Dynamics

Adam Mally
CIS 700-006 Spring 2017
University of Pennsylvania

Advanced Character Physics

- Much of these slides' content is based on the paper "Advanced Character Physics" by Thomas Jakobsen
- Discusses algorithms developed for IO Interactive's game "Hitman: Codename 47"
  - Released in November of 2000!
  - Ragdoll physics
  - Cloth sim
  - Rigid body collisions

Particle Systems

- The basis for most physics simulations is the **particle system**
- Easy to compute the motion of simple points in space
- A fun demonstration

Particle Systems: Euler integration

- Particles usually have three core attributes
  - Position
  - Velocity
  - Acceleration
- We use a repeating update function to simulate the motion of our particles:

  $p' = p + v \cdot \Delta t$

  $v' = v + a \cdot \Delta t$
- The equations above are simple Euler integration

Particle Systems: Verlet integration

- Euler integration is rather unstable when our time step is too large
- What if we tried Verlet integration?
  - Velocity is implicit since we store our previous position p*

$$p' = 2p - p^* + a \cdot \Delta t^2$$

$$p^* = p$$

- More difficult for position and velocity to become desynchronized

Particle Systems: Verlet integration

$p' = 2p - p^* + a \cdot \Delta t^2$

$p^* = p$

- The logic behind this becomes clearer if we rewrite it slightly:

$p' = p + (p - p^*) + a \cdot \Delta t^2$

- The statement within the parentheses is an approximation of the particle's current velocity, with the Δt already multiplied in

Particle Systems: Runge-Kutta Integration

- Pronounced (sort of) “roon-geh koot-tuh”
  - They’re German last names
  - We’re not disassembling a ladder
- A method that can be used to numerically solve differential equations, provided we can define our equation in the form:

  $y' = f(t, y)$

  $y(t_0) = y_0$

Particle Systems: Runge-Kutta Integration

- A bit more complex to write out than Verlet or Euler integration, but is numerically stable for larger time steps
  - Useful if you want to simulate something in real time

$$y_{n+1} = y_n + h/6(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + h/2, y_n + k_1 \cdot h/2)$$

$$k_3 = f(t_n + h/2, y_n + k_2 \cdot h/2)$$

$$k_4 = f(t_n + h/2, y_n + k_3 \cdot h)$$

- **h is our time step**
- We're evaluating the slope of our function at four points along our time interval: k_1 is essential Euler integration, k_2 and k_3 are midpoint slopes, and k_4 is the slope at the end of the interval

Handling collisions

- We have methods for moving our particles around in a vacuum
- How can we make particles interact with everything else in the scene?
  - Let’s not worry about particle-particle interaction just yet

Handling collisions

- We just project our particles back out of the colliding objects
- Translate them along the normal vector of the surface through which they moved
- What will be the effective motion of our particles when they collide with surfaces?
  - Euler integration case?
  - Verlet integration case?

Handling collisions

- We just project our particles back out of the colliding objects
- Translate them along the normal vector of the surface through which they moved
- What will be the effective motion of our particles when they collide with surfaces?
  - Euler integration case?
  - Verlet integration case?
- They'll slide along the surfaces they hit, generally

Handling collisions: Example

- What if we want to force our particles to stay inside a 1000x1000 box?
- How might we code this?

Handling collisions: Example

- What if we want to force our particles to stay inside a 1000^3 box?
- How might we code this?

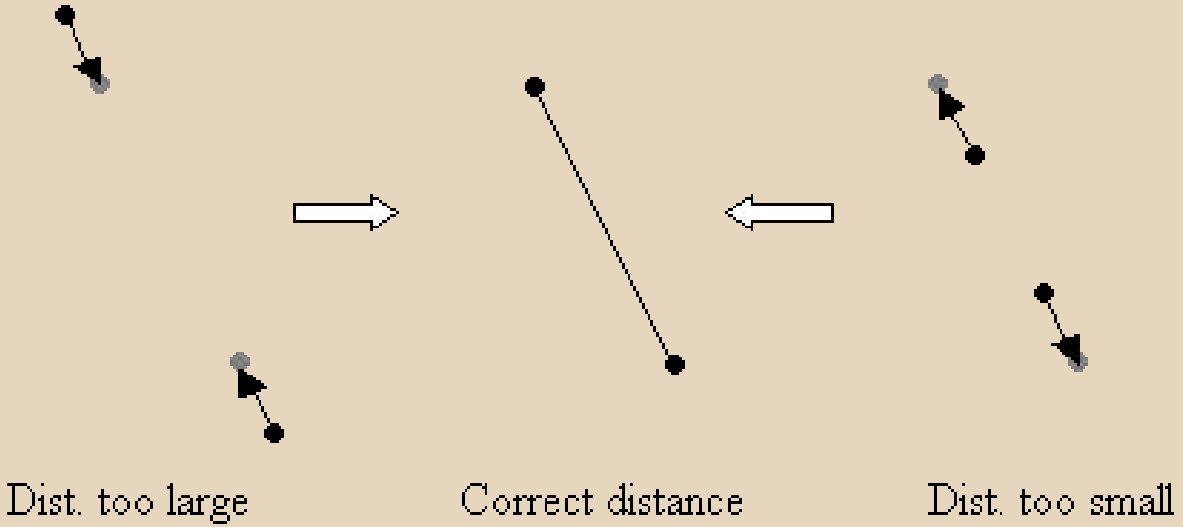
```
for(Particle p : particles)
   p = min(max(p, vec3(0)), vec3(1000))
```

Particle Constraints

- What if we force certain particles to stay within certain distances of one another?
- Let's say we have two particles, and we want them to always remain 100 units away from each other
  - $|p_1 - p_2| = 100$
- If we integrate as before, it's likely that the particles will soon break our constraint
- How do we fix this?

Particle Constraints

- We just move both particles closer if they're too far away, and push them away if they're too close



Particle Constraints

- The end result is as though we've attached a spring with infinite stiffness to the particle pair
  - Visually, the particles are *always* 100 units away
  - We fix their positions between frames
- But what if we want to impose additional constraints on our particles?
  - Let's say they both have to fit inside a box with side lengths at least 100 units long
- We may repeatedly invalidate one constraint, then the other

Particle Constraints: Relaxation

- Let’s just repeatedly solve for our constraints until they are both satisfied

```
while(!c1.satisfied() && !c2.satisfied()) {
   //Force particles to be 100 units away
   //Force particles to fit in the box
}
```

- Seems naïve, but it will converge to the desired result
- Known as Jacobi iteration, or just “relaxation”

Particle Constraints: Relaxation

- Can improve relaxation by making it adaptive based on the amount of state change applied by the previous loop of relaxation
- Can change to a FOR loop to stop it early to avoid long or infinite iteration
  - May not be exactly physically correct, but should be close enough
- What happens if we make our springs less stiff?

Cloth Simulation: Springs

- Let’s say we have a collection of particles where each particle is attached to at least two others via springs
- We relax every particle each frame
  - As it turns out, we only need to perform one iteration in relaxation
- Hence, our algorithm becomes O(n), which is great for real time purposes

Cloth Simulation: Springs

- Let’s discuss how to represent springs in code
- We need some sort of constraint structure:

```
struct Constraint {
   Particle *p1, *p2;
   float restLength;
};
```

Cloth Simulation: Springs

- We also need to write the code that applies our spring constraints to the particles
- Given a constraint c between two particles:

```
Particle& p1 = *(c.p1); Particle& p2 = *(c.p2);
vec3 diff = p2 - p1;
float dist = sqrt(diff.x^2 + diff.y^2 + diff.z^2); // Costly!
float diffNorm = (dist - c.restLength) / dist;
p1 += diff * 0.5 * diffNorm;
p2 -= diff * 0.5 * diffNorm;
```

- We use this code on all constraints for N iterations

Cloth Simulation: Springs

- We can handle particles of nonuniform mass as well

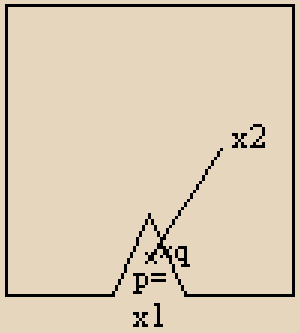
```
Particle& p1 = *(c.p1); Particle& p2 = *(c.p2);
vec3 diff = p2 - p1;
float dist = sqrt(diff.x^2 + diff.y^2 + diff.z^2); // Costly!
float diffNorm = (dist - c.restLength) /
    (dist * (1/p1.mass + 1/p2.mass));
p1 += diff * 1/p1.mass * diffNorm;
p2 -= diff * 1/p2.mass * diffNorm;
```

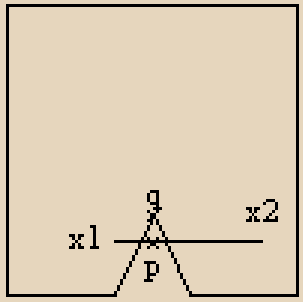
Rigid Bodies

- We can represent rigid bodies as collections of particles constrained by infinitely stiff springs
- This is fine as long as the world is composed of convex objects
- We have to handle concave shapes specially

Rigid Bodies

- We’re going to hand wave much of this part now because it is hard! Yay!
- Say we have a line that intersects a concave scene
- We need to know how far it sticks into the collider
- If we know this distance as well as the shortest path to make the line leave the collision, we can translate both endpoints to satisfy our collision constraints





Miscellaneous particle concepts

- Joints holding together rigid bodies
  - A pin joint is a single particle that is shared by two rigid bodies
  - A hinge joint is a pair of particles that are shared by two rigid bodies

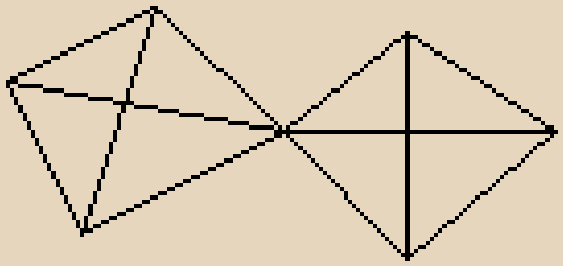


Figure 7a. A pin joint.

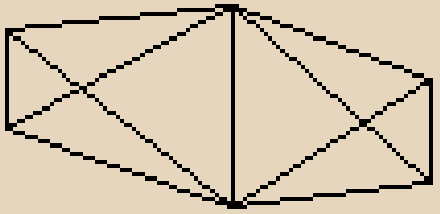


Figure 7b. A hinge.

Miscellaneous particle concepts

- How might we simulate an entire human body using particles and constraints?

Miscellaneous particle concepts

- How might we simulate an entire human body using particles and constraints?

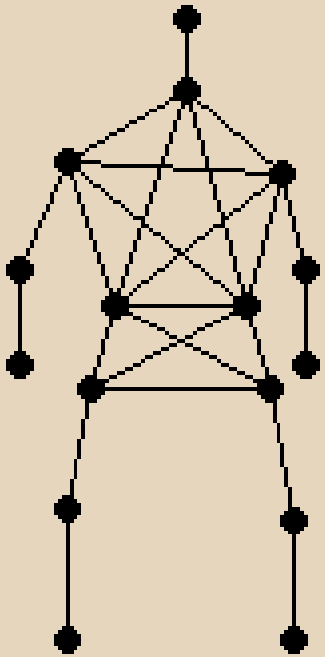


Figure 9. The particle/stick configuration used in Hitman for representing the human anatomy.

Miscellaneous particle concepts

- How can we determine the strength of friction that should be applied to an object colliding with a surface?

Miscellaneous particle concepts

- How can we determine the strength of friction that should be applied to an object colliding with a surface?

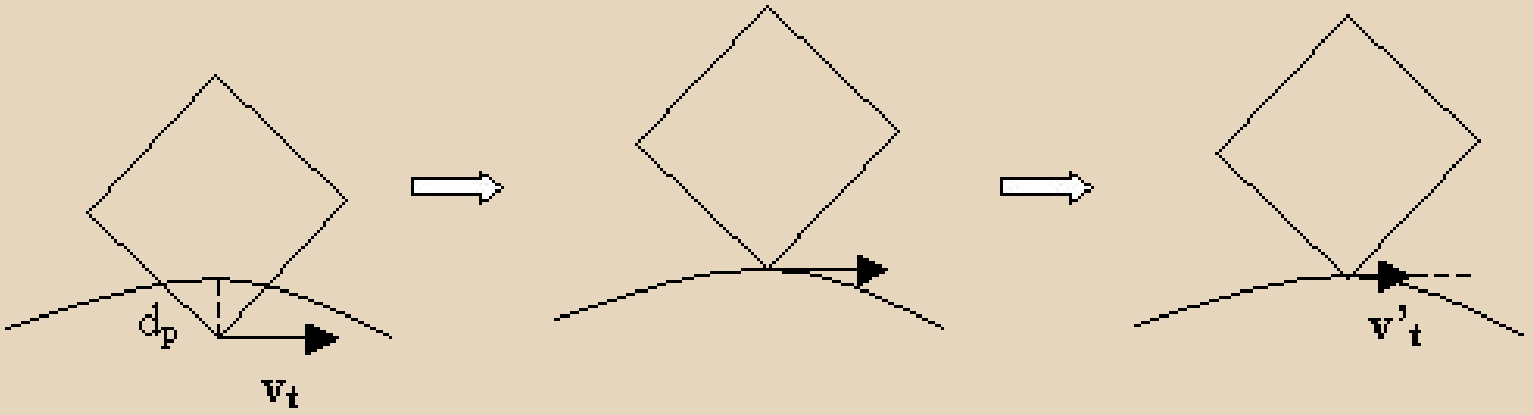


Figure 10. Collision handling with friction (projection and modification of tangential velocity).