Course on Virtual Reality

Collision Detection

Overview

- History
- Basics
- Bounding Volumes
- Space Partitioning

A Brief History

- 70's: Applications of 3D collision detection in robotics and automation
  Continuous time axis!
- 80's: Computational Geometry as a new research area
  Geometric algorithms from a mathematical point of view
- 1984: ELITE – First 3D video game (space combat)
  Polygonal spaceships
  CD based on boxes and spheres only
- 1989: From keyframe animation to PBM (D. Baraff)
  Necessary to resolve collisions automatically
- 90's: From animation to interactive applications
- 1995: I-COLLIDE – First CD library
- 1996: PC game „Quake" – BSP trees
- 1996: Spheres as bounding volumes (e.g., Hubbard)
- 1996: Oriented boxes as bounding volumes (M.C. Lin)
- 1997: AABB trees as bounding volumes (van den Bergen)
- 1998: Discrete Oriented Polytopes (Klosowski)

CD versus CG

Similar algorithms and spatial data structures in CG and CD!

Ray Tracing – Object selection via Ray Casting
Voxel grids, octrees, binary space-partitioning trees, …

Sources of Collisions

- User interaction, e.g., selection of objects
- Dynamic virtual environments
  - Movement of rigid objects
    - Translation
    - Rotation
    - ➢ Collision between pairs of objects
  - Deformation of objects
    - Scaling
    - Shearing
    - Complex deformations (e.g., FEM, see later in this course)
    - ➢ Collision between pairs of objects
    - ➢ Self-collisions

Collision Detection: Real versus Virtual Worlds

- **Real world:**
  - Two material objects cannot occupy the same point in space at the same time
  - Collision: two objects hit each other
    (Contact: an object stays on other object surface for some time)
  - Collision response: Physical behaviour

- **Virtual world:**
  - Two geometric objects can occupy the same point in space at the same time
  - Collisions: configurations of interpenetrating objects
    - Collision queries: detect collisions at a discrete time step
    - Compute response data (e.g., penetration depth)
  - Collision response: real-time simulation of physical behaviour

Collision Detection Challenges

- Correctness
  - Discrete time in digital computers
  - Objects are just a geometrical approximation

- **Real-Time / Performance**
  - Complex scenes (many objects, detailed shapes) versus limited processing power
  - Limited amount of memory

- Precision / Stability
  - Floating-point arithmetics: rounding errors
  - Incorrect answers to collision queries

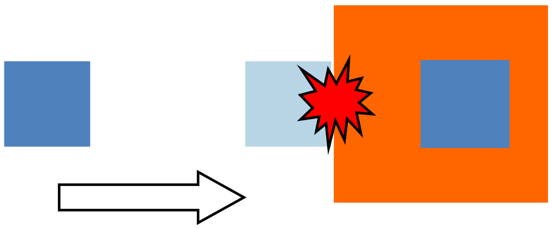
Interference and Collision Detection

- *Interference* detection or *collision query*
  - Static setting
  - Detect if an object is inside other object (penetration depth?)

- *Collision* detection
  - Dynamic setting
  - Detect if object trajectory intersects another object (when?, where?)
  - Most collision detection algorithms consist of repeated application of collision queries
  - Can miss interaction of fast moving objects

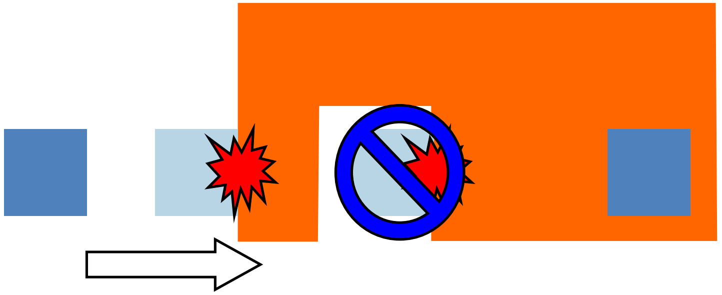
Collision Detection Sampling

- The *exact time and location* of first contact may need to be found.



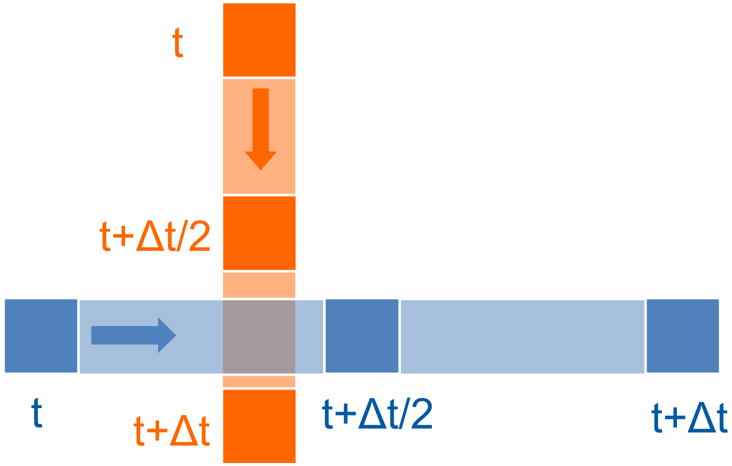
- Sampling at discrete intervals may miss a collision entirely:
  In-between collisions

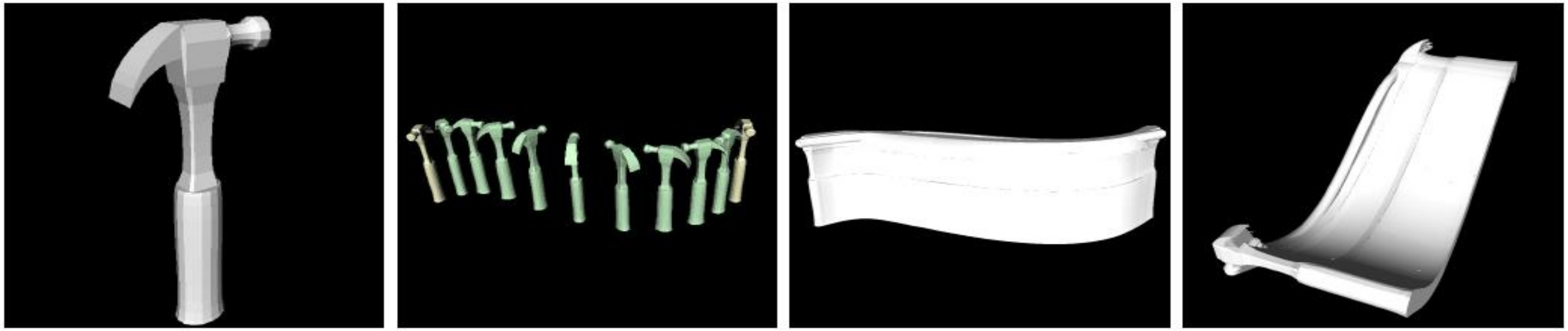
- Sampling at discrete intervals may give the wrong collision.



Swept volumes

- Swept volumes are a *sufficient but not necessary* condition for determining if objects are collision-free
  - Swept volumes may overlap, even when the objects have not collided
  - Subdivision is needed

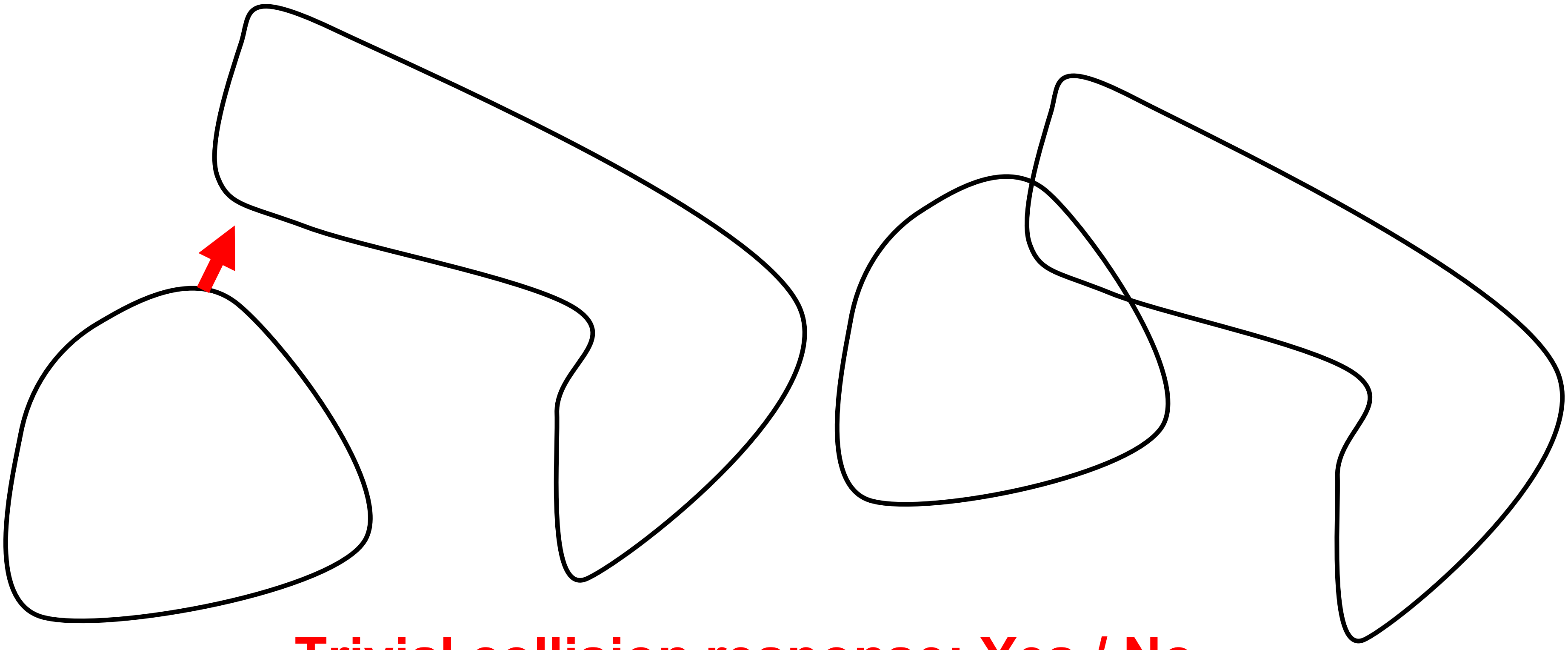




Interference Test

t_{i-1}: no collision

t_i: collision

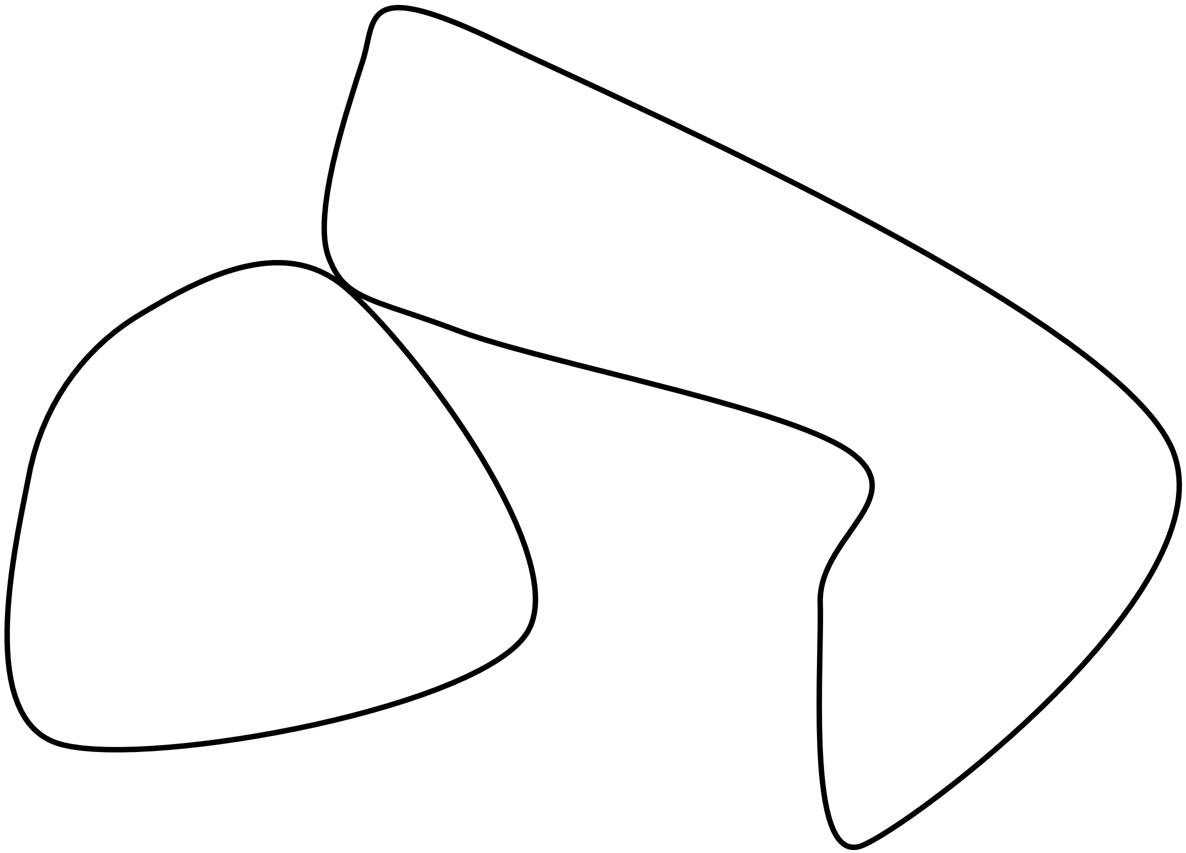


Trivial collision response: Yes / No

Collision Response

- **Ego Shooters:** Interference ok (e.g., object disappears)
- **Object selection:** Interference ok (e.g., object changes colour)

Contact!?



Collision Response

- **Ego Shooters:** Interference ok (e.g., object disappears)
- **Object selection:** Interference ok (e.g., object changes colour)
- **Physically-based virtual environments:**
  Compute reaction forces and impulses that resolve the collision
  - When? ***Time of collision***
  - Where? ***Contact point***
    Point on both object surfaces where the objects first touch
  - How? ***Contact normal***
    Normal to a plane that
    - passes through the contact point
    - separates the objects (at least near the contact point)

Approximations

- Simply take t_{i-1}
- Simply take t_i
- Repeatedly bisect time interval
  - Interference test at $t_{i-0.5}$
  - Collision: Interference test at $t_{i-0.75}$
  - No collision: Interference test at $t_{i-0.25}$
  - ➢ Computationally expensive
  - ➢ No guarantee to find the earliest collision time

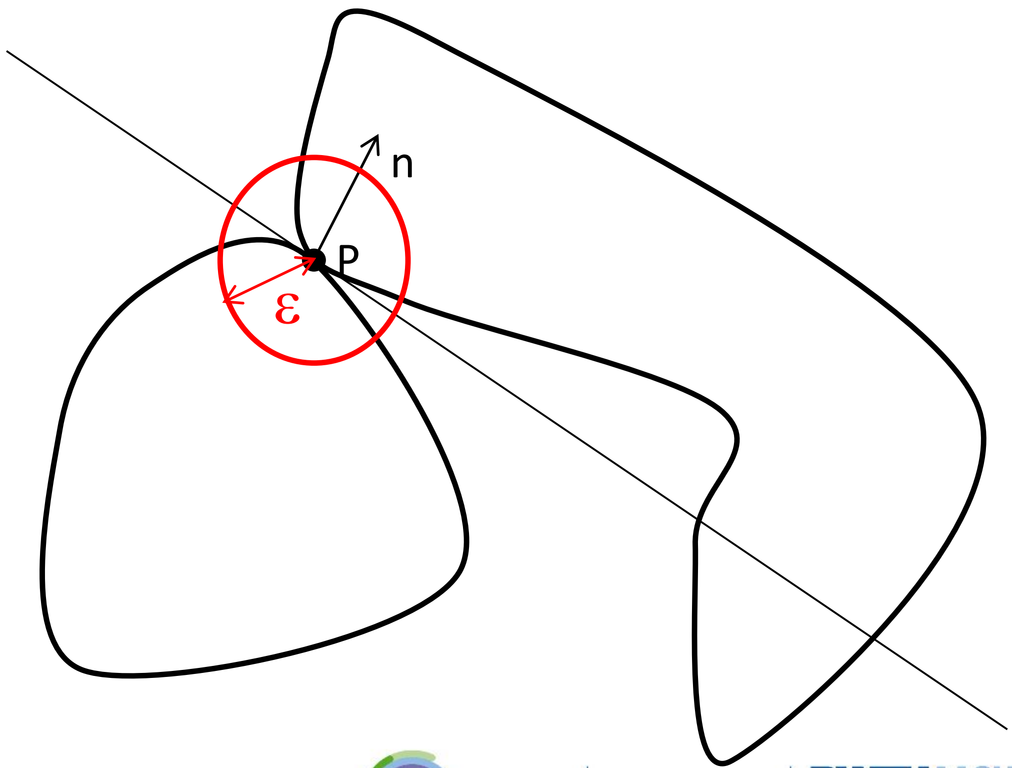
Collision Response

- **Ego Shooters:** Interference ok (e.g., object disappears)
- **Object selection:** Interference ok (e.g., object changes colour)
- **Physically-based virtual environments:**
  Compute reaction forces and impulses that resolve the collision
  - When? ***Time of collision***
  - Where? ***Contact point***
    Point on both object surfaces where the objects first touch
  - How? ***Contact normal***
    Normal to a plane that
    - passes through the contact point
    - separates the objects (at least near the contact point)

ε-Neighborhood

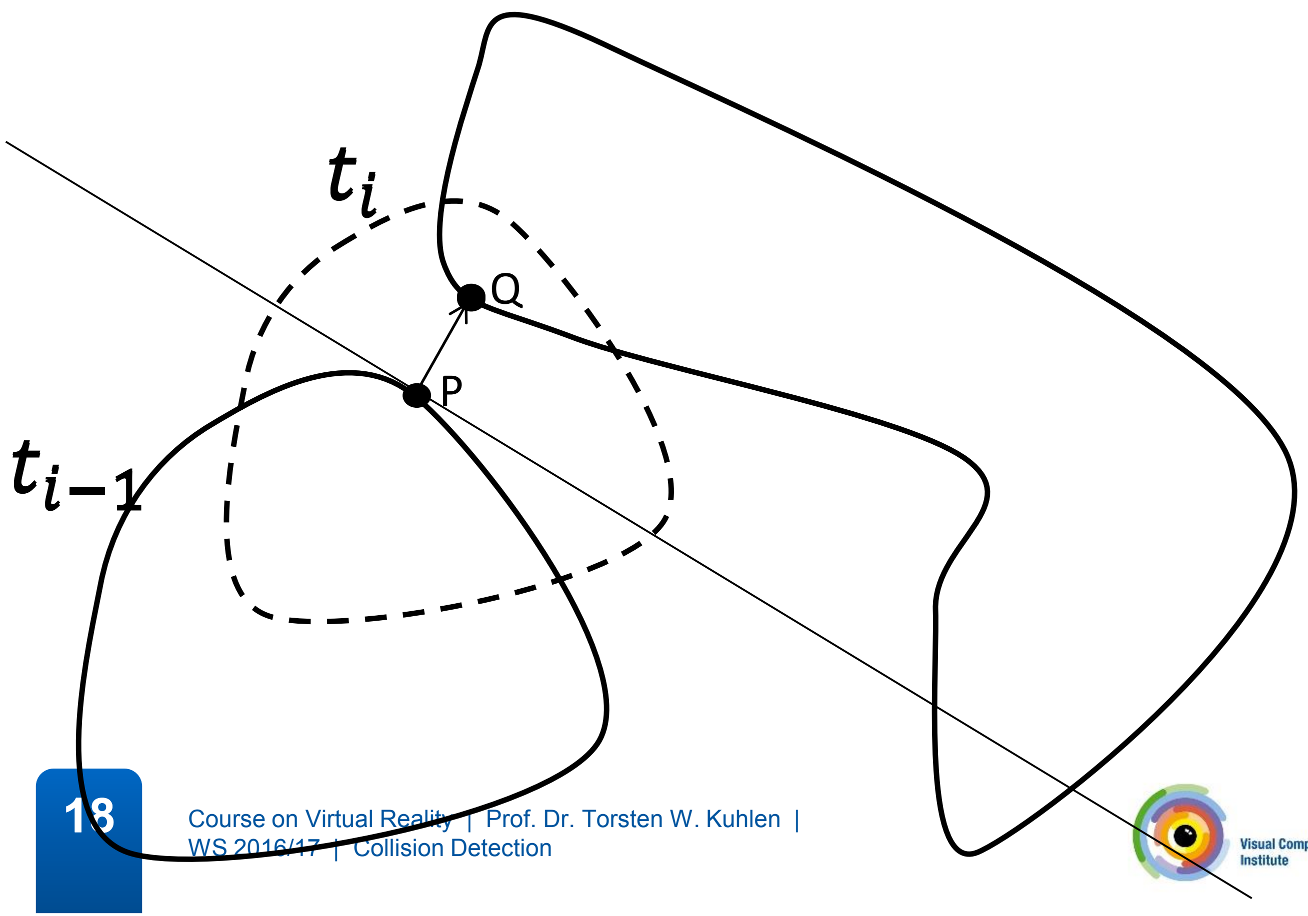
- Small sphere with radius ε and center point P
- Assumption: Objects are convex within the ε-neighborhood

➢ Contact plane always exists



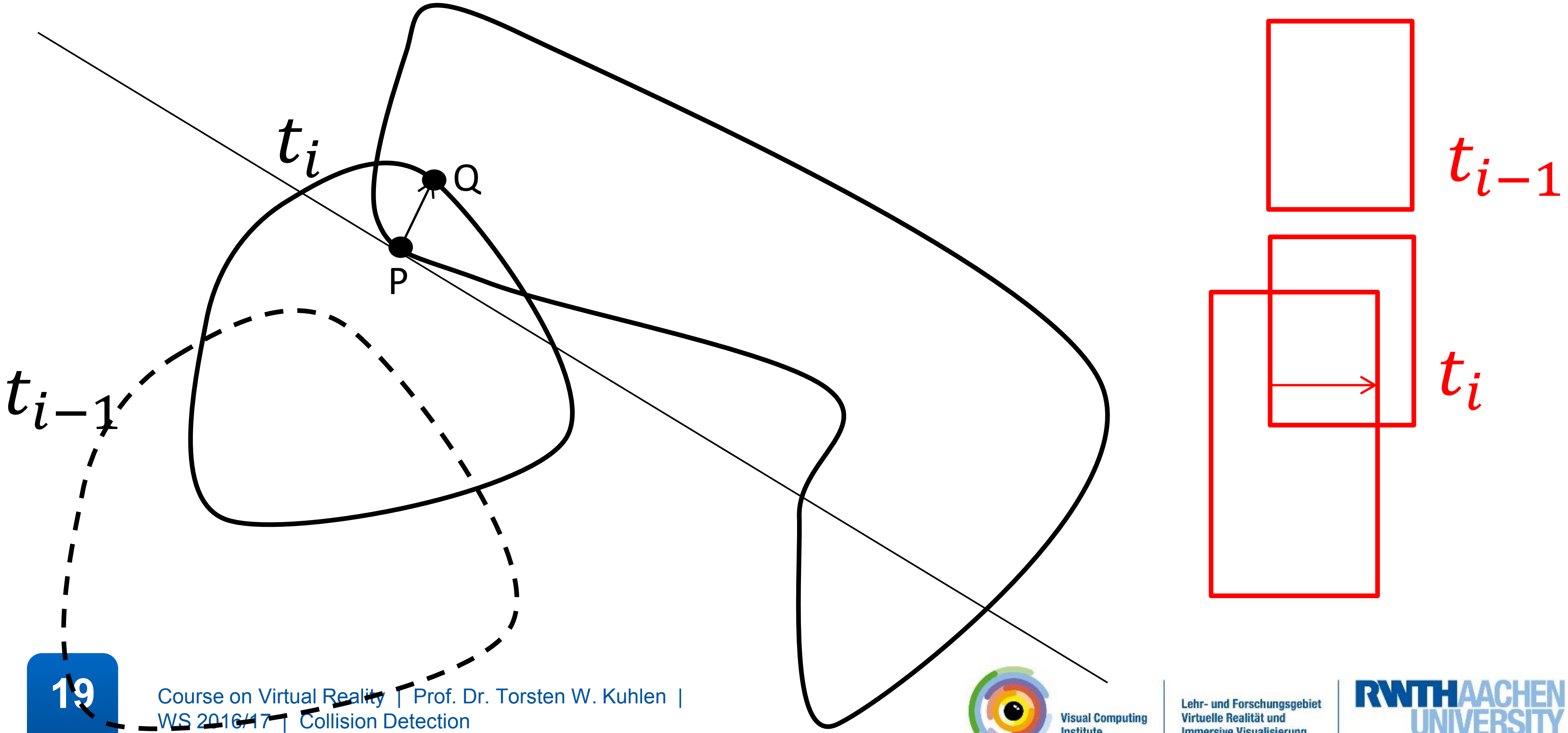
How to Find the Contact Point

- Compute a pair of closest points at t_{i-1}
- Contact normal: difference of the closest points, Q - P



How to Find the Contact Point: Alternative

- Compute penetration depth at t_i
  Penetration depth: length of the shortest vector over which one of the objects needs to be translated in order to bring the pair in touching contact
- Contact normal: Penetration depth vector Q - P

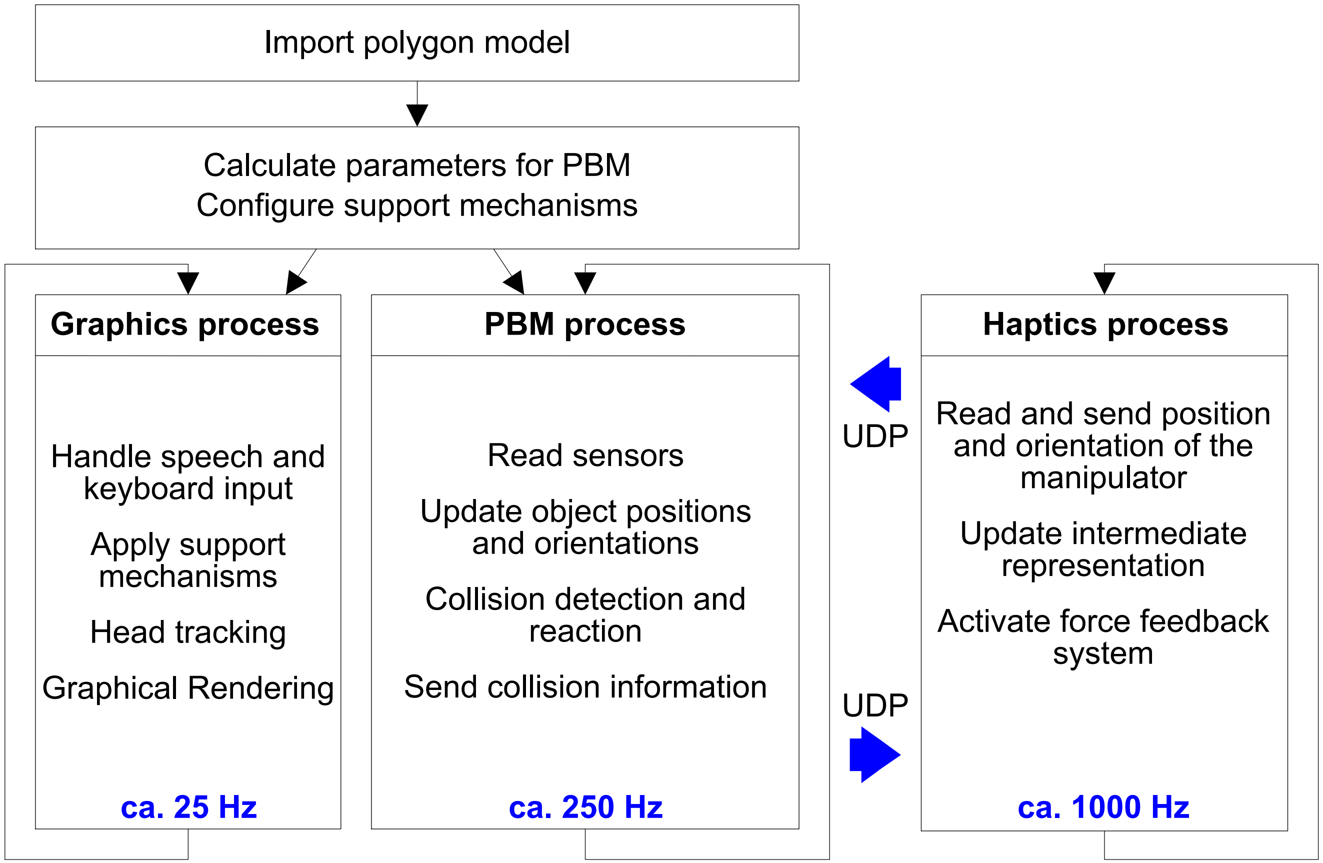


Interference and Collision Detection

- *Interference* detection or *collision query*
  - Static setting
  - Detect if an object is inside other object (penetration depth?)

- *Collision* detection
  - Dynamic setting
  - Detect if object trajectory intersects another object (when?, where?)
  - Most collision detection algorithms consist of repeated application of collision queries
  - Can miss interaction of fast moving objects

Increase Sample Rate



Basic Detection Algorithm

```
Application ()
{
    For t ← t_o to t_1 in steps of Δt_r {
        Get user input
        Update behavior of each object in {O_0,…,O_{N−1}} as of t
        Do {
                result ← detect(t, {O_0,…,O_{N−1}})
                if (result contains collisions)
                        Compute response data /* PBM */
        } while (result contains collisions)
        render each object in {O_0,…,O_{N−1}} as of t
    }
}
```

Basic Detection Algorithm (cont.)

```
/* The variable t_prev persists between calls */
/* Before first call, t_prev is initialized to t_0 */
detect(t_curr, {O_0, ..., O_{N-1}})
{
    For t ← t_prev to t_curr in steps of Δt_d {
        Move each object in {O_0, ..., O_{N-1}} to its position as of t
        For each object O_i ∈ {O_0, ..., O_{N-1}} {
            For each object O_j ∈ {O_i, ..., O_{N-1}} {
                if (pair-processing algorithm says O_i, O_j intersect)
                    add O_i, O_j to collisions
        }
        if (collisions exist) {
            t_prev ← t
            return (collisions, t)
        }
    }
    t_prev ← t_curr
    return (no collisions, t_curr )
}
```

Collision Detection Challenges

- Correctness
  - Discrete time in digital computers
  - Mathematical issue
- **Real-Time / Performance**
  - Complex scenes (many objects, detailed shapes) versus limited processing power
  - Limited amount of memory
- Precision / Stability
  - Floating-point arithmetics: rounding errors
  - Incorrect answers to collision queries

Collision Query: Computational Cost

- Basic equation: $F = N_p \times C_p$

  F ... total cost of interference detection
  N_p ... number of (primitive) pairs tests
  C_p ... cost of (primitive) pair test

- Naïve approach: $N_p = \binom{N}{2} = \frac{1}{2} N(N-1) \approx O(N^2)$

  N: Number of objects in the scene

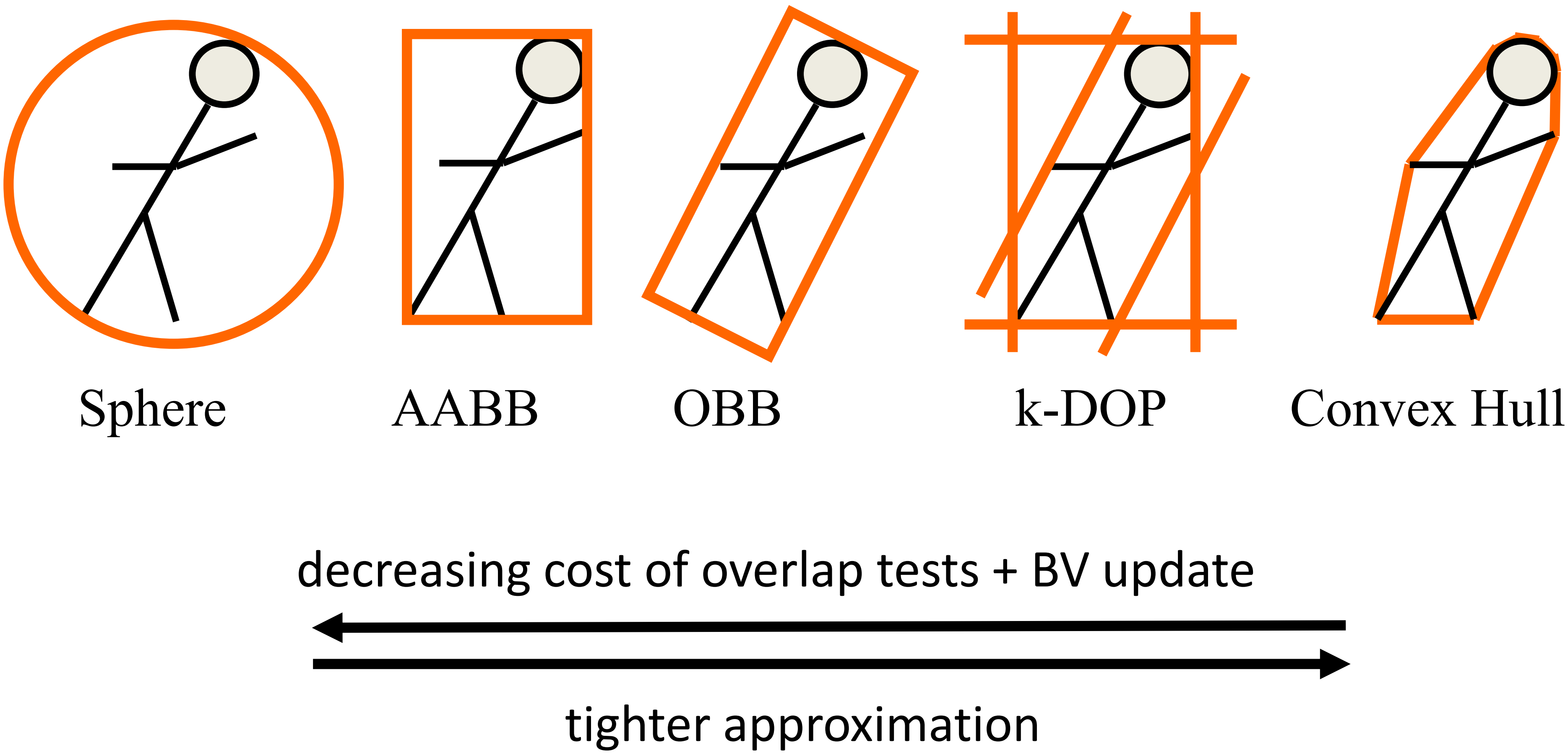
- Reduce N_p
  - Spatial coherence
  - (Temporal coherence in dynamic settings)
- Optimize C_p
  - Reduce complexity of object shapes (accuracy vs. performance)

Overview

- History
- Basics
- Bounding Volumes
- Space Partitioning

Bounding Volumes

Types of Bounding Volumes (BV)

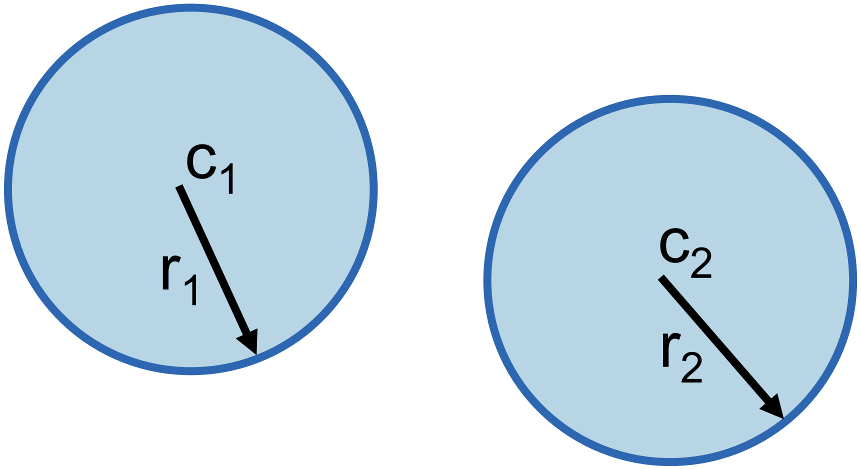


Requirements for Bounding Volumes

- Requirements:
  - tight fitting of the object
  - fast overlap tests
  - efficient recomputation of a BV in case of transformation or modification of the original object

- Observations:
  - simple primitives (spheres, AABBs, etc.) cannot fit some long skinny objects tightly
  - more complex primitives (OBBs, etc.) provide tight fits, but checking for overlap between them is relatively expensive

Bounding Spheres



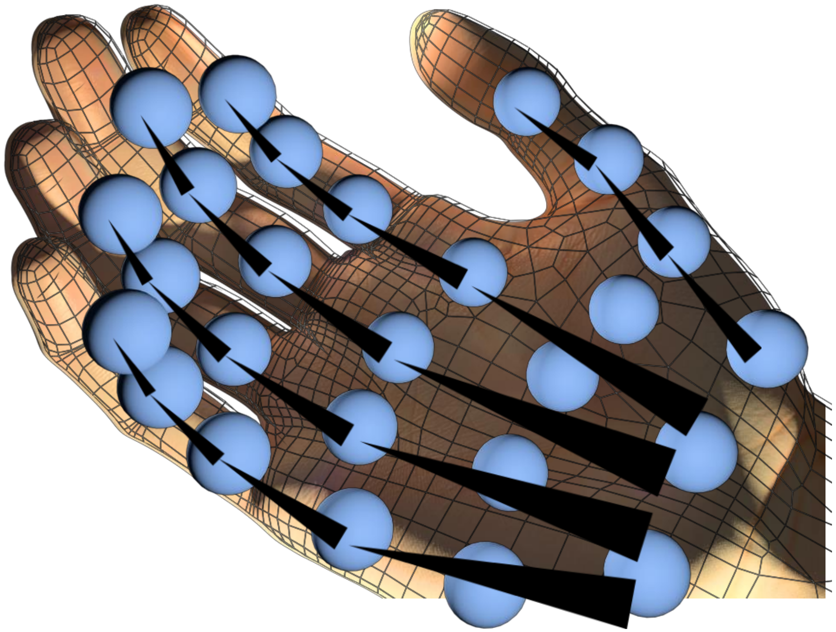
- Simple check of overlaps between two bounding spheres
- Invariant to rotations
- Translation is applied to the centers (if objects are rigid)
- Not always the best approximation

Spheres do not overlap, iff

$$\left| c_1 - c_2 \right| > r_1 + r_2$$

Natural Manipulation of Virtual Objects: A Challenge!

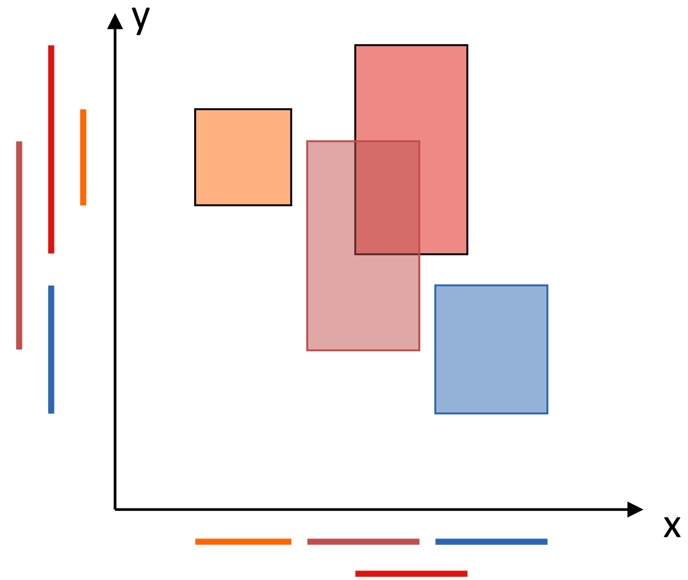
- Geometry-based
  - \+ accurate
  - \- computationally expensive



- Simplified sensor model
  - e.g., spheres
  - \+ easy collision detection
  - \- grasping only at contact points

Axis Aligned B. Boxes - Dimension Reduction

- In order for two AABBs to overlap in 3D, they must overlap along every 1D axis
  - a special case of the separating plane theorem (more on it later)
- Sort the box extents in each dimension
- Find all 1D overlapping intervals
- Tag pairs that overlap in all dimensions



- X overlaps: green-red, red-blue
- Y overlaps: orange-red,orange green, green-red, green-blue
- 2D overlaps: green-red

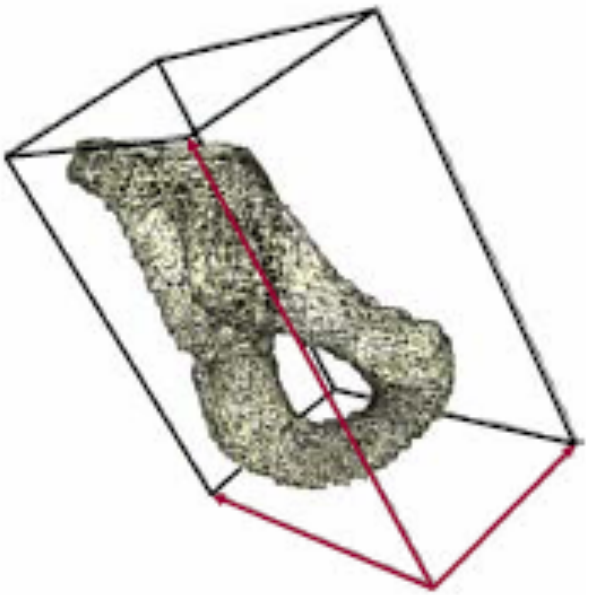
Oriented Bounding Boxes - Definition

- The orientation of the box that best fits the data
- Find the principal components
  - Point sample the convex hull of the geometry to be bound → n vertices v_i
  - Find the mean and covariance matrix of the samples

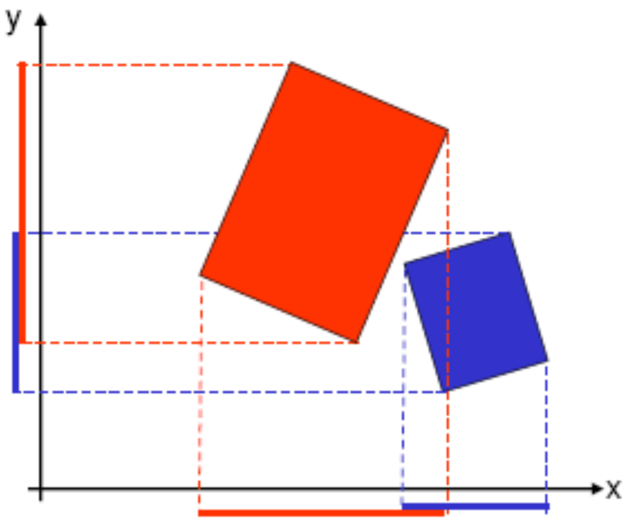
$$\mu = \frac{1}{n}\sum_{i=1}^{n} v_i \qquad C_{jk} = \frac{1}{n}\sum_{i=1}^{n} \bar{v}_{ij}\bar{v}_{ik}$$

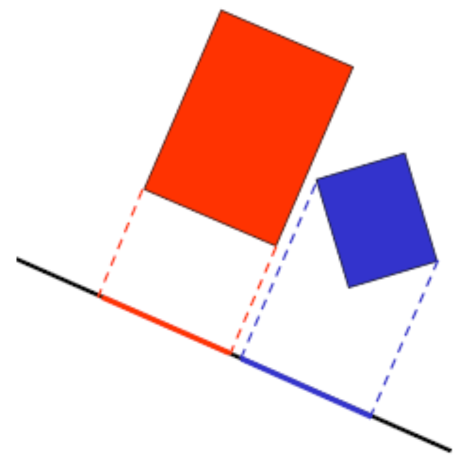
$$\bar{v}_i = v_i - \mu \qquad 1 \le j,k \le 3$$

  - The mean will be the center of the box
  - The eigenvectors of the covariance matrix are the principal directions – they are used for the axes of the box



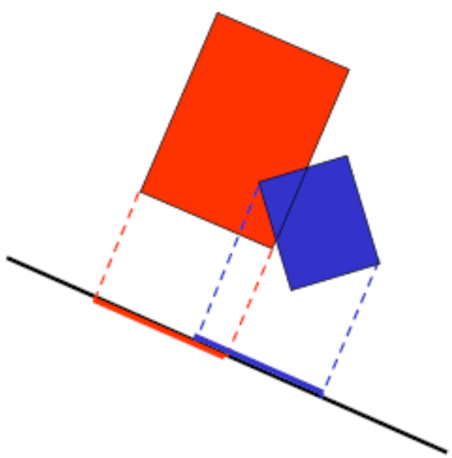
Oriented Bounding Boxes – Overlap Test

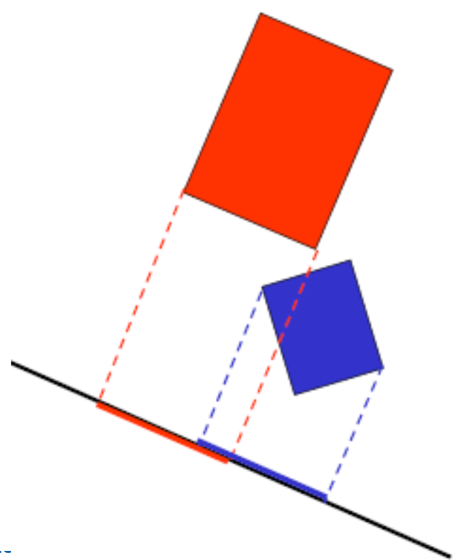




- Doesn't work

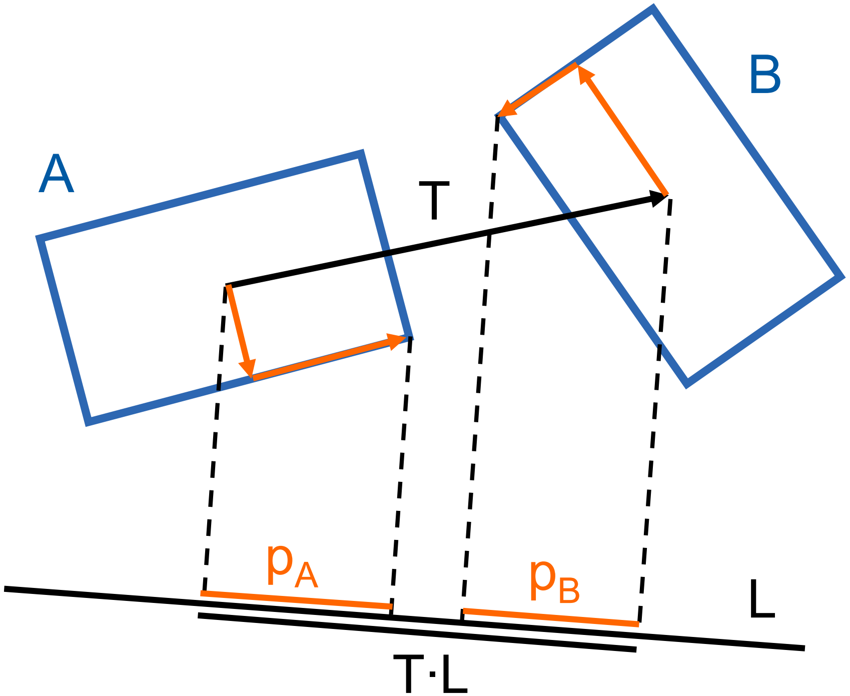
- Works!





- Not all planes do work!

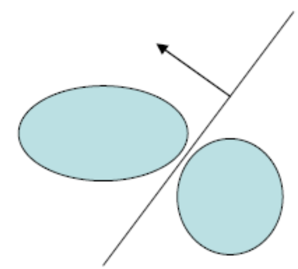
Oriented Bounding Boxes – Overlap Test



Projection and Interval Test
No Overlap, iff

$$\exists L : T \cdot L > p_A + p_B$$

- Two **convex** objects do not intersect iff there exists a plane that separates them
- Can also use the *separating axis* - the normal of the separating plane (more convenient for testing)



- How to find a separating axis?
- Which planes are candidates?

Separating Axis Theorem

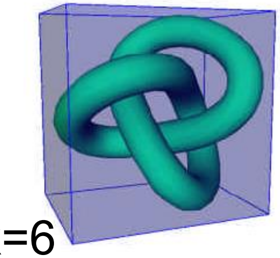
- Two convex polytopes A and B are disjoint iff there exists a separating axis which is:

  perpendicular to a face from either
  (direction of a face normal)
  or
  perpendicular to an edge from each
  (direction of the cross product of the edges)

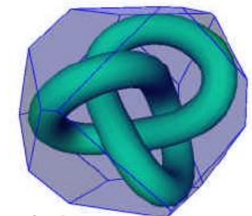
- Given two generic polytopes, each with E edges and F faces, number of candidate axes to test is: $2F + E^2$
- OBBs have only E = 3 distinct edge directions, and only F = 3 distinct face normals. OBBs need at most 15 axis tests.
- AABBs need at most 3 axis tests.

Discrete Oriented Polytopes

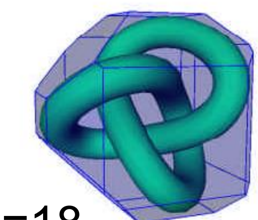
- A k-DOP is a convex polytope whose facets are determined by halfspaces whose outward normals come from a small **fixed** set of *k* orientations
- A k-DOP is represented by k/2 directions and k/2 pairs of *min, max* values



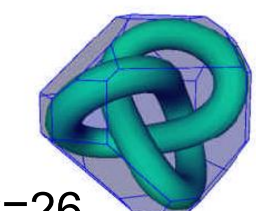
k=6



k=14

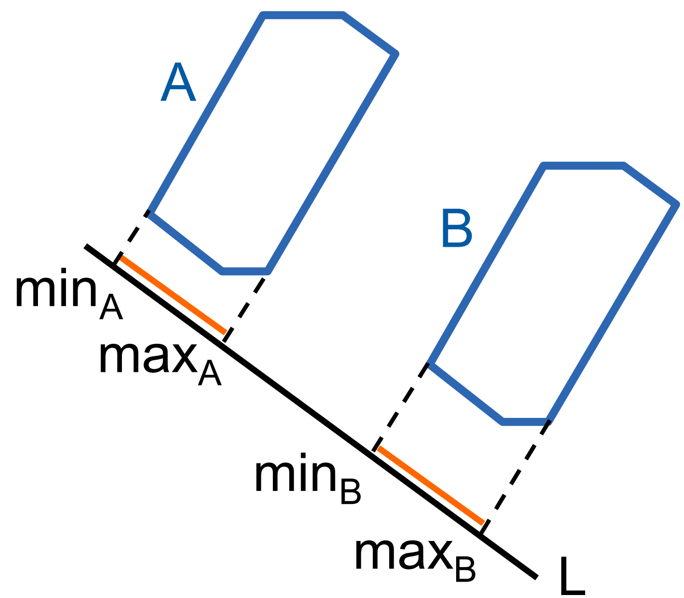


k=18



k=26

- Two k-DOPs do not overlap iff their projections in at least one direction do not overlap



$$S_l := (n_l, d_l^{min}, d_l^{max})$$

$$A := \bigcap_{1 \le l \le k/2} S_l$$

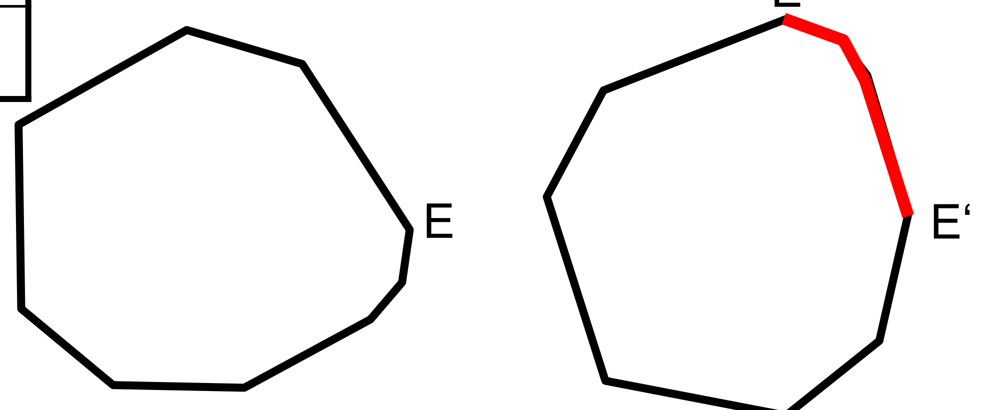
Object Transformations

- Some object transformations can be simply applied to the bounding volumes.

	translation	rotation
Sphere	☺	☺ ☺
AABB	☺	🚫
OBB	☺	☺
k-DOP	☺	🚫

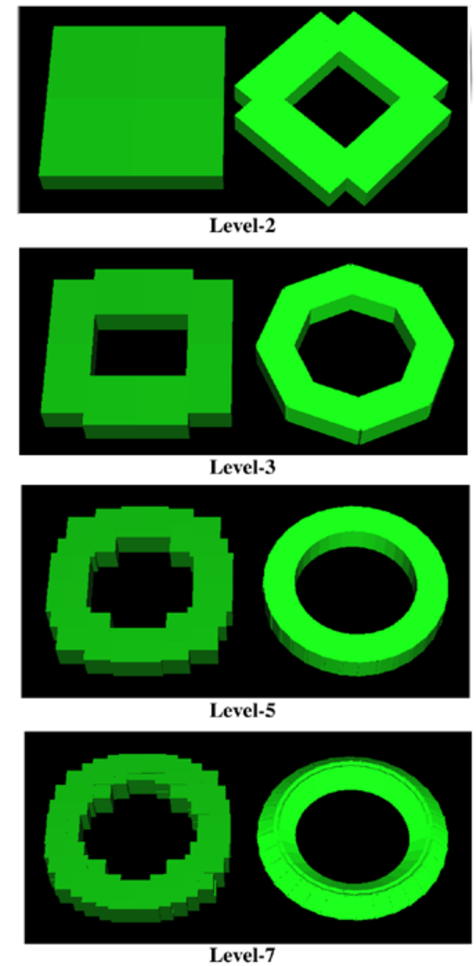
- For AABB and DOPs the BV principal directions are fixed in space → rotation not possible

- Recomputation of AABB and DOPs:
  - additional storage of the convex hull which is rotated with the object
    - check if extremal vertices are still extremal after rotation
    - compare with adjacent vertices of the convex hull
    - “climb the hill” to the extremal vertex

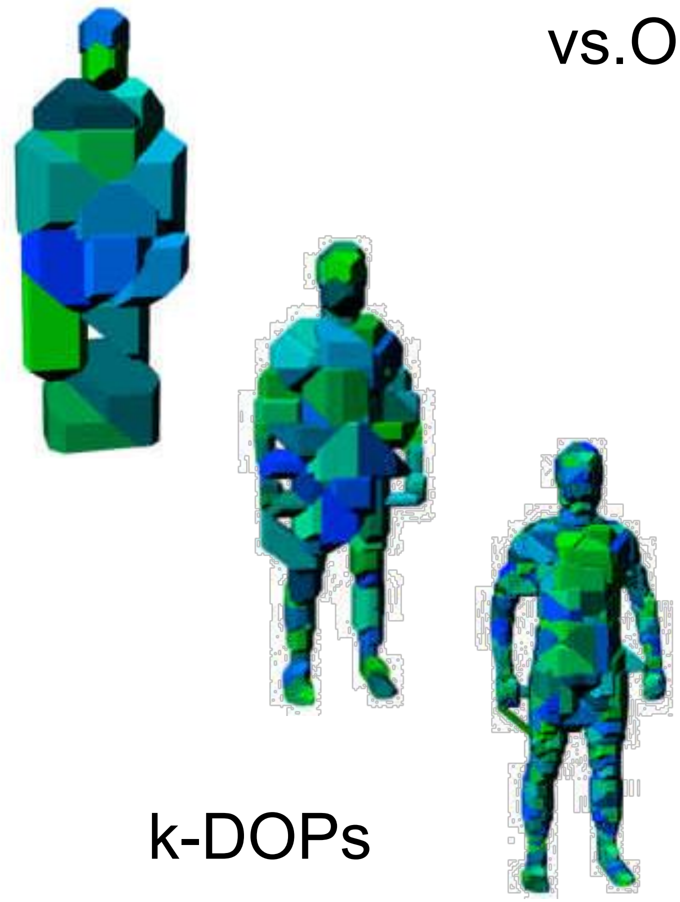


Examples of BV Hierarchies

AABB
vs.OBB



Spheres



k-DOPs

Construction of a BVH

- ***Top-Down (node split)***
  - Start with object
  - Fit a bounding volume to the object
  - Split object and bounding volume recursively according to heuristic
  - Stop, if all bounding volumes in a level contain less than *n* primitives

- ***Bottom-Up (node grouping)***
  - Start with object-representing primitives
  - Fit a bounding volume to given number of primitives
  - Group primitives and bounding volumes recursively
  - Stop in case of a single bounding volume at a hierarchy level

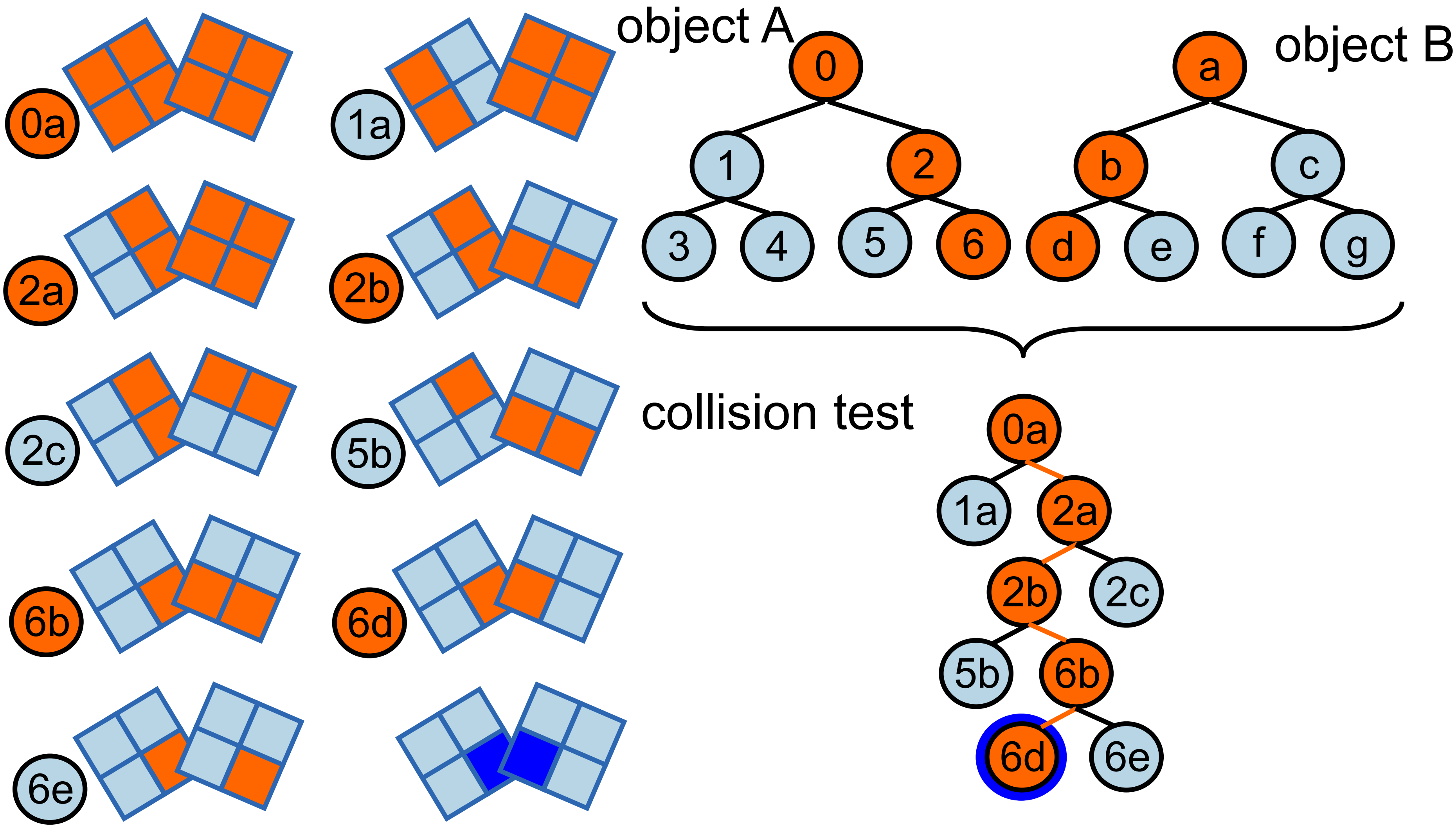
Bounding Volumes Hierarchy (Tree)

- ***Parameters***
  - Bounding volume
  - Bottom-up/top-down
  - Branching factor (binary, 4-ary, k-d-tree, …)
  - Heuristic to subdivide/group object primitives or bounding volumes
  - How many primitives in each leaf of the BV tree
- ***Goals***
  - Balanced tree
  - Tight-fitting bounding volumes
  - Minimal redundancy (primitives in more than one BV per level)

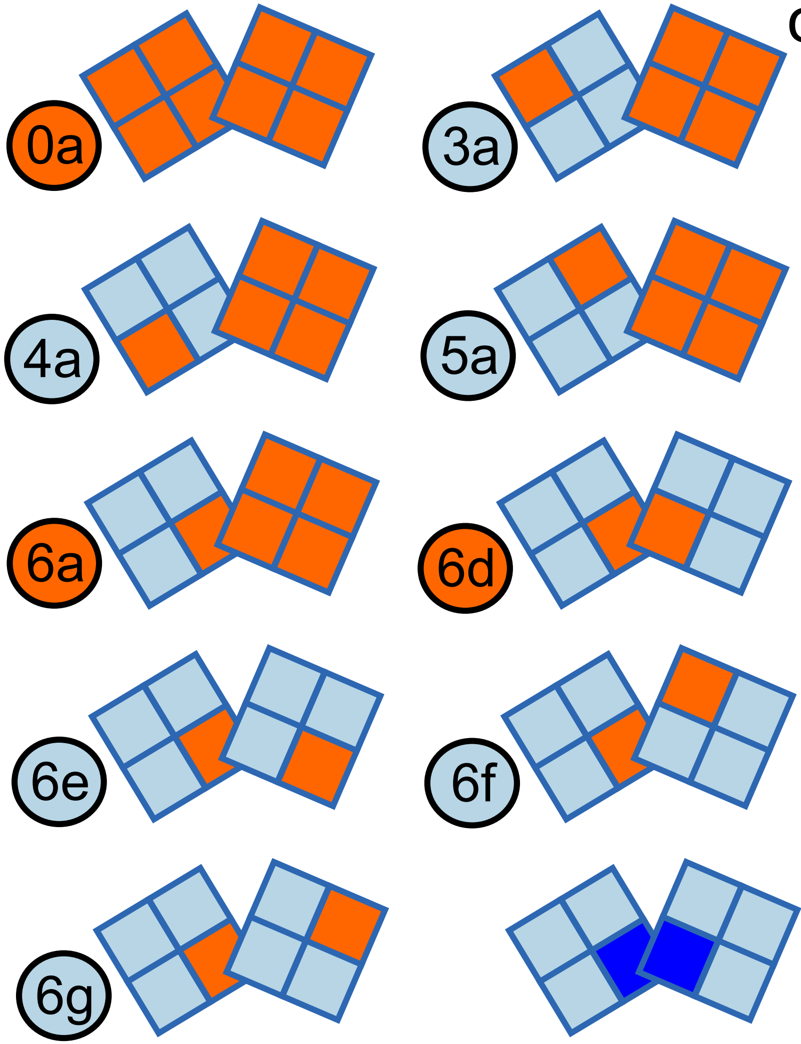
BVH Interference Check

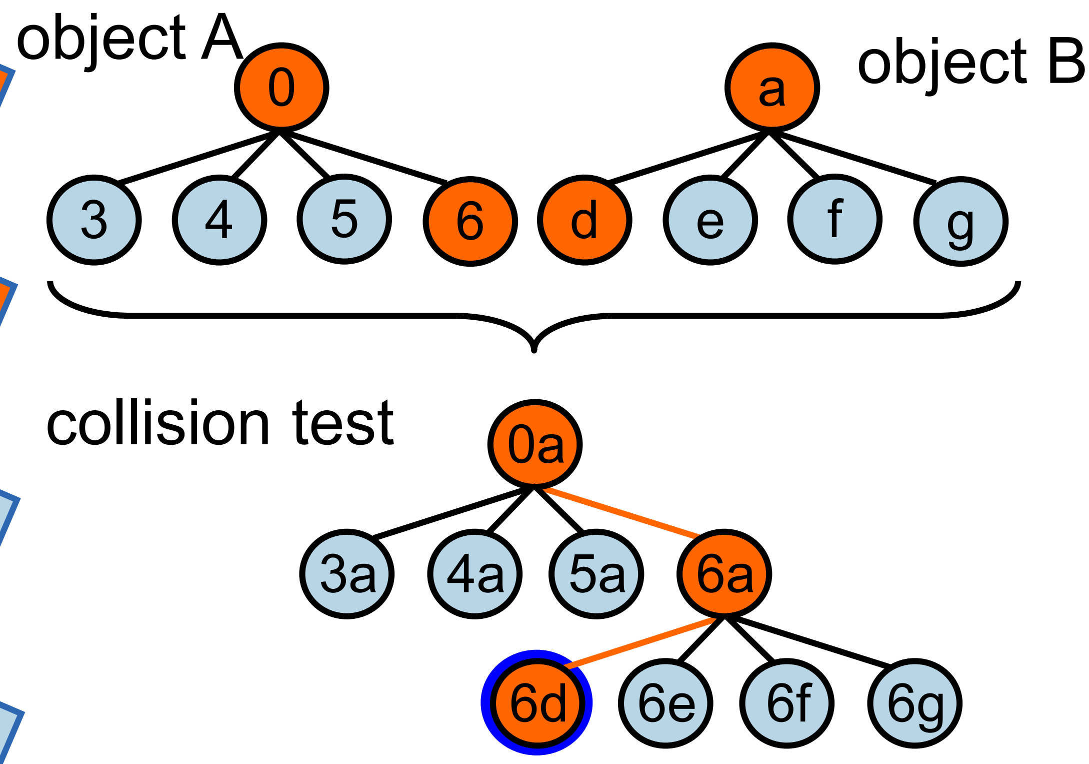
1. interference check for two nodes (start with roots)
2. if no interference then return “no collision” else
3. all children of one node are checked against all children of the other node
4. if no interference then return “no collision” else
5. if at leave nodes then “collision” else go to 3

Binary Trees



Higher Order Trees





fewer nodes → lower total update costs
lower recursion depth → lower memory requirements on the stack

Collision Query: Computational Cost

- Basic equation: $F = N_p \times C_p$

  F ... total cost of interference detection
  N_p ... number of (primitive) pairs tests
  C_p ... cost of (primitive) pair test

- Naïve approach: $N_p = O(N^2)$ N: Number of objects in the scene

- Reduce N_p
  - Spatial coherence
  - (Temporal coherence in dynamic settings)
- Optimize C_p
  - Reduce complexity of object shapes (accuracy vs. performance)

BVH - Computational Cost

- $F = N_u \times C_u + N_{bv} \times C_{bv} + N_p \times C_p$

  F ... total cost of interference detection
  N_u ... number of BV updated, C_u ... cost of updating a BV
  N_{bv} ... number of BV pair overlap tests, C_{bv} ... cost of BV pair overlap test
  N_p ... number of primitive pairs tests, C_p ... cost of primitive pair test

- infrequent BV updates minimize N_u
- C_{bv} and N_p depend on broad phase's scheme
  - cheap test reduces C_{bv}, but might result in an increased N_p
  - OBB-Tree is good at N_p and not bad at C_{bv}, whereas AABB is good at C_{bv} but not so good at N_p

Rigid Bodies vs. Deformable Bodies

- Rigid Objects:
  - use OBBs as they are usually tighter fitting and can be updated by applying translations and rotations.
  - update complete BVH by applying transformations
  - usually small number of collisions occur
- Deformable Object:
  - use DOPs as update costs are lower than for OBBs
  - update by refitting or rebuilding each BV separately (top-down, bottom-up)
  - high number of collisions may occur
  - Self-collisions need to be detected
  - use higher order trees (4-ary, 8-ary)

Overview

- History
- Basics
- Bounding Volumes
- Space Partitioning

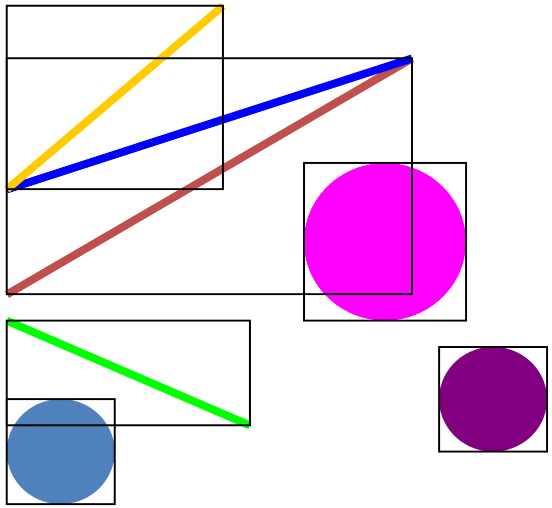
Bounding Volumes vs. Spatial Partitioning

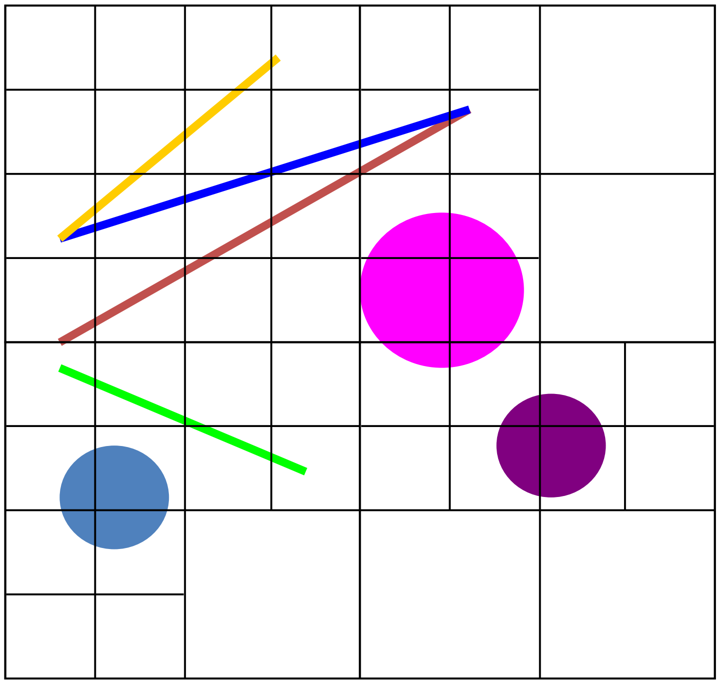
BV:

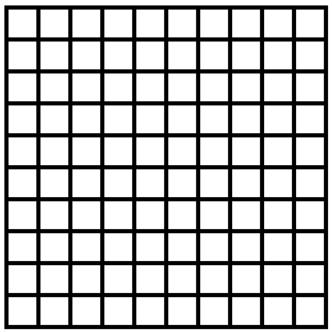
- Object centric
- Spatial redundancy

SP:

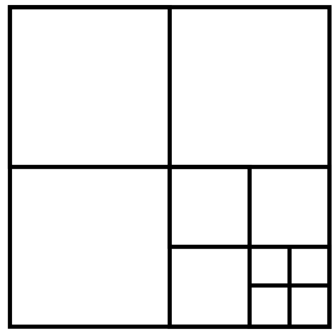
- Space centric
- Object redundancy



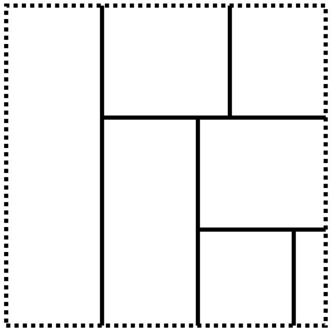




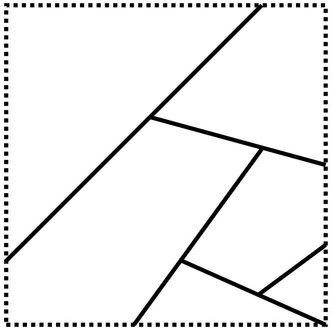
voxel grid



octree



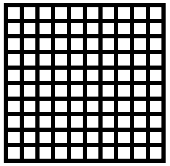
k-d tree



BSP-tree

- cells maintain references to primitives intersecting the cell
- only test pairs of objects that share a cell
- information is updated for each object transformation
- octree, k-d tree and BSP-tree are object dependent
- voxel grid is object independent

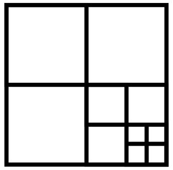
Voxel Grid

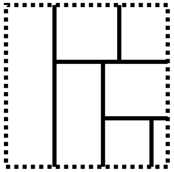


- space partitioning into uniform rectangular, axis-aligned cells
- primitives per cell are found by
  - scan conversion of primitives to the grid or
  - scan conversion of AABBs of the primitives
- fast cell access
- optimal cell size?
  - large cells increase the number of primitives per cell
  - small cells cause spreading of primitives to a large number of cells
- less efficient in case of non-uniform primitive distribution

Octree and k-d Tree

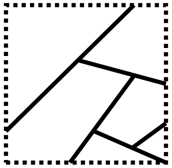
- hierarchical structures
- space partitioning into rectangular, axis-aligned cells
- octree root node corresponds to AABB of a scene
- leaves represent cells which maintain primitive lists
- uniform or non-uniform subdivision
- adaptive to distribution of primitives
  - large cells in case of low density of primitives
  - small cells in case of high density
- dynamic update (can be compute-intensive!)
  - cells with many primitives can be subdivided
  - cells with less primitives can be merged





BSP Tree

- generalized k-d tree
- space is subdivided by arbitrarily oriented planes
- space partitioning into convex cells



This was it!