Kinetic Bounding Volume Hierarchies for Deformable Objects

René Weller
Clausthal University of Technology, Germany
weller@in.tu-clausthal.de

VRCIA ’06, June 2006, Hong Kong
Motivation

- Bounding volume hierarchies (BVHs) are widely employed in many areas of computer science to accelerate geometric queries
  - ray-tracing
  - occlusion culling
  - collision detection
Deformable BVH

- BVHs are constructed in a pre-processing step
- The pre-processed hierarchy becomes invalid when the object deforms
  → The BVH must be rebuilt or updated after deformations
Brute Force Update of single BV

Frame 2

Max
x 1.0
y 0.9

Min
x 0.3
y 0.4
Problems

- Discrete time sampling
  - Many update operations
  - Missing changes between queries
- No adequate use of spatial and temporal coherence
- Other approaches:
  - Hybrid updates [van den Bergen, 1998]
  - Lazy updates [Mezger et al. 2003]
  - Restriction of deformation schemes [James and Pai, 2004]
  - Intrinsic collision test on the GPU [Wong and Baciu 2005]
  - Chromatic decompositions [Govindaraju et al. 2005]
Our Approach

- Motion in the physical world is normally continuous
- Changes in the **combinatorial structure** of the BHVs occur only at discrete time points
  → We store only the combinatorial structure of the BVH and use an event based approach for updates
Kinetic Updates

Event Queue
(t1, Q, R, Max x)
Advantages

- Fewer update operations
- Valid BVHs at every point in time
- Independent of query sampling frequency
- Can handle all kinds of objects
  - polygon soups, point clouds, and NURBS models
- Can handle insertions/deletions during run-time
- Can handle all kinds of deformations
  - Only a flightplan is required for every vertex
  - These flightplans may change during simulation
Recap: Kinetic Data Structures

- **KDS** are a framework for designing and analyzing algorithms for objects in motion [Basch et al. 1997]
- KDS framework leads to event-based algorithms that samples the state of parts of a system only as often as necessary for a special task (e.g. a bounding box)
KDS terminology

- The task is called the **attribute**
- A KDS consists of **certificates**
- Certificate failures are called **events**
- If the attribute changes at the time of an event, the event is called **external**, otherwise **internal**
Quality of a KDS

- A KDS is **compact**, if it requires only little space
- A KDS is **responsive** if we can update it quickly in case of a certificate failure
- A KDS is **local**, if one object is involved in not too many events
- A KDS is **efficient**, if the overhead of internal events with respect to external events is reasonable
Kinetic AABB Tree

- Kinetization of the AABB tree
- Pre-processing: Build the tree by any algorithm suitable for static AABB trees
  - It is only required that the height of the BVH is logarithmic
- Store with every node the indices of those points that determine the BV
- Initialize the event queue
Kinetic AABB Tree Events

- Leaf Event

Event Queue
(t1, Q, R, Max x)
Kinetic AABB Tree Events

- Tree Event

- Flightplan Update Event

Event Queue
(t1, R, P, Max x)
Simulation Loop

while simulation runs

determine time $t$ of next rendering

e ← min event in event queue

while $e\cdot$timestamp < $t$

    processEvent($e$)

    e ← min event in event queue

check for collisions (or cast ray, or ...)

render scene
Event Handling

- Leaf Event
Event Handling

- Leaf Event cont

Event Queue
(t5, T, R, Max x)
Analysis

- **Theorem 1:** The kinetic AABB tree is compact (\(O(n)\)), local (\(O(\log n)\)), responsive (\(O(\log n)\)) and efficient. Furthermore, the kinetic AABB tree is a valid BVH at every point of time.

- **Theorem 2:** Given \(n\) vertices, we assume that each pair of flightplans intersect at most \(s\) times. Then, the total number of events is in nearly \(O(n \log n)\).
Kinetic Boxtree

- Kinetic AABB tree needs up to six events for every BV

=> The kinetic Boxtree which uses less memory than the kinetic AABB tree

- Combination of k-d tree and AABB
Event Computation

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

1, 2, 5
1, 2
5
1 2

3, 4, 6
3, 4
5 6
1 2 3 4

7, 8, 9, 10, 11
7, 8, 9
10, 11
7
10 11
7
8 9

1, 2, 3, 4, 5, 6
1 2 3 4 5 6

Kinetic BoxTree
BoxTree Events

\[ \text{split}_x \]
\[ \text{split}_y \]
\[ \text{split}_x \]
\[ \text{split}_y \]
Analysis

- Theorem: The kinetic BoxTree is compact, local and efficient. The responsiveness holds only in the one-dimensional case. Furthermore, the kinetic BoxTree builds a valid BVH at every point of time.
Test Scenes
Results

#Events and #Updates

![Graph showing #Events and Updates](image-url)
Updating time

![Graph showing performance comparison between different methods.]
Total time (= Update + Collision Check)

- kinetic AABB
- kinetic Boxtree
- Bottom-Up
Conclusions

- Two novel data structures for updating a BVH over deformable objects fast and efficient
- Efficiency due to event based approach
- Theoretic Analysis:
  - Upper bound of nearly $O(n \log n)$ for the updates that are required to keep a BVH valid
  - Our kinetic AABB tree and kinetic BoxTree are optimal in number of updates
- Up to 20 times faster than bottom-up updates in practically relevant scenarios
Future Work

- Use our kinetic Data Structures also for continuous collision detection
- Utilize our data structures for other kinds of motion
  - physically-based simulations
  - other animation schemes
- Use our KDS for other applications like ray-tracing or occlusion culling
Acknowledgements

- Gabriel Zachmann, Clausthal University of Technology
- Johannes Mezger, University of Tübingen
- Stefan Kimmerle, University of Tübingen
- DFG grant ZA 292/1-1 ("Aktionsplan Informatik")