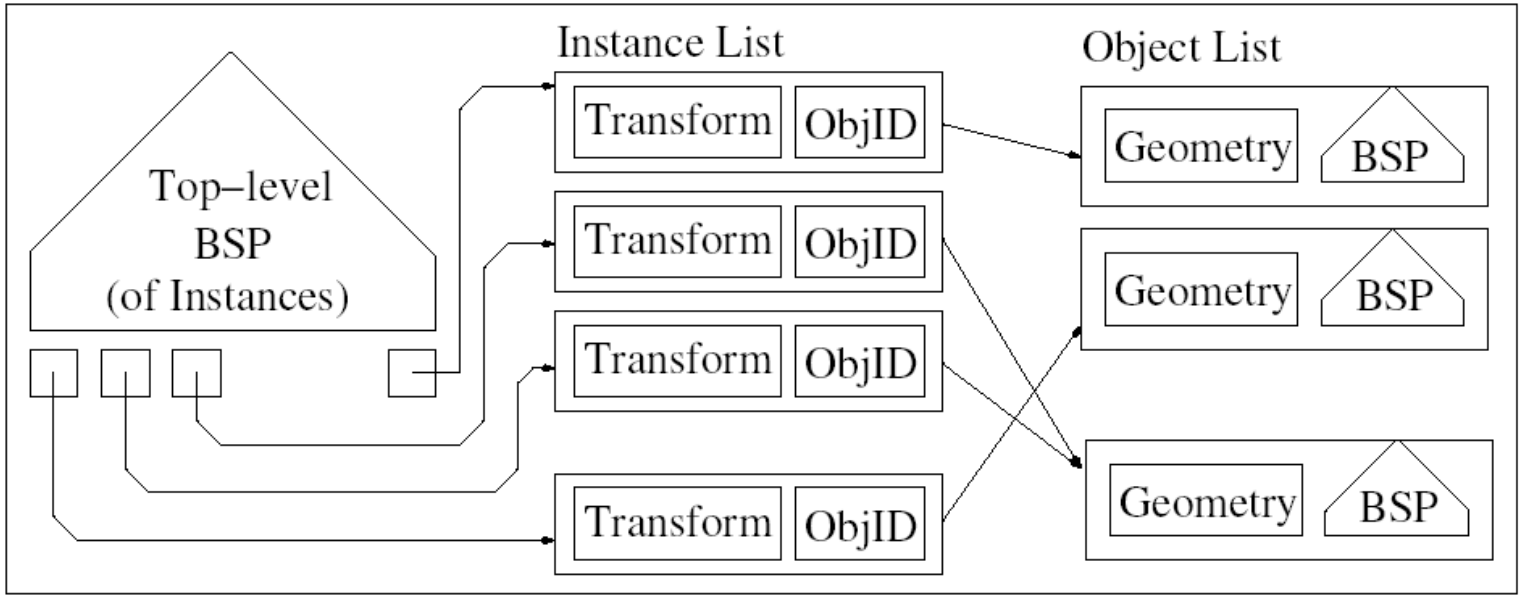


Divide & Conquer Approach

- 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: k times O(log n)



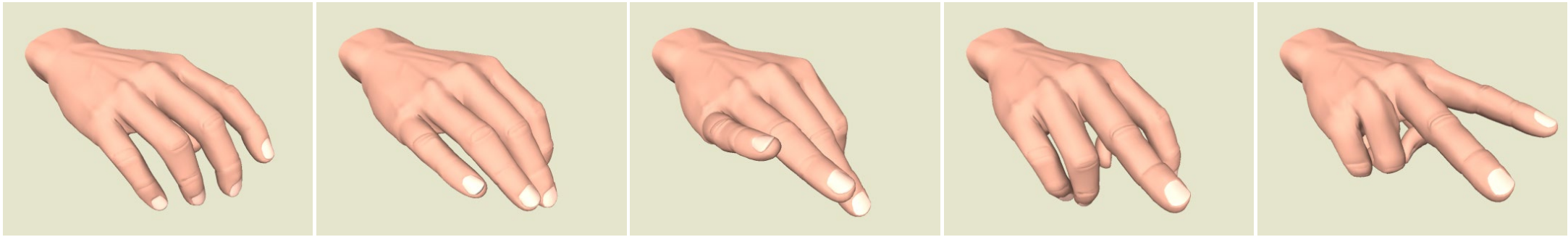




Updating the Index Structure

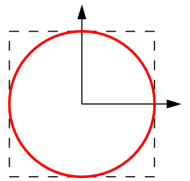
- Idea
  - „Make dynamic scenes static”

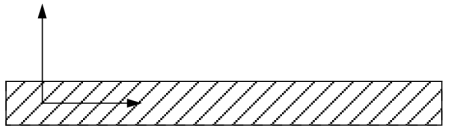
- Assumptions:
  - Deformation of a base mesh (constant connectivity)
  - All frames of animation known in advance
  - Continuous motion



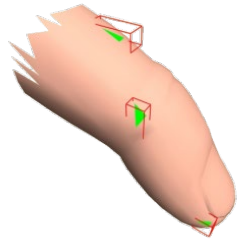
Method Overview

- Motion decomposition
  - Affine transformations
  - + residual motion

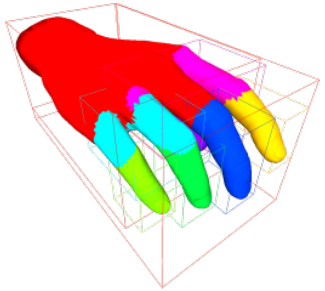




- Fuzzy kd-tree
  - Handles residual motion

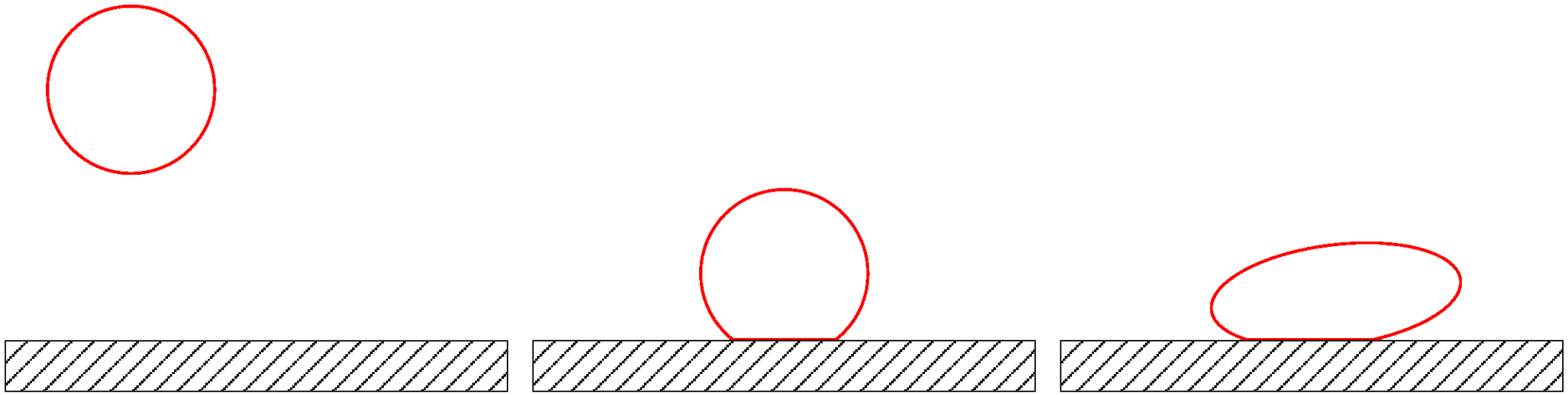


- Clustering
  - Exploit local coherent motion



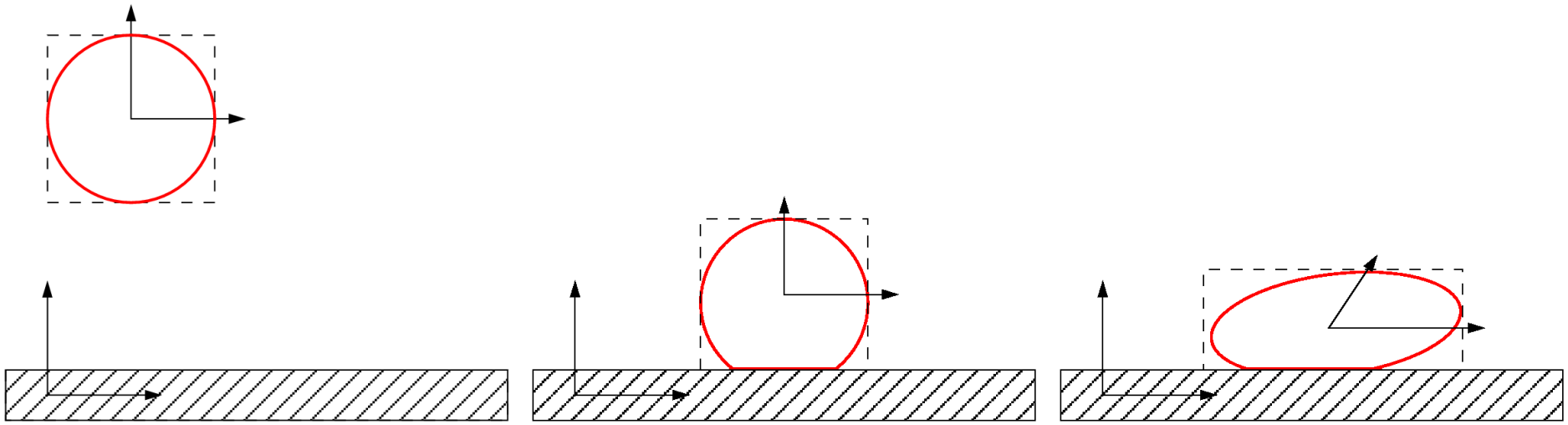
Motion Decomposition

- Dynamic scene: ball thrown onto floor



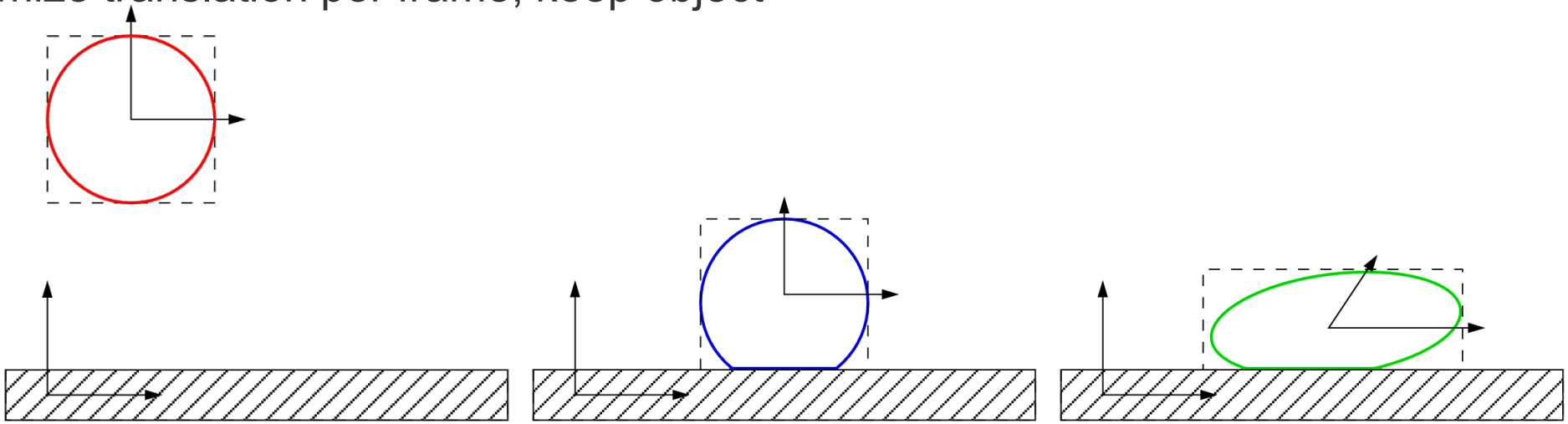
Motion Decomposition

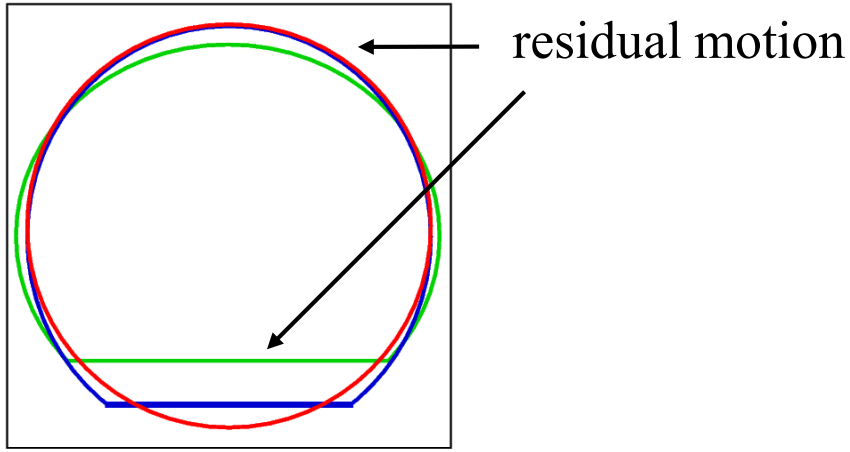
- Affine transformations
  - Approximate deformations
  - Include shearing (3rd frame)



Motion Decomposition

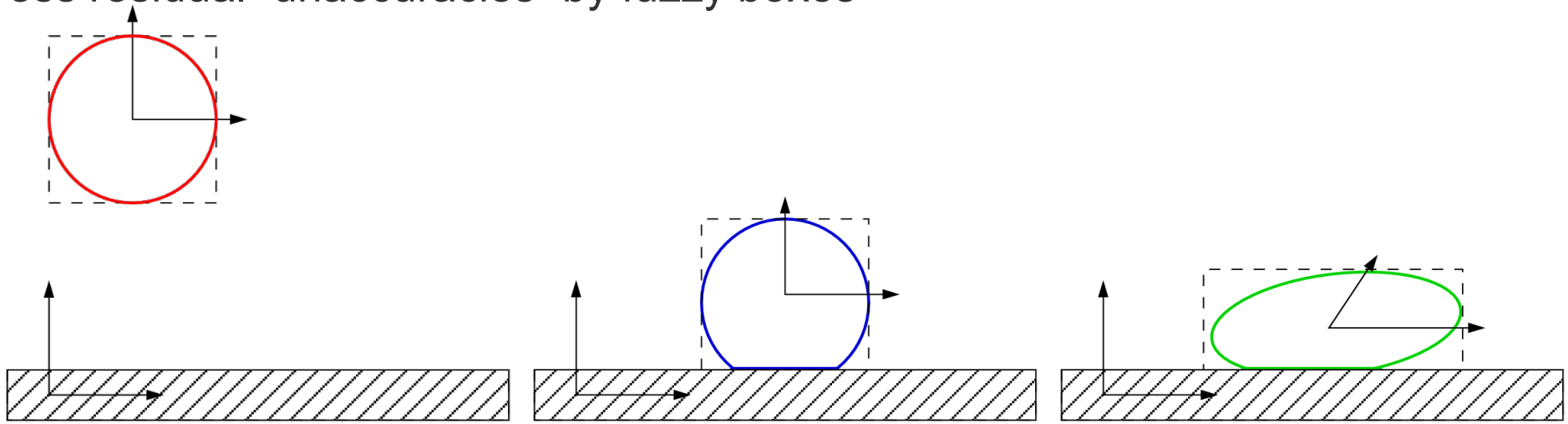
- Optimize translation per frame, keep object

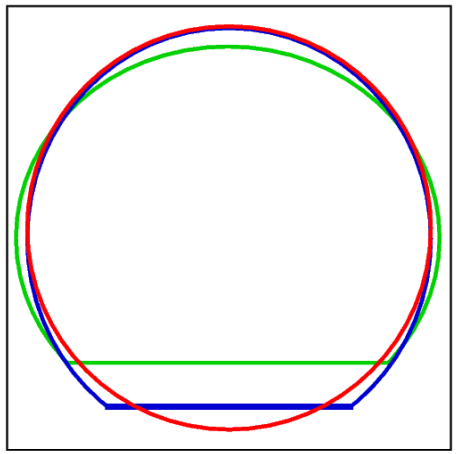


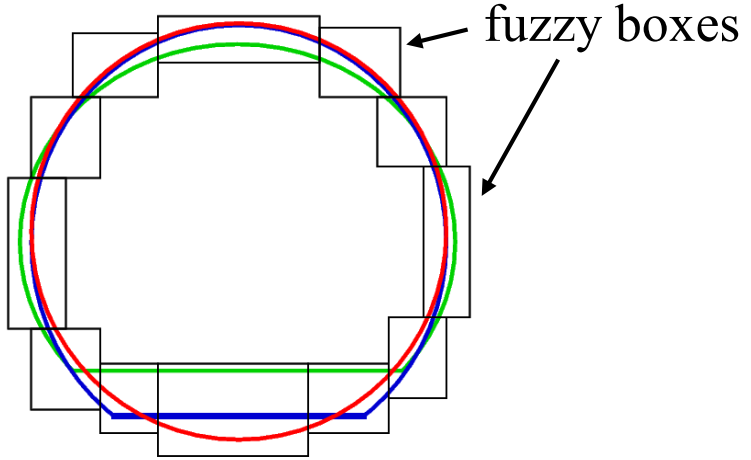


Motion Decomposition

- Express residual “unaccuracies” by fuzzy boxes



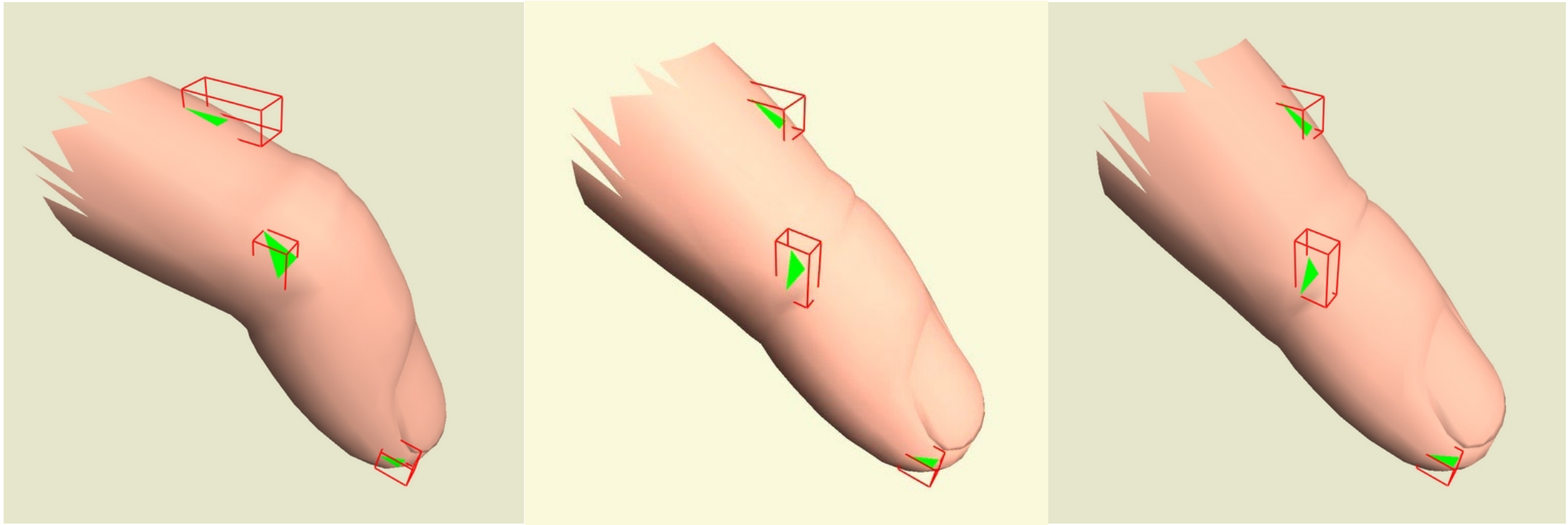




Fuzzy KD-Tree

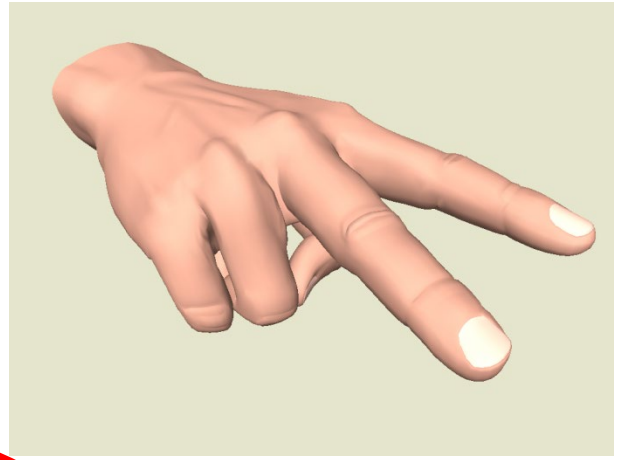
- Handles residual motion

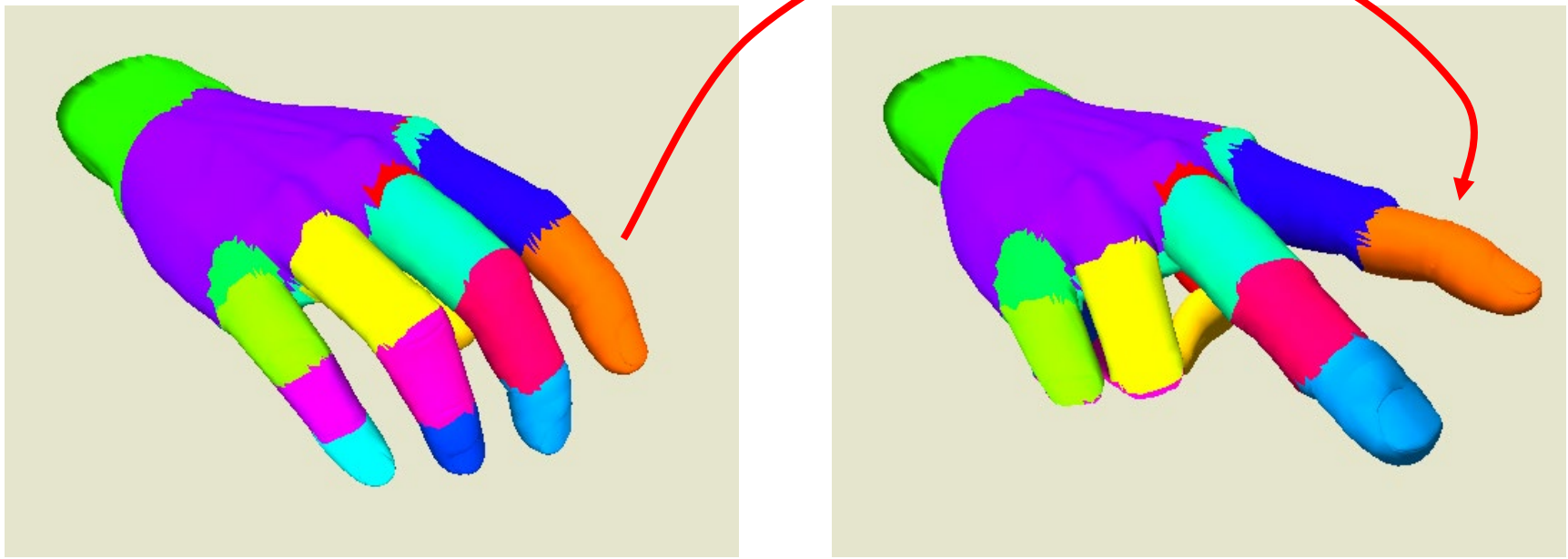
- KD-Tree over the fuzzy boxes of triangles
  - Valid over complete animation



Clustering

- Determine groups of primitives with similar motion



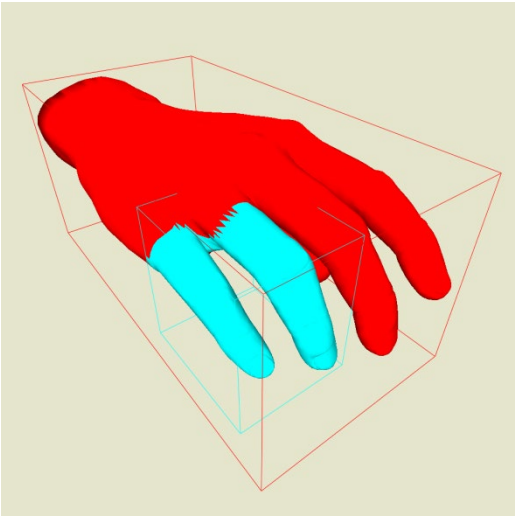


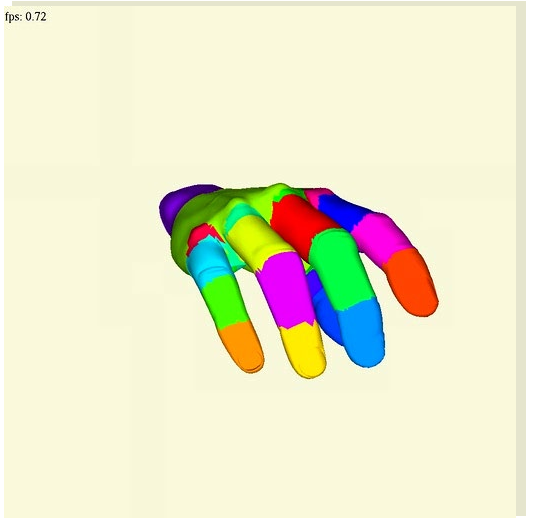
Details: Clustering

- Efficient ray tracing:
  - Requires small fuzzy boxes
  - Must minimize residual motion
  - Should cluster coherently moving triangles
    - Results in fewer objects
- 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

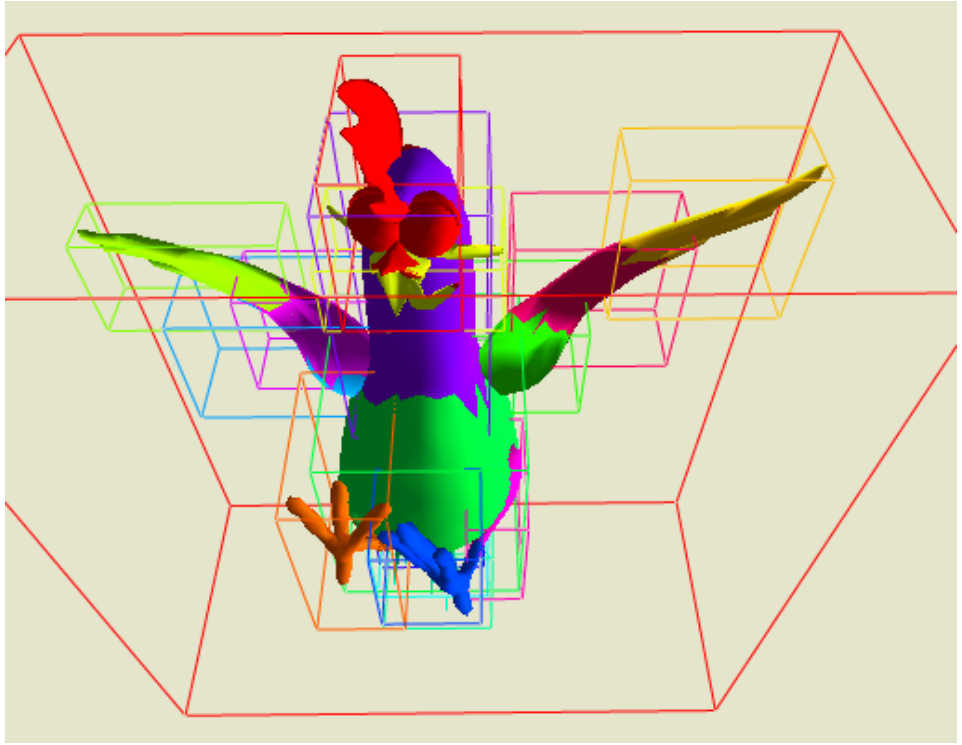


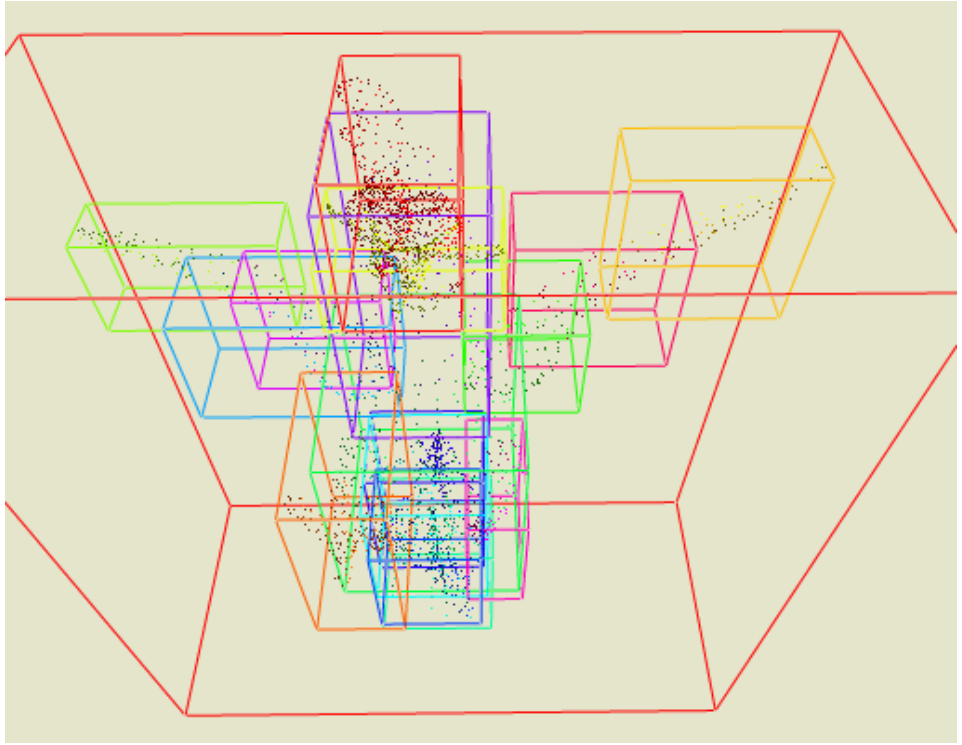


Ray Tracing: Two-level Approach

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- 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

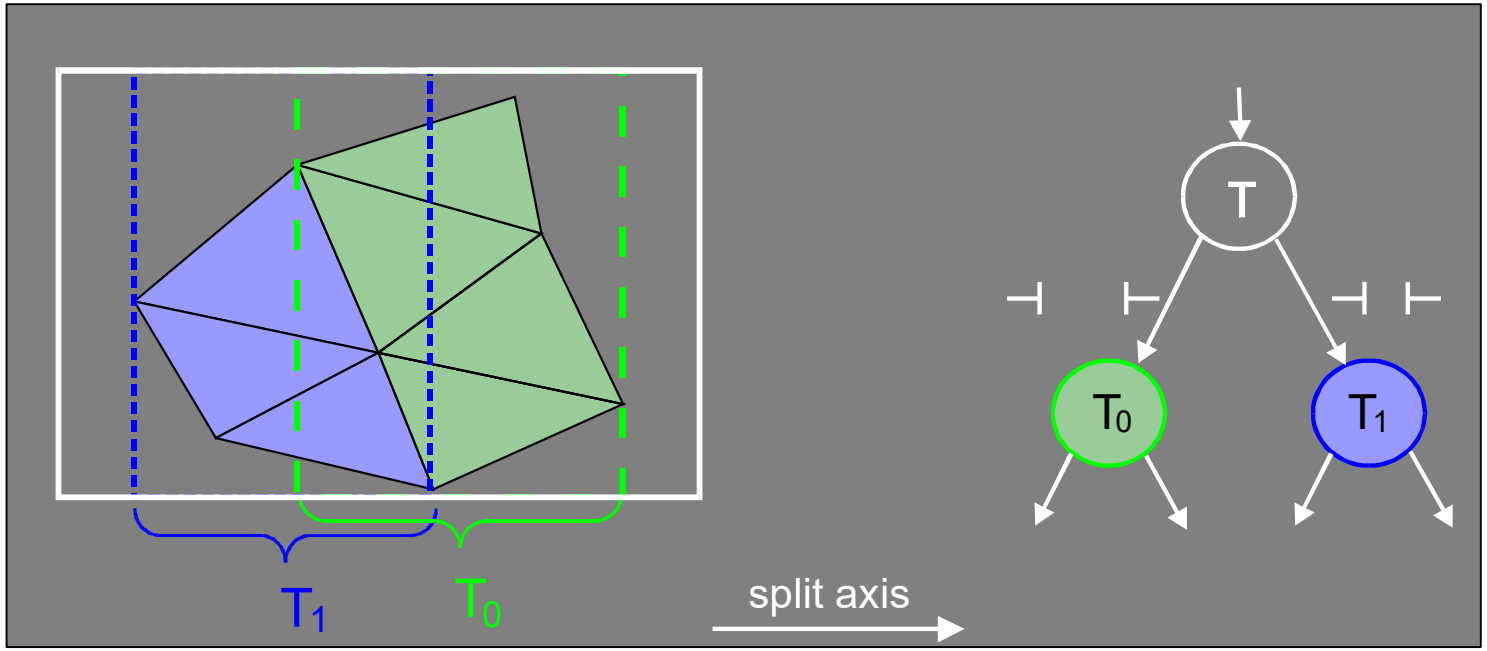




Bounding KD-Trees: Mixing Bounding Volumes and KD-trees

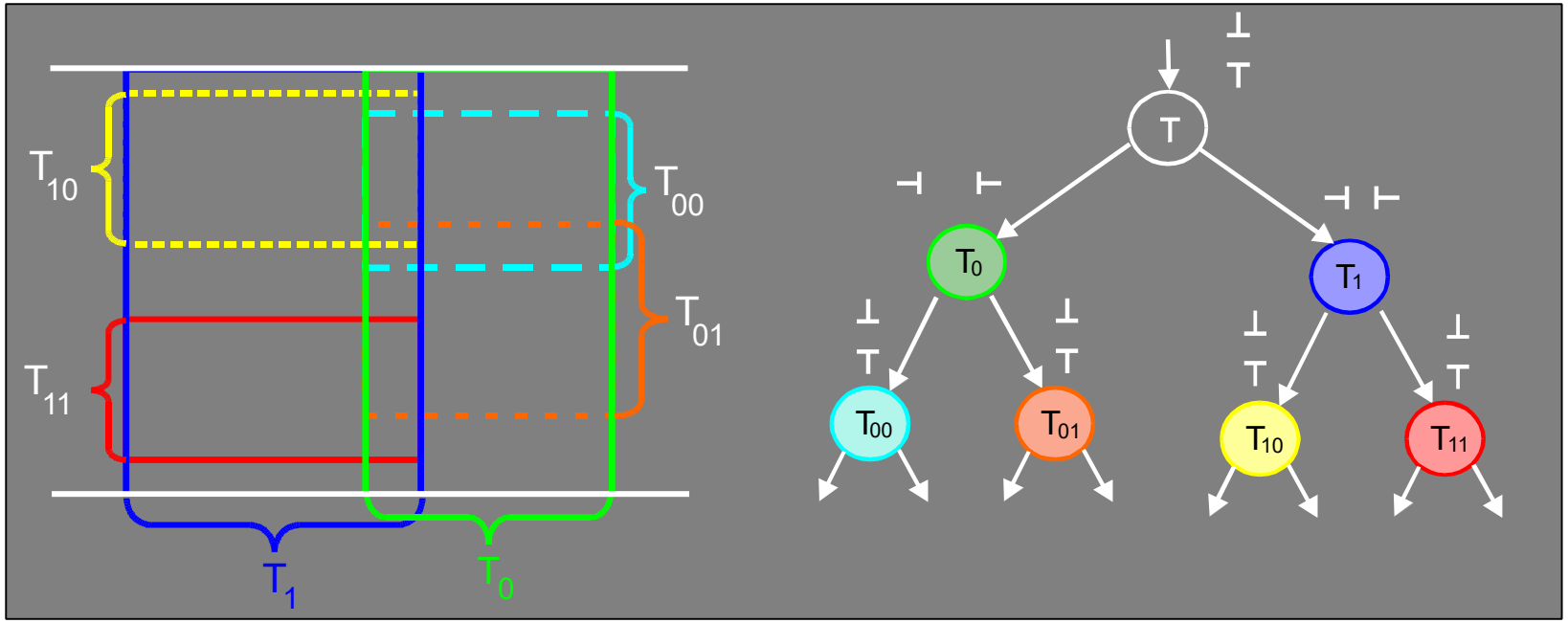
Definition of B-KD Trees

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



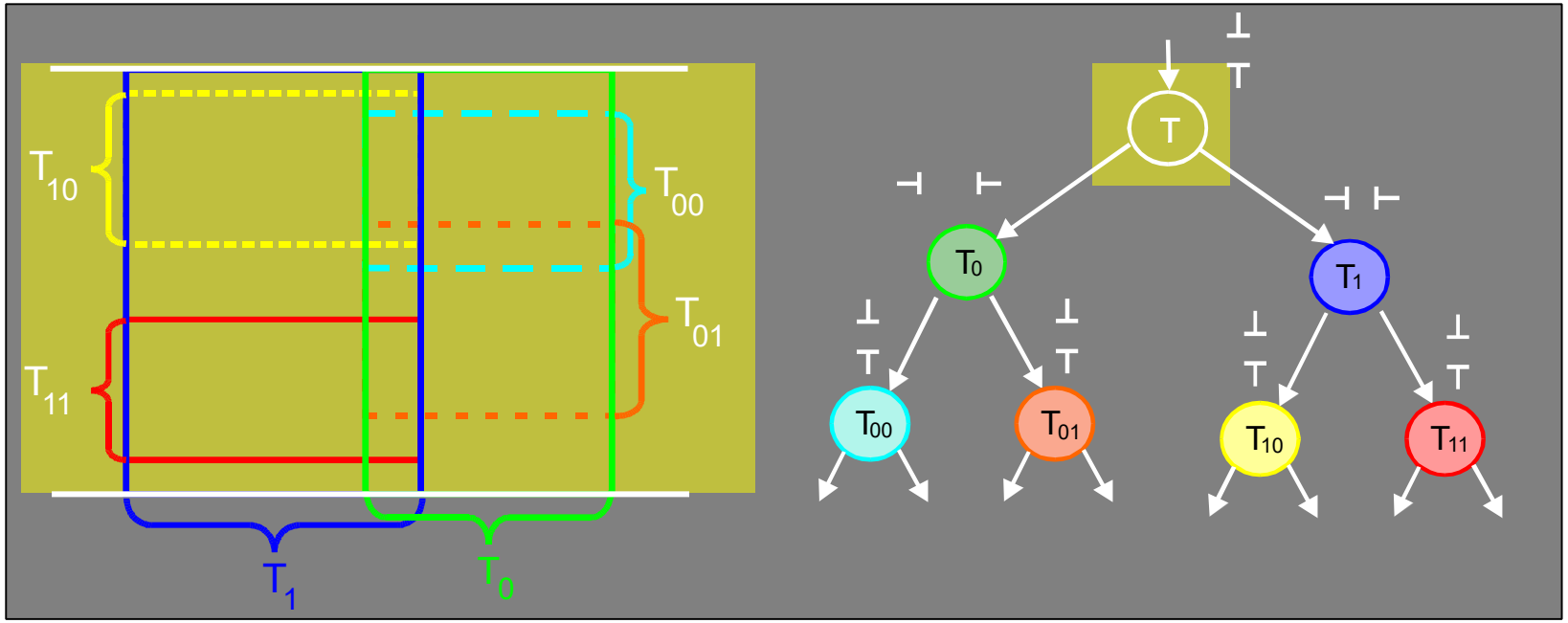
B-KD Tree Subdivision

- 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



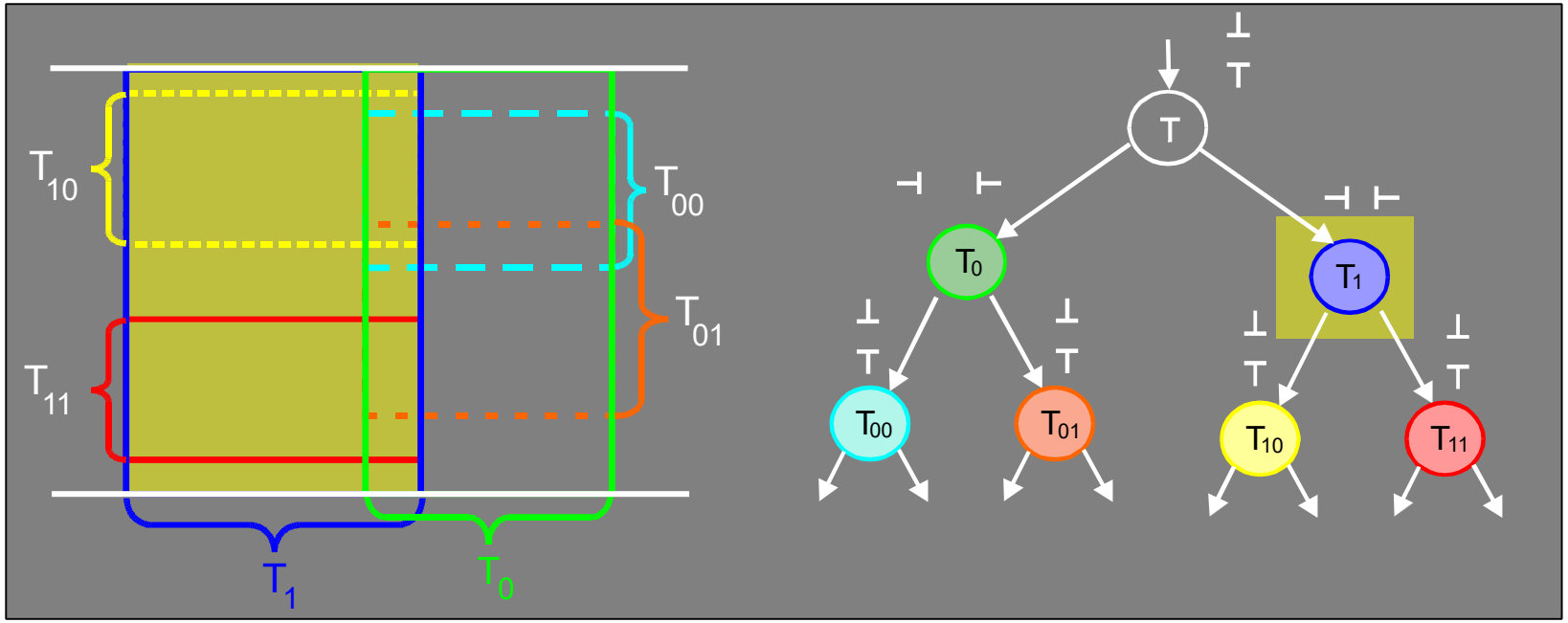
B-KD Tree Subdivision

- 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



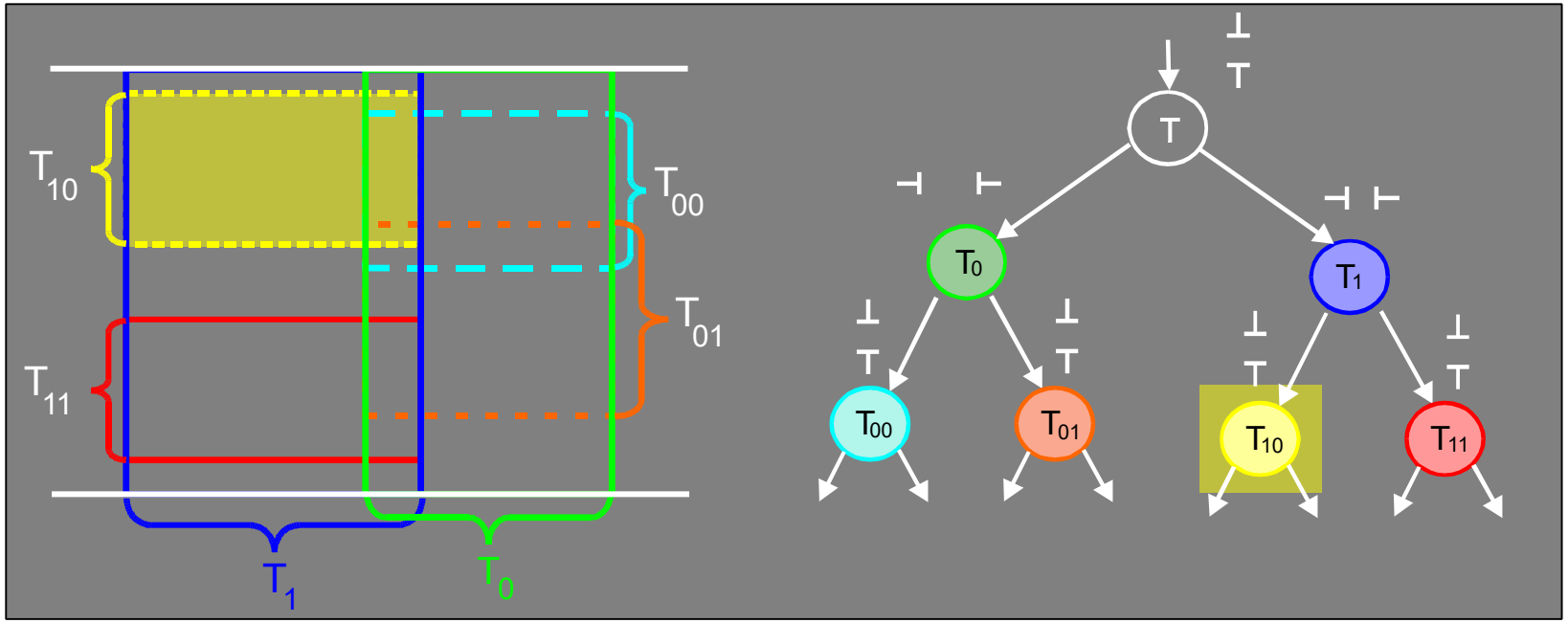
B-KD Tree Subdivision

- 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



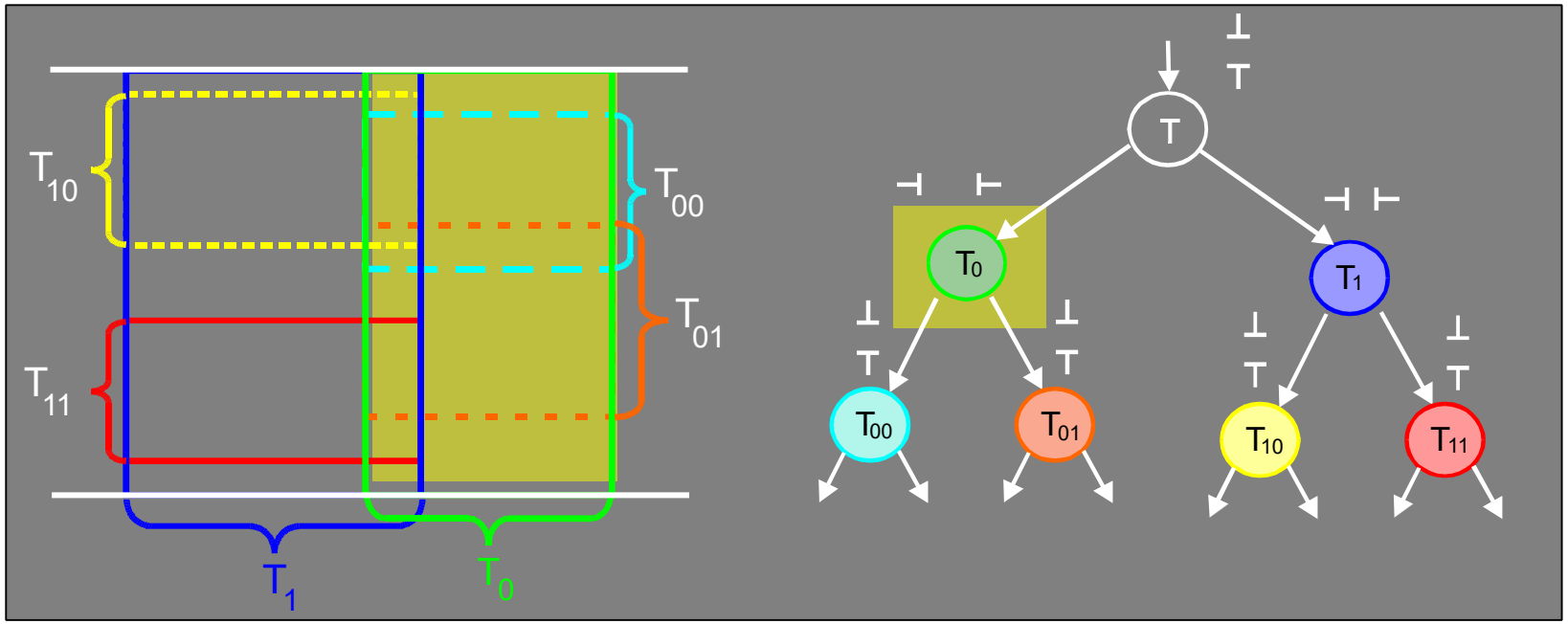
B-KD Tree Subdivision

- 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



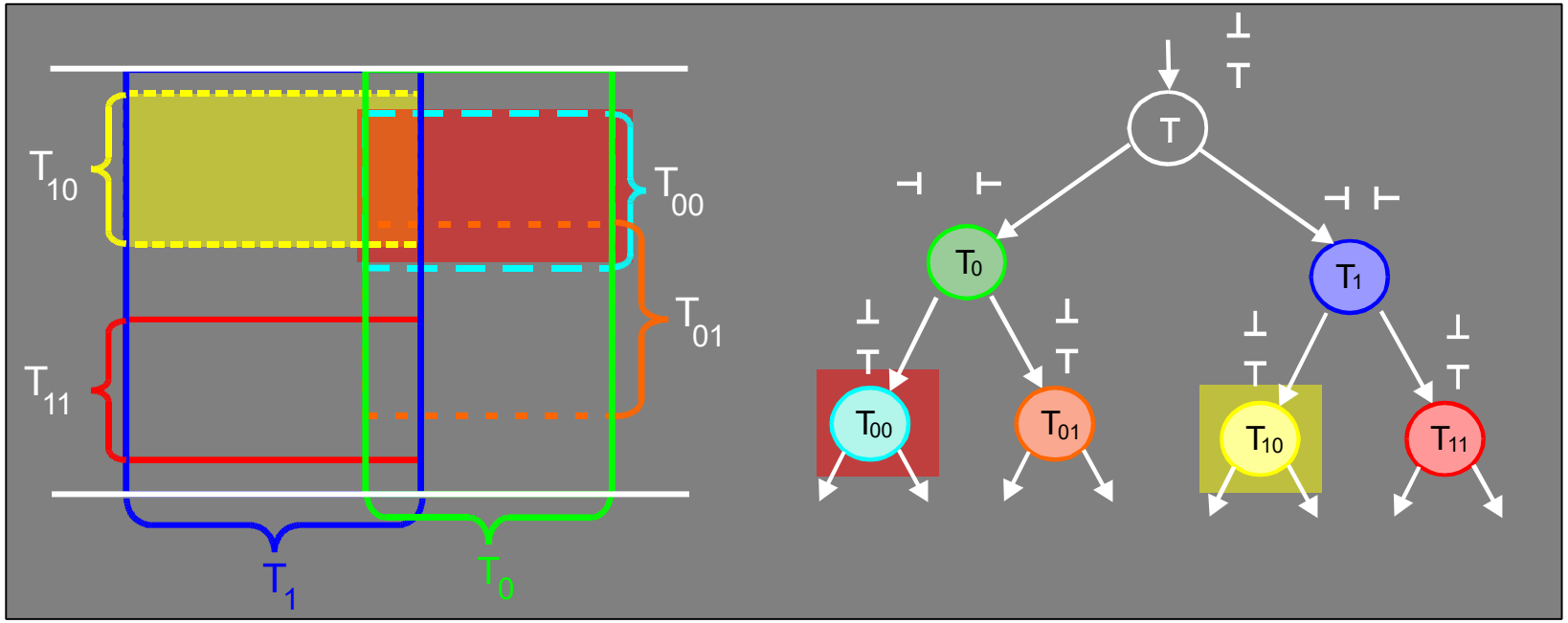
B-KD Tree Subdivision

- 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



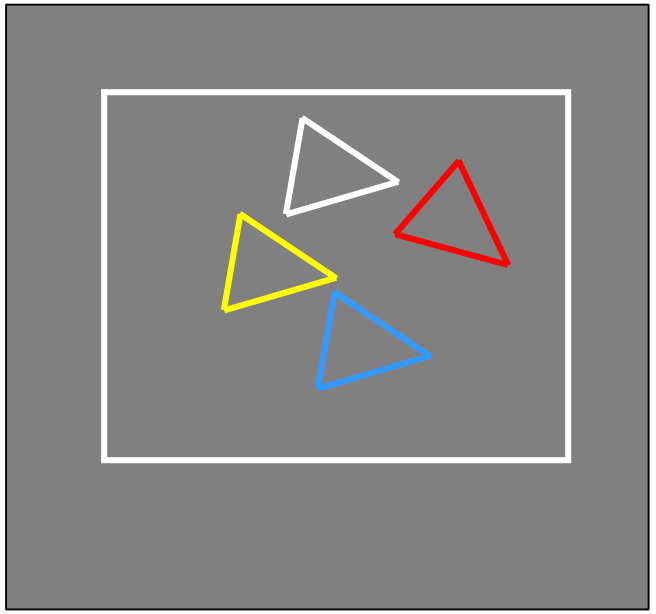
B-KD Tree Subdivision

- 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



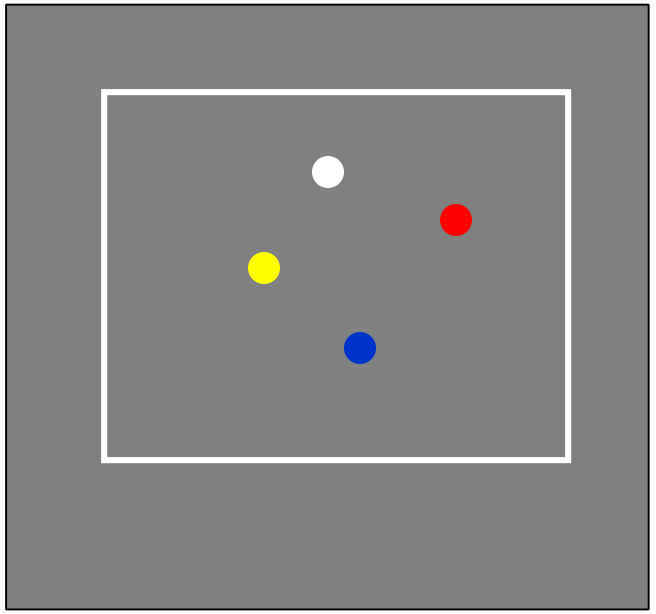
B-KD Tree Construction

- If #primitives > 1 then



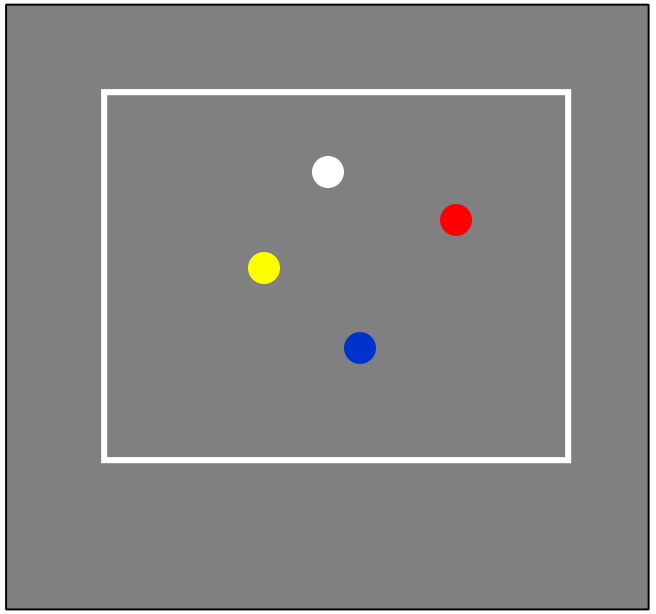
B-KD Tree Construction

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



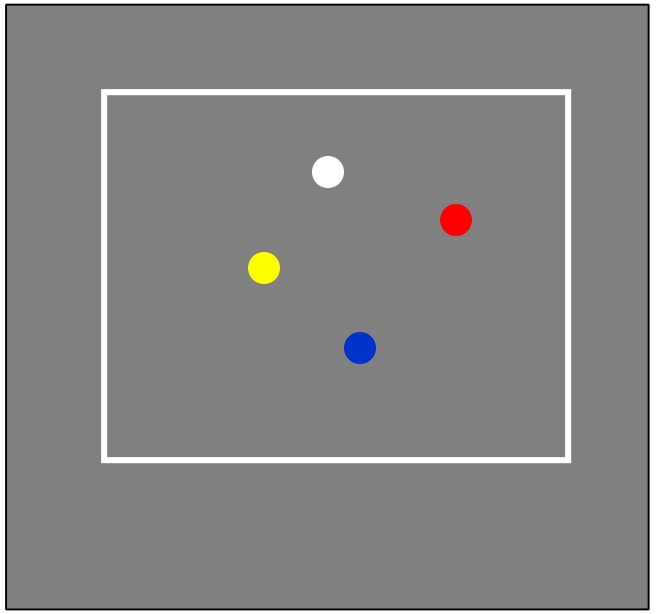
B-KD Tree Construction

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



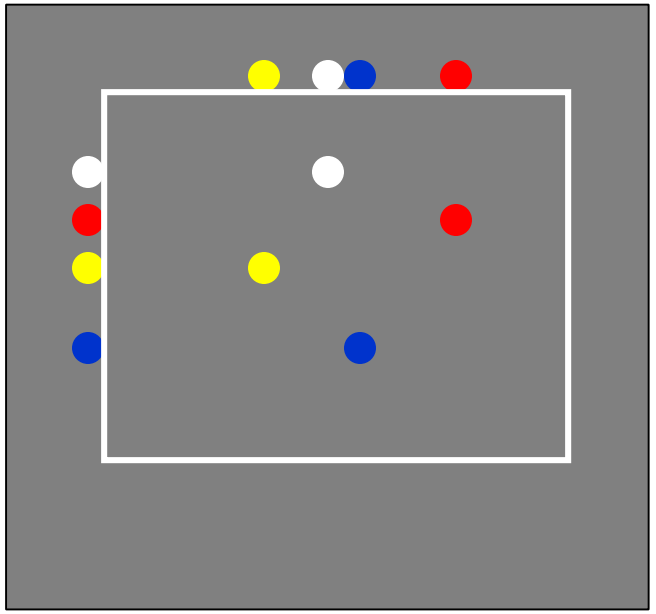
B-KD Tree Construction

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



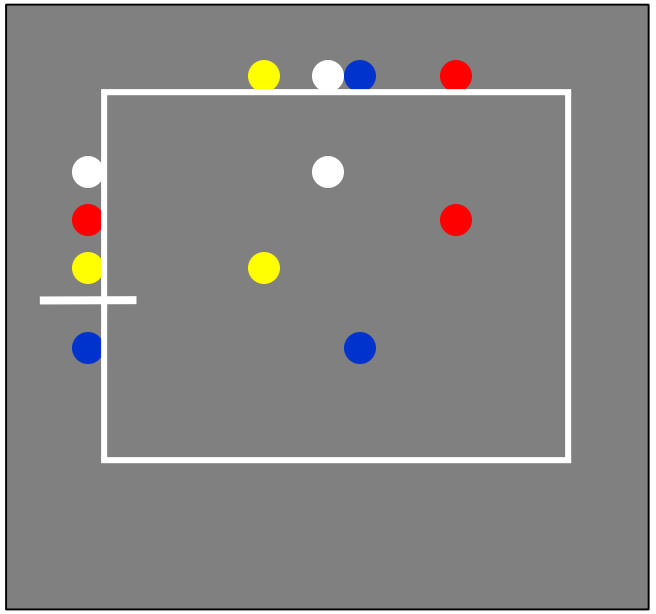
B-KD Tree Construction

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



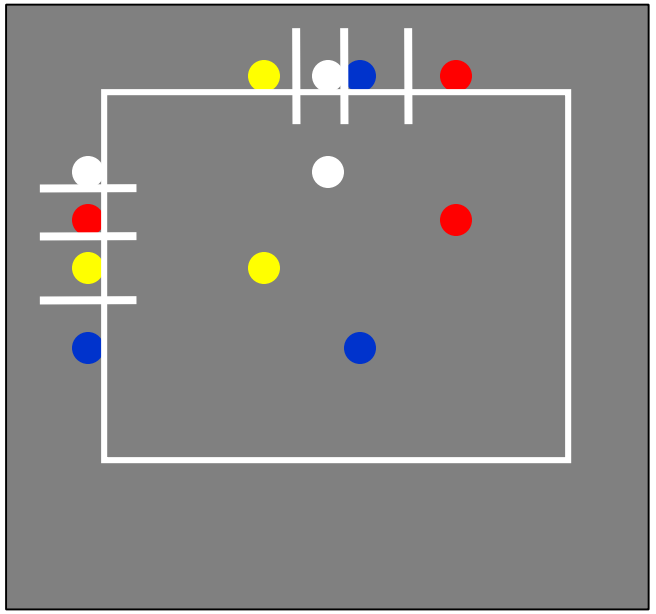
B-KD Tree Construction

- 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



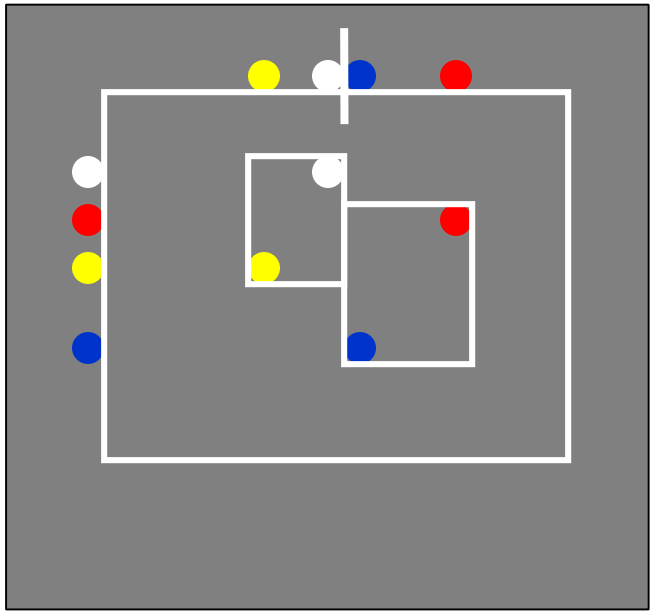
B-KD Tree Construction

- 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



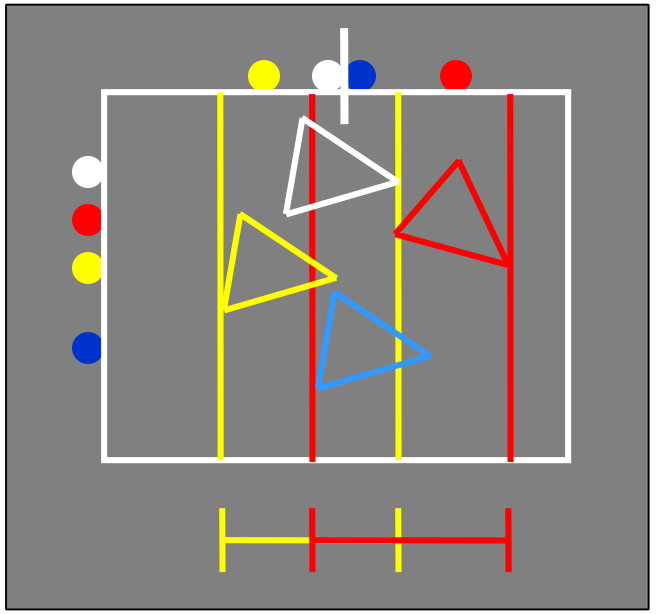
B-KD Tree Construction

- 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

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- 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

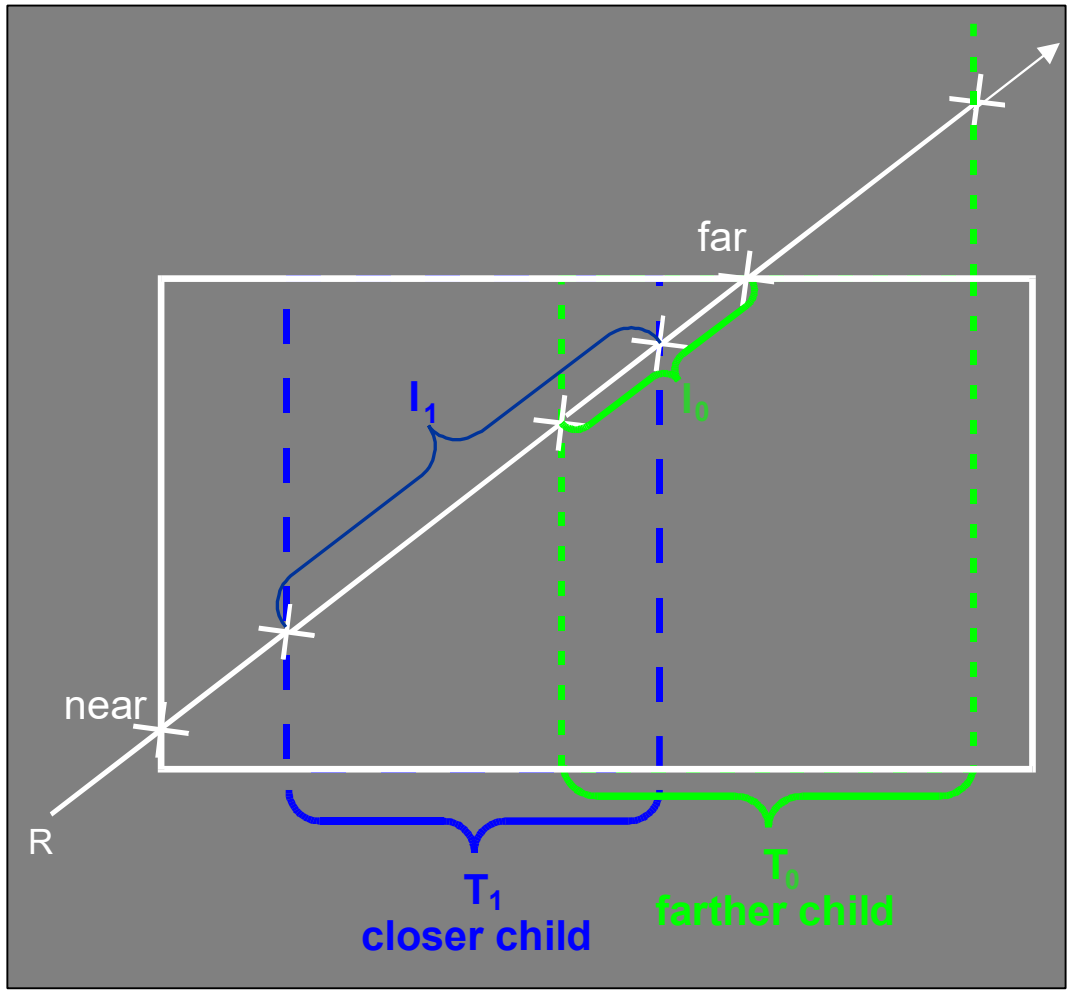
B-KD Tree Construction

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- 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

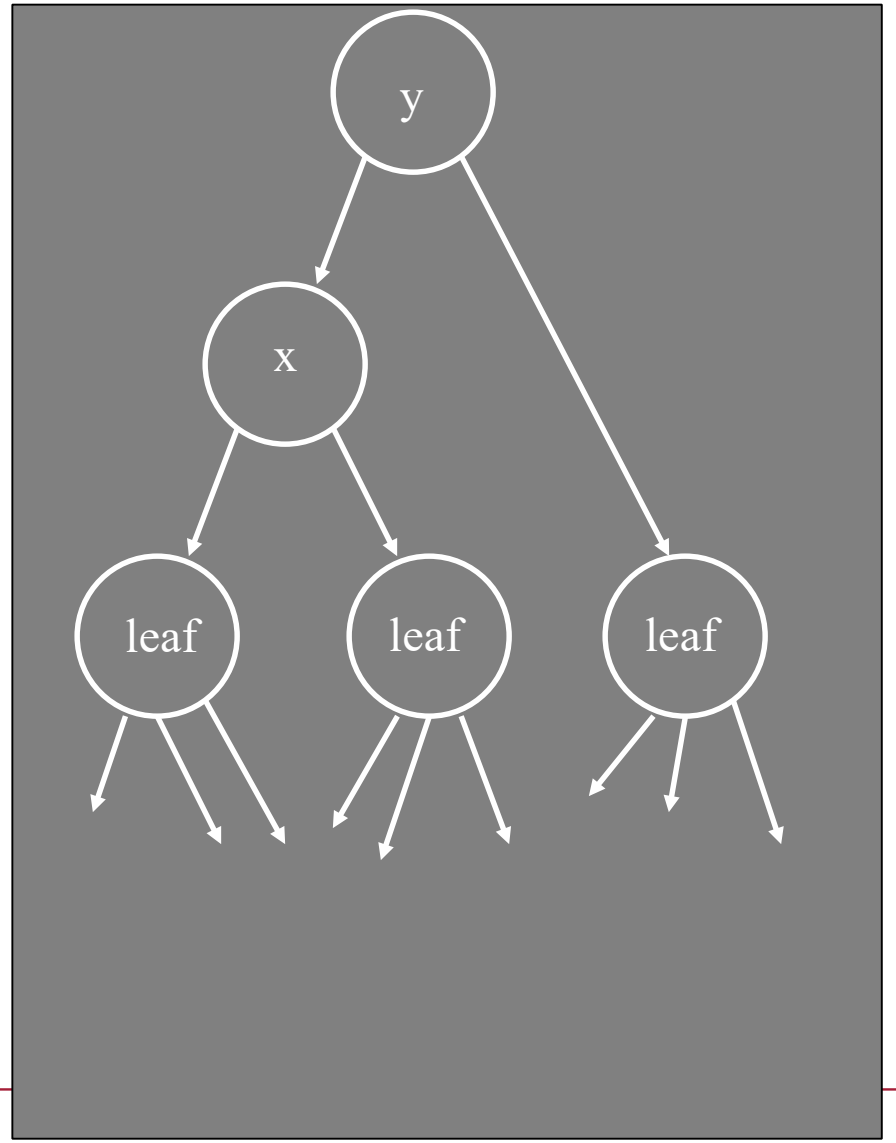
Traversal of B-KD Trees

- Traversal of B-KD Trees
- Early ray termination
- Clipping of near/far interval against both bounding intervals
- 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



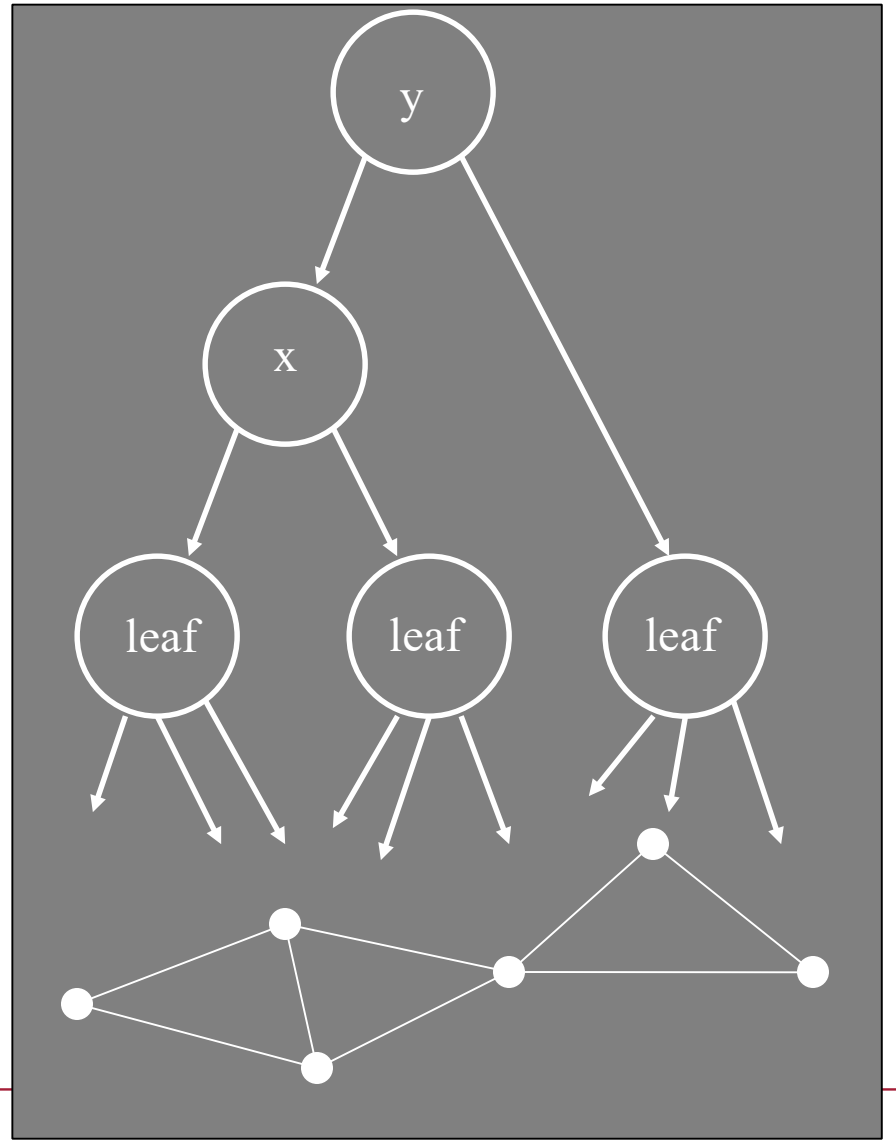
Update of B-KD Trees

- Leaf Node



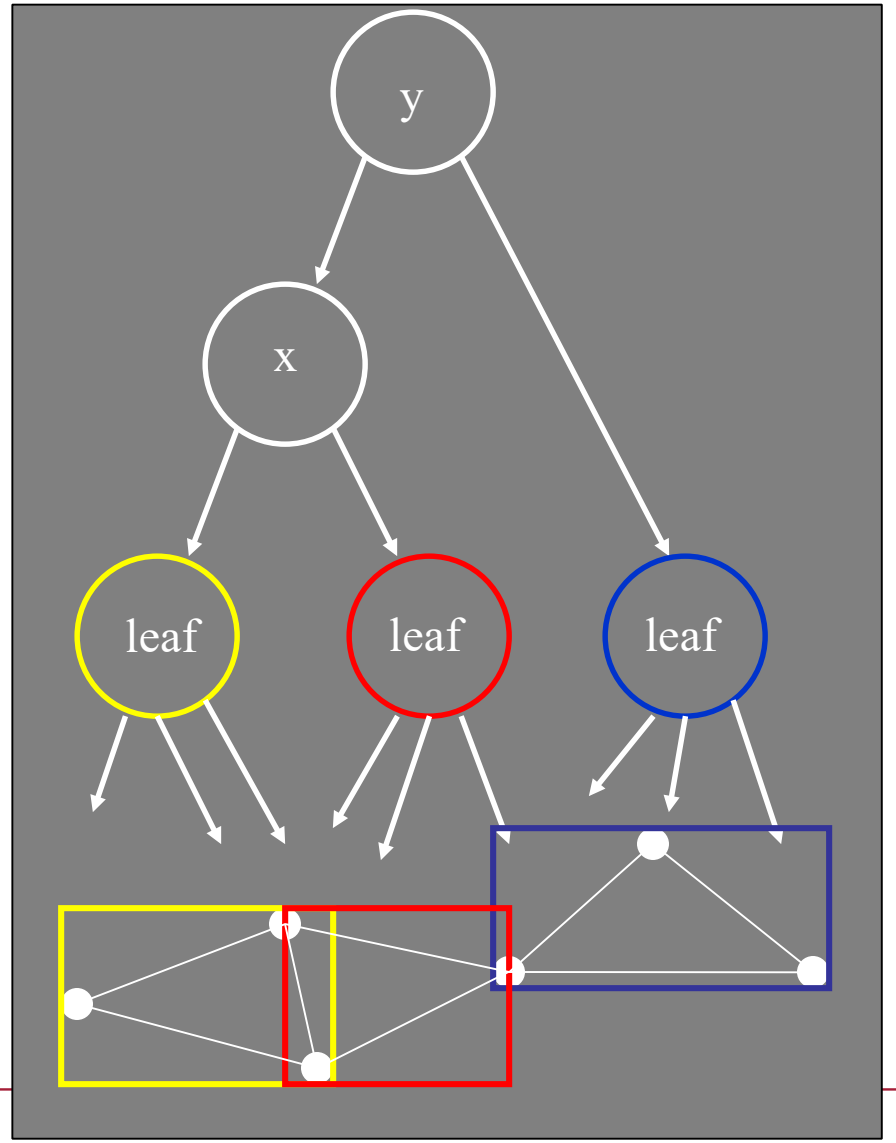
Update of B-KD Trees

- Leaf Node
  - Fetch vertices



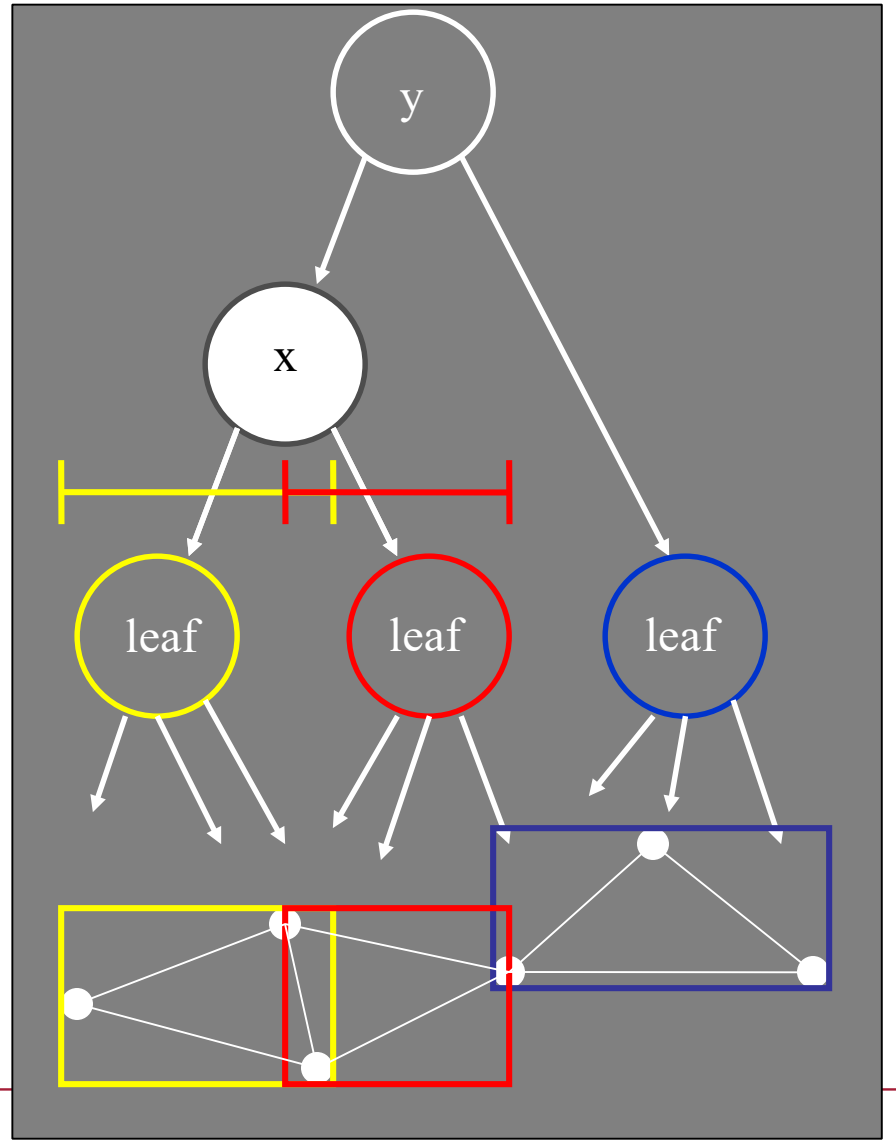
Update of B-KD Trees

- Leaf Node
  - Fetch vertices
  - Compute leaf boxes



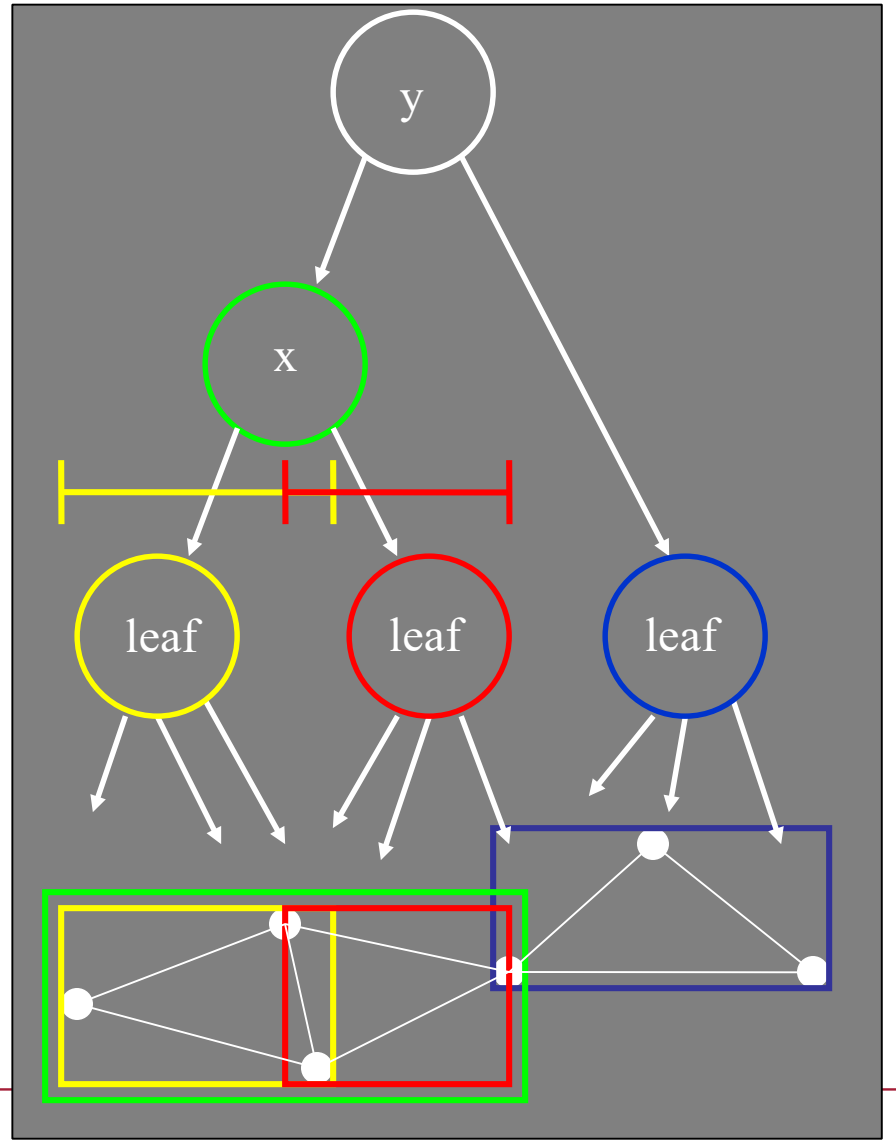
Update of B-KD Trees

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



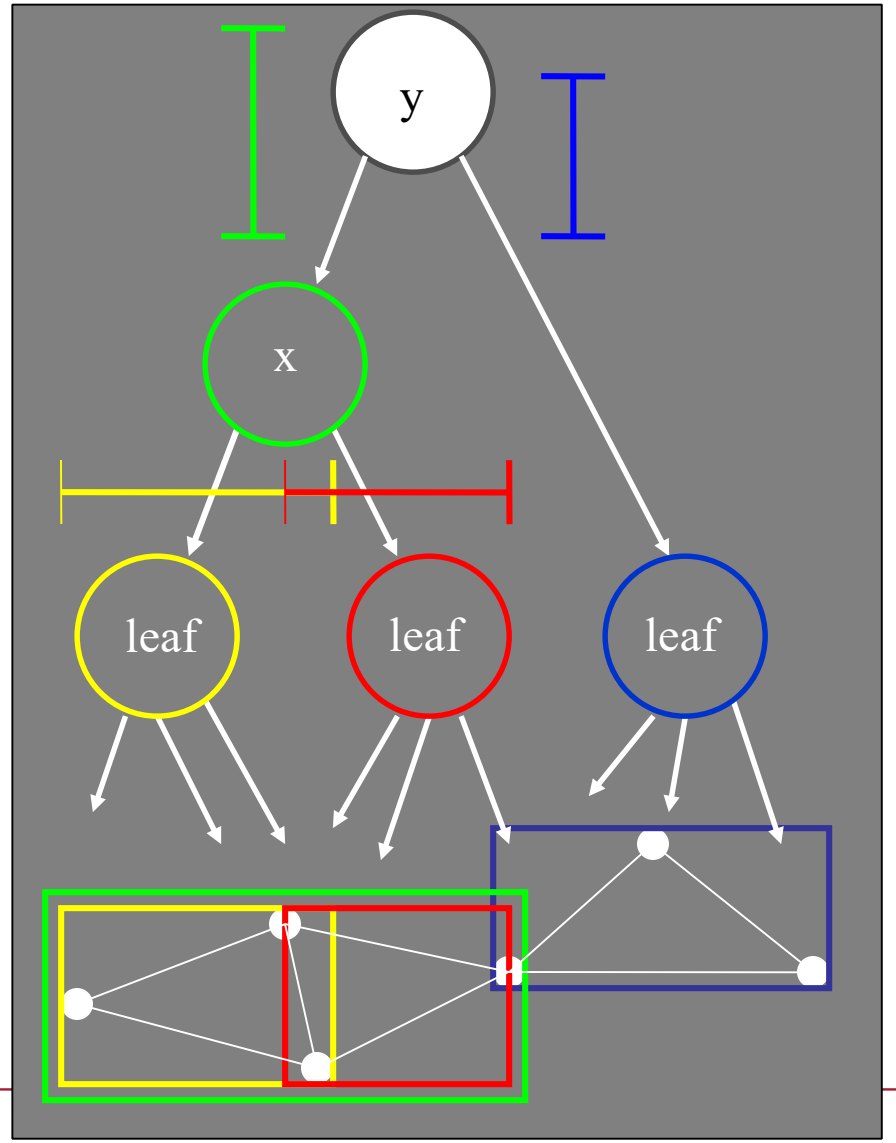
Update of B-KD Trees

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



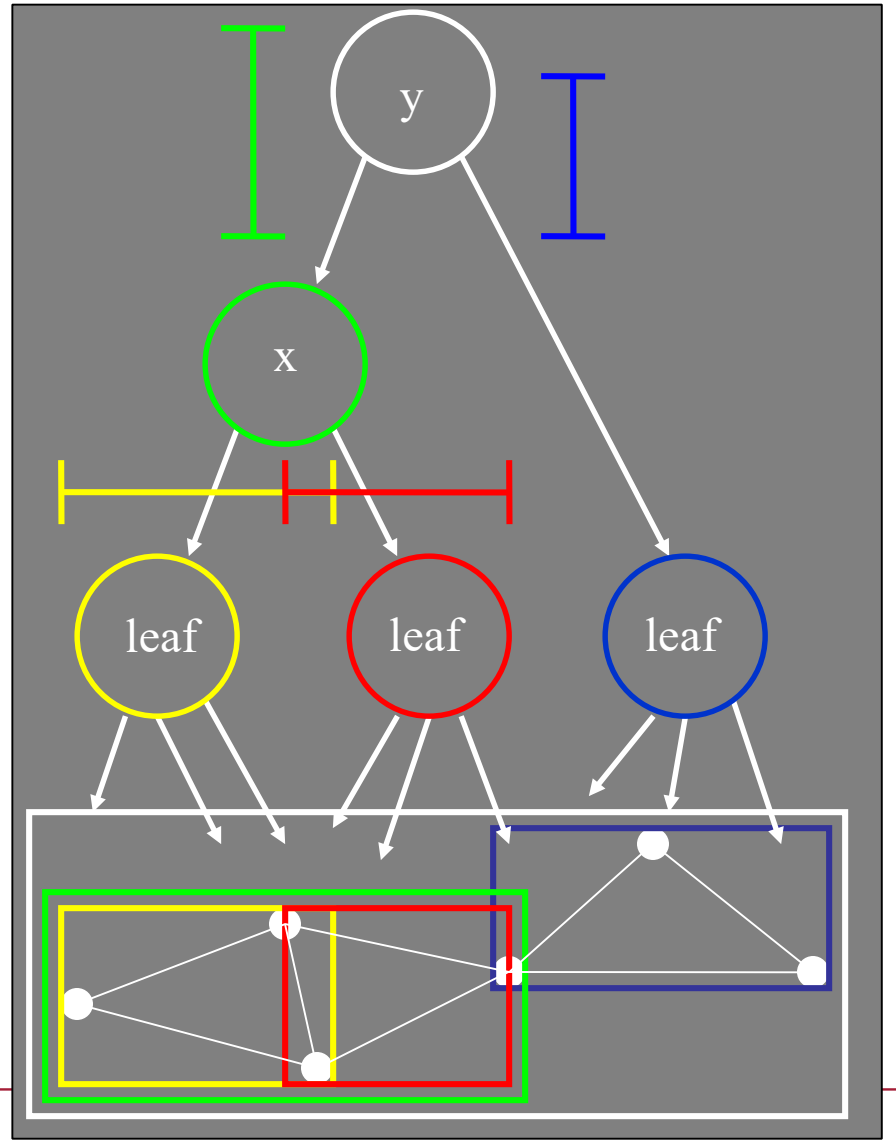
Update of B-KD Trees

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

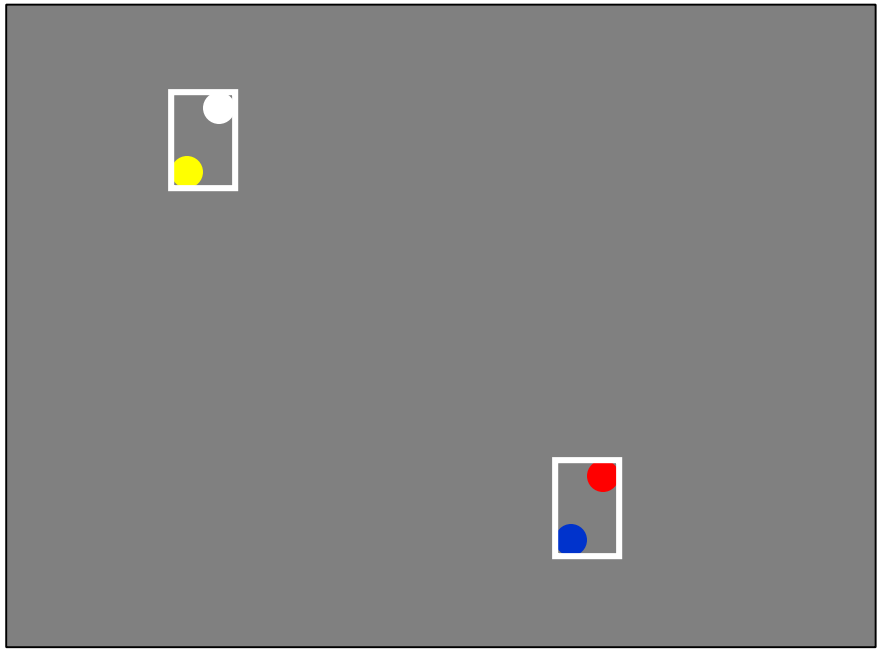


Update of B-KD Trees

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

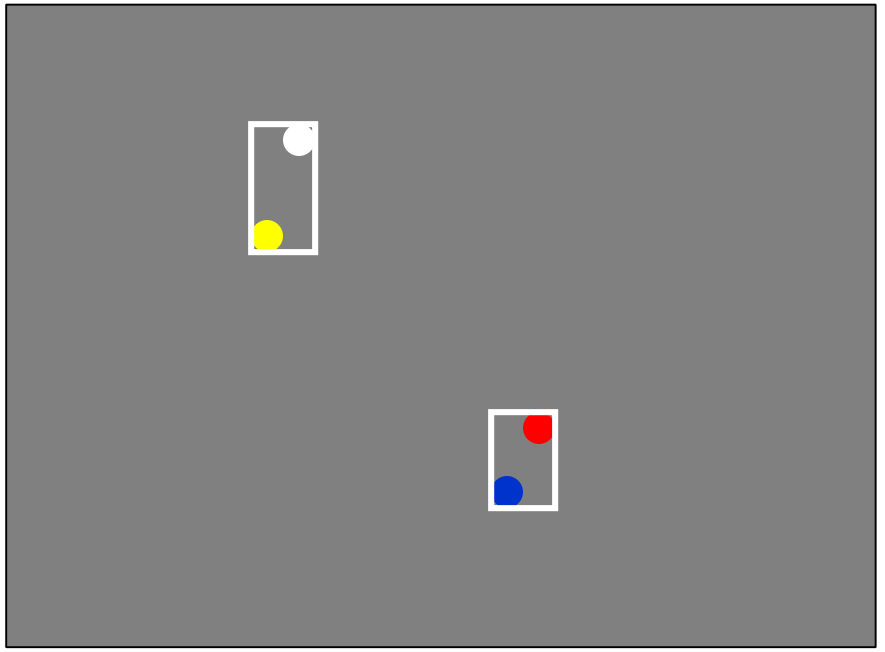


Examples



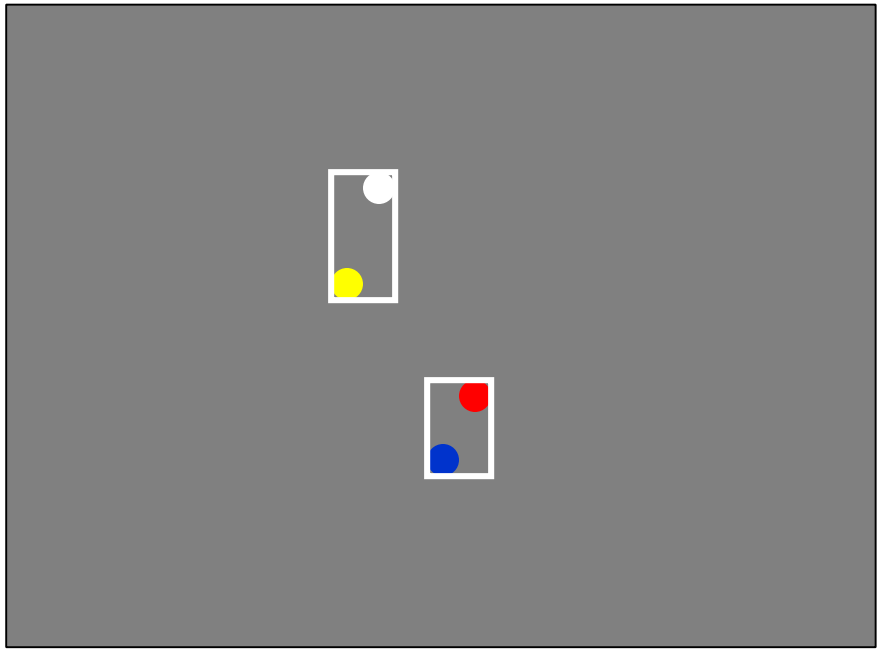
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



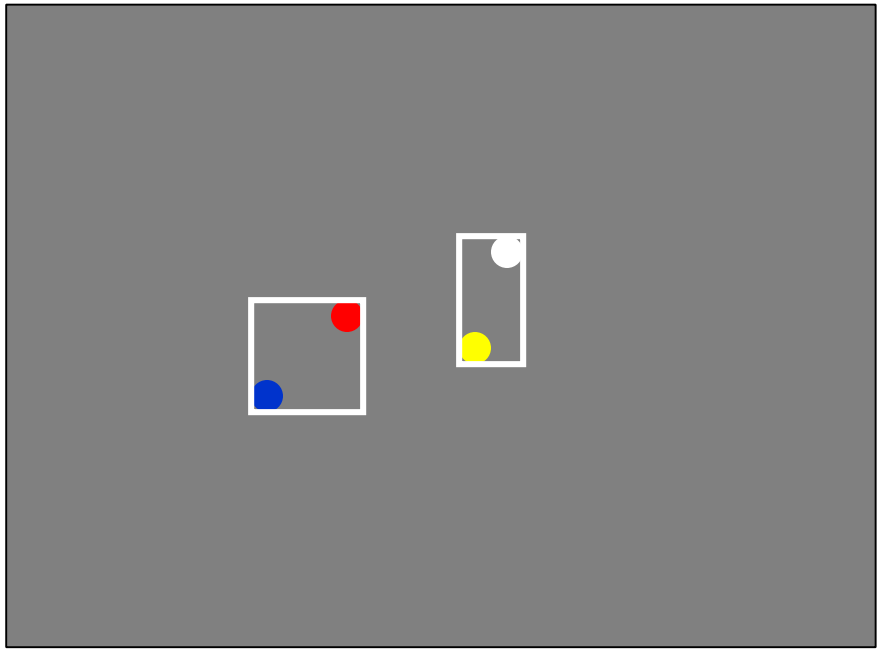
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



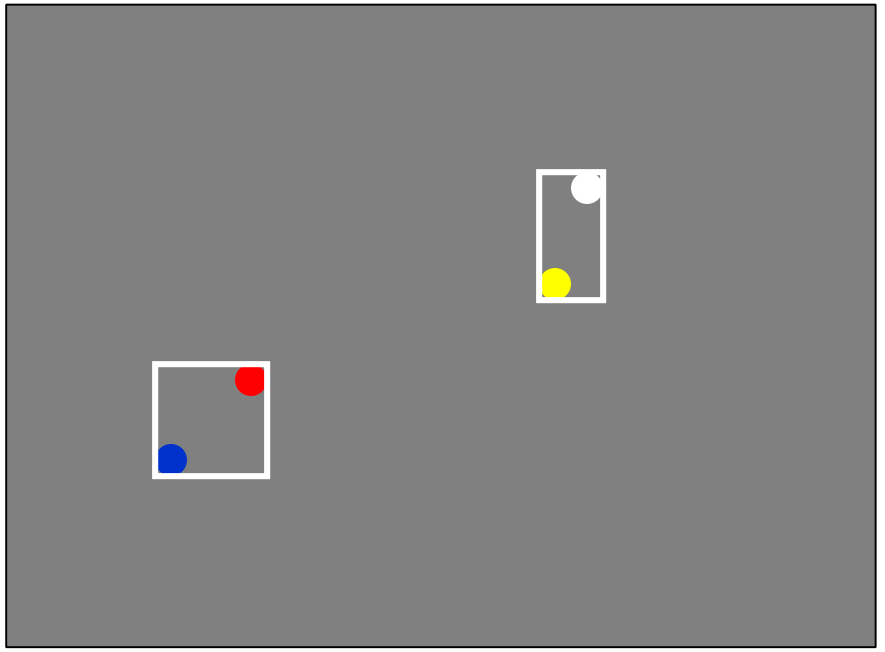
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



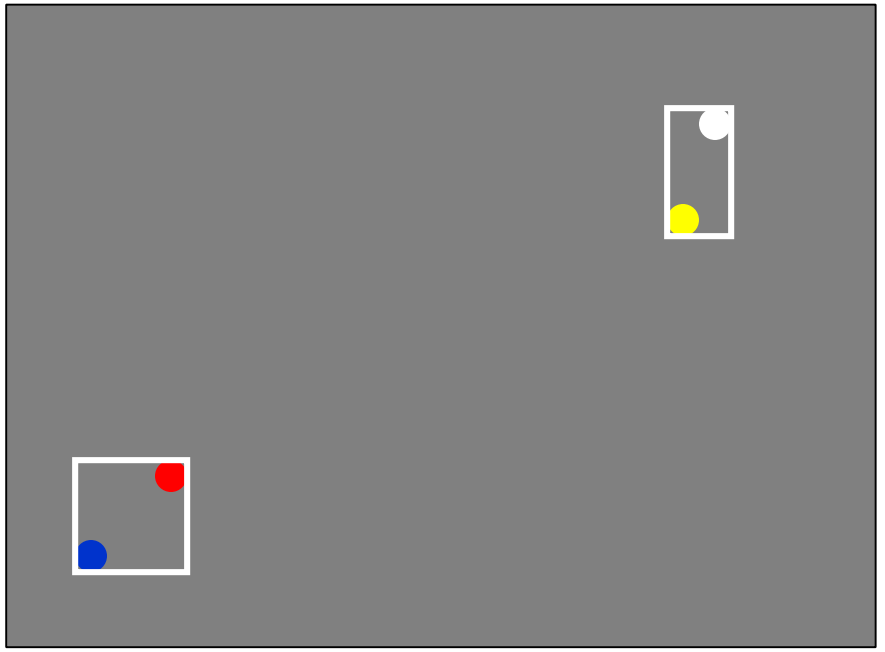
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



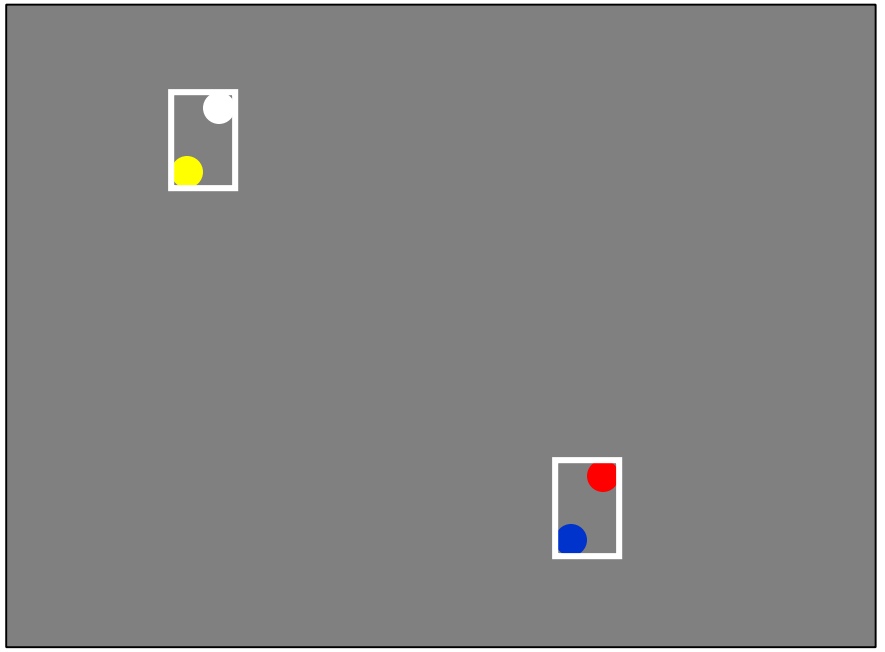
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



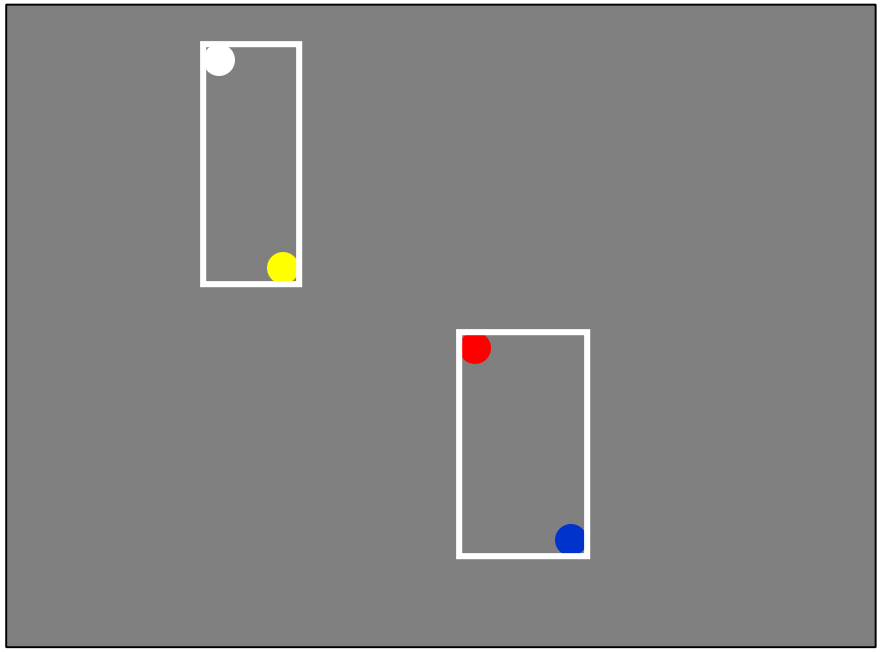
- Bounding approaches perform well for
  - Continuous motion
  - Structure of motion must match tree structure
  - E.g. skinned meshes, characters, water surfaces, ...

Examples



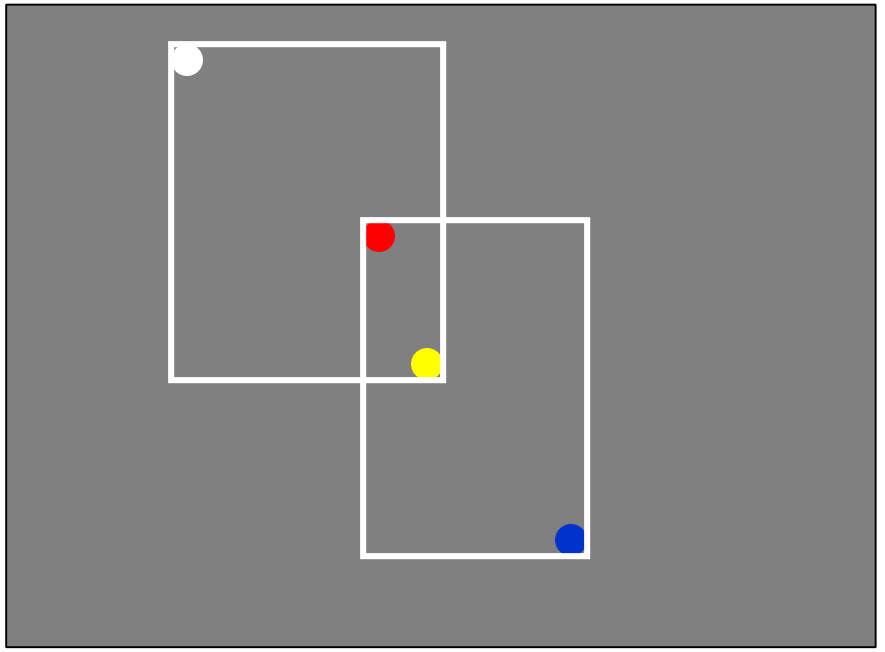
- Bounding volume approaches are less efficient for
  - Non-continuous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

Examples



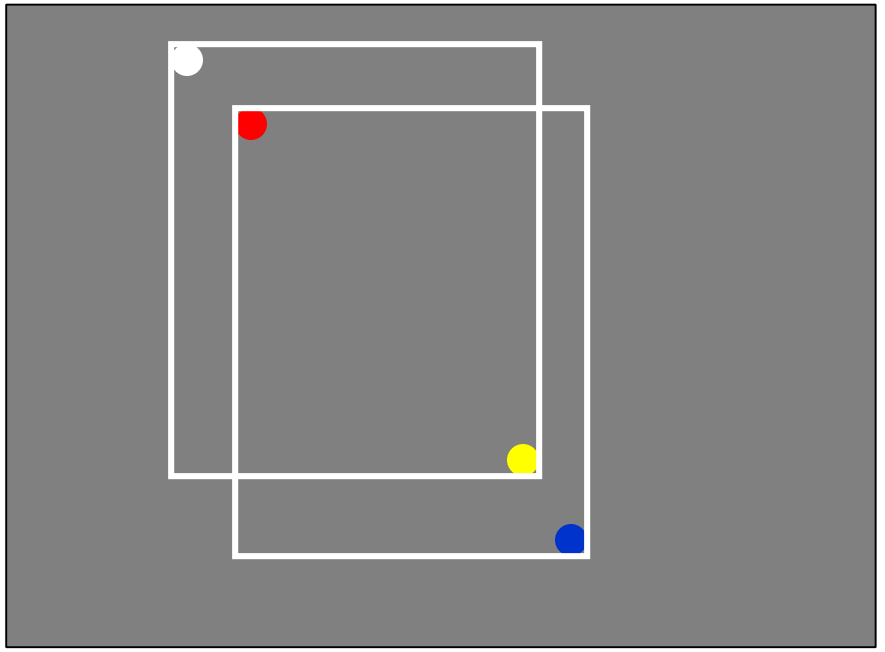
- Bounding volume approaches fail for
  - Non-continuous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes

Examples



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

Examples



- Bounding volume approaches fail for
  - Non-continuous motion
  - Structure of motion does not match tree structure
  - High traversal cost due to large overlapping boxes
    → might require to rebuild B-KD tree structure or parts of it

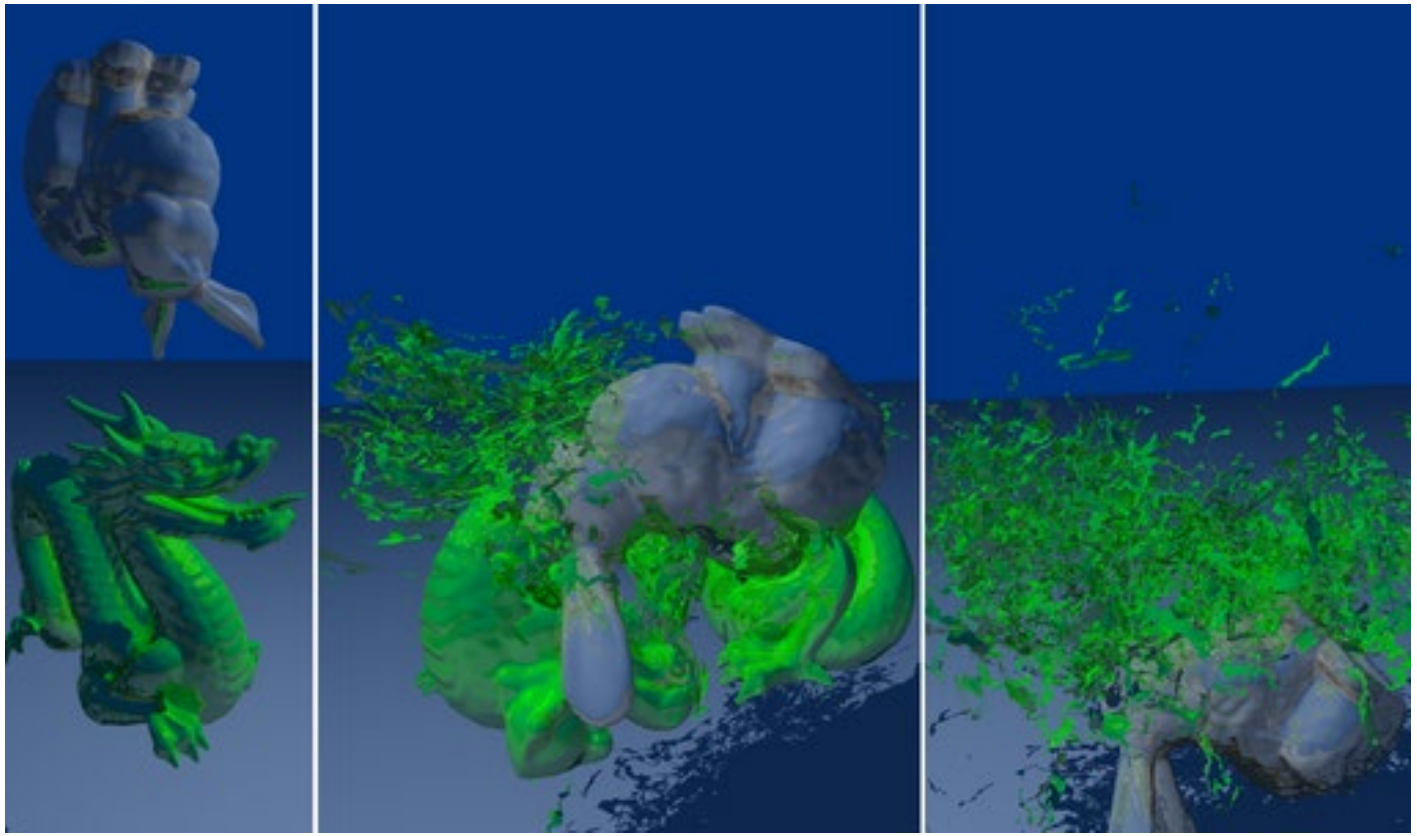
BVH Updates

- 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 BVH over entire animation
  - Build BVH hierarchy with respect to best partitioning over all animation frames
  - Usually not much improvement
    - Shows that hierarchies stay good during animations

Fast Dynamic Ray-tracing

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- [Yoon et al. EGSR 2007]
  “Ray Tracing Dynamic Scenes using Selective Restructuring”
- massive dynamic scenes
- based on B-KD



Summary

- Large scenes
- Dynamic scenes
- Dynamic update of data structures

Questions

- Compare lazy build and multi-level hierarchies for representing large scenes.
- How can one adapt BVHs for dynamic scenes?

Next Lecture

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Light transport - Shading

References

- Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays, Holger Dammertz, Johannes Hanika and Alexander Keller, p. 1225-1234 in: Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering), 2008

- Two-Level ray Tracing with Reordering for Highly Complex Scenes, Johannes Hanika, Alexander Keller, and Hendrik Lensch, p. 145-152 in: Proceedings of Graphics Interface 2010.

- Ray Tracing Dynamic Scenes using Selective Restructuring, Sung-Eui Yoon, Sean Curtis, Dinesh Manocha . In: Computer Graphics Forum (Proc. 19th Eurographics Symposium on Rendering), 2007