Physics based animation

Lecture 09 - Collision detection
Part 2 - Bounding volumes

Grégory Leplâtre

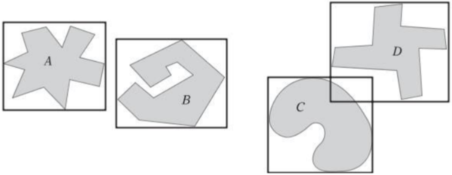
g.leplatre@napier.ac.uk, room D32
School of Computing
Edinburgh Napier University

Semester 1 - 2016/2017

Plan

- Why bounding volumes?
- Different types of bounding volumes
  - Definition
  - Intersection test
  - Creation and update

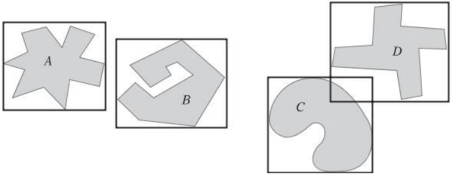
Bounding volumes



- Inexpensive intersection tests
- tight fitting
- Inexpensive to compute
- Easy to rotate and translate
- Use little memory

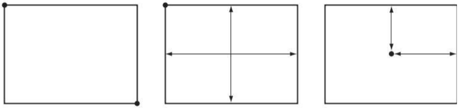
Outline

Axis aligned bounding volumes (AABBs)



- rectangular six-sided box (in 3D)
- faces normals parallel to the axes of coordinate system

Axis aligned bounding volumes (AABBs)



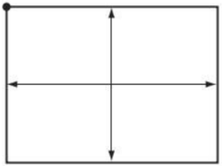
- rectangular six-sided box (in 3D)
- faces normals parallel to the axes of coordinate system
- Three common representations:
  - min-max
  - min-widths
  - center-radius

Axis aligned bounding volumes (AABBs)

min-max representation

```
// Region R={(x,y,z) |
// min.x <= x <=max.x
// min.y <= y <=max.y
// min.z <= z <=max.z
struct AABB \{
  Point min;
  Point max;
  \};
```

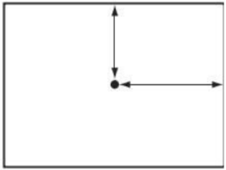
Axis aligned bounding boxes (AABBs)



min-widths representation

```
// Region R={(x,y,z) |
// min.x <= x <=min.x+dx
// min.y <= y <=min.y+dy
// min.z <= z <=min.z+dz
struct AABB \{
  Point min;
  float d[3]; // dx, dy, dz
  \};
```

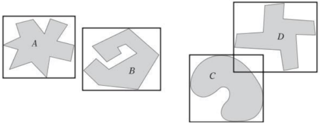
Axis aligned bounding volumes (AABBs)



center-radius representation

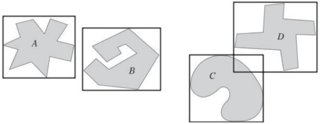
```
// Region R={(x,y,z) |
// |c.x-x| <= rx
// |c.y-y| <= ry
// |c.z-z| <= rz
struct AABB \{
  Point c; // center point of BB
  float r[3]; // rx, ry, rz
  \};
```

AABB-AABB intersection



- AABBs will overlap if they overlap on **all three axes**
- Exercise: overlap test for min-max representation

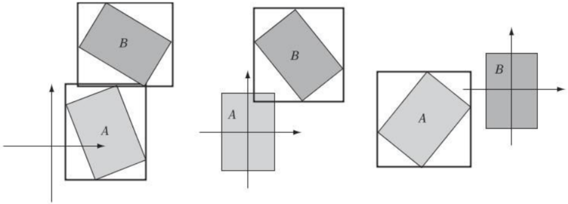
AABB-AABB intersection



- AABBs will overlap if they overlap on **all three axes**
- Exercise: overlap test for min-max representation

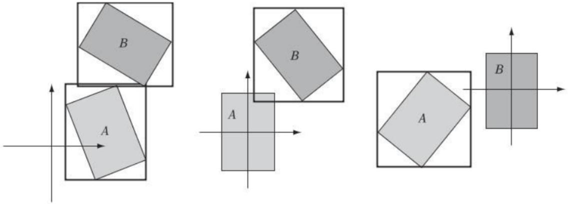
```
int intersect(AABB a, AABB b){
  // exit with no intersection if no intersection on one axis
  if (a.max[0]<b.min[0] || a.min[0]>b.max[0]) return 0;
  if (a.max[1]<b.min[1] || a.min[1]>b.max[1]) return 0;
  if (a.max[2]<b.min[2] || a.min[2]>b.max[2]) return 0;
  // otherwise exit with intersection
  return 1;
}
```

Computing and updating AABBs



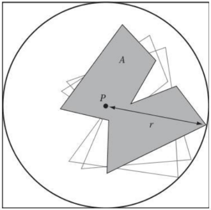
- Which coordinate system? (world vs local)

Computing and updating AABBs



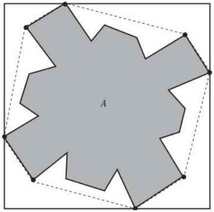
- Which coordinate system? (world vs local)
- Reconstruction strategies (after a rotation):

Computing and updating AABBs



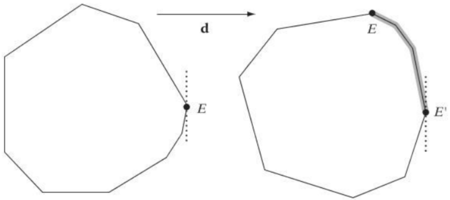
- Which coordinate system? (world vs local)
- Reconstruction strategies (after a rotation):
  - Loose fixed-size AABB that **always encloses the object**

Computing and updating AABBs



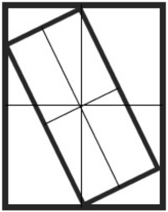
- Which coordinate system? (world vs local)
- Reconstruction strategies (after a rotation):
  - Loose fixed-size AABB that **always encloses the object**
  - Tight dynamic reconstruction **from original set point** (describe algorithm)

Computing and updating AABBs



- Which coordinate system? (world vs local)
- Reconstruction strategies (after a rotation):
  - Loose fixed-size AABB that **always encloses the object**
  - Tight dynamic reconstruction **from original set point** (describe algorithm)
  - Tight dynamic reconstruction using **hill climbing**

Computing and updating AABBs



- Which coordinate system? (world vs local)
- Reconstruction strategies (after a rotation):
  - Loose fixed-size AABB that **always encloses the object**
  - Tight dynamic reconstruction **from original set point** (describe algorithm)
  - Tight dynamic reconstruction using **hill climbing**
  - approximate dynamic **reconstruction from rotated AABB**

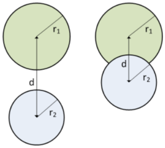
Outline

Bounding Sphere

- Simple, memory efficient
- Inexpensive intersection tests
- Rotationally invariant

```
// Region R = {(x, y, z) | (x-c.x)^2+(y-c.y)^2+(z-c.z)^2
    <= r^2}
struct Sphere{
  Point c; // Sphere center
  float r; // Sphere radius
};
```

Sphere-sphere intersection




```
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;
}
```

Computing a Bounding Sphere

Computing a Bounding Sphere

- From AABB:
  - take midpoint of AABB as sphere center
  - radius is distance to furthest point from center.

Computing a Bounding Sphere

- From AABB:
  - take midpoint of AABB as sphere center
  - radius is distance to furthest point from center.
- Ritter's bounding sphere algorithm (Ritter 1990)
  1. Make one (quick) pass through the N points. Find these six points:
     - The point with minimum x, the point with maximum x,
     - The point with minimum y, the point with maximum y,
     - The point with minimum z, the point with maximum z.

     This gives three pairs of points. **Pick the pair with the maximum point-to-point separation**. Calculate the initial sphere, using this pair of points as a diameter.
  2. Make a second pass through the N points: **for each point outside the current sphere, update the current sphere to the larger sphere passing through the point on one side**, and the back side of the old sphere on the other side.

Computing a Bounding Sphere

- Other approaches:

Computing a Bounding Sphere

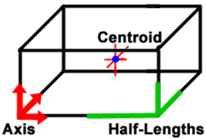
- Other approaches:
  - Finding the direction of Maximum spread using Principal Component Analysis (Wu 1992)

Computing a Bounding Sphere

- Other approaches:
  - Finding the direction of Maximum spread using Principal Component Analysis (Wu 1992)
  - Minimum bounding sphere. See (Welzl 1991). Based on the following observations
    - Assume that a minimum Bounding Sphere S has been found for a point set P
    - Let's add a new point q to P
    - The minimum BS for P U q is a new BS that has q on its boundary

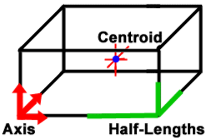
Outline

Orientated Bounding Boxes (OBBs)



- Similar to AABBs, but with arbitrary orientation
- Possible representations:
  - eight vertices
  - six planes
  - three slabs (pairs of parallel planes)
  - Center point + orientation matrix + three halfedge lengths (preferred)

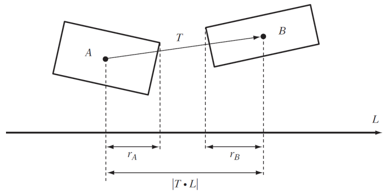
Orientated Bounding Boxes (OBBs)



- Center point + orientation matrix + three halfedge lengths (preferred)

```
// Region R = {x | x=c+r*u[0]+s*u[1]+t*u[2]}, |r|<=e[0],
    |s|<=e[1], |t|<=e[2]
struct OBB{
  Point c; // OBB center point
  Vector u[3]; // Local x, y, and z axes
  Vector e; // Positive halfwidth extents of OBB along
    each axis
};
```

OBB-OBB intersection



- **Separating axis** test:
  - A and B are separate if for **a number of axes L**:

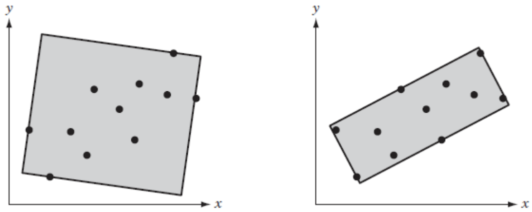
$$|\boldsymbol{T} \bullet \boldsymbol{L}| > r_A + r_B$$

  - At most 15 such tests must be performed
  - If the boxes fail to overlap on any of the 15 axes, they are not intersecting.

OBB-OBB intersection

L	$\lvert T \cdot L \rvert$	r_A	r_B
$\mathbf{u}_0^A$	$\lvert \mathbf{t}_0 \rvert$	$\mathbf{e}_0^A$	$\mathbf{e}_0^B \lvert r_{00} \rvert + \mathbf{e}_1^B \lvert r_{01} \rvert + \mathbf{e}_2^B \lvert r_{02} \rvert$
$\mathbf{u}_1^A$	$\lvert \mathbf{t}_1 \rvert$	$\mathbf{e}_1^A$	$\mathbf{e}_0^B \lvert r_{10} \rvert + \mathbf{e}_1^B \lvert r_{11} \rvert + \mathbf{e}_2^B \lvert r_{12} \rvert$
$\mathbf{u}_2^A$	$\lvert \mathbf{t}_2 \rvert$	$\mathbf{e}_2^A$	$\mathbf{e}_0^B \lvert r_{20} \rvert + \mathbf{e}_1^B \lvert r_{21} \rvert + \mathbf{e}_2^B \lvert r_{22} \rvert$
$\mathbf{u}_0^B$	$\lvert \mathbf{t}_0 r_{00} + \mathbf{t}_1 r_{10} + \mathbf{t}_2 r_{20} \rvert$	$\mathbf{e}_0^A \lvert r_{00} \rvert + \mathbf{e}_1^A \lvert r_{10} \rvert + \mathbf{e}_2^A \lvert r_{20} \rvert$	$\mathbf{e}_0^B$
$\mathbf{u}_1^B$	$\lvert \mathbf{t}_0 r_{01} + \mathbf{t}_1 r_{11} + \mathbf{t}_2 r_{21} \rvert$	$\mathbf{e}_0^A \lvert r_{01} \rvert + \mathbf{e}_1^A \lvert r_{11} \rvert + \mathbf{e}_2^A \lvert r_{21} \rvert$	$\mathbf{e}_1^B$
$\mathbf{u}_2^B$	$\lvert \mathbf{t}_0 r_{02} + \mathbf{t}_1 r_{12} + \mathbf{t}_2 r_{22} \rvert$	$\mathbf{e}_0^A \lvert r_{02} \rvert + \mathbf{e}_1^A \lvert r_{12} \rvert + \mathbf{e}_2^A \lvert r_{22} \rvert$	$\mathbf{e}_2^B$
$\mathbf{u}_0^A \times \mathbf{u}_0^B$	$\lvert \mathbf{t}_2 r_{10} - \mathbf{t}_1 r_{20} \rvert$	$\mathbf{e}_1^A \lvert r_{20} \rvert + \mathbf{e}_2^A \lvert r_{10} \rvert$	$\mathbf{e}_1^B \lvert r_{02} \rvert + \mathbf{e}_2^B \lvert r_{01} \rvert$
$\mathbf{u}_0^A \times \mathbf{u}_1^B$	$\lvert \mathbf{t}_2 r_{11} - \mathbf{t}_1 r_{21} \rvert$	$\mathbf{e}_1^A \lvert r_{21} \rvert + \mathbf{e}_2^A \lvert r_{11} \rvert$	$\mathbf{e}_0^B \lvert r_{02} \rvert + \mathbf{e}_2^B \lvert r_{00} \rvert$
$\mathbf{u}_0^A \times \mathbf{u}_2^B$	$\lvert \mathbf{t}_2 r_{12} - \mathbf{t}_1 r_{22} \rvert$	$\mathbf{e}_1^A \lvert r_{22} \rvert + \mathbf{e}_2^A \lvert r_{12} \rvert$	$\mathbf{e}_0^B \lvert r_{01} \rvert + \mathbf{e}_1^B \lvert r_{00} \rvert$
$\mathbf{u}_1^A \times \mathbf{u}_0^B$	$\lvert \mathbf{t}_0 r_{20} - \mathbf{t}_2 r_{00} \rvert$	$\mathbf{e}_0^A \lvert r_{20} \rvert + \mathbf{e}_2^A \lvert r_{00} \rvert$	$\mathbf{e}_1^B \lvert r_{12} \rvert + \mathbf{e}_2^B \lvert r_{11} \rvert$
$\mathbf{u}_1^A \times \mathbf{u}_1^B$	$\lvert \mathbf{t}_0 r_{21} - \mathbf{t}_2 r_{01} \rvert$	$\mathbf{e}_0^A \lvert r_{21} \rvert + \mathbf{e}_2^A \lvert r_{01} \rvert$	$\mathbf{e}_0^B \lvert r_{12} \rvert + \mathbf{e}_2^B \lvert r_{10} \rvert$
$\mathbf{u}_1^A \times \mathbf{u}_2^B$	$\lvert \mathbf{t}_0 r_{22} - \mathbf{t}_2 r_{02} \rvert$	$\mathbf{e}_0^A \lvert r_{22} \rvert + \mathbf{e}_2^A \lvert r_{02} \rvert$	$\mathbf{e}_0^B \lvert r_{11} \rvert + \mathbf{e}_1^B \lvert r_{10} \rvert$
$\mathbf{u}_2^A \times \mathbf{u}_0^B$	$\lvert \mathbf{t}_1 r_{00} - \mathbf{t}_0 r_{10} \rvert$	$\mathbf{e}_0^A \lvert r_{10} \rvert + \mathbf{e}_1^A \lvert r_{00} \rvert$	$\mathbf{e}_1^B \lvert r_{22} \rvert + \mathbf{e}_2^B \lvert r_{21} \rvert$
$\mathbf{u}_2^A \times \mathbf{u}_1^B$	$\lvert \mathbf{t}_1 r_{01} - \mathbf{t}_0 r_{11} \rvert$	$\mathbf{e}_0^A \lvert r_{11} \rvert + \mathbf{e}_1^A \lvert r_{01} \rvert$	$\mathbf{e}_0^B \lvert r_{22} \rvert + \mathbf{e}_2^B \lvert r_{20} \rvert$
$\mathbf{u}_2^A \times \mathbf{u}_2^B$	$\lvert \mathbf{t}_1 r_{02} - \mathbf{t}_0 r_{12} \rvert$	$\mathbf{e}_0^A \lvert r_{12} \rvert + \mathbf{e}_1^A \lvert r_{02} \rvert$	$\mathbf{e}_0^B \lvert r_{21} \rvert + \mathbf{e}_1^B \lvert r_{20} \rvert$

Computing an OBB



- An OBB can be computed like an AABB (easy but loose)
- tighter OBB computation more difficult
  - $O(n)$ algorithm by (Gottschalk 2000) if performed from convex hull.
  - $O(n^3)$ algorithm by (O'Rourke 1985) guarantees minimum volume OBB, but too complex and expensive.

Outline

Sphere-swept Volumes

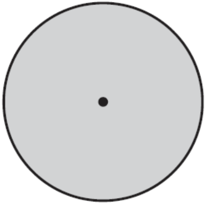
After boxes and spheres, how about cylinders?

Sphere-swept Volumes

After boxes and spheres, how about cylinders?

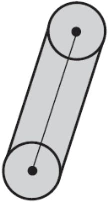
- Intersection math unfriendly
- friendlier if you cap them with half sphere

Sphere-swept Volumes



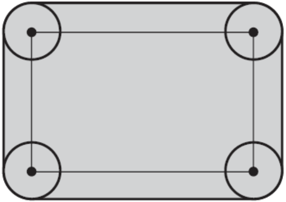
- Equivalent to extending the center of a Bounding Sphere
  - From a point (sphere)

Sphere-swept Volumes



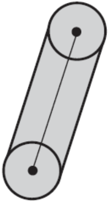
- Equivalent to extending the center of a Bounding Sphere
  - From a point (sphere)
  - To a line (capsule)

Sphere-swept Volumes



- Equivalent to extending the center of a Bounding Sphere
  - From a point (sphere)
  - To a line (capsule)
  - or a rectangle (lozenge)

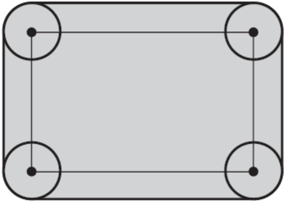
Sphere-swept Volumes



- Example 1: capsule

```
// region R = { x | (x - [StartPoint + (EndPoint - StartPoint)*t])^2 <= Radius }, 0 <= t <= 1
struct Capsule {
  Vector3 StartPoint
  Vector3 EndPoint
  float   Radius
}
```

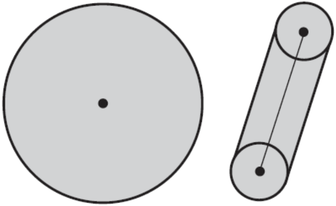
Sphere-swept Volumes



- Example 2: Lozenge

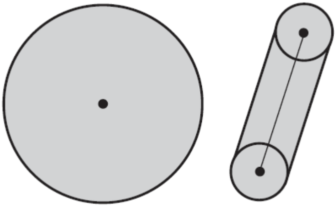
```
// region R = { x | (x - [Centre + Axis[0]*s + Axis[1]*t
    ])^2 <= Radius }, 0 <= s,t <= 1
struct Lozenge{
  Vector3 Centre
  Vector3[2] Axis
  float   Radius
}
```

Sphere-swept Volumes



- **Intersection**: equates to computing the distance between the swept primitives. For example, sphere-capsule: distance between point and line.

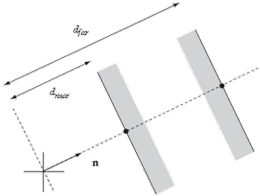
Sphere-swept Volumes



- **Intersection**: equates to computing the distance between the swept primitives. For example, sphere-capsule: distance between point and line.
- **Conputing SSVs**: principal axes have to be determined.

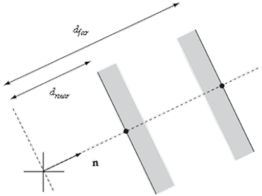
Outline

Discrete-Oriented Polytopes (k-DOPs)



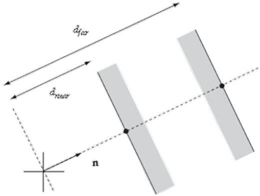
- Based on Kay-Kajiya slabs:
  - Slab: **infinite region between two parallel planes**
  - At least **Three slabs** required to form **closed 3D volume**

Discrete-Oriented Polytopes (k-DOPs)



- Based on Kay-Kajiya slabs:
  - Slab: **infinite region between two parallel planes**
  - At least **Three slabs** required to form **closed 3D volume**

Discrete-Oriented Polytopes (k-DOPs)



- Based on Kay-Kajiya slabs:
  - Slab: **infinite region between two parallel planes**
  - At least **Three slabs** required to form **closed 3D volume**
- k-DOP: **tightest fixed set of slabs whose normals are defined as a fixed set of axes shared among all k-DOP bounding volumes**

Discrete-Oriented Polytopes (k-DOPs)

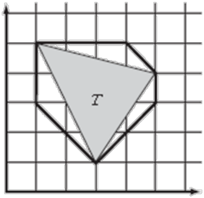
- 8-DOP example

```
struct DOP8{
  float min[4]; // minimum distance (from origin) along axes 0 to 3
  float max[4]; // maximum distance (from origin) along axes 0 to 3
}
```

Discrete-Oriented Polytopes (k-DOPs)

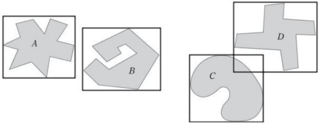
- Examples
  - An AABB is a 6-DOP. (but not all 6-DOPs are AABBs)

Discrete-Oriented Polytopes (k-DOPs)



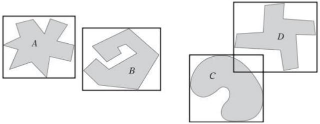
- Examples
  - An AABB is a 6-DOP. (but not all 6-DOPs are AABBs)
  - The above image shows (in 2D) a triangle bounded by an 8-DOP

k-DOP intersection



- Simple extension to AABB intersection test:

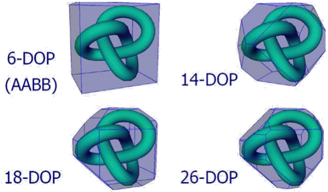
k-DOP intersection



- Simple extension to AABB intersection test:

```
int intersect(KDOP a, KDOP b, int k){
  // check if
  for (int i = 0; i < k/2; i++){
    if (a.min[i] > b.max[i] || a.max[i] < b.min[i])
      return 0;
  return 1;
}
```

k-DOP pros and cons



- fast intersection tests
- approximately the same amount of storage needed to store 14-DOP as an OBB, but the 14-DOP likely to be tighter.
- Disadvantage: updating k-DOP after rotation.
- $\Rightarrow$ k-DOPs better for scenes with many static objects and a limited amount of moving objects.

Outline

Summary

- Most frequently, **spheres and boxes** are used

Summary

- Most frequently, **spheres and boxes** are used
- If tighter fit required, **slab volumes** or **convex hulls** should be used

Summary

- Most frequently, **spheres and boxes** are used
- If tighter fit required, **slab volumes** or **convex hulls** should be used
- loose BVs used as early overlap rejection tests before more expensive tests on bound objects need to be carried out.

Summary

- Most frequently, **spheres and boxes** are used
- If tighter fit required, **slab volumes** or **convex hulls** should be used
- loose BVs used as early overlap rejection tests before more expensive tests on bound objects need to be carried out.
- Typically, bounding volumes computed during **preprocessing stage**. Only **updates** need to be carried out at **runtime**

Summary

- Most frequently, **spheres and boxes** are used
- If tighter fit required, **slab volumes** or **convex hulls** should be used
- loose BVs used as early overlap rejection tests before more expensive tests on bound objects need to be carried out.
- Typically, bounding volumes computed during **preprocessing stage**. Only **updates** need to be carried out at **runtime**
- To take things further: Read **Chapter 4** of Ericson (2004).

References

- Ericson, C. (2004). Real-time collision detection. CRC Press.
- Ritter, J. (1990, August). An efficient bounding sphere. In Graphics Gems (pp. 301-303). Academic Press Professional, Inc.
- Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). In New results and new trends in computer science (pp. 359-370). Springer Berlin Heidelberg.
- Wu, X. (1992, July). A linear-time simple bounding volume algorithm. In Graphics Gems III (pp. 301-306). Academic Press Professional, Inc..