Dynamic Physics for Simulation and Game Programming

Discrete Dynamics

2. Force equals mass times acceleration (F=ma)

\[ a = \frac{F}{m} \]

\[ \frac{\Delta v}{\Delta t} = a \rightarrow \Delta v = a \Delta t \rightarrow v = \sum \Delta v = \sum a \Delta t \]

\[ \frac{\Delta x}{\Delta t} = v \rightarrow \Delta x = v \Delta t \rightarrow x = \sum \Delta x = \sum v \Delta t \]

These can be thought of as summing areas under curves
Integrating the Physics Equations if Acceleration is Constant

\[ \Delta v = a \Delta t = \text{area} \]

What if Acceleration is not Constant?

\[ \Delta v = a \Delta t = \text{area} \]
What is \( v(t+\Delta t) \)?

The velocity is off by this much!

Accumulated \( v \) so far

\[ v(t+\Delta t) = v(t) + \Delta v = v(t) + a\Delta t \]

This is close, but is clearly not exactly right!

The problem is that we are treating all of the quantities as if they always have the value that they had at the start of the Time Step, even though they don't.

This is known as a **First Order solution**.
What does a First Order Solution look like in a Program?

You need a way to hold the entire state of the system. This will be the input to our numerical integrator.

You also need a way to return the derivatives once you determine them.

```c
struct state State;
{
float time;
float x;
float vx;
};
struct derivatives Derivatives;
{
float vx;
float ax;
};
```

The inputs are the state, which consists of all variables necessary to completely describe the state of the physical system. The outputs are the derivatives of each state variable.

```c
void GetDerivs( State, Derivatives );
{
// ... 
}
```

void AdvanceOneTimeStep( )
{
GetDerivs( State, Derivatives ); // get derivatives
State.x = State.x + Derivatives.vx * Δt; // use derivatives
State.vx = State.vx + Derivatives.ax * Δt; // use derivatives
State.t = State.t + Δt;
}

The application, then, consists of:

```
Initialize( );
AdvanceOneTimeStep( );
Finish( );
```
What is $v(t+\Delta t)$?

A Second Order solution is obtained by doing the First Order solution, determining all quantities at $time = t + \Delta t$ then averaging them with the quantities at $time = t$ and then treating them as constant throughout the interval.

$$a(t) = \frac{F(t)}{m}$$

$$a(t + \Delta t) = \frac{F(t + \Delta t)}{m}$$

$$\Delta v = a_{avg}\Delta t = \frac{a(t) + a(t + \Delta t)}{2}\Delta t$$

$$v(t + \Delta t) = v(t) + \Delta v$$

What does a Second Order Solution look like in a Program?

```c
void AdvanceOneTimeStep( )
{
    GetDerivs( State, Derivatives1);
    State2.t = State.t + \Delta t;
    State2.x = State.x + Derivatives1.vx * \Delta t;
    State2.vx = State.vx + Derivatives1.ax * \Delta t;
    GetDerivs( State2, Derivatives2 );
    float aavg = ( Derivatives1.ax + Derivatives2.ax ) / 2.;
    float vavg = ( Derivatives1.vx + Derivatives2.vx ) / 2.;
    State.x = State.x + vavg * \Delta t;
    State.vx = State.vx + aavg * \Delta t;
    State.t = State.t + \Delta t;
}
```

The application, then, consists of:

- Initialize( );
- AdvanceOneTimeStep( );
- Finish( );
The Runge-Kutta Fourth Order Solution

\[
\begin{align*}
\{ v_1 \} &= GetDerivs(t, x, v) \\
\{ a_1 \} &= GetDerivs(t + \frac{\Delta t}{2}, x + v_1 \frac{\Delta t}{2}, v + a_1 \frac{\Delta t}{2}) \\
\{ v_2 \} &= GetDerivs(t + \frac{\Delta t}{2}, x + v_2 \frac{\Delta t}{2}, v + a_2 \frac{\Delta t}{2}) \\
\{ a_2 \} &= GetDerivs(t + \Delta t, x + v_2 \Delta t, v + a_2 \Delta t) \\
\{ v_3 \} &= GetDerivs(t + \Delta t, x + v_3 \Delta t, v + a_3 \Delta t) \\
\{ a_3 \} &= GetDerivs(t + \Delta t, x + v_3 \Delta t, v + a_3 \Delta t) \\
\{ v_4 \} &= GetDerivs(t + \Delta t, x + v_4 \Delta t, v + a_4 \Delta t) \\
\{ a_4 \} &= GetDerivs(t + \Delta t, x + v_4 \Delta t, v + a_4 \Delta t)
\end{align*}
\]

\[
x(t + \Delta t) = x + \frac{\Delta t}{6} \left( v_1 a_1 + 2v_2 a_2 + 2v_3 a_3 + v_4 a_4 \right)
\]

---

void AdvanceOneTimeStep( )
{
    GetDerivs( State, Derivatives1 );
    State2.t = State.t + \Delta t/2.;
    State2.x = State.x + Derivatives1.vx * (\Delta t/2.);
    State2.vx = State.vx + Derivatives1.ax * (\Delta t/2.);
    GetDerivs( State2, Derivatives2 );
    State3.t = State.t + \Delta t/2.;
    State3.x = State.x + Derivatives2.vx * (\Delta t/2.);
    State3.vx = State.vx + Derivatives2.ax * (\Delta t/2.);
    GetDerivs( State3, Derivatives3 );
    State4.t = State.t + \Delta t;
    State4.x = State.x + Derivatives3.vx * \Delta t;
    State4.vx = State.vx + Derivatives3.ax * \Delta t;
    GetDerivs( State4, Derivatives4 );

    State.x = State.x + (\Delta t/6.) * ( Derivatives1.vx + 2.*Derivatives2.vx + 2.*Derivatives3.vx + Derivatives4.vx );
    State.vx = State.vx + (\Delta t/6.) * ( Derivatives1.ax + 2.*Derivatives2.ax + 2.*Derivatives3.ax + Derivatives4.ax );
}
Solving Motion where there is a Spring

\[ F_{spring} = -ky \]

This is known as Hooke’s law

\[ \Delta v = \frac{\sum F}{m} \Delta t = \frac{-W - ky}{m} \Delta t \]

void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = ( -W - K*State.y ) / MASS;
}

Mechanical Schematic Symbol:

Air Resistance Force

\[ F_{drag} = \frac{1}{2} \rho v^2 A C_d \]

Drag Coefficient

Fluid density

Y Velocity

Cross-sectional area

Body Falling

Air Resistance always acts in a direction opposite to the velocity of the object
### Some Drag Coefficients

<table>
<thead>
<tr>
<th>$C_d$</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>a smooth brick</td>
</tr>
<tr>
<td>0.9</td>
<td>a typical bicycle plus cyclist</td>
</tr>
<tr>
<td>0.4</td>
<td>rough sphere</td>
</tr>
<tr>
<td>0.1</td>
<td>smooth sphere</td>
</tr>
<tr>
<td>0.001</td>
<td>laminar flat plate</td>
</tr>
<tr>
<td>0.005</td>
<td>turbulent flat plate</td>
</tr>
<tr>
<td>0.295</td>
<td>bullet</td>
</tr>
<tr>
<td>1.0-1.3</td>
<td>person (upright position)</td>
</tr>
<tr>
<td>1.28</td>
<td>flat plate perpendicular to flow</td>
</tr>
<tr>
<td>1.0-1.1</td>
<td>skier</td>
</tr>
<tr>
<td>1.0-1.3</td>
<td>wires and cables</td>
</tr>
<tr>
<td>1.3-1.5</td>
<td>Empire State Building</td>
</tr>
<tr>
<td>1.8-2.0</td>
<td>Eiffel Tower</td>
</tr>
</tbody>
</table>

http://en.wikipedia.org/wiki/Drag_coefficient

### Solving Motion where there is Air Resistance

\[
\Delta v = \sum_{m} F_{x} \Delta t = \frac{-W - \text{Sign}(v_{x}) \frac{1}{2} \rho v_{x}^2 A C_d}{m \Delta t}
\]

\[
\rho_{air} = 1.293 \frac{kg}{m^3}
\]

The $\text{Sign()}$ function returns +1. or -1., depending on the sign of argument

```c
void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = (-W - 0.5*\text{Sign(State.vy)}*DENSITY* State.vy * State.vy * AREA*DRAG ) / MASS;
}
```
Terminal Velocity

When a body is in free fall, it is being accelerated by the force of gravity. However, as it accelerates, it is encountering more and more air resistance force. At some velocity, these two forces balance each other out and the velocity becomes constant, that is, $\Delta v = 0$. This is known as the terminal velocity.

The velocity becomes constant when $\Delta v = 0$:

$$W - \frac{1}{2} \rho v^2 AC_d = 0$$

$$v_t = \sqrt{\frac{2W}{\rho AC_d}}$$

Human Terminal Velocity

Assume:

- Weight = 200 pounds = 890 Newtons
- $C_d = 1.28$
- $A = 6 \text{ ft}^2 = 0.558 \text{ m}^2$
- $\rho_{\text{air}} = 1.293 \frac{\text{kg}}{\text{m}^3}$

$$v_t = \sqrt{\frac{2W}{\rho AC_d}} = 43.90 \frac{m}{\text{sec}} \approx 98 \text{ mph}$$
How about a Cliff Jumper on a Bungee Cord?

\[ F_{\text{spring}} = -ky \]

\[ F_{\text{drag}} = -\text{Sign}(v_y) \frac{1}{2} \rho v_y^2 C_d A \]

\[ \Delta v = \sum \frac{F}{m} \Delta t = \frac{-W - ky - \text{Sign}(v_y) \frac{1}{2} \rho v_y^2 A C_d}{m} \Delta t \]

void GetDerivs( State, Derivatives )
{
    Derivatives.vy = State.vy;
    Derivatives.ay = (-W - K*State.y -.5*Sign(State.vy)*DENSITY*State.vy*STATE*AREA*DRAG) / MASS;
}

Coulomb Damping

This is very much like drag force, but it is the resistance of a fluid being squeezed through a small opening. The resisting force is proportional to the velocity:

\[ F_{\text{damping}} = -cv \]
Lift – Another Good Force to Know About

\[ F_{	ext{lift}} = \frac{1}{2} \rho v^2 A C_L \]

Coefficient of Lift, for a given angle of attack

Air density  Airspeed  Platform area

http://en.wikipedia.org/wiki/Lift_%28force%29

Coefficient of Lift vs. Angle of Attack

http://en.wikipedia.org/wiki/Lift_coefficient
Lift and Drag Dramatically Working Together – Flight of a Frisbee

Friction Force – Another Good Force to Know About

\[ F_{friction} = \mu N \]

Normal force (i.e., amount of force that is perpendicular to the surface)

Coefficient of Friction
Some Coefficients of Friction

<table>
<thead>
<tr>
<th>Materials</th>
<th>μ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dry &amp; clean</td>
</tr>
<tr>
<td>Aluminum Steel</td>
<td>0.61</td>
</tr>
<tr>
<td>Copper Steel</td>
<td>0.53</td>
</tr>
<tr>
<td>Brass Steel</td>
<td>0.51</td>
</tr>
<tr>
<td>Cast iron Copper</td>
<td>1.05</td>
</tr>
<tr>
<td>Cast iron Zinc</td>
<td>0.85</td>
</tr>
<tr>
<td>Concrete (wet)</td>
<td>Rubber</td>
</tr>
<tr>
<td>Concrete (dry)</td>
<td>Rubber</td>
</tr>
<tr>
<td>Concrete Wood</td>
<td>0.82</td>
</tr>
<tr>
<td>Copper Glass</td>
<td>0.68</td>
</tr>
<tr>
<td>Glass Glass</td>
<td>0.94</td>
</tr>
<tr>
<td>Metal Wood</td>
<td>0.2–0.6</td>
</tr>
<tr>
<td>Polythene Steel</td>
<td>0.2</td>
</tr>
<tr>
<td>Steel Steel</td>
<td>0.80</td>
</tr>
<tr>
<td>Steel Teflon</td>
<td>0.04</td>
</tr>
<tr>
<td>Teflon Teflon</td>
<td>0.04</td>
</tr>
<tr>
<td>Wood Wood</td>
<td>0.25–0.5</td>
</tr>
</tbody>
</table>

Buoyancy – Another Good Force to Know About

Archimedes' Principle says that the buoyancy force on an object in a fluid is the weight of the fluid that is being displaced by the object.

\[
\begin{align*}
\rho_{\text{air}} &= 4.66 \times 10^{-5} \text{ pounds / in}^3 \\
\rho_{\text{helium}} &= 0.65 \times 10^{-5} \text{ pounds / in}^3
\end{align*}
\]

Densities

So, for a helium balloon that is one foot in diameter (i.e., radius=6 inches), it has its weight pulling it down and a buoyancy force pushing it up. The net force pushing it up because of the gas inside the balloon is:

\[
F_{\text{buoyancy}} = V_{\text{balloon}} \rho_{\text{helium}} - V_{\text{balloon}} \rho_{\text{air}} = V_{\text{balloon}} (\rho_{\text{helium}} - \rho_{\text{air}})
\]

\[
V_{\text{balloon}} = \frac{4}{3} \pi r^3 = 904.78 \text{in}^3
\]

\[
F_{\text{buoyancy}} = 904.78 \text{in}^3 (4.01 \times 10^{-5} \text{ pounds / in}^3) = 0.036 \text{ pounds}
\]

Note that this must still counterbalance the weight of the balloon material, or the balloon will not fly.
Spinning Motion: Dynamics

Moment of Inertia = an angular “mass” (newton-meters-sec^2/kg-meters^2)
Torque = an angular “force” (newton-meters)

\[ T = I \alpha \]

\[ \alpha = \frac{\Delta \omega}{\Delta t} = \frac{T}{I} \]

\[ \Delta \omega = \alpha \Delta t = \frac{T}{I} \Delta t \]

What does this look like in a Program?

```c
void GetDerivs( State, Derivatives )
{
    Derivatives.vx = State.vx;
    Derivatives.x = SomeOfAllForces / MASS;
    Derivatives.omega = State.omega;
    Derivatives.alpha = SomeOfAllTorques / INERTIA
}
```

The state and derivative vectors now include angular components.