Physical Simulation

Elements of Graphics
CS324e
Spring 2017
Newton’s Equations of Motion

- Equations that describe motion over time
- Provide model for relating forces to object trajectory
  - \( F = ma \)
- Integrating over time captures a system’s physical behaviors
- How are we discretizing these equations for computer simulation?
Particles Along a Trajectory

- Particle has a position and a velocity
- Calculate position over time by starting at a point and considering velocity for that given time interval
Euler’s Method

- Take linear time steps ($\Delta t$) along the flow:

$$\tilde{x}(t + \Delta t) = \tilde{x}(t) + \Delta t \cdot \tilde{x}(t) = \tilde{x}(t) + \Delta t \cdot g(\tilde{x},t)$$

- Write as a time iteration:

$$\tilde{x}^{i+1} = \tilde{x}^i + \Delta t \cdot \tilde{v}^i$$

- Visualized across time steps:
Accounting for Mass

- Particle has mass $m$
- Particle is in a force field $\mathbf{f}$
- Newton’s Second Law:

$$\mathbf{f} = m\mathbf{a} = m\ddot{x}$$
Particle With Mass Example

//Class method to apply forces
applyForces(float fx, float fy) {

    // Class fields
    float x, y;
    float vx, vy;
    float ax, ay;
    float m;

    ax = fx/m;
    ay = fy/m;
    vx += ax;
    vy += ay;
    x += vx;
    y += vy;
}

Problems

- Inaccurate over larger time steps
- Creates numeric instabilities as error accumulates
- Better, more stable methods exist, so explicit Eulerian is rarely used
- But it should be okay for our purposes in this class!
Verlet Integration

- A slightly better solver that doesn’t require much additional mathematics

- Consider our forward Euler equations:

  \[ v_{t+1} = v_t + a \Delta t \]

  \[ p_{t+1} = p_t + v_{t+1} \Delta t \]

- Verlet looks like this:

  \[ p_{t+1} = p_t + (p_t - p_{t-1}) + a \Delta t^2 \]

  \[ p_{t-1} = p_t \]
Spring Forces

- Spring force is based on:
  - Spring stiffness \((k)\)
  - Amount of stretch from resting position \((X)\)
- Hooke’s Law: \(f = -kX\)
Spring Example

float y;
float vy;
float m = 1.0;
float ry = 250;
float ks = 0.1;

void setup() {
    size(500, 500);
}

void draw() {
    background(210);
    float f = -(ks * (y - ry));
    float a = f/m;
    vy = vy + a;
    y += vy;
    rect(200, y, 100, 20);
}
Spring Damping

- If force due to a spring is:
  \[ F = -k_s X \]

- Spring force with damping is:
  \[ F = -k_s X - k_d v \]
Dampening Force

```cpp
float y;
float vy;
float m = 1.0;
float ry = 250;
float ks = 0.1;
float kd = 0.1;

void setup() {
    size(500, 500);
}

void draw() {
    background(210);
    float f = -((ks * (y - ry)) + kd*vy);
    float a = f/m;
    vy = vy + a;
    y += vy;
    rect(200, y, 100, 20);
}
```
Further Extensions

- Fixed-length springs (springs that have a resting distance between the end positions) resemble physical-world springs
- Multi-part system of springs resemble ropes and cords etc
Uses of Springs

❖ A sequence of particles can simulate:
   ❖ Hair
   ❖ Rope
   ❖ Grass
❖ A network of particles can simulate:
   ❖ Cloth
Hands-on: Using Masses and Springs

Today’s activities:

1. Implement the basic mass example, and experiment with different types of masses and forces

2. Implement the basic spring example, and experiment with different types of stiffness and dampening

3. Create a sequence of multiple particles connected by springs, where each particle’s position is based on the previous particle’s position. Include mouse controls, so the sequence can be moved around the screen