Data Structures for Computer Graphics

# Static Collision Detection

Slides courtesy of Ladislav Kavan

Lectured by Vlastimil Havran

# Problem Description

Input: two (or more) 3D objects

Task: find the intersections

3D objects A, B given by triangular meshes

- triangle soup (no topology)
- find all pairs of intersecting triangles (*collisions*)

Assume: object A has $m$ triangles, B has $n$ triangles

- worst case: $m \cdot n$ colliding triangle pairs
  - e.g. when A = B then formally $O(N^2)$ complexity
- no algorithm can be better than quadratic in the worst case

# Applications of Collision Detection

- Robotics: motion planning for robot without collision
- Animation and simulation systems including game industry
  - Scene consistency test (at frame rate up to 1kHz)
  - Interactive physically based modeling (action-reaction)
- CAD, mechanical engineering
  - Parts assembly test (car industry etc.)
- Chemistry: molecular modeling, fitting sequences of molecules into tunnels

# Collision Detection (CD)

Simple quadratic algorithm:

for each triangle $t_A \in A$

  for each triangle $t_B \in B$

    if ($t_A$ intersects $t_B$) report ($t_A$, $t_B$)

- optimal in the worst-case
- not useful in practice
  - common models $\approx$ thousands triangles
  - CD query must be fast ... 25+ FPS for animation
  - even 2000 FPS needed for haptic devices!

Fortunately, the number of collisions is typically smaller than $m \cdot n$

# Output-Sensitive Algorithms

An algorithm is *output-sensitive* if its execution time depends on the size of the result

- important speedup of CD
  - typical situations: only few collisions
- e.g. objects far away - very fast answer
- sometimes only yes/no result
  - non-empty/empty set of colliding triangles

*Static* CD: objects A, B fixed

- disregards motion (but not rigid body transformations)

Note: dynamic (continuous) CD considers motion

# Bounding Volumes

Idea: quickly discard distant triangles

Example: enclose objects A, B by spheres $S_A$, $S_B$

- if ($S_A$, $S_B$ disjoint) no collisions
- sphere-sphere intersection test: very fast
- if ($S_A$, $S_B$ intersecting)
  - A, B may be colliding
  - A, B may be disjoint (false-positive)

Solution: build a bounding volumes hierarchy (BVH)

- most popular, but not the only possibility:
  - space partitioning
  - Voronoi diagrams

# Bounding Volume Hierarchy

*bounding volume* - geometrically simple object enclosing the original geometry

*hierarchy* - a tree with different properties

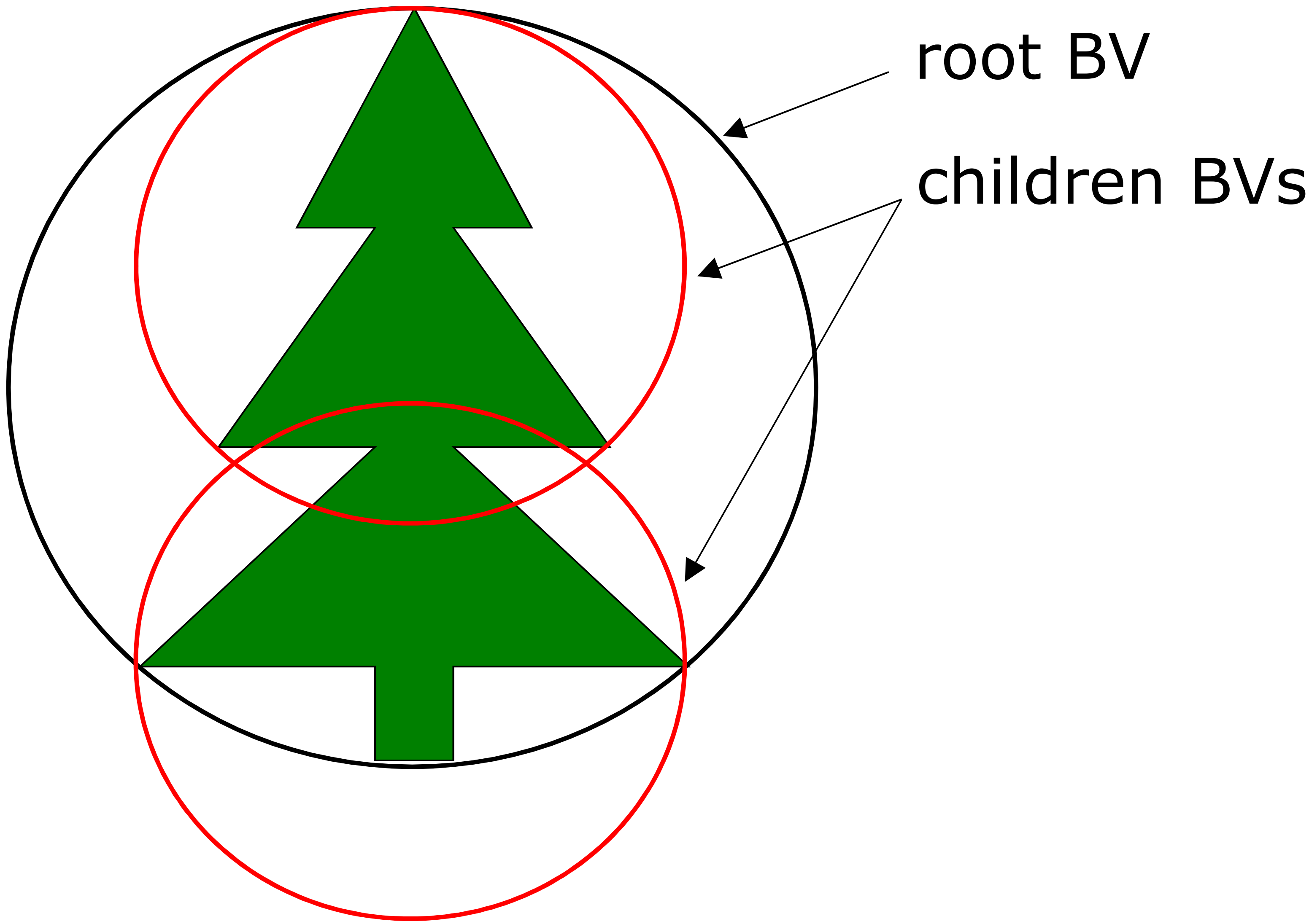
*BVH* - a tree with BVs in the nodes

- the BVs of the children enclose the same geometry as the parent
  - the BV of the root encloses the whole model

Bounding volume

- quick collision test
- tight bounding (approximation)

# BVH: Example



# CD Based on a BVH of objects A and B

Input: roots $r_A$, $r_B$ of BVHs of two objects

Output: all colliding triangles between A and B

CDTEST($r_A$, $r_B$)

1. if (BV($r_A$), BV($r_B$) disjoint) return empty set
2. if ($r_A$ leaf && $r_B$ leaf) test all triangles of $r_A$ against all triangles of $r_B$ and return colliding ones
3. if ($r_A$ leaf && $r_B$ not leaf) return union of CDTEST(x, $r_A$) for each child x of $r_B$
4. if ($r_B$ leaf && $r_A$ not leaf) return union of CDTEST(x, $r_B$) for each child x of $r_A$
5. $r_X$ = the node **with larger BV**; $r_Y$ = the other one
6. return union of CDTEST(x, $r_Y$) for each child x of $r_X$

# CD Based on a BVH: Yes/No Query

- terminate after first collision found
- it is faster

CDTEST2($r_A$, $r_B$)

1. if (BV($r_A$), BV($r_B$) disjoint) return NO
2. if ($r_A$ leaf && $r_B$ leaf) test all triangles of $r_A$ against all triangles of $r_B$ and return YES/NO
3. if ($r_A$ leaf && $r_B$ not leaf) return YES for the first child x of $r_B$ giving CDTEST2(x, $r_A$) = YES; NO if none
4. if ($r_B$ leaf && $r_A$ not leaf) return YES for the first child x of $r_A$ giving CDTEST2(x, $r_B$) = YES; NO if none
5. $r_X$ = the node **with larger BV**; $r_Y$ = the other one
6. return YES for the first child x of $r_X$ giving CDTEST2(x, $r_Y$) = YES; NO if none

# Choice of Bounding Volumes

Trade-off between

- fast intersection test of two BVs
- tight bounding
- Cost model again:

$$\text{Total time} = N_V \cdot C_V + N_P \cdot C_P + N_U \cdot C_U + C_B$$

- $N_V$ ... number of tested BV pairs
- $C_V$ ... cost of BV intersection test
- $N_P$ ... number of tested primitive (triangle) pairs
- $C_P$ ... cost of primitive (triangle) intersection test
- $N_U$ ... number of updated BV nodes
- $C_U$ ... cost of updating BV nodes
- $C_B$ ... cost of initial building data BVH

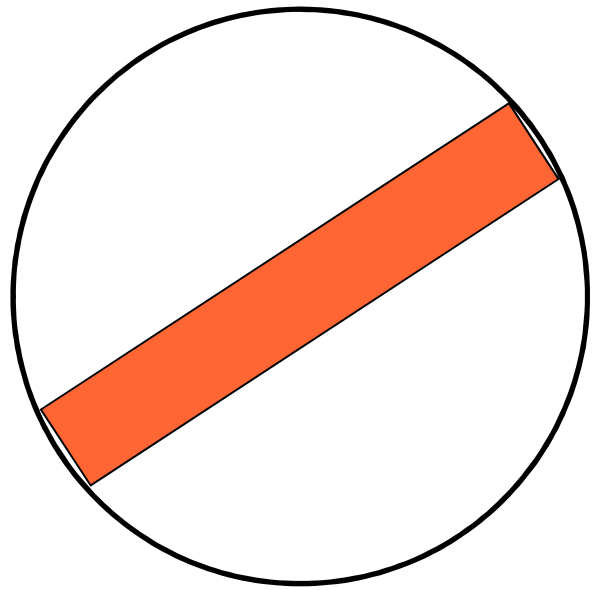
# Bounding Sphere

- given by a center $\mathbf{c} \in A^3$ and $r \in R$

Very fast intersection test of spheres A, B:

if $(\langle \mathbf{c_A}-\mathbf{c_B}, \mathbf{c_A}-\mathbf{c_B} \rangle > r_A^2+r_B^2)$ disjoint

else intersecting



- bad bounding tightness
- simple to update
  - rotation invariant
  - sufficient to translate the center
- computation:
  - simple approximation
  - smallest enclosing sphere: randomized algorithm (Bernd Gaertner, Emo Welzl), O(N) complexity. Exact algorithm relatively slow, approximate algorithm fast.

# Axis-Aligned Bounding Box (AABB)

- a box with faces aligned with the world coordinate system
- other view: 3D interval
- $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$

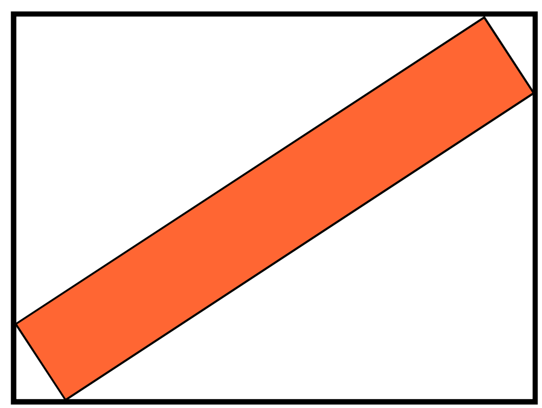
Intersection test:

AABBs disjoint iff all intervals are disjoint

Intersection of intervals [a,b] and [c,d]:

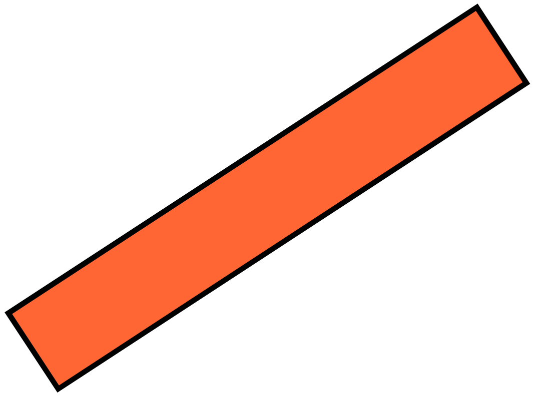
if (a<c) return (c<b)

else return (d>a)



- slightly better bounding
- computation: simple

# Oriented Bounding Box (OBB)



- arbitrary (non-aligned) box
- given by a frame & intervals
- good bounding tightness

Computation (approximate):

- construct convex hull of vertices
- compute mean (center of frame)
- covariance matrix M
- eigenvectors of M form a good OBB basis
- Details can be found in the thesis of Gotschalk, ***Collision Queries using Oriented Bounding Boxes,*** 2000, available at:

http://www.mechcore.net/files/docs/alg/gottschalk00collision.pdf

# Intersection Test of OBBs

Idea: search for a *separating axis*

Choose an axis (direction vector) and project OBBs to this axis

- if (projected intervals disjoint) OBBs disjoint
- else OBBs may or may not be disjoint

Separating Axis Theorem (SAT): For OBB-OBB intersection it is sufficient to test following 15 axes

- the normals of faces (3+3)
- the cross products of edges (3x3)
- if none of the above axes separates, then the OBBs are disjoint

# Discrete Orientation Polytope (DOP)

given a fixed set of k/2 directions (→ k-DOP, k even)

- unit vectors $\mathbf{d_1}, \ldots, \mathbf{d_{k/2}}$

k-DOP: a polytope with face normals $\mathbf{d_1}, \ldots, \mathbf{d_{k/2}}$, $\mathbf{-d_1}, \ldots, \mathbf{-d_{k/2}}$

represented by k/2 intervals $[l_1, h_1], \ldots, [l_{k/2}, h_{k/2}]$

DOP Construction (exact):

for i = 1 to k/2 do

{

$l_i = \min \langle \mathbf{v}, \mathbf{d_i} \rangle$ for each vertex $\mathbf{v}$

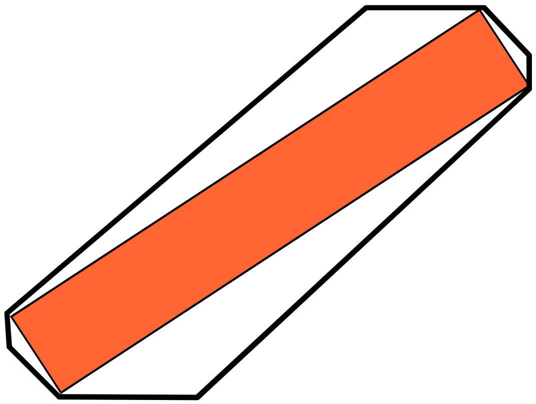
$h_i = \max \langle \mathbf{v}, \mathbf{d_i} \rangle$ for each vertex $\mathbf{v}$

}

# Common k-DOPs

- k=6: AABBs (6-DOP is exactly AABB)
  - directions (1,0,0), (0,1,0), (0,0,1)
- k=14: cut corners
  - add directions (1,1,1), (1,-1,1), (-1,1,1), (1,1,-1) (normalized)
- k=18: cut edges
  - add directions (1,1,0), (1,-1,0), (1,0,1), (1,0,-1), (0,1,1), (0,1,-1) (normalized)
- k=26: cut corners & edges

Example (in 2D): 8-DOP



# Intersection Test of DOPs

Conservative test of two k-DOPs A and B:

for i = 1 to k/2 do

if (intervals $[l_i^A, h_i^A]$ and $[l_i^B, h_i^B]$ disjoint)

return DISJOINT

return POTENTIALLY_INTERSECTING

What may happen:

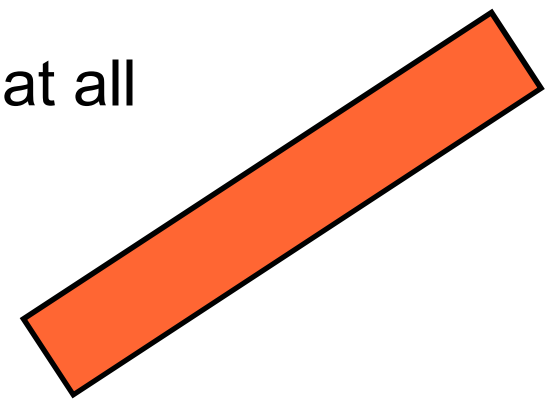
- all intervals intersecting & DOPs disjoint
  - treat them as intersecting (proceed to children)
  - does not violate the correctness of the CD algorithm
  - conservative test

# Convex Hull (CH)

- convex hull is the optimal convex BV in terms of tightness (recall the definition)
  - typically only convex BVs used: convex sets can be always separated by a plane
- computation: easy in 2D, more difficult in 3D

Intersection test: slow

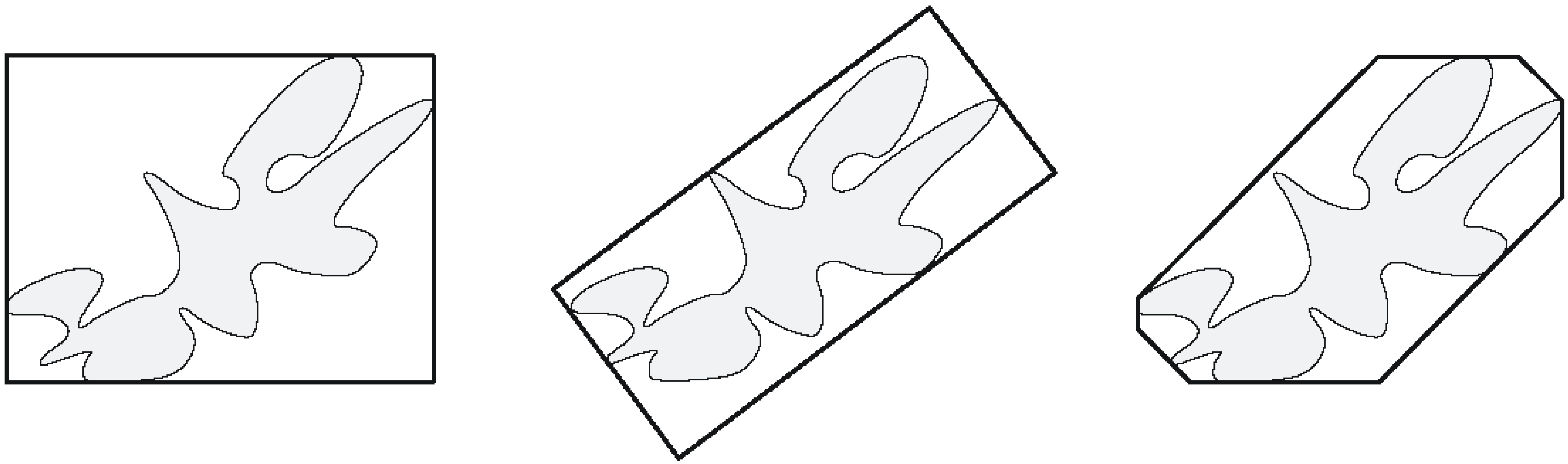
- CH may not simplify the geometry at all



k-DOP: approximation of CH

(better for higher k)

# Comparison of BVs



AABB OBB 8-DOP

- better tightness → more expensive intersection test
- good compromise necessary

# Building the BVH

Two basic approaches: bottom-up & top-down

Bottom-up (merging) construction:

- create BVs & (single-node) trees for individual triangles
- pick several neighboring trees $\mathbf{n}_1$, ..., $\mathbf{n}_m$ and create a common parent **p**
  - m is the order of the tree
  - the BV(**p**) must enclose all the triangles enclosed by nodes $\mathbf{n}_1$, ..., $\mathbf{n}_m$ (needs not enclose BV($\mathbf{n}_1$), ..., BV($\mathbf{n}_m$))
- repeat until single tree remains (the result)

Tricky bit: "pick several neighboring trees"

# Top-down BVH construction

BuildTree(T)

- create a node **n** and BV enclosing the whole mesh T
- split the geometry T into m parts: $M_1$, ..., $M_m$
- if (m==1) return **n**   // no further splitting possible
- for i = 1 to m do
  i-th child of **n** = BuildTree($M_i$)
- return **n**

Tricky bit: splitting rule

- simple but efficient heuristic: build an AABB
- split in the middle of the longest side

# Update of a BVH

- consider a rigid-body motion of the object

Translation

- no problem for any BV

Rotation

- simple for spheres & OBBs
- k-DOPs (& AABBs):
  - re-compute intervals for rotated vertices ... slow
  - compute new k-DOP of rotated original k-DOP (or convex hull) ... sub-optimal tightness
  - more sophisticated methods exist

# Literature

- Gino van den Bergen: Collision Detection in interactive 3D environments, 2004
- Christer Ericson: Real Collision Detection, Morgan Kaufmann 2005
- Additional reading text on course webpage: F. Madera: An introduction to the Collision Detection Algorithms, 2011.
- Lukáš Korba: Simulace řízení vozidla, diplomová práce 2008, ČVUT FEL.