
CMSC 498M: Chapter 9c Artificial Intelligence: Motion Planning II

Sources:

- Lecture notes from: S. Chenney (U. Wisconsin).
- Boids page (<http://www.red3d.com/cwr/boids/>) by Craig Reynolds.

Overview:

- Dynamic planning
- Potential-based path planning
- Flocking
- Particle systems

Chapter 9, Slide 1
Copyright © David Mount and Amitabh Varshney

Overview

- **Dynamic planning**
- Potential-based path planning
- Flocking
- Particle systems

Chapter 9, Slide 2
Copyright © David Mount and Amitabh Varshney

Dynamic Path Planning

Dynamic path planning: What happens when the environment changes after the plan has been made?

- The **player** alters the environment (e.g., blows up a bridge).
- Other **agents** get in the way.
- Moving **objects** (e.g., falling debris) blocks the way.

Possible approaches:

- Try to **avoid** the problem. (e.g. control your agents/world better.)
- **Re-plan** when something goes wrong.
- **Reactive** planning.

Chapter 9, Slide 3
Copyright © David Mount and Amitabh Varshney

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:** Use **greedy** search and only plan one step at a time.
 - Common:** **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.
- Optimality will be sacrificed.
- **Side Benefit:** more uniform frame rates.

Re-Planning: If your plan has gone **wrong**, create a **new** one,

- Assumes that the dynamic changes are now **permanent**,
- Usually used in conjunction with **avoidance** strategies:
 - Re-planning is **expensive**, so try to avoid having to do it.
 - No point in generating a plan that will be **redone**.

Chapter 9, Slide 4
Copyright © David Mount and Amitabh Varshney

Overview

- Dynamic planning
- Potential-based path planning
- Flocking
- Particle systems

Chapter 9, Slide 5
Copyright © David Mount and Amitabh Varshney

Reactive Planning

Reactive Agent:

- Plans **one step at a time**.
- Use only **local information** (not globally optimal).

Example: Potential-field path planning.

- Set up a **repulsive field** around obstacles (and other agents).
- Set up an **attractive 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 - potential field defines a **force**.

Why is this reactive?

- Agent's behavior depends only the **local gradient** at any instant.

Widely used in low-level robotics navigation.

Chapter 9, Slide 6
Copyright © David Mount and Amitabh Varshney

Potential Field Navigation

Key:

Red: Start point.

Blue: Goal point.

Green: Obstacles.

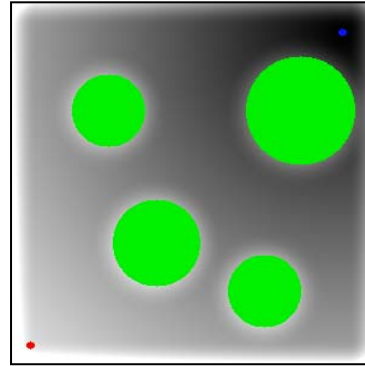
Field Strength: (Shown in gray scale)

Dark: Attractive.

Light: Repulsive.

Potential Function: is the sum of:

- **Repulsive force** around the obstacles (including the world boundary).
- **Attractive force** at the goal point.



Chapter 9, Slide 7
Copyright © David Mount and Amitabh Varshney

Path Planning with Potential Fields

Potential-Field Navigation:

Intuition: Moving in 2D space, the potential field is a **terrain**. It is **hard** to move **uphill**, and **easy** to slide **downhill**.

Path Plan: Agent follows laws of **physics** and rolls downhill.

More Formally: Your agent is moving subject to a **potential field** $U(p)$, that is defined at every point p of the environment. $U(p)$ defines the **strength** of the field (**height** of the terrain) at point p .

Gradient: Is a vector that points in the direction of **steepest descent**:

$$-\nabla U(p) = \left(-\frac{\partial U}{\partial x}, -\frac{\partial U}{\partial y} \right)^T$$

Force: The gradient behaves like a **force vector** $F(p) = -\nabla U(p)$, impelling the agent to steer downhill.

Chapter 9, Slide 8
Copyright © David Mount and Amitabh Varshney

Defining the Fields

Attraction towards Goal:

Naïve: Simple **linear potential** based on the **distance** from the goal:

$$U_{\text{goal}}(\mathbf{p}) = c \cdot \|\mathbf{p} - \mathbf{p}_{\text{goal}}\|$$

Where c is an adjustment parameter. Potential function is a cone.
The gradient is constant vector directed towards the goal.

Smarter: Use a **parabolic well potential**:

$$U_{\text{goal}}(\mathbf{p}) = \frac{c}{2} \cdot \|\mathbf{p} - \mathbf{p}_{\text{goal}}\|^2$$

The gradient still points towards the goal, but the magnitude of the gradient vector **increases** linearly with the **distance** from the goal.

$$\begin{aligned}\nabla U_{\text{goal}}(\mathbf{p}) &= \nabla \left(\frac{c}{2} \cdot \|\mathbf{p} - \mathbf{p}_{\text{goal}}\|^2 \right) = \frac{c}{2} \cdot \nabla \left(\|\mathbf{p} - \mathbf{p}_{\text{goal}}\|^2 \right) \\ &= c \cdot \|\mathbf{p} - \mathbf{p}_{\text{goal}}\| \left(\nabla \|\mathbf{p} - \mathbf{p}_{\text{goal}}\| \right)\end{aligned}$$

Chapter 9, Slide 9
Copyright © David Mount and Amitabh Varshney

Defining the Fields

Repulsion from Obstacles:

- Obstacle i contributes a field strength based on the distance from its boundary: $U_i(\mathbf{p}) = f(\|\mathbf{p} - \text{obst}_i\|)$, where f is an decreasing function of distance.
- Typical **potential functions**: (c is adjustment parameter)
 - Quadratic/Polynomial**: $f(d) = c/d^2$, or generally $f(d) = c/d^p$.
 - Negative Exponential**: $f(d) = c \cdot e^{-d}$.
- **Taper** so that field at some distance is zero. Why?
- **Strength** determines how likely the agent is to **avoid** it.

Total Field Strength: Add the sub-fields together.

(Does this remind you of a modeling technique we discussed?)

Chapter 9, Slide 10
Copyright © David Mount and Amitabh Varshney

Following the Field

Steepest Descent:

- At each step, the agent needs to know which direction is "downhill."

Total field gradient:

Compute: Gradients of each sub-field and sum.

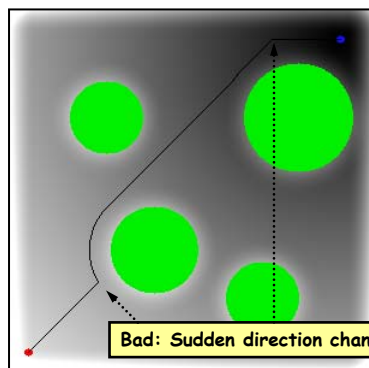
Gradient: The vector of **partial derivatives** in x, y, and z.

Gradient induces a Force: (and hence, acceleration)

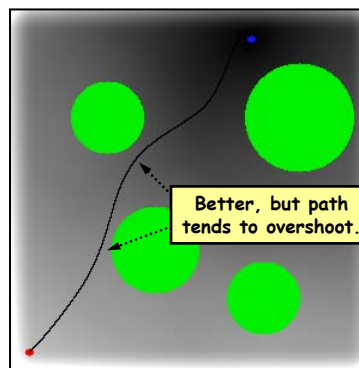
- Automatically **avoids sharp turns** and provides **smooth motion**.
- Higher mass can make large objects turn more **slowly**.
- Easy to make **frame-rate independent**.
- But, high velocities can cause **collisions** if field is not **strong enough** to turn the object in time.
- One solution is to **limit velocity** - want to do this anyway because the field is only a guide, not a true force.

Chapter 9, Slide 11
Copyright © David Mount and Amitabh Varshney

Following the Field: Examples



No momentum - Go in direction of lowest field strength.



Momentum - but with linear obstacle field strength and moved goal.

Chapter 9, Slide 12
Copyright © David Mount and Amitabh Varshney

Discrete Approximation

Discretization: Compute the field on a grid.

- Can **precompute** fixed sub-fields, e.g., for **fixed obstacles**.
- Sub-fields for moving obstacles computed with each time step.

Flow:

- Go to **neighboring node** with **smallest** field value.

Pros & Cons:

- **Faster**.
- Requires more **space** for grid.
- Only **approximate**.

Chapter 9, Slide 13
Copyright © David Mount and Amitabh Varshney

Potential Problems with Potential Fields

Requires lots of Tuning:

- **Strength** of the field around each obstacle.
- **Function** for field strength around obstacle (quadratic, exp, etc).
- **Steepness** of force toward the goal.
- **Maximum velocity** limit.

Goals can conflict:

High field strength: Avoids collisions, but produces big forces and hence jerky motion.

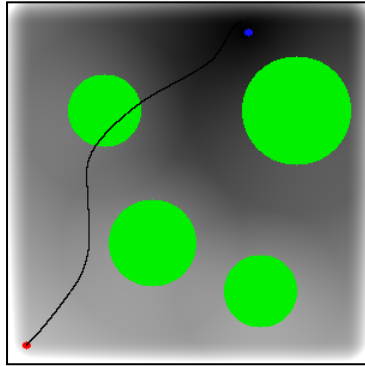
Low field strength: Smoother paths, but increases likelihood of collisions due to overshooting.

Local minima:

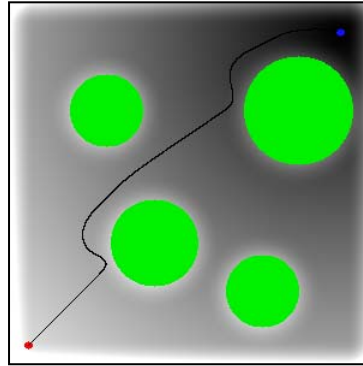
- Cannot see the **"big picture."** All decisions are **purely local**.
- May be arbitrarily far from optimal.
- Easy to get **trapped** in concavities.

Chapter 9, Slide 14
Copyright © David Mount and Amitabh Varshney

Potential-Field Problems: Examples



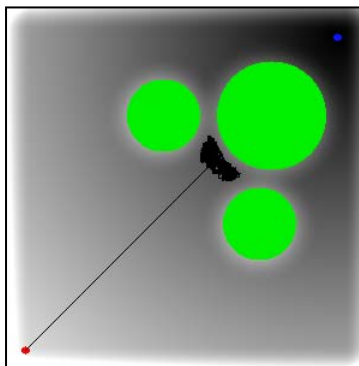
Field too weak:
Obstacle collision.



Field too strong:
Jerky motion.

Chapter 9, Slide 15
Copyright © David Mount and Amitabh Varshney

Potential-Field Problems: Examples



Getting stuck in local minima.

Chapter 9, Slide 16
Copyright © David Mount and Amitabh Varshney

Local Minima Problem

Problem Source:

- Path planning is an **optimization** problem, typically with many **local minima**.
- Potential field planning is a form of **gradient descent** optimization.
- Gradient descent methods seek **local minima**, and will get stuck unless other techniques are applied.

Possible Solutions:

Virtual Obstacles: Created at the point where the search gets stuck.

Virtual Sub-Goals: Backtrack along the path and attract it in a new direction.

Randomized Potential Field: Add a random noise to potential field values and try again. (Mixture of random walk and potential field.)

Chapter 9, Slide 17
Copyright © David Mount and Amitabh Varshney

Overview

- Dynamic planning
- Potential-based path planning
- **Flocking**
- Particle systems

Chapter 9, Slide 18
Copyright © David Mount and Amitabh Varshney

Flocking

Motion of multiple agents: Objectives:

- **Collision avoidance.**
- **Cohesion:**
 - Tendency to stick together.
 - **Examples:** School of fish, flock of birds, herd of cattle.
- **Common goal?**
 - Yes → Group should seek a common objective: A squadron of soldiers.
 - No → Motion is independent: People walking in a crowded street.

Emergent behavior:

- Define **simple rules** on individuals based on purely local information.
- Each rule induces a **force**. Total force affects acceleration.
- Interesting **global behavior** naturally emerges as a result.

Chapter 9, Slide 19
Copyright © David Mount and Amitabh Varshney

Flocking Rules

Boids: (virtual birds)

- Term coined by Craig Reynolds (1986).
- Behavior determined by the following 4 rules.

Separation:

Avoid collisions with nearby boids.
E.g., each boid generates a **repulsive potential field** of limited radius.

Alignment:

Align flight direction with neighboring boids.

Cohesion:

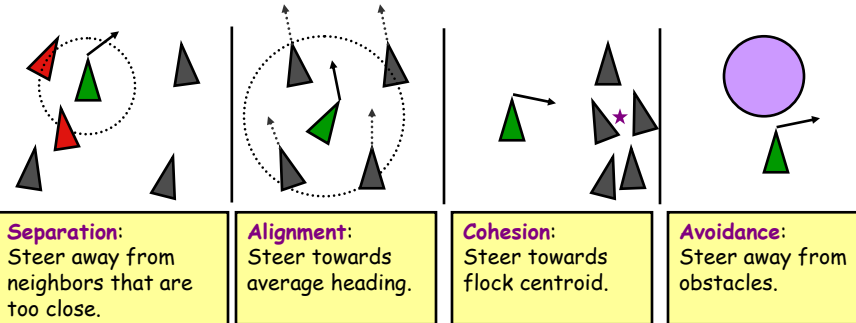
Attraction to centroid (center of mass) of the flock.
E.g., flock centroid generates an **attractive potential field**.

Avoidance:

Avoid obstacles in the environment.
E.g., each obstacles generates a **repulsive potential field**.

Chapter 9, Slide 20
Copyright © David Mount and Amitabh Varshney

Flocking Rules



Chapter 9, Slide 21
Copyright © David Mount and Amitabh Varshney

Combining Commands

Steering Rule:

- Steering rules act as **forces**; alter acceleration.
- Set a **maximum acceleration** to avoid **jerky** behavior.

Tuning Behavior:

- Assign a weight to each of the flocking rules.
- Avoidance should be high.
- Cohesion should be lower.

Option 1: Apply rules in decreasing weight order, until max acceleration is reached.

Responsive: Ensures that high priority forces are applied.

Option 2: Take weighted sum and truncate to max acceleration.

Fair: Ensures that all forces affect final motion.

Demo: <http://www.red3d.com/cwr/boids/>

Chapter 9, Slide 22
Copyright © David Mount and Amitabh Varshney

Flocking Evaluation

Advantages:

Simple: Complex behavior from simple rules.

Flexible: Many varieties of behavior from a small set of different rules and varying parameter settings.

Disadvantages:

Tuning: Can be difficult to set parameters to achieve desired result.

Local Minima: All the problems of potential fields regarding strength of forces.

Chapter 9, Slide 23
Copyright © David Mount and Amitabh Varshney

Overview

- Dynamic planning
- Potential-based path planning
- Flocking
- Particle systems

Chapter 9, Slide 24
Copyright © David Mount and Amitabh Varshney

General Particle Systems

Particle system:

- A variation on the theme of **flocking**.
- Particles are **light-weight objects**: E.g., point masses.
- Simple rules **control** how particles:
 - are born
 - move, grow/shrink, change appearance
 - die.

Flexible: Can model many **complex, distributed phenomena**.

- Fireworks.
- Waterfalls, spray, foam.
- Explosions (smoke, flame, chunks of debris).
- Clouds.
- Crowds, flocks, herds.

Widely used in movies and games.

Chapter 9, Slide 25
Copyright © David Mount and Amitabh Varshney

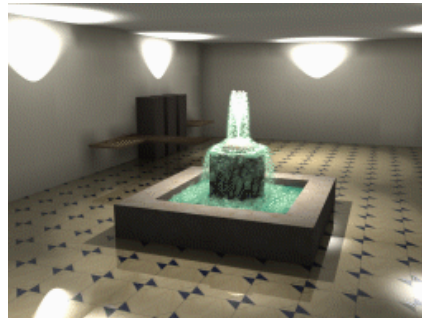
Particle System Demos

Demo source: Flow particle animation application by Mark B. Allan.

<http://users.rcn.com/mba.dnai/software/flow/>

Demos:

- [Kettle](#)
- [Face](#)
- [Smoke Ball](#)
- [Lava2](#)
- [Molten](#)



Chapter 9, Slide 26
Copyright © David Mount and Amitabh Varshney

Particle System Steps

1. Inject any **new particles** into the system and assign them their initial **attributes**.
 - There may be multiple sources.
 - Particles might be generated at **random** (clouds), in a **constant stream** (waterfall), or according to a **script** (fireworks).
2. **Remove** (kill) 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 **scripts**.
 - Script typically refers to the neighboring particles and the environment.
4. **Render** all the current particles.
 - Many options for rendering.

Chapter 9, Slide 27
Copyright © David Mount and Amitabh Varshney

Particle System Example: Puffs of Smoke

Birth:

- Particles are spawned at a **constant rate**.
- They have **zero initial velocity**, or maybe a small velocity away from the rocket.

Update Rules:

- Particles rise or fall slowly (drift with the wind).
- Current particle state described by parameter, **size**, that grows quickly then falls over time.

Death:

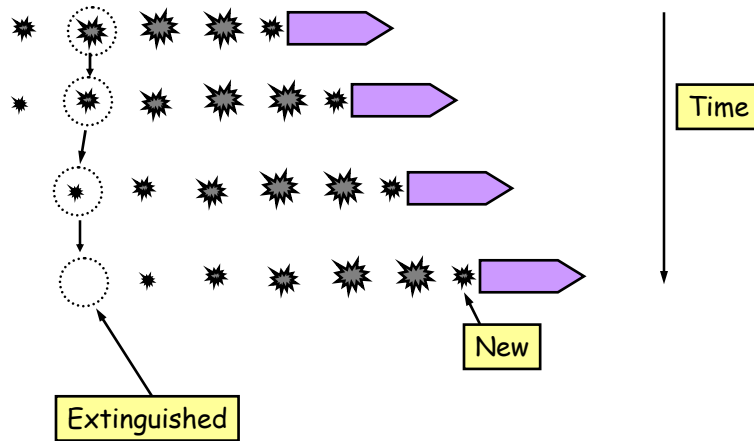
- Kill particle when **size** becomes very small.

Render:

- Scale **size** depending on the value of size.

Chapter 9, Slide 28
Copyright © David Mount and Amitabh Varshney

Particle System Example: Puffs of Smoke



Chapter 9, Slide 29
Copyright © David Mount and Amitabh Varshney

Particle System Example: Explosions

Birth:

- System starts when the target is **explodes**.
- Target is broken into **many small pieces**.
- One **particle** assigned for **each piece**.
- Each particle is assigned an **initial velocity** away from the center of the explosion.

Update Rules: Doesn't need to be perfect, just convincing.

- Move **continuously** unless there is a **collision**.
- Model accurate rigid body rotation, or just do **random rotation**.
- Collisions handled as with **colliding balls**. (Recall physics lectures.)

Death: (not applicable)

Rendering:

- Draw the particle at the current position and orientation.

Chapter 9, Slide 30
Copyright © David Mount and Amitabh Varshney

Summary

Summary:

- Dynamic planning
- Potential-based path planning
- Flocking
- Particle systems