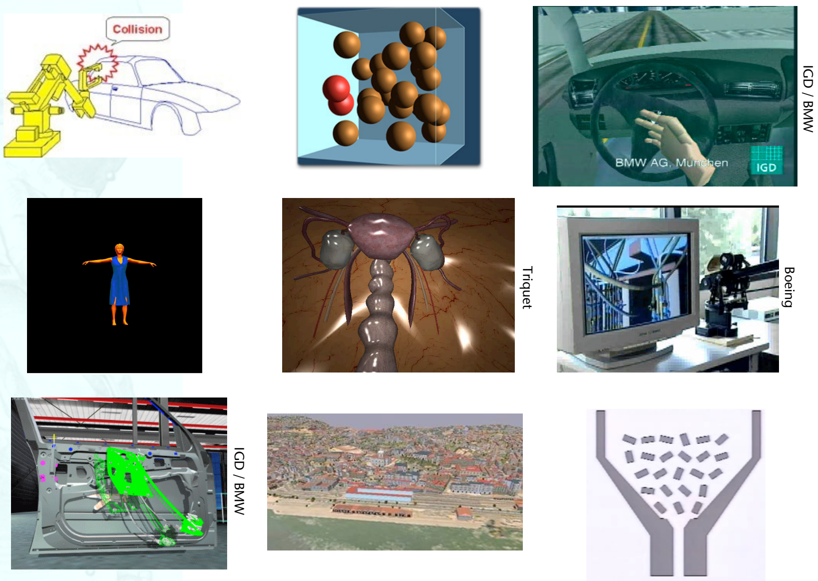
Collision Detection as a Fundamental Technology in VR Based Product Engineering

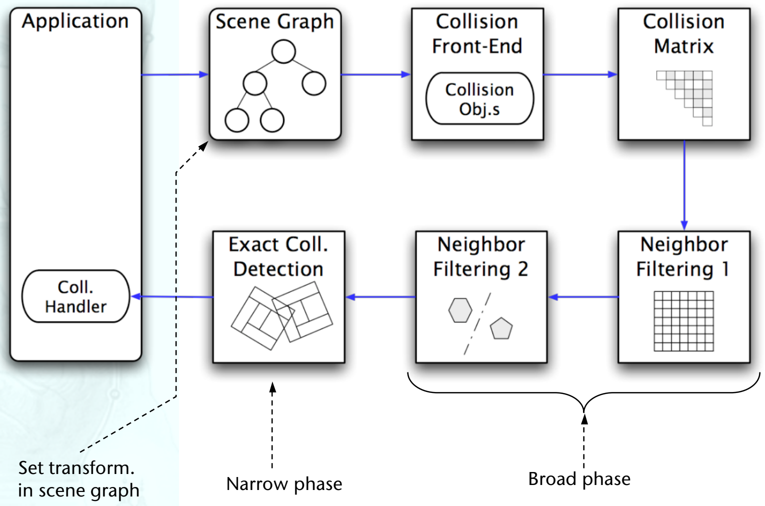
Prof. Gabriel Zachmann
Clausthal University, Germany
zach@tu-clausthal.de

2nd Advanced Study Institute on "Product Engineering: Tools and Methods based on Virtual Reality Technology"
Chania, Creete, 30 May — 6 June 2007

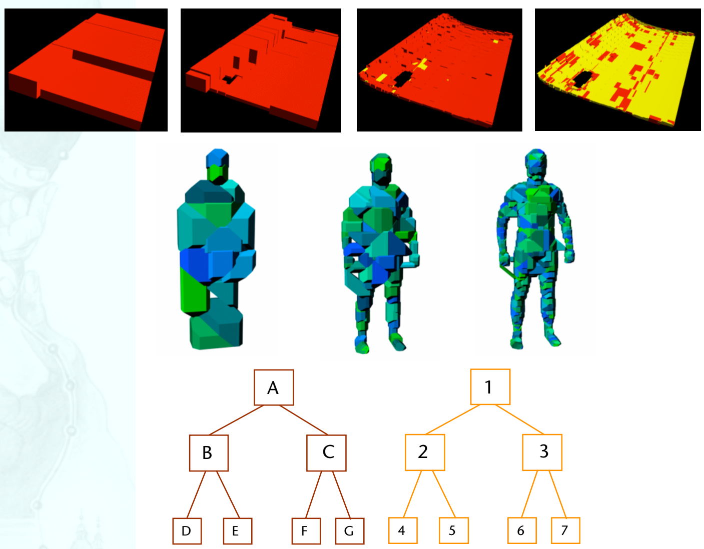
Motivation



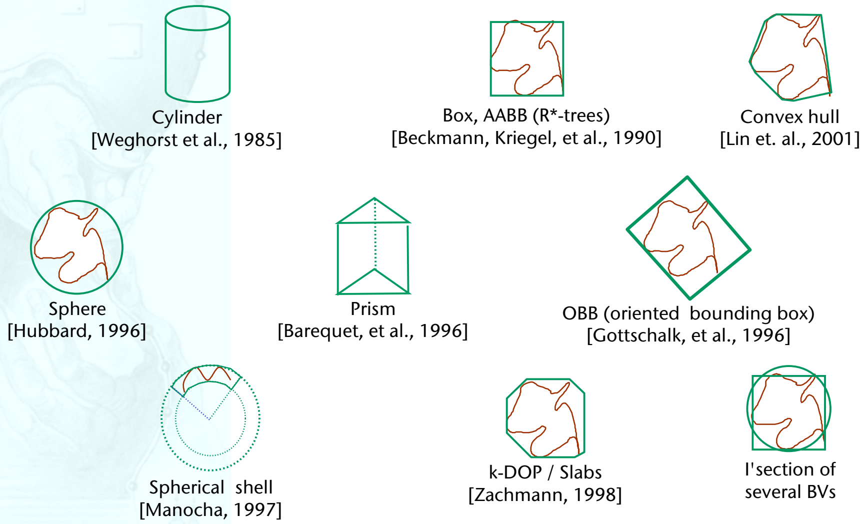
Collision Detection Pipeline



Hierarchical Collision Detection

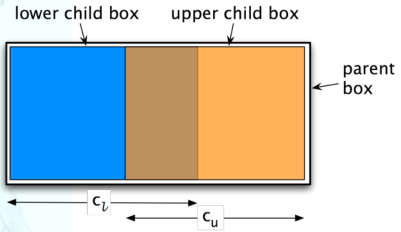


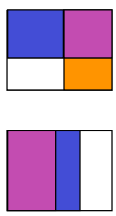
Examples of Common Bounding Volumes



Restricted Boxtrees / SKD-Trees

- Observation: on most sides, child boxes almost touch parent box
- Combination of kd-tree and AABB / generalization of kd-tree



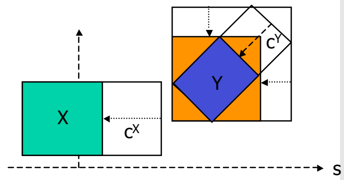


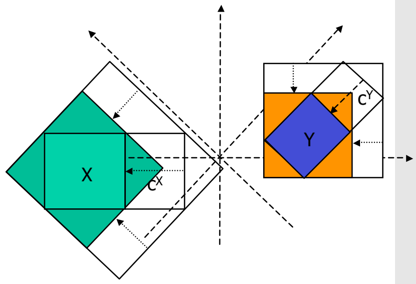
- Minimal storage: 2 floats, 2 axes IDs, 1 pointer

Intro BoxTrees GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook

Overlap Tests

- Re-alignment:
  - 12 FLOPs
- SAT:
  - 82 FLOPs
- SAT lite:
  - 24 FLOPs
- Sphere test:
  - 29 FLOPs





Constructing Restricted Boxtrees

- Approach: top-down
  - Compute BV covering input
  - Split input into two subsets
- Splitting criterion:
  - Expected traversal cost:

$$C(X, Y) = 4c + \sum_{i,j=1,2} P(X_i, Y_j) C(X_i, Y_j)$$

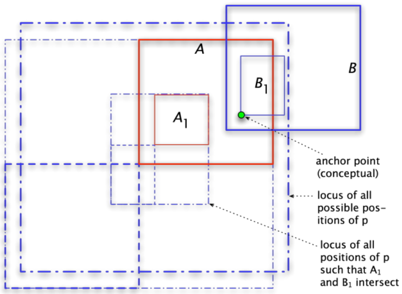
$$\approx \sum_{i,j=1,2} P(X_i, Y_j) N_i N_j$$

Intro BoxTrees GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook

- Estimation of $P(X_i, Y_j)$:

$$A_i \cap B_j \neq \emptyset \;\Leftrightarrow\; \mathbf{p}_i \in A_i \oplus B_j$$

$$P(A_i, B_j) = \frac{V(A_i \oplus B_j)}{V(A) \oplus V(B)} \approx \frac{V(A_i) + V(B_j)}{V(A) + V(B)}$$

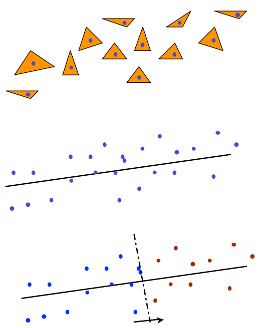


- Overall criterion: split set of polygons so as to minimize

$$V(X_1)N_1 + V(X_2)N_2$$

Construction Algorithm

- Represent polygons by points
- Compute axis of largest variance (PCA)
- Sort along that axis
- Sweep plane along that axis, looking for minimum of split heuristic
- Split set of polygons



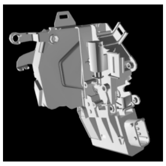
- Complexity:

$$T(n) = T(\alpha n) + T((1-\alpha)n) + n \log n \;\in\; O(n \log^2 n)$$

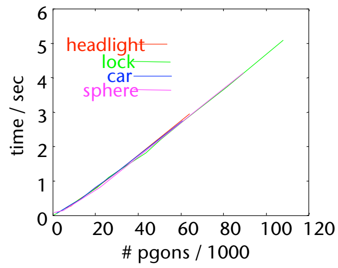
Results

- Suite:
- Platform:
  - Pentium III, 1GHz
- Construction:

Door lock (BMW)

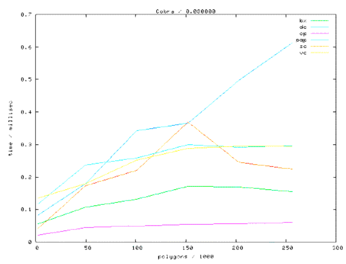


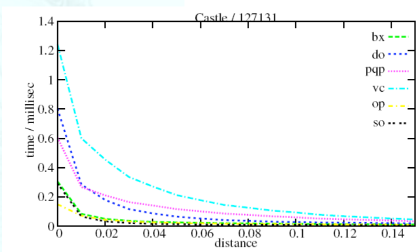
Car (courtesy VW)

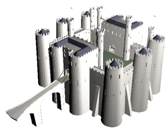


Results

BVH	Bytes / BV
Restricted Boxtree	8
Sphere Tree	20
AABB Tree	28
OBB Tree	52
24-DOP-Tree	100



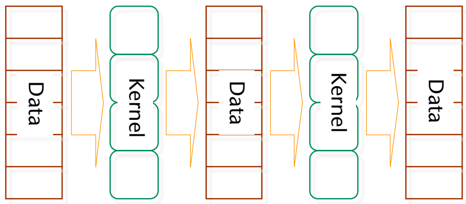




Object-Space Coll. Detection on the GPU

- Background on streaming architectures (and GPUs):
  - Stream Programming Model =
    "Streams of data passing through computation kernels."
  - *Stream* = ordered, homogenous set of data of arbitrary type
  - *Kernel* = program to be performed on *each* element of the input stream
- Sample stream program:

```
{
   stream A, B, C;
   ...
   kernelfunc1( input: A,
                output: B);
   kernelfunc2( input: B,
                output: C);
  ...
}
```

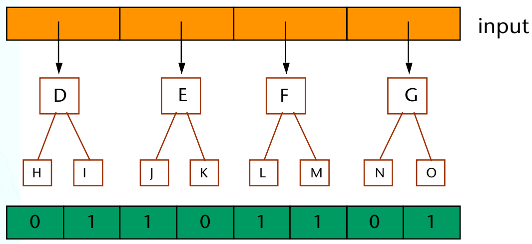


Application to Collision Detection

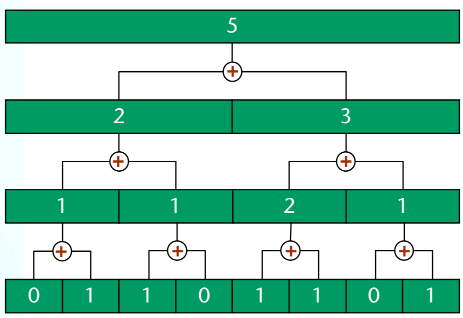
- Elementary operation during traversal is overlap test of BVs
- Different pairs of BVs on same level can be tested independently
- Idea: implement as fragment program
  - Texture = set of pairs of BVs to be tested
  - Output = new texture
- Problem:
  - Conceptually 1 execution unit per output element
  - "0"s are still in the output
  - Need to "pack" output texture tightly after each level

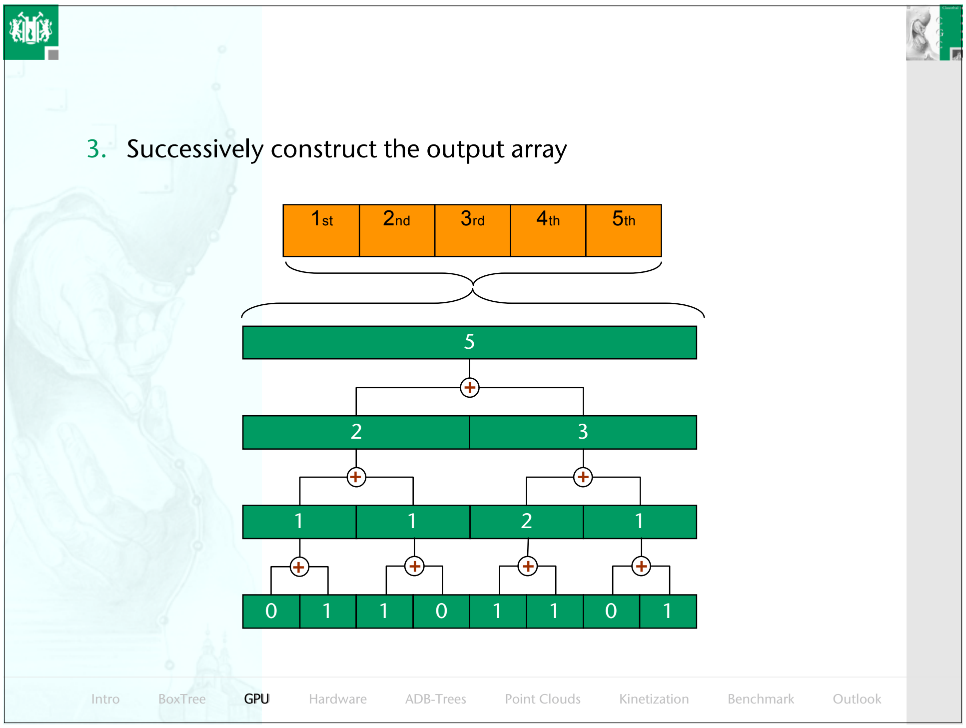
- Assumption (for sake of simplicity):
  - 1D Array (texture)
  - Element = pair of BVs
  - Result of 1 overlap test = overlap status for 2 child pairs

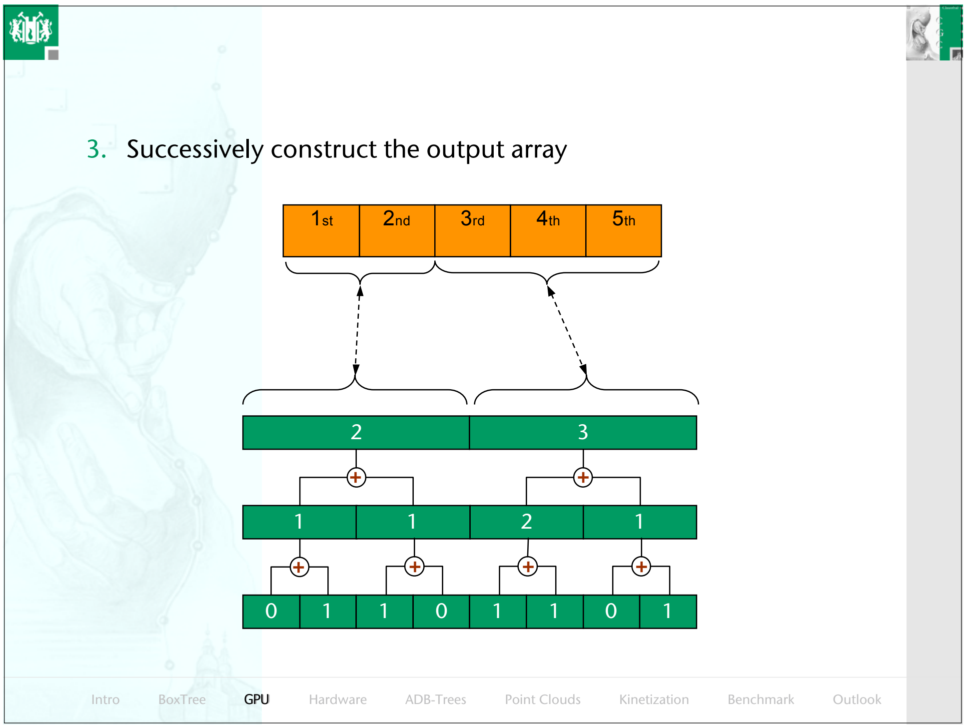
1. Step: calculate overlap status for all potential child pairs



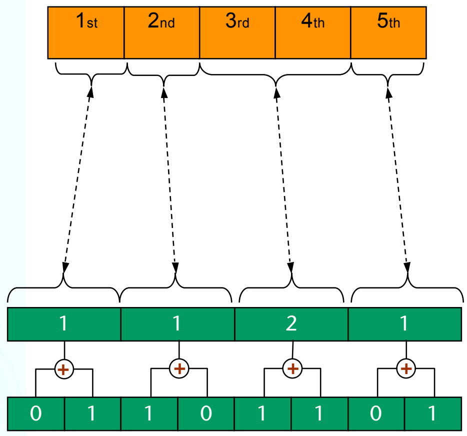
2. Build a tree by summing up overlap counts
   - corresponds to a mip-map; total size $O(n)$



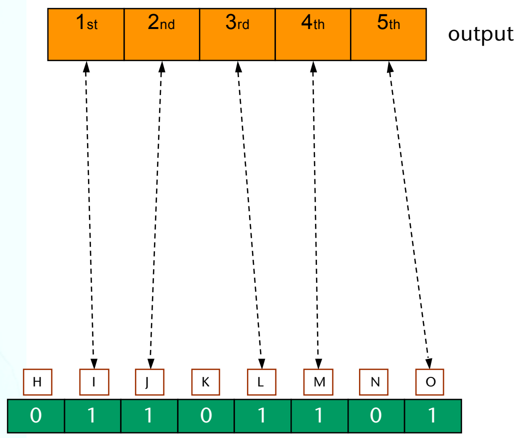

3. Successively construct the output array
1st 2nd 3rd 4th 5th
5
2 3
1 1 2 1
0 1 1 0 1 1 0 1
Intro BoxTree GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook


3. Successively construct the output array
1st 2nd 3rd 4th 5th
2 3
1 1 2 1
0 1 1 0 1 1 0 1
Intro BoxTree GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook

3. Successively construct the output array

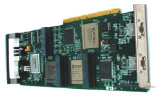


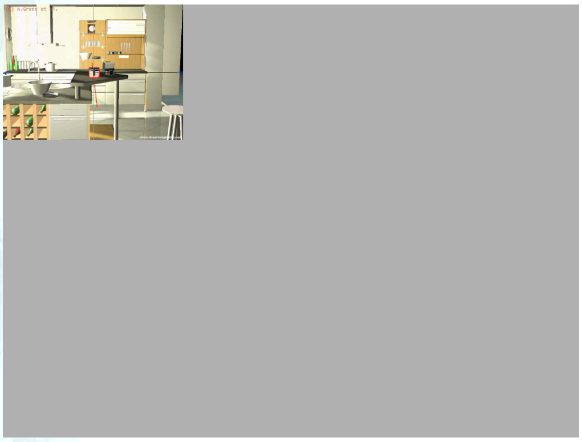
4. Copy data for child BV pairs to output array



Watch the GPU

- Top left = interactive scene
- Other panels = textures

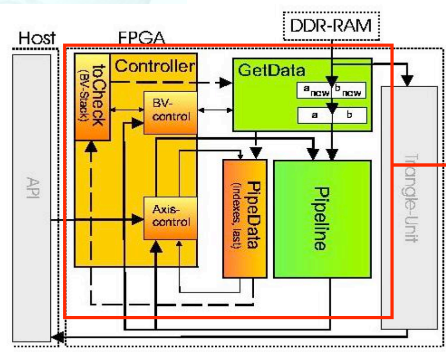


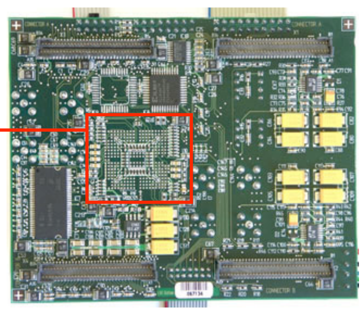


Dedicated Hardware for Coll.Det.

universitätbonn

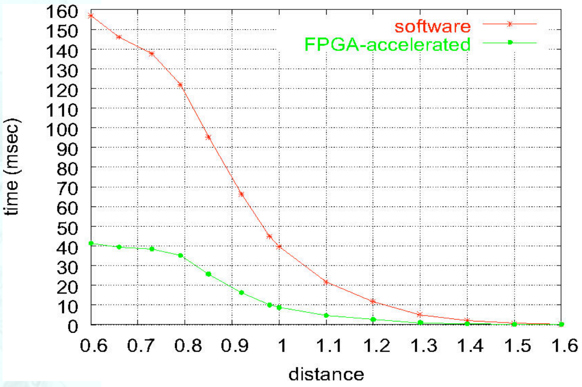
- General problem of "general purpose" computations on the GPU — competition among resources





- FPGA board (Xilinx Virtex II Pro) for prototyping

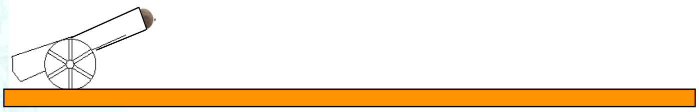
Results



- FPGA implementation has no cache yet(!)
- FPGA is much slower than ASIC (100 MHz, slow mem interface)
- With FPGA, the CPU is completely idle

Stochastic Collision Detection

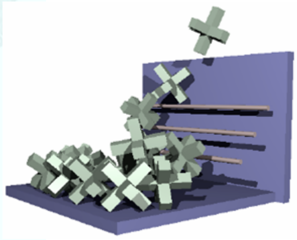
- An Experiment



Set of "plausible paths" for a cannonball.

Motivation

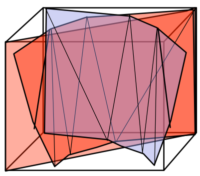
- Observation: absolute accuracy is often not necessary



→ New notion: approximative collision detection

- Goal: continuous and controlled balancing between running time and accuracy
- Benefit: time-critical computation

"Gedankenexperiment"



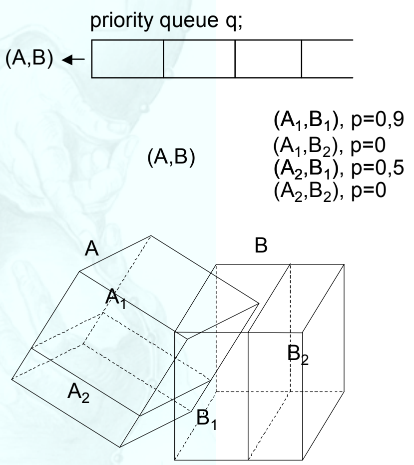
ADB-Trees

- Average-Case approach:
  - Estimate probability of intersection for whole sets of polygons (at inner nodes of BVH)

→ BVH traversal guided by probability (P-Queue)

- Modification of BVHs: store simple description
- Advantage of our approach: can be applied to (almost) any kind of BVH / hierarchical collision detection

Probability-Guided BVH Traversal

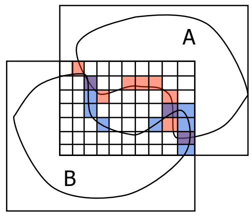


Traverse(A,B)

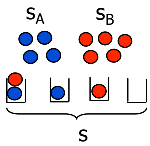
```
p-queue q
q.insert(A,B,1)
while q not empty
  A,B ← q.pop
  forall Ai, Bj
    p ← Pr[ collision in Ai, Bj ]
    if p ≥ pmin
      return "collision"
    if p ≥ 0
      q.insert(Ai, Bj, p)
return "no collision"
```

Estimating the Probability

1. Partition A ∩ B with grid of s cells
2. Compute number of "well-filled" cells: s_A
3. Dito for B: s_B
4. Compute probability that x cells are "well-filled" from both A and B:



$$Pr[c(A \cap B) \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{s_B}{t}\binom{s-s_B}{s_A-t}}{\binom{s}{s_A}}$$



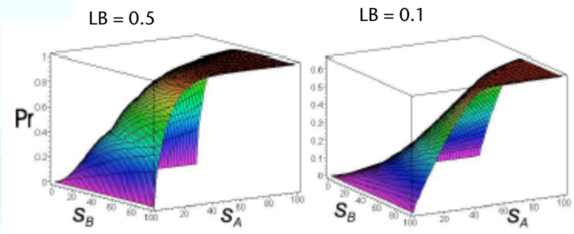
Taking curvature into account

- So far, $Pr[c(A \cap B) \geq x]$ only yields number of collision cells, not collisions (i.e., cells with actual intersections)
- Curvature of surfaces A and B in collision cell ...
  - Low → intersection likely
  - High → not likely
- Heuristic:
  - Cell large/small compared to object → surface in cell can make many/few "bends"
  - Depth *d(A)* , *d(B)* of BVs *A, B* indicates size of cells
- "Curvature factor":

$$LB(A,B) := \frac{d(A) + d(B)}{d^{\max}(\text{Obj } 1) + d^{\max}(\text{Obj } 2)}$$

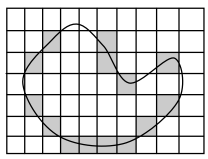
- Overall function to estimate probability of intersection:

$$\max_{x \leq \min\{s_A, s_B\}} \left\{ Pr[c(A \cap B) \geq x] \cdot \left(1 - (1 - LB(A \cap B))^x\right) \right\}$$



Efficient Estimate

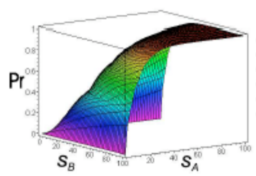
- Preprocessing:
  - Partition each BV of BVH by grid
  - Count number s_A of well-filled cells
  - Store with each node of BVH



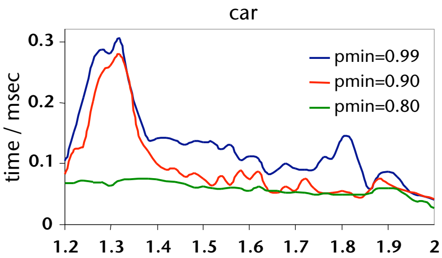
- At runtime estimate s'_A and s'_B :

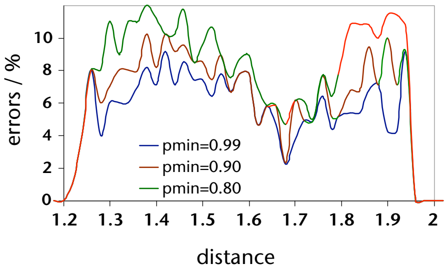
$$s'_A \approx s_A \frac{\mathrm{Vol}(A \cap B)}{\mathrm{Vol}(A)}$$

- Precompute LUT for function *Pr* for all possible input values
- Evaluate only for x ≤ 10



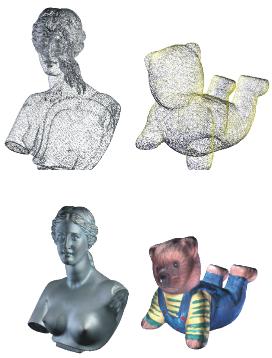
Results





Point Clouds

- Motivation: renaissance of points as object representation because of 3D scanners



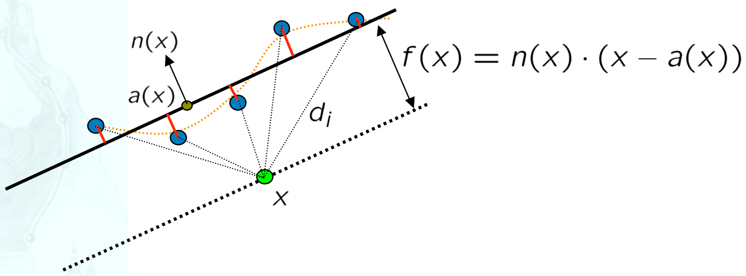
- Goal:
  - Fast collision detection between 2 given point clouds
  - No polygonal reconstruction

Definition of the surface

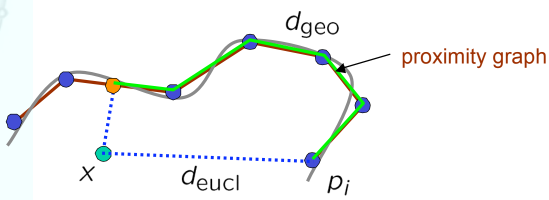
- Idea: assume "distance function" f from surface, then surface S is

$$S = \{x \in \mathbb{R}^3 \mid f(x) = 0\}$$

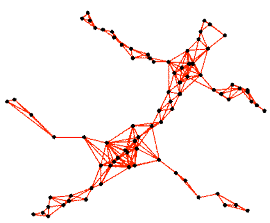
- "Distance" function f by Weighted Least Squares:



- Cause and solution:

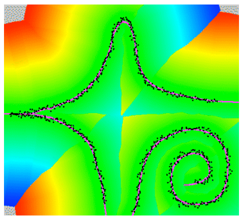


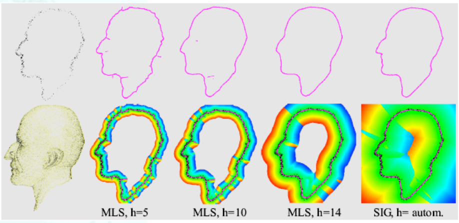
- Which neighborhood graph?
  → k-SIG (sphere-of-influence graph)



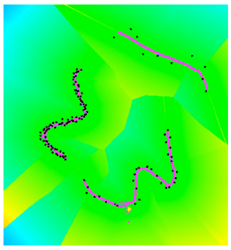
Benefits

- Much less artifacts
- Automatic, sampling-density independent detection of boundaries
- Automatic kernel bandwidth selection → handles different sampling densities automatically

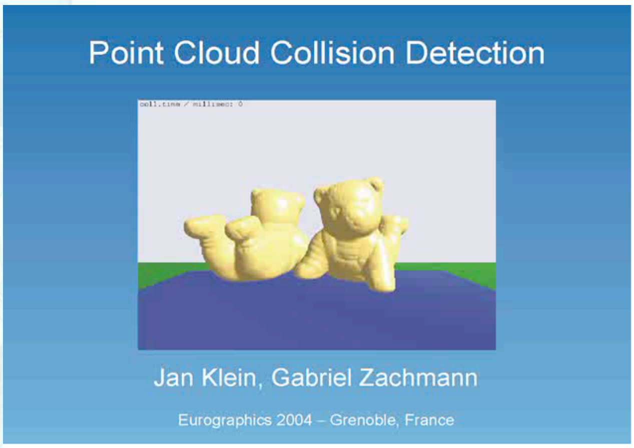




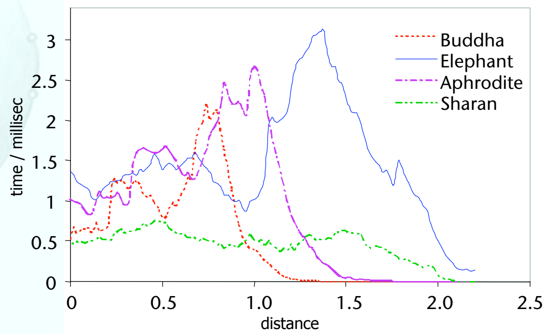
orig. PC old method our method

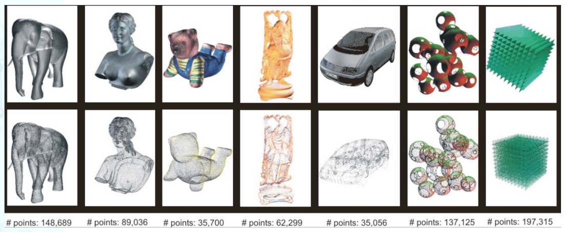


CD using Point Cloud Hierarchies



Results





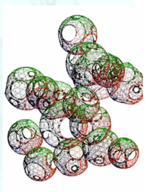
Coll.Det. of PCs using Stochastic Sampling

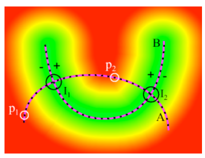
- Given two point clouds A and B (or subsets thereof), construct a sampling of

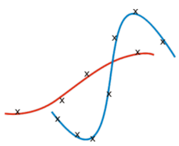
$$\mathcal{Z} = \{x \mid f_A(x) = f_B(x) = 0\}$$

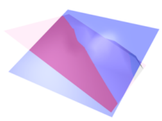
- Overall method:

A,B → $(p_i,p_j) \in A$ on different sides of B → Approx. intersection points → Refined intersection point

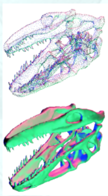




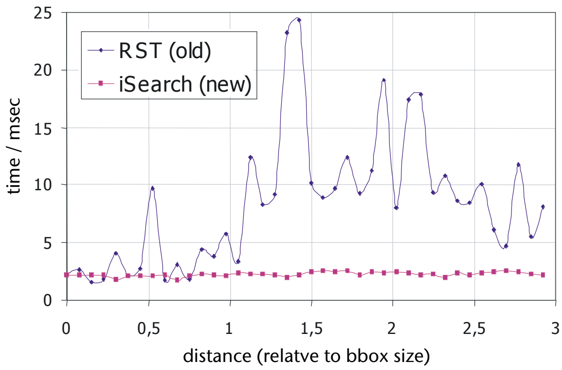




Results



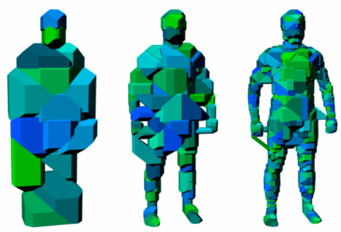
28,000 points



- Theoretical complexity: $O(\log \log N)$

Kinetic Bounding Volume Hierarchies

- For collision detection of deformable objects …
- … but not just for collision detection!
  - Can be applied to ray-tracing, occlusion culling, etc.

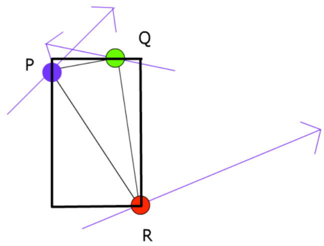


- Pre-processed hierarchy becomes invalid when object deforms
  → BVH must be rebuilt or updated after deformations

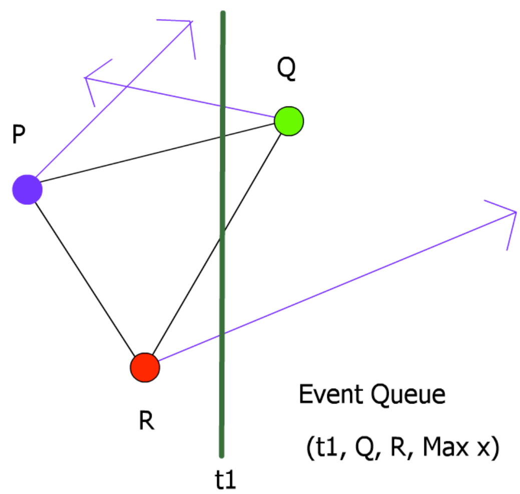
Our Approach

- Observation:
  - Motion in the physical world is normally continuous
  - Changes in the combinatorial structure of the BVHs occur only at discrete time points

→We store only the combinatorial structure of the BVH and use an event-based approach for updating (kinetization)



Kinetic Toy Example



Max
x Q
y Q

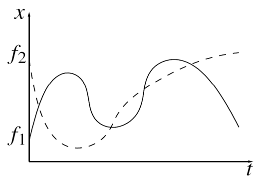
Min
x P
y R

Event Queue

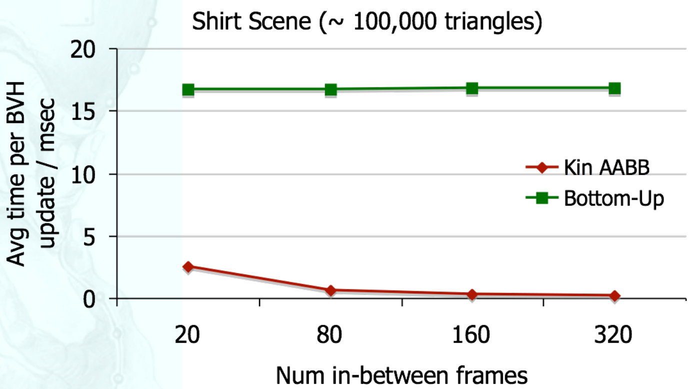
(t1, Q, R, Max x)

Theoretical analysis

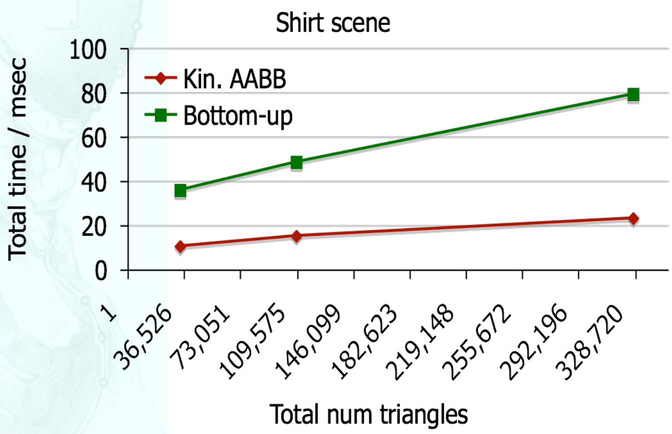
- Theorem:
  Assume the objects have n vertices,
  and the number of intersections of each pair of flightplans is bounded from above by a constant for the duration of the animation.
  Then the total number of events in order to update the BVH is in nearly $O(n \log n)$.
- Remark: this bound is independent of the query frequency.



Results



Total time incl. collision detection time

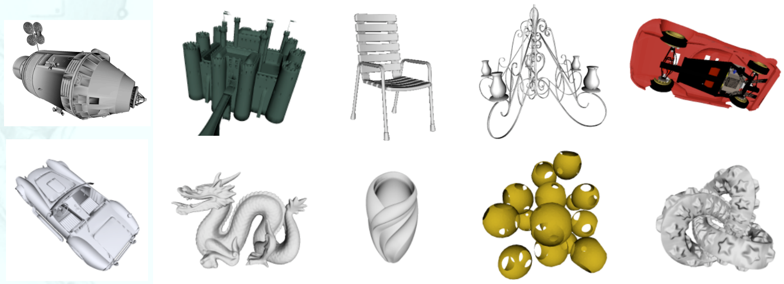


A Benchmarking Suite

- Goal: provide standard benchmark for existing and future collision detection algorithms
- Running time is very sensitive to
  - object shapes,
  - objects complexity,
  - orientation,
  - distance between the objects.

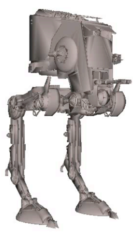
Approach

- Collect substantial set of different objects



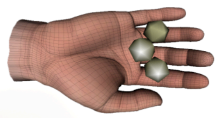
- Compute several LODs for each
- For every pair of objects, and every distance: pre-compute large set of configurations (rotation / translation)
- Downloadable at http://cg.in.tu-clausthal.de/research/colldet_benchmark/

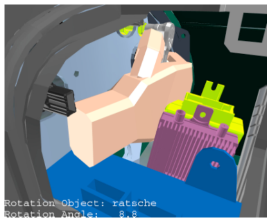
Benchmark @ Work



Natural Interaction

- Direct manipulation is more intuitive and sometimes even more efficient
- Goal:
  - Model and simulate the real human hand
  - Interaction between virtual environment and virtual hand
  - Not necessarily physically correct but physically plausible
- Applications:
  - Virtual assembly Simulation
  - 3D Sketching
  - Medical surgery training

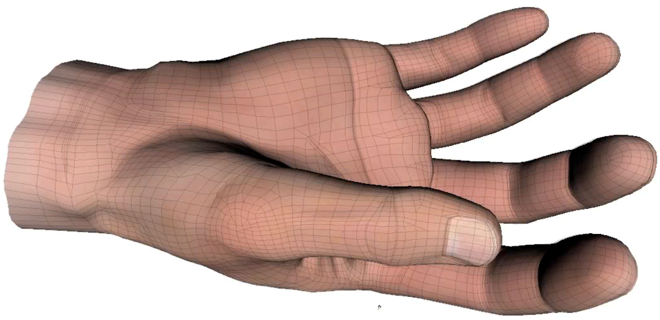




Implementation

- 17k quad mesh hand model
- Skeletal representation
- OpenSG for visualization
- Data Collection with VRJuggler
- Physical simulation by OpenDE
- Spring model for virtual grasping
- Does not rely on heuristics to estimate user intend or grasp state

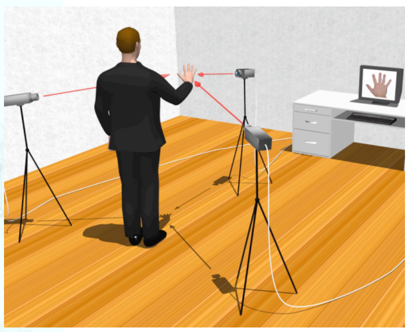
Result (work in progress)



Intro BoxTree GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook

Real-Time Camera-Based 3D Hand Tracking

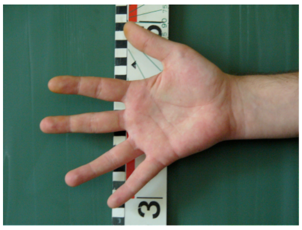
- Goals
  - Observe hand with cameras
  - Determine global hand position and orientation in 3d-space
  - Determine hand state, i.e. angles between fingers



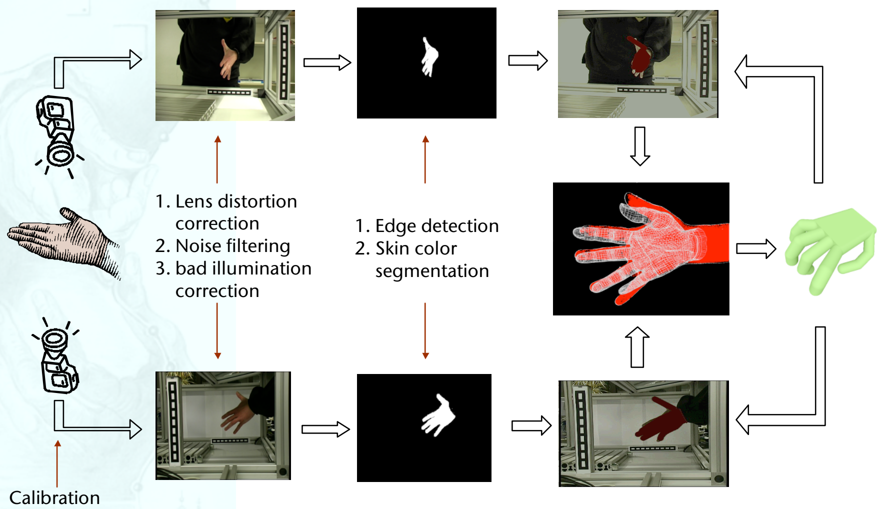
Intro BoxTree GPU Hardware ADB-Trees Point Clouds Kinetization Benchmark Outlook

Challenges

- Measurement noise
- Camera lens distortion
- Uncontrolled illumination
- Mutual occlusions of the hand
- Large working volume
- Fast hand motion
- High problem dimensionality (~ 27 DOFs)



Overview of the System



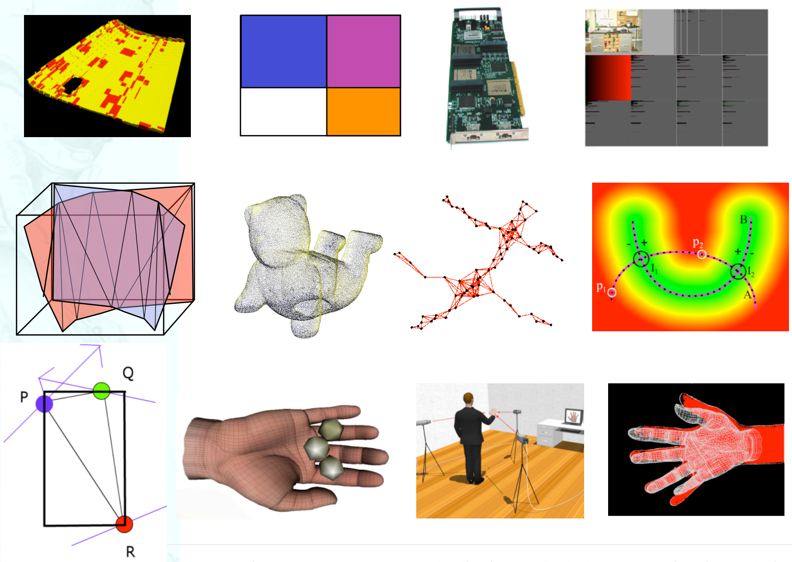
Preliminary Results

	Indoor, neon light, white skin	Indoor, daylight, white skin	Indoor, daylight, white skin	Outdoor, daylight, dark skin

Our method

Jones & Rehg

Summary



References

http://zach.in.tu-clausthal.de/papers/

http://cg.in.tu-clausthal.de/publications.shtml