Game Physics

(Particle System)
Introduction to Game Physics

- Traditional game physics
  - Particle system
  - Rigid body dynamics
  - Flexible body dynamics
- Some state-of-art topics
  - Car physics
  - Fluid dynamics
  - Rag-doll physics
- Physics
  - Rigid body kinematics
  - Newton’s Laws
  - Forces
  - Momentum
  - Energy
Introduction to Particle System

- Point mass
  - Mass center
- Newton’s Laws
  - Forces
    - Acceleration
    - Velocity
    - Position
- Easy to implement
- Good for game FXs
  - Attach a billboard on each particle
  - A basic feature of a game engine
- Fundamentals of Game Physics
In general
- Particle system for combat effects

First-person shooting
- Rag-doll system
  - Physical character motion simulation
- Rigid-body dynamics
- Fracture mechanics

Sports
- Rag-doll system
  - Physical character motion simulation
- Fluid dynamics

Car racing
- Car physics
- Rigid-body dynamics
- Fracture mechanics
Game Physics Applications in Games (2/2)

- Fighting
  - Flexible-body dynamics
    - Cloth simulation
    - Rag-doll
  - Etc
Game Physics Middleware (Physics Engine)

- Freeware
  - ODE
    - Open Dynamics Engine (ODE)
    - http://www.ode.org/
- Commercial ones
  - nVIDIA PhysX (Ageia PhysX)
    - Powered by CUDA
  - Harvok
    - http://www.havok.com/
Differential Equation Basics

- Initial value problems
  - Ordinary differential equation
- Numerical solutions
  - Euler’s method
  - The midpoint method
  - Runge-Kutta method
An ODE

\[ \dot{x} = f(x, t) \]

where \( f \) is a known function
\( x \) is the state of the system, \( \dot{x} \) is the \( x \)'s time derivative

\( x \) & \( \dot{x} \) are vectors
\( x(t_0) = x_0 \), initial condition

Vector field
Solutions
- Symbolic solution
- Numerical solution
Euler’s Method

\[ x(t + \Delta t) = x(t) + \Delta t \, f(x, t) \]

- A numerical solution
  - A simplification from Tayler’s series
- Discrete time steps starting with initial value
- Simple but not accurate
  - Bigger steps, bigger errors
  - \( O(\Delta t^2) \) errors
- Can be unstable
- Not even efficient
The Runge-Kutta Method

- Runge-Kutta method of order 4
  - $O(h^5)$

\[
\begin{align*}
  k_1 &= h f(x_0, t_0) \\
  k_2 &= h f(x_0 + k_1/2, t_0 + h/2) \\
  k_3 &= h f(x_0 + k_2/2, t_0 + h/2) \\
  k_4 &= h f(x_0 + k_3, t_0 + h) \\
  x(t_0+h) &= x_0 + \frac{1}{6} k_1 + \frac{1}{3} k_2 + \frac{1}{3} k_3 + \frac{1}{6} k_4
\end{align*}
\]
Newton’s Laws (1/3)

- First Law
  - In the absence of external forces, an object at rest will remain at rest.
  - If the object is in motion and no external forces acting on it, the object remains in motion with constant velocity.
  - Only forces can change an object’s motion.
Second Law

For an object with constant mass over time, its acceleration $a$ is proportional to the force $F$ and inversely proportional to the mass $m$ of the object: $a = F/m$.

If the mass changes over time, the more general statement of the law is

$$F = \frac{d}{dt} (mv) = ma + \frac{dm}{dt} v$$

An object’s path of motion is determined from applied forces.
Newton’s Laws (3/3)

- **Third Law**
  - If the force exerted on one object, there is a force with equal magnitude but opposite direction on some other that interacts with it.
  - Action and reaction always occur between interacting objects.
Forces

- Gravitational forces
- Spring forces
- Friction and other dissipative forces
Gravitational Forces (1/2)

- Given two point masses $m$ and $M$ that have gravitational interaction, they attract each other with forces of equal magnitude but opposite direction, as indicated by Newton’s third law.

\[
F_{\text{gravity}} = \frac{GmM}{r^2}
\]

\[
G = 6.67 \times 10^{-11} \text{ newton-meter}^2 \text{ per kg}^2
\]

(universal gravitational constant)
Assume the earth’s surface is flat and the direction of gravitation force is normal to the plane, the gravitational force exerted on the object by the earth is:

\[ F = -mgU \]

\[ g = 9.81 \text{ meters per sec}^2 \]

\( U \) as unit-length upward direction
Spring Forces

- Hooke’s Law

\[ F = -c \Delta U \]

- \( c > 0 \), spring constant
- \( \Delta U \), displacement

Pull

\[ F \quad \Delta U > 0 \]

Push

\[ F \quad \Delta U < 0 \]
A dissipative force is one for which energy of the system decreases when motion takes place.

A simple model for a dissipative force:

\[ F = -c \ |v|^n \]

Two common cases:

- Friction
- Viscosity
A frictional force between two objects in contact opposes the sliding of one (moving) object over the surface of adjacent object (nonmoving).

The friction force is tangent to the contact surface and opposite in direction to the velocity of the moving object.

The magnitude of the frictional force is assumed to be proportional to the magnitude of the normal force between surfaces.

\[ c_k = \frac{F_{\text{friction}}}{F_{\text{normal}}} \]

The kinetic friction force is modeled as

\[ F = -c_k \frac{v}{|v|}, \quad \text{if } v \text{ is not zero} \]
\[ = 0, \quad \text{if } v \text{ is zero} \]

\( c_k \) is referred to as the coefficient of kinetic friction.
Friction (2/2)

- Static friction
  - $c_s$, coefficient of static friction

$$c_s = \frac{\text{max}(F_{\text{friction}})}{F_{\text{normal}}}$$

**Static case**

$$\theta_1$$

**Kinetic case**

$$\theta_2 > \theta_1$$
Viscosity

- A typical occurrence of this type of force is when an object is dragged through a thick fluid.
- The force is modeled to have the direction opposite to that of the moving object.

\[ F = -F_{\text{dissipative}} \frac{\mathbf{v}}{|\mathbf{v}|} = -(c|\mathbf{v}|) \frac{\mathbf{v}}{|\mathbf{v}|} = -c\mathbf{v} \]

\[ c > 0 \]
Particle Dynamics

- Particles are objects with
  - Mass
  - Position
  - Velocity
  - Respond to forces
- But no spatial extent (no size!)
  - Point mass
- Based on Newton Laws
  - \( f = ma \)
  - \( \ddot{x} = f / m \)
  - \( \dot{v} = f / m, \quad v = \dot{x} \)
Euler’s Method for Particle System

- Discrete time steps starting with initial value
  - \( t = t_0, t_1, t_2, \ldots \)
  - Fixed frame rate but adaptive sampling rate
- Simple but not accurate
  - Bigger steps, bigger errors
  - \( O(\Delta t^2) \) errors
- Can be unstable
- Not even efficient
  - If the time step is very tiny
- Calculate the velocity first
  - Then the position

\[
\begin{align*}
\mathbf{v}(t + \Delta t) &= \mathbf{v}(t) + \Delta t \mathbf{f}'(\mathbf{v}, t) \\
\mathbf{x}(t + \Delta t) &= \mathbf{x}(t) + \Delta t \mathbf{f}(\mathbf{x}, t)
\end{align*}
\]
Particle States

- Position
- Velocity
- Mass
- Life
- Etc
  - Bounce rate
  - Opacity
  - Color
- In game application, we always put a billboard object with texture animation on a particle to simulate the geometry shape of the particle.
typedef struct {
    float m;     /* mass */
    float *x;    /* position */
    float *v;    /* velocity */
    float *f;    /* force accumulator */
} *Particle;

typedef struct {
    Particle *p  /* array of pointers to particles */
    int n;       /* number of particles */
    float t;     /* simulation clock */
} *ParticleSystem;
/* gather states from the particles */

void ParticleGetState(ParticleSystem p, float *dst) {
    int i;
    for (i = 0; i < p->n; i++) {
        *(dst++) = p->p[i]->x[0];
        *(dst++) = p->p[i]->x[1];
        *(dst++) = p->p[i]->x[2];
        *(dst++) = p->p[i]->v[0];
        *(dst++) = p->p[i]->v[1];
        *(dst++) = p->p[i]->v[2];
    }
}
/* scatter states into the particles */
void ParticleSetState(ParticleSystem p, float *src)
{
    int i;
    for (i = 0; i < p->n; i++) {
        p->p[i]->x[0] = *(src++);
        p->p[i]->x[1] = *(src++);
        p->p[i]->x[2] = *(src++);
        p->p[i]->v[0] = *(src++);
        p->p[i]->v[1] = *(src++);
        p->p[i]->v[2] = *(src++);
    }
}
/* calculate derivative, place in dst */
void ParticleDerivative(ParticleSystem p, float *dst) {
    int i;
    ClearForce(p);
    ComputeForce(p);

    for (i = 0; i < p->n; i++) {
        *(dst++) = p->p[i]->v[0];
        *(dst++) = p->p[i]->v[1];
        *(dst++) = p->p[i]->v[2];
        *(dst++) = p->p[i]->f[0]/p->p[i]->m;
        *(dst++) = p->p[i]->f[1]/p->p[i]->m;
        *(dst++) = p->p[i]->f[2]/p->p[i]->m;
    }
}
/* Euler Solver */
void EulerStep(ParticleSystem p, float DeltaT)
{
    ParticleDeriv(p, temp1);
    ScaleVector(temp1, DeltaT);
    ParticleGetState(p, temp2);
    AddVector(temp1, temp2, temp2);
    ParticleSetState(p, temp2);
    p->t += DeltaT;
}