# Computer Animation and Games I

CM50244

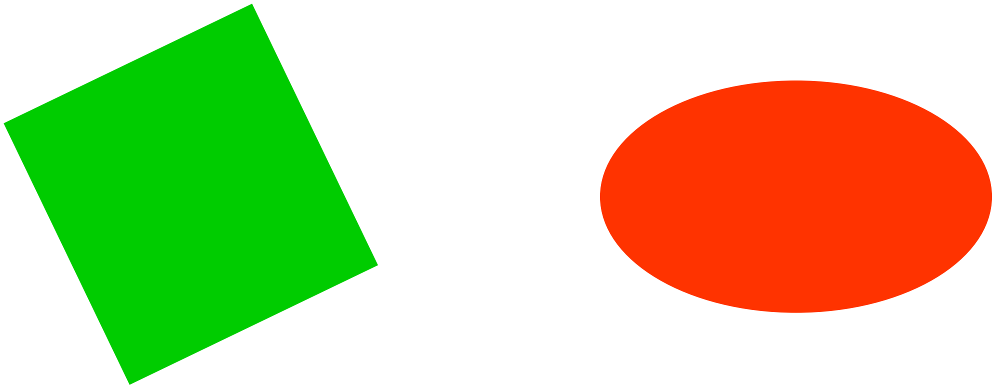
# Today's Lectures

- **Collision Detection**
- Skeletal Motion Capture
- Principles of Animation

# Collision Detection

# Problem Statement

- The problem can be defined as if the two objects intersect and where is the intersection



- Collision Detection is an important problem in fields like
  - computer animation
  - virtual reality
  - computer games

# The Simple Solution

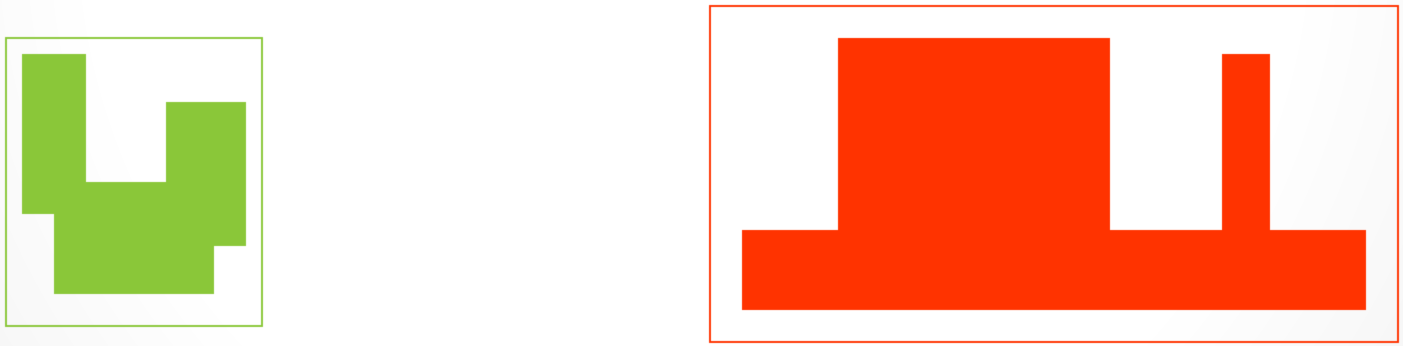
- Pairwise collision check of all triangles the objects have
- Problem:
    - complexity $O(n^2)$
    - not acceptable for reasonable number $n$ of polygons
    - not applicable for real-time application

# Overview

- **Bounding volumes**
- Hierarchy
- Multiple Objects

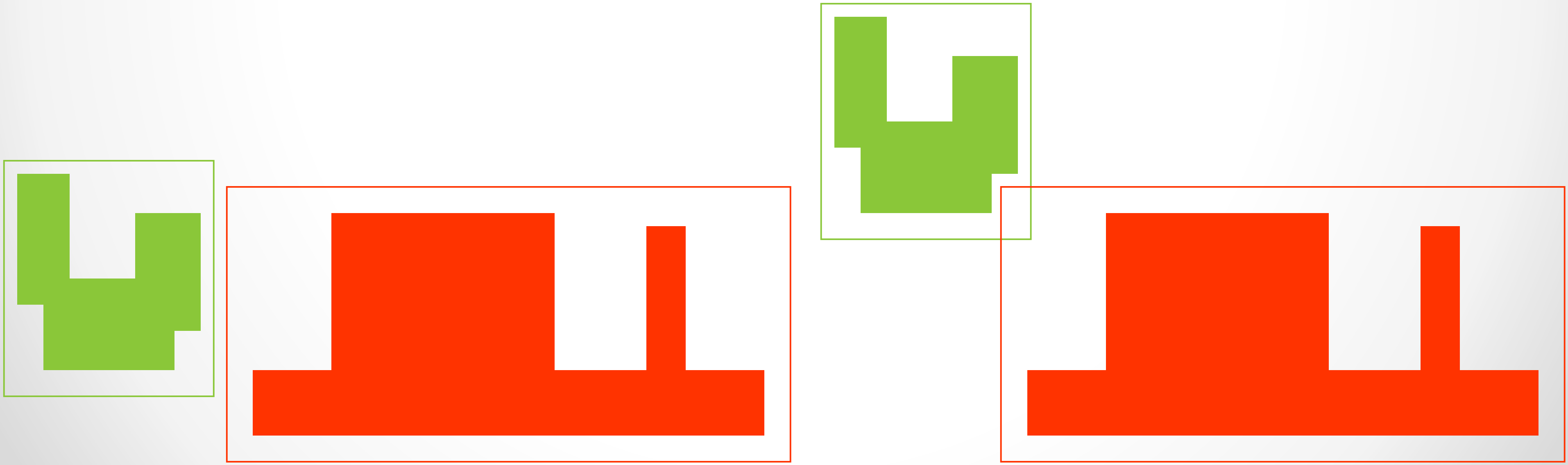
# Bounding Volumes

- Reduce complexity of collision computation by substitution of the (complex) original object with a simpler object containing the original one.



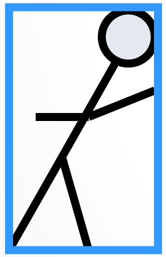
# Bounding Volumes

- The original objects can only intersect if the simpler ones do.
- In other words, if the simpler objects do NOT intersect, the original objects won't either.

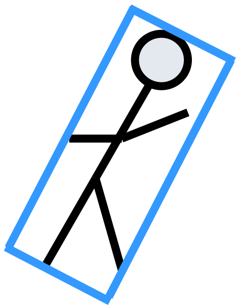


# Different BVs

- Axes Aligned Bounding Boxes (AABB)
- Oriented Bounding Boxes (OBB)
- Spheres
- *k*-Discrete Oriented Polytopes (*k*-DOP)

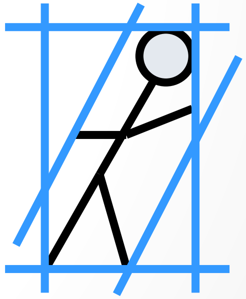


AABB



OBB

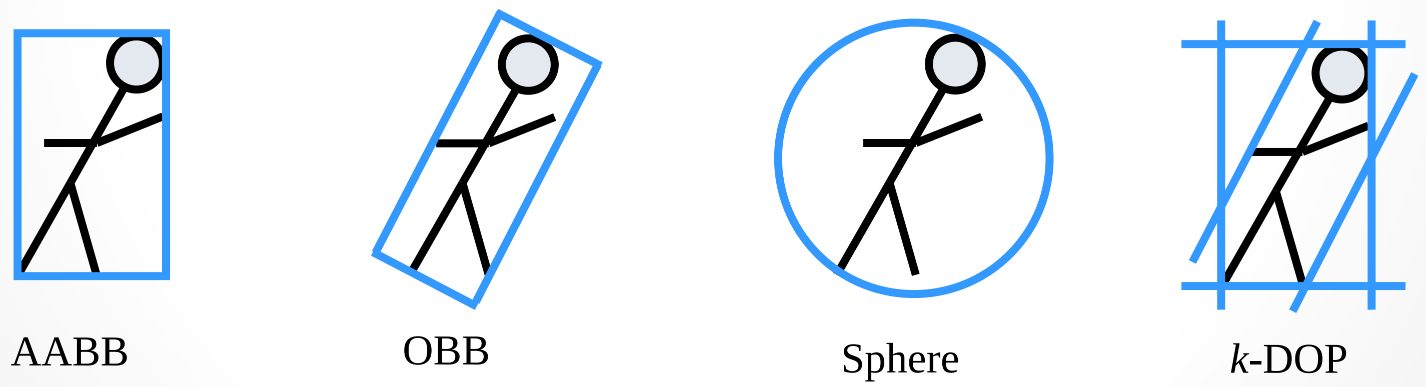
Sphere



*k*-DOP

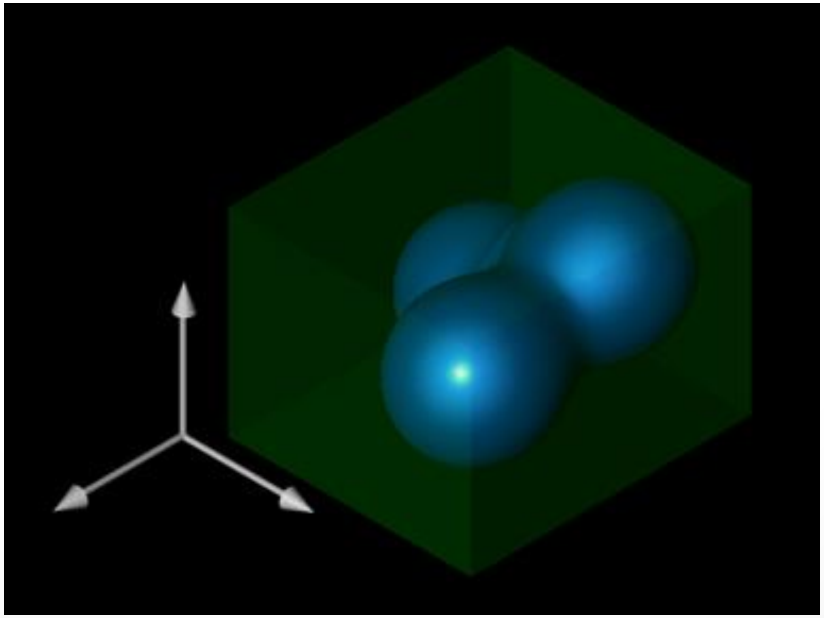
# How to Choose BVs

- Object approximation behavior ('Fill efficiency')
- Computational simplicity
- Behavior on transformation, easy to update?
- Memory efficiency



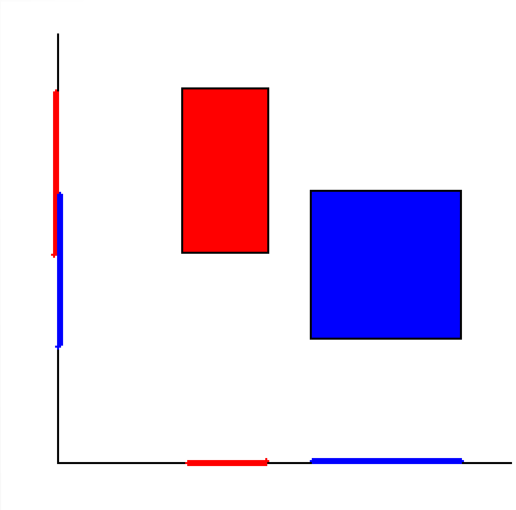
# Axes Aligned Bounding Box

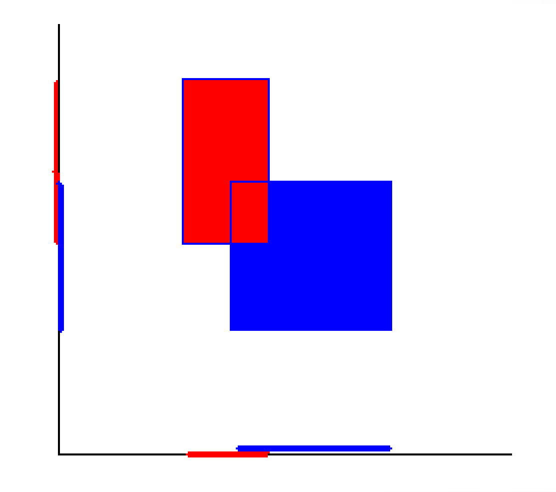
- Align axes to the coordinate system
- Simple to create
- Computationally efficient
- Unsatisfying fill efficiency
- Not invariant to basic transformations, e.g. rotation



# Axes Aligned Bounding Box

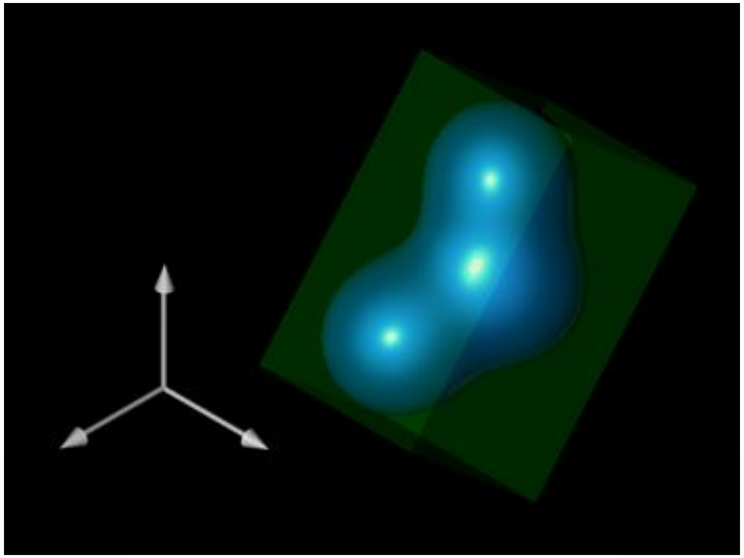
- Collision test: project BBs onto coordinate axes. If they overlap on each axis, the objects collide.





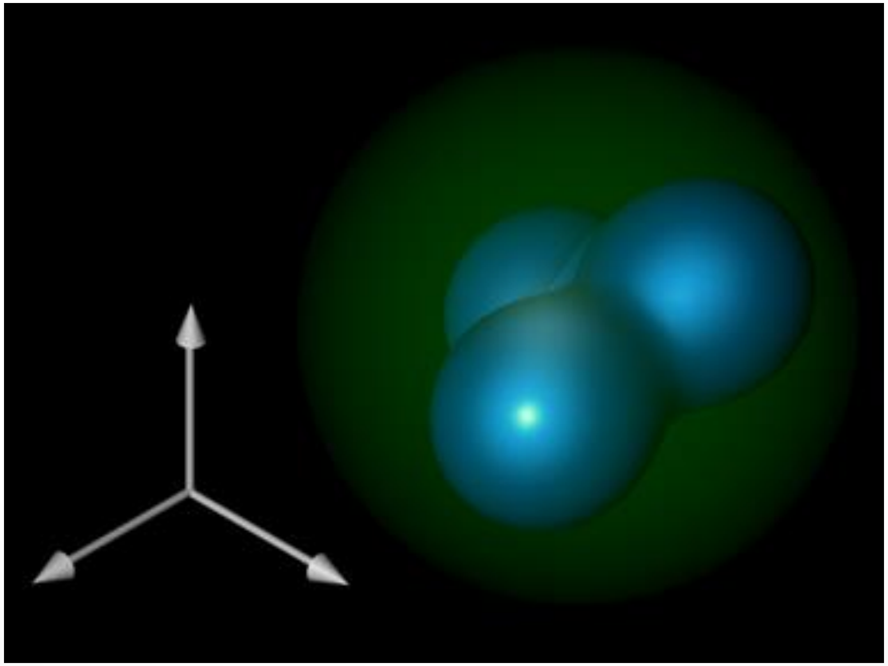
# Oriented Bounding Box (OBB)

- Align box to object such that it fits optimally in terms of fill efficiency
  - Computationally expensive
  - Invariant to rotation
  - Complex intersection check



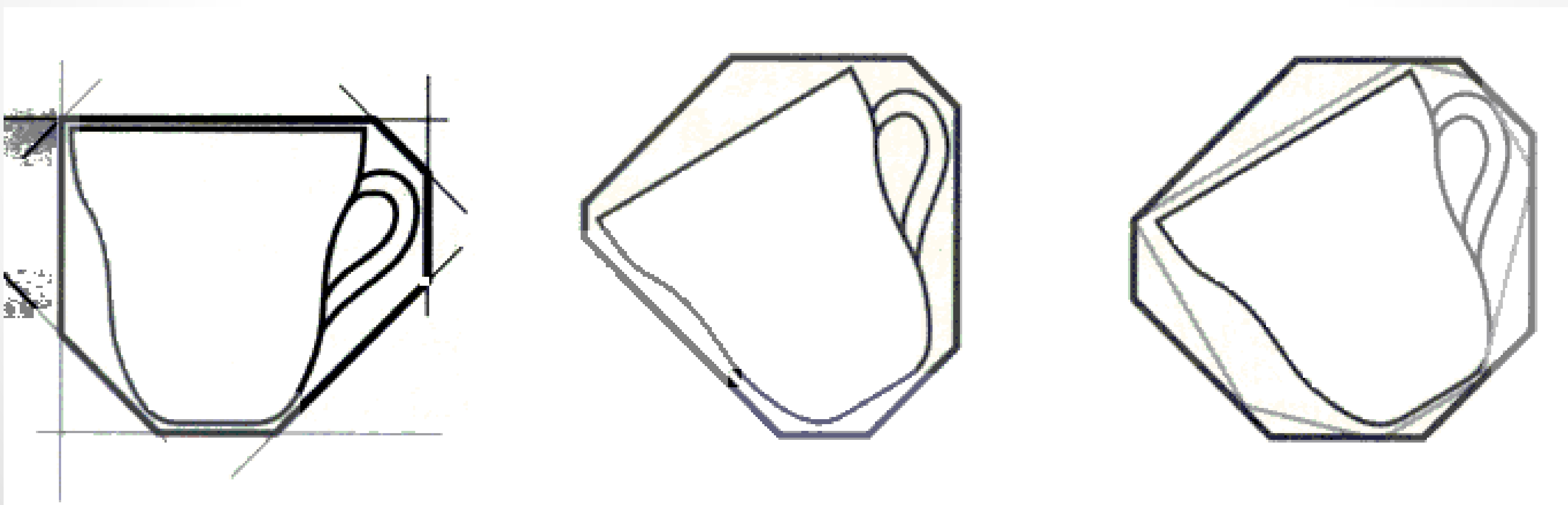
# Sphere

- Relatively complex to compute
- Bad fill efficiency
- Simple overlap test
- Invariant to rotation



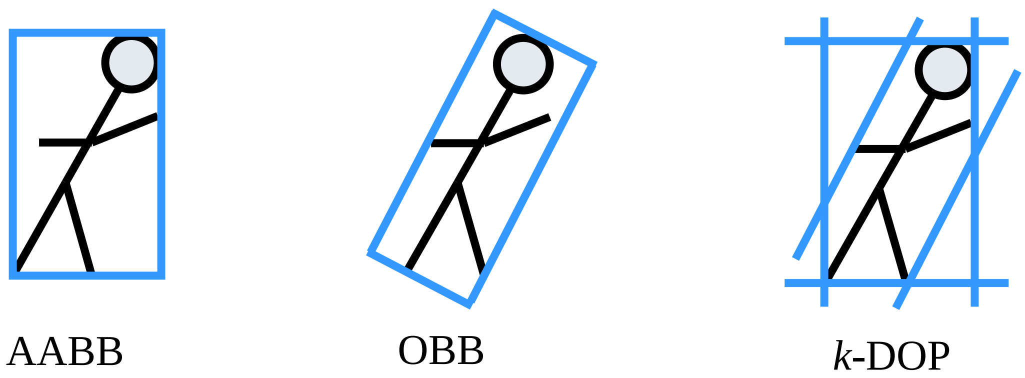
# *k*-DOP

- Easy to compute
- Good fill efficiency
- Simple overlap test
- Not invariant to rotation



# *k*-DOP

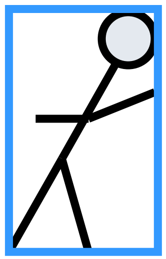
- *k*-DOP is considered to be a trade off between AABBs and OBBs.



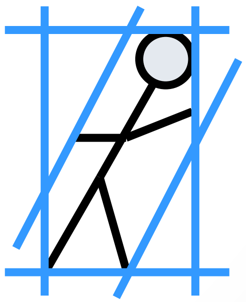

- Its collision check is a general version of the AABB collision check, having *k*/2 directions

# *k*-DOP

- *k*/2 directions $\boldsymbol{B}_i$ define *k* planes ($\boldsymbol{B}_i$ are the normals of the planes)
- These *k* planes define/bound the *k*-DOP bounding volume.
- AABB is a special *k*-DOP with k=4, two directions are x-direction and y-direction



AABB



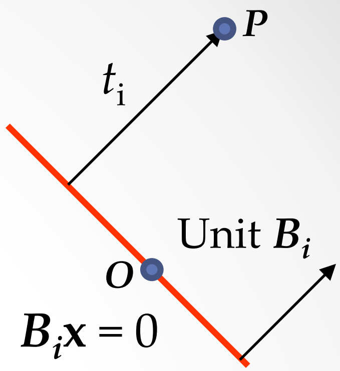
k-DOP

# *k*-DOP

- *k*-DOPs are used e.g. in the game ‘Cell Damage’ (XBOX, Pseudo Interactive, 2002)

# 3D Example: UNREAL-Engine

# Compute $k$-DOP




- Again: plane $H_i = \{\mathbf{x} \mid \boldsymbol{B}_i\, \mathbf{x} - t_i = 0\}$

- If the directions $\boldsymbol{B}_i$ are predefined, only the distance $t_i$ must be computed to specify the plane $H_i$ .

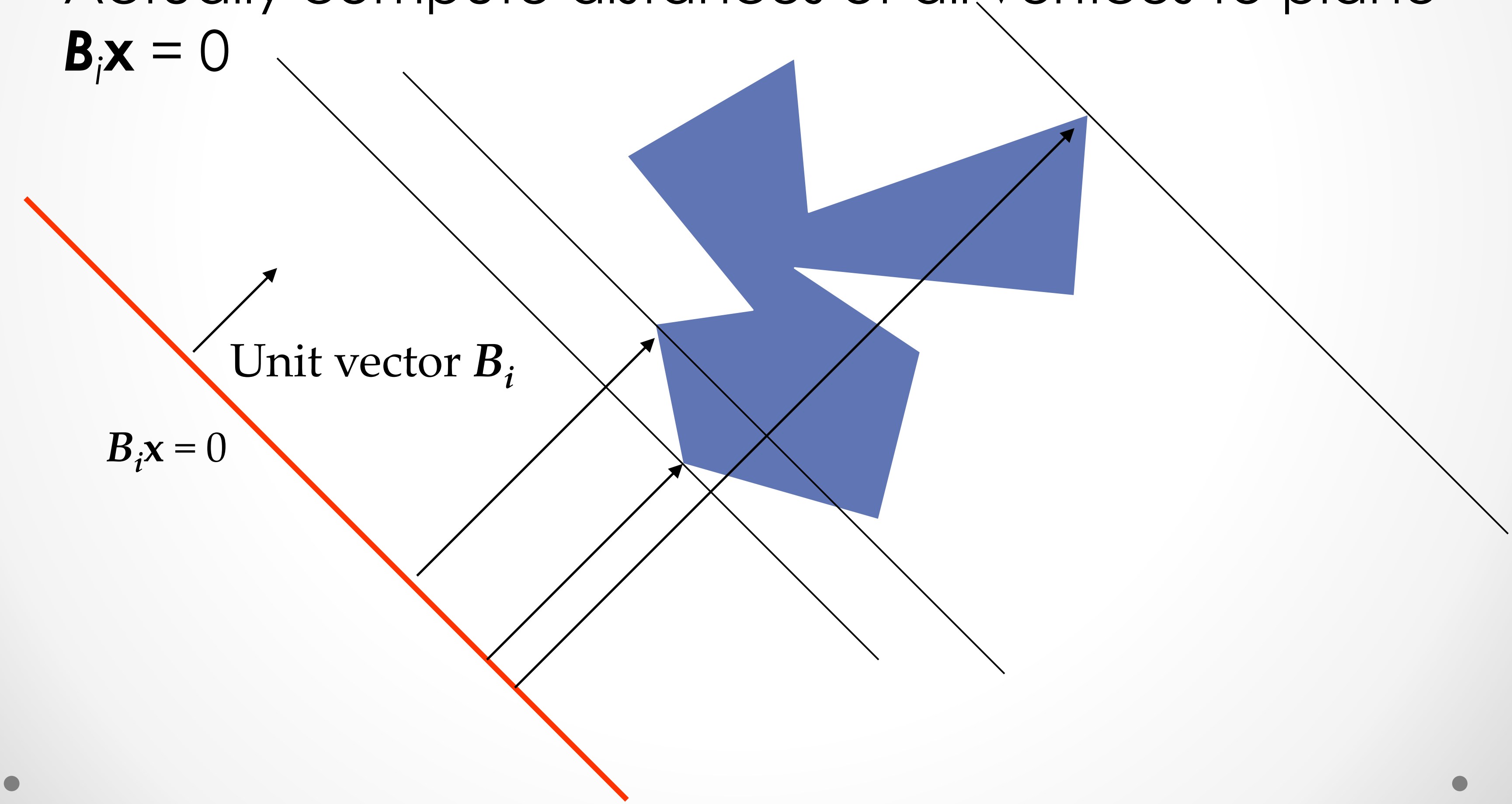
- We have two bouding planes in each direction. So two scalar values per direction ($d_i$, $D_i$ ; $d_i < D_i$).

- $\boldsymbol{B}_i\, \mathbf{x} - t_i = 0 \quad \Rightarrow \quad t_i = \boldsymbol{B}_i\, \mathbf{x}$

- $d_i = \min t_i$ , $D_i = \max t_i$

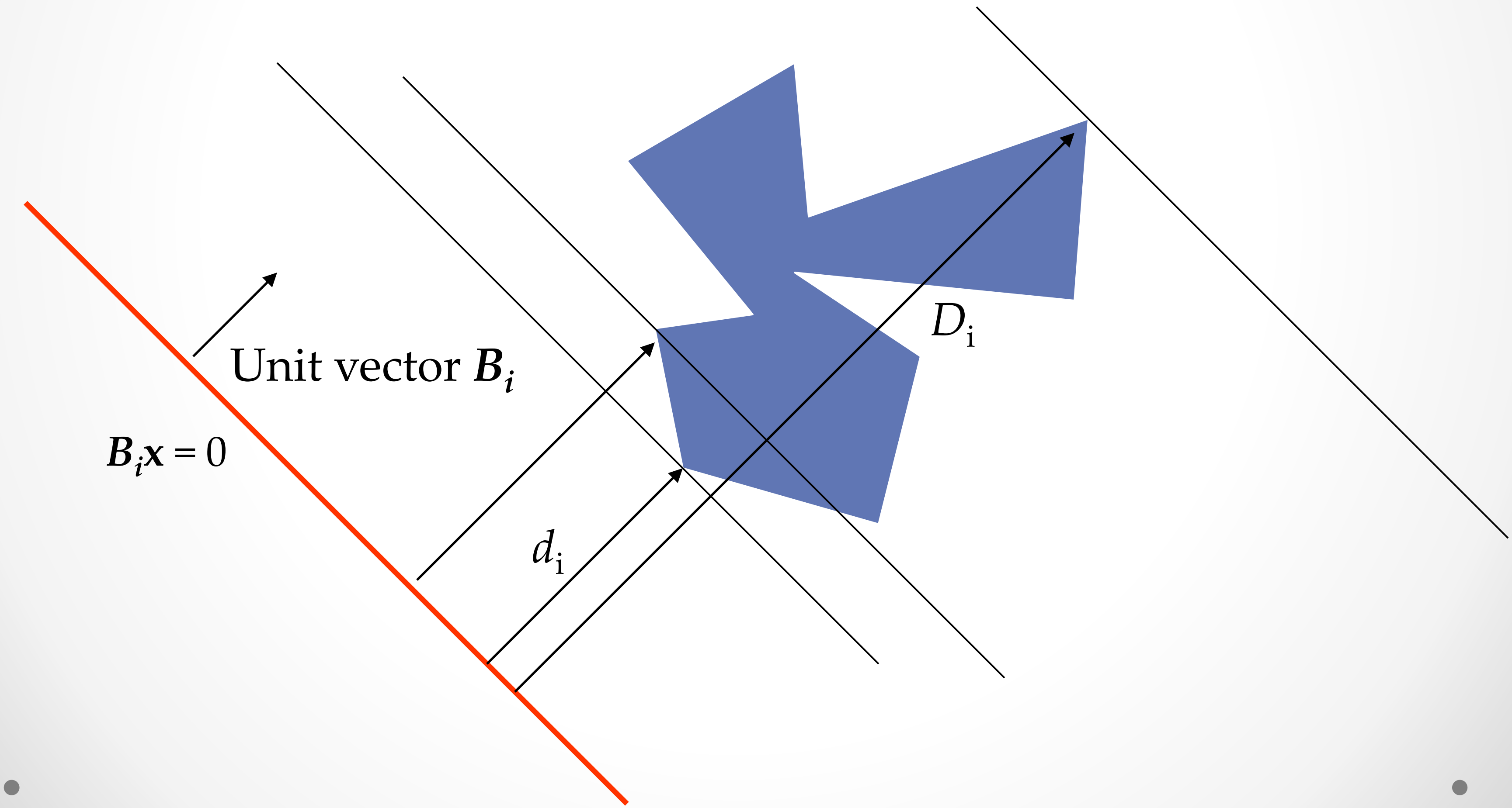
# Compute $k$-DOP

- Multiply (dot product) each vertex with the unit normal vector $\boldsymbol{B}_i$
- Actually compute distances of all vertices to plane $\boldsymbol{B}_i\mathbf{x} = 0$



# Compute $k$-DOP

- $d_i$ is the minimum distance of the object to the plane $\boldsymbol{B}_i\mathbf{x} = 0$, $D_i$ is the maximum distance

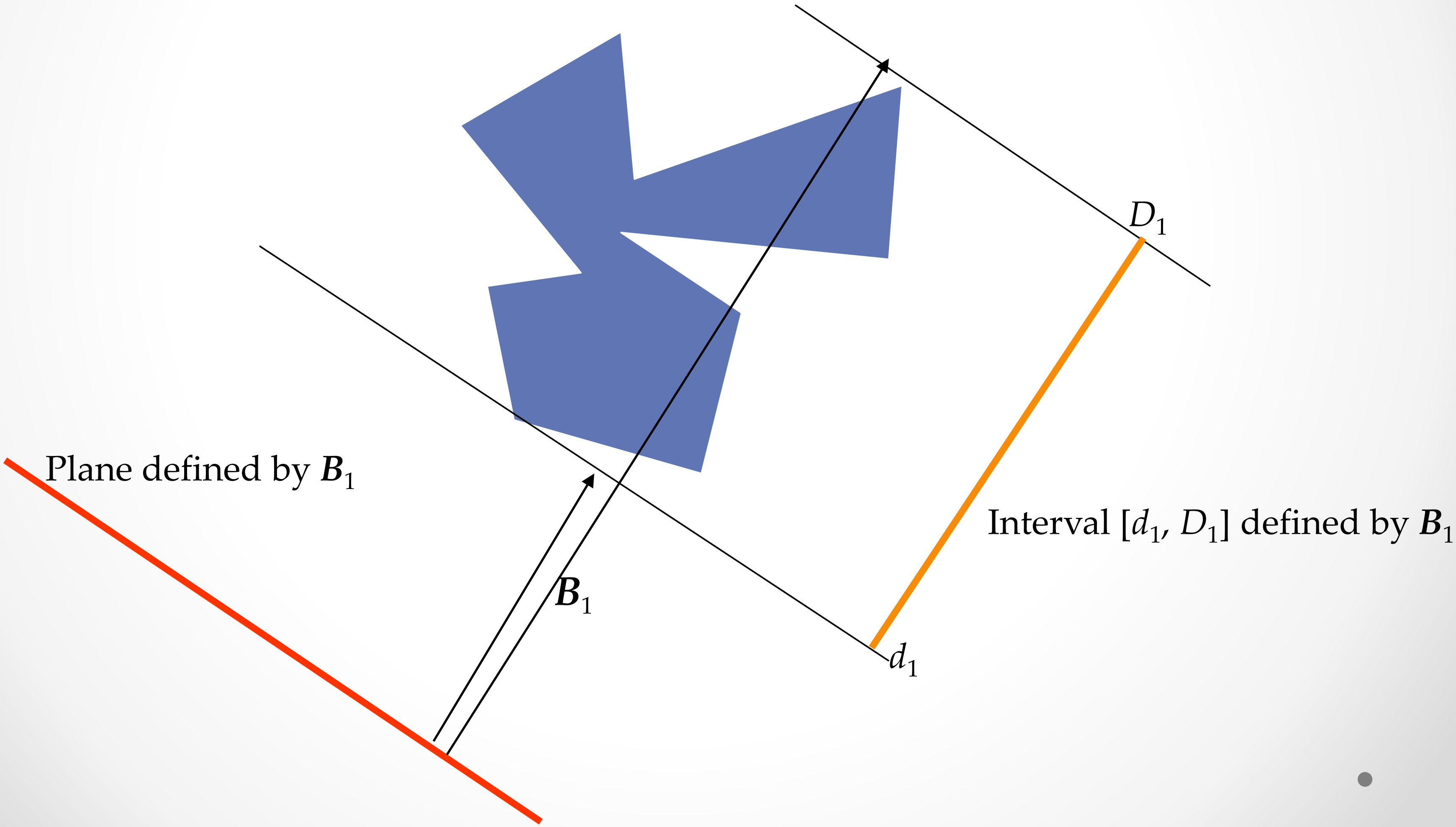


# Collision from *k*-DOP

- Given: $k$ directions $\boldsymbol{B}_i$ and $V$ = set of vertices of object.
- Compute $d_i = \min\{\boldsymbol{B}_i \mathbf{v} \mid \mathbf{v} \text{ in } V\}$ and $D_i = \max\{\boldsymbol{B}_i \mathbf{v} \mid \mathbf{v} \text{ in } V\}$
  $d_i$ and $D_i$ define an interval on the axis given by $\boldsymbol{B}_i$ .
- This is the interval needed for the collision detection
- Overall there are $k/2$ intervals

# Collision from *k*-DOP

- Check overlap for *k*/2 intervals, a general version of the AABB collision check

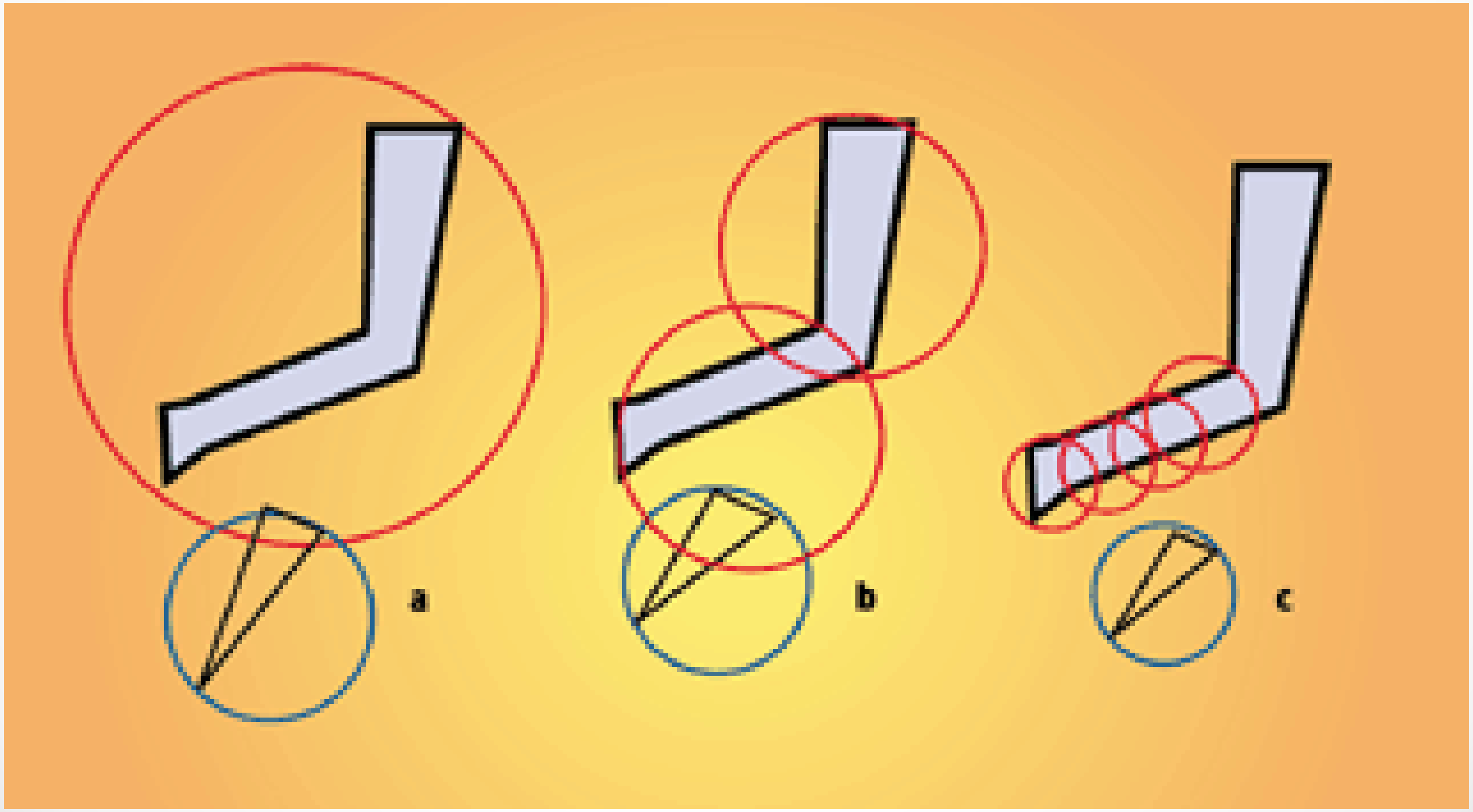


# Overview

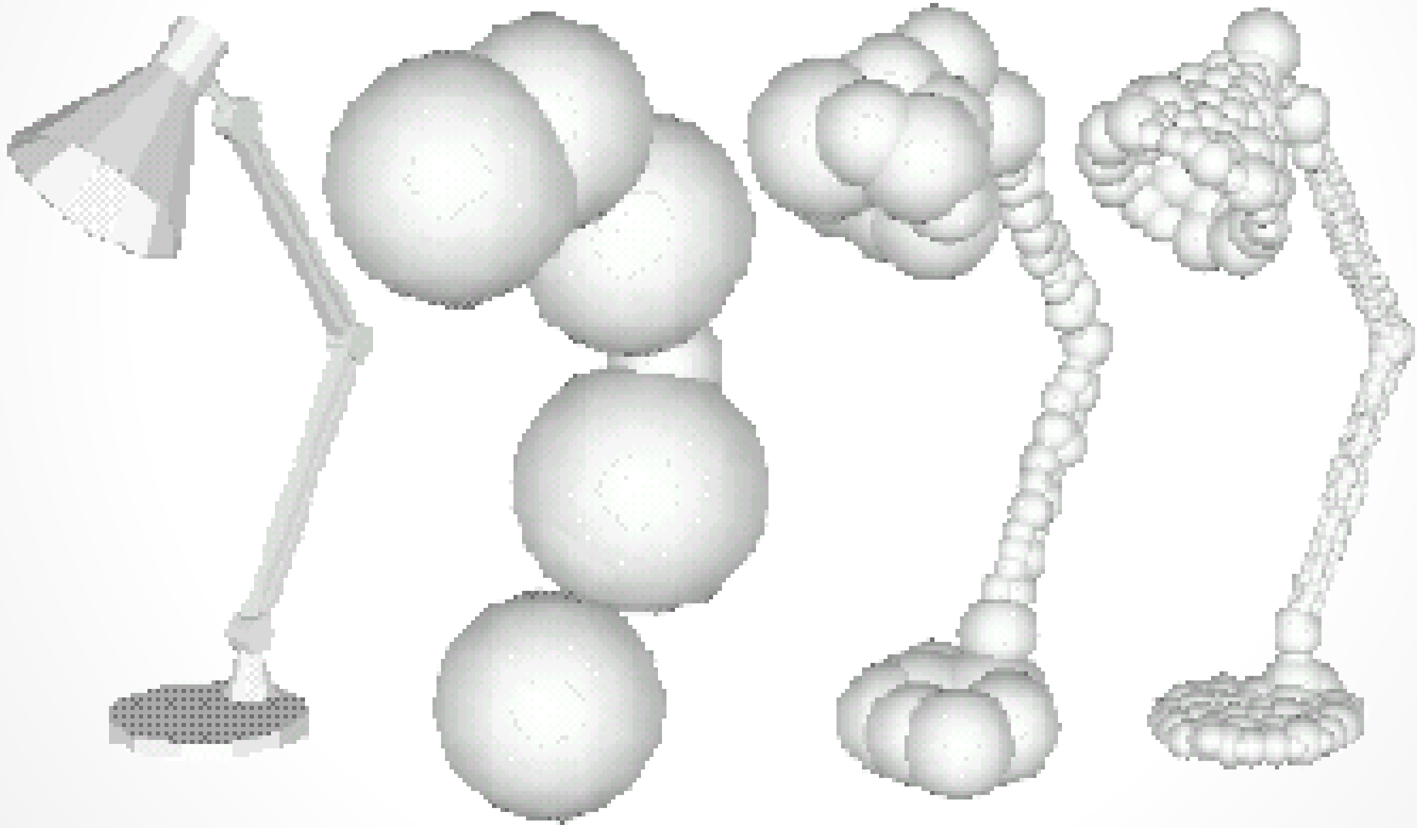
- Bounding volumes
- **Hierarchy**
- Multiple Objects

# Basic Idea

- To achieve higher exactness in collision detection, build a multiscale BV representation of the object



# Hierarchical Bounding Spheres



# Different Hierarchies

- The hierarchy is stored in a tree, named by the underlying BV scheme:
  - AABB – tree
  - OBB – tree
  - Sphere – tree
  - K-DOP – tree

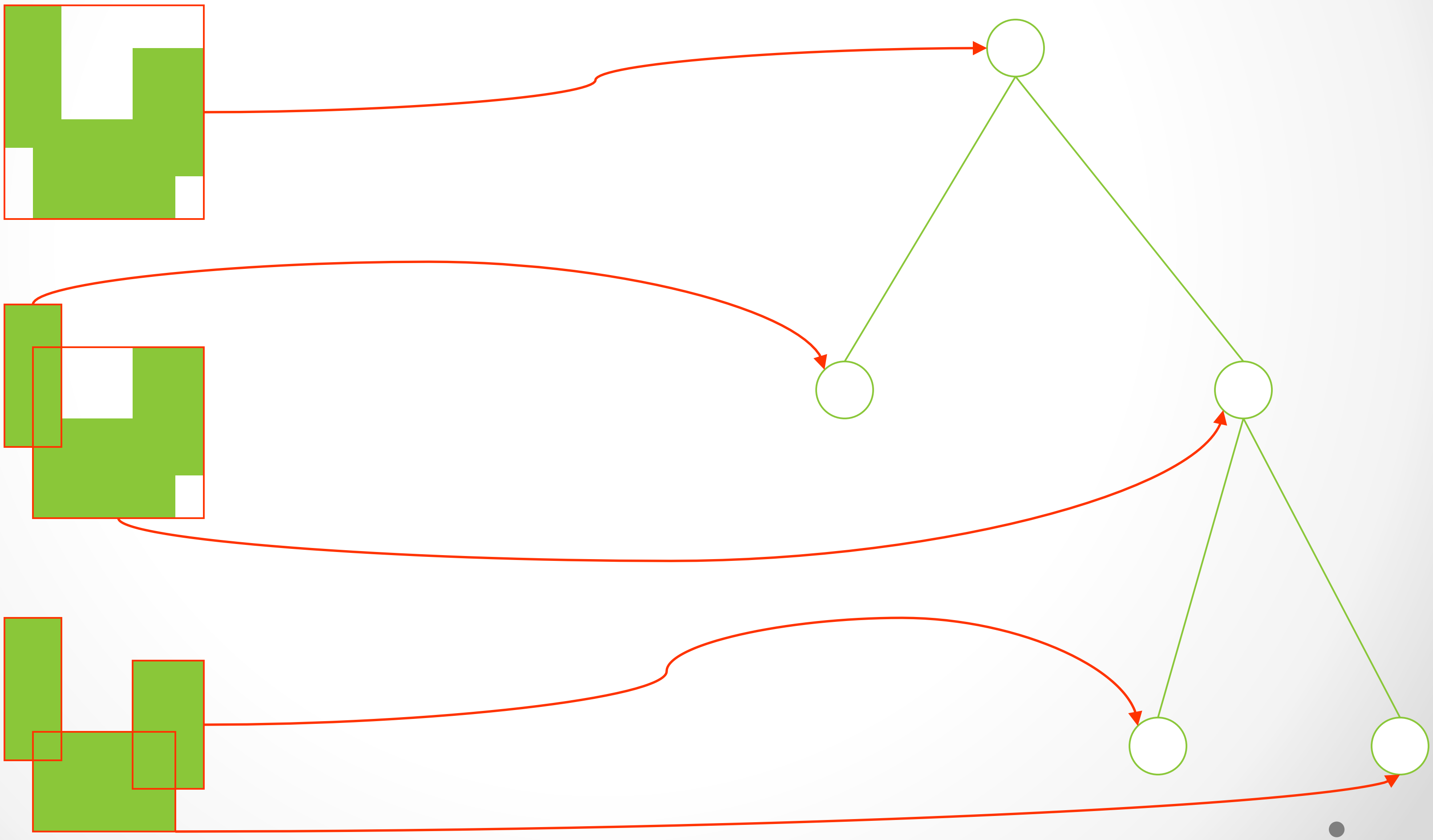
Sphere Trees are used in “Gran Tourismo”

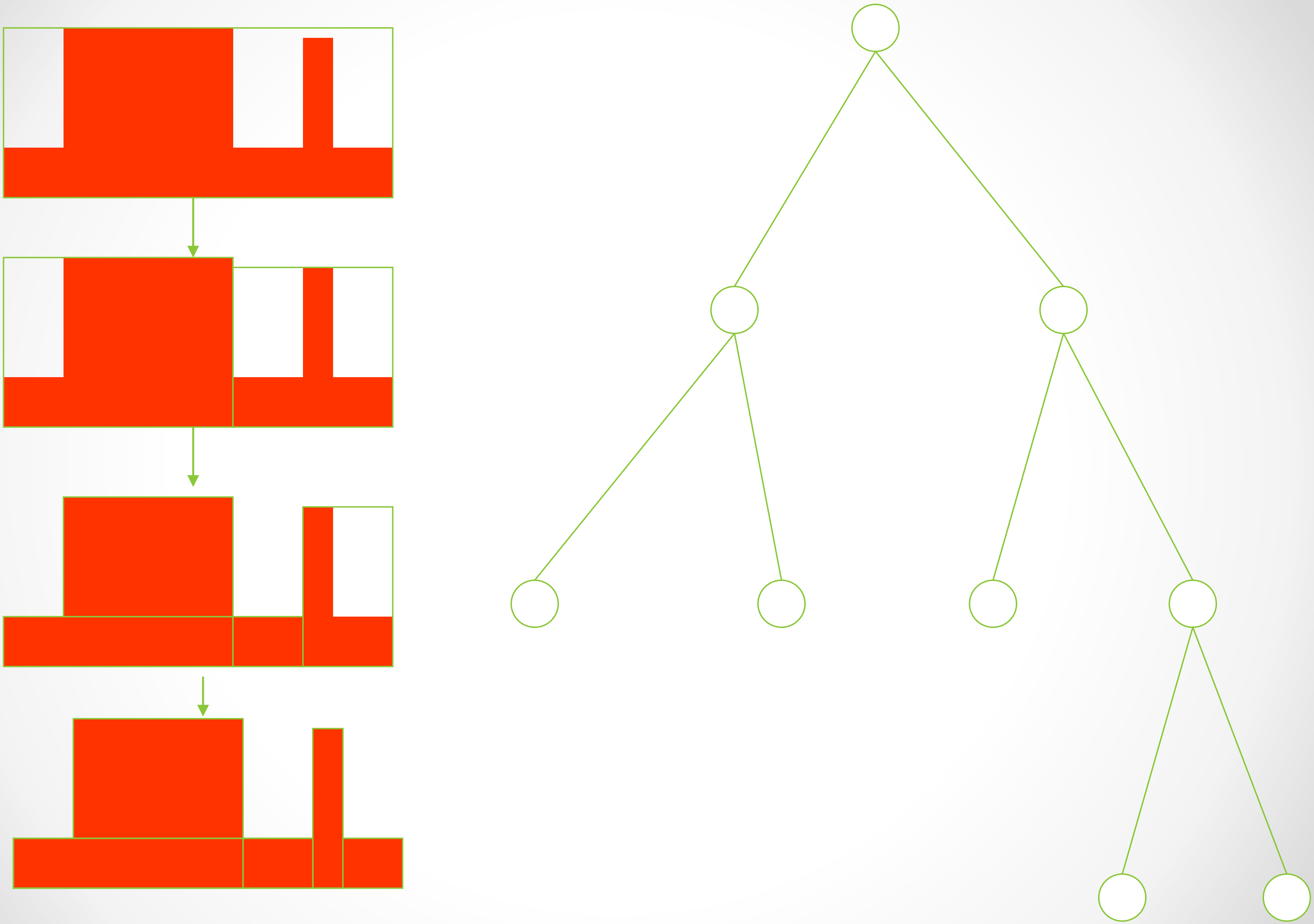
# Create a Hierarchy Tree

- Top down approach
  - Use single BV covering whole object
  - Split object, construct BV for each part
  - Continue recursively until some stopping condition (e.g., high fill efficiency for leaf BV)

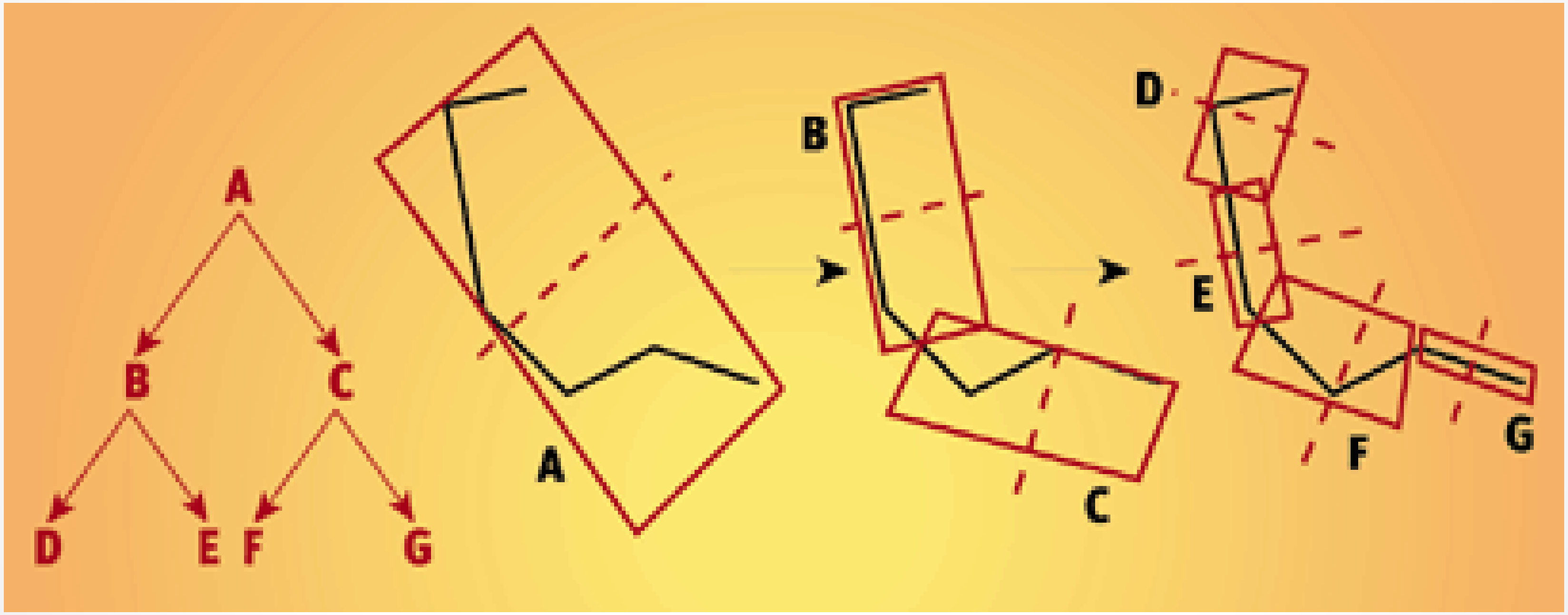
# Binary AABB Tree

- Each node contains all primitives of its subtree
- Leaves contain single primitive



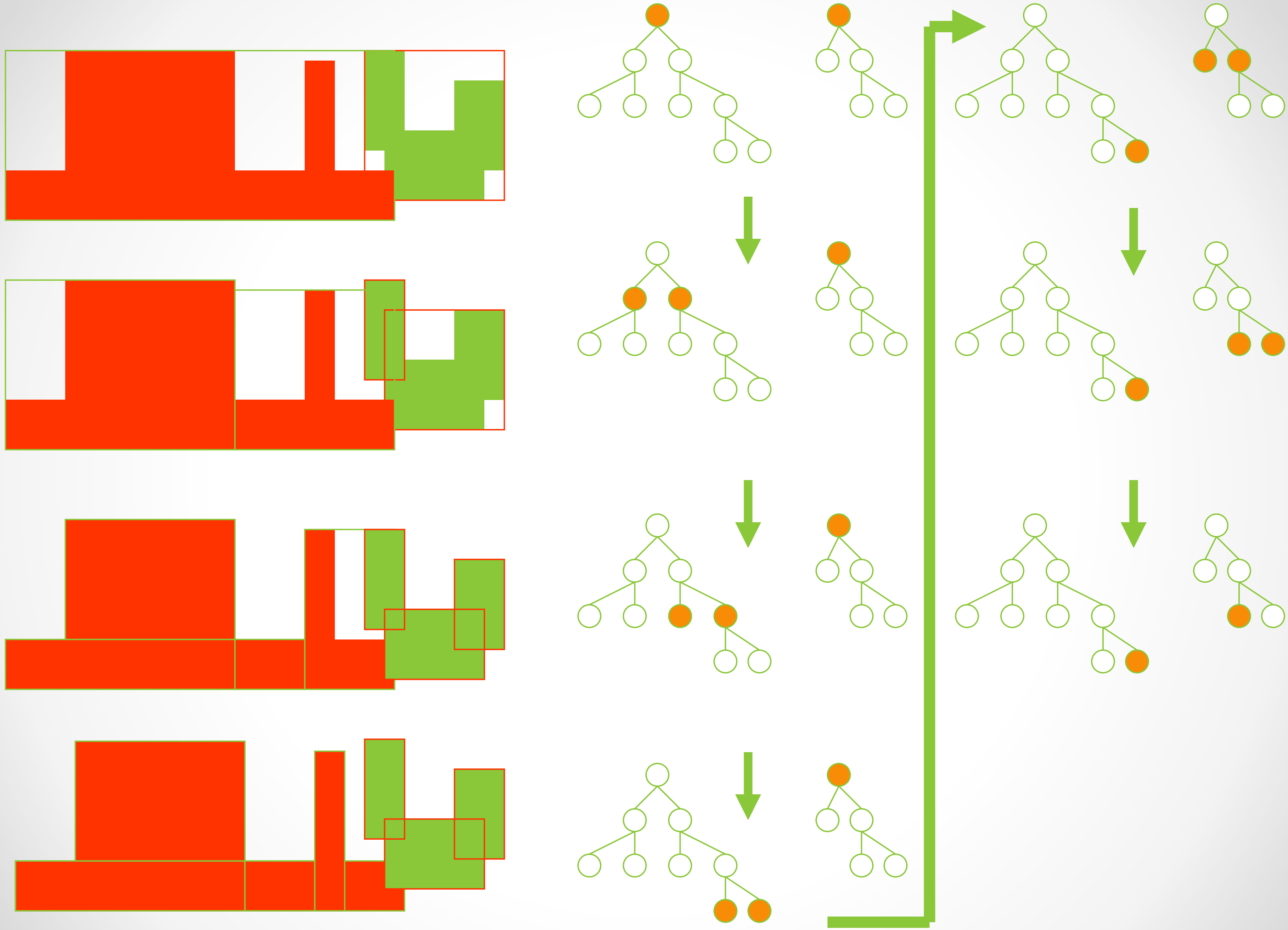


# Binary Tree of OBB



# Hierarchical Collision Detection

- Start by check collision at the root
- If no intersection, return false
- If not, recursively check collision with child nodes
- For two leaf nodes, check exact collision if two BVs intersect



# Overview

- Bounding volumes
- Hierarchy
- **Multiple Objects**

# Basic Idea

- Virtual environment usually consists of more than 2 objects. Pairwise detailed collision between all objects is too slow.

- Solution again:
  - Exclude non colliding objects
  - Check collision between remaining objects

# Grid-based Method

- Create 3d grid volume overlay
- Only check collision between objects sharing at least one cell