CS 378: Computer Game Technology

Dynamic Path Planning, Flocking

Spring 2012

Dynamic Path Planning

- What happens when the environment changes *after* the plan has been made?
  - The player does something
  - Other agents get in the way (in this case, you know that the environment will change at the time you make the plan)
- The solution strategies are highly dependent on the nature of the game, the environment, the types of AI, and so on
- Three approaches:
  - Try to avoid the problem
  - Re-plan when something goes wrong
  - Reactive planning

Avoiding Plan Changes

- Partial planning: Only plan short segments of path at a time
  - Stop A* after a path of some length is found, even if the goal is not reached - use best estimated path found so far
    - Extreme case: Use greedy search and only plan one step at a time
    - Common case: Hierarchical planning and only plan low level when needed
  - Underlying idea is that a short path is less likely to change than a long path
  - But, optimality will be sacrificed
  - Another advantage is more even frame times
- Other strategies:
  - Wait for the blockage to pass - if you have reason to believe it will
  - Lock the path to other agents - but implies priorities

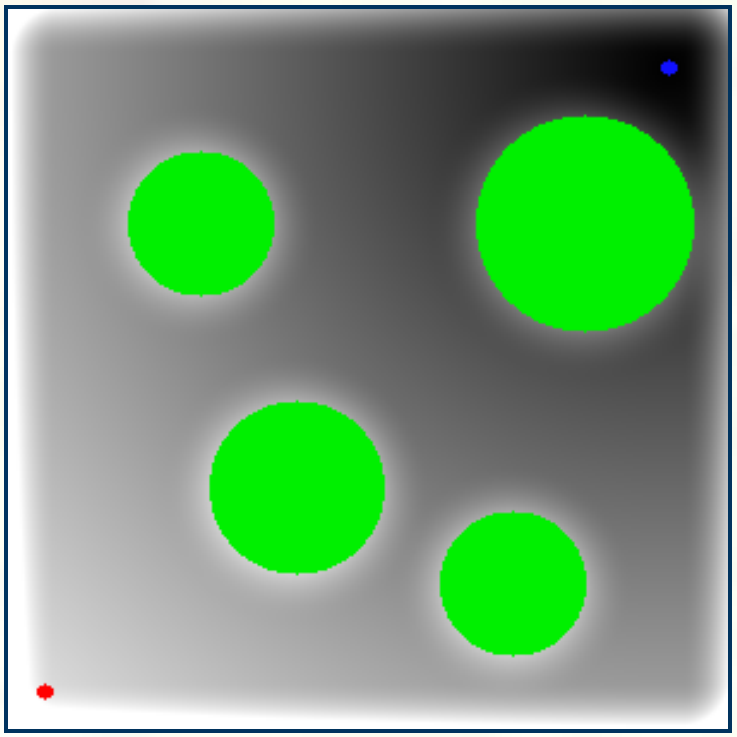
Re-Planning

- If you discover the plan has gone wrong, create a new one
- The new plan assumes that the dynamic changes are permanent
- Usually used in conjunction with one of the avoidance strategies
  - Re-planning is expensive, so try to avoid having to do it
  - No point in generating a plan that will be re-done - suggests partial planning in conjunction with re-planning

Reactive Planning

- A *reactive* agent plans only its next step, and only uses immediately available information
- Best example for path planning is *potential field* planning
  - Set up a force field around obstacles (and other agents)
  - Set up a gradient field toward the goal
  - The agent follows the gradient downhill to the goal, while the force field pushes it away from obstacles
  - Can also model velocity and momentum - field applies a *force*
- Potential field planning is reactive because the agent just looks at the local gradient at any instant
- Has been used in real robots for navigating things like hallways

Potential Field



- Red is start point, blue is goal
- This used a quadratic field strength around the obstacles
- Note that the boundaries of the world also contribute to the field

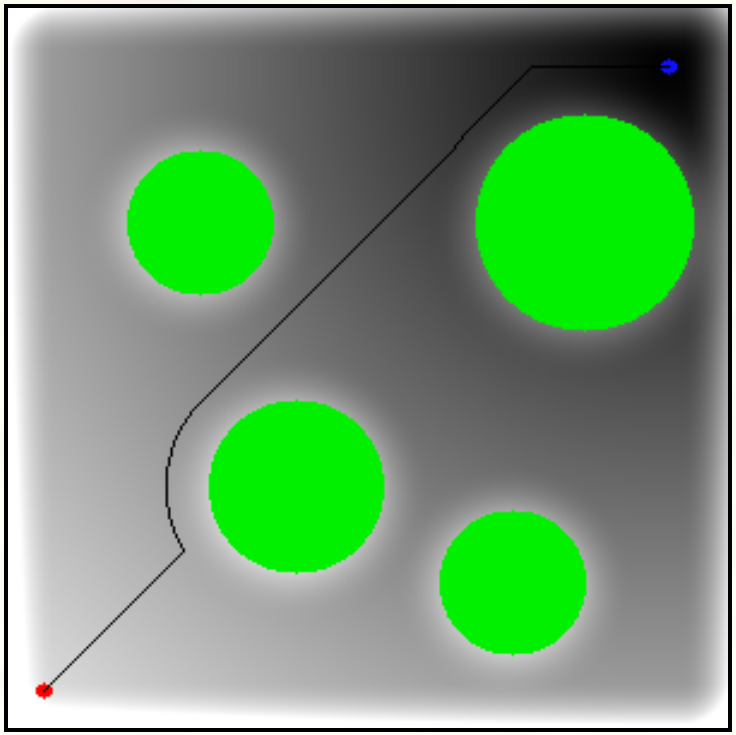
Creating the Field

- The constant gradient can be a simple linear gradient based on distance from the goal, d_{goal}: $f_{goal}=k\, d_{goal}$
- The obstacles contribute a field strength based on the distance from their boundary, $f_i(d_i)$
  - Linear, quadratic, exponential, something else
  - Normally truncate so that field at some distance is zero. Why?
  - Strength determines how likely the agent is to avoid it
- Add all the sub-fields together to get overall field
  - Do you recall what modeling technique this resembles?

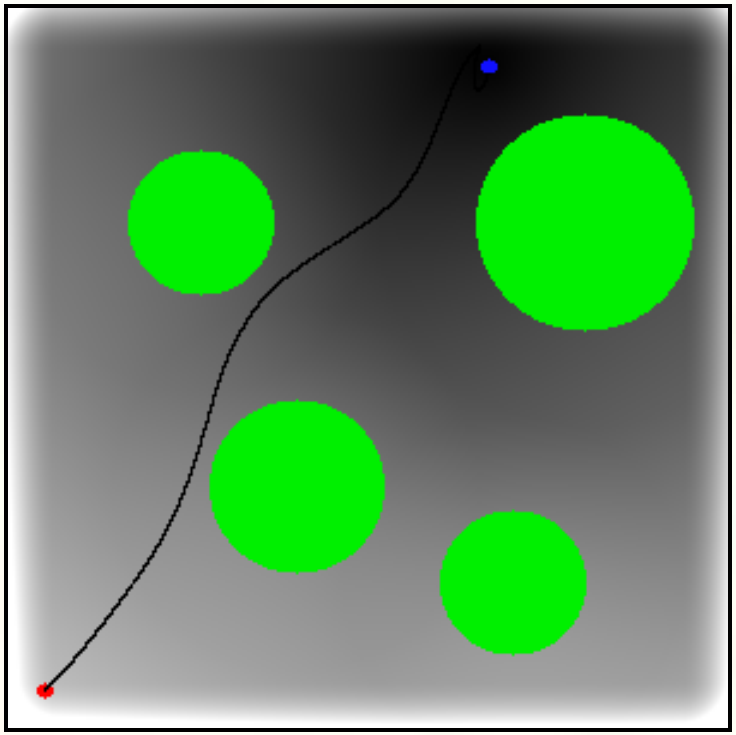
Following the Field

- At each step, the agent needs to know which direction is "downhill"
- Compute the gradient of the field
  - Compute the gradients of each component and add
  - Need partial derivatives in x and y (for 2D planning)
- Best approach is to consider the gradient as an acceleration
  - Automatically avoids sharp turns and provides smooth motion
  - Higher mass can make large objects turn more slowly
  - Easy to make frame-rate independent
  - But, high velocities can cause collisions because field is not strong enough to turn the object in time
  - One solution is to limit velocity - want to do this anyway because the field is only a guide, not a true force

Following Examples



No momentum - choose to go to neighbor with lowest field strength



Momentum - but with linear obstacle field strength and moved goal

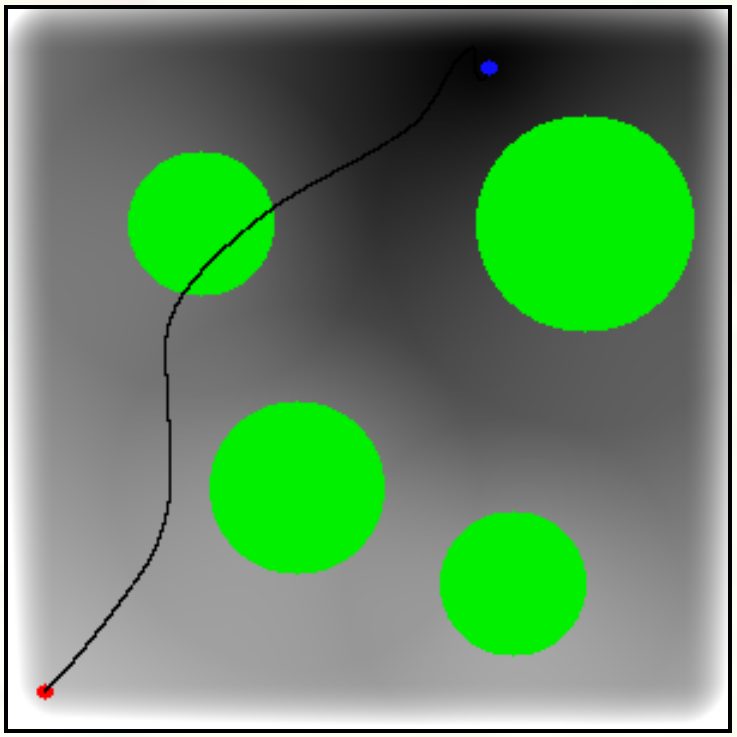
Discrete Approximation

- Compute the field on a grid
  - Allows pre-computation of fields that do not change, such as fixed obstacles
  - Moving obstacles handled as before
- Use discrete gradients
  - Look at neighboring cells
  - Go to neighboring cell with lowest field value
- Advantages: Faster
- Disadvantages: Space cost, approximate
- Left example on previous slide is a (very fine) discrete case

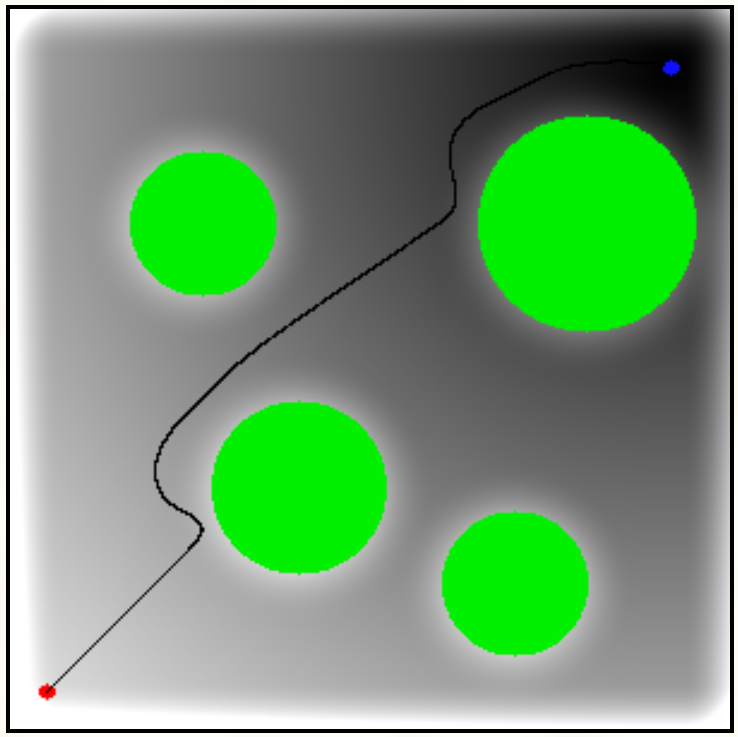
Potential Field Problems

- There are many parameters to tune
  - Strength of the field around each obstacle
  - Function for field strength around obstacle
  - Steepness of force toward the goal
  - Maximum velocity and mass
- Goals conflict
  - High field strength avoids collisions, but produces big forces and hence unnatural motion
  - Higher mass smoothes paths, but increases likelihood of collisions
- Local minima cause huge problems

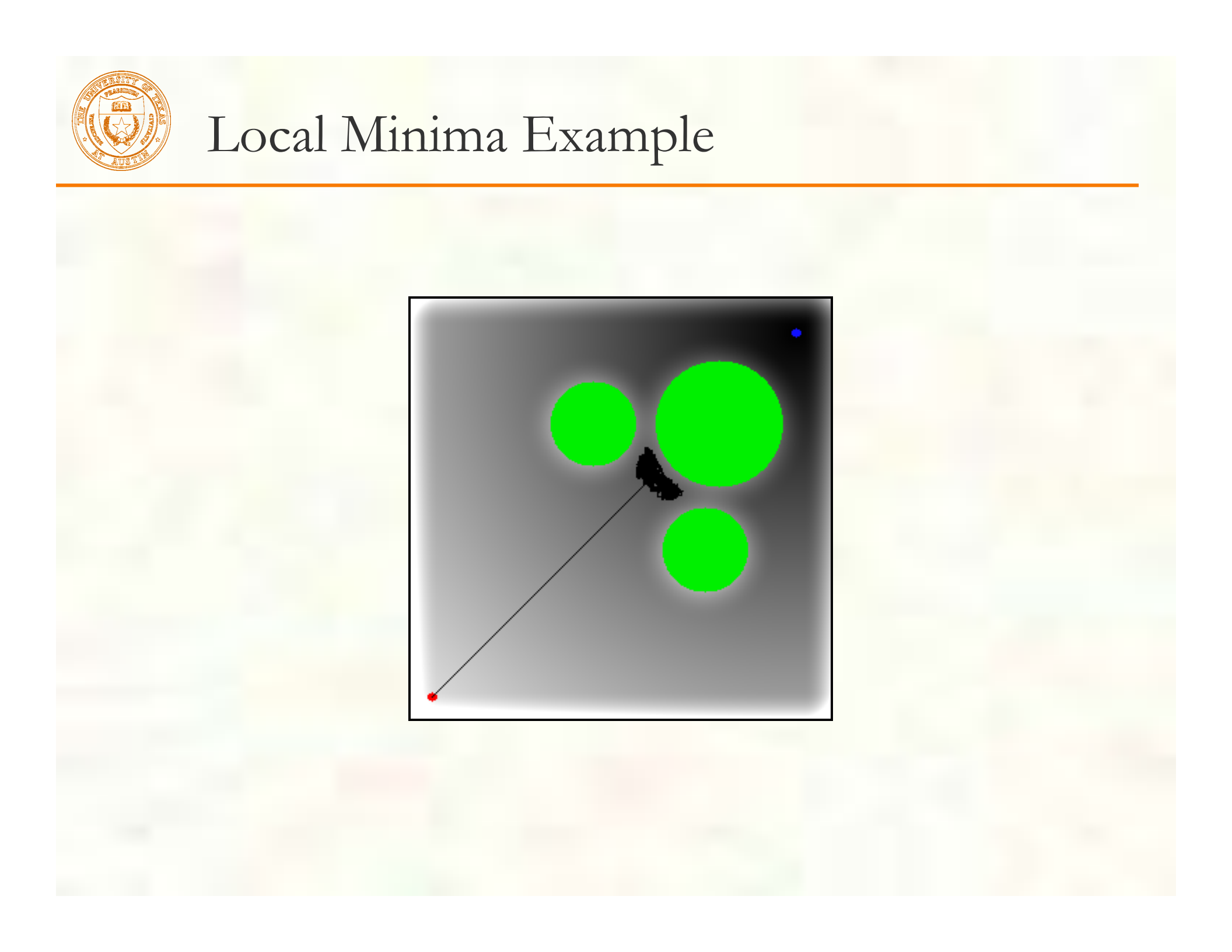
Bloopers



Field too weak



Field too strong



Local Minima Example

The Local Minima Problem

- Recall, path planning can be viewed as optimization
- Potential field planning is gradient descent optimization
- The biggest problem with gradient descent is that it gets stuck in local minima
- Potential field planning suffers from exactly the same problem
- Must have a way to work around this
  - Go back if a minima is found, and try another path
- With what sorts of environments will potential fields work best?
- What form of waypoint-based search is it similar to?

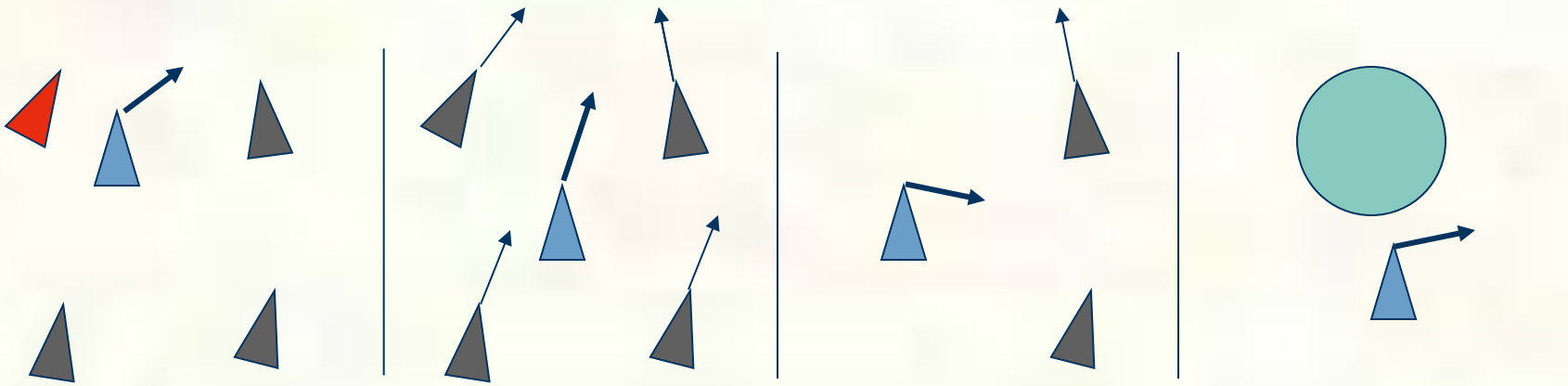
Flocking Models (Reynolds 87)

- Potential fields are most often used in avoiding collisions between the members of a group
  - Each member pushes on its neighbors to keep them from colliding
- Additional rules for groups can be defined - the result is a *flocking model,* or herding, or schooling, …
- Each rule contributes a desired direction, which are combined in some way to come up with the acceleration
- The aim is to obtain *emergent behavior*:
  - Define simple rules on individuals that interact to give interesting global behavior
  - For example, rules for individual birds make them form a flock, but we never explicitly specify a leader, or the exact shape, or the speed, …

Flocking Rules

- Separation: Try to avoid running into local flock-mates
  - Works just like potential fields
  - Normally, use a *perception volume* to limit visible flock-mates
- Alignment: Try to fly in same direction as local flock-mates
  - Gets everyone flying in the same direction
- Cohesion: Try to move toward the average position of local flock-mates
  - Spaces everyone out evenly, and keep boundary members toward the group
- Avoidance: Try to avoid obstacles
  - Just like potential fields

Rules Illustrated



Separation:
Fly away away from neighbors that are “too close”

Alignment: steer toward average velocity

Cohesion: steer toward average position

Avoidance: steer away from obstacles

Combining Commands

- Consider commands as accelerations
- Give a weight to each desire
  - High for avoidance, low for cohesion
- Option 1: Apply in order of highest weight, until a max acceleration is reached
  - Ensures that high priority things happen
- Option 2: Take weighted sum and truncate acceleration
  - Makes sure some part of everything happens

Flocking Demo

http://www.red3d.com/cwr/boids/

Flocking Evaluation

- Advantages:
  - Complex behavior from simple rules
  - Many types of behavior can be expressed with different rules and parameters
- Disadvantages:
  - Can be difficult to set parameters to achieve desired result
  - All the problems of potential fields regarding strength of forces

General Particle Systems

- Flocking is a special case of a *particle system*
- Objects are considered point masses from the point of view of simulating their motion (maybe with orientation)
- Simple rules are applied to control how the particles move
- Particles can be rendered in a variety of ways to simulate many different things:
  - Fireworks
  - Waterfalls, spray, foam
  - Explosions (smoke, flame, chunks of debris)
  - Clouds
  - Crowds, herds
- Widely used in movies as well as games
  - The ocean spray in *Titanic* and *Perfect Storm*, for instance

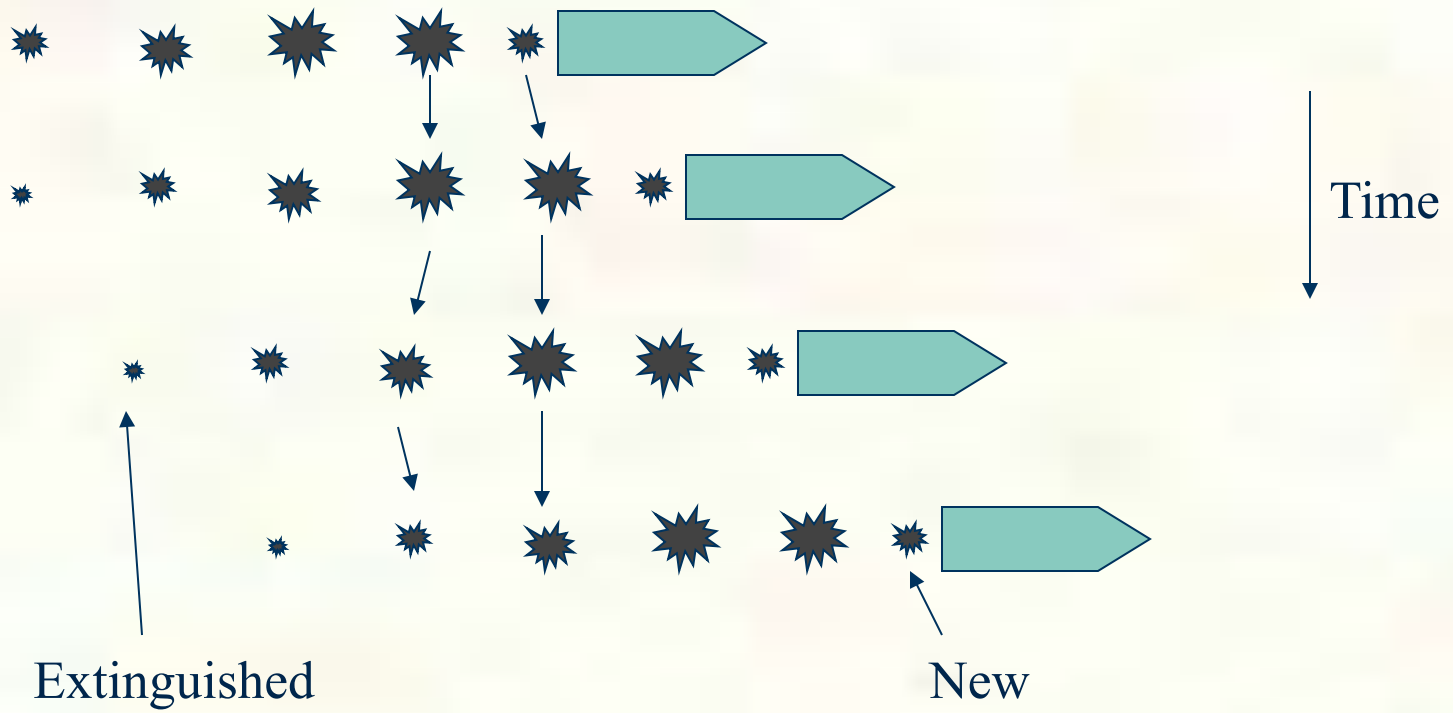
Particle System Step

1. Inject any new particles into the system and assign them their individual attributes
   - There may be one or more sources
   - Particles might be generated at random (clouds), in a constant stream (waterfall), or according to a script (fireworks)
2. Remove any particles that have exceeded their lifetime
   - May have a fixed lifetime, or die on some condition
3. Move all the current particles according to their script
   - Script typically refers to the neighboring particles and the environment
4. Render all the current particles
   - Many options for rendering

Example: Smoke Trails

- Particles are spawned at a constant rate
- They have zero initial velocity, or maybe a small velocity away from the rocket
- Rules:
  - Particles could rise or fall slowly (drift with the wind)
  - Attach a parameter, *density*, that grows quickly then falls over time
- Extinguish when density becomes very small
- Render with an billboard facing the viewer, scaled according to the density of the puff

Smoke Trails



Example: Explosions

- System starts when the target is hit
- Target is broken into pieces and a particle assigned for each piece
- Each particle gets an initial velocity away from the center of the explosion
- Particle rules are:
  - Move ballistically unless there is a collision
  - Could take into account rigid body rotation, or just do random rotation
  - Collisions are resolved by reflecting the velocity about the contact normal
- Rendering just draws the appropriate piece of target at the particles location

Particles Demo