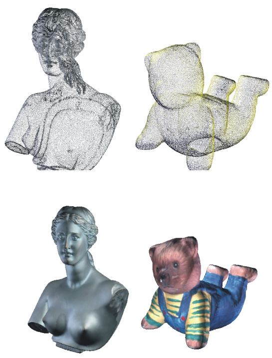
Collision Detection of Point Clouds

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

EG '06, Sep 2006, Vienna, Austria

Motivation

- Renaissance of points as object representation because of 3D scanners



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

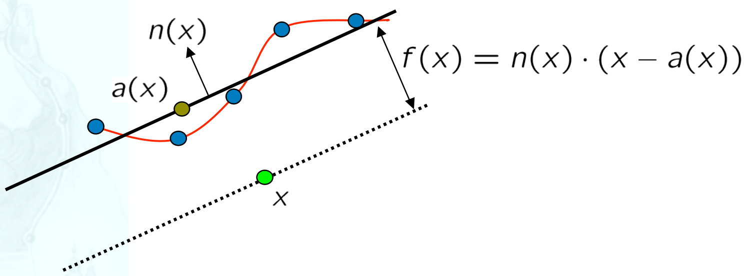
Contents

Definition of the surface

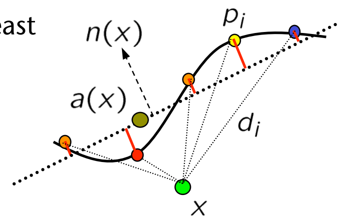
- Idea: assume "distance function" f from surface, then surface S is

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

- "Distance" function f :



- Definition of n by Weighted Least Squares:



$$n(x) = \underset{n,|n|=1}{\arg\min} \sum_{i=1}^{n} \left(n \cdot (a(x) - p^i) \right)^2 \cdot \theta(x - p^i)$$

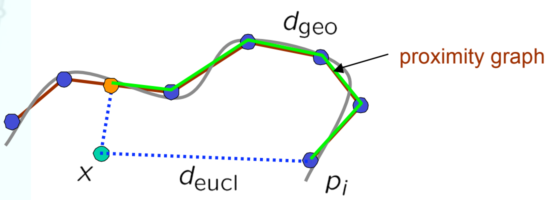
$$C = (P - A)\Theta(P - A)^{\mathsf{T}} \quad , \ (P - A) \in \mathbb{R}^{3 \times n}$$

- Weighting function (kernel):

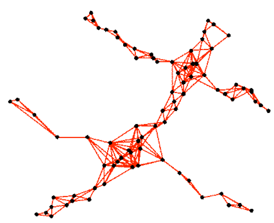
$$\theta(d_i) = e^{-d_i^2/h^2}$$

$$d_i = \text{"distance" between } p_i \text{ and } x$$

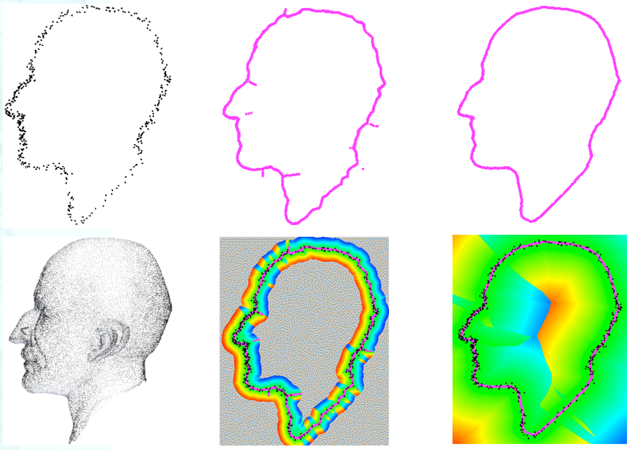
- Cause and solution:



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



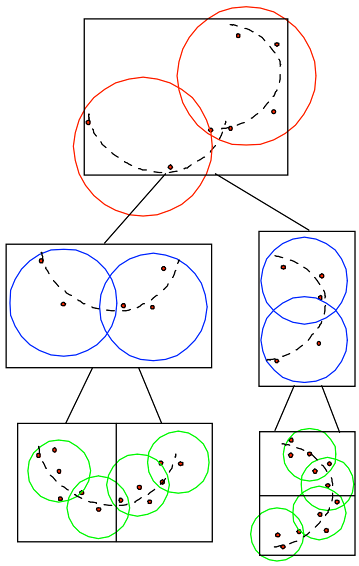
Result



Point Cloud Hierarchy

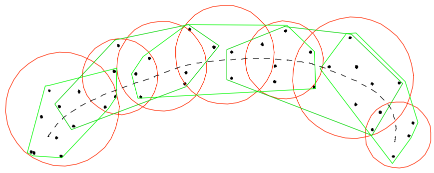
- Build BVH according to some local criterion (e.g., volume of child BVs)
- Construct subsampling and sphere covering at inner nodes

→Efficient storage



Sphere covering

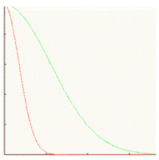
- Observation: surface stays (usually) within set of convex hulls C^i



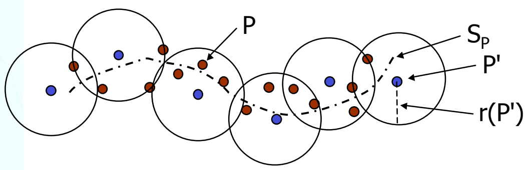
- Randomised technique:
  - Basic operation: construct random point inside $\bigcup C^i$
  - Draw set of samples from orig. point set for sphere centeres
  - Compute common radius like Monte-Carlo integration

Automatic bandwidth detection

- Which bandwith h in kernel $\theta(d_i) = e^{-d_i^2/h^2}$?
  - Too small → noisy surface
  - Too large → no details



- Introduce "Sampling Radius" of a point cloud:
  Given point cloud P in BV A, P' ⊆ P.
  Sampling radius r(P') := smallest radius,
  so that spheres about P' with this radius cover
  surface S_P, defined by P, within A.



- Compute bandwidth h from $r(P)$:

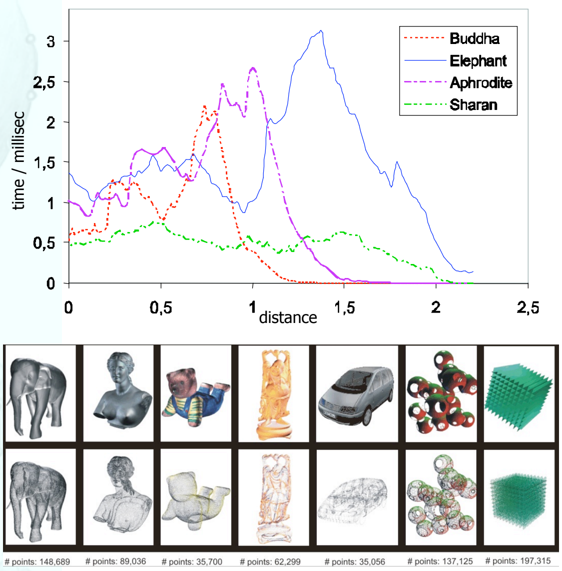
$$h(P) = \frac{\eta\, r(P)}{\sqrt{-\log \theta_\varepsilon}}$$

η = sampling-independent bandwith,
Θ_ε = small threshold (e.g., machine precision)
r(P) = sampling radius of (sub-) point cloud

- Estimate $r(P')$:

$$r(P') \approx \sqrt{|P|/|P'|} \cdot r(P)$$

Result

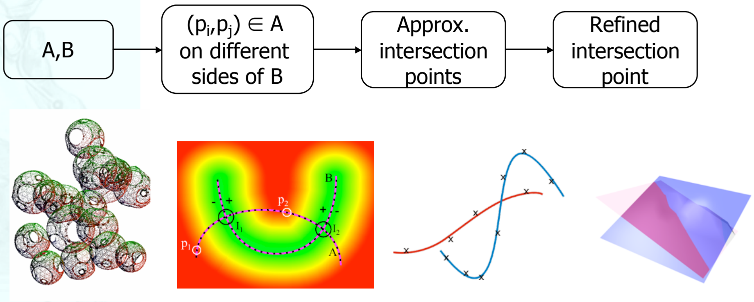


Coll.Det. of PCs using Stochastic Sampling

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

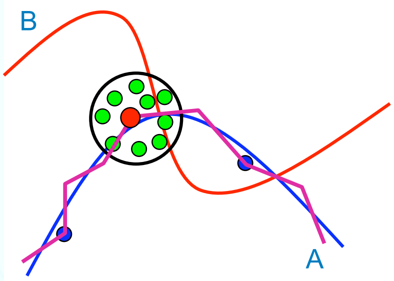
$$\mathcal{Z} = \{x \mid f_A(x) = f_B(x) = 0\}$$

- Overall method:



Algorithm Overview

1. Bracket intersections by pairs of points
2. Find approximate intersection point (AIP) by interpolation search
3. Refine AIP by (randomized) sampling



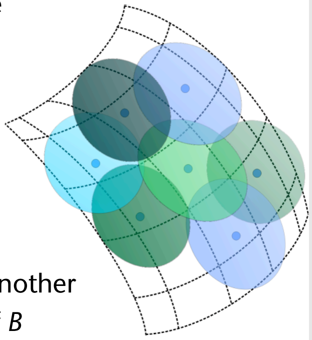
1. Root Bracketing

- Goal:
  - The pairs should evenly sample the surface A
  - The two points should not be too far apart
  - Do it *without* explicit spatial data structure

a) First step: draw number of points (to be one side of the root brackets)

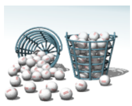
- Thought experiment:
  - Assume surface is covered by p surfels.
  - Draw enough points from PC *A* so that each surfel is hit by at least one point

b) Second step: for each point, try to find another point from *A* lying on the "other" side of *B* (completing the brackets)



a) drawing the points

- Avoid spatial data structure (e.g., grid)
- Pursue probabilistic approach: occupy all *p* surfels with high probability
- Assumption: PC *A* is uniformly sampled



- Lemma [WSCG'05] →

  draw $O(p \ln p + c \cdot p)$ random and independent points from $A \cap \mathrm{Vol}(A \cap B)$

  then each surfel is hit by probability $Pr = e^{-e^{-c}}$
- Depending on app: choose *p* constant or adapt *p* to sampling density

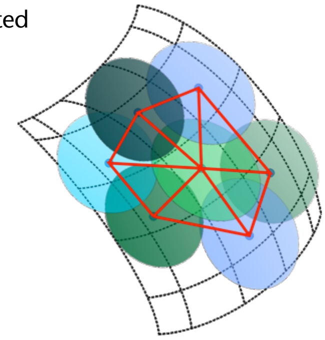
b) completing the brackets

- Use $f_B(p_i) \cdot f_B(p_j) < 0$ as an indicator
- Test only points p_j that
  - belong to the randomly chosen points
  - are close to each other

  → Only very few points p_j need to be tested
- Solution: neighborhood graph (e.g., Sphere-of-Influence graph)
- Complexity of finding brackets:

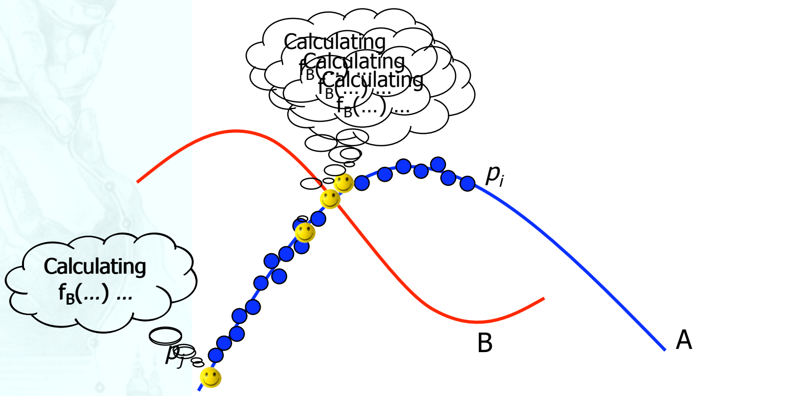
  $O(d \cdot p \log p)$

  where d = max. out-degree



2. Interpolation Search

- Find $\hat{p} \in A$ along shortest path $\overline{p_i p_j}$ in the geometric proximity graph, such that $|f_B(\hat{p})|$ is minimal.
- Utilize interpolation search → $O(\log \log m)$, m = # pts on path

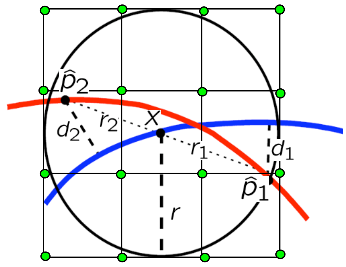


3. Precise Intersection Points

- Intercept Theorem (assuming surface is not curved):

$$\frac{d_1}{d_1 + d_2} = \frac{r_1}{||\hat{p}_1 - \hat{p}_2||}$$

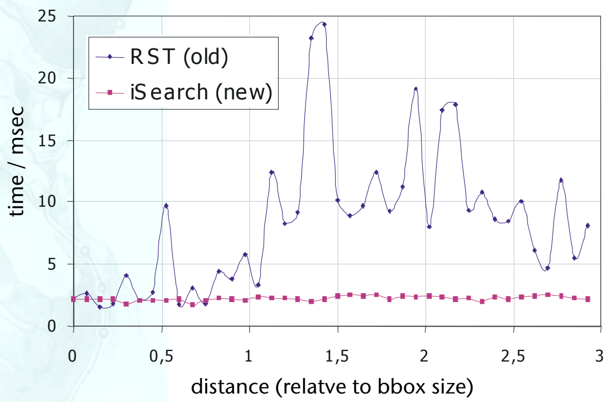
$$x = \frac{1}{d_1 + d_2}(d_2 \cdot \hat{p}_1 + d_1 \cdot \hat{p}_2)$$

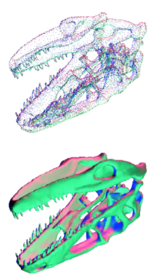


- True intersection point is in sphere S(x,r), with
  $r = \max(r_1, r_2)$
- Sample sphere by points on regular grid

Results

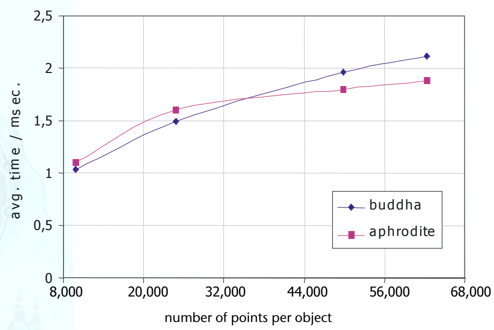
- Benchmarking old vs. new method





28,000 points

- Theoretical complexity: $O(\log \log N)$
  - Assumptions: $f(x)$ monotone along paths $\overline{p_i p_j}$; and, evenly sampled point cloud.
- Experimental complexity:



Conclusion of Stochastic Point Cloud CD

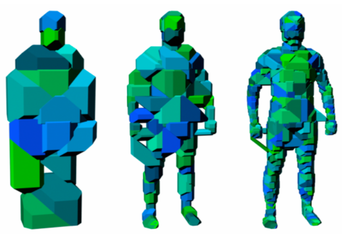
- Technique:
  - utilizes a proximity graph for collision detection and surface definition
  - needs no BV hierarchies and no spatial partitioning data structure
  - any BV hierarchy can be augmented by this technique to increase performance
- Runtime:
  - fast (approximate) collision detection
  - overall runtime: $O(\log \log N)$ in average case
- Quality/resolution of output (intersection points) can be adjusted (→ "surfel" radius)

Conclusions on Stochastic Approach

- Cannot be proven error-free
- Good for plausible & fast simulations
- Interesting alternative to BVHs in specific cases
- Can often be combined with BVHs
- Naturally yield time-critical collision detection

Kinetic AABB Trees

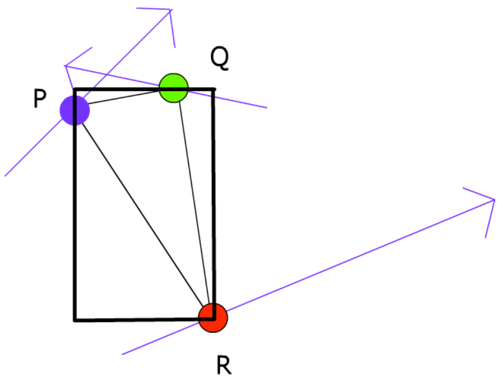
- Not just for collision detection (ray-tracing, occlusion culling, collision detection)



Courtesy GRIS, Tübingen

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

Brute Force Updates



Max
x 1.0
y 0.9

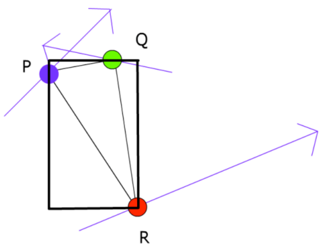
Min
x 0.3
y 0.4

Frame 2

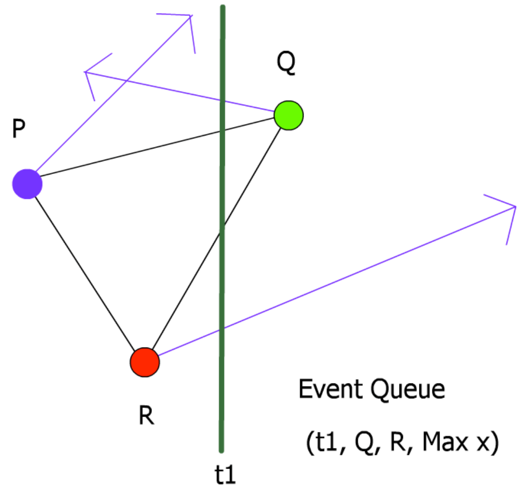
Our Approach

- 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 Updates



Max
x Q
y Q

Min
x P
y R

Event Queue

(t1, Q, R, Max x)

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
- With every node, store the indices of those points that determine the BV
- Initialize the event queue

Animation Loop

while animation runs

    determine time t of next rendering

    e ← min event in event queue

    while e.timestamp < t

        processEvent(e)

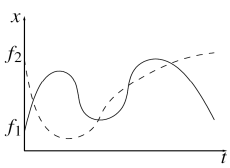
        e ← min event in event queue

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

    render scene

Advantages

- Fewer update operations
- Valid BVHs at every point in time
- Independent of query sampling frequency
- Can handle all kinds of object representations
  - 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 animation

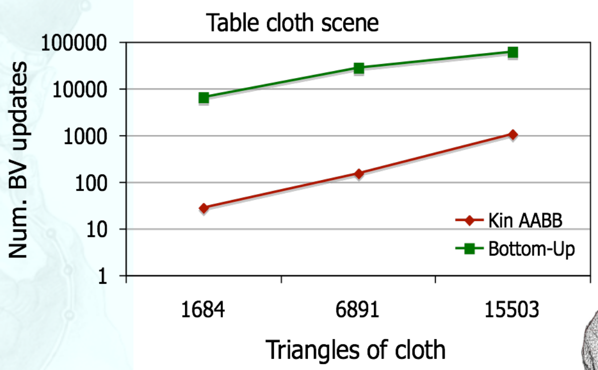


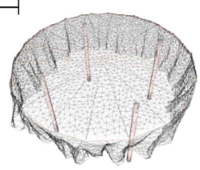
Analysis

- Theorem 1:
  Let n = number of vertices.
  The kinetic AABB tree requires only O(n) space. Each vertex participates in only O(log n) events. The kinetic AABB tree can be updated in only O(log n) time when an event occurs. Finally, it is a valid BVH at every point of time.
- Theorem 2:
  Given n vertices, we assume that the number of intersections for each pair of vertex flightplans is bounded by a constant. Then, the total number of events is in O(n log n), i.e., it is independent of the number of queries (within the given flightplan duration).

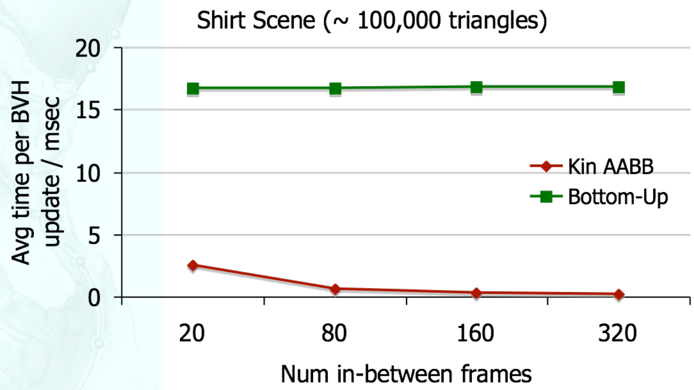
Results

- Total number of BV updates for complete animation sequence (note logarithmic scale):

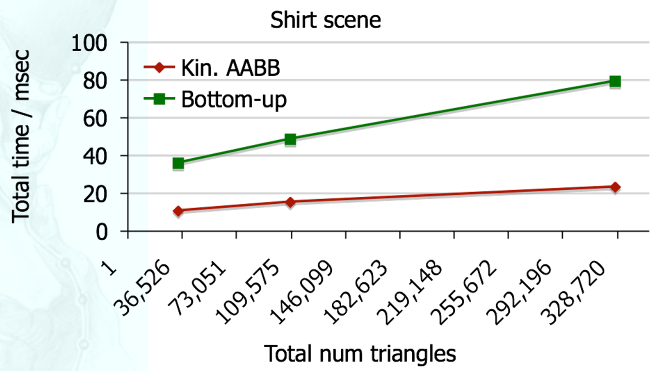




- Average update time per frame:



- Total time incl. collision detection time:

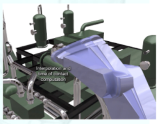


References

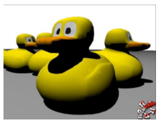
- Jan Klein and Gabriel Zachmann, “ADB-Trees: Controlling the Error of Time-Critical Collision Detection”, *Proc. VMV ’03*
- Jan Klein and Gabriel Zachmann, “Time Critical Collision Detection Using an Average Case Approach”, *Proc. VRST ’03*
- M.C. Lin and J.F. Canny “Efficient Collision Detection for Animation”, *Eurographics Workshop on Animation and Simulation ’92*
- Stephane Guy and Gilles Debunne, “Monte Carlo Collision Detection”, *INRIA Technical Report RR-5136*, 2004
- Laks Raghupathi et al. “Real-time Collisions and Self-Collisions for Virtual Intestinal Surgery”, *Surgical Simulation and Soft Tissue Modeling*, pp.38-46, Springer, 2003
- Laks Raghupathi et al. “An Intestinal Surgery Simulator: Real-Time Collision Processing and Visualization”, *IEEE TVCG*, Vol. 10, No. 6,
- Stefan Kimmerle, Matthieu Nesme and Francois Faure, “Hierarchy Accelerated Stochastic Collision Detection”, *Proc. VMV ‘04*
- Jan Klein and Gabriel Zachmann: "The Expected Running Time of Hierarchical Collision Detection", SIGGRAPH 2005, Poster
- Jan Klein and Gabriel Zachmann: "Interpolation Search for Point Cloud Intersection", Proc. of WSCG 2005

- Jan Klein and Gabriel Zachmann: "Nice and Fast Implicit Surfaces over Noisy Point Clouds", SIGGRAPH 2004, Sketches and Applications
- Adams & Dutre: "Interactive boolean operations on surfel bounded solids", 2003
- Hubbard: "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", 1996
- Dingliana, O‘Sullivan: "Graceful Degradation of Collision Handling in Physically Based Animation", 2000
- Adamson & Alexa: "Approximating and Intersecting Surfaces from Points", 2003
- Jan Klein and Gabriel Zachmann: "Point Cloud Collision Detection", Proc. EUROGRAPHICS 2004
- Otaduy & Lin: "Sensation Preserving Simplification for Haptic Rendering", 2003
- Otaduy & Lin:"CLODs: Dual Hierarchies for Multiresolution Collision Detection", Symp. on Geometry Processing (2003)
- Weller, Zachmann: "Kinetic Bounding Volume Hierarchies for Deformable Objects", VRCIA (2006)

Wrap-Up



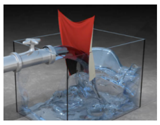
Rigid bodies
Stephane Redon



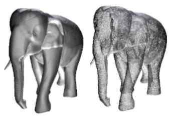
Deformable objects
Matthias Teschner

Cloth
Pascal Volino

Hair
Marie-Paule Cani



Fluids
Robert Bridson



Point Clouds
Gabriel Zachmann

The End