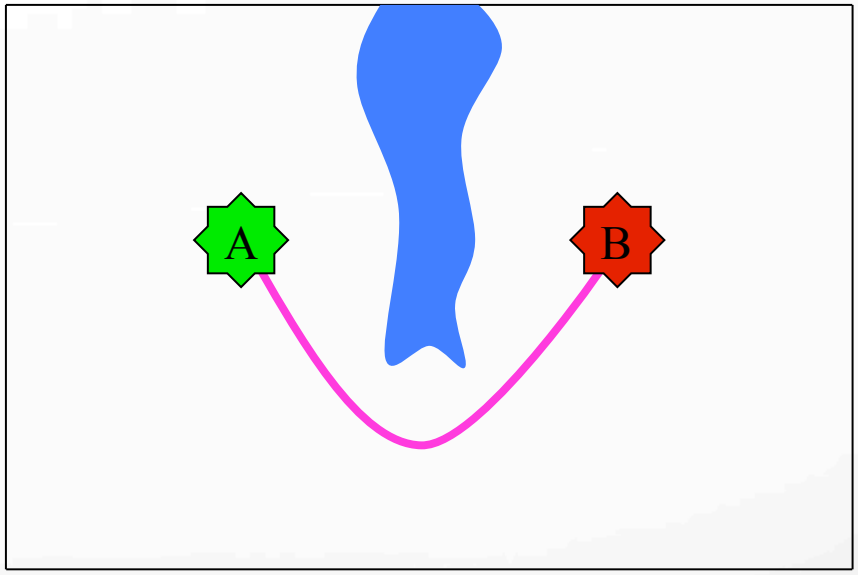
Guest Lecture:

Steering in Games

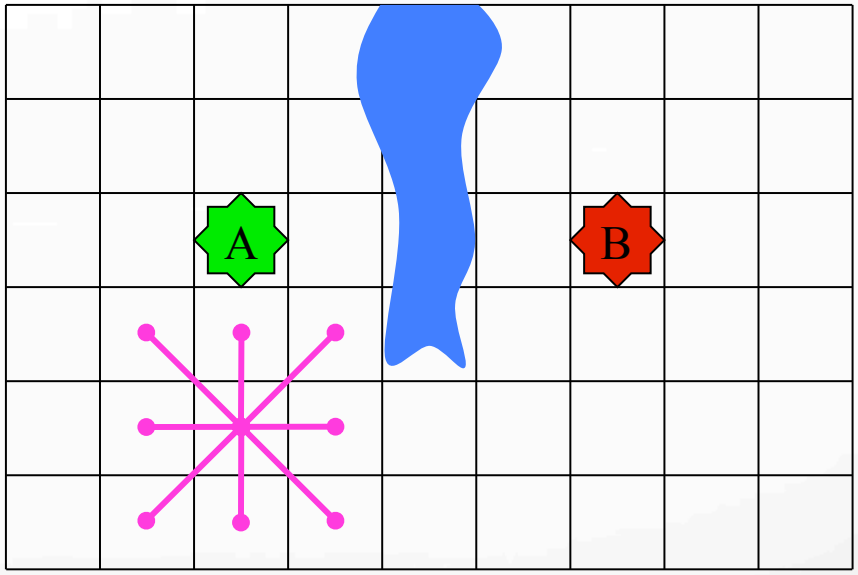
Pathfinding

- You are given
  - Starting location *A*
  - Goal location *B*
- Want **valid** path *A* to *B*
  - Avoid "impassible" terrain
  - Eschew hidden knowledge
- Want **natural** path *A* to *B*
  - Reasonably short path
  - Avoid unnecessary turns
  - Avoid threats in the way



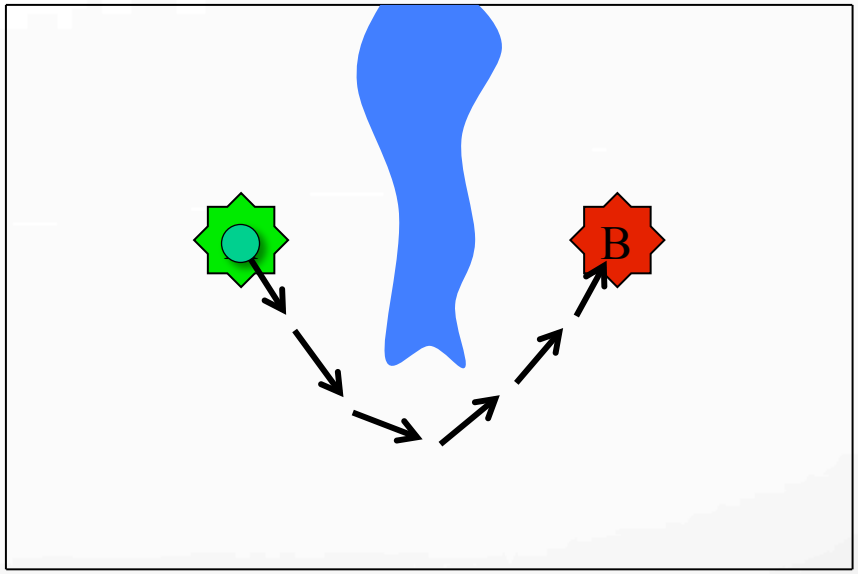
Pathfinding as Search

- Break world into grid
  - Roughly size of NPCs
  - Terrain is all-or-nothing
    - Majority terrain of square
    - Terrain covering "center"
- Gives us a weighted graph
  - Nodes are grid centers
  - Each node has 8 neighbors
  - Weight = distance/terrain
- Search for shortest path!



Steering

- Uses **physics**, not **search**
  - Use forces to get velocity
  - Goal pulls object towards
  - Obstacles push object away
- Good as "local" search
  - Just trying to find next step
  - Recomputed every step
  - Not great for long paths
- **Why would you do this?**



Which One is Best?

Pathfinding

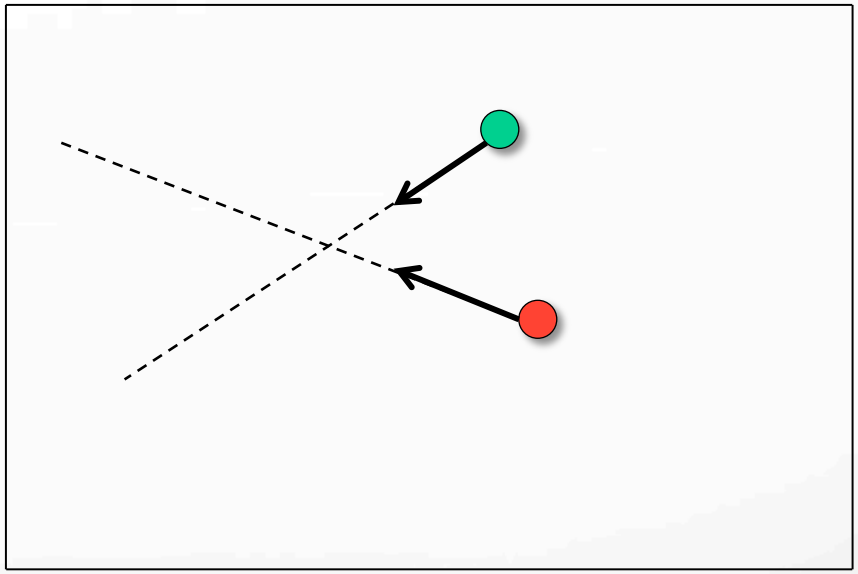
- **Cost** depends on grid size
  - Grid provides terrain graph
  - Algorithms are expensive
- Ideal for static **terrain**
  - Finds complex, winding paths
  - Compute the path only once
  - Recompute if obstacles move
- Ideal for coordinated **NPCs**
  - Use "military formations"

Steering

- **Cost** depends on # obstacles
  - Must look at all obstacles
  - Distance to obstacle not issue
- Ideal for dynamic **terrain**
  - Only finds the next step
  - Recomputed every time step
  - Reacts to moving obstacles
- Ideal for uncoordinated **NPCs**
  - Treat other NPCs as obstacles

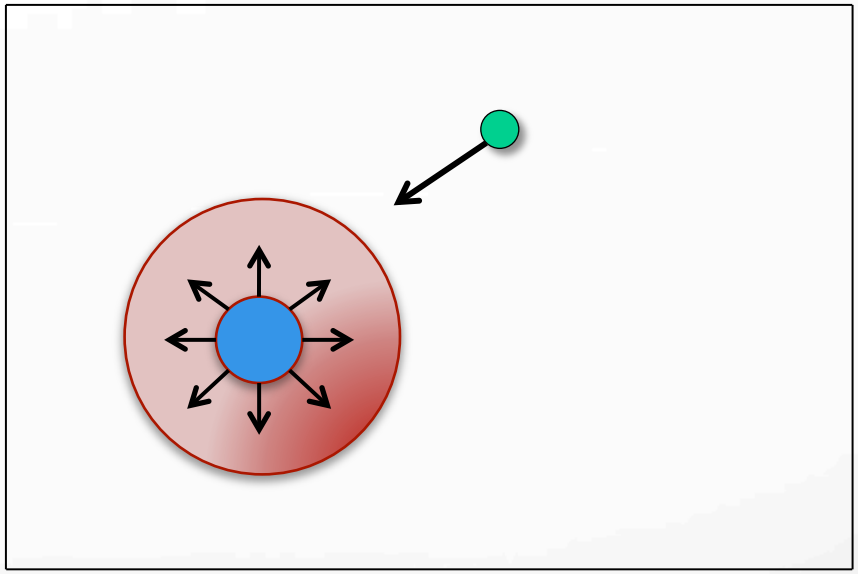
Types of Steering

- **Ad hoc methods**
  - Move in straight line
  - If possible collision, move
- Artificial potential fields
  - Obstacles have "charges"
  - Creates electrical field
  - Natural path through field
- Vortex fields
  - Obstacles have "currents"
  - Push around obstacles



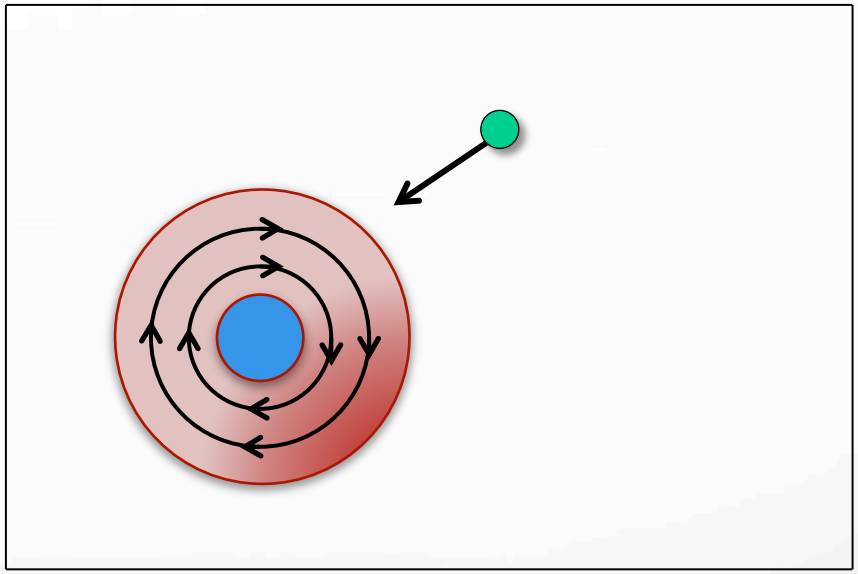
Types of Steering

- Ad hoc methods
  - Move in straight line
  - If possible collision, move
- **Artificial potential fields**
  - Obstacles have "charges"
  - Creates electrical field
  - Natural path through field
- Vortex fields
  - Obstacles have "currents"
  - Push around obstacles



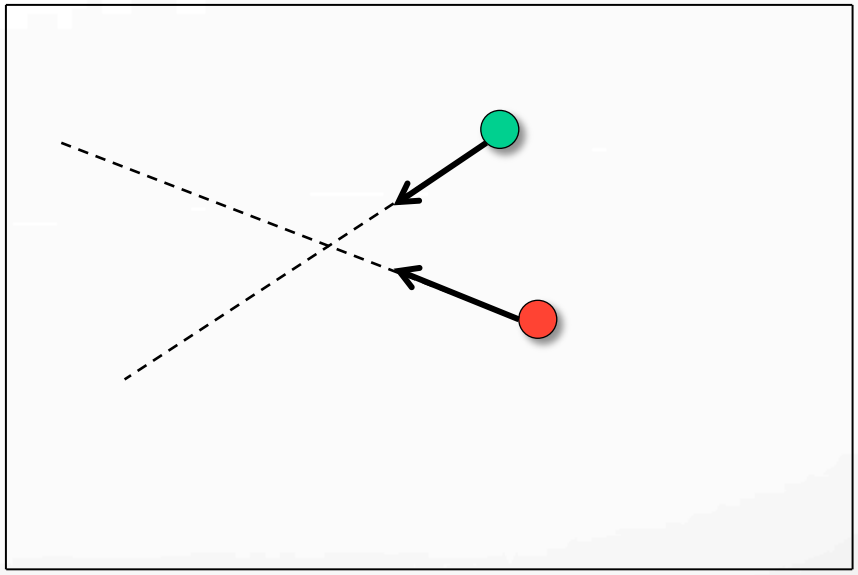
Types of Steering

- Ad hoc methods
  - Move in straight line
  - If possible collision, move
- Artificial potential fields
  - Obstacles have "charges"
  - Creates electrical field
  - Natural path through field
- **Vortex fields**
  - Obstacles have "currents"
  - Push around obstacles



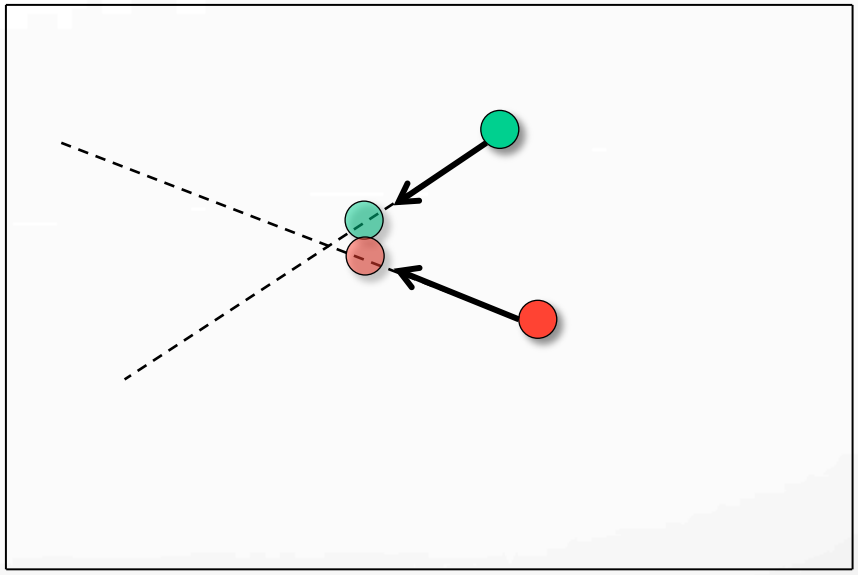
Ad Hoc Steering

1. **Compute nearest approach**
   - Assume constant velocity
   - Find time that it is closest
   - Repeat for all objects



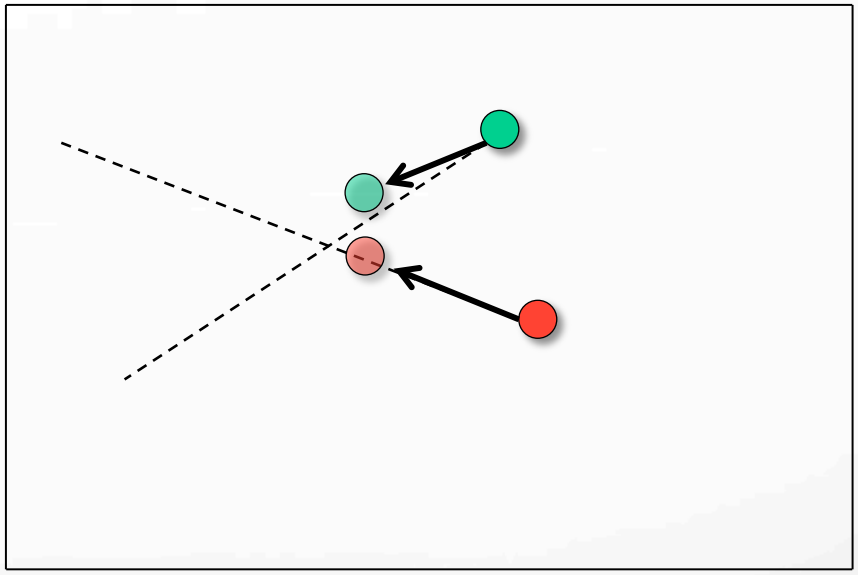
Ad Hoc Steering

1. Compute nearest approach
   - Assume constant velocity
   - Find time that it is closest
   - Repeat for all objects
2. **Find first collision**
   - Plug in time from last step
   - See if objects collide
   - Pick one with time least

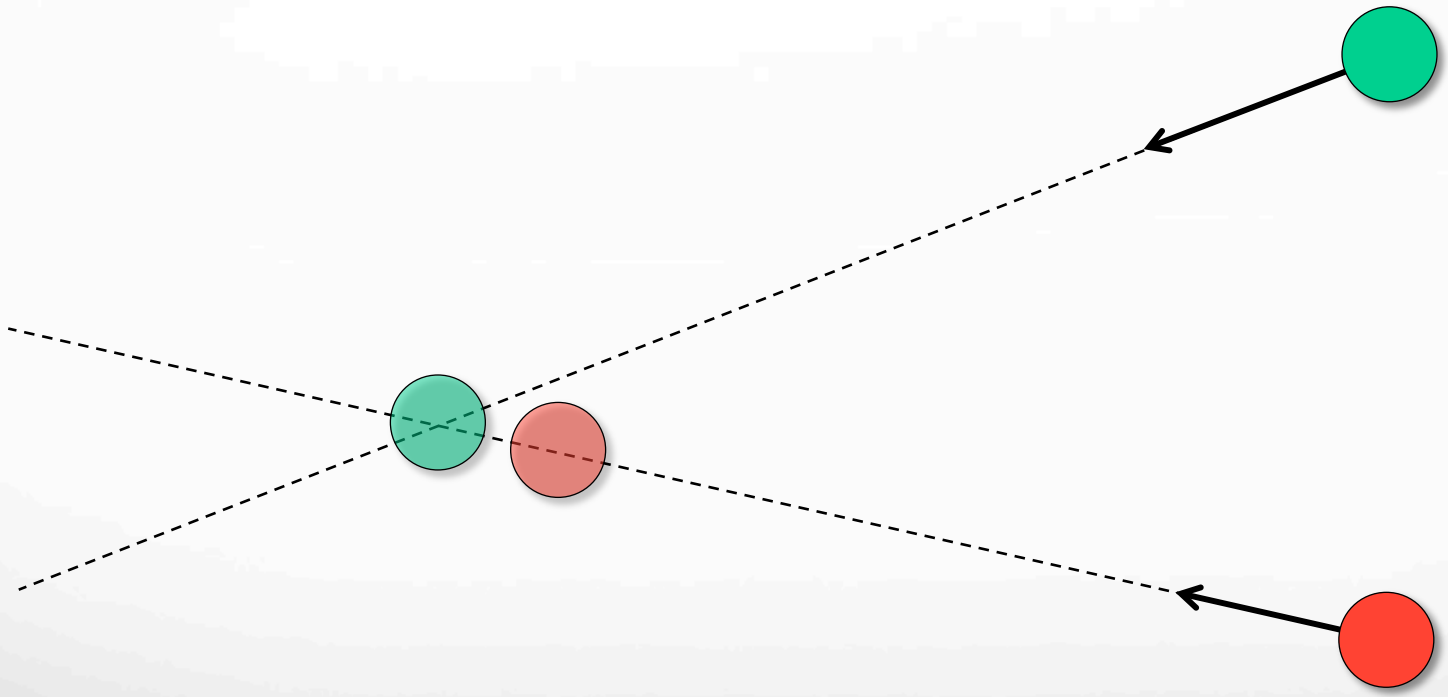


Ad Hoc Steering

1. Compute nearest approach
   - Assume constant velocity
   - Find time that it is closest
   - Repeat for all objects
2. Find first collision
   - Plug in time from last step
   - See if objects collide
   - Pick one with time least
3. **Adjust object velocity**
   - Nudge from the collision
   - Determine new velocity

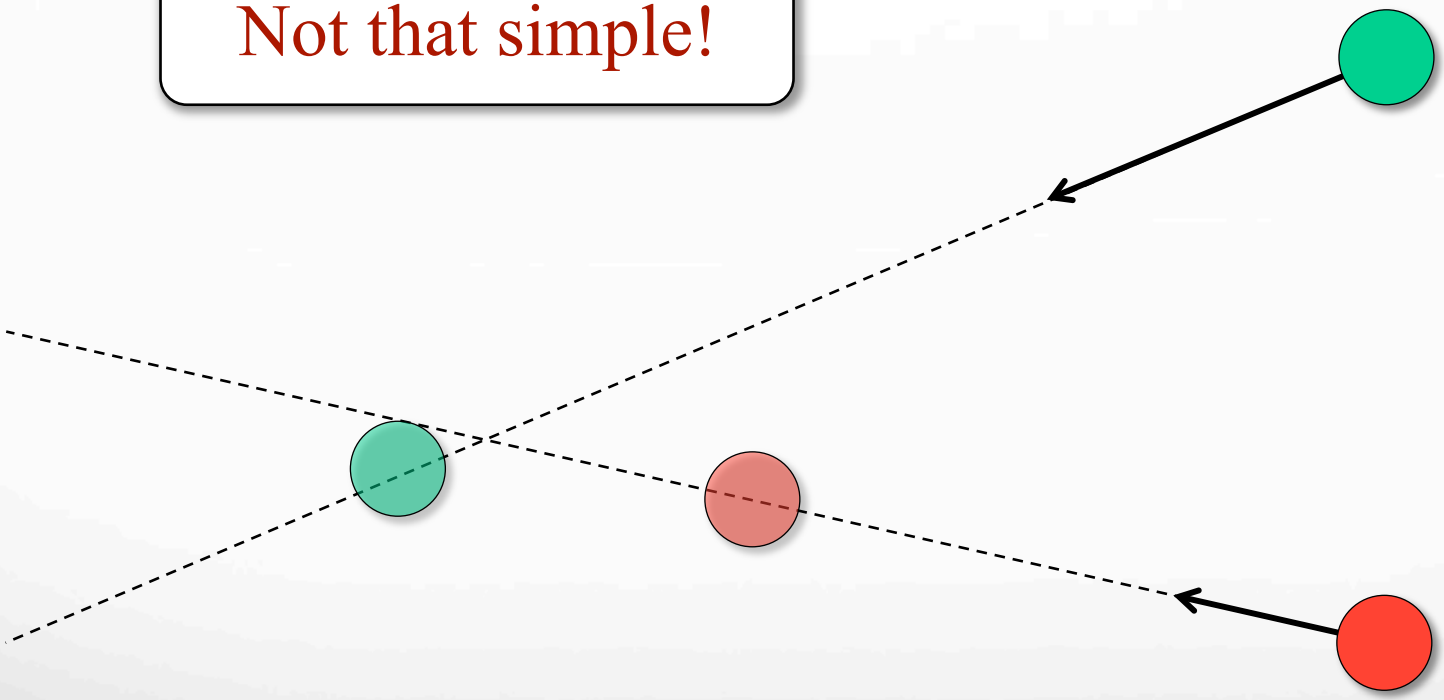


Computing Nearest Approach



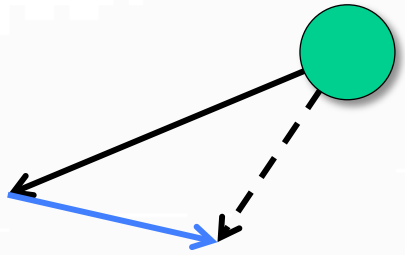
Computing Nearest Approach

Not that simple!



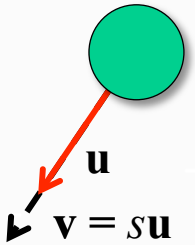
Computing Nearest Approach

1. Compute relative velocity



Computing Nearest Approach

1. Compute relative velocity
2. Normalize the vector
3. Compute relative speed

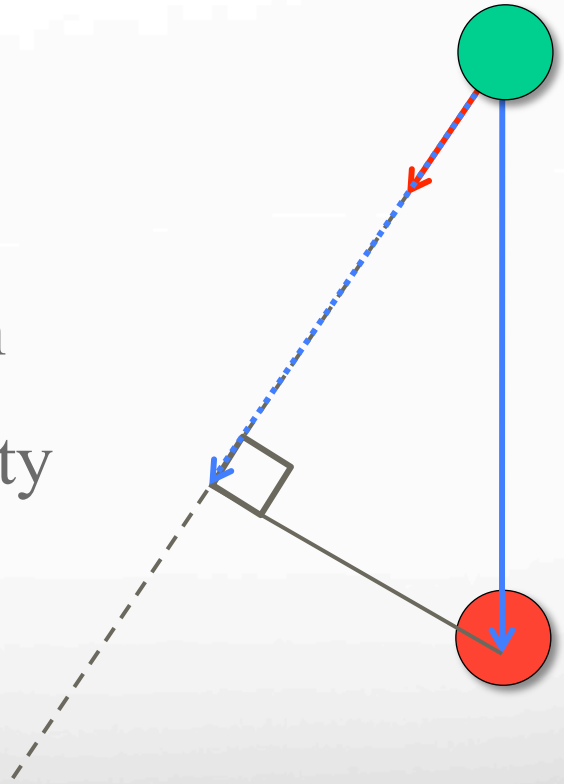


Computing Nearest Approach

1. Compute relative velocity
2. Normalize the vector
3. Compute relative speed
4. Compute relative position

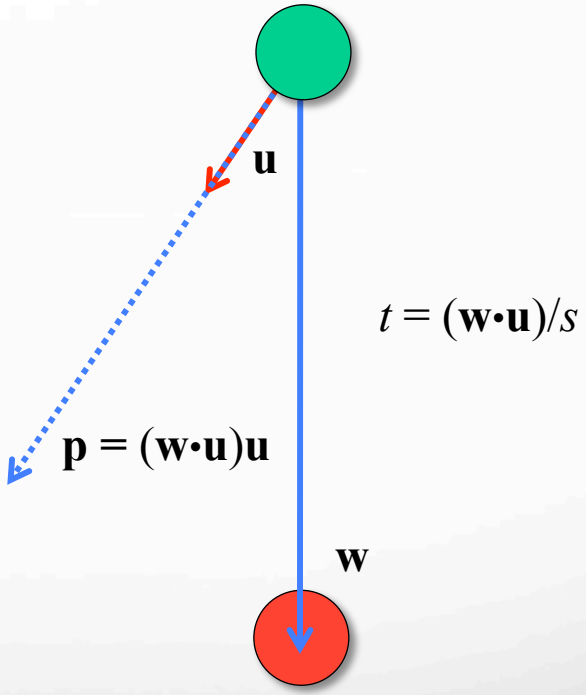
Computing Nearest Approach

1. Compute relative velocity
2. Normalize the vector
3. Compute relative speed
4. Compute relative position
5. Project position on velocity

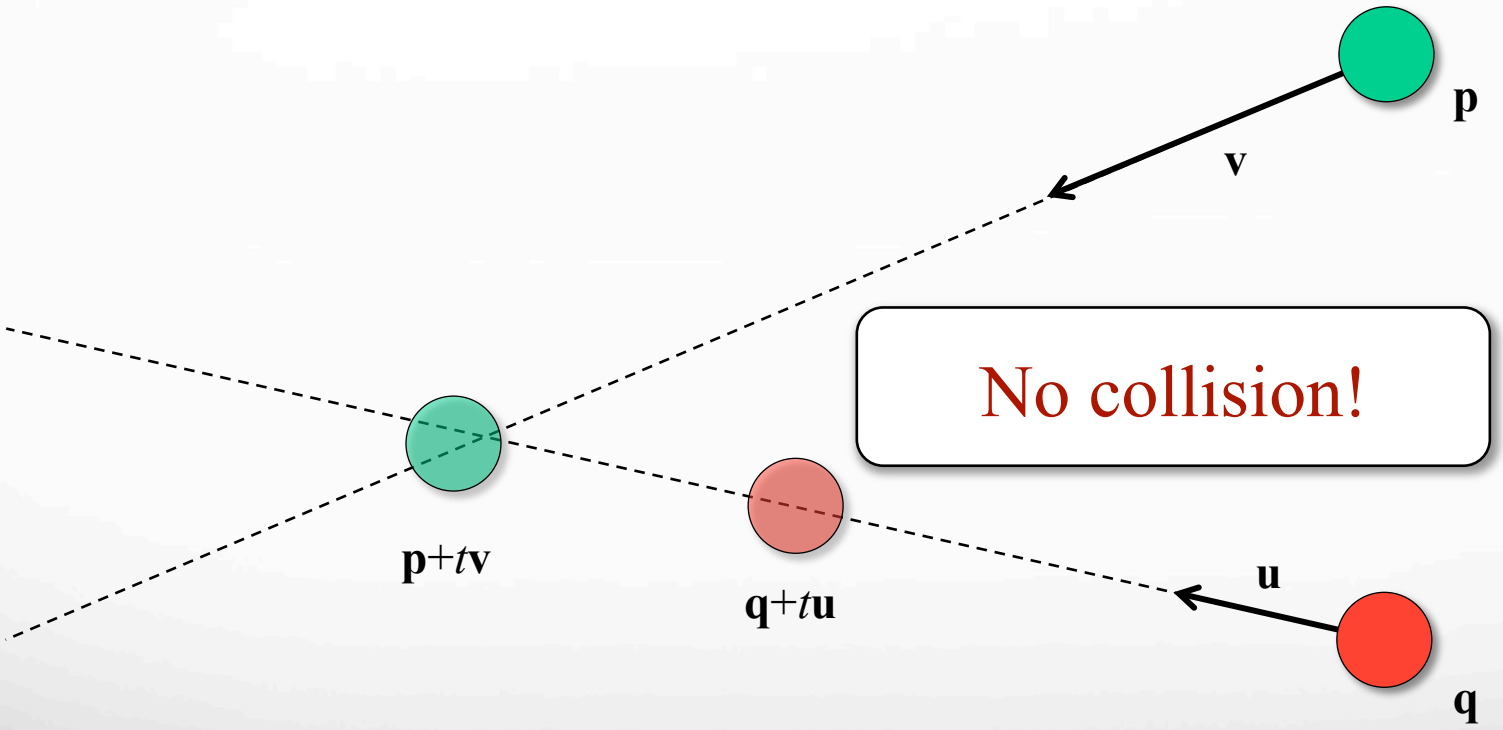


Computing Nearest Approach

1. Compute relative velocity
2. Normalize the vector
3. Compute relative speed
4. Compute relative position
5. Project position on velocity
6. Get time from projection

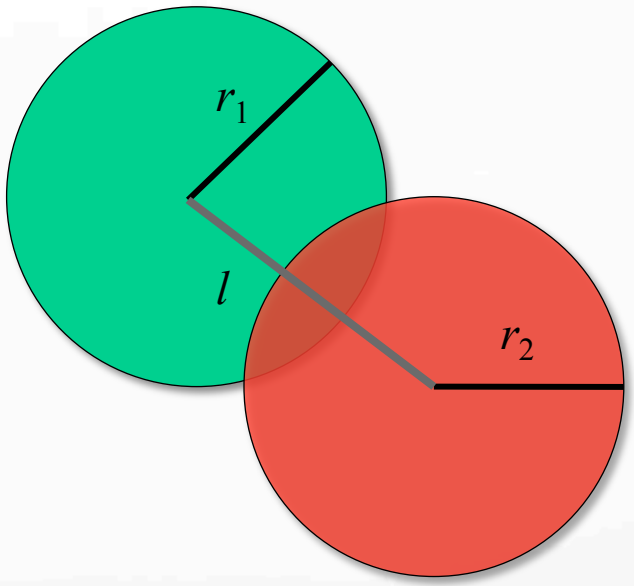


Check For Collision



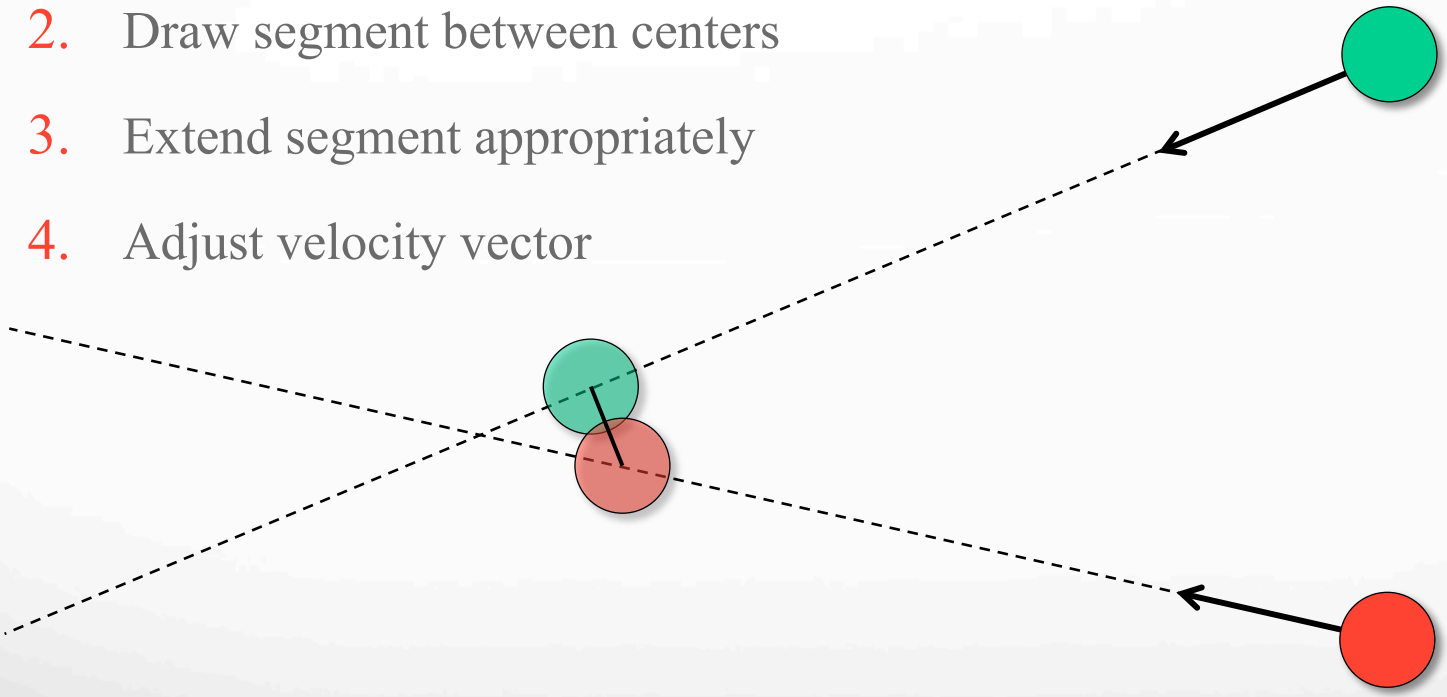
Check For Collision

- Objects != points
  - Must have size & shape
  - Collide if any overlap
- Easy if using circles
  - Line between centers
  - Get segment length l
  - Check if $l < r_1 + r_2$
- **Find first collision!**



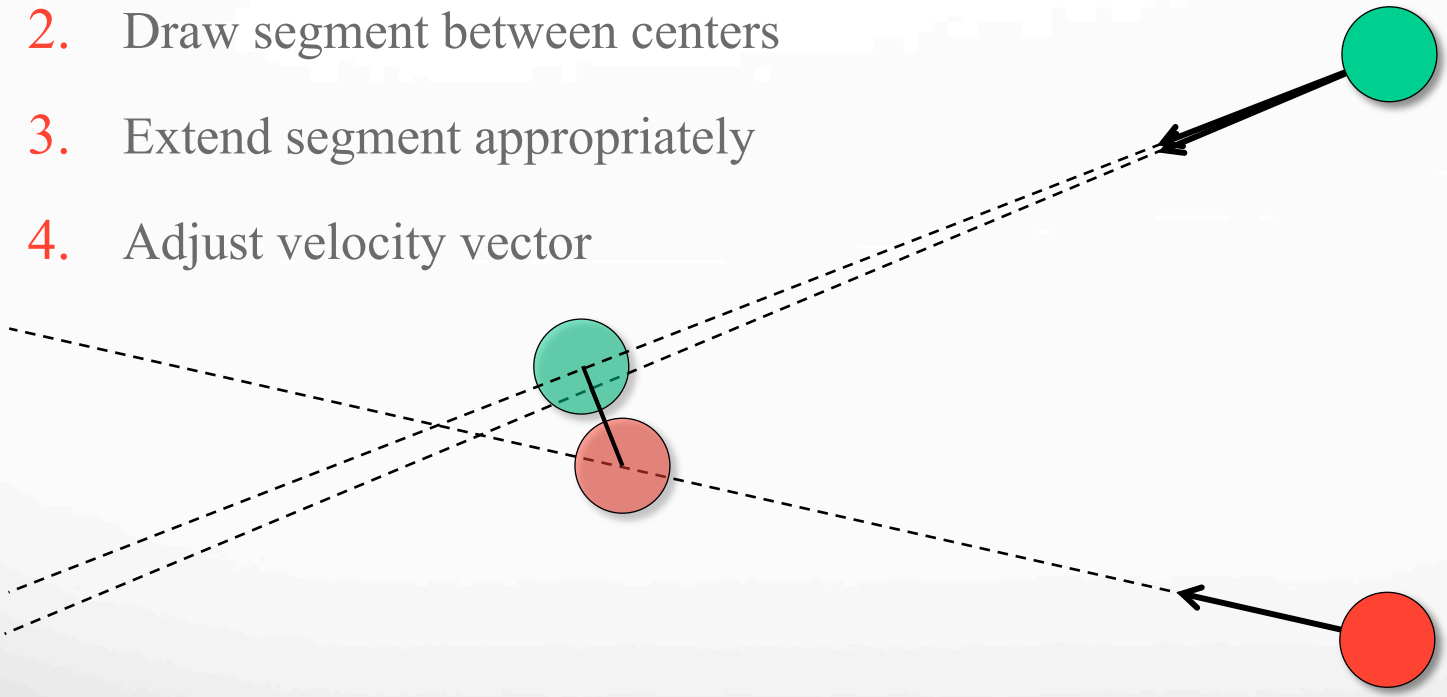
Adjust Object Velocity

1. Move objects to position
2. Draw segment between centers
3. Extend segment appropriately
4. Adjust velocity vector



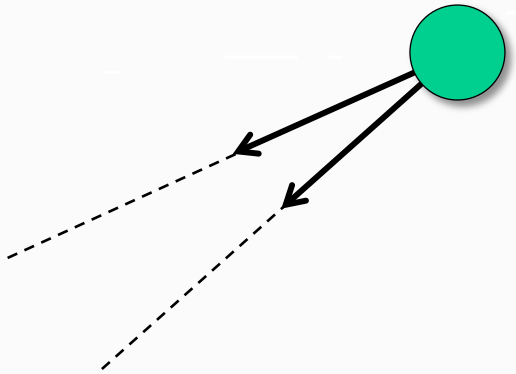
Adjust Object Velocity

1. Move objects to position
2. Draw segment between centers
3. Extend segment appropriately
4. Adjust velocity vector



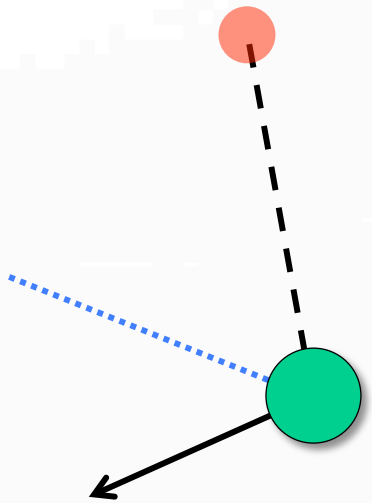
Adjust Object Velocity

- No set way to adjust
  - Hence “ad hoc” steering
- Can rotate on to new line
  - Compute old magnitude
  - Compute new unit vector
  - Multiply by old magnitude
- Can project on to line
  - Start with old vector is **v**
  - Compute new unit vector **u**
  - New vector is (**v**•**u**) **u**



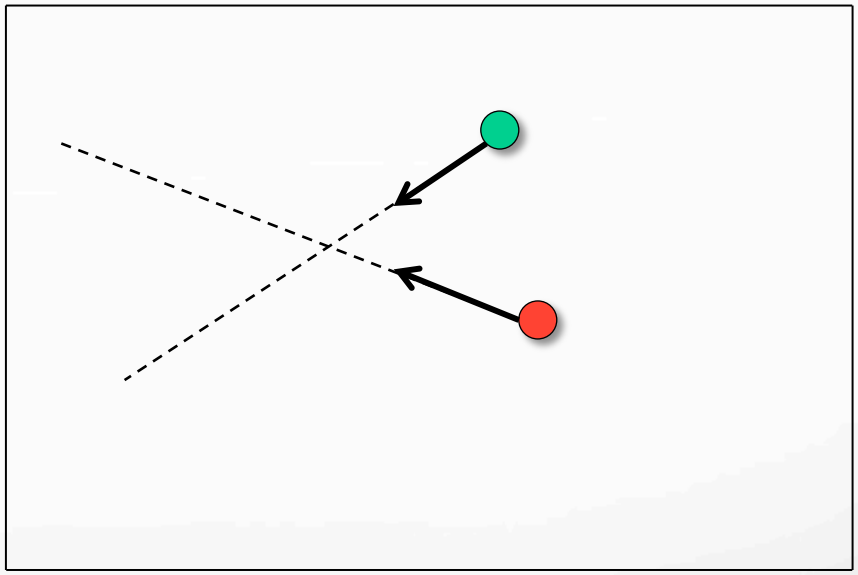
What If No Collisions?

- Go to your goal!
  - Going somewhere, right?
  - Also a velocity adjustment
- Avoid "whiplash"
  - Limit angle of change
  - "Turning radius"
- Exact method up to you



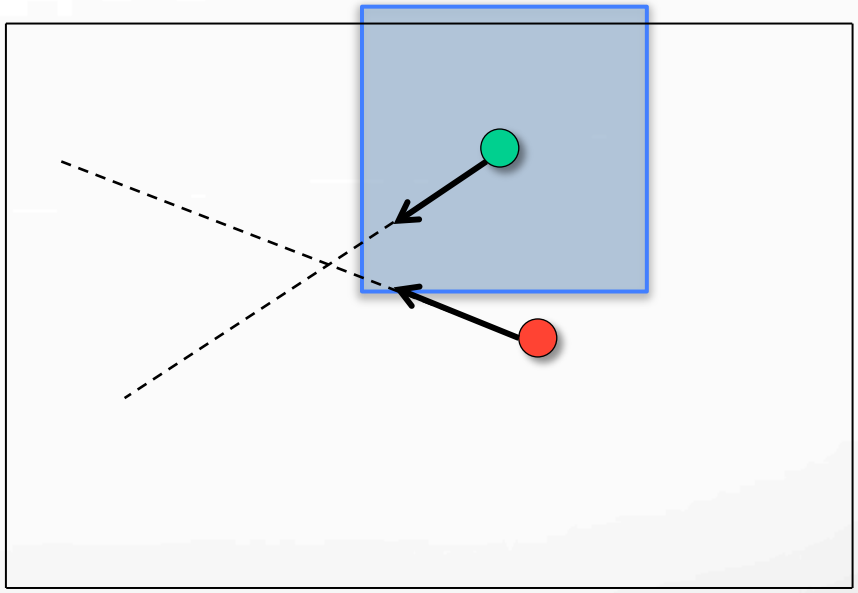
Optimizations

- Bound your search!
  - This is lots of linear algebra
  - Other object might move
  - Only move if you have to
- Bounding boxes
  - Center box on object
  - Only check objects inside box
  - Relatively quick test
- Angular cones
  - Can only turn so much
  - Limit to inside turning cone
  - Test is two dot products



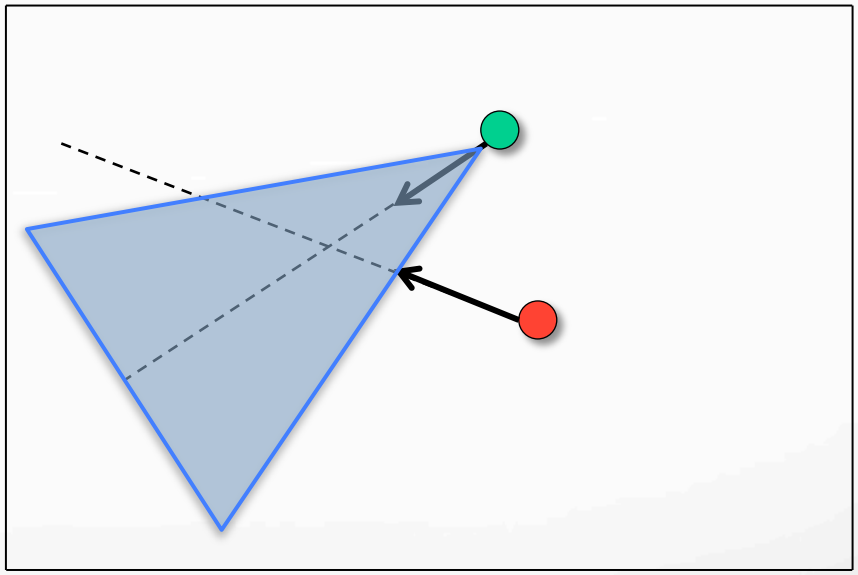
Optimizations

- Bound your search!
  - This is lots of linear algebra
  - Other object might move
  - Only move if you have to
- **Bounding boxes**
  - Center box on object
  - Only check objects inside box
  - Relatively quick test
- Angular cones
  - Can only turn so much
  - Limit to inside turning cone
  - Test is two dot products

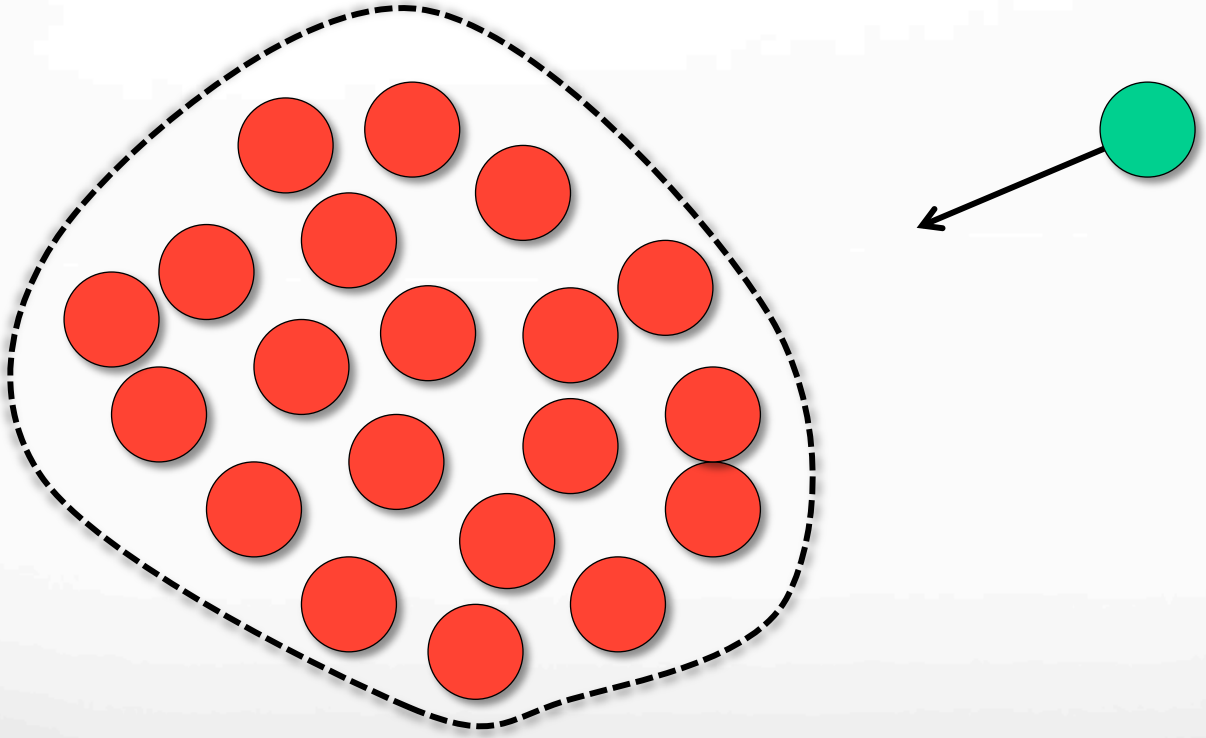


Optimizations

- Bound your search!
  - This is lots of linear algebra
  - Other object might move
  - Only move if you have to
- Bounding boxes
  - Center box on object
  - Only check objects inside box
  - Relatively quick test
- **Angular cones**
  - Can only turn so much
  - Limit to inside turning cone
  - Test is two dot products

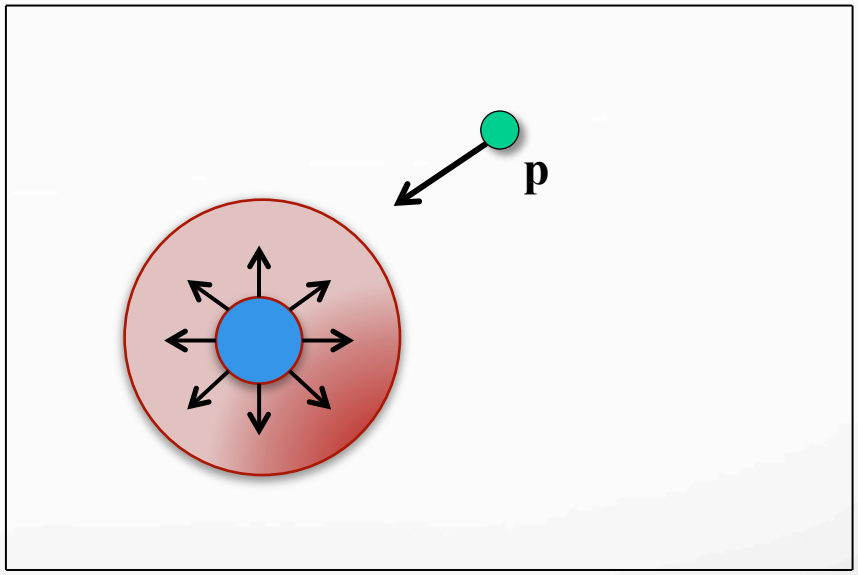


Ad Hoc Steering: Problem



Potential Fields

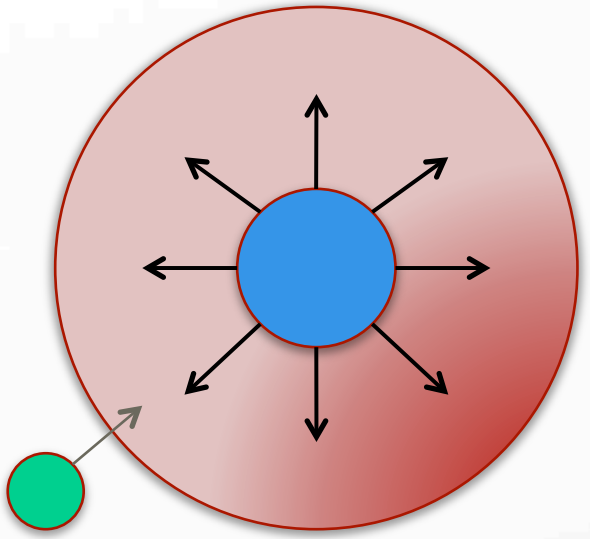
- Much more physics based
  - Goal pulls towards
  - Obstacles push away
- Create an energy field
  - $E(\mathbf{p})$ = energy field
  - $F = \nabla_{\mathbf{p}} E$
  - "Steepest decent"
- Get velocity from force
  - $\mathbf{v} = F/m$



"Solves" cluster problem

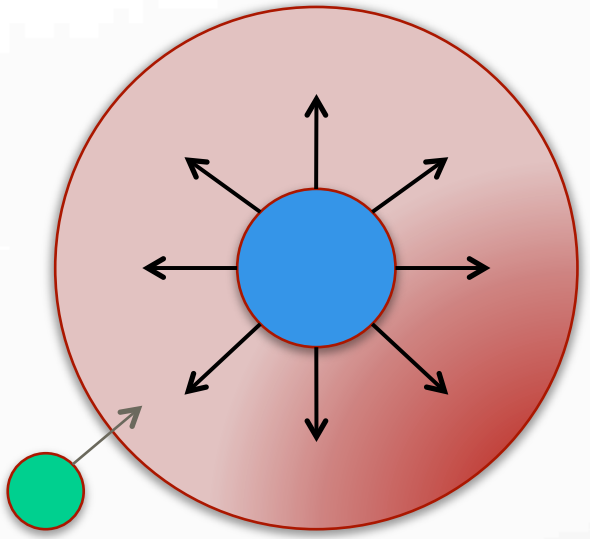
Potential Fields

- $E = A_{\mathbf{g}} + \sum_{\mathbf{q}} R_{\mathbf{q}}$
  - $A_{\mathbf{g}}$ = attraction force
  - $R_{\mathbf{q}}$ = repulsion force
- Gradient is linear
  - $F = \nabla_{\mathbf{p}} A_{\mathbf{g}} + \sum_{\mathbf{q}} \nabla_{\mathbf{p}} R_{\mathbf{q}}$
  - Just need it for $A_{\mathbf{g}}$, $R_{\mathbf{q}}$
  - Or fast approximations



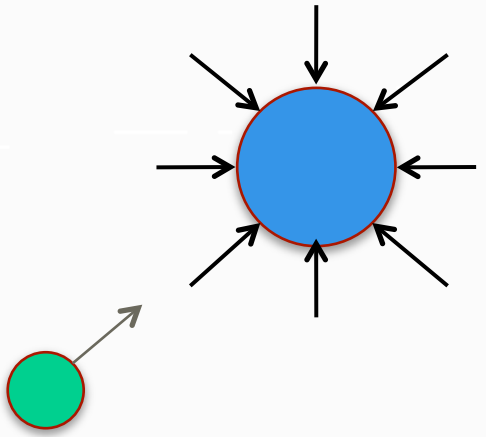
Potential Fields

- **Repulsion Field**
  - $\mathbf{q}$: obstacle location
  - r_0: vehicle radius
  - r_1: obstacle radius
  - s: separation factor (=1.05)
  - c_0: vehicle "charge"
  - c_1: goal "charge"
- Field computation
  - $d = |\mathbf{q} - \mathbf{p}| - s * (r_0 + r_1)$
  - $R_{\mathbf{q}} = \frac{c_0 c_1}{d}$



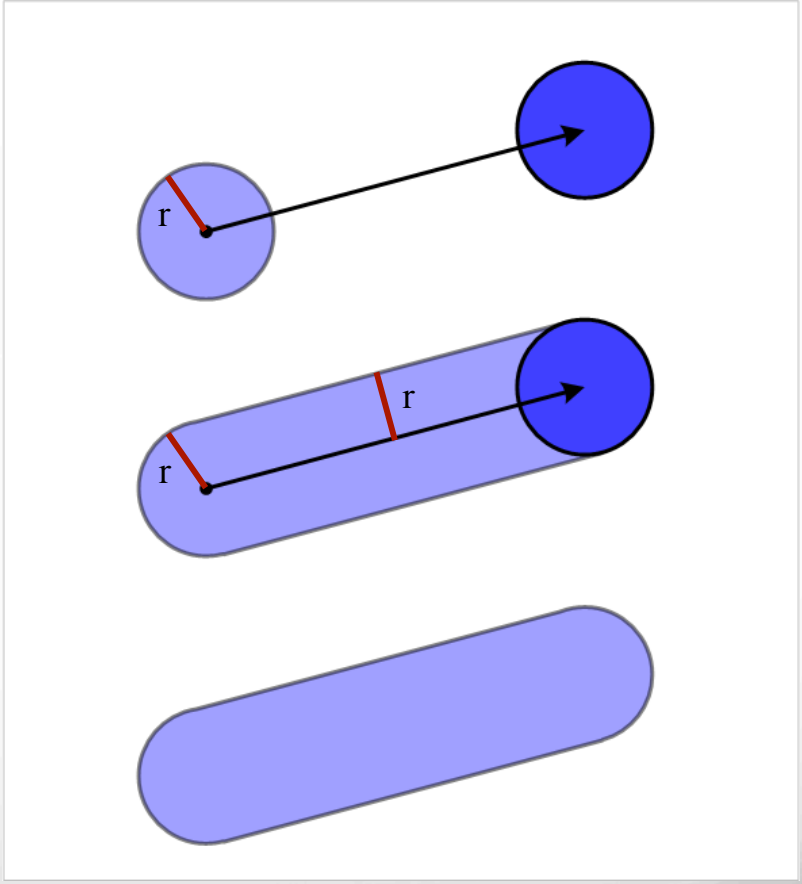
Potential Fields

- **Attraction Field**
  - k: coefficient (= 1/e)
  - b: breaking distance (= 1)
  - $\mathbf{g}$: goal location
  - c_0: vehicle "charge"
  - c_1: goal "charge"
- Field computation
  - $a(d) = \begin{cases} (d-b) * (\frac{-2}{b*\mathrm{e}}) + \frac{1}{\mathrm{e}} & \text{if } d > b \\ \mathrm{e}^{-\frac{d^2}{b^2}} & \text{otherwise} \end{cases}$
  - $A_{\mathbf{g}} = -kc_0c_1a(|\mathbf{g} - \mathbf{p}|)$



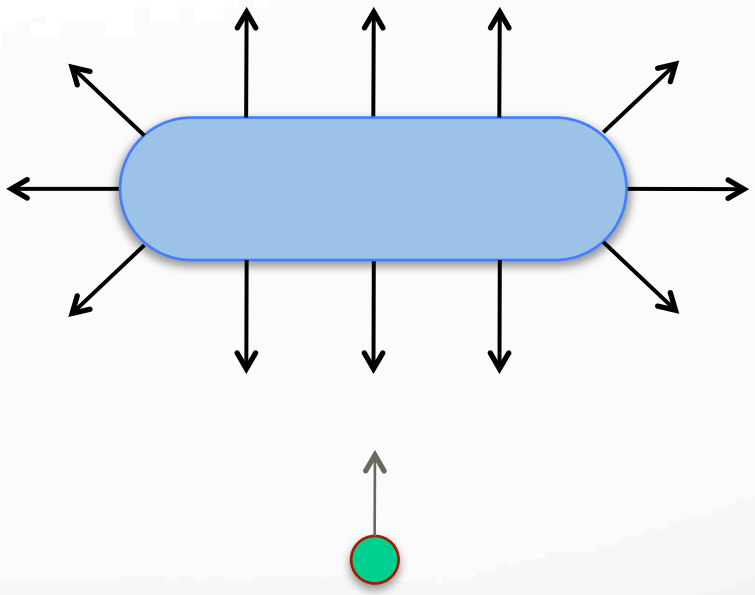
Generalizing Potential Fields

- Not limited to circles!
  - Take obstacle shape
  - Surround it by contours
  - Equidistant from shape
- Basic idea: **capsules**
  - "Line with radius"
  - Distance to line in R_q
  - Use on polygon boundary
- Not needed for assignment



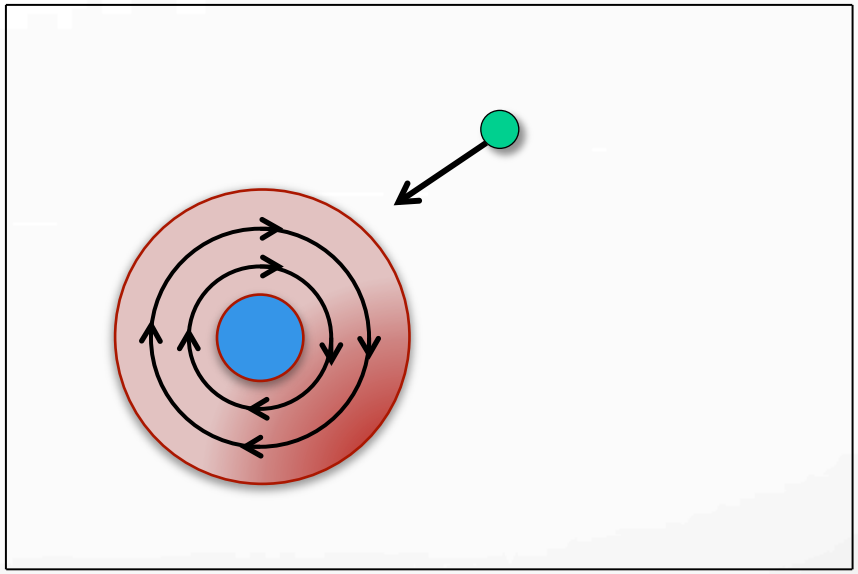
Potential Fields: Problems

- Expensive!
  - Did you see an exponent?
  - Need fast approximations
  - **Idea**: Taylor polynomials
- **Local minima**
  - All forces add to zero
  - Goal on other side of wall
  - Why pathfinding is better
- Possibly jittery
  - Lots of overcompensation



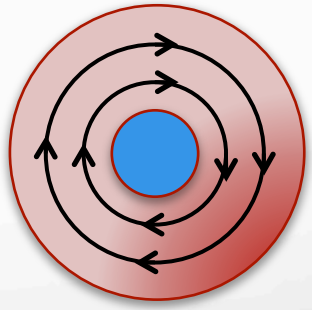
Vortex Fields

- Like ad hoc steering
  - Move in straight line
  - If any collision, move
- Vortex tells how to move
  - Pushes object around
  - Only use nearest field
- State of the art steering
  - Used in Clancy games
- Tricker to get right



Vortex Fields

```
Vector3 VortexForce(Agent a, Obstacle o) {
  Vector3 distV = (o.getPos()—a.getPos());
  float  distSq = distV.magSquare();
  if ( distSq <= o.fieldRadiusSq) {
    float cross =
      distV.cross(a.getVelocity()).getZ();
    if (cross < 0 ) {
      return turnLeft(distV);
    } else {
      return turnRight(distV);
    }
}
```



Pathfinding in Practice

- Navigation Meshes
  - Indicates walkable areas
  - 2D geometric representation
  - Connected convex shapes
  - Graph: center-to-center
- Pathfinding + Steering
  - A* search on graph
  - Only pathfind once
  - Steering node-to-node

Summary

- Steering is an alternative to pathfinding
  - Recompute velocity each round
  - Reacts well to dynamic obstacles
- Three main forms of steering
  - Ad hoc steering ← **best for assignment**
  - Potential fields
  - Vortex fields