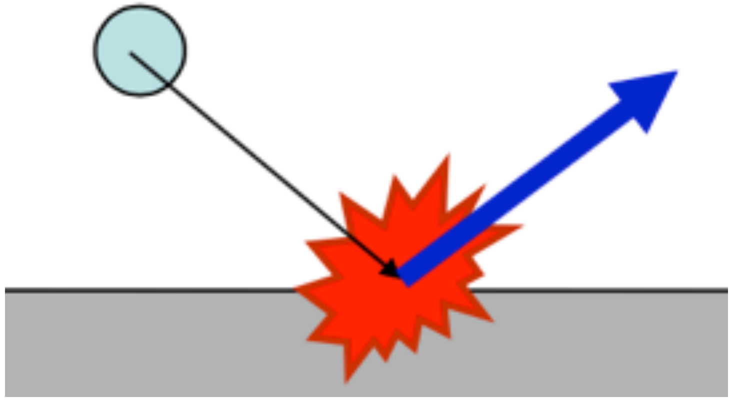
Collision Detection

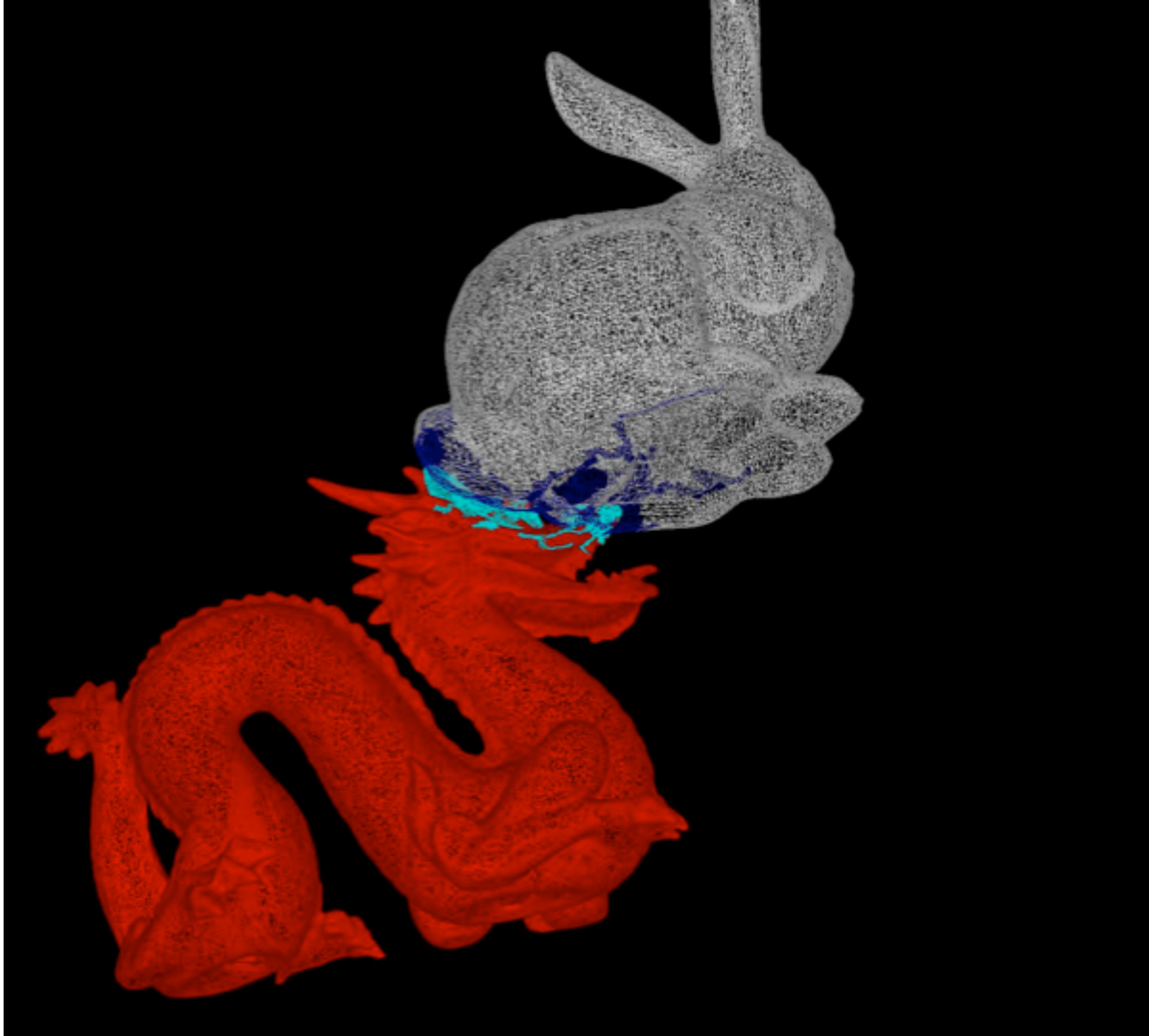
[Ericson: Real-Time Collision Detection]

Collisions

- Detection

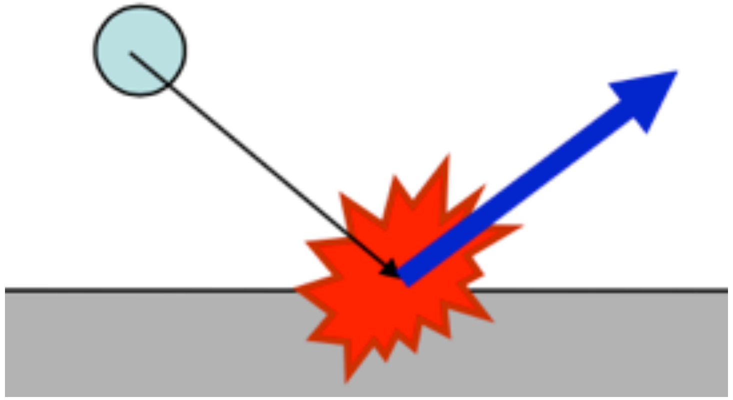
- Response

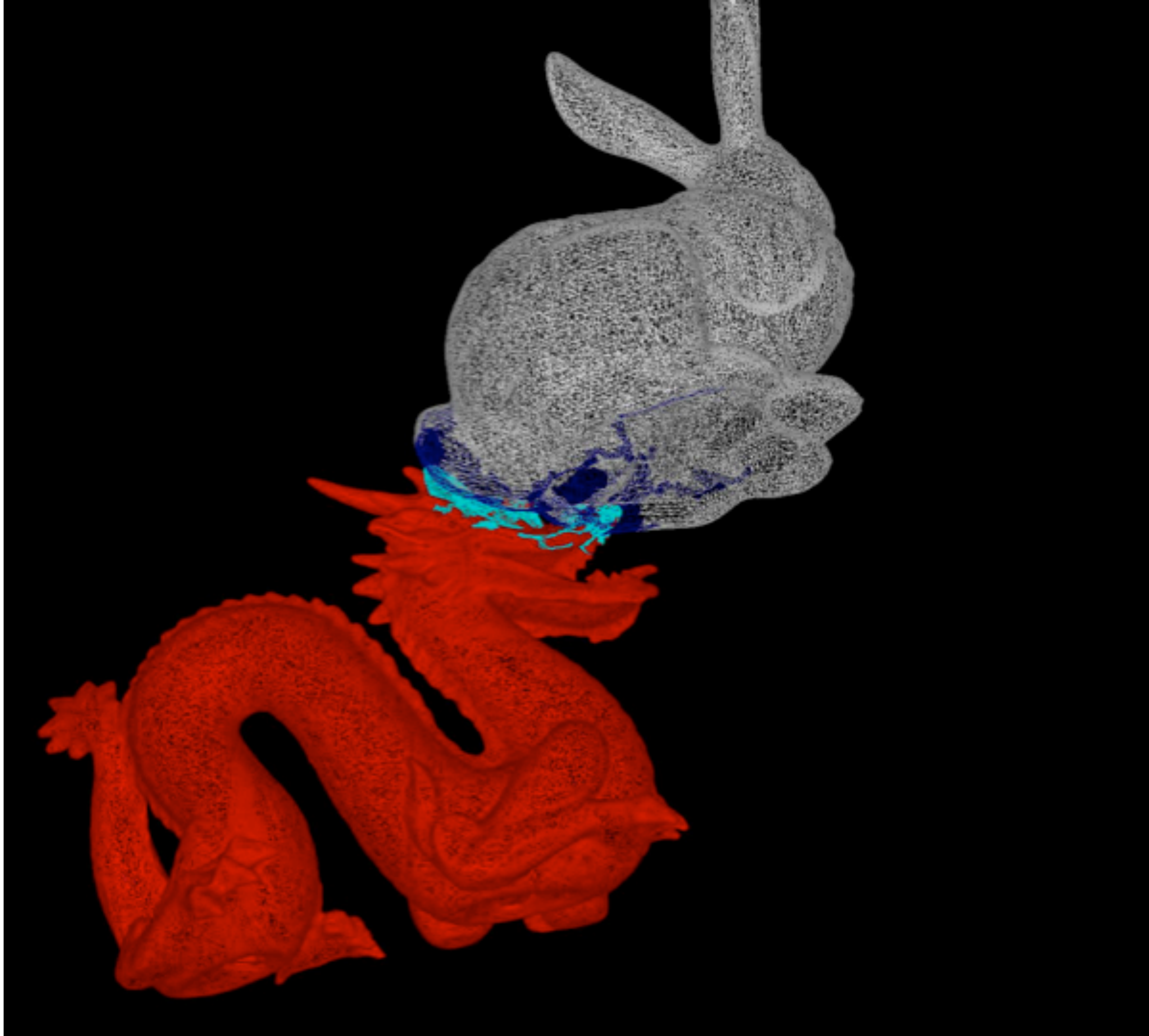




Collisions

- Detection
  - Narrow-phase
  - Broad-phase
- Response





Narrow-Phase

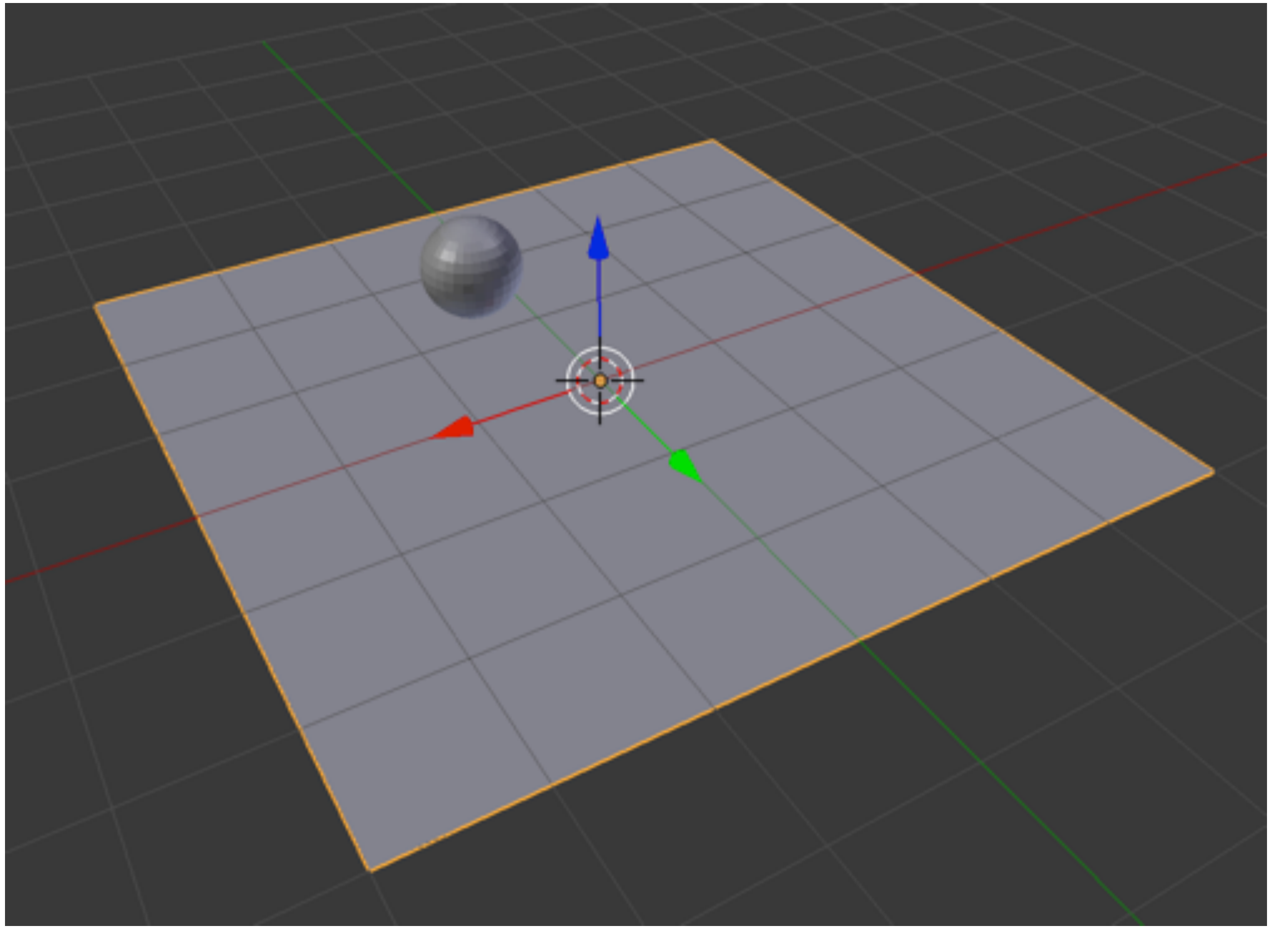
- Pair-wise tests
    - **Primitive tests**
    - Bounding volumes

Primitive tests

- Point-Plane
- Line-Triangle
- Triangle-Triangle
- Polygon-Point
- Many others…

Point-Plane

- Simple case: check against the XY plane
  - Z <= 0 → collision
  - Z > 0 → ok



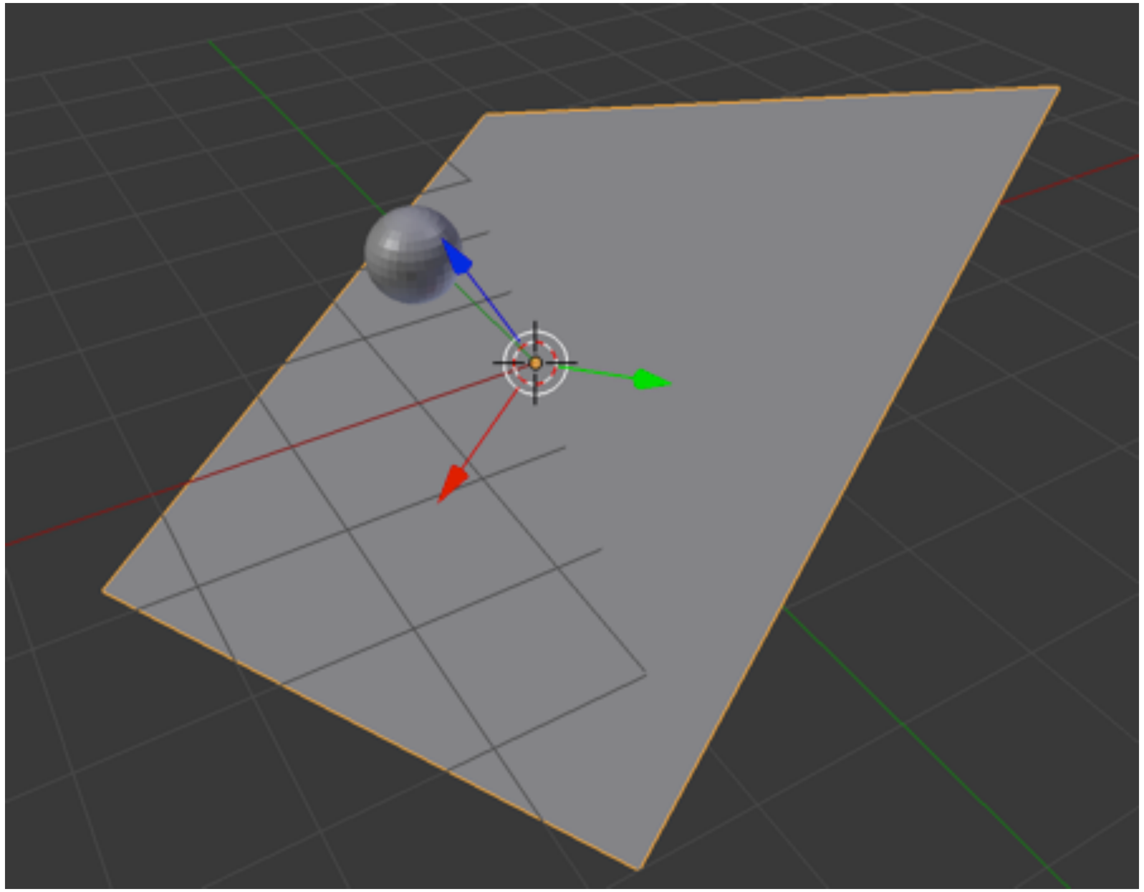
Particle collision detection

- General case: transform particle to the plane's local space

$$x_{local} = M^{-1} x_{world}$$

- Then check its z-coord

$$\left(x_{local}\right)_z < 0\,?$$



Triangle-Triangle

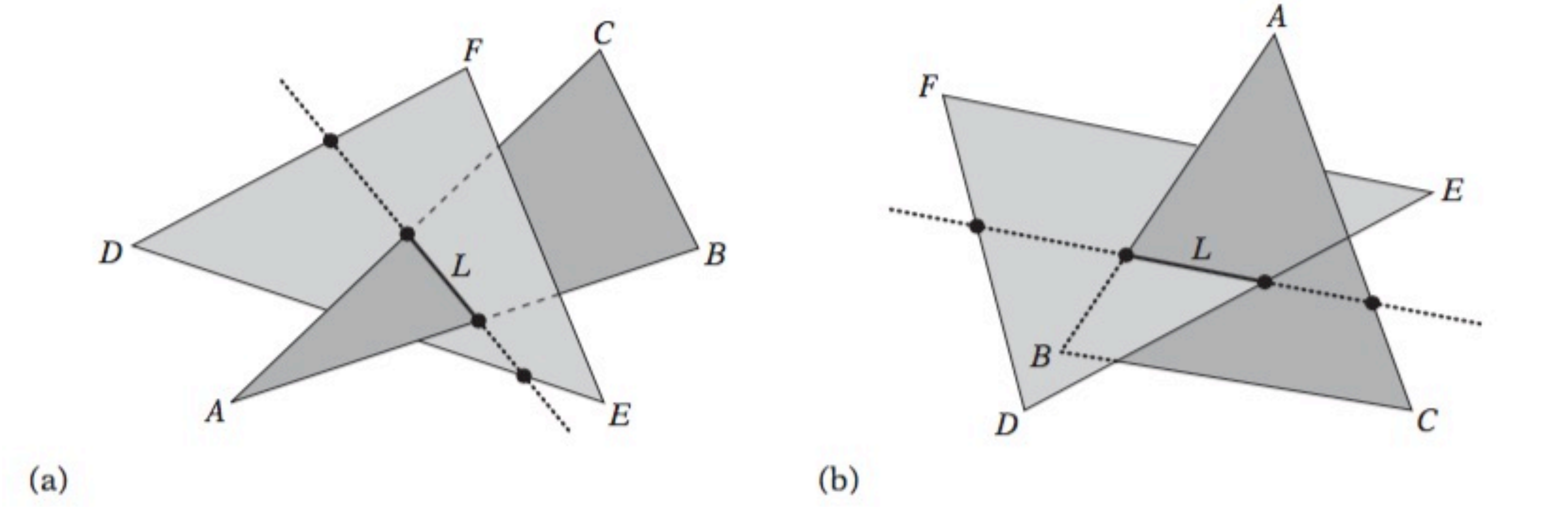


Figure 5.19 In the general case, two triangles intersect (a) when two edges of one triangle pierce the interior of the other or (b) when one edge from each pierces the interior of the other.

[Ericson: Real-Time Collision Detection]

Narrow-Phase

- Pair-wise tests
    - Primitive tests
    - **Bounding volumes**

Bounding volumes

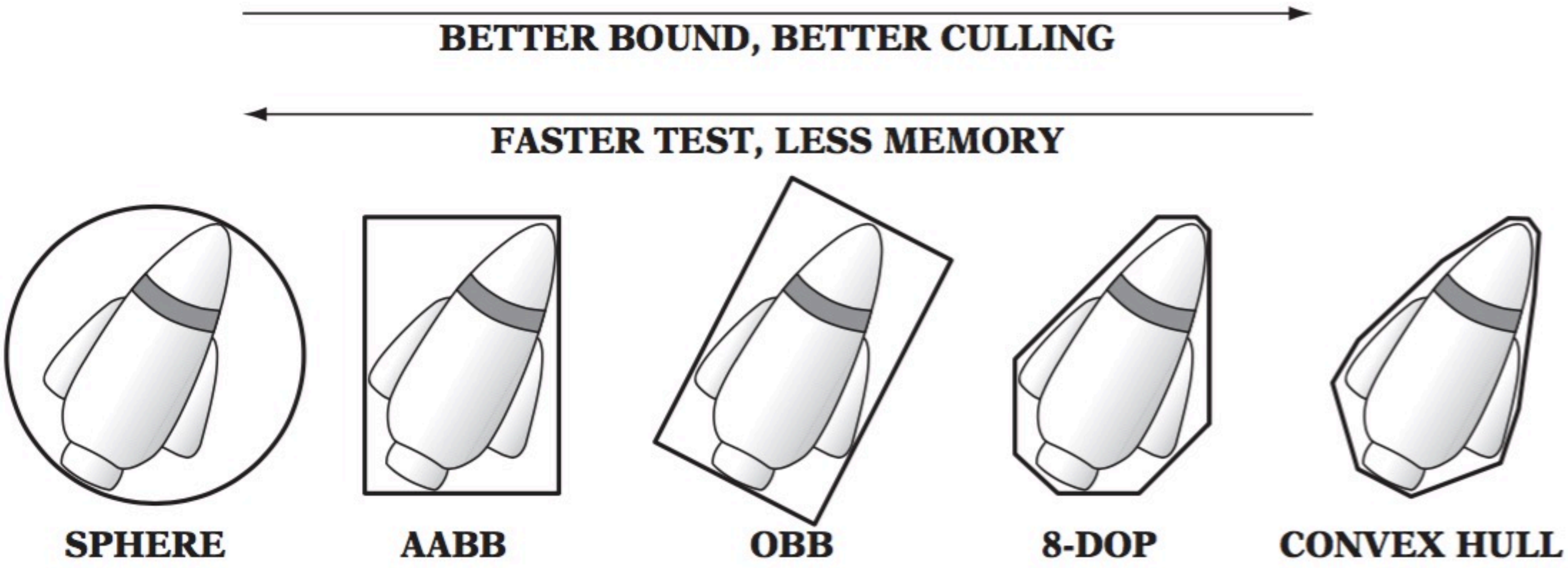
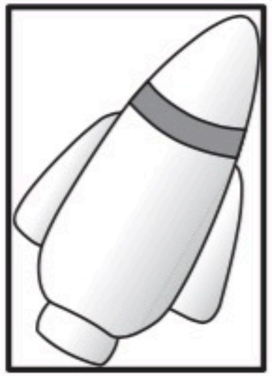


Figure 4.2 Types of bounding volumes: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull.

[Ericson: Real-Time Collision Detection]

Axis Aligned Bounding Boxes



AABB

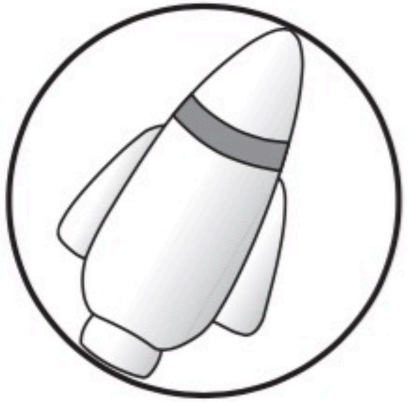
- Easy to construct
- Easy to test

```
int TestAABBAABB(AABB a, AABB b)
{
    // Exit with no intersection if separated along an axis
    if (a.max[0] < b.min[0] || a.min[0] > b.max[0]) return 0;
    if (a.max[1] < b.min[1] || a.min[1] > b.max[1]) return 0;
    if (a.max[2] < b.min[2] || a.min[2] > b.max[2]) return 0;
    // Overlapping on all axes means AABBs are intersecting
    return 1;
}
```

[Ericson: Real-Time Collision Detection]

Spheres

- Almost as easy to construct
- Easy to test

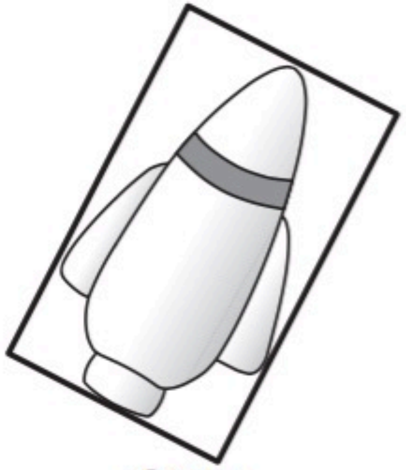



```
int TestSphereSphere(Sphere a, Sphere b)
{
    // Calculate squared distance between centers
    Vector d = a.c - b.c;
    float dist2 = Dot(d, d);
    // Spheres intersect if squared distance is less than squared sum of radii
    float radiusSum = a.r + b.r;
    return dist2 <= radiusSum * radiusSum;
}
```

[Ericson: Real-Time Collision Detection]

Oriented Bounding Boxes

- Construction is non-trivial



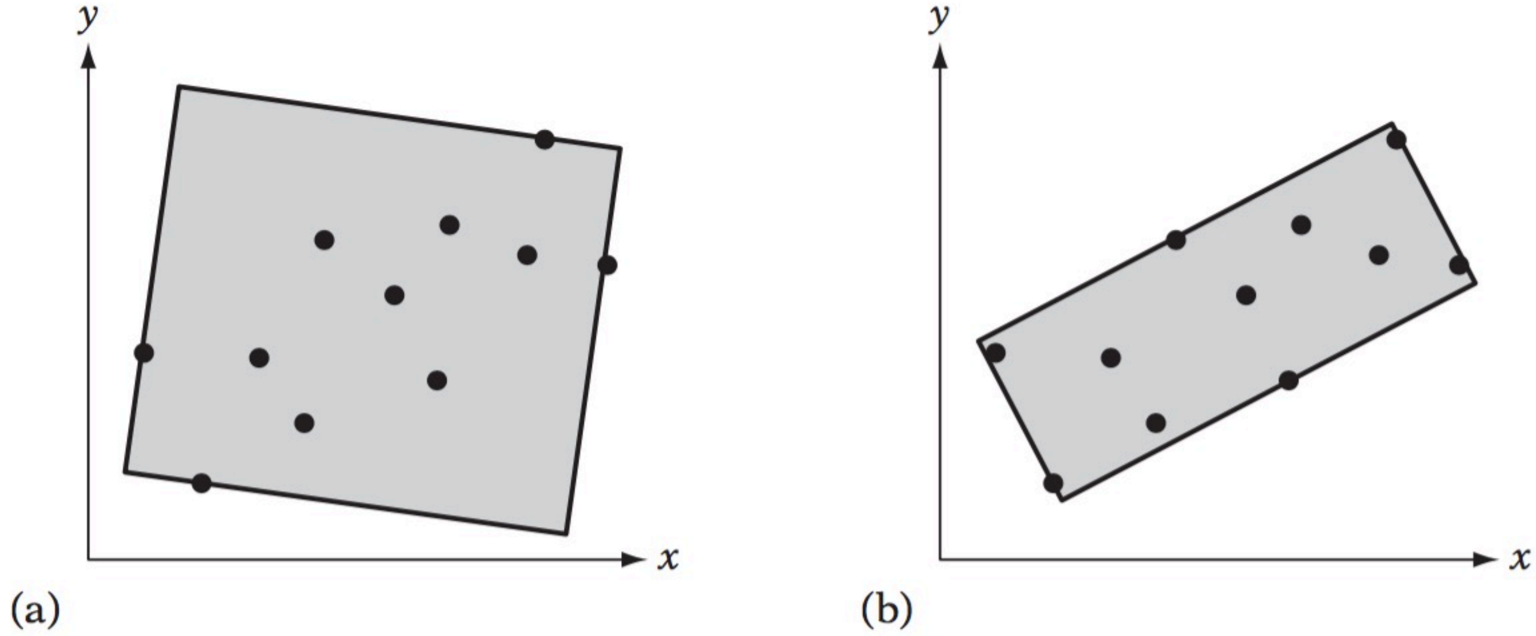
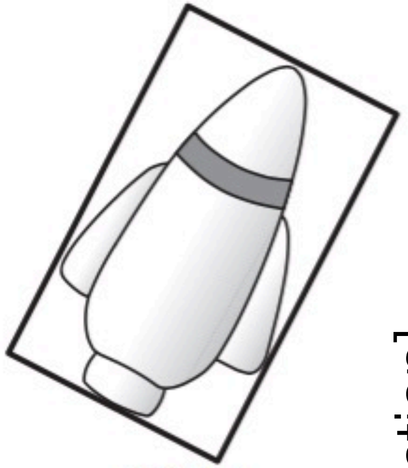


Figure 4.10 (a) A poorly aligned and (b) a well-aligned OBB.

[Ericson: Real-Time Collision Detection]

Oriented Bounding Boxes



OBB

- Testing is also non-trivial
  - Transform Object B into Object A’s coordinate space
  - Apply “separating axis test”

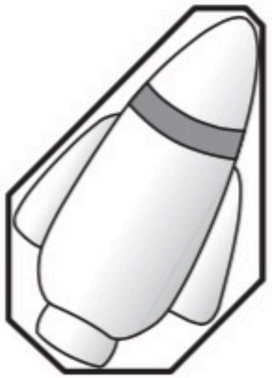
```
int TestOBBOBB(OBB &a, OBB &b)
{
    float ra, rb;
    Matrix33 R, AbsR;

    // Compute rotation matrix expressing b in a's coordinate frame
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            R[i][j] = Dot(a.u[i], b.u[j]);
    // Compute translation vector t
    Vector t = b.c - a.c;
    // Bring translation into a's coordinate frame
    t = Vector(Dot(t, a.u[0]), Dot(t, a.u[2]), Dot(t, a.u[2]));

    // Compute common subexpressions. Add in an epsilon term to
    // counteract arithmetic errors when two edges are parallel and
    // their cross product is (near) null (see text for details)
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            AbsR[i][j] = Abs(R[i][j]) + EPSILON;

    // Test axes L = A0, L = A1, L = A2
    for (int i = 0; i < 3; i++) {
        ra = a.e[i];
        rb = b.e[0] * AbsR[i][0] + b.e[1] * AbsR[i][1] + b.e[2] * AbsR[i][2];
        if (Abs(t[i]) > ra + rb) return 0;
    }
```


```
    // Test axes L = A0, L = A1, L = A2
    for (int i = 0; i < 3; i++) {
        ra = a.e[i];
        rb = b.e[0] * AbsR[i][0] + b.e[1] * AbsR[i][1] + b.e[2] * AbsR[i][2];
        if (Abs(t[i]) > ra + rb) return 0;
    }
```


```
    // Test axes L = B0, L = B1, L = B2
    for (int i = 0; i < 3; i++) {
        ra = a.e[0] * AbsR[0][i] + a.e[1] * AbsR[1][i] + a.e[2] * AbsR[2][i];
        rb = b.e[i];
        if (Abs(t[0] * R[0][i] + t[1] * R[1][i] + t[2] * R[2][i]) > ra + rb) return 0;
    }
    // Test axis L = A0 x B0
    ra = a.e[1] * AbsR[2][0] + a.e[2] * AbsR[1][0];
    rb = b.e[1] * AbsR[0][2] + b.e[2] * AbsR[0][1];
    if (Abs(t[2] * R[1][0] - t[1] * R[2][0]) > ra + rb) return 0;

    // Test axis L = A0 x B1
    ra = a.e[1] * AbsR[2][1] + a.e[2] * AbsR[1][1];
    rb = b.e[0] * AbsR[0][2] + b.e[2] * AbsR[0][0];
    if (Abs(t[2] * R[1][1] - t[1] * R[2][1]) > ra + rb) return 0;

    // Test axis L = A0 x B2
    ra = a.e[1] * AbsR[2][2] + a.e[2] * AbsR[1][2];
    rb = b.e[0] * AbsR[0][1] + b.e[1] * AbsR[0][0];
    if (Abs(t[2] * R[1][2] - t[1] * R[2][2]) > ra + rb) return 0;

    // Test axis L = A1 x B0
    ra = a.e[0] * AbsR[2][0] + a.e[2] * AbsR[0][0];
    rb = b.e[1] * AbsR[1][2] + b.e[2] * AbsR[1][1];
    if (Abs(t[0] * R[2][0] - t[2] * R[0][0]) > ra + rb) return 0;

    // Test axis L = A1 x B1
    ra = a.e[0] * AbsR[2][1] + a.e[2] * AbsR[0][1];
    rb = b.e[0] * AbsR[1][2] + b.e[2] * AbsR[1][0];
    if (Abs(t[0] * R[2][1] - t[2] * R[0][1]) > ra + rb) return 0;

    // Test axis L = A1 x B2
    ra = a.e[0] * AbsR[2][2] + a.e[2] * AbsR[0][2];
    rb = b.e[0] * AbsR[1][1] + b.e[1] * AbsR[1][0];
    if (Abs(t[0] * R[2][2] - t[2] * R[0][2]) > ra + rb) return 0;

    // Test axis L = A2 x B0
    ra = a.e[0] * AbsR[1][0] + a.e[1] * AbsR[0][0];
    rb = b.e[1] * AbsR[2][2] + b.e[2] * AbsR[2][1];
    if (Abs(t[1] * R[0][0] - t[0] * R[1][0]) > ra + rb) return 0;

    // Test axis L = A2 x B1
    ra = a.e[0] * AbsR[1][1] + a.e[1] * AbsR[0][1];
    rb = b.e[0] * AbsR[2][2] + b.e[2] * AbsR[2][0];
    if (Abs(t[1] * R[0][1] - t[0] * R[1][1]) > ra + rb) return 0;

    // Test axis L = A2 x B2
    ra = a.e[0] * AbsR[1][2] + a.e[1] * AbsR[0][2];
    rb = b.e[0] * AbsR[2][1] + b.e[1] * AbsR[2][0];
    if (Abs(t[1] * R[0][2] - t[0] * R[1][2]) > ra + rb) return 0;

    // Since no separating axis is found, the OBBs must be intersecting
    return 1;
}
```

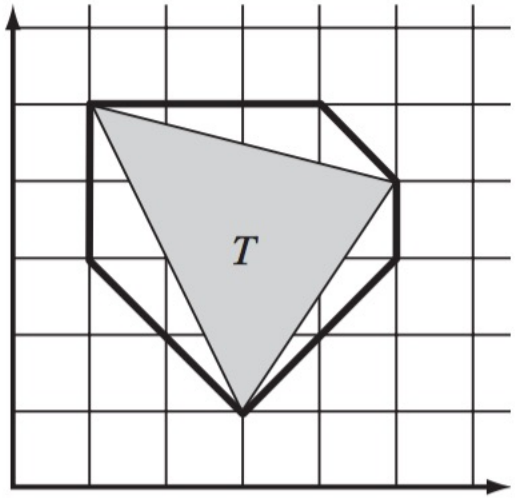
[Ericson: Real-Time Collision Detection]

Discrete-Orientation Polytopes



8-DOP

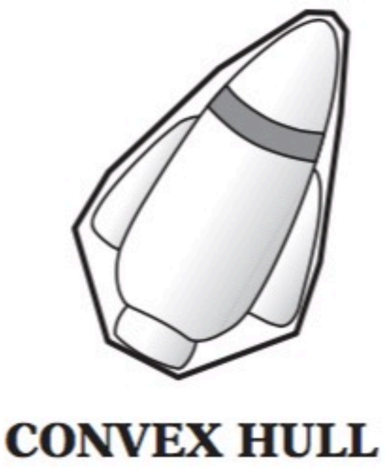
- “k-DOPs”
- Construction: relatively expensive
- Testing: relatively expensive



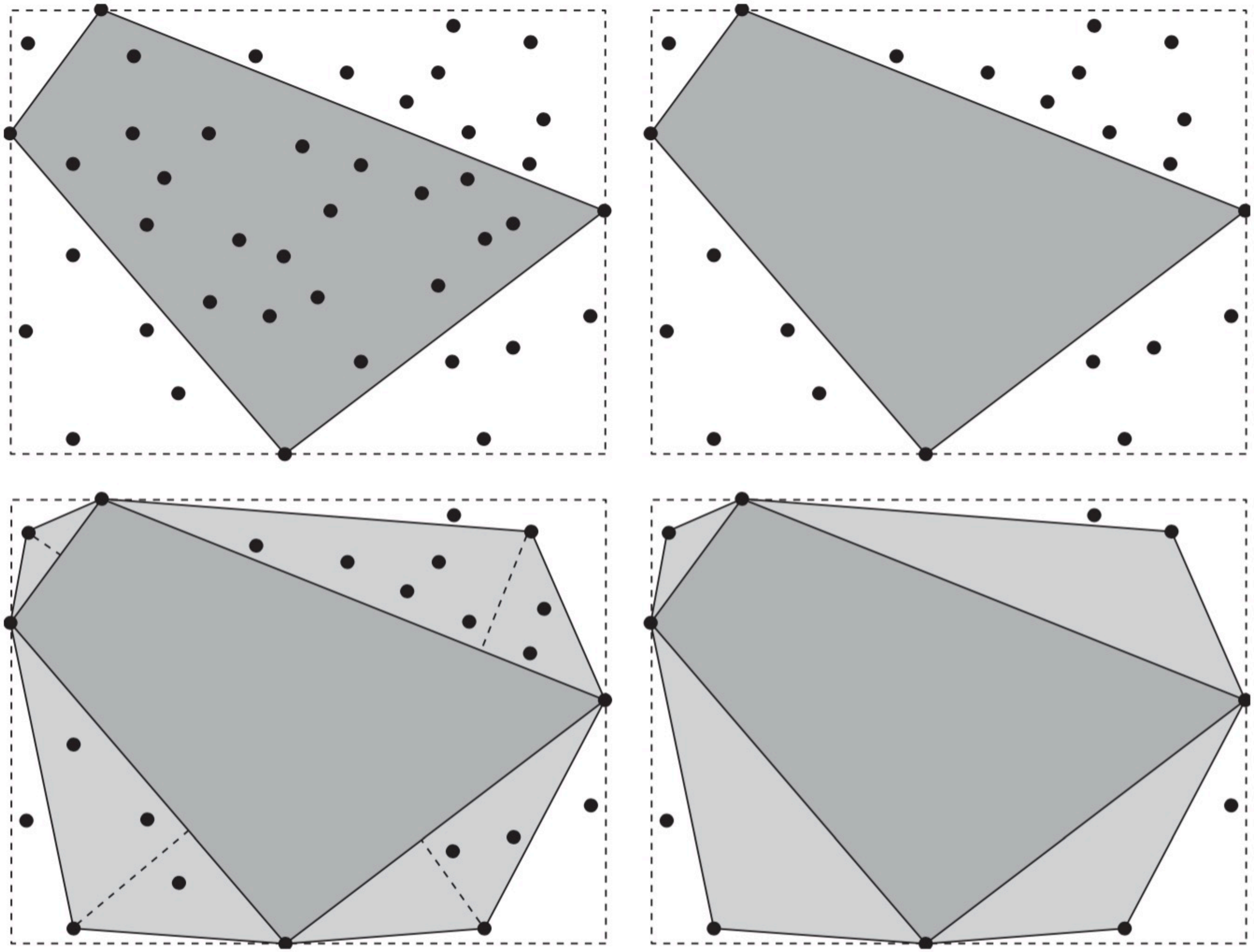
[Ericson: Real-Time Collision Detection]

Convex Hulls

- Construction: relatively expensive
- Testing: relatively expensive



The Quickhull Algorithm



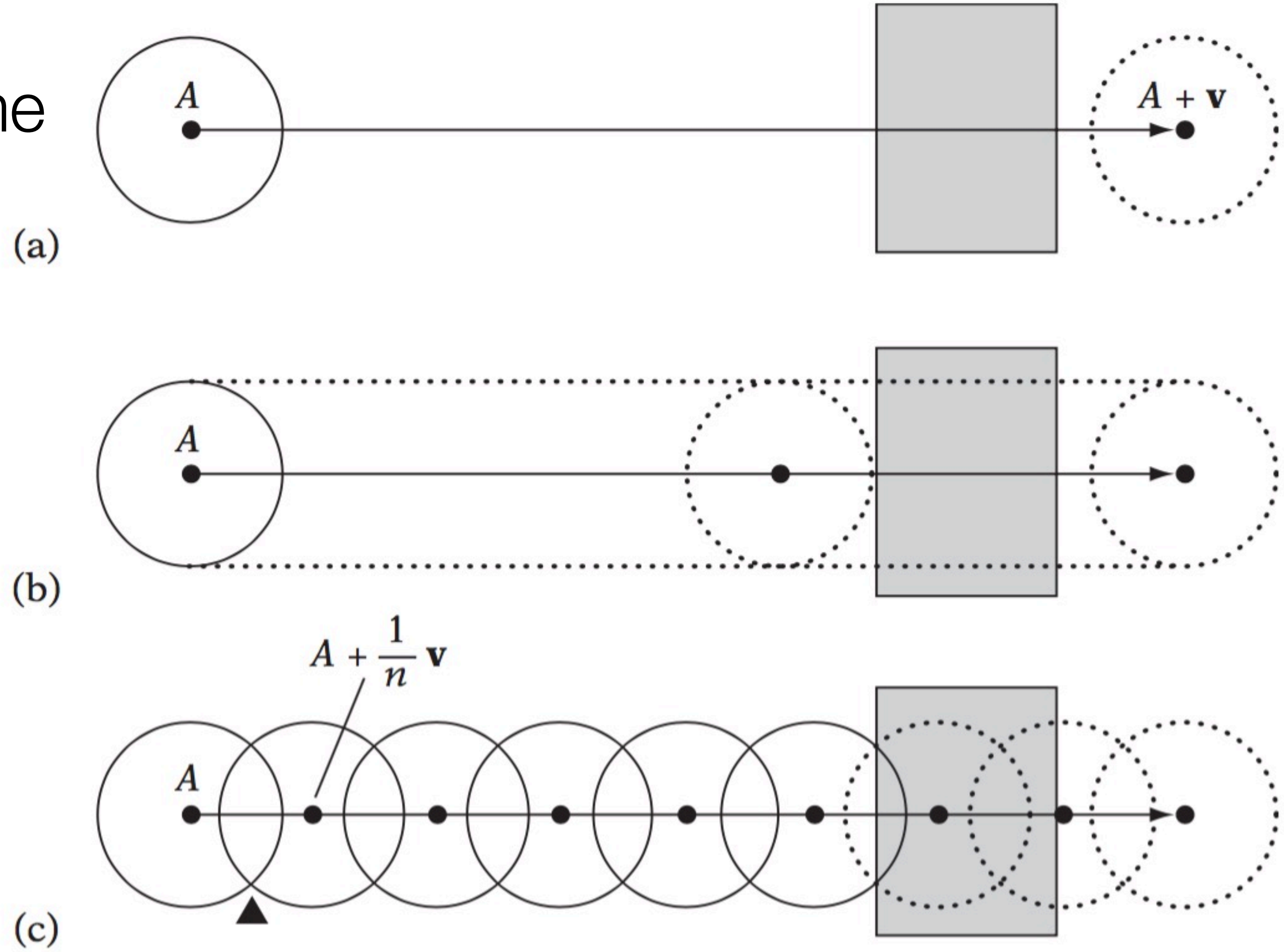
[Ericson: Real-Time Collision Detection]

Narrow-Phase

- Pair-wise tests
    - Primitive tests
    - Bounding volumes
- **Continuous collision detection**

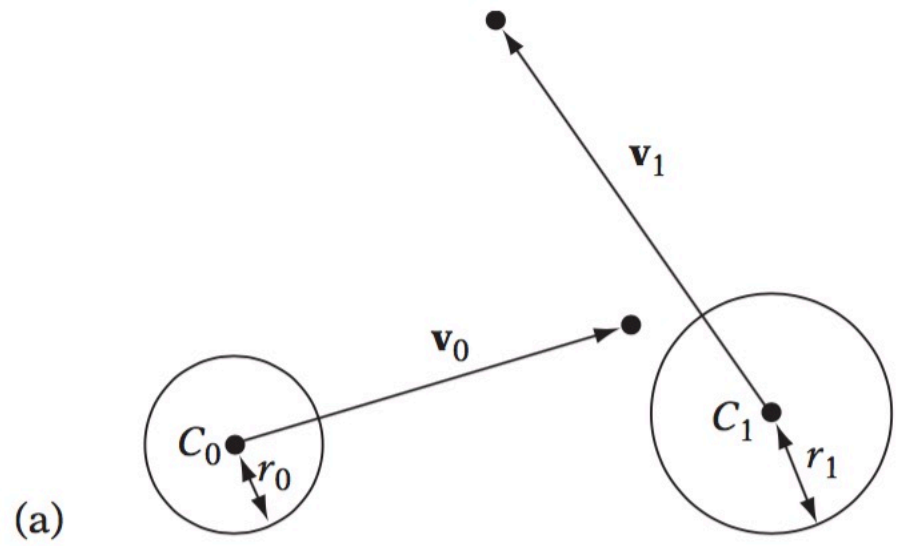
Continuous Collision Detection

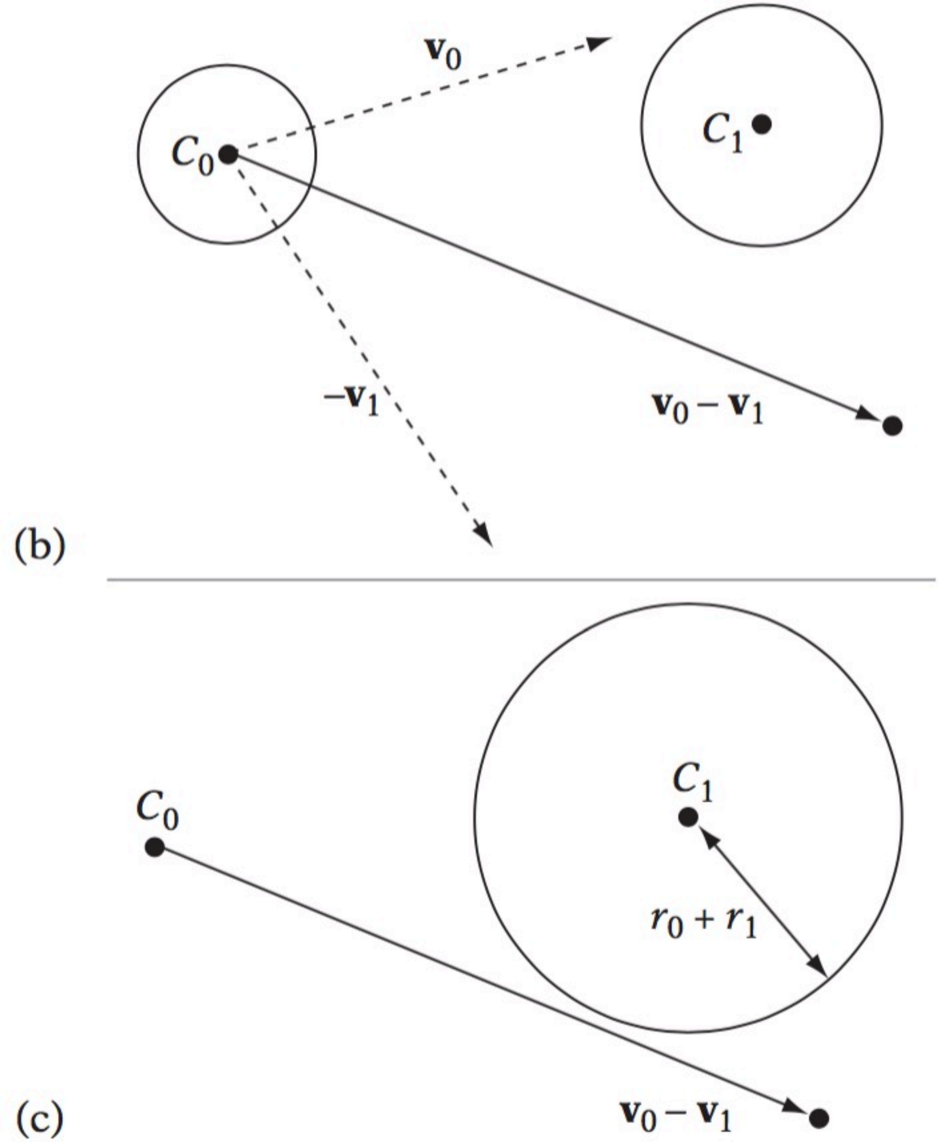
- Swept-volume in space-time



[Ericson: Real-Time Collision Detection]

Transformation to ray-sphere test





[Ericson: Real-Time Collision Detection]

Broad-Phase

- Imagine we have n objects. Can we test all pairwise intersections?
  - Quadratic cost $O(n^2)$!

- Simple optimization: separate static objects
  - But still $O(\text{static} \times \text{dynamic} + \text{dynamic}^2)$

Broad-Phase

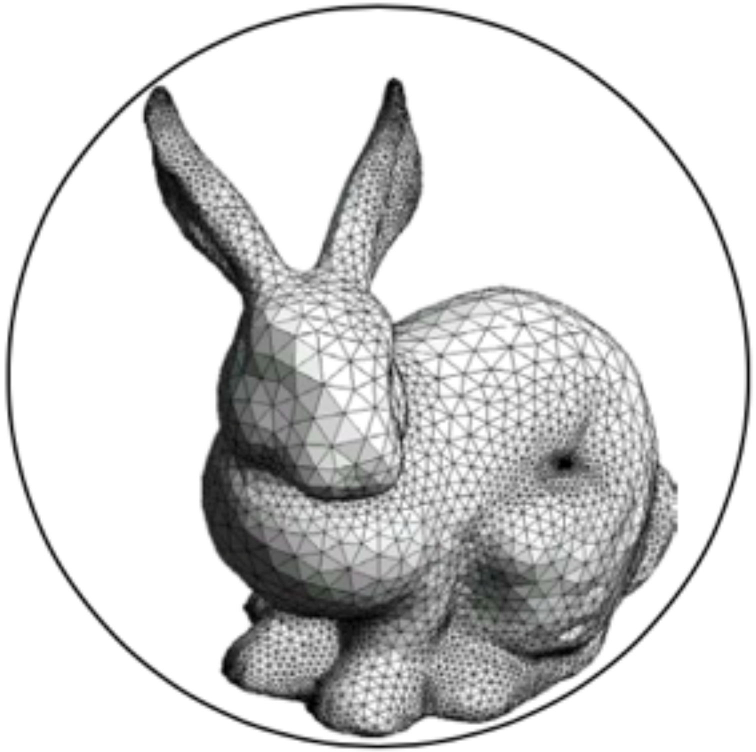
- Bounding Volume hierarchies
- Spatial Partitioning
- Sort and Sweep

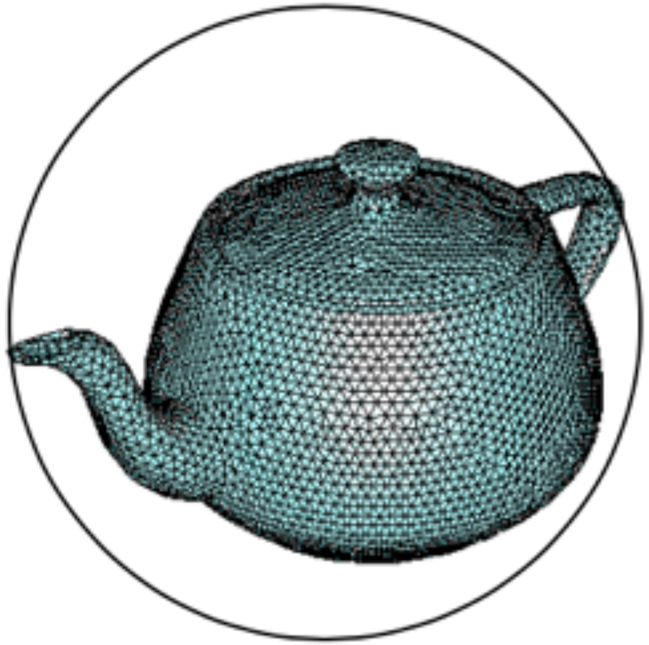
Hierarchical Collision Detection

- Use simpler conservative proxies (e.g. bounding spheres)
- Recursive (hierarchical) test
  - Spend time only for parts of the scene that are close
- Many different versions—we will cover only one

Bounding Spheres

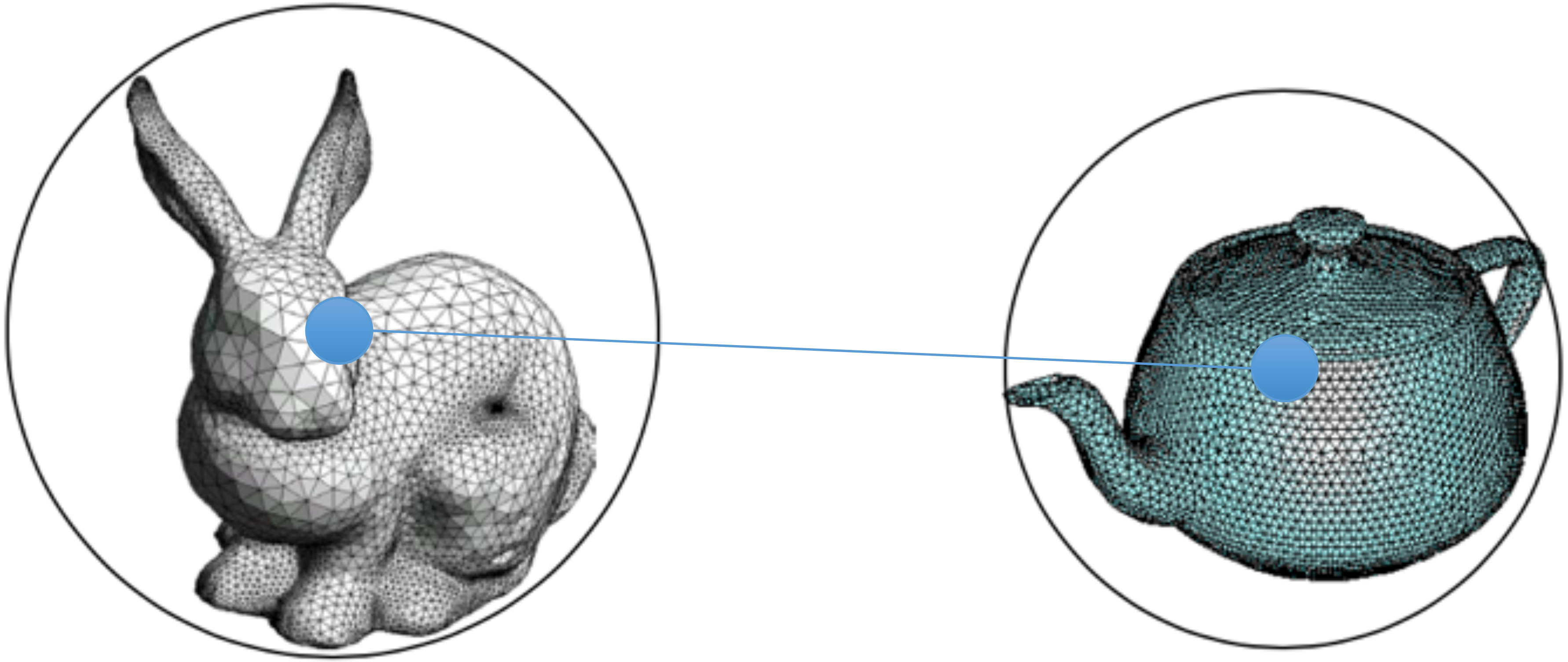
- Place spheres around objects
- If spheres do not intersect, neither do the objects!





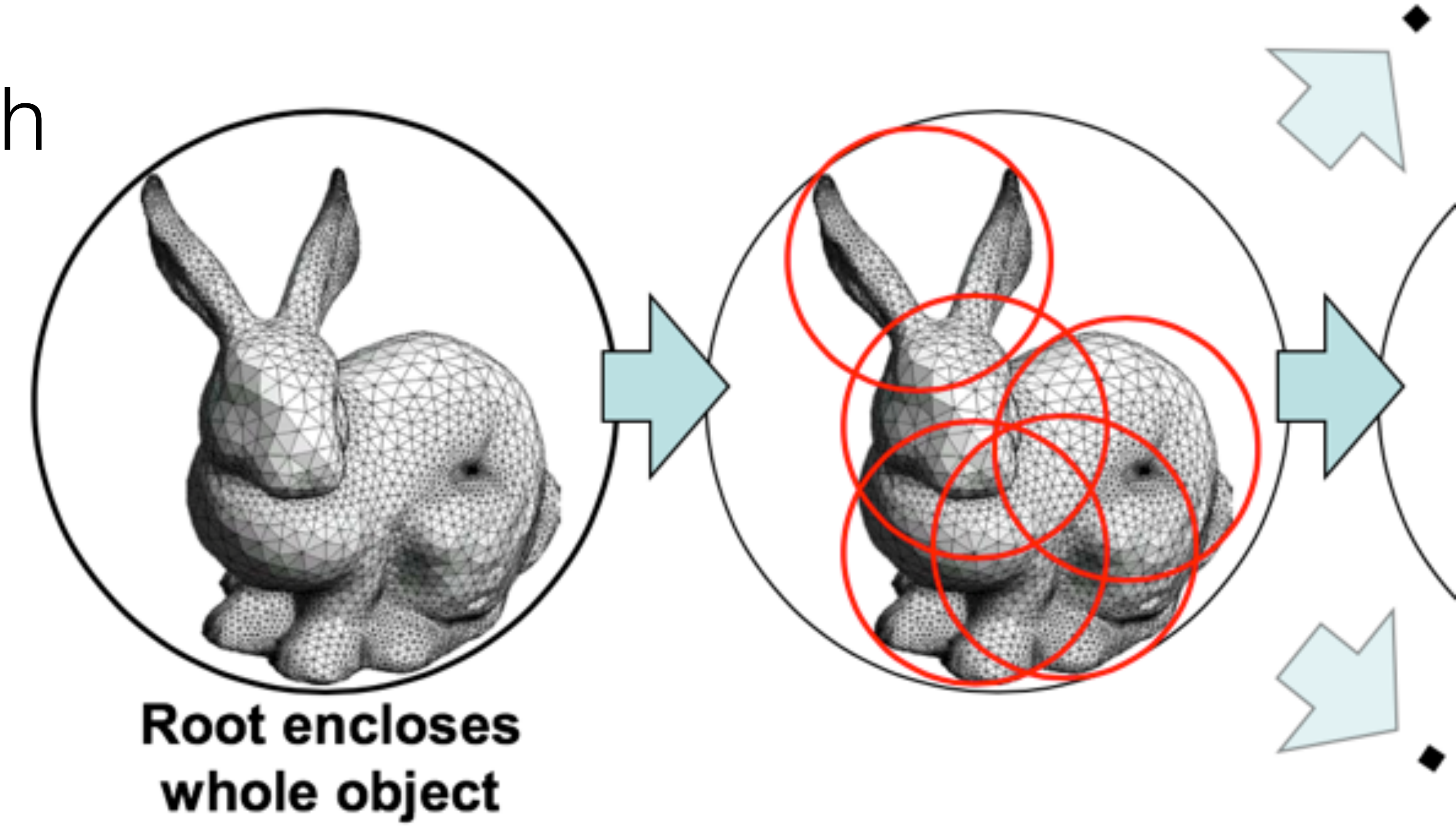
Sphere-Sphere Collision Test

- Two spheres, centers C1 and C2, radii r1 and r2
- Intersect only if ||C1-C2||<r1+r2



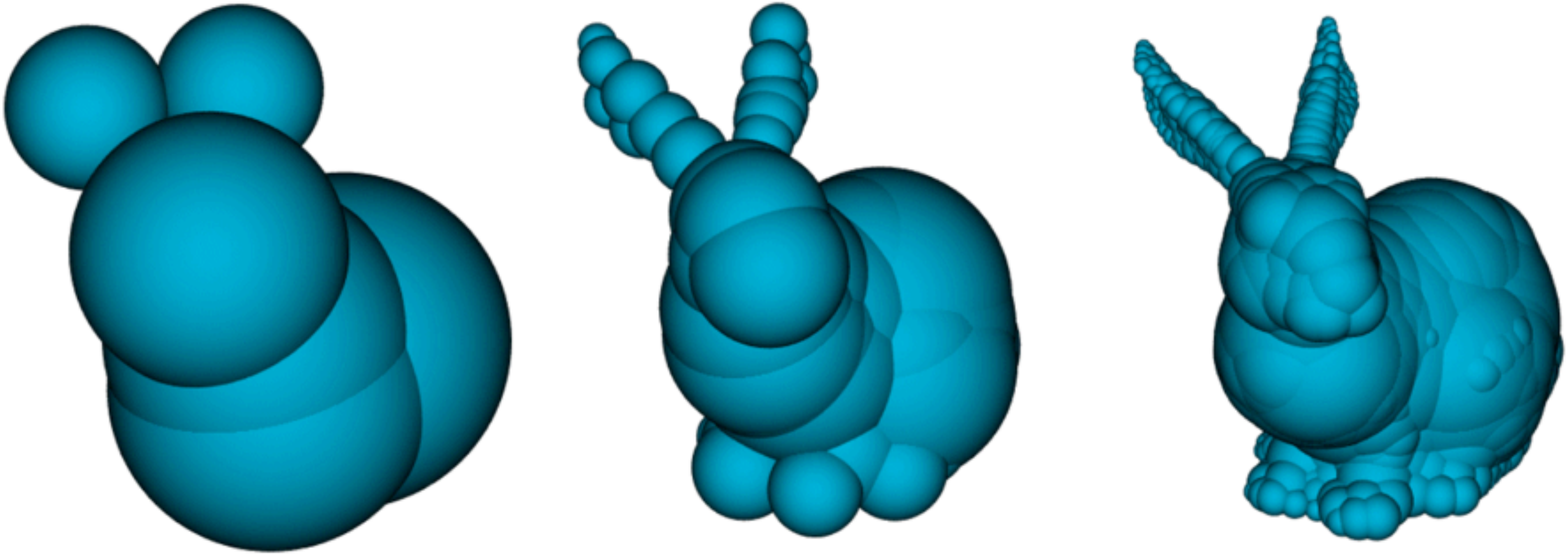
Hierarchical Collision Test

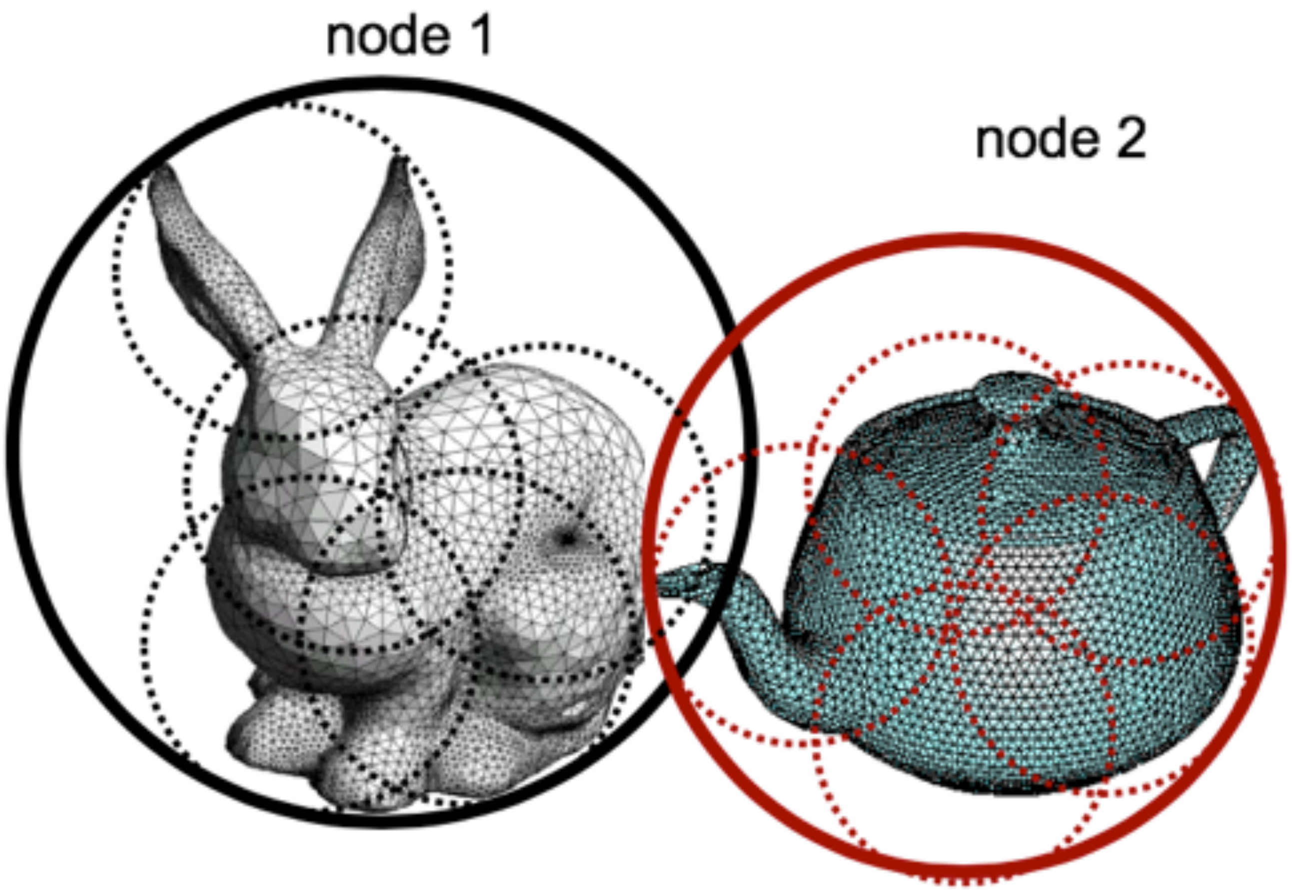
- Hierarchy of bounding spheres
  - Organized in a tree
- Recursive test with early pruning

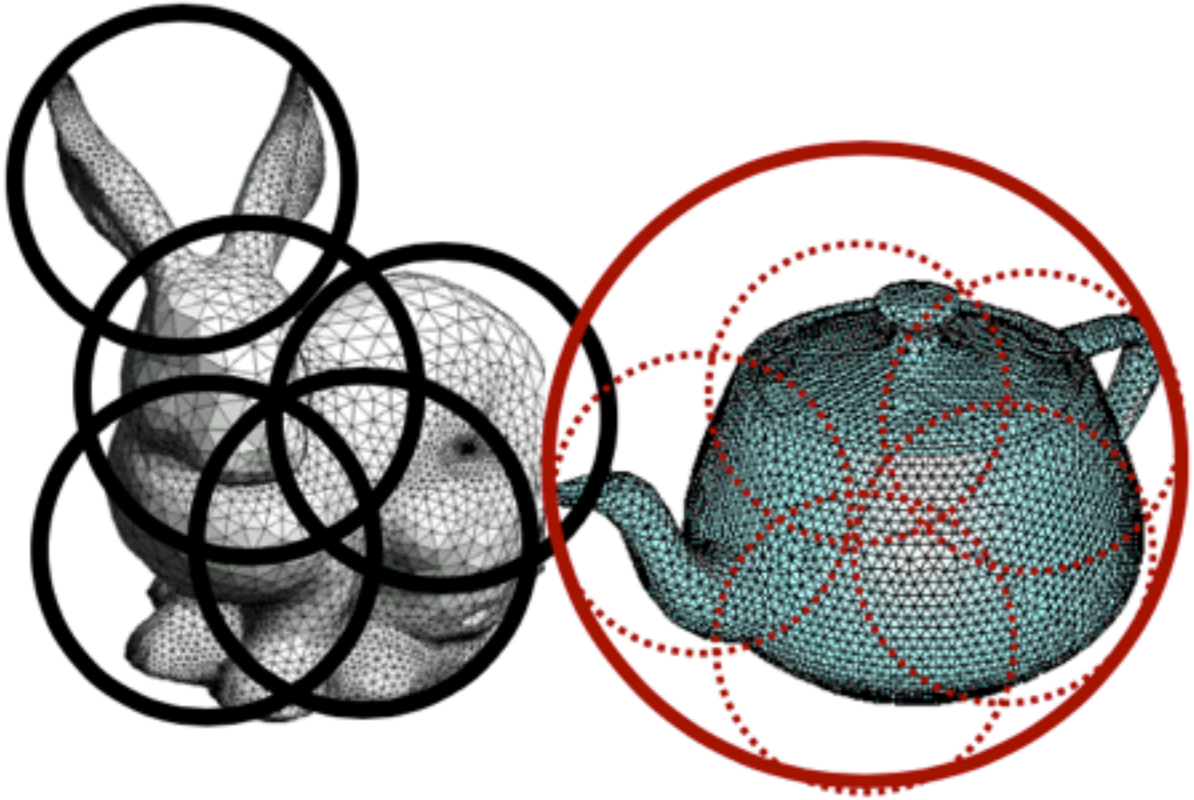


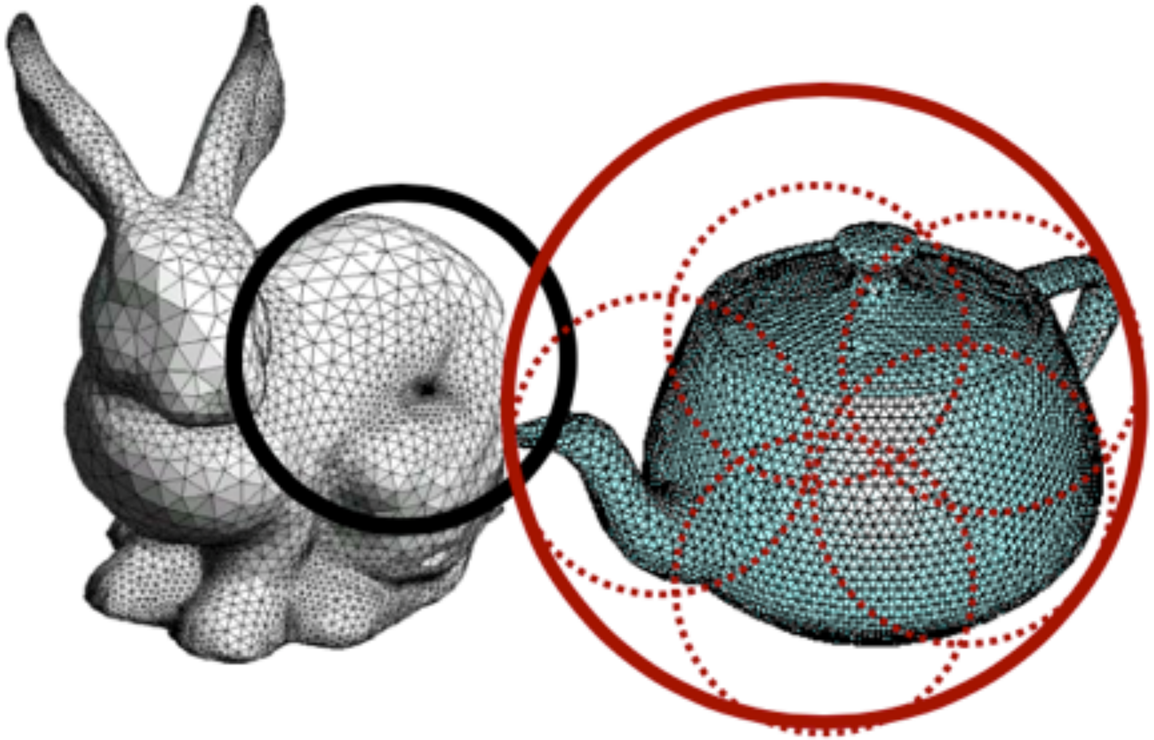
Example of hierarchy

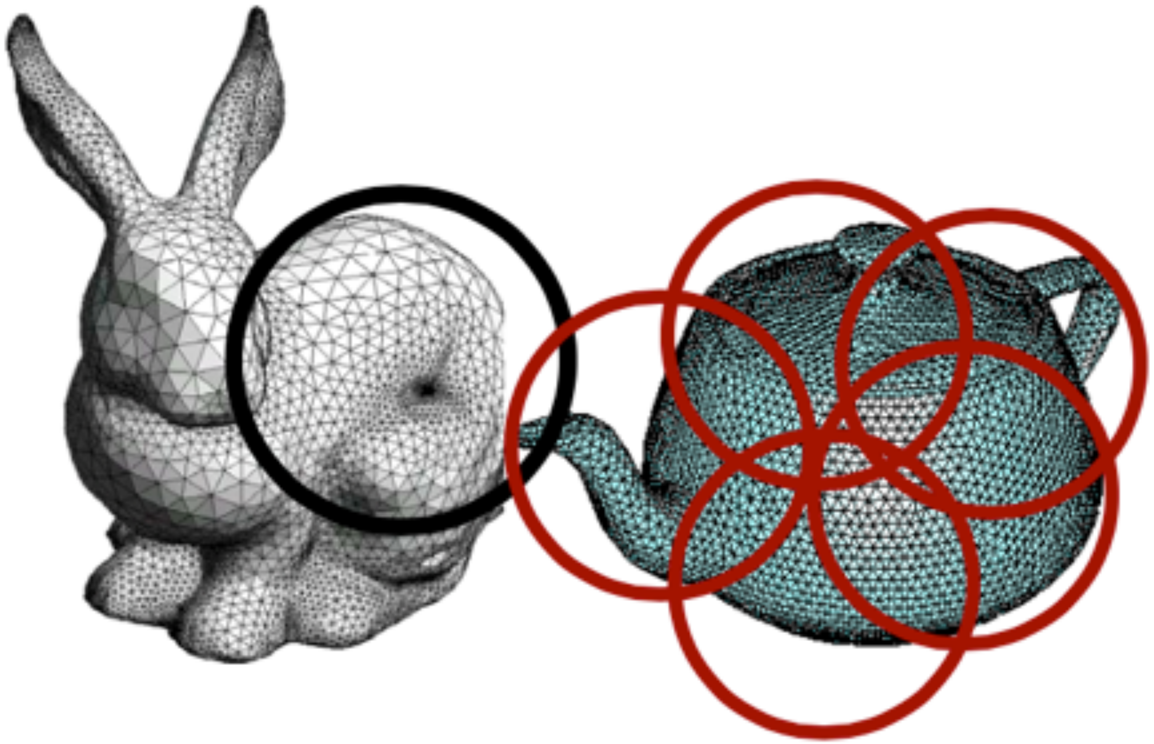
- http://isg.cs.tcd.ie/spheretree/

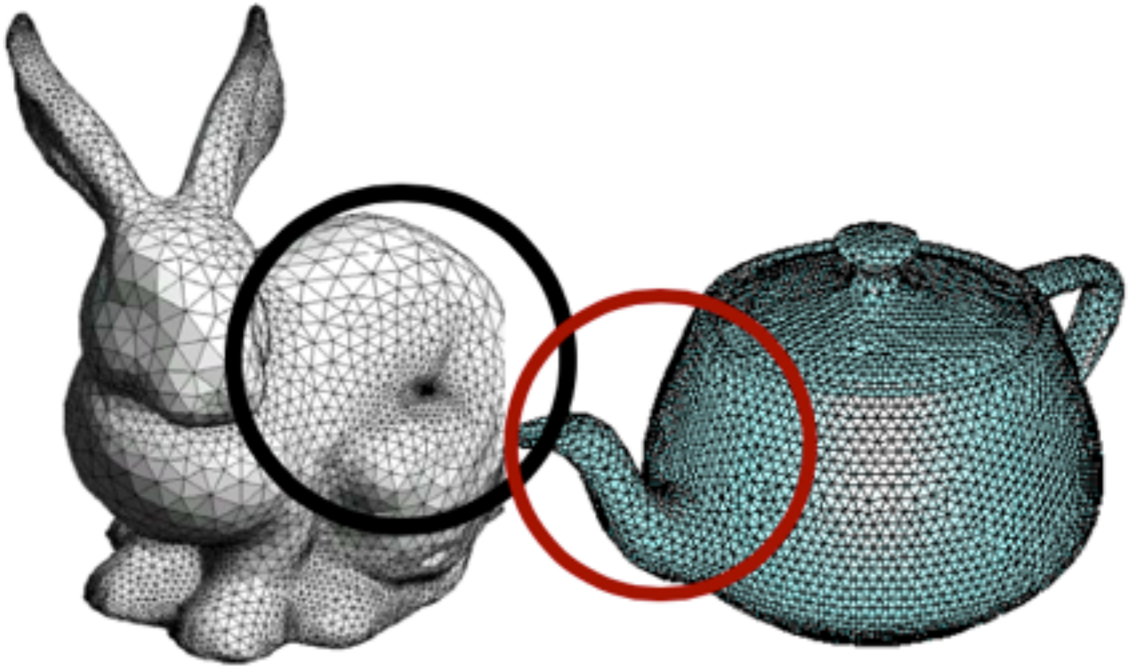


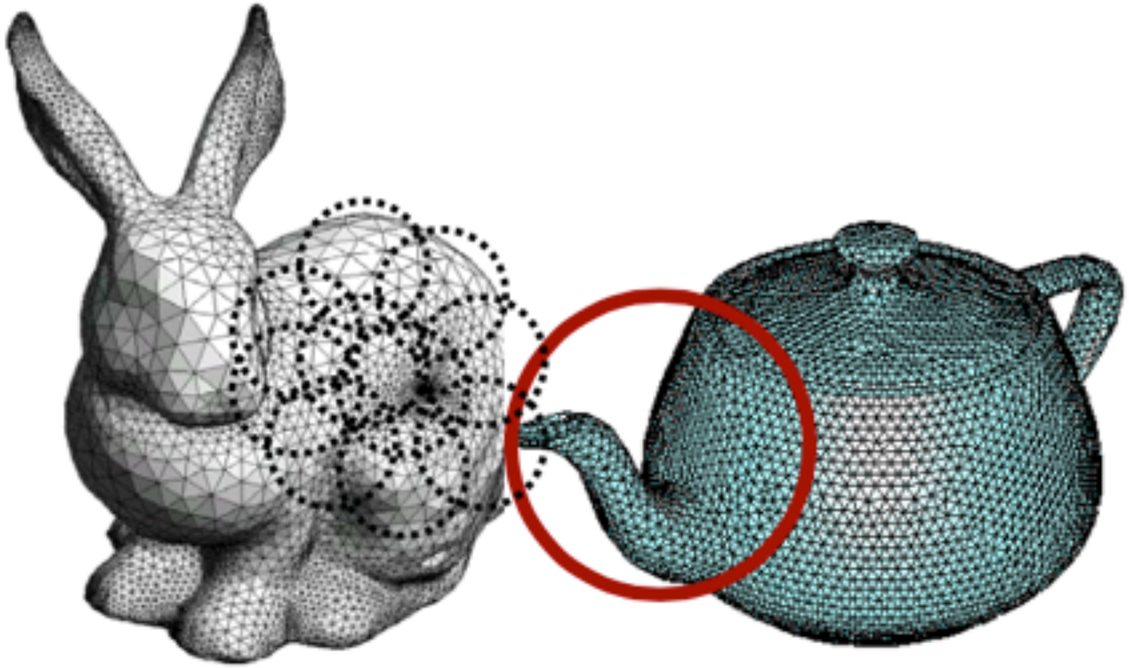

node 1
node 2





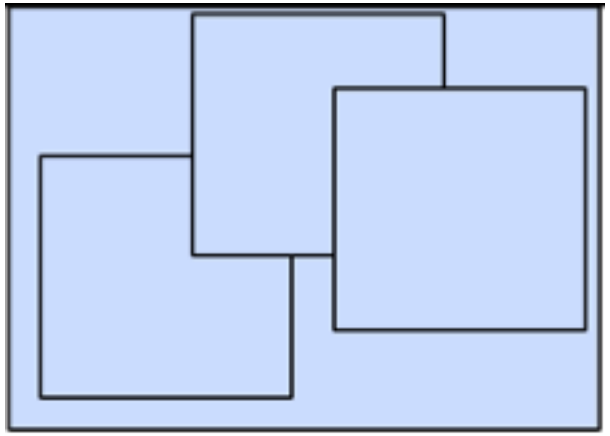




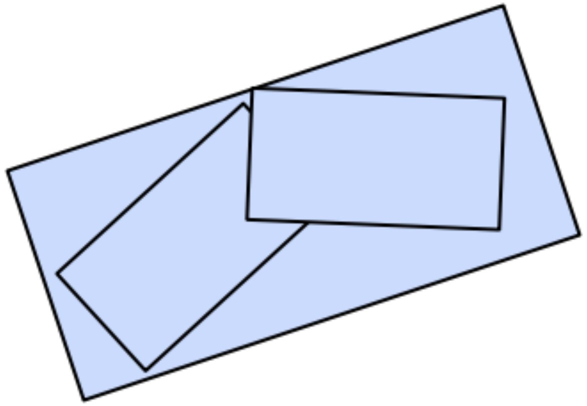


Other options

- Axis Aligned Bounding Boxes – “R-Trees”



- Oriented bounding boxes



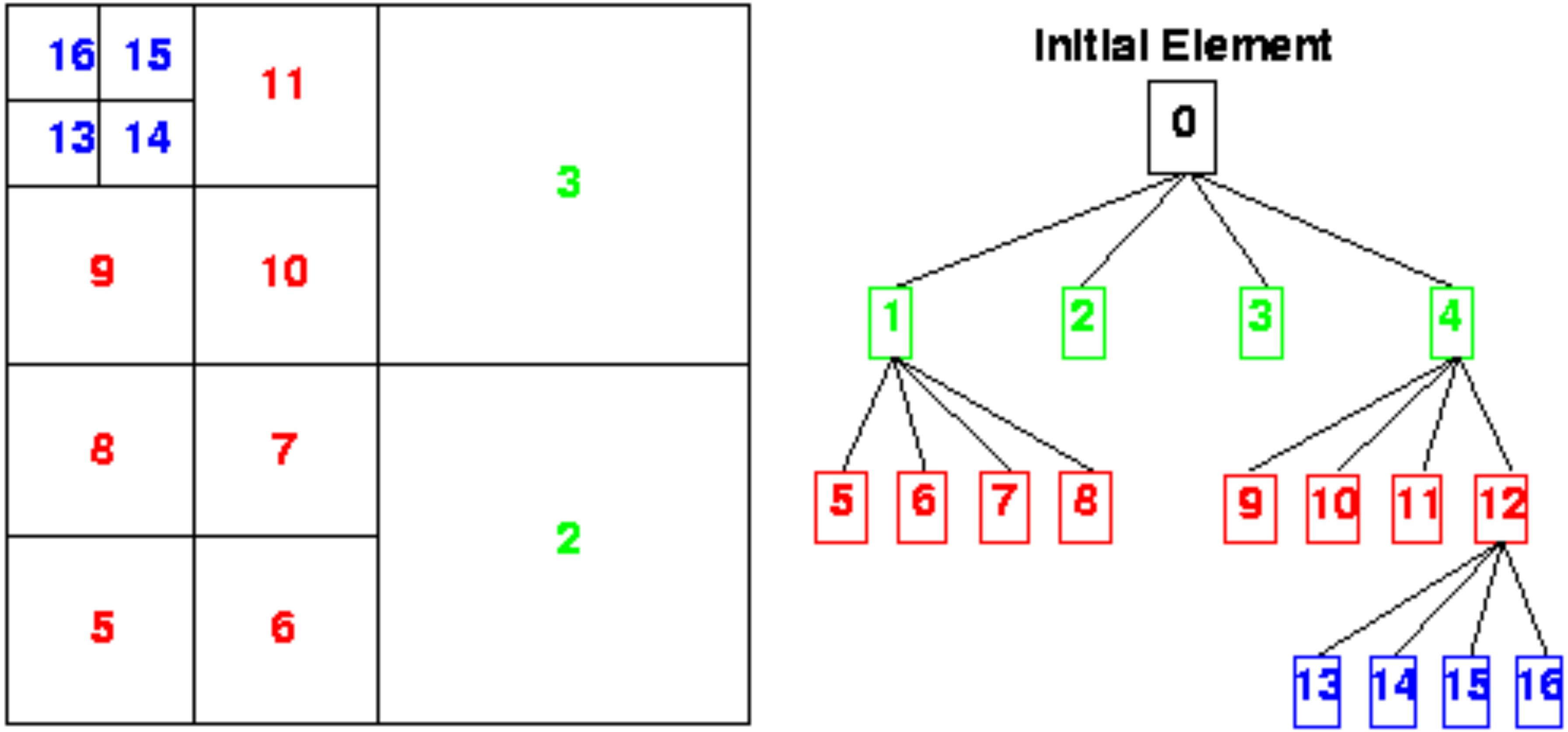
Broad-Phase

- Bounding Volume hierarchies
- **Spatial Partitioning**
- Sort and Sweep

Spatial Partitioning

- Divide 3D space into regions
  - Regular grid
  - Octree (quadtree)
  - k-D tree
- Perform pair-wise collisions only if in same region

Quadtree



[Karen Devine (kddevin@cs.sandia.gov)]

k-D tree

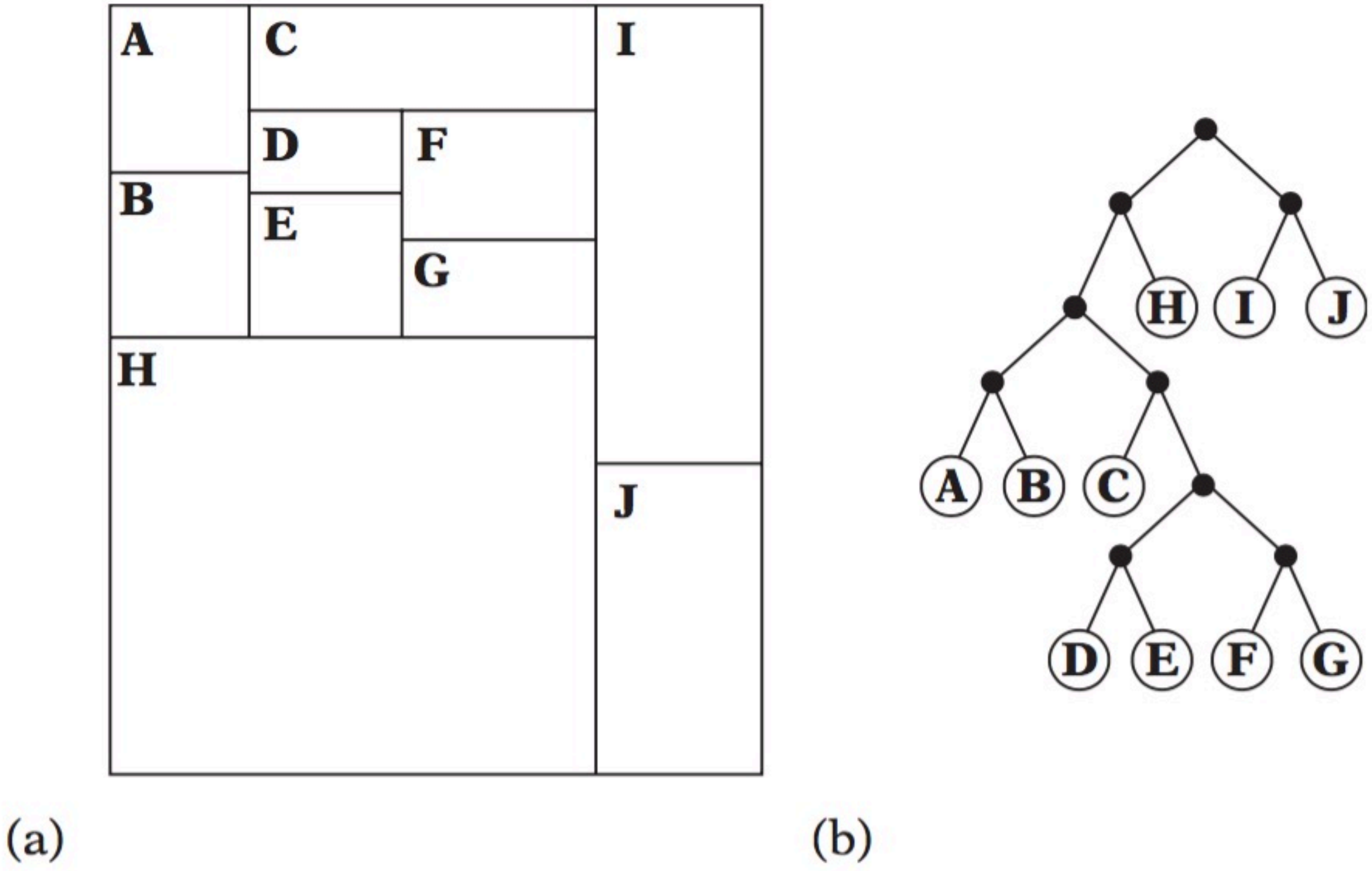


Figure 7.14 A 2D *k*-d tree. (a) The spatial decomposition. (b) The *k*-d tree layout.

[Ericson: Real-Time Collision Detection]

Broad-Phase

- Bounding Volume hierarchies
- Spatial Partitioning
- **Sort and Sweep**

Sort and Sweep

- AKA Sweep and Prune

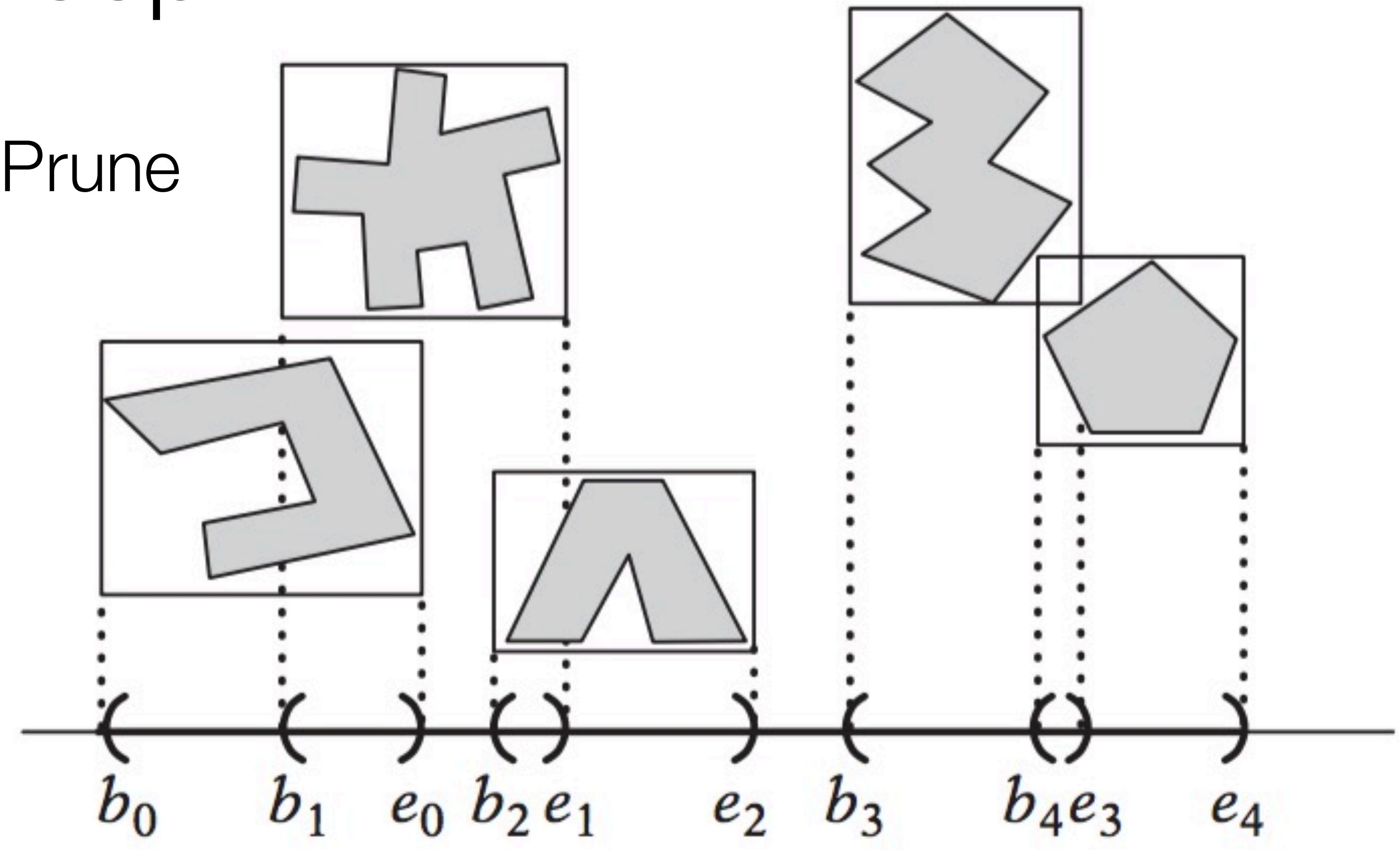
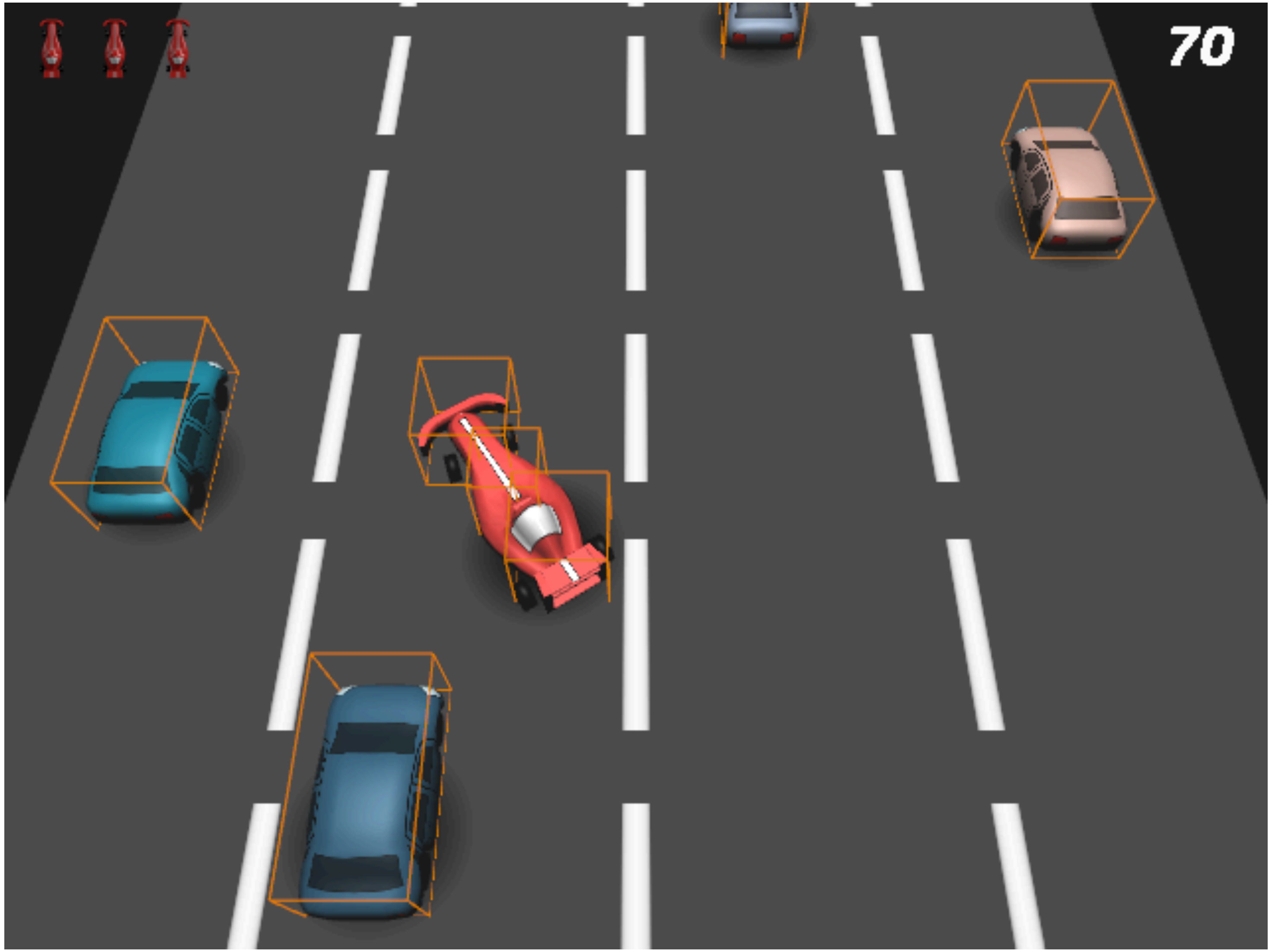


Figure 7.19 Projected AABB intervals on the *x* axis.

[Ericson: Real-Time Collision Detection]

Urge to Merge (471 W2012)



http://users.csc.calpoly.edu/~zwood/teachin
g/csc471/finalprojw12/rlight/

Questions?

- http://www.youtube.com/watch?v=b_cGXtc-nMg
- http://www.youtube.com/watch?v=nFd9BIcpHX4
- http://www.youtube.com/watch?v=2SXixK7yCGU