Math for Gameplay / AI

John O'Brien
Senior Gameplay Programmer
Insomniac Games, Inc
jobrien@insomniacgames.com
jobrien@nc.rr.com

GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9

Overview

- Basic Object Intersection Tests
- Real gameplay example(s)
- AI-specific topics
  - Bayes' Theorem
  - Fuzzy Logic

Object Intersections

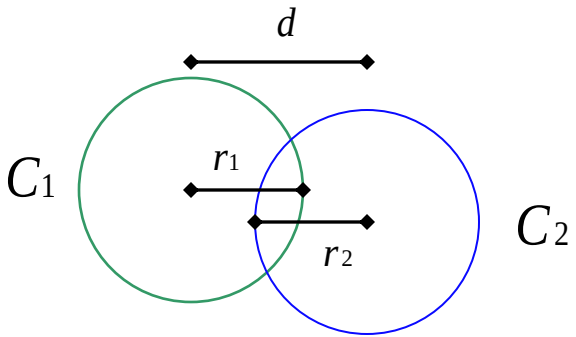
- In simulation and AI code, we rarely want to use actual object meshes to test for collision.
- Simplified collision representations are used instead.

Object Intersections

- Gameplay and AI code often requires checks between “real” game objects and abstract things like zones.
- AI usually needs to analyze and represent the world situation in a simplified way.

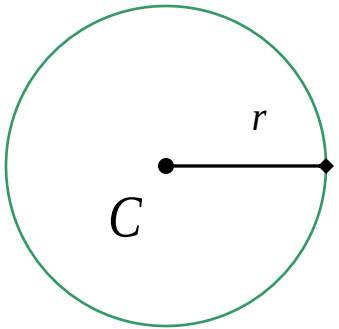
Bounding Spheres

- Simplest representation.
- Often used as 1st pass or screening check before using more detailed technique.



Bounding Sphere Representation

- Only need a Point and a radius.



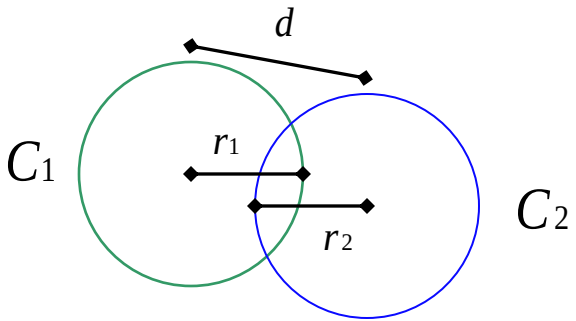
Bounding Sphere Pros and Cons

- Sphere intersection tests very simple and cheap
- Spheres are rarely a close fit to the shape of an object in the game world.

Sphere – Sphere Intersection

- The spheres intersect if $d <= r_1 + r_2$
- A fast computation of this check takes the form:

$$(C_1 - C_2) \cdot (C_1 - C_2) <= (r_1 + r_2)^2$$



Sphere – Plane Intersection

- Using the center of the sphere as the origin, the sphere can be described as:

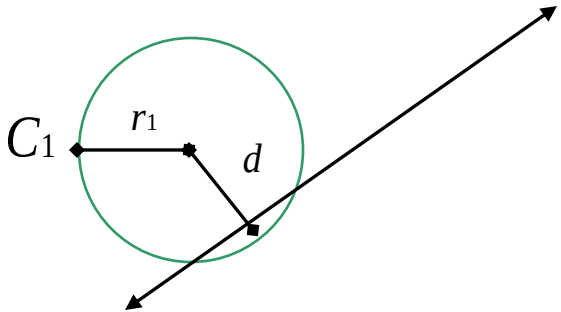
$$x^2 + y^2 + z^2 = r^2$$

- The equation for the plane is:

$$ax + by + cz = d$$

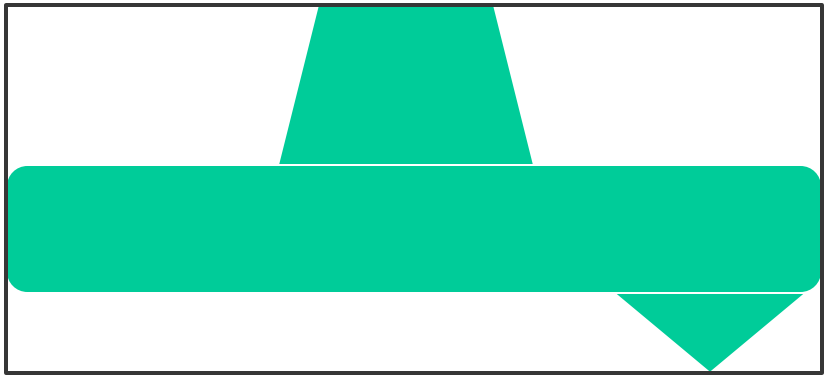
Sphere – Plane Intersection

- Sphere and a plane intersect if d < r, where d equals the absolute value of the *plane equation*.



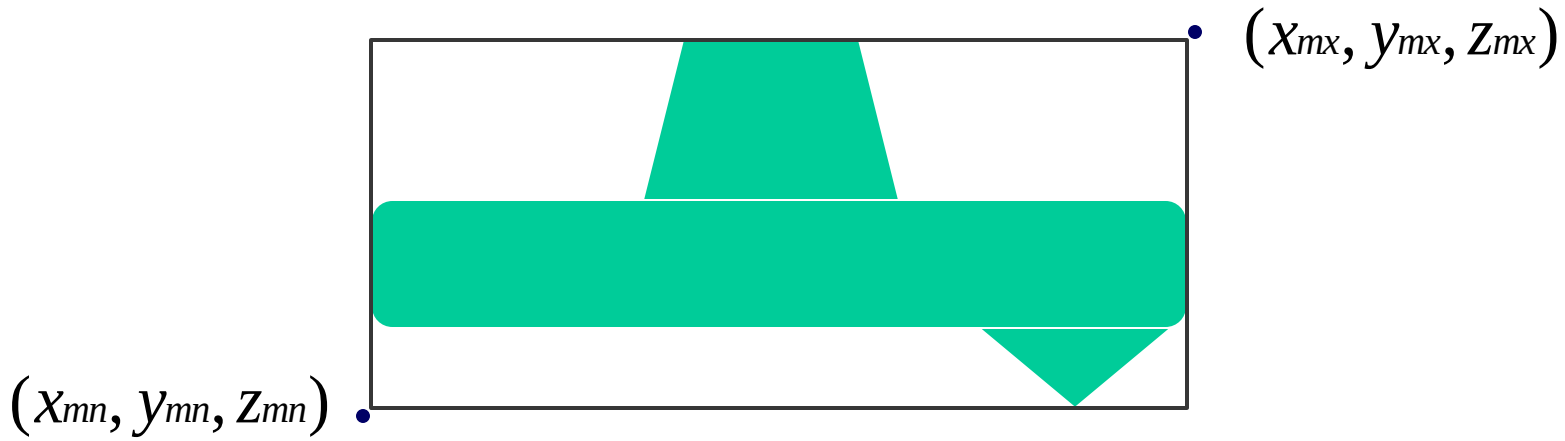
Axis-Aligned Bounding Boxes

- An AABB is a box surrounding an object which is aligned to the world-axes.



AABB Representation

- Can be represented by 2 points – the minimum and maximum x,y,z positions



AABB Pros and Cons

- Pros
  - Simple representation
  - Very cheap intersection computations
- Cons
  - Need to be re-computed each frame
  - Not necessarily a good fit to a model

AABB – AABB Intersection

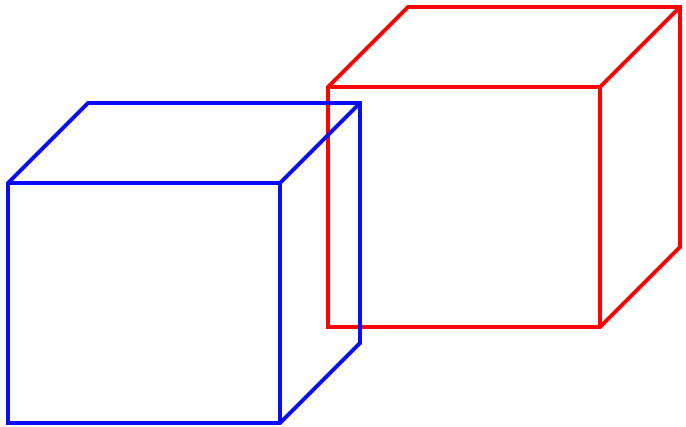
- Very simple to calculate. For each axis, and each object, check if the maximum for one object is less than the minimum for the other. If this is true for any axis, the objects do not intersect.

```
if ( (A.max.x < B.min.x) || (B.max.x < A.min.x) )
    return false;
```

  - Repeat for each axis.

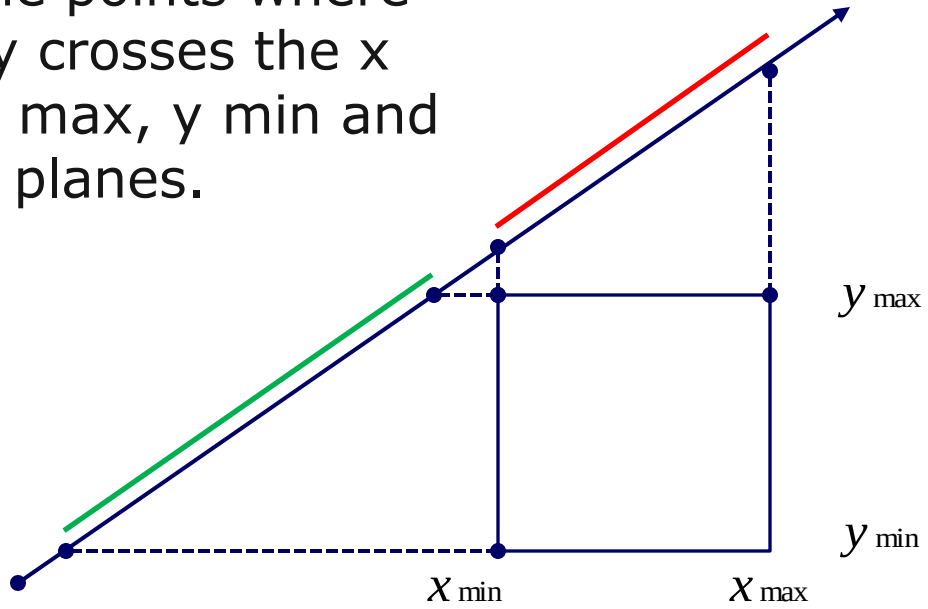
AABB – AABB Intersection

- For each axis, the points describing these AABBs overlap.



AABB – Ray Intersection

Find the points where the ray crosses the x min, x max, y min and y max planes.



If the resulting line intervals overlap, there is an intersection.

AABB – Ray Intersection

- To find the points x_1, x_2 representing the intersections with the x min and x max planes, given a ray described by a point *P* and a vector *v*:

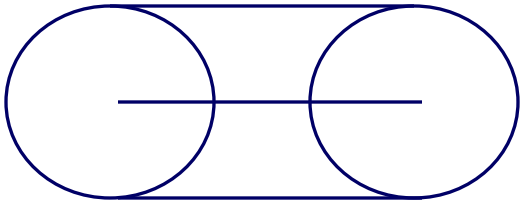
$$x_1 = \frac{x_{\min} - P_x}{v_x} \qquad x_2 = \frac{x_{\max} - P_x}{v_x}$$

AABB – Ray Intersection

- Test for overlap among (x1,x2), (y1,y2) and (z1, z2) segments.
- There are many algorithms and approaches to this problem.

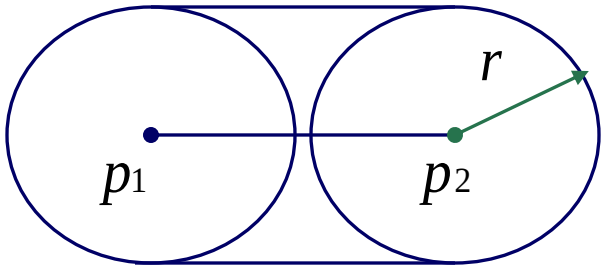
Swept Spheres / Capsules

- A collision type that is oriented to a model rather than the world coordinate system.
- Still relatively fast, and likely to be a better fit for many common objects in games.



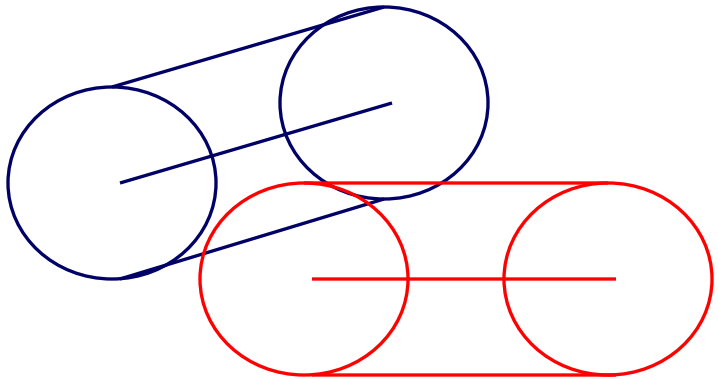
Swept Sphere Representation

- A swept sphere can be described by a line segment and a radius.



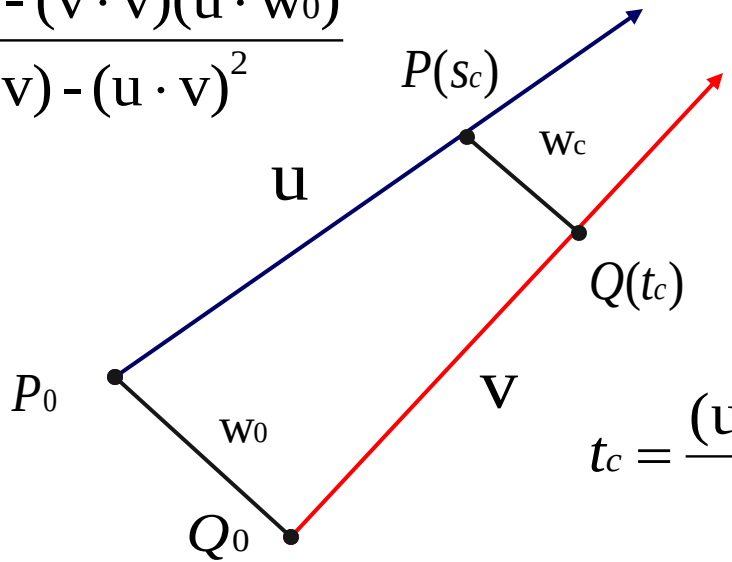
Swept Sphere Intersection

- Two swept spheres intersect if the shortest distance between their respective line segments is less than the sum of their radii.



Shortest Distance Between Two Lines

$$s_c = \frac{(u \cdot v)(v \cdot w_0) - (v \cdot v)(u \cdot w_0)}{(u \cdot u)(v \cdot v) - (u \cdot v)^2}$$



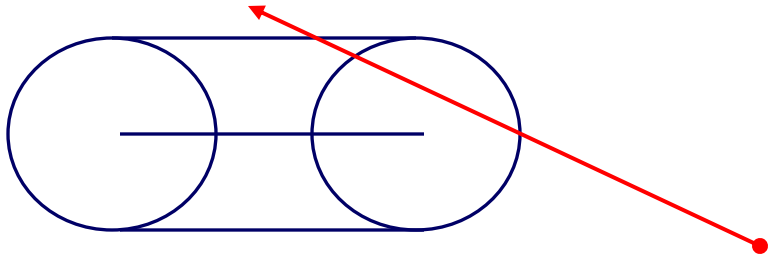
$$t_c = \frac{(u \cdot u)(v \cdot w_0) - (u \cdot v)(u \cdot w_0)}{(u \cdot u)(v \cdot v) - (u \cdot v)^2}$$

Swept Sphere Intersection

- Clamp resulting $P(s_c)$, $Q(t_c)$ to the end points of the two line segments.
- This gives you w_c. If its length is less than $r_1 + r_2$, then the swept spheres intersect.

Swept Sphere – Ray Intersection

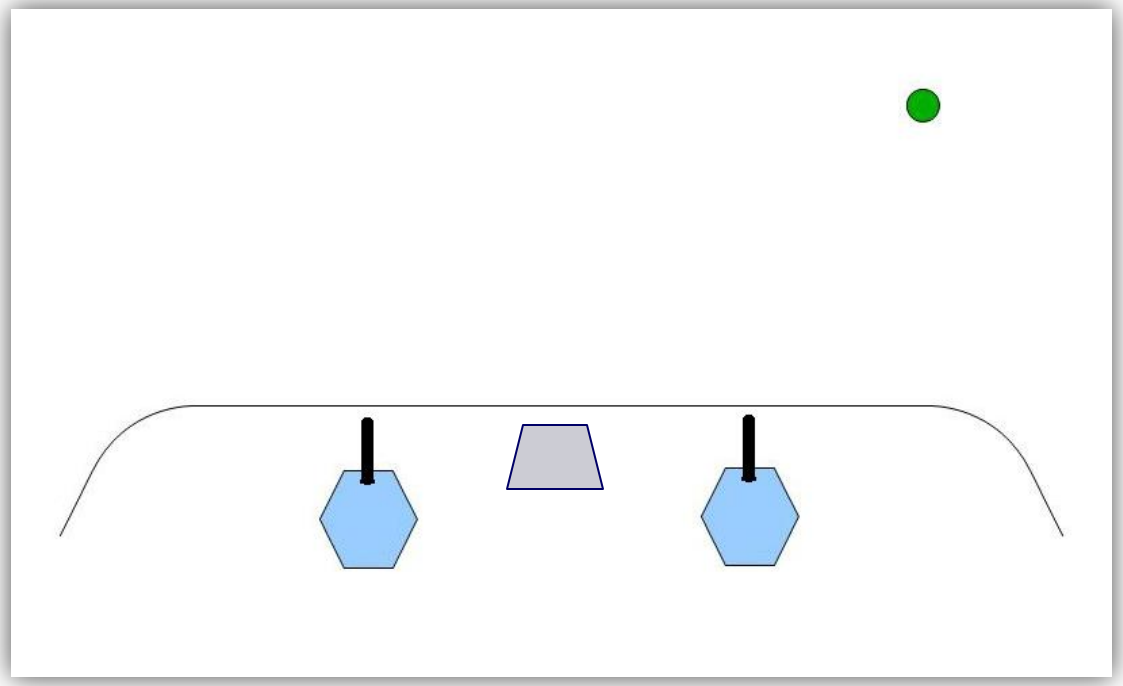
- Essentially the same as two swept spheres.
- Line to ray distance is calculated, then compared to the radius.



Object Intersection Wrapup

- Understanding basic collision/intersection techniques is still valuable.
- Gameplay and AI code will always have abstracted views of the game world.

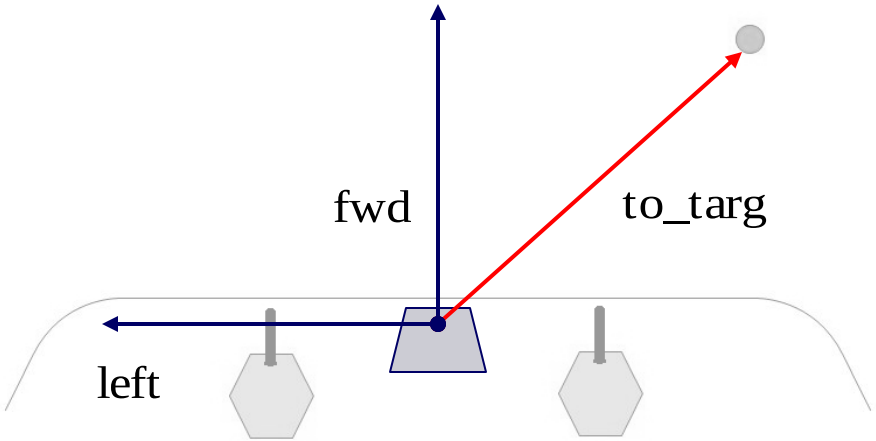
Example : Turrets on a Hill



Example : Turrets on a Hill

- Determine which turret should fire at the target
- Calculate turret facing
- Calculate angle of the gun barrel
- Convert to local object space for our engine.

Which turret should fire?



Which turret should fire?

- One easy way is to take the dot product of our to_targ vector and the bunker's left vector.

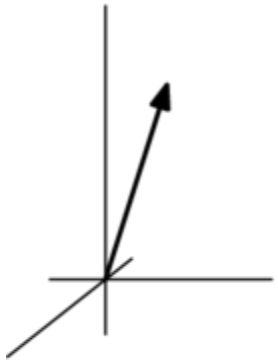
```
if (dot(to_targ, left) > 0.0)
        turret = left_turret
else
        turret = right_turret
```

Turret Facing

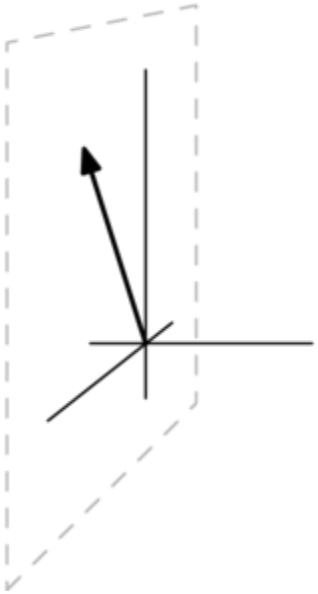
- Turret can only rotate horizontally
- We need the component of the vector from the gun barrel to the target that maps to the plane formed by the turret's fwd and left vectors.

Orthographic Projection

- Mapping of a 3D point to a 2D plane



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \\ 0 \\ z \end{pmatrix}$$



Source: http://en.wikibooks.org/wiki/Linear_Algebra/Topic:_Geometry_of_Linear_Maps

Orthographic Projection

- To project 3D point a_x, a_y, a_z onto a 2D point b_x, b_y, using a point parallel to the y axis:

$$\begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} c_x \\ c_z \end{bmatrix}$$

where *s* is scale, and *c* is an offset

Projection to a Plane

- To take the vector from the turret barrel to the target (v), and project it onto the turret's horizontal plane, where (n) is the unit normal to that plane (up vector in this case):

$$a = v \cdot n$$

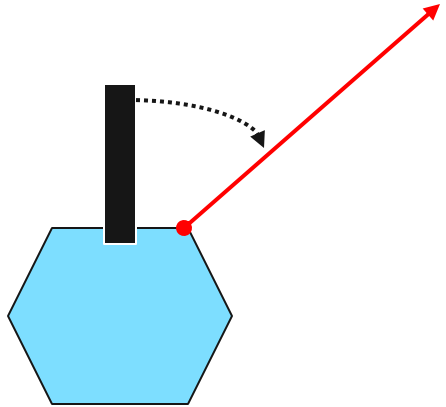
$$r_x = v_x - (n_x \times a)$$

$$r_y = v_y - (n_y \times a)$$

$$r_z = v_z - (n_z \times a)$$

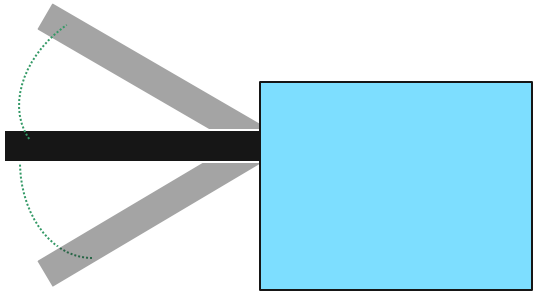
Turret Facing

- Now we have a vector along the plane the turret swivels in to use as desired facing.



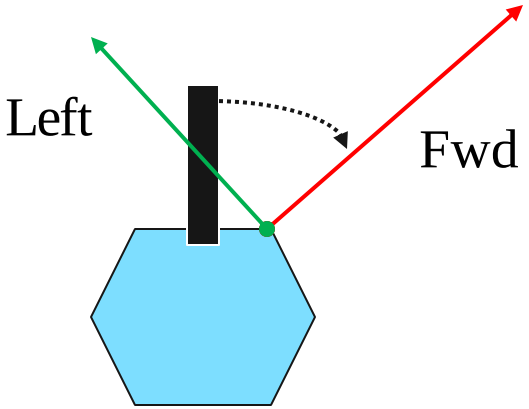
Gun Barrel Aiming

- Assuming that the gun barrel can only rotate up and down relative to the turret's position…



Gun Barrel Aiming

- We can isolate that desired vector by again projecting to a plane.
- The desired Left vector is the normal to this plane.



Projection to a Plane Redux

- We perform the same calculation using the desired left vector for the turret as n.

$$a = v \cdot n$$

$$r_x = v_x - (n_x \times a)$$

$$r_y = v_y - (n_y \times a)$$

$$r_z = v_z - (n_z \times a)$$

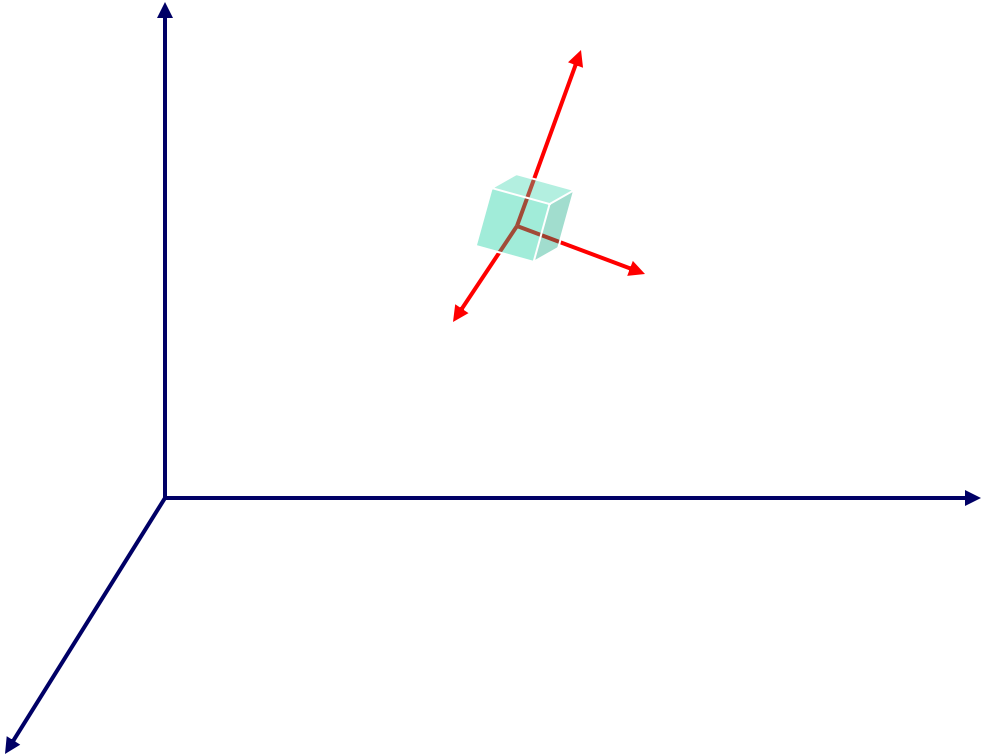
One More Step

- We now have desired turret facing and gun barrel pitch vectors, in world space.
- What if our engine requires them in local (object) space?

World -> Local Transform

- Very common in games to move back and forth between local and world coordinates.
- Local coordinates are the coordinate system where the center (usually) of an object is at (0,0,0), and the axes are usually aligned to the orientation of the object.

World and Local Coordinates



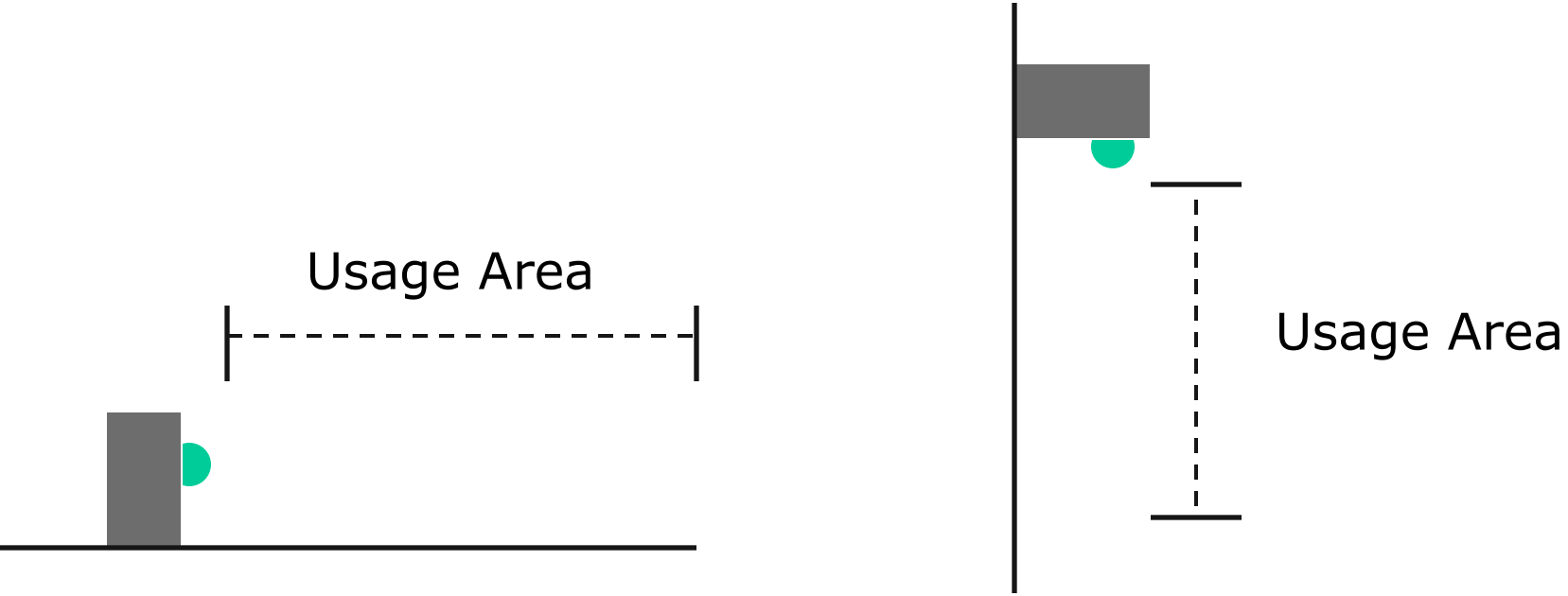
World ->Local Transform

- To transform to local coordinates, we need a 4x4 matrix specifying the object's axes and position.
- Multiplying a vector by this matrix will yield a vector transformed into local space.
- Use the inverse of the object's matrix to convert back to world space.

Why Local Space?

- Often easier to work with components of an object this way (ie. Joint hierarchies)
- AI and gameplay calculations will frequently be based on something's position relative to an object or actor.

Local Space AI Example

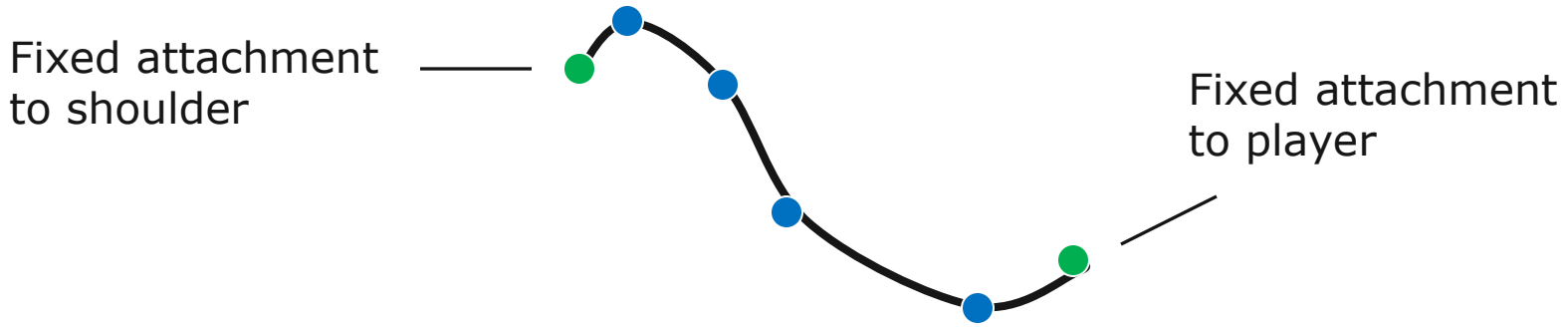


Interpolation

- Lerp and slerp have many more uses besides just generating curves.
- e.g. We can simulate the motor of our turret revving up by lerp-ing from the current position to the desired using an appropriate function.

Splines

- In *Ratchet & Clank : All 4 One*, there is an enemy with tentacle-style arms that can grab players. Splines are used to control the joint positions.



Example Wrapup

- Basic linear algebra is incredibly useful in games.
- In games, we frequently need to express our algorithms in terms of a planar projection or an object's local coordinate system.

Bayes' Theorem

- Named for Thomas Bayes, who used it to express subjective degrees of belief given pieces of evidence.
- Was first published in 1763, 2 years after his death.
- Did not gain widespread usage until the 20th century.

Bayes' Theorem

- A useful method for computing conditional probabilities
- For game AI, it can be used to weight decision-making in situations with limited information.

Bayes' Theorem

- The probability of event A1, given that event B has subsequently occurred, is:

$$P(A1 \mid B) = \frac{P(A1) \cdot P(B \mid A1)}{\left[P(A1) \cdot P(B \mid A1)\right] + \left[P(A2) \cdot P(B \mid A2)\right]}$$

Bayes' Theorem

- If we know something about the conditional relationship between A and B, we can speculate on the likelihood of A occurring if B already has.

Bayes' Theorem - Example

- Event A1 - an attendee falls asleep in a Math tutorial during a discussion of Bayes' Theorem
- Event A2 - an attendee remains awake
- Event B - a conference attendee is badly jet-lagged when attending the Math tutorial

Bayes' Theorem - Example

- Known probabilities:
  - P(A1) : 2% of attendees fall asleep
  - P(A2) : 98% of attendees remain awake
  - P (B | A1) : 90% of sleeping attendees are jet-lagged
  - P (B | A2) : 5% of awake attendees are jet-lagged

Bayes' Theorem - Example

- If we know an attendee is jet-lagged (event B), what is the probability of them falling asleep (event A1) while I am speaking?

Bayes' Theorem - Example

- P(A1) * P(B | A1) : 0.02 * 0.9 = 0.018
- P(A2) * P(B | A2) : 0.98 * 0.05 = 0.049

$$\frac{0.018}{0.018+0.049} = 0.2686 = 26.9\%$$

Bayes' Theorem Usages

- Useful in any situation with incomplete information:
  - In an RTS, to estimate enemy army compositions.
  - In a shooter, to predict enemy tactics
- Help eliminate "saddle points" in your AI

Fuzzy Logic

- A method that involves approximate reasoning, rather than strict true or false propositions.
- A given proposition has a degree of truth ranging from 0.0 to 1.0
- Deals in possibilities, not probabilities

Fuzzy Logic History

- First formulated in 1965 in a proposal on Fuzzy Set Theory by Lofti Zadeh.
- Has been successfully used in AI systems, particularly robotics and other hardware control systems.
- Good for games for the same reason it is good for robotics : it handles chaotic environments and incomplete information.

Fuzzy Sets

- Traditional sets have absolute membership: my ammo clip is either empty or full.
- A fuzzy set allows for degrees of membership. If I have 70% of my ammo, then we might say my clip has a 0.7 degree of membership in the Full set.

Fuzzy Sets

- Another key point is that fuzzy logic sets are not mutually exclusive: my ammo clip belongs to both the Full and Empty sets to some degree.

Fuzzy Logic in Game AI

- Can help us escape from hard “if-then-else” rules, which can break down given unexpected or rare input.
- Helps make an AI more likely to do something “reasonable” even in a very unlikely or unanticipated situation.

Fuzzy Logic in Game AI

- Consider the following rules:
  - Low on ammo : take cover and reload
  - Under heavy fire : take cover
  - Badly outnumbered: take cover
- In a traditional implementation, these are each absolutes, that need to be satisfied completely in order to be processed.

Fuzzy Logic in Game AI

- A Fuzzy approach would be：
  - Low ammo factor + taking fire factor + outnumbered factor = weight of taking cover
- We can weight our entire action set this way, and choose a good course of action.

References

- Van Verth, Jim & Bishop, Lars. *Essential Mathematics for Games And Interactive Applications*. Morgan Kaufman, 2004
- Yudkowsky, Elizier. *An Intuitive Explanation of Bayes' Theorem*. http://yudkowsky.net/rational/bayes
- Wolfram MathWorld. http://mathworld.wolfram.com
- Kaehler, Steven. *Fuzzy Logic – An Introduction*. http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html