Collision Detection
(SENG 463 - Game Programming)

Dr.Çağatay ÜNDEĞER
Research and Innovation Director
SimBT Inc.

e-mail :
cagatay.undeger@simbt.com.tr
cagatay@undeger.com

Outline

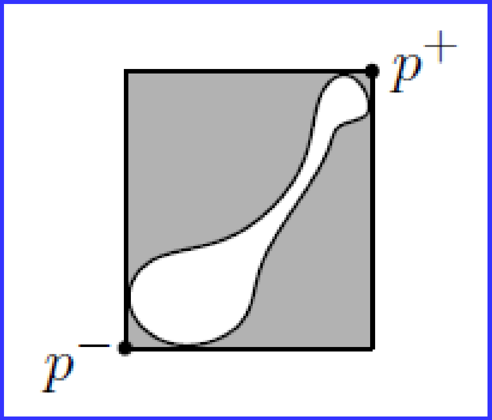
- Bounding Enclosures (Summary)
- Collision Detection

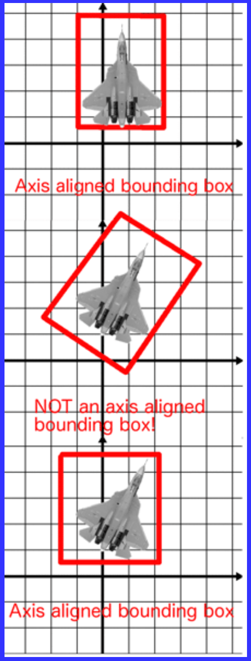
Bounding Enclosures

- When storing complex geometric objects in a spatial data structure
- Common to first approximate the object by a simple enclosing structure.
- Bounding enclosures are often very valuable to approximate an object as a filter in
- Objects in rendering, collision detection, etc.

Axis-Aligned Bounding Boxes

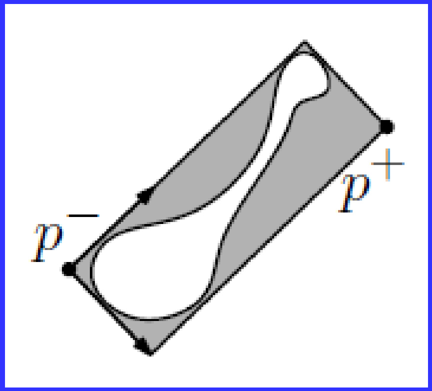
- This is an enclosing rectangle whose sides are parallel to the coordinate axes
- It is not possible to rotate the object without recomputing the entire bounding box.





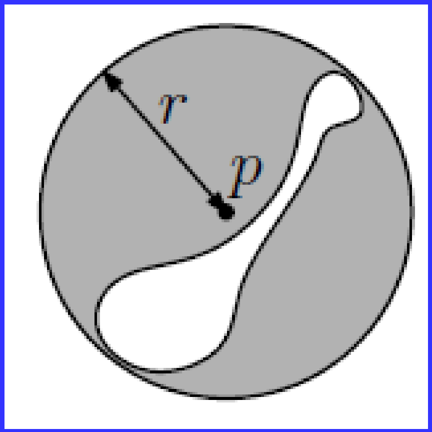
General Bounding Boxes

- The principal shortcoming of axis-parallel bounding boxes is:
- That it is not possible to rotate the object without recomputing the entire bounding box.
- In contrast, general (arbitrarily-oriented) bounding boxes can be rotated without the need to recompute them



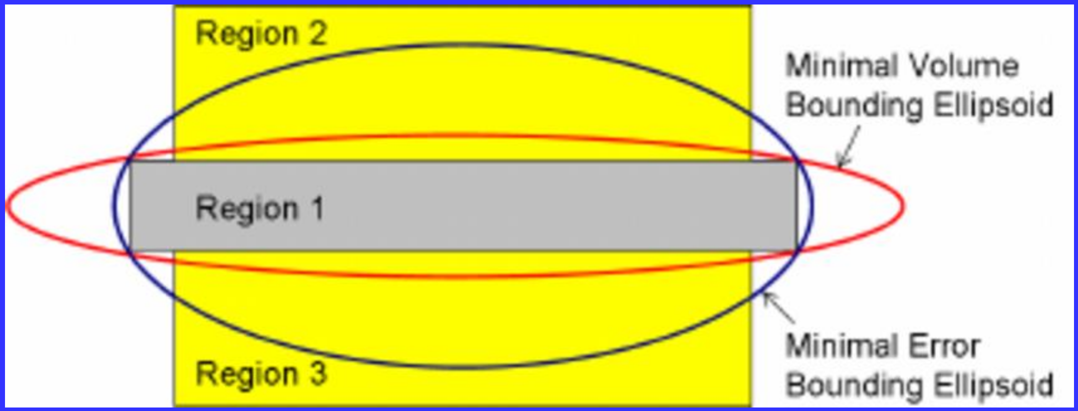
Bounding Spheres

- A sphere can be represented by a center point p and a radius r
- Spheres are invariant under rigid transformations,
  - translation and rotation.



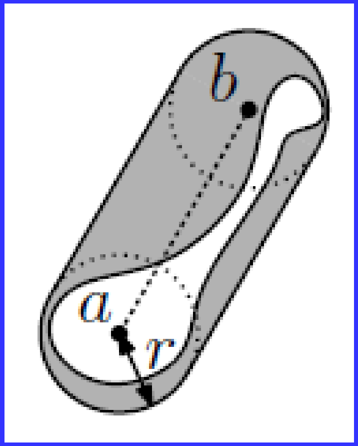
Bounding Ellipsoids

- The main problem with spheres is that some objects are not well approximated by a sphere *(problem also exists with axis-parallel bounding boxes)*
- An ellipsoid is just a sphere under an affine transformation.



Bounding Capsules

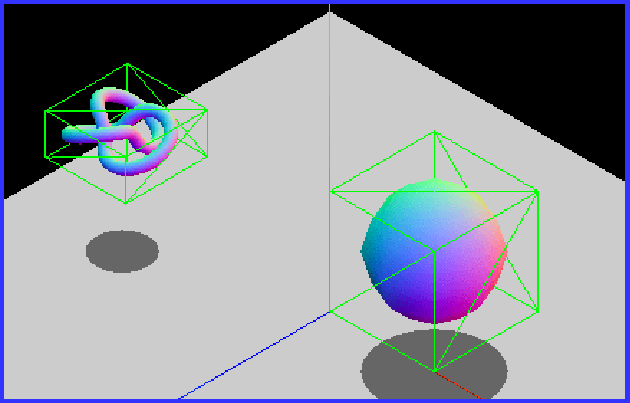
- This shape can be thought of as a rounded cylinder.
- It consists of the set of points that lie within some distance r of a line segment ab

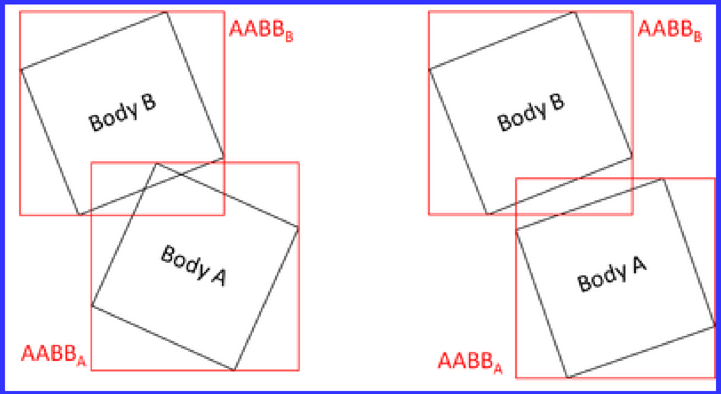


- A line segment is chosen
- All the points that is within distance r to that line segment is inside bounds

Collision Detection

- By enclosing an object within a bounding enclosure,
- Collision detection cost is reduced by predetermining whether two such enclosures may intersect each other or not.
- Note that if we support k different types of enclosure, we need to handle all possible pairs of combinations of collisions.

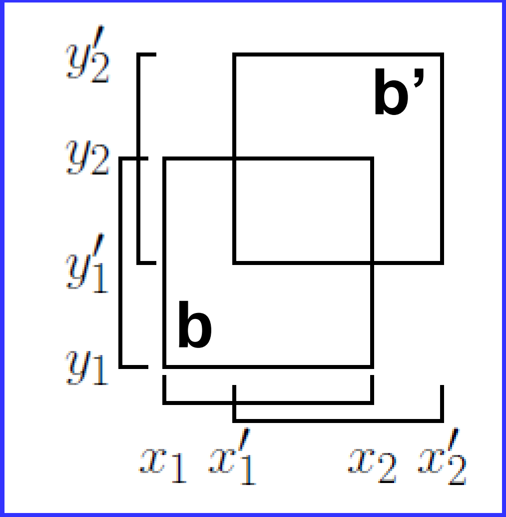




AABB to AABB Test

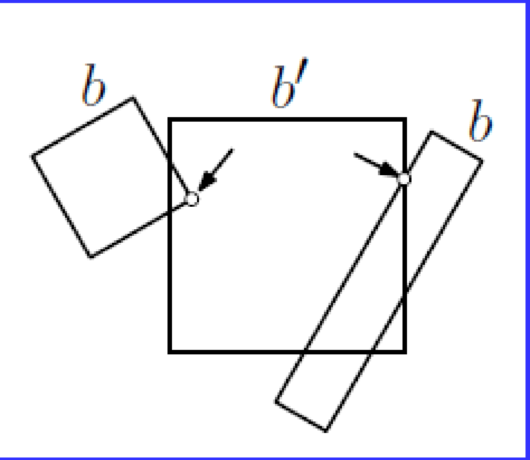
- We can test whether two axis-aligned bounding boxes overlap by testing that all pairs of intervals overlap.
- Suppose that we have two boxes b and b'
- These boxes overlap if and only if:

$$[x_1, x_2] \cap [x'_1, x'_2] \neq \emptyset \quad \text{and} \quad [y_1, y_2] \cap [y'_1, y'_2] \neq \emptyset$$



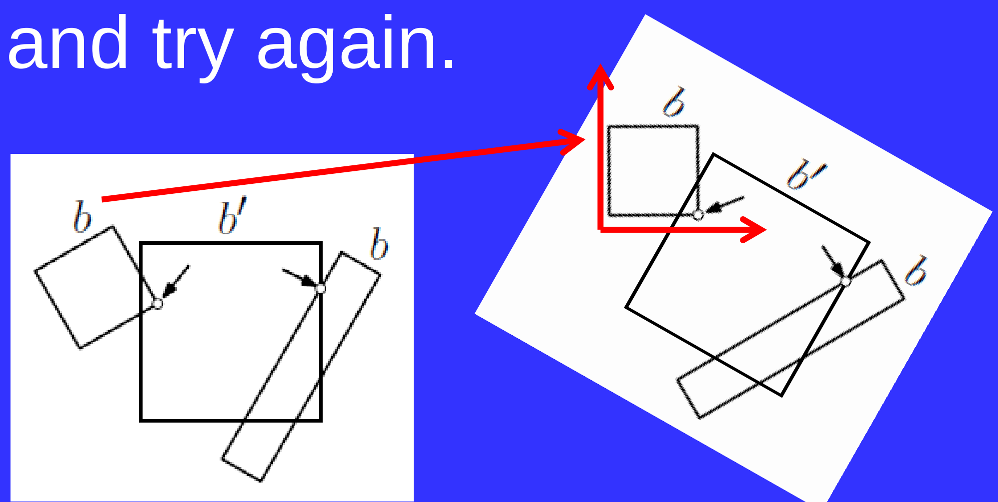
General BB to BB Test

- Determining whether two arbitrarily oriented boxes b and b’ intersect is a nontrivial task.
- If they do intersect, one of the following must happen:
  - A vertex of b lies within b’ or vice versa.
  - An edge of b intersects a face of b’ or vice versa.



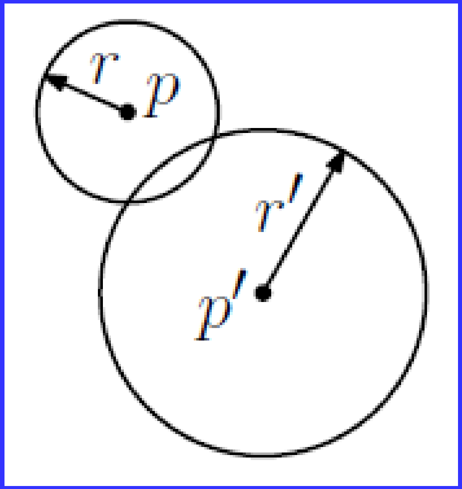
General BB to BB Test

- One way to simplify these computations is to first compute a rotation that aligns one of the two boxes with the coordinate axes.
- Determining point membership or edge-face intersection with an AABB is simpler than for general boxes.
- If both tests fail, we reverse the roles of the two boxes and try again.



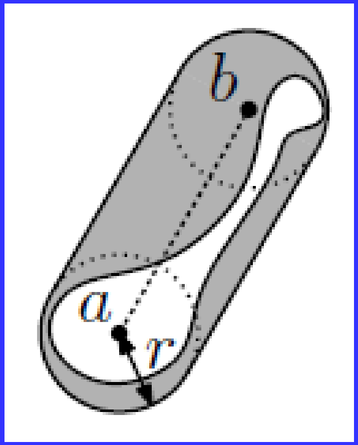
Shere to Sphere Test

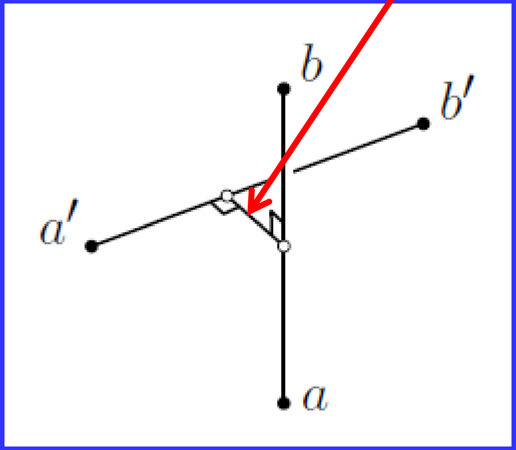
- We can determine whether two spheres intersect by computing the distances between their centers.
- Given two spheres, one with center p and radius r and the other with center p’ and radius r’,
  - They intersect if and only if dist(p,p’) <= r + r’



Capsule to Capsule Test

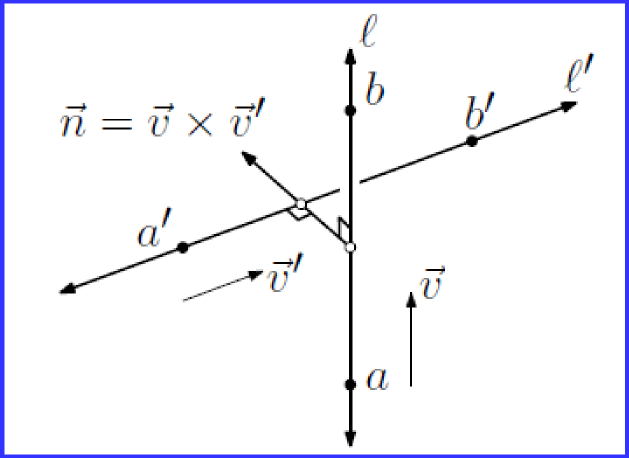
- Capsule consists of the set of points that lie within some distance r of a line segment ab
- We can determine whether two capsules intersect by computing the nearest distance between their line segments.
- Capsules intersect if and only if the shortest distance between the line segments <= r + r'





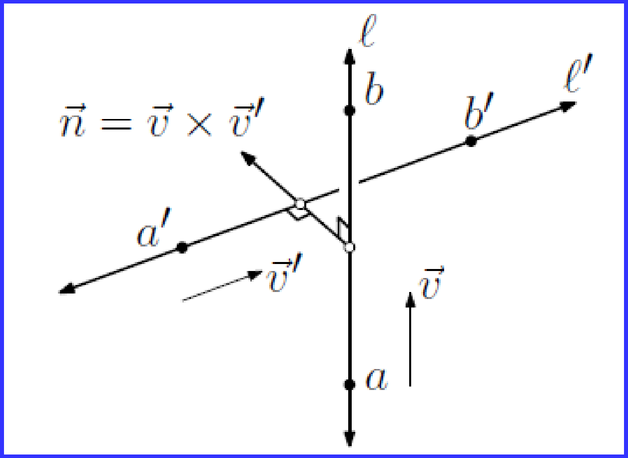
Computing Line Distance

- Computing the distance between two segments
- First compute the distance between the two infinite lines l and l’
- Points with minimum distance might lie outside of the associated line segments
- We simply clamp the result to the closest segment endpoint on this line.



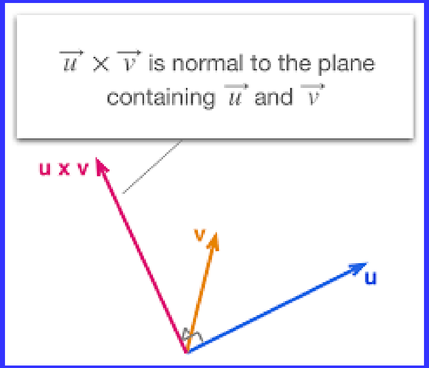
Computing Line Distance

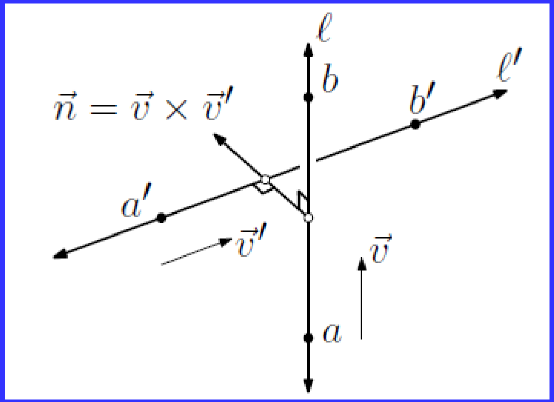
- For computing the distance between two lines.
  - We'll assume the general case where the lines a and a' are skew (neither parallel nor intersecting)
- Let v = b – a be the directional vector for l
- let v' = b' – a' be the directional vector for l'



Computing Line Distance

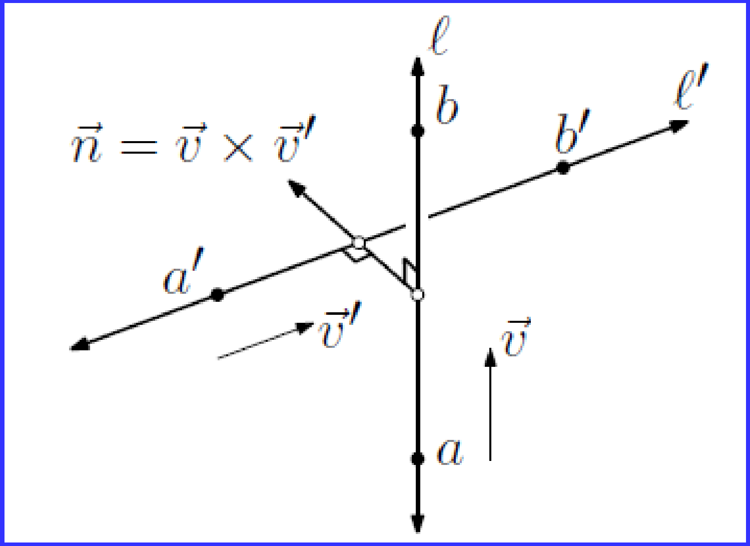
- By properties of the cross product,
  - Vector vn = cross (v x v')
    - Is perpendicular to both lines.
  - Normalize this vector to unit length,
    - We have vn = vn / ||vn||
  - The distance between l and l' is just the distance between projection points of a and a' on to perpendicular vector vn
  - Distance = dot ((a' – a), vn)





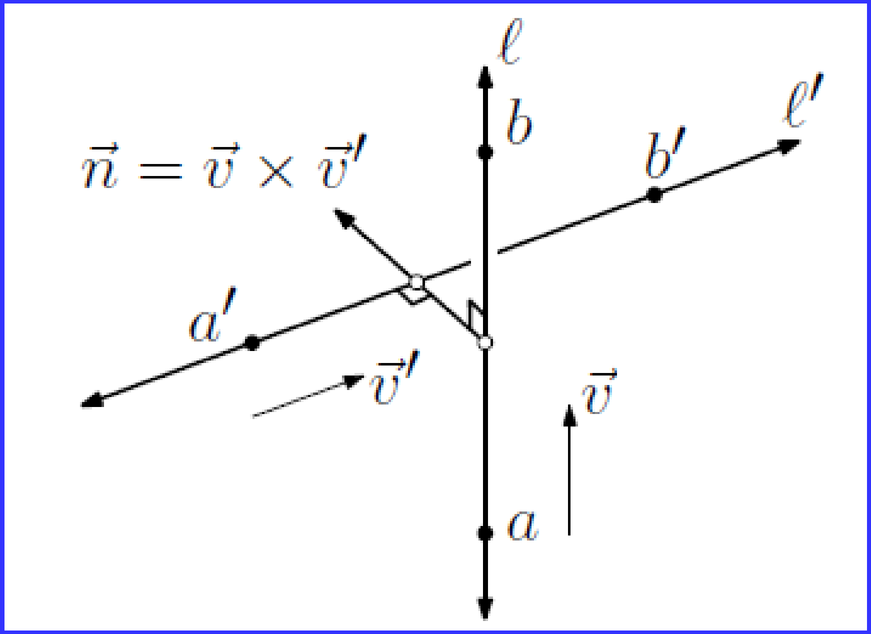
Computing Line Distance

- Vector3 a = (LineA_P2.position - LineA_P1.position).normalized;
- Vector3 b = (LineB_P2.position - LineB_P1.position).normalized;
- Vector3 vn = Vector3.Cross(b, a).normalized;
- Distance = Vector3.Dot(LineB_P1.position - LineA_P1.position, vn);



Computing Closest Points

- Computing closest points on these lines are more complicated than computing distance
- You need to solve and find 3 variables in 3 different linear equations



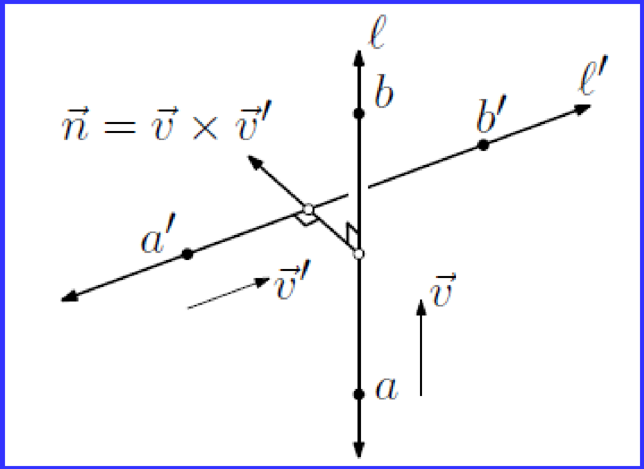
Computing Closest Points

- Vector3 projection_ = Vector3.Dot(LineB_P1.position - LineA_P1.position, a) * a;

- Vector3 rejection = LineB_P1.position - LineA_P1.position -
  Vector3.Dot(LineB_P1.position - LineA_P1.position, a) * a -
  Vector3.Dot(LineB_P1.position - LineA_P1.position, vn) * vn;

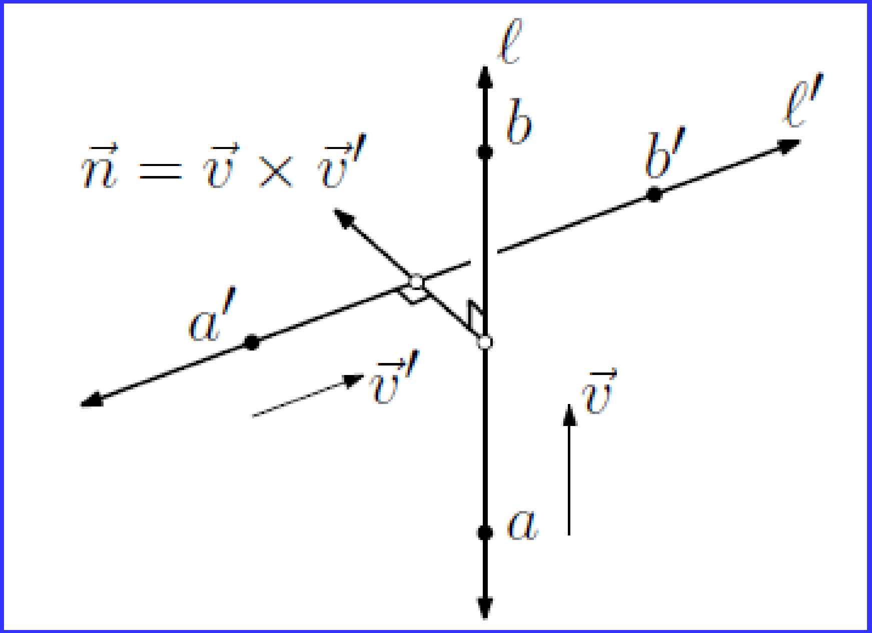
- Vector3 closest_approach_B = LineB_P1.position - b * rejection.magnitude /
  Vector3.Dot(b, rejection.normalized);

- Vector3 closest_approach_A = closest_approach_B - Distance * vn;



Computing Closest Segment Points

- If closest points are out of line segments,
  - Closest points shall be corrected



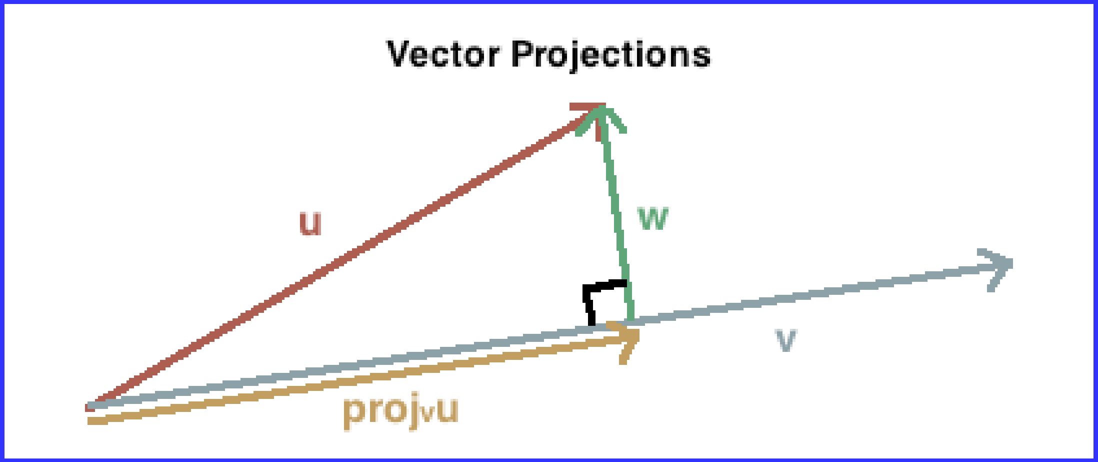
Computing Closest Segment Points

```
float Closest_A = Vector3.Dot((closest_approach_A - LineA_P1.position), (LineA_P2.position - LineA_P1.position)) /
                                                                        (LineA_P2.position - LineA_P1.position).sqrMagnitude;
if (Closest_A < 0)
{
    closest_approach_A = LineA_P1.position;
}
else
if (Closest_A > 1)
{
    closest_approach_A = LineA_P2.position;
}

SIMILAR FOR closest_approach_AB
```

Dot product

$$\mathrm{proj}_{\vec{v}}\vec{u} = \frac{\vec{u}\cdot\vec{v}}{\|\vec{v}\|^2}\,\vec{v}$$



Capsule to Capsule Test

- Capsules intersect if and only if
  - Distance between
    - closest_approach_A and
    - closest_approach_B
  - Is smaller or equal to r + r’