Research compilation

The model used before BOIDs was the localised flock centering behaviour.

Craig Reynolds & BOIDS (1986)

https://dl.acm.org/doi/abs/10.1145/37401.37406

Approach

- Define how a bird moves and flies when next to other birds
- Make a bunch of birds
- Huge correlation to the theory of emergent systems and behaviours.

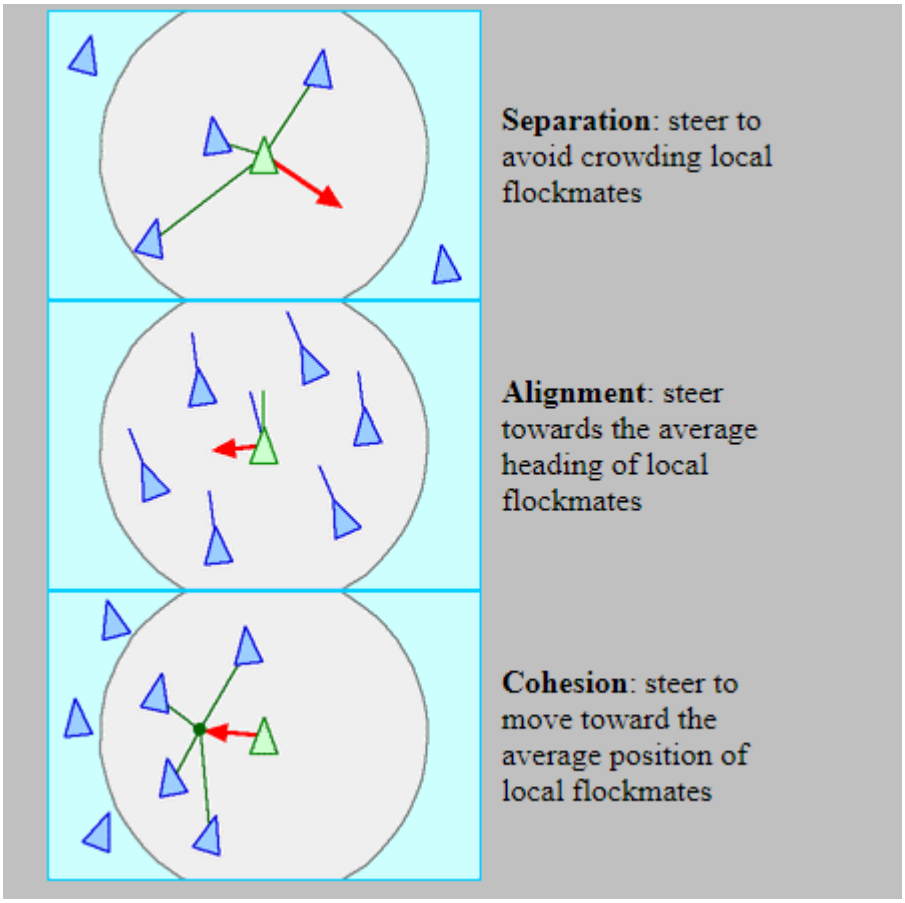
Individually simulate each member of the flock, and when each individual was simulated, complex flocking behaviour arose naturally

Individual movement

- Usually define a minimum speed
- Also usually device a maximum acceleration, to provide smooth movement and more realistic movement too.
- *Constant time algorithml*: The amount of “thinking” a bird has to do is largely independent of the number of birds in the flock.

In order of decreasing precedence, the behaviours that lead to simulated flocking are:

1. **Collision Avoidance:** avoid collisions with nearby flockmates
2. **Velocity Matching:** attempt to match **velocity** (heading and speed) with nearby flockmates.
    - Additionally, by trying to match the velocity of its neighbours, it also decreases the chance of collision.
    - Velocity matching is based only on velocity and ignores position.
3. **Flock Centering:** Attempt to stay close to nearby flockmates
    - BOIDS only have limited, localised perception. “Center of the flock” actually means centre of nearby flockmates.



Actually deciding where to go

- Each of the 3 aspects submits an **acceleration request**, “if I were in charge i’d steer here.”
- A central management **navigation module** takes in these requests, determines the **strength and prioritisation** of each one, and based on that decides where to go.
- Easiest way to combine them is by using a *weighted average* system. But this comes wither problems
- Real solution is **Prioritised acceleration allocation** based on a strict priority ordering of all component behaviours.
- Basically, goes “Okay, I have [maximum acceleration] amount of speed to spend,” I’m going to spend acceleration to fulfil priority 1. If i have leftover acceleration, I fulfil priority 2, and so on and so forth

Field of vision

- Current implementation of the neighbourhood is defined as a spherical zone of sensitivity. A more realistic model would probably have forwards weighted perception and sensitivity, but that’s quite computationally complex for not enough benefit
- How much a bird in the neighbourhood affects movement was weighted based on the inverse square of the distance.

Scripted Flocking

David Hahn flock simulation

https://ddavidhahn.github.io/cs184_projfinal/

- Boids models
- Contains the details on **Avoidance and attraction Entities**
- Uses a 4th factor called Cruise from Emilee Chen's project

Emilee Chen & Kelly Cho 2D Fish flocking simulator

https://www.emileechen.com/projects/flocking/

- Smoother than the david hahn solution
- Uses the two papers after this as a basis

Cruise Speed

- Similar to Craig's idea of minimum and maximum acceleration. Cruise speed is a force based on all the other forces that keeps the agent from moving too slow or too fast.
- Uses **weighted forces** with **different radiuses of perception**.
- Consider adding blind spots to the perception cones.
- The alignment in **fish specifically has blind spots in the front and back**
- They also recalculate the radius of all fields based on their density. The radii are calculated as a linear interpolation of the previous radii and a newly calculated radii
- **Cohesion and alignment are recalculated, but separation stays constant**.

```
s = smoothness * Δt;                                  // Weight for interpolation
d = max_radius - (influence * num_neighbors(t));      // Density dependent radius
radius(t + Δt) = max(min_radius, ((1 - s) * radius(t)) + (s * d));
```

The smoothness value determines how quickly the radius is allowed to change per calculation by prioritizing either the current radius or the calculated density dependent radius. The smaller the smoothness value, the smoother the change in radius.

The influence value determines how many neighbours it requires for the radius to constrict. The larger the influence value, the more quickly the radius reduces. If this value is too large, then the agents display a swirling behaviour since it effectively turns off alignment.

Self-Organized Shape and frontal density of fish schools

https://pure.rug.nl/ws/files/86301693/Hemelrijk_et_al_2008_Ethology.pdf

- Also builds of the 3 concepts of the BOIDS model
- Where cruise speed originated from

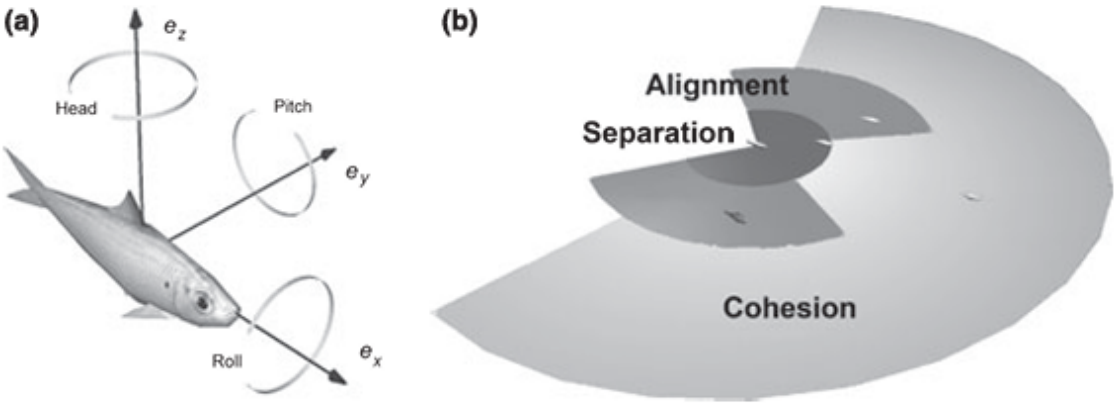
Cruise Speed

- “Preferred moving speed” with lower and upper variation limits for speeding/slowing down
- Normal fish cruise spelled: 2 Body Lengths per second (slow) and 4 BL/s (fast)
- Variable radius is also introduced here. Meant to mimic the fact that the more densely populated the neighbourhood is, the harder it is to see other fish further away from them because they are blocked.

Note: adding radius recalculation for all individuals might be expensive

Fish

- Individual fish are characterised by its position, its speed, and orientation (roll, pitch, and head)
- Movement is calculated from speed and orientation



- When recalculating radii, alignment range cant be smaller than separation range

smaller than the separation range R_{min}. The new perception radius, $R(t+\Delta t)$, is calculated as linear interpolation of the current radius $R(t)$ and a density-dependent term:

$$R_i' = R_{max} - n_w \cdot n(t)$$

$$R_i(t+\Delta t) = max\{R_{min}, (1-s)\cdot R(t) + s.R'\};\ s = n_i \cdot \Delta t \quad (1)$$

where n(t) is the number of perceived neighbours at time t. The parameter n_w indicates the influence of a single neighbour and n_i controls the smoothness of the radius adaptation. Note that this adaptable view is supported by empirical evidence for a relatively fixed number of interactants in starling flocks (Ballerini et al. 2007).

The full steering behaviour

- Consists of the sum of the three steering forces plus additional variables for the control of speed (cruise), the correction of unrealistic acceleration, and random noise
- Random noise can probably be taken out.
- The paper **has all the equations for calculating separation, alignment, and cohesion**
- Seems to also use weighted average in its calculations?

Cruise speed

- The three steering forces also have a length, which may cause individuals to slow down or speed up. However, each individual prefers to swim at cruise speed. Deviations from this are reduced by a compensating force.

Fish movement

- During migration real fish do not show large pitch angles over longer periods and they virtually never roll
- If, once the sum of all forces is added, it exceeds a maximum magnitude, it is rescaled to the maximum magnitude of acceleration (3 BL per second^2)

- Look into redoing all the fish calculations every other frame. **Or also just on fixed update.**

Table 1: Summary of model parameters. The unit of length is one body length (BL) and the unit of mass is one body mass (BM)

Parameter	Unit	Symbol	Value(s) explored
Number of individuals	1	N	10-2000
Time step	s	Δt	0.05
Zone of separation			
Radius	BL	R_{min}	2
Blind angle back	Degrees	–	60
Zone of alignment			
Maximum Radius	BL	–	5 (adaptive)
Blind angle back	Degrees	–	60
Blind angle front	Degrees	–	60
Zone of cohesion			
Maximum Radius	BL	R_{max}	15 (adaptive)
Blind angle back	Degrees	–	90
Cruise speed	BL/s	v_0	2; 4
Weights			
Separation	$BL^2 \cdot BM/s^2$	w_s	10
Alignment	BM/s^2	w_a	5
Cohesion	BM/s^2	w_c	9
Relaxation time	s	τ	0.2
Pitch control	$BL^2 \cdot BM/s^2$	w_{pc}	2
Roll control	$BL^2 \cdot BM/s^2$	w_{rc}	5
Random noise	$BL^2 \cdot BM/s^2$	$\|f_\xi\|$	0.5
Max. force	$BL^2 \cdot BM/s^2$	f_{max}	3

Efficient algorithms for finding neighbours

- This model uses the Holbert R-Tree and optimised for dynamic data
- Kamel & Faloutsos

- **Look into more efficient ways of finding neighbours.**

Results of this particular model:

- Larger groups are expected to be denser than smaller ones
- Larger groups are mode oblong and their core is located more towards the front
- Slower schools are less polarised, more oblong, and their tail is denser than their core compared to fast schools

Steering Behaviours for autonomous characters (Reynolds Again)

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9d19157fa8da0a7d216f44d6a45a73b59b6da23f

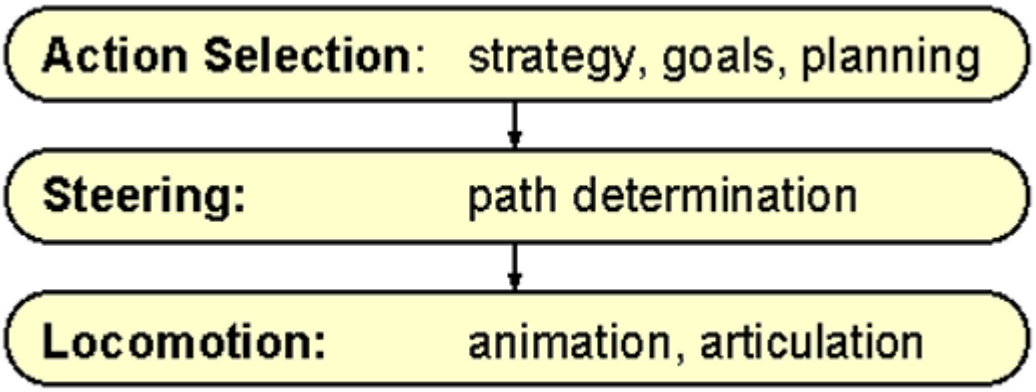


Figure 1: A hierarchy of motion behaviors

Locomotion:
- Represents how the directions from the steering layer are converted into motion of the character's body

The Coding Train - Explaining Reynolds steering code

https://www.youtube.com/playlist?list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvlW

- Daniel Shiffman
- Coding Professor at the Interactive Telecommunications Program at NYU's Tisch School of the Arts
- Serves on the Board of Directors of the Processing Foundation
- https://thecodingtrain.com/about

6.1: Autonomous Agents and Steering

https://www.youtube.com/watch?v=JIz2L4tn5kM&list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvlW

Autonomous Agent
- Limited ability to perceive its environment.
- Process the environment & calculate an action
- No global plan/leader (not a hard rule)

Vehicle 3 steps

- Action/Selection: Choosing a desired velocity
- Steering: Calculated Force based on the desired velocity
- Locomotion: Fulfilling that force based on movement scheme

6.2: Steering Behaviours: Seek

https://www.youtube.com/watch?v=4zhJlkGQTvU&list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvlW&index=2

Steering

Steering Force Calculation:

Steering = Desired - Velocity

- Desired: The path to the target. **Vector from unit location to target location**. The **length** of that vector is the **maximum speed** of the unit
- Velocity: **Current velocity** of the object
- This vector then gets **normalizes and scaled to max speed**

Attraction behaviour:

- One object is attracted to another

Pseudocode:

```
Class Vehicle {
 Float maxSpeed;
  //Maximum Turning speed

Pvector desired = PVector.sub(target, location); //Gets desired

Desired.setMag (maxSpeed) //scales it

PVector steering = PVector.sub(desired, velocity)
Steering.limit(maxForce)
}
```

6.3: Steering Behaviours: Arrive

Arrival

- Slow down when approaching target and stop when on target

Steps:

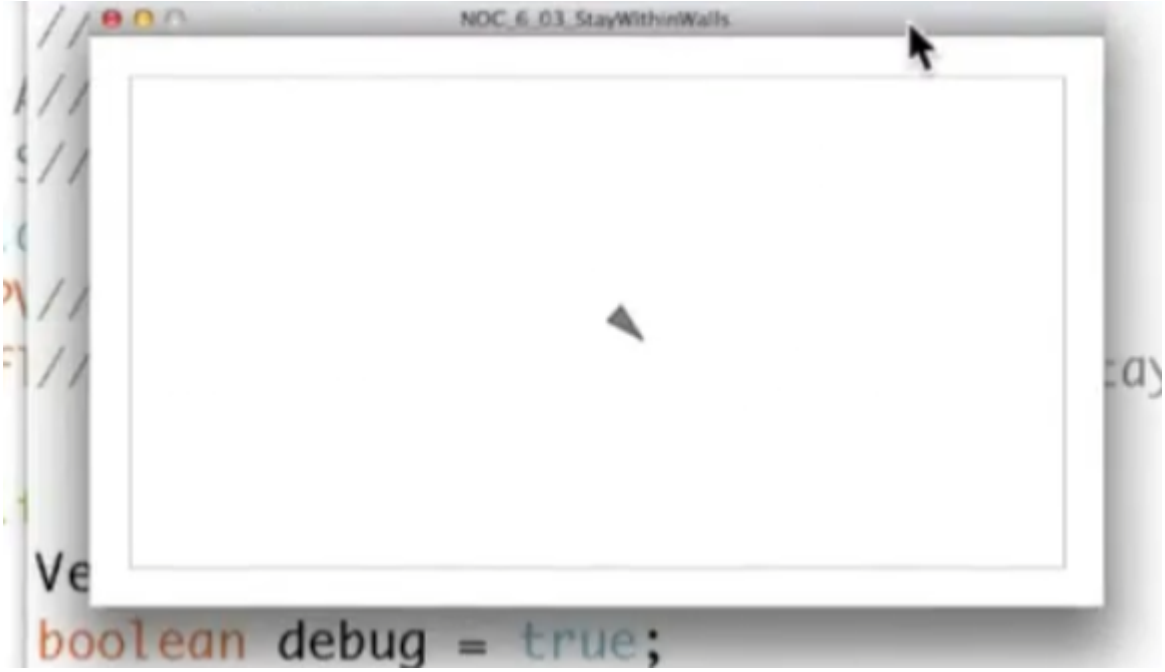
- Define a radius from the target where you want to start slowing down

- Desired speed is maxSpeed at the edge of the radius. At the very centre, the maxSpeed is 0

- Get **Distance**: Between location and target.
- Float speed = **map (d, 0, 100, 0, maxSpeed);** //maps distance (d), based on the range of 1-100, and the value of 0 - maxSpeed.
- Unity equivalent might be this: https://docs.unity3d.com/Packages/com.unity.mathematics@1.2/api/Unity.Mathematics.math.remap.html

Useful Example

- Look at example 6_03_StayWithinWalls



6.4: Steering Behaviours: Flow Field Following

https://www.youtube.com/watch?v=XXEK3UEDDIg&list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvIW&index=4

- Flow fields, seems mostly irrelevant.
- Can be used to apply **Perlin Noise** randomization
- Or used to replicate the **flowing path of a river** and have things flow around a static object.

- In the 3D space, this flow field would have to be implemented with a 3-dimensional array of vectors.

Maths and finding out which part of the grid you're on

- Finding out which grid space you're on can be seen in 5:30
- After you have the grid space (Ex., 4, 2), you just look up column 4 and row 2 data in some Lookup table stored somewhere.

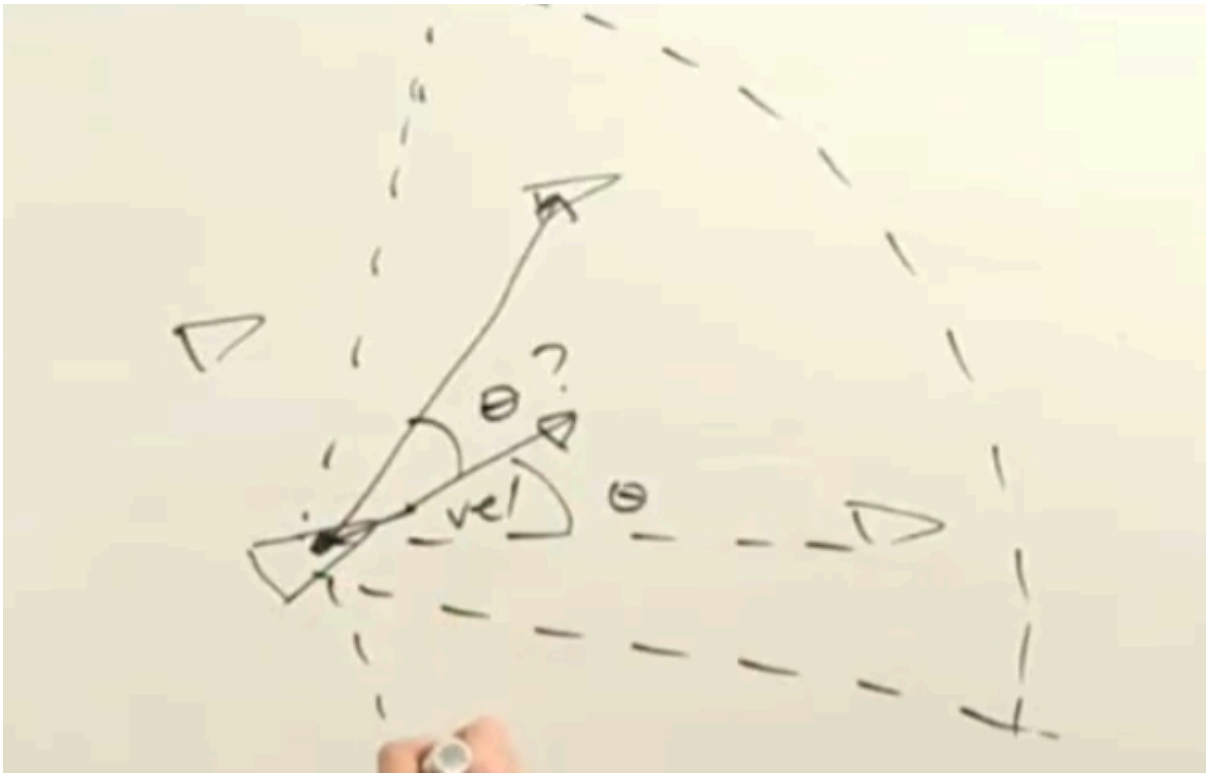
Calculating the Flow Field

6.5: Vectors: Dot Product and Scalar Projection

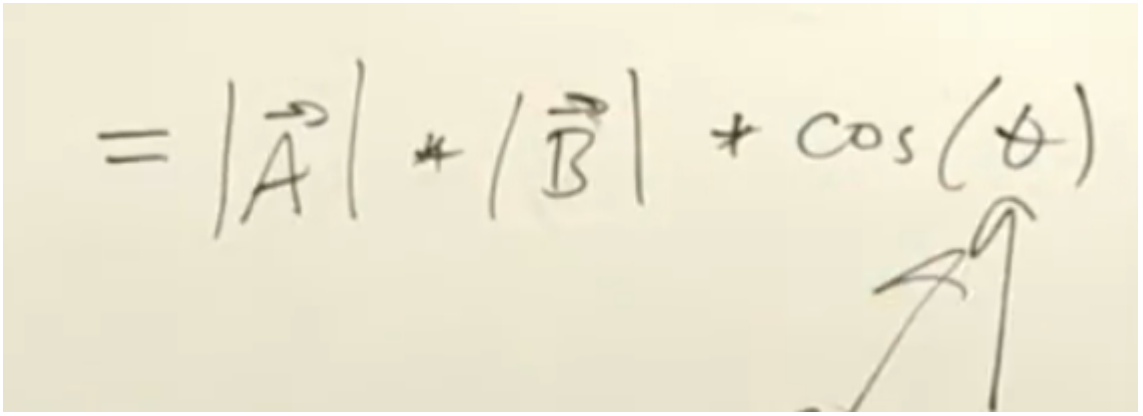
https://www.youtube.com/watch?v=_ENEsV_kNx8&list=PLRqwX-V7Uu6YHt0dtyf4uiw8tKOxQLvlW&index=5

Why do we need the dot product?

- **Modelling Vision**: How can we tell what other vehicles one can see.
- If you get the **angle between** the unit's **current movement vector**, and the **vector from this unit to another one**, you can then see if that angle is within a certain range and is there for "within the cone of vision"
- The Dot product **gives the angle between two vectors**

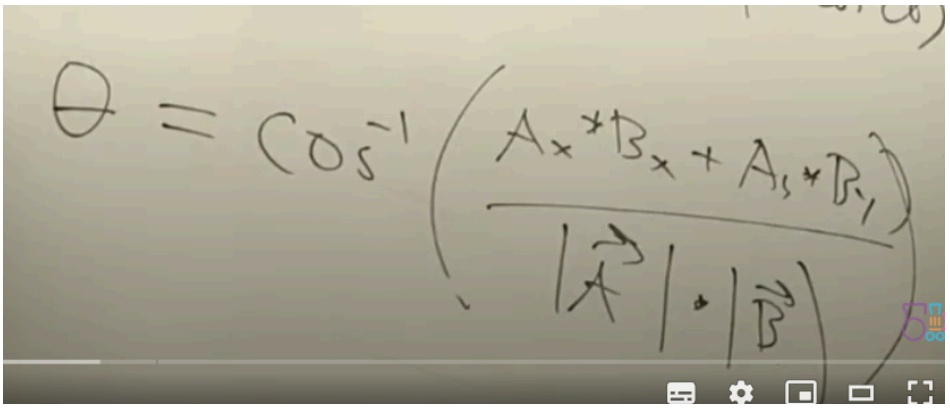


Dot Product Formula:



Magnitude of A * Magnitude of B * Cos(theta)

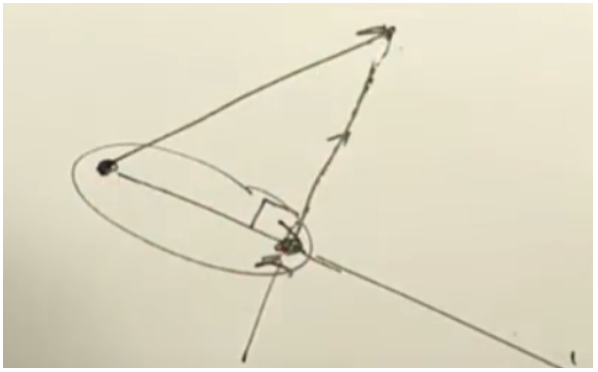
Getting Theta:



- **Also just use unity's dot product/angle between function**

Scalar Projection

- The distance between a point and a line. However, specifically, **the shortest distance between a point and a line**.
- The resulting vector forms a right angle with the line
- **The scalar projection is actually that segmented part of the line**



6.6 Steering Behaviours: Path Following

Making something follow a path:

Craig Steps:
1. Check Vehicle's future location
    - Use it's current velocity to calculate where it would be __ frames later

2. Think: Is that future location on the path?
    - if(yes) -> Do nothing
    - if(no) -> find the **normal** to the path (closest point on the path)
    - (cont) -> move along the path a little bit & **seek that target**

How do we know a future location on a path?
- Take that projection, the normal.
- See if the **distance/magnitude** is greater or less than the radius of the path
- Check the video for a function for getting the normal

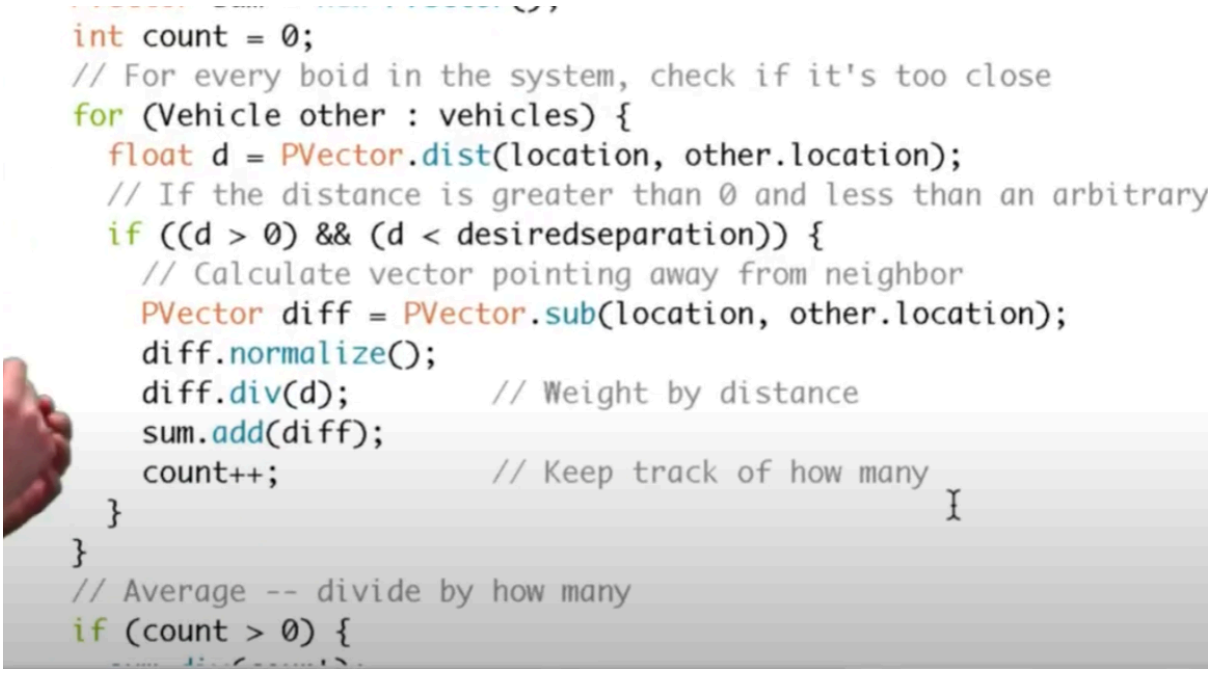
Curved line
- Get the normal between all segments
- Follow the shortest distance one

6.7: Group Steering Behaviours

- YEAHHH BABY FLOCK AND SWARMING

Alignment Code
- Hey, lemme start with a sum of 0
- Let me look at every other vehicle and add their velocity to my sum
- And then i will divide that number by the total amount of vehicles
- **Yes, we got the average velocity of all things around us**
- And now **I** want to move in that direction
- Can add additional maths to also scale it by the other people's average speed

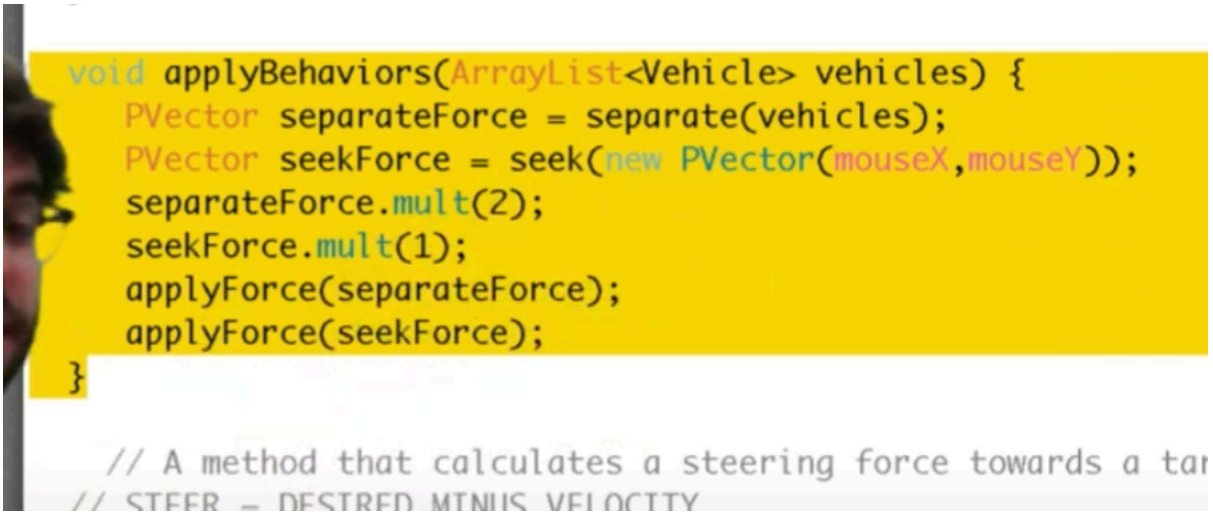
Separation



```
int count = 0;
// For every boid in the system, check if it's too close
for (Vehicle other : vehicles) {
  float d = PVector.dist(location, other.location);
  // If the distance is greater than 0 and less than an arbitrary
  if ((d > 0) && (d < desiredseparation)) {
    // Calculate vector pointing away from neighbor
    PVector diff = PVector.sub(location, other.location);
    diff.normalize();
    diff.div(d);        // Weight by distance
    sum.add(diff);
    count++;            // Keep track of how many
  }
}
// Average -- divide by how many
if (count > 0) {
```

6.8: Combining Steering Behaviours

- Weights!!



```
void applyBehaviors(ArrayList<Vehicle> vehicles) {
  PVector separateForce = separate(vehicles);
  PVector seekForce = seek(new PVector(mouseX,mouseY));
  separateForce.mult(2);
  seekForce.mult(1);
  applyForce(separateForce);
  applyForce(seekForce);
}

  // A method that calculates a steering force towards a tar
// STEER = DESIRED MINUS VELOCITY
```

- “The computational beauty of nature” by Garry William Flake
- Talks about bird-specific flocking

Big 1000 Page book (2019)

https://learning.oreilly.com/library/view/ai-for-games/9781351053280/xhtml/08_Chapter01.xhtml#ch1-1-2

1.1.2 Game AI

- Literally just regurgitates steering behaviours.
- Has pseudocode for all steering behaviours.
- Games that rely on physics engines typically include drag instead of a maximum speed
- Also breaks down and has pseudocode for alignment, separation, and cohesion
- Although, **details a wandering behaviour**.
- Chapter of obstacle and wall avoidance might also be of interest.
- Uses the multi-case raycast for collisions detection.

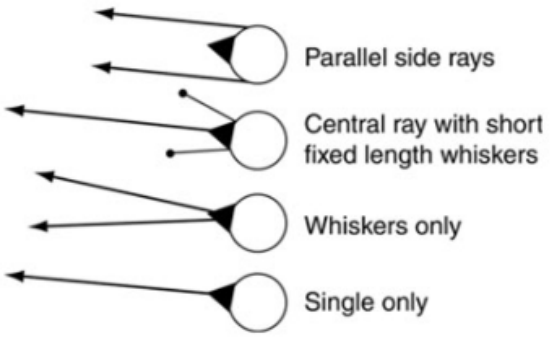
3.3.14 Collision Avoidance:

- Simple approach: use a variation of the evade or separation behaviour. If the object, can “see” the object, apply separation.
- However, this doesn't really work when there’s particularly a large amount of obstacles.

Obstacle and wall avoidance

- Demonstrates the multiple raycasting method of collision detection
- Offers many different raycast options and lists their pros and cons

---



- One thing to look into is the fan angle and number of raycasts, some developers make the fanning to change dynamically based on collisions in recent memory.

- Another solution is to perform the collision detection using a projected volume rather than a ray
- For now, the best solution with the most bang for your buck, seems to be **adaptive fan angles with one long ray cast and two shorter whiskers**.

Movement in narrow spaces

- Steering behaviours fall short when navigating tight, narrow spaces, best with open worlds and levels
- Only way to get around this limitation is to integrate pathfinding ai into the thing

Behaviour priority vs. behaviour blending

- Pure vector blending is problematic because sometimes they cancel each other out. Or, because a vector gets diluted, maybe a fish cant turn fast enough to avoid danger.
- So, consider a priority system instead of a blending system.
- A priority system places the different acceleration requests in a priority queue
- Similar to what i read in the original craig article. You execute the first priority, then if it is below a certain acceleration threshold, you do the next.

Deciding how to blend or prioritise things

- Depends. There's many ways with different pros and cons depending on the aim.
- Decision trees (most common), state machines, etc. or also a cooperative arbitration approach.

Uniform spatial subdivision to improve Boids algorithm in a gaming environment

https://www.ijarnd.com/manuscripts/v3i10/V3I10-1144.pdf

- The algorithm is effective, but is very time consuming due to how the algorithm evaluates the whole crowd when searching for possible nearest neighbours
- The most computationally expensive part of BOIDS is finding nearby neighbours. A brute-force approach makes it very hard to simulate large crowds with a good frame rate. Boids is the backbone for most algorithms in crowd simulation (Sajwan, Gosain & Surani, 2014).

Base implementation of Boids

- O(n^2) time complexity. Bad, awful, cubic time complexity as the direct result of near-neighbour queries (Silva, et al., 2009)
- This si because for each agent to determine its proximity from neighbours, it has to compare its distance from every other agent's position in the scene.

Crowd Simulation Models

- Either microscopic or macroscopic.
- Macroscopic: threat the crowd as a while and use flows to characterise crowd behaviour and movement
- Microscopic: consider movement and behaviour individually for each agent

Microscopic models:

- Each agent is regarded as an independent decision making entity. Agents reevaluate their surroundings in real time. (Fachada, Lopes, Martins & rosa 2016)

Boids

- “The advantage of the boids algorithm over most of contemporary simulation algorithms is it’s ease of implementation whilst providing highly realistic results (Afanasyeva, 2014).

Parallel Processing:

- Breaking down the computations so that they can be processes on different cores in a parallel manner (Fachada et al., 2016).
- One can use multi-threading as a way of performing these computations on different processor cores
- Another can use either GPGPU or modern heterogeneous systems such as NVisia’s Compute Unit Device Architecture.
- Drozd, 2015, made a simulation where agents used information from their surrounding environment to performs gives tasks such as food foraging. By exploiting multi core CPU’s for parallel processing, he could simulate signal propagation for over 10,000 agents at interactive frame rates.
- **Parallel programming**.

Occlusion Culling

- Silva et al, 2009
- Boids must only query near neighbour information from other boids that are visible.
- Ignores all boids that would be invisible due to the presence of other boids, using a visibility culling algorithm.
- GPU only technique tho

Spatial Subdivision.

- Spatial subdivision: structural partitioning of geometry (Rhodes, 2014).
- General optimization technique that has been applied to several computational problems like ray tracing and cloth simulation.
- Subdivide a given geometrical space into smaller sub-spaces or cells, which can then be used to speed up proximity or locality queries.

Object independent spatial structures:

- Do not take into account the distribution of objects in space. Regular grids and the bin-lattice method are examples of this.
- Bin-Lattice (Reynolds, 2006): space is subdivided into smaller box called bins. Agents are assigned bins based on their position at the start of a simulation. Every time they move, they keep track if which bin they’re in. Near neighbour queries will be limited to members of the same and adjacent bins. Reduced time complexity from O(n) to O (kn)
- Uniform Spatial Subdivisions: Partition the space into many uniform cells, then perform tests only for primitive pairs belonging to the same call (ho et al., 2012). But it does not adapt with the distribution of objects in a scene like quadtrees.

- Good, but, the manner in which objects are distributed over the grid is a major determination of success (Rhodes, 2014). Worst case is the whole population being crammed within a single cell on the grid

This research:

- Use a goal-driven boids model. A boid, aside from regular boid things, is also driven by a set of goals that avoid it from moving into cells that are already densely allocated.
- They assigned a maximum capacity that each cell can hold.
- It worked

Object dependent spatial structures:

- Considers the distribution of objects in space, which result in irregular subdivision.
- Quadtrees, Octrees, and (hughex et-al, 2013).
- Eliminates large portions of the flock from near neighbour considerations. However, building these quadtrees can itself be complex and expensive.
- Octrees seem to have better results

Downsides of parallel programming in BOIDS:

- The effective way in which the model must be represented in order for parallelism to work (Fachada et al). Since parallel processing requires us to decompose the model into several components that can be processed independently, Fachada argued that model decomposition can introduce changes that modify the model dynamid due to implementation details.

Aside: Parallel programming and threads
https://stackoverflow.com/questions/806499/threading-vs-parallelism-how-do-they-differ

- Threads have to be used to take advantage of multiple cores from the software.
- Parallel programming HAS to use threads to take advantage of multiple cores. But if you use multiple threads on a single-core thing, it is not parallel programming.
- Parallelism" multiple threads are running at any given time. Do one thing in less time.

Dynamic Organization of flocking behaviors in a large-scale boids model,
https://link.springer.com/content/pdf/10.1007/s42001-019-00037-9.pdf

Dynamic organisation of flocking behaviours in a large-scale boids model

https://link.springer.com/content/pdf/10.1007/s42001-019-00037-9.pdf

- Paper of people that used BOIDs but with parallel programming on a computer cluster AND Object independent spatial subdivision
- Used DBSCAN, a density-based clustering algorithm that

Things that have used boids
- half-life

https://www.linkedin.com/pulse/birds-crowd-simulation-hitman-absolution-may-broome

- hitman : Absolution
- Abzu: Additionally: the “the making of the art of abzu” might help me run the thing with less frames too. https://medium.com/fragmentblog/simulating-flocking-with-the-boids-algorithm-92aef51b9e00

https://youtu.be/bqtqltqcQhw?si=8uhFAdb2PshkyEeu

https://ieeexplore.ieee.org/abstract/document/10051636?casa_token=wGvi44xRedgAAAAA:4ai6rC0PVe542ON8jwj8i1jG_8HF51-znDGgRGE2tJGCJvITe2fojtb2BVOhrSkb_AAugmKvlkw

- Paper that explores deep learning to improve obstacle collision in aerial vehicles that use boids.

INTERVIEW OF THE ABZU DEVS AND HE TALKED ABOUT USING FLOCKING

https://www.gamedeveloper.com/design/-i-abzu-i-and-the-challenge-of-conveying-emotions-through-game-design

- BOIDS
- Used uniform subdivision to make it easier to find neighbours.
- Also memory manipulation

2012 GDC Presentation about crowds in hitman: absolution

https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2012/slides/Programming%20Track/Fauerby_Kasper_CrowdsInHitman.pdf

Visibility Culling
https://www.researchgate.net/publication/220686531_Boids_that_see_Using_self-occlusion_for_simulating_large_groups_on_GPUs

- Visibility is used to avoid computing the influence of occluded boids

Craig Reynolds and Bin-Lattice
https://www.red3d.com/cwr/papers/2000/pip.pdf

Devlin, Quadtrees for boids
https://nccastaff.bournemouth.ac.uk/jmacey/MastersProject/MSc16/05/thesis.pdf

Yilmaz & GPU processing
https://open.metu.edu.tr/handle/11511/19468