
MARCH 5-9
2007
SAN FRANCISCO
TAKE
CONTROL
www.gdconf.com
MOSCONE
CENTER

Embodied Agents in Dynamic Worlds

John O’Brien & Bryan Stout

Pathfinding: Then

- Use entirely pre-generated data
- Nav mesh/grid + A* = 
- Limited dynamic avoidance necessary

Pathfinding: Now

- Users want everything to blow up
- Dynamic environments becoming the norm
- Physics and destruction part of gameplay

Dynamic Worlds

In today's games:

- Large objects move around
- Paths open up and close off

The Problem

- Pre-calculated data is pre-calculated for a reason!
- Modifying navigation data at runtime can be prohibitively expensive

Solutions

- Dynamic Motion/Avoidance Techniques
- Dynamic Pathing Techniques

Dynamic Motion

Typical AI Motion System

- Pre-generated navigation data
- A* or derivative produces a series of path points
- Motion code moves agent from point to point, avoiding other agents as it goes

Motion Models

- Some AI systems heavily dependent on animation states
- Others have complete freedom of movement
- Both can benefit from force-based steering solutions

Physics & Collision

- Raycasts against collision geometry typically too expensive for widespread AI use
- We need less expensive methods that can be applied to many agents simultaneously

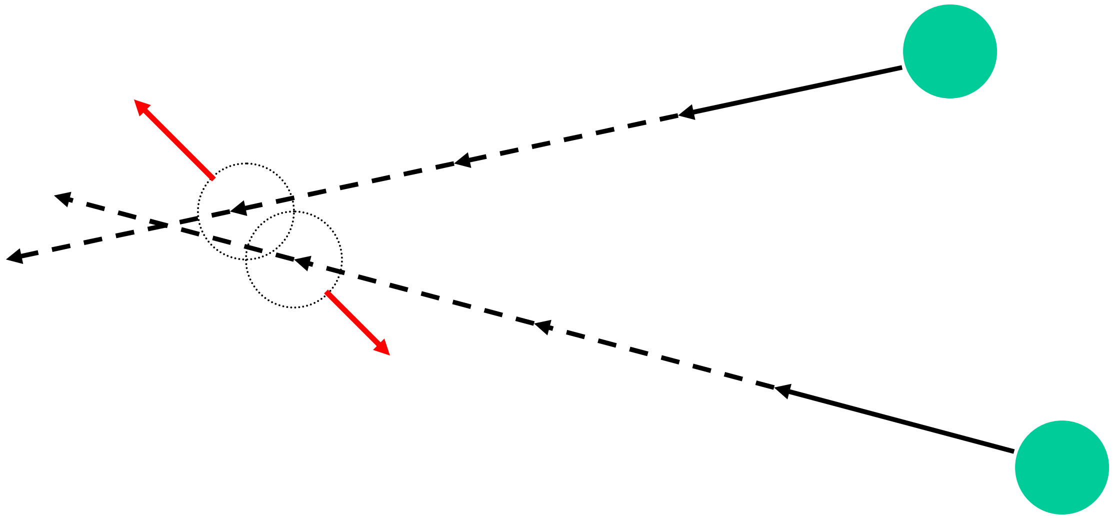
Topic List

- Avoidance Steering Behaviors
- Agent-based potential fields
- Shared potential fields

Outer Collision Avoidance

- At a distance, force-based steering methods try to achieve gentle course correction.
- This is often combined with strong repulsion close to an obstacle.

Unaligned Collision Avoidance

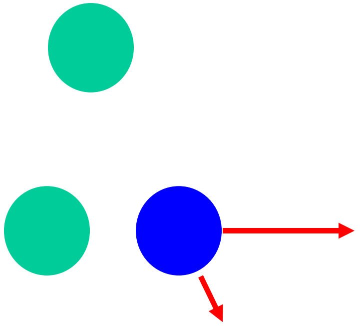


Unaligned Collision Avoidance

```
Vector3 UnalignedAvoidance( Agent a1, Agent a2)
{
        Vector3 relativeVel = a2.velocity – a1.velocity;
        float relativeSpeed = relativeVel.Length;
        relativeVel.Normalize();
        Vector3 relativePos = a1.pos – a2.pos;
        float projection = Dot( relativeVel, relativePos );
        float deltaT = projection / relativeSpeed;
                        :               // Calc future positions at +deltaT
        return (a1FuturePos – a2FuturePos);
}
```

Inner Collision: Separation

- Nearby agents strongly repel others
- Simple and effective, cheap to calculate.

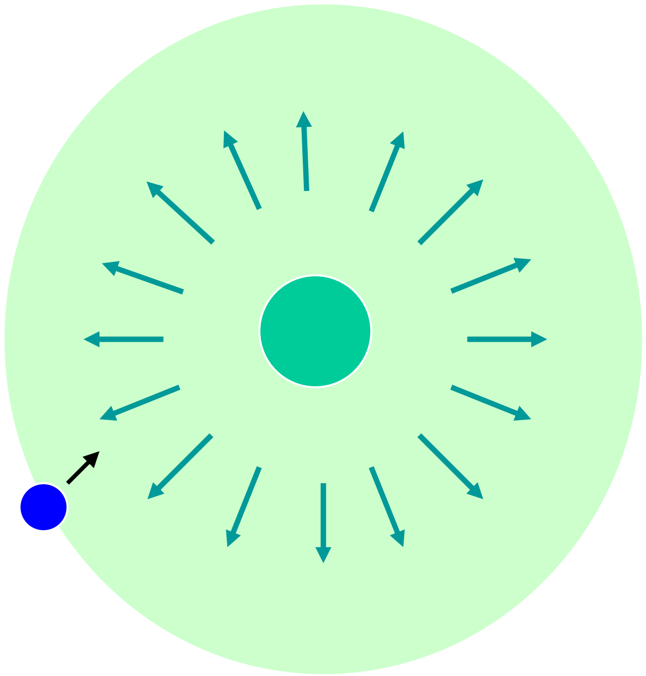


Unaligned Collision Avoidance: Analysis

- Good for avoiding other agents (small obstacles which will also avoid you)
- Straight repulsion and single point check problematic for larger objects
- Several Sqrt()s per check become expensive as environment becomes more crowded

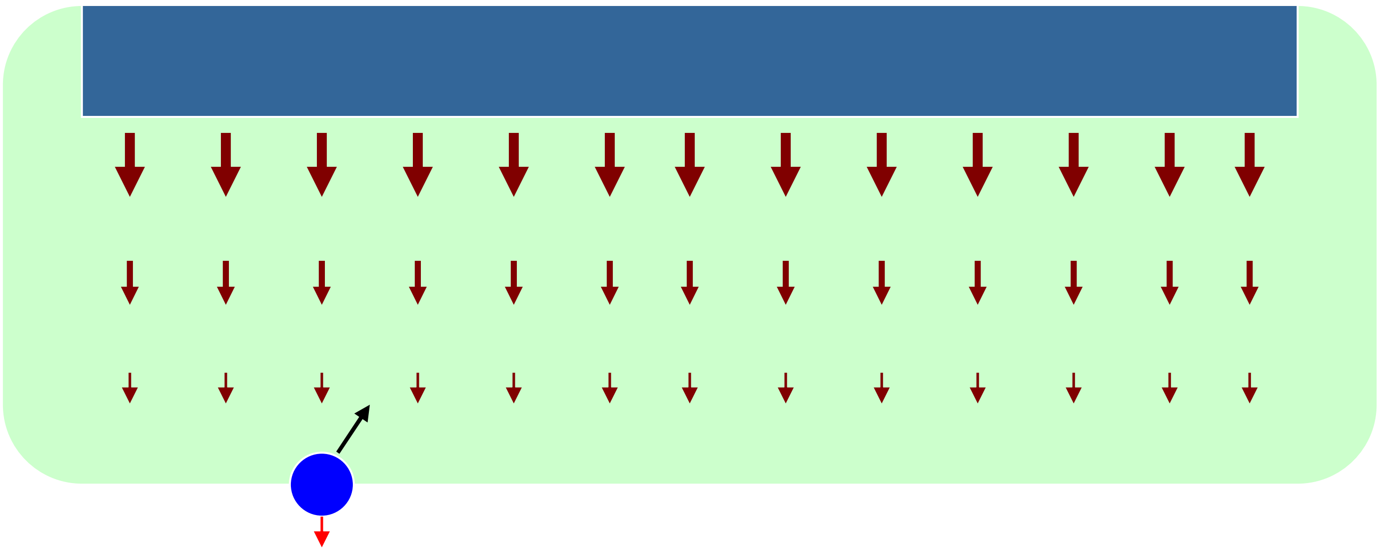
Agent-Based Potential Fields

- Less interested in specific collision detection
- Conceptually like magnetic fields



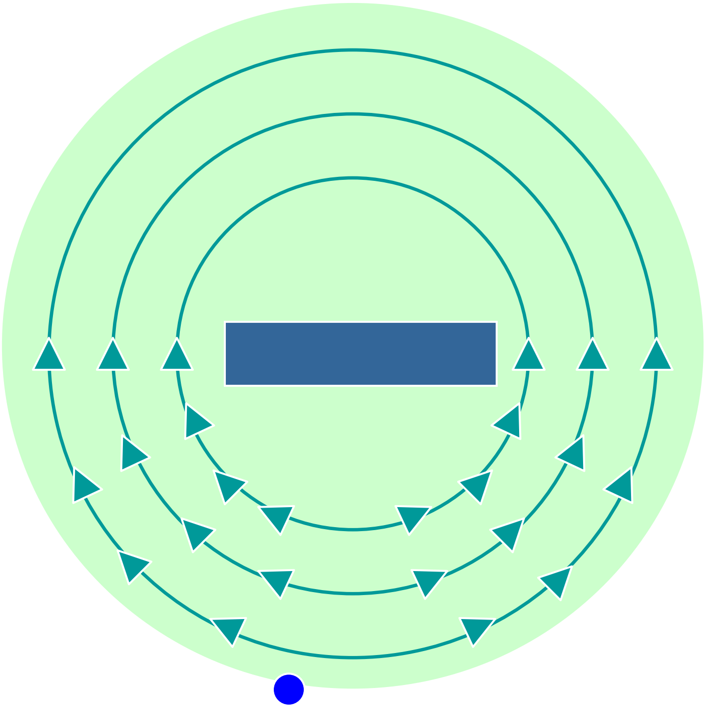
Standard Repulsive Field

- Repulsive force increases as agent draws closer



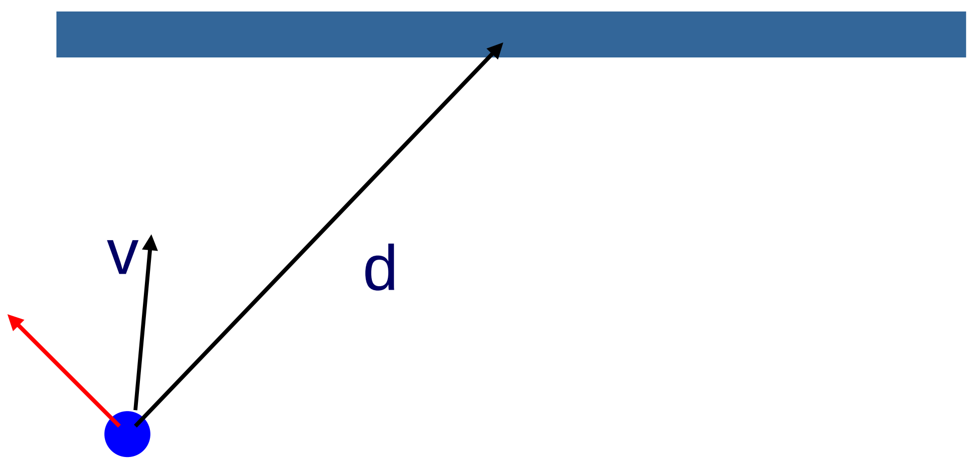
Vortex Fields

- Repulsion can still increase with closeness, but pushes agent off to the sides, perpendicular to distance vector



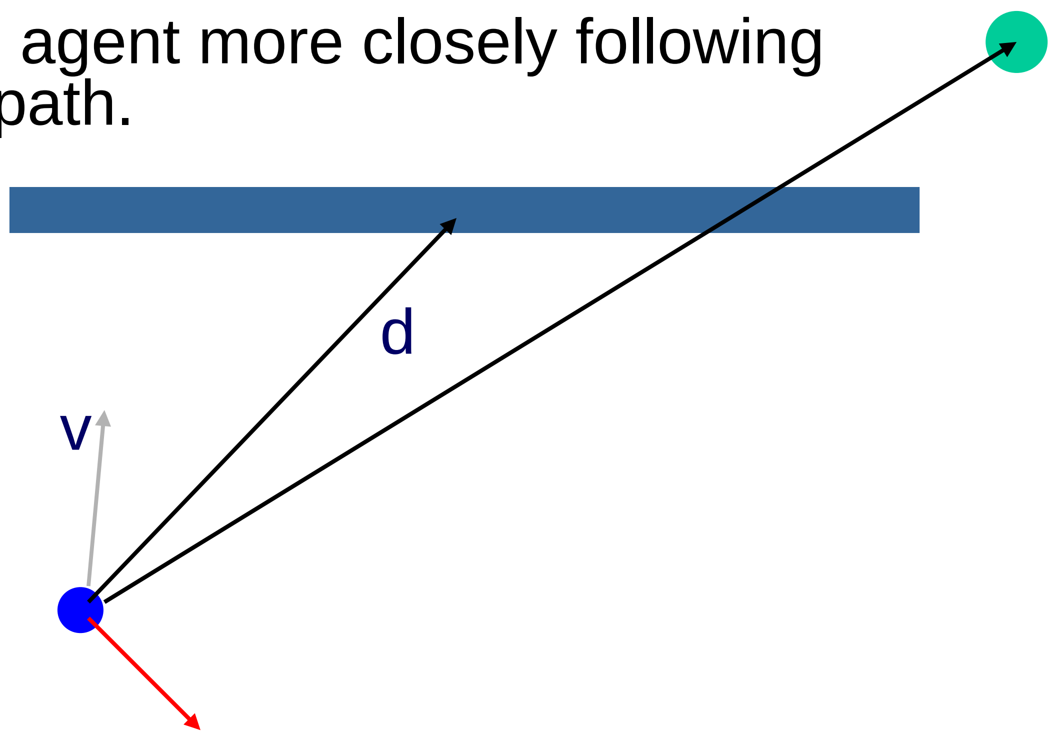
Vortex Fields: Choosing Direction

- One method: use *distance* X *velocity*.



Vortex Fields: Choosing Direction

- Alternatively: use vector to goal instead of velocity.
- Can keep agent more closely following intended path.

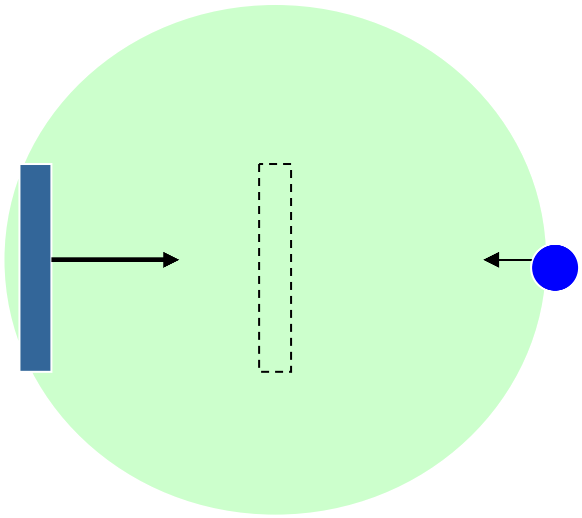


Vortex Fields

```
Vector3 CalcGyroscopicForce(Agent a, Obstacle o)
{
        Vector3 distV = (o.GetPos() – a.GetPos());
        float LengthSq = DistSq( distV );
        if ( distSq <= o.fieldRadiusSq )
        {
                float cross = Cross( distV, a.GetVelocity() ).z;
                if (cross < 0 )
                        return TurnLeft( distV );
                else
                        return TurnRight( distV );
        }
}
```

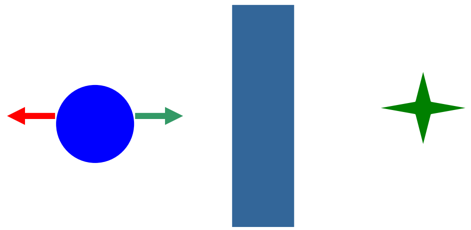
Vortex Fields: Prediction

- Scale center of field based on obstacle velocity and distance to agent.
- Gives same effect of agent avoiding a future collision as we saw previously

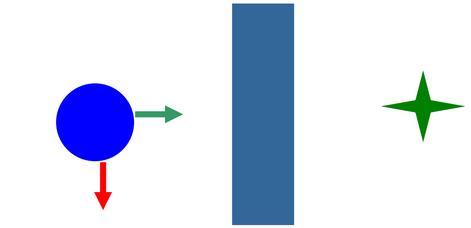


Force-Based Steering Issues

- Local minima
  - Attraction == repulsion

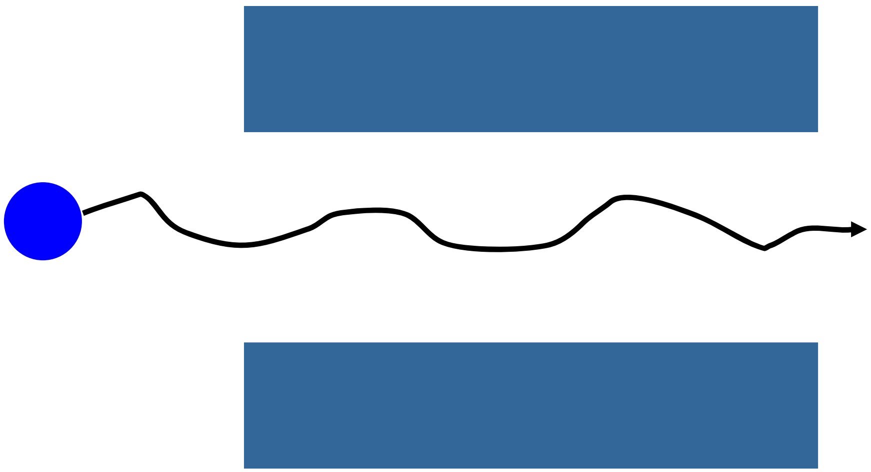


- Vortex fields
  - Tend to guide object in general direction of attraction



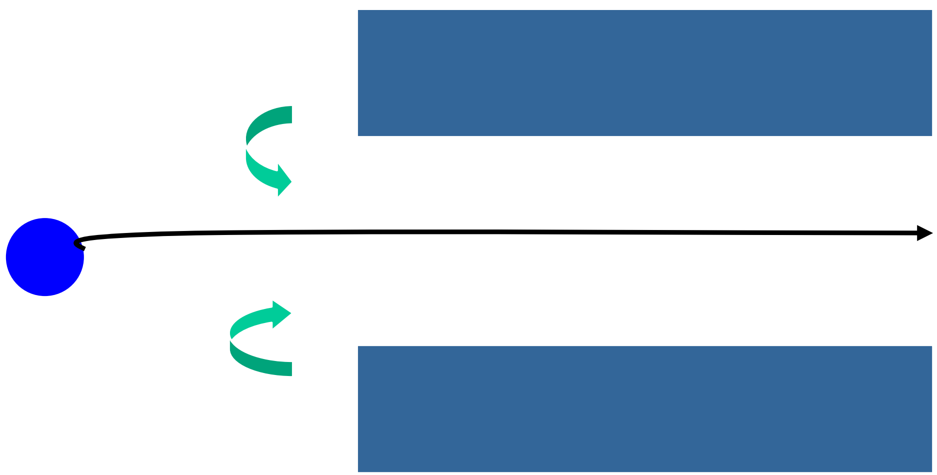
Force-Based Steering Issues

- Oscillation
  - Agent will swing back and forth, especially in the presence of multiple obstacles.



Force-Based Steering Issues

- Gyroscopic repulsion helps agents navigate narrow areas more smoothly.



Inner Collision

- For large, unevenly shaped obstacles, inner collision will most likely require more than a sphere representation
- Capsules work well if you can use them
- 1st pass physics collision rep can work also

Vortex Field Analysis

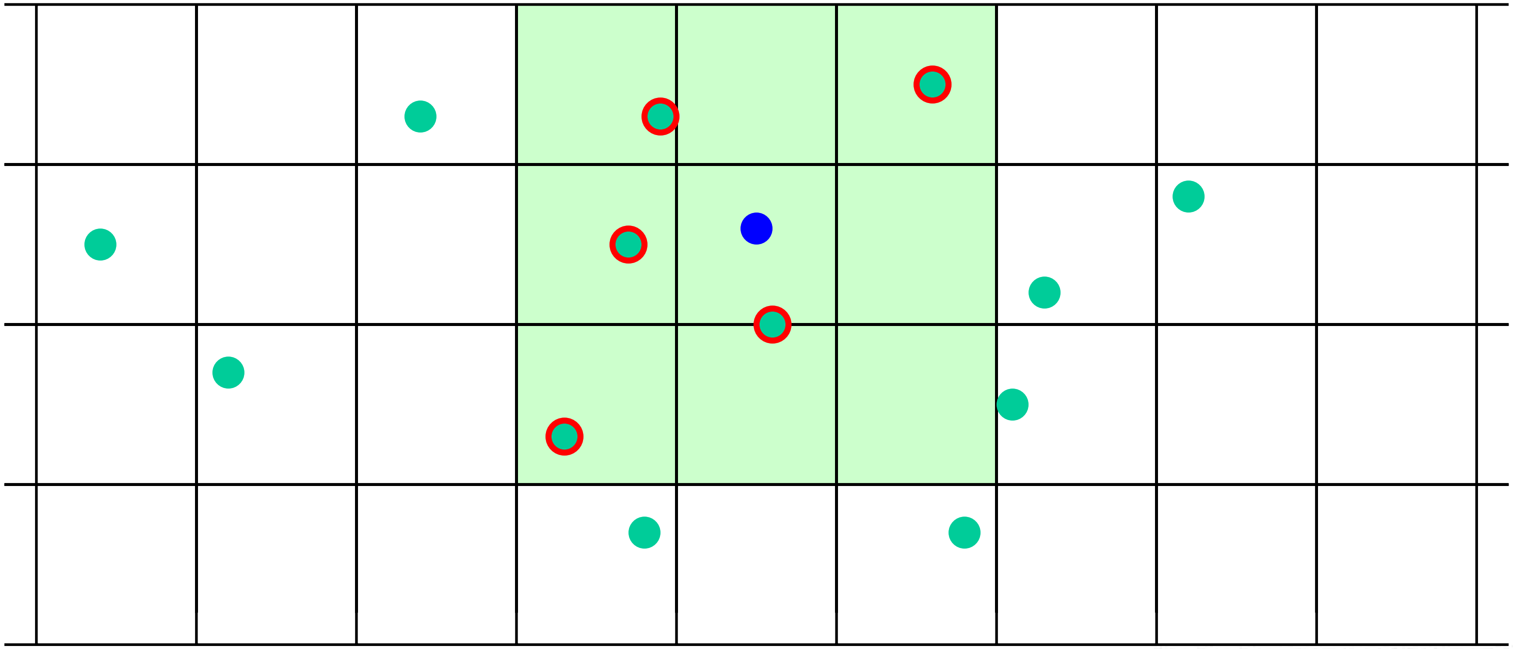
- Fairly smooth avoidance at a distance
- Reasonably lightweight processor usage
- Interact with each other in a more favorable way than straight repulsion techniques

Collision Candidate Filtering

- Eliminating unnecessary checks is a key component of performance
- Smooth, believable motion relies on eliminating unwanted influences

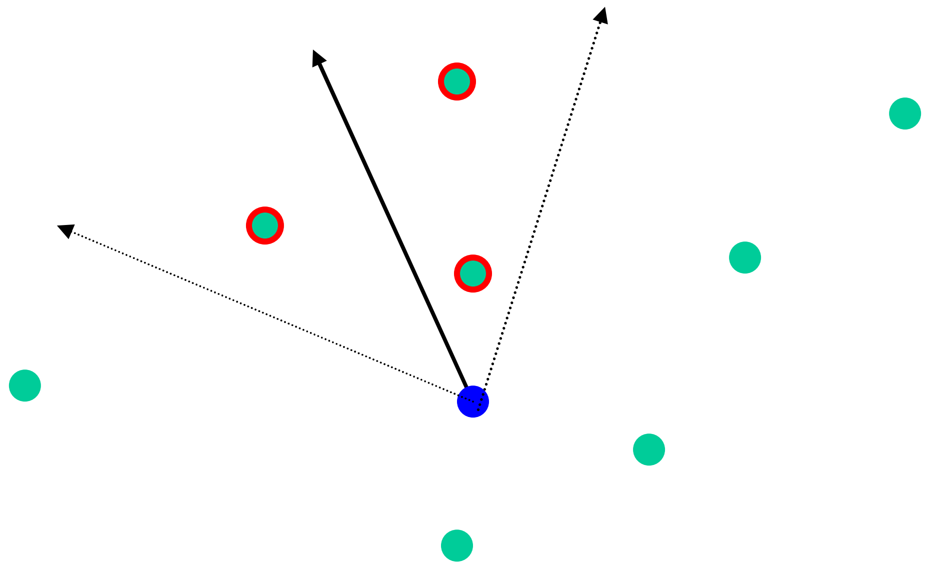
Collision Candidate Filtering

- "Collision Buckets"



Collision Candidate Filtering

- Angle Tests
  - Exclude objects that are not within a certain angle of agent's forward movement.

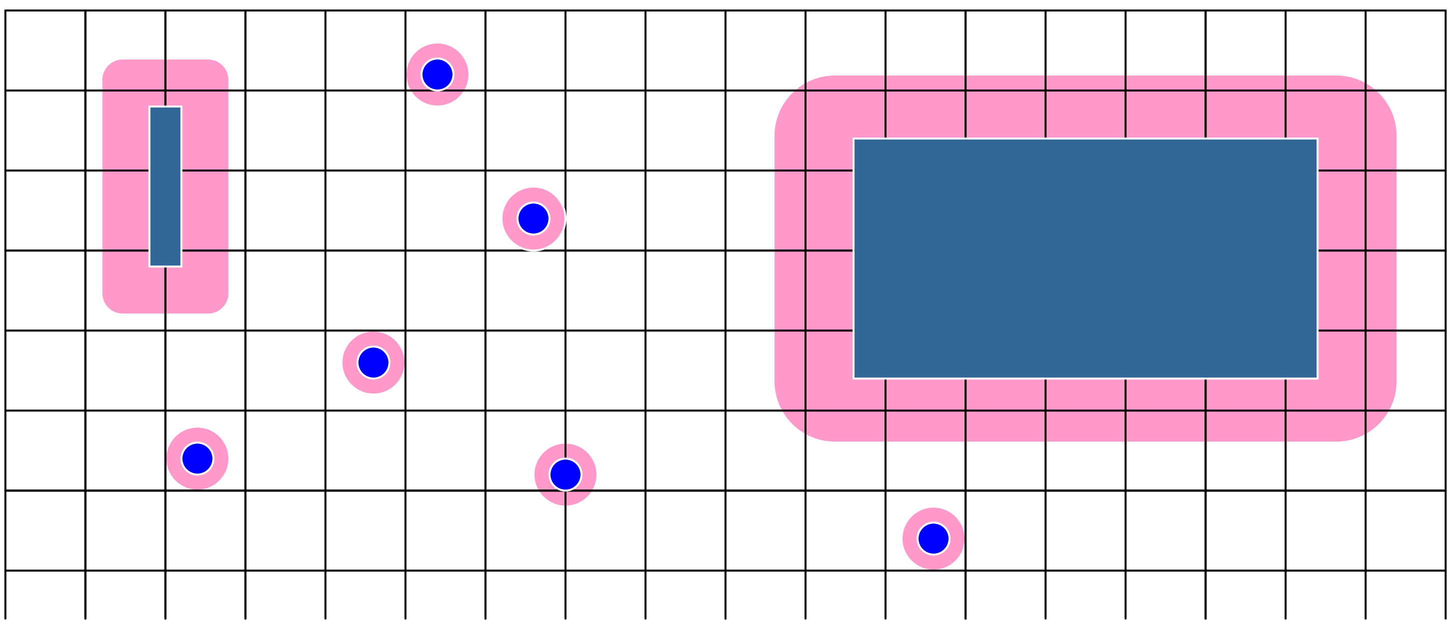


Demo

- Agents move with constant attraction to goal (except close in)
- Agent sim clamps turning and velocity changes
- Weak *separation* behavior between agents

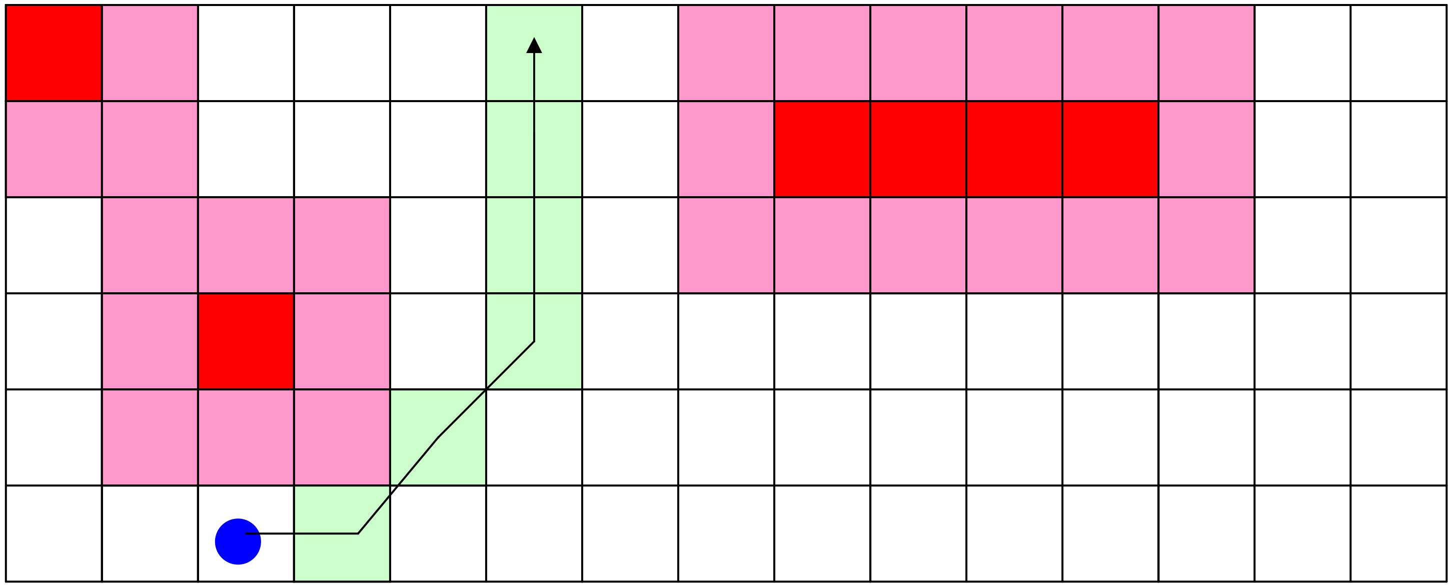
Shared Potential Fields

- A shared data structure representing the potential field may be viable for large crowd scenes.



Shared Potential Fields

- Agents traverse the terrain trying to remain in areas of high movement potential.
- Can be used to simulate attractive areas like roads and pathways in addition to repulsive areas like obstacles



Continuum Crowds

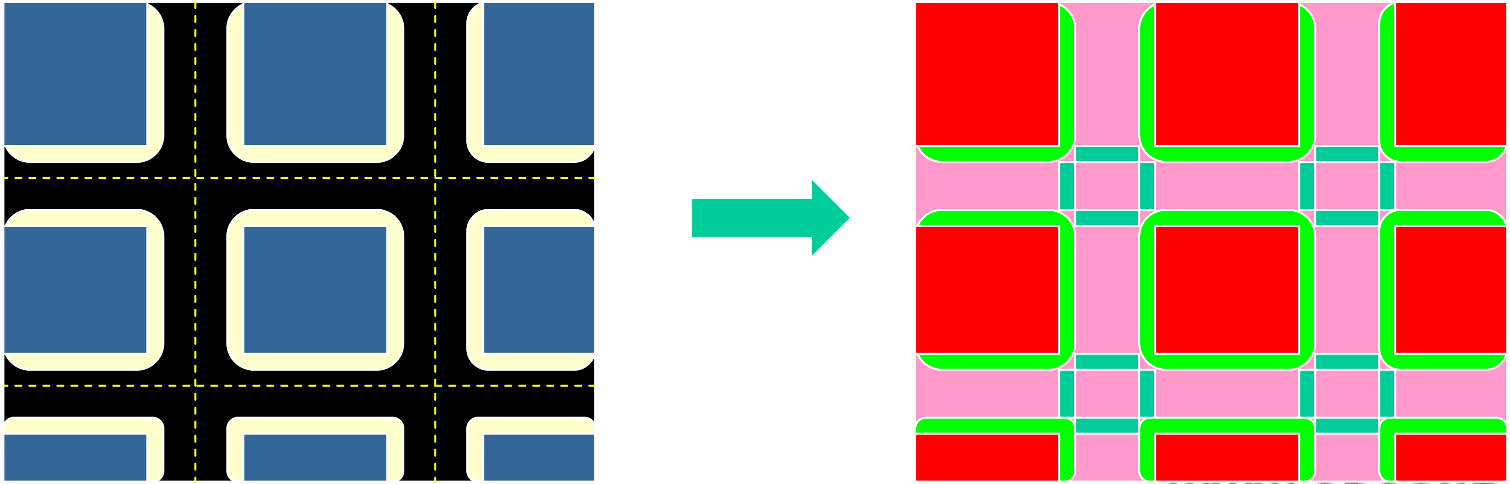
- Talk given at SIGGRAPH ’06
- Crowd simulation using principles of fluid dynamics.
- Apparently capable of simulating large crowds with realistic movement.

Continuum Crowds

- Build series of state grids
  - Crowd density
  - Goal locations
  - Impassable areas
- Combine into single potential field
- Move agents opposite to gradient of the field

Continuum Crowds

- Impressive city street crowd modeling
- "Discomfort fields" used to keep agents on sidewalk.
- Projected density out in front of moving objects like vehicles



Conclusion

- Game worlds will continue to become more and more dynamic.
- AI agents will need to react well to changes at runtime, and rely less on pregenerated solutions.

References

- Reynolds, C. 1999. *Steering Behaviors for Autonomous Characters*, GDC 1999. http://www.red3d.com/cwr/papers/1999/gdc99steer.html
- Stout, B. 2004. *Artificial Potential Fields for Navigation and Animation*, GDC 2004.
- Treuille, A., Cooper, S., Popovic, Z, 2006, *Continuum Crowds*, SIGGRAPH 2006. http://grail.cs.washington.edu/projects/crowd-flows/