# Dynamic Physics for Simulation and Game Programming

Oregon State University

**Mike Bailey**

**mjb@cs.oregonstate.edu**


Think Geek



Oregon State University
Computer Graphics

# Discrete Dynamics

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

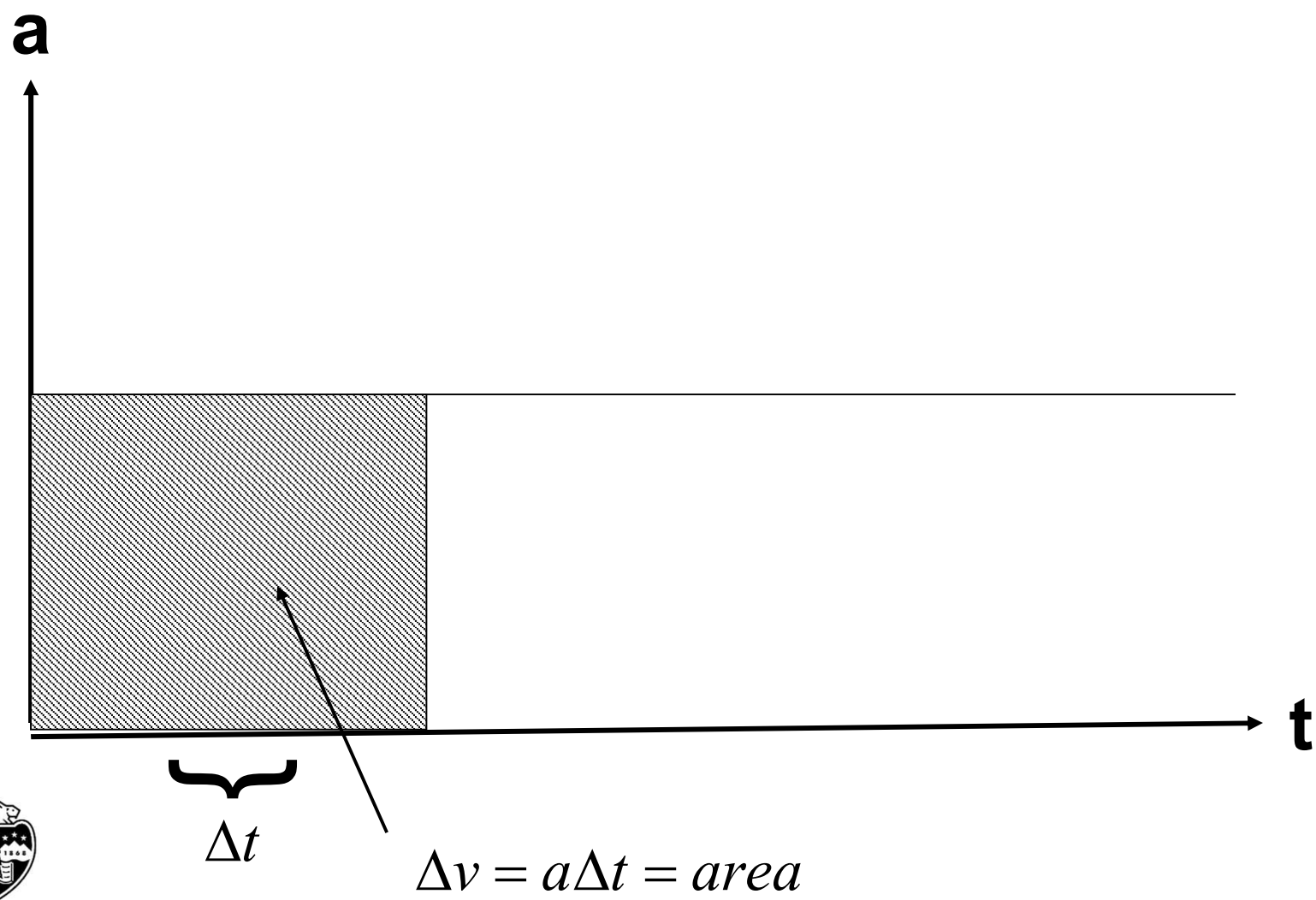
$$a = \frac{F}{m}$$

$$\frac{\Delta v}{\Delta t} = a \quad \Rightarrow \quad \Delta v = a\Delta t \quad \Rightarrow \quad v = \sum \Delta v = \sum a\Delta t$$

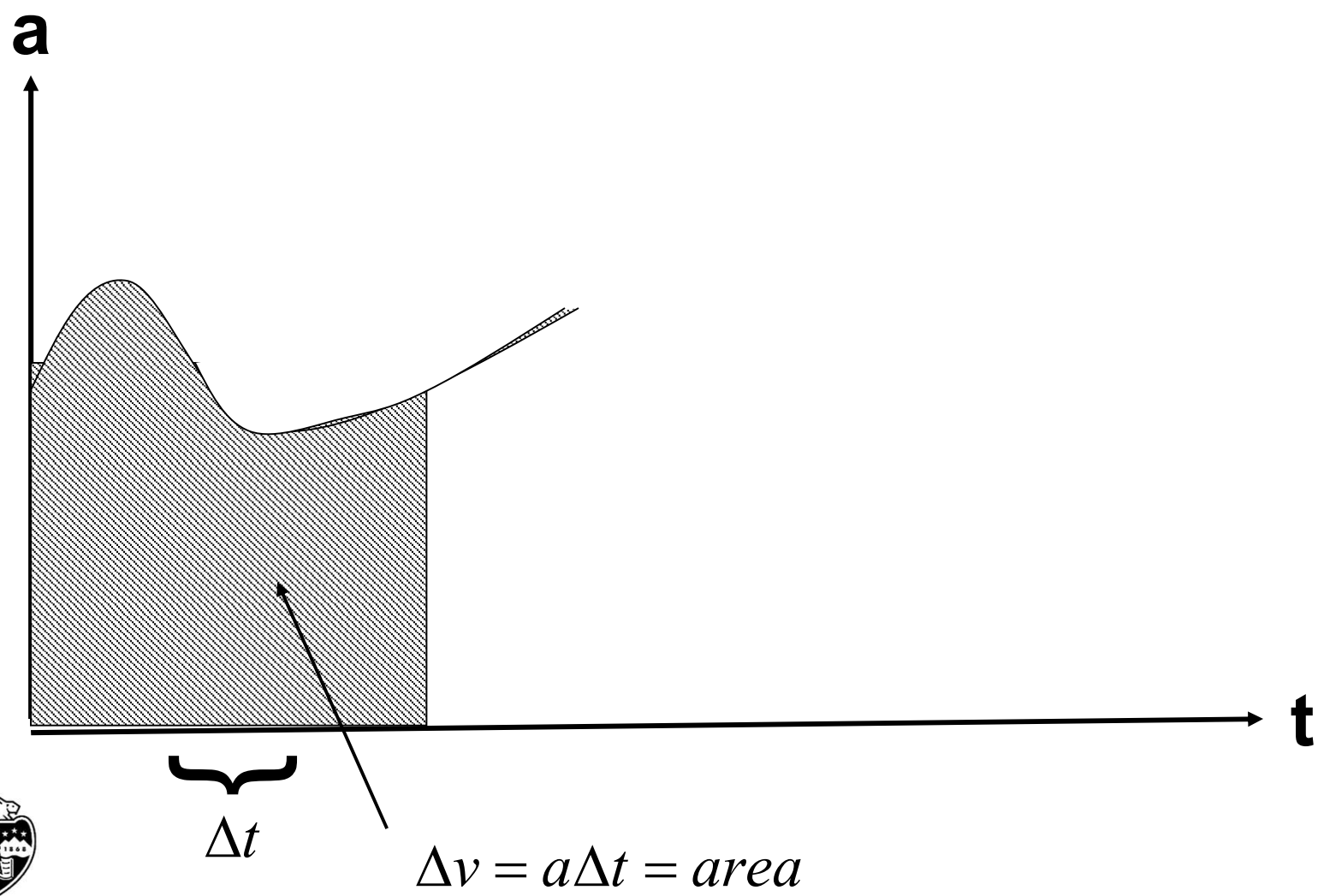
$$\frac{\Delta x}{\Delta t} = v \quad \Rightarrow \quad \Delta x = v\Delta t \quad \Rightarrow \quad x = \sum \Delta x = \sum v\Delta t$$

These can be thought of as summing areas under curves

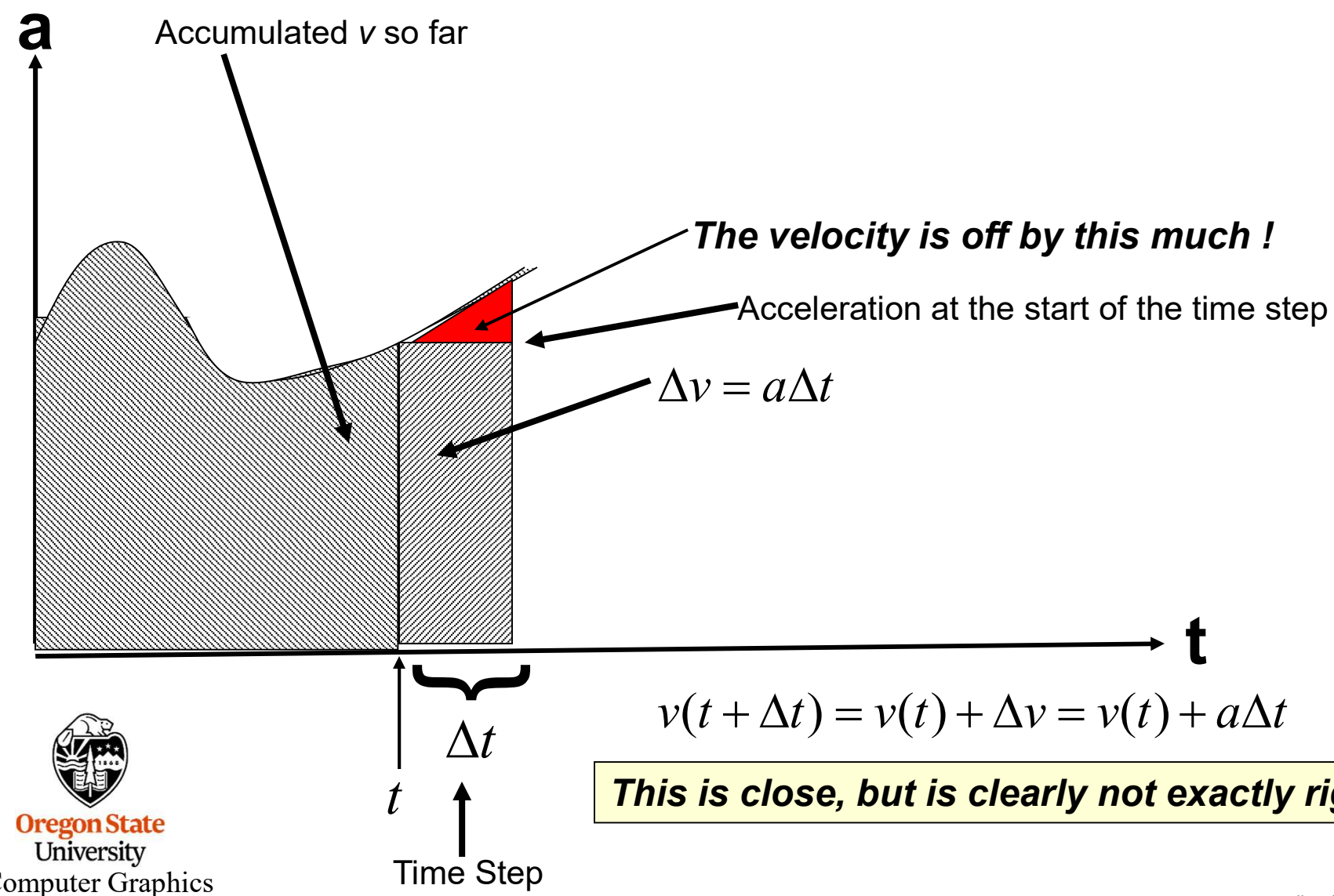
# Integrating the Physics Equations if Acceleration is *Constant*



# What if Acceleration is not Constant?



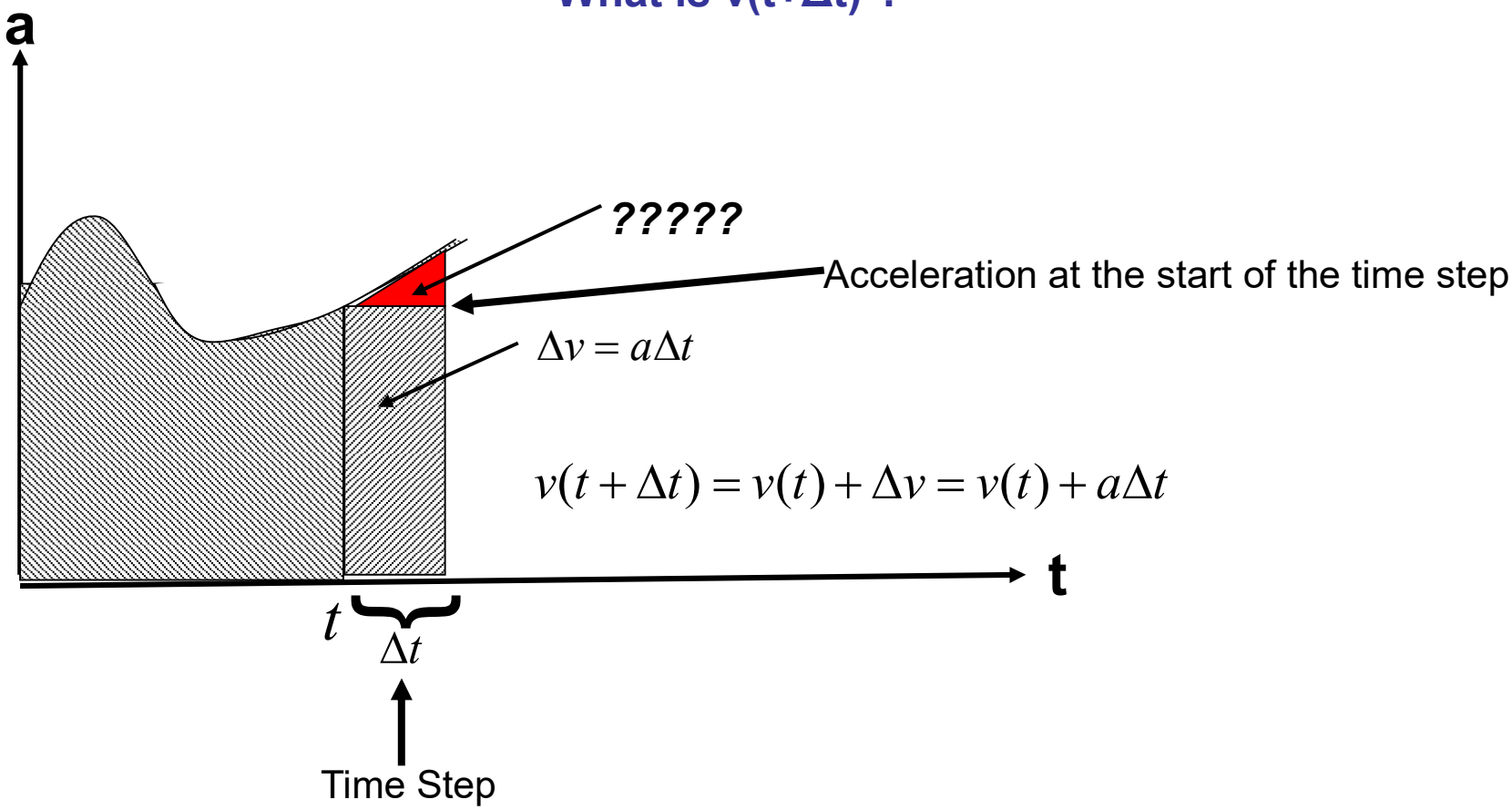
# What is v(t+Δt) ?




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

***This is close, but is clearly not exactly right!***

## What is v(t+Δt) ?




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.

```
struct state
{
        float time;
        float x;
        float vx;
} ;
```

$$\frac{dx}{dt} \Rightarrow v_x$$

$$\frac{dv_x}{dt} \Rightarrow a_x$$

```
struct derivatives
{
        float vx;
        float ax;
} ;
```

```
struct state state;
```

```
struct derivatives derivs;
```

```
void
GetDerivs( state, &derivs )
{
        . . .
}
```

The outputs are the derivatives of the state variables.

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.

# What does a First Order Solution look like in a Program?

```
void
AdvanceOneTimeStep( )
{
        GetDerivs( state, &derivs );            // get derivatives

        state.x  = state.x  + derivs->vx * Δt;  // use derivatives
        state.vx = state.vx + derivs->ax * Δt;  // use derivatives
        state.t  = state.t  + Δt ;
}
```

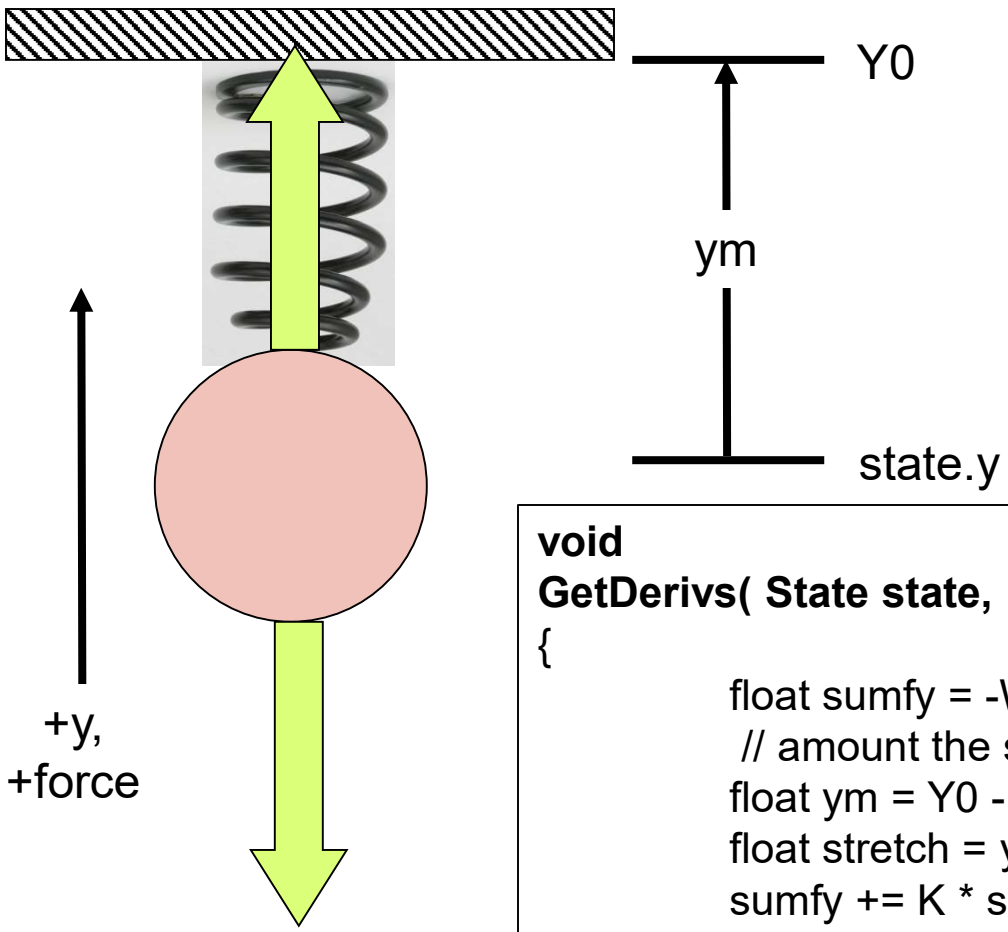
**The application, then, consists of:**

Initialize( );

AdvanceOneTimeStep( );

Finish( );

# What does a GetDerivs function look like in a Program?



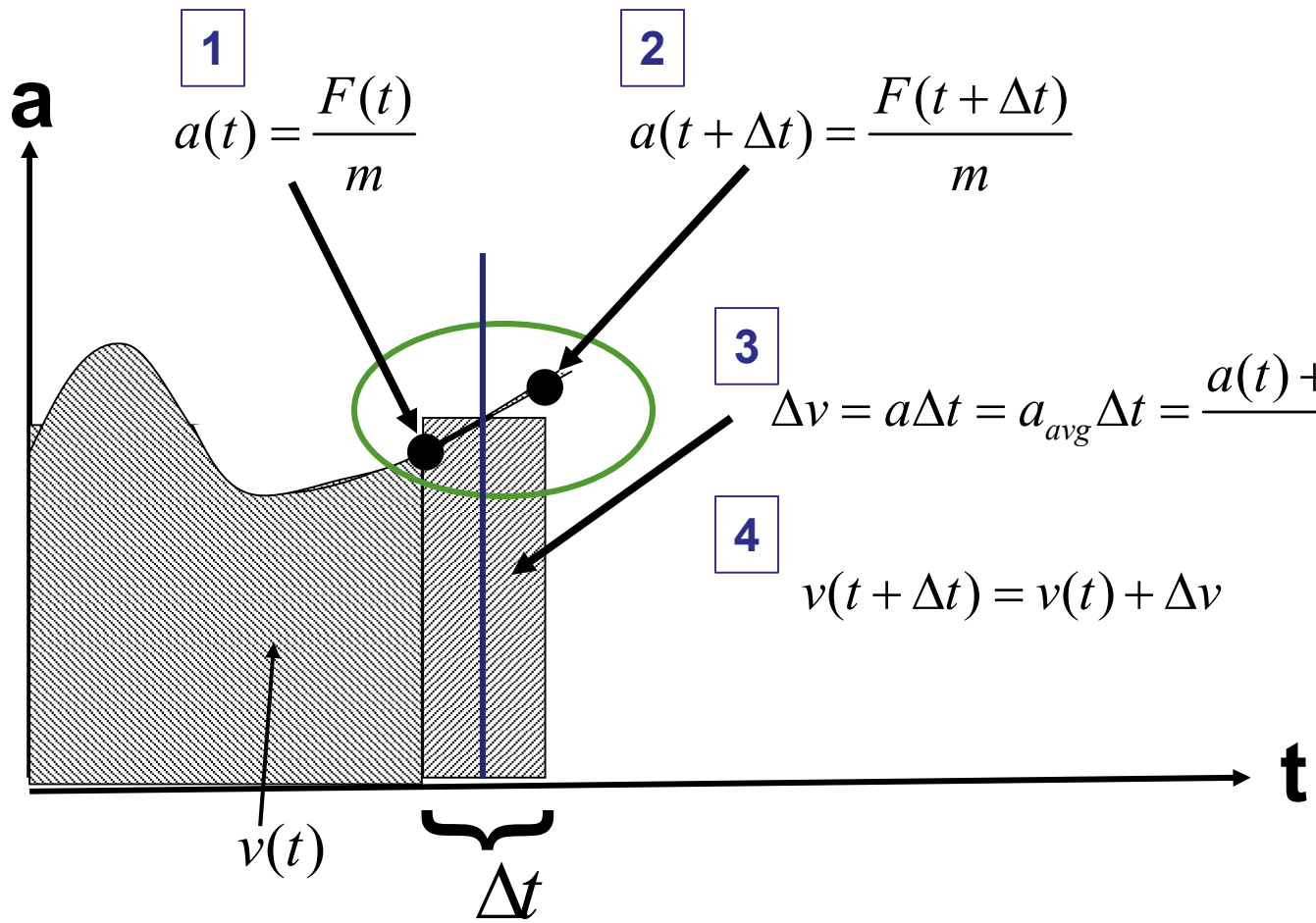

```
void
GetDerivs( State state, Derivatives *derivs )
{
        float sumfy = -Weight;
         // amount the spring is stretched:
        float ym = Y0 - state.y;
        float stretch = ym - LENGTH0;
        sumfy += K * stretch;
        sumfy -= Cd * state.vy;          // the damping

        derivs->vy = state.vy;
        derivs->ay = sumfy / Mass;
}
```

Oregon State University
Computer Graphics

## What is v(t+Δt) ?

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.




**1**

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

**2**

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

**3**

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

**4**

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

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

```
void
AdvanceOneTimeStep( )
{
        GetDerivs( State, &Derivatives1);
        State2.t    = State.t   + Δt;
        State2.x    = State.x   + Derivatives1->vx * Δt;
        State2.vx = State.vx + Derivatives1->ax * Δt;

        GetDerivs( State2, &Derivatives2 );
        float aavg = ( Derivatives1->ax + Derivatives2->ax) / 2.;
        float vavg = ( Derivatives1->vx + Derivatives2->vx) / 2.;

        State.x    = State.x   + vavg * Δt;
        State.vx = State.vx + aavg * Δt;
        State.t    = State.t   + Δt ;
}
```

The application, then, consists of:

Initialize( );

AdvanceOneTimeStep( );

Finish( );

# The Runge-Kutta Fourth Order Solution

$$\begin{Bmatrix} v_1 \\ a_1 \end{Bmatrix} = GetDerivs(t, x, v)$$

$$\begin{Bmatrix} v_2 \\ a_2 \end{Bmatrix} = GetDerivs(t + \frac{\Delta t}{2}, x + v_1 \frac{\Delta t}{2}, v + a_1 \frac{\Delta t}{2})$$

$$\begin{Bmatrix} v_3 \\ a_3 \end{Bmatrix} = GetDerivs(t + \frac{\Delta t}{2}, x + v_2 \frac{\Delta t}{2}, v + a_2 \frac{\Delta t}{2})$$

$$\begin{Bmatrix} v_4 \\ a_4 \end{Bmatrix} = GetDerivs(t + \Delta t, x + v_3 \Delta t, v + a_3 \Delta t)$$

$$\boxed{\begin{Bmatrix} x(t+\Delta t) \\ v(t+\Delta t) \end{Bmatrix} = \begin{Bmatrix} x \\ v \end{Bmatrix} + \frac{\Delta t}{6}\left(\begin{Bmatrix} v_1 \\ a_1 \end{Bmatrix} + 2\begin{Bmatrix} v_2 \\ a_2 \end{Bmatrix} + 2\begin{Bmatrix} v_3 \\ a_3 \end{Bmatrix} + \begin{Bmatrix} v_4 \\ a_4 \end{Bmatrix}\right)}$$



Adapted from: http://en.wikipedia.org/wiki/Runge-Kutta

# The Runge-Kutta Fourth Order Solution

```
void
AdvanceOneTimeStep( )
{
        GetDerivs( State, &Derivatives1 );
        State2.t    = State.t   + Δt/2.;
        State2.x    = State.x  +  Derivatives1->vx * (Δt/2.);
        State2.vx = State.vx + Derivatives1->ax * (Δt/2.);

        GetDerivs( State2, &Derivatives2 );
        State3.t    = State.t   + Δt/2.;
        State3.x    = State.x  +  Derivatives2->vx * (Δt/2);
        State3.vx = State.vx + Derivatives2->ax * (Δt/2.);

        GetDerivs( State3, &Derivatives3 );
        State4.t    = State.t   + Δt;
        State4.x    = State.x  +  Derivatives3->vx * Δt;
        State4.vx = State.vx + Derivatives3->ax * Δt;

        GetDerivs( State4, &Derivatives4 );

    State.x   = State.x  +  (Δt/6.) * ( Derivatives1->vx + 2.*Derivatives2->vx + 2.*Derivatives3->vx + Derivatives4->vx );
    State.vx = State.vx +  (Δt/6.) * (Derivatives1->ax + 2.*Derivatives2->ax + 2.*Derivatives3->ax + Derivatives4->ax );
}
```

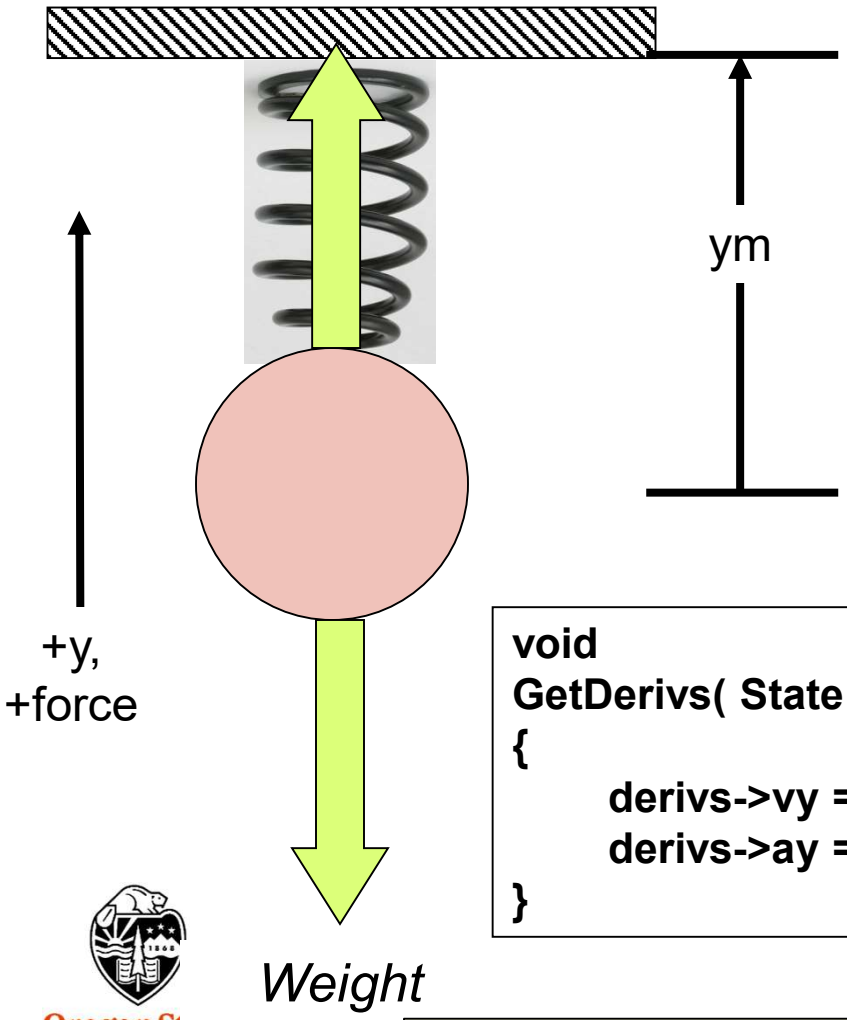


Adapted from: http://en.wikipedia.org/wiki/Runge-Kutta

# Solving Motion where there is a Spring

$$F_{spring} = -ky$$

This is known as Hooke's law




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

Mechanical Schematic Symbol for a spring:

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

For clarity, I've left out the "**- LENGTH0**", but it should be there

# Air Resistance Force

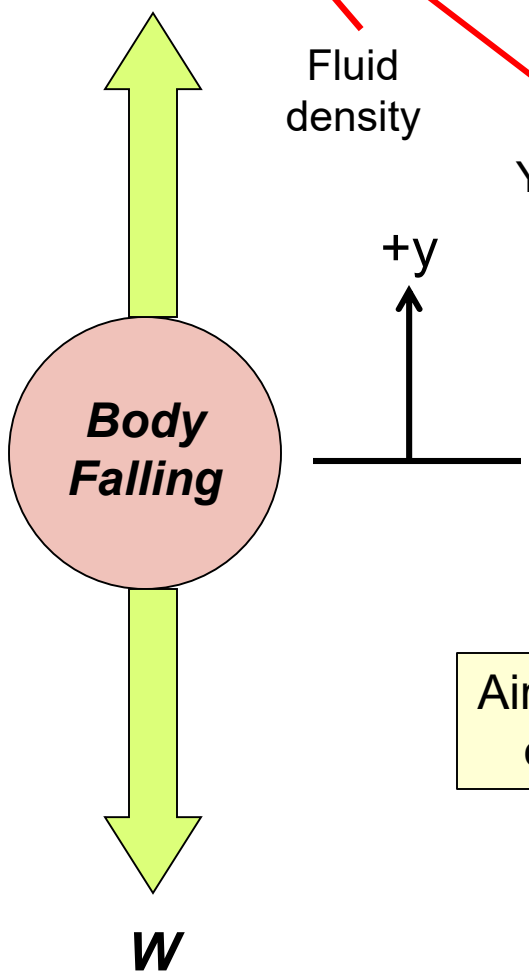
$$F_{drag} = \frac{1}{2}\rho v_y^2 A C_d$$

Drag Coefficient

Fluid density

Y Velocity

Cross-sectional area




Air Resistance always acts in a direction opposite to the velocity of the object

## Some Drag Coefficients

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

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

# Solving Motion where there is Air Resistance

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

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

The **Sign( )** function returns +1. or -1., depending on the sign of argument

```
void
GetDerivs( State state, Derivatives *derivs )
{
        derivs->vy = state.vy;
        derivs->ay = (–W – .5*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, Δv=0.

This is known as the **terminal velocity**.

$$\Delta v = \frac{-W + \frac{1}{2}\rho v_y^2 A C_d}{m}\Delta t$$

The velocity becomes constant when Δv = 0 :

$$W - \frac{1}{2}\rho v_y^2 A C_d = 0$$

$$v_t = \sqrt{\frac{2W}{\rho A C_d}}$$

# Human Terminal Velocity

**Assume:**

Weight = 200 pounds = 890 Newtons

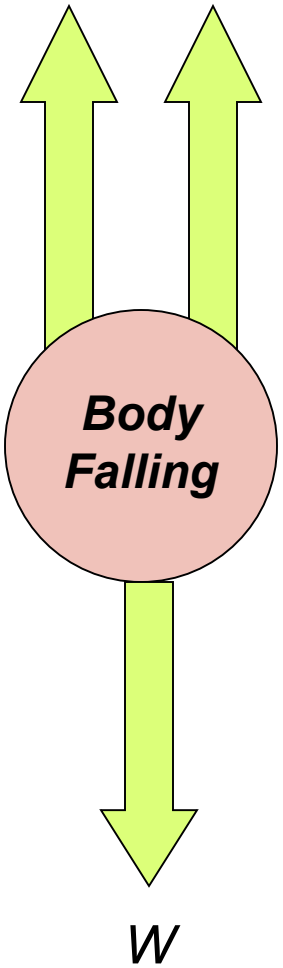
$C_d$ = 1.28

A = 6 $ft^2$ = 0.558 $m^2$

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

$$v_t = \sqrt{\frac{2W}{\rho A C_d}} = 43.90 \frac{m}{\sec} \approx 98 mph$$

# How about a Cliff Jumper on a Bungee Cord?




$$F_{spring} = -ky$$

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

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

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

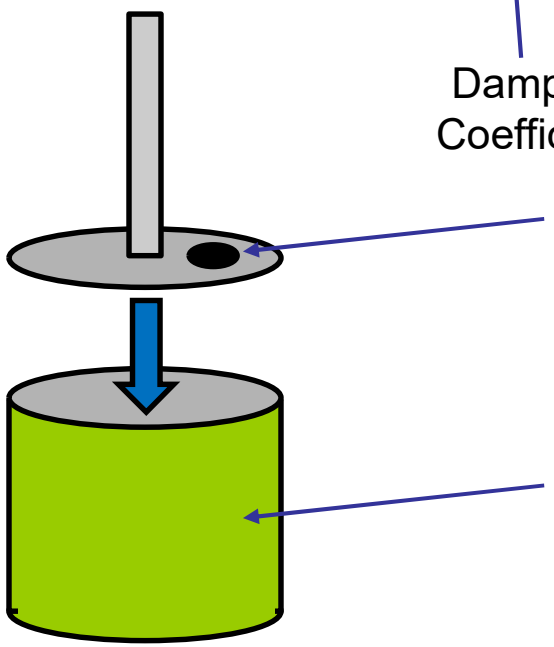
# Coulomb Damping

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:

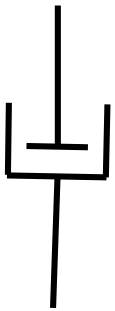
$$F_{damping} = -cv$$

Velocity

Damping Coefficient




**Mechanical Schematic Symbol:**

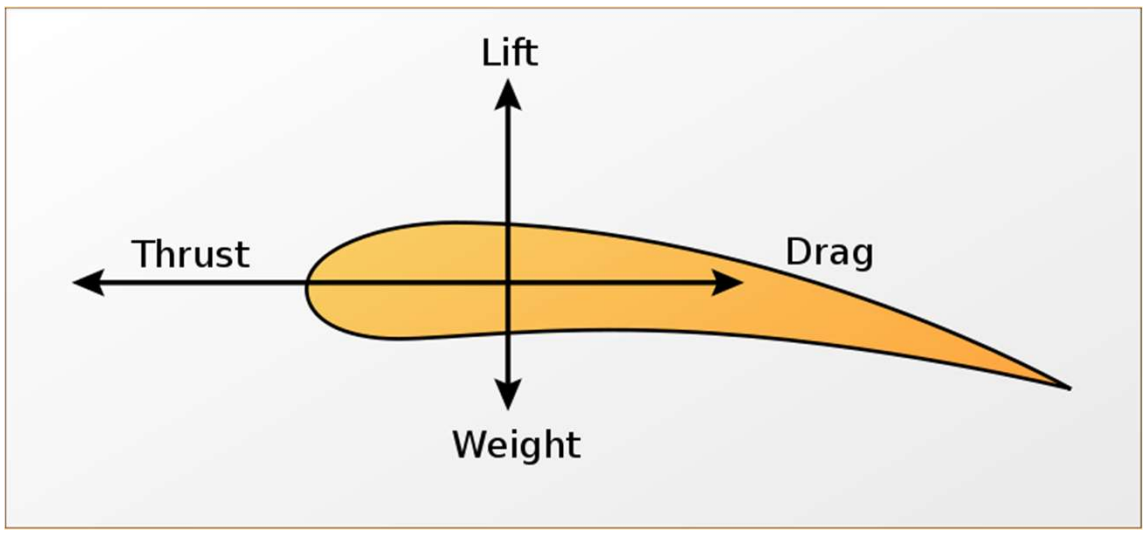


# Lift – Another Good Force to Know About

$$F_{lift} = \frac{1}{2}\rho v^2 A C_L$$

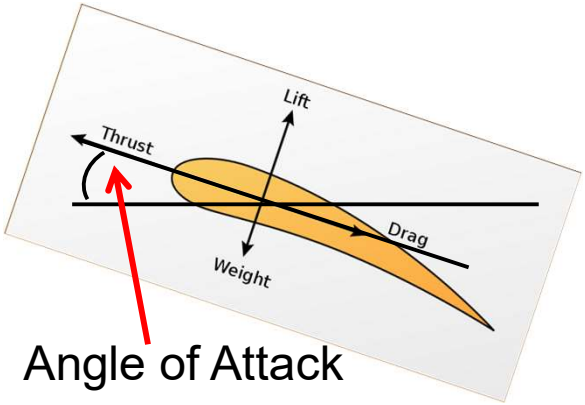
Coefficient of Lift, for a given angle of attack

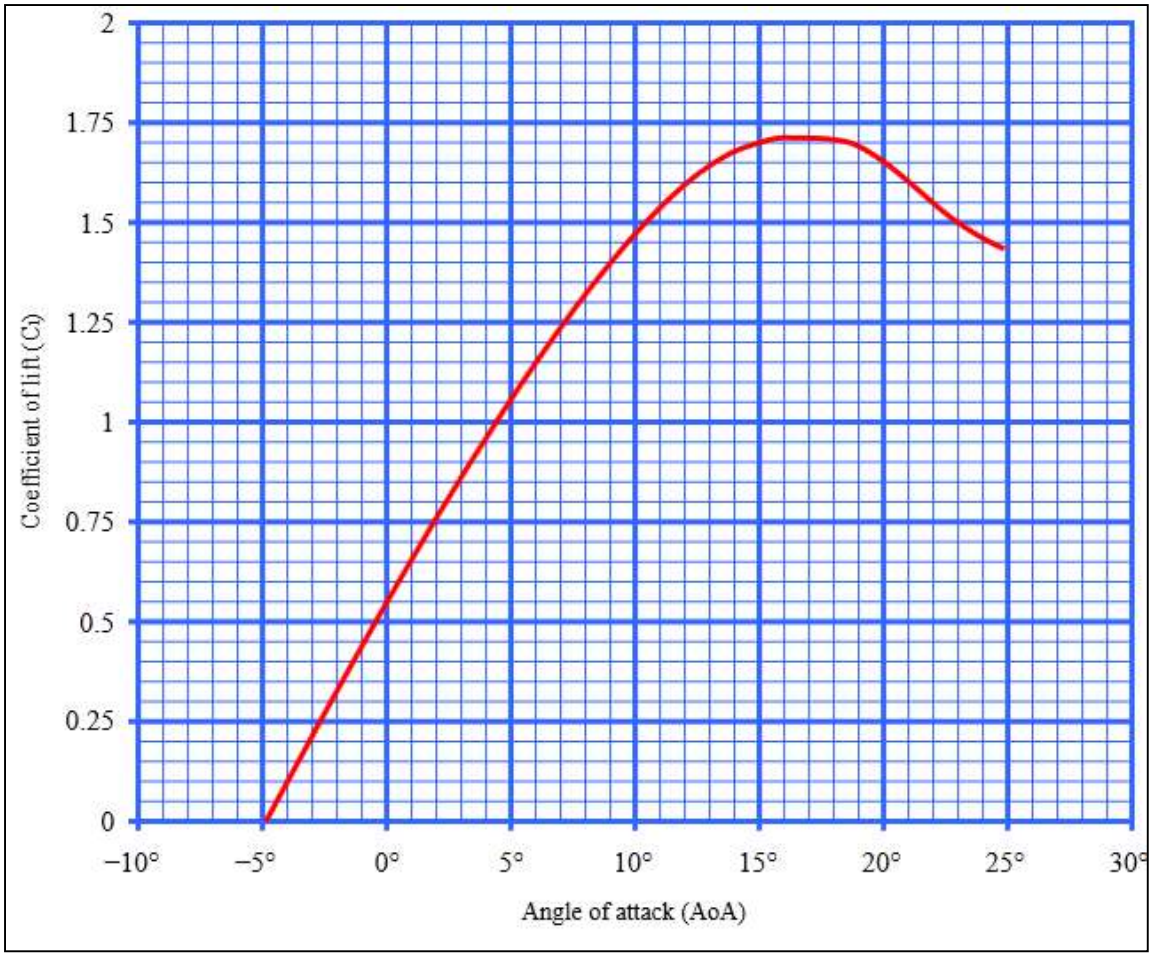
Air density  Airspeed  Platform area




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

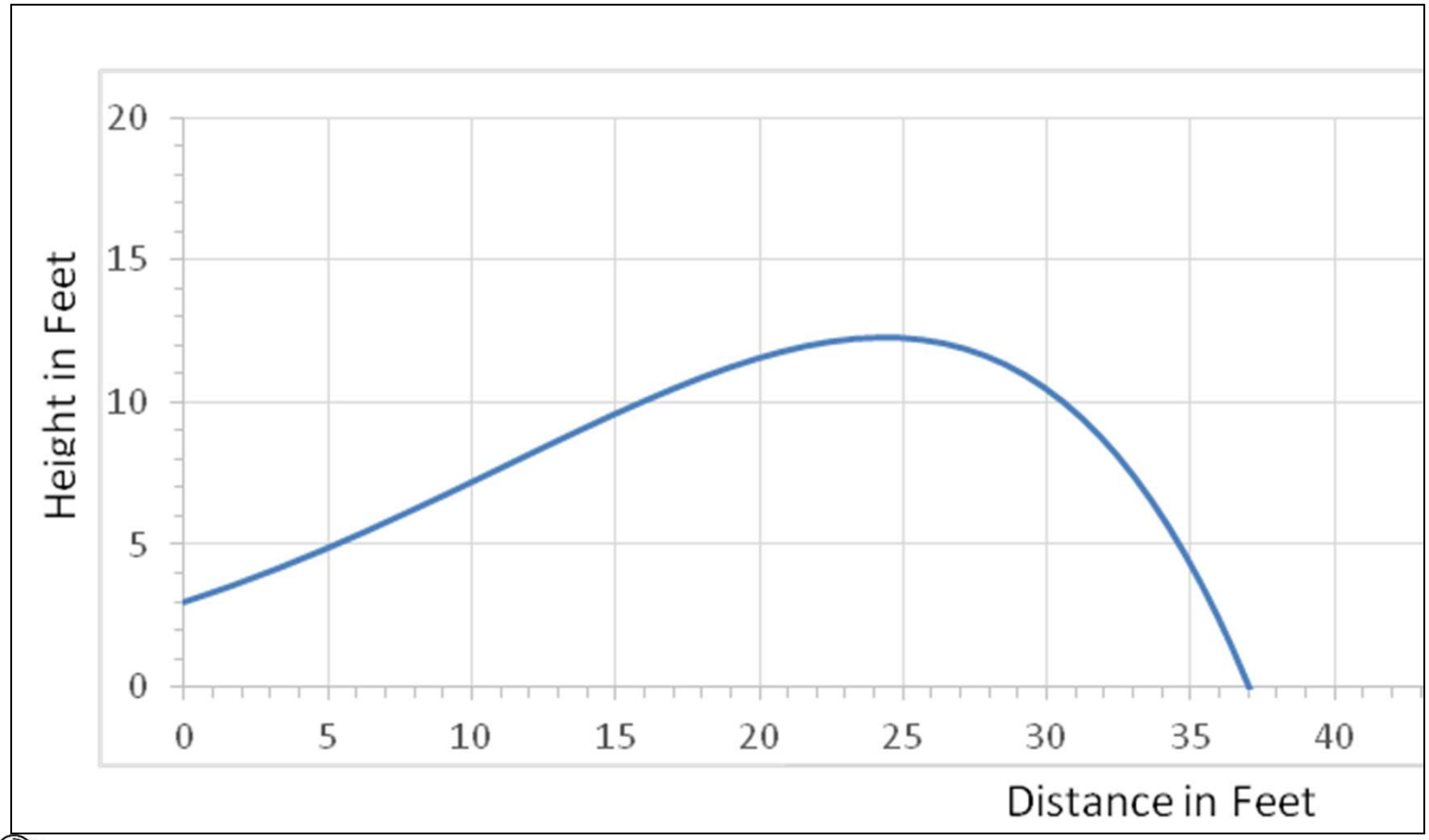
# Coefficient of Lift vs. Angle of Attack




**http://en.wikipedia.org/wiki/Lift_coefficient**

# Lift and Drag Dramatically Working Together – Flight of a Frisbee

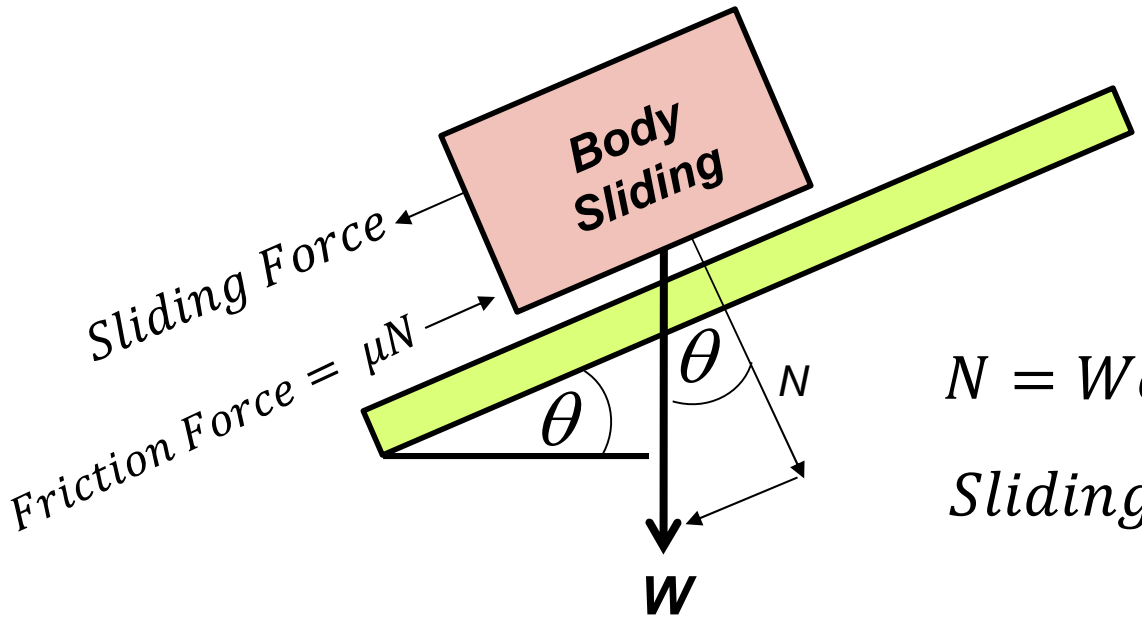


# Friction Force – Another Good Force to Know About

$$F_{friction} = \mu N$$

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

Coefficient of Friction




$$N = W cos\theta$$

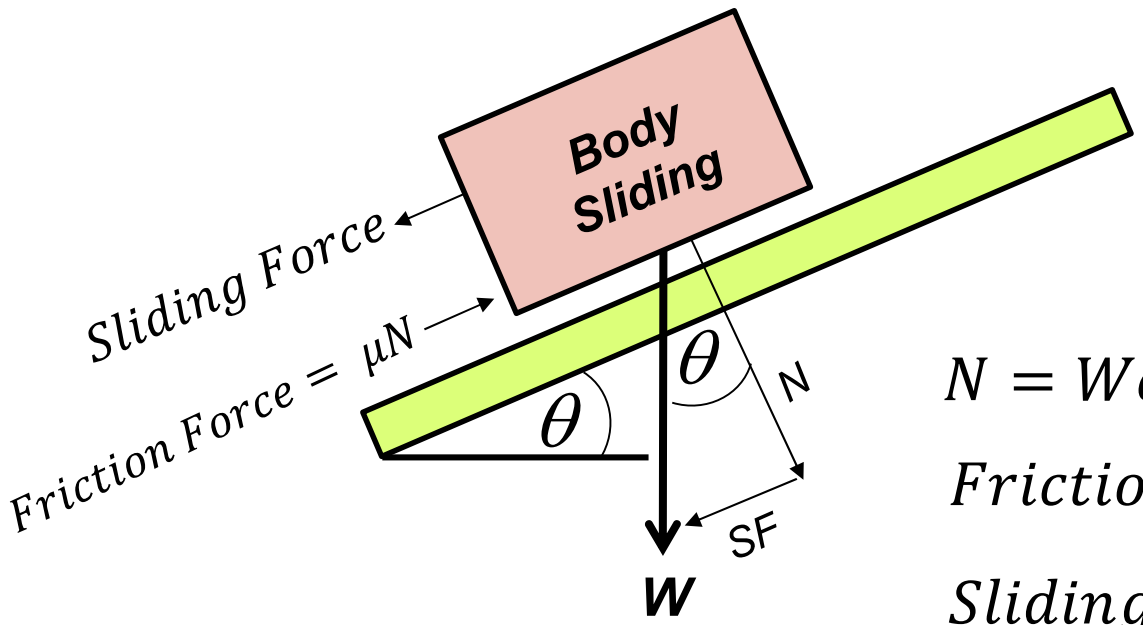
$$Sliding\ Force = W sin\theta$$

# Some Coefficients of Friction

| Materials | | μ | |
|---|---|---|---|
| | | Dry & clean | Lubricated |
| Aluminum | Steel | 0.61 | |
| Copper | Steel | 0.53 | |
| Brass | Steel | 0.51 | |
| Cast iron | Copper | 1.05 | |
| Cast iron | Zinc | 0.85 | |
| Concrete (wet) | Rubber | 0.30 | |
| Concrete (dry) | Rubber | 1.0 | |
| Concrete | Wood | 0.62 | |
| Copper | Glass | 0.68 | |
| Glass | Glass | 0.94 | |
| Metal | Wood | 0.2–0.6 | 0.2 (wet) |
| Polythene | Steel | 0.2 | 0.2 |
| Steel | Steel | 0.80 | 0.16 |
| Steel | Teflon | 0.04 | 0.04 |
| Teflon | Teflon | 0.04 | 0.04 |
| Wood | Wood | 0.25–0.5 | 0.2 (wet) |

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

# Determining the Coefficient of Friction Experimentally




$$N = W\cos\theta$$

$$Friction\ Force = \mu W\cos\theta$$

$$Sliding\ Force = W\sin\theta$$

At what angle, Θ, does the block *just* begin to slide?
It begins to slide when the sliding force equals the friction force:

$$W\sin\theta = \mu W\cos\theta$$

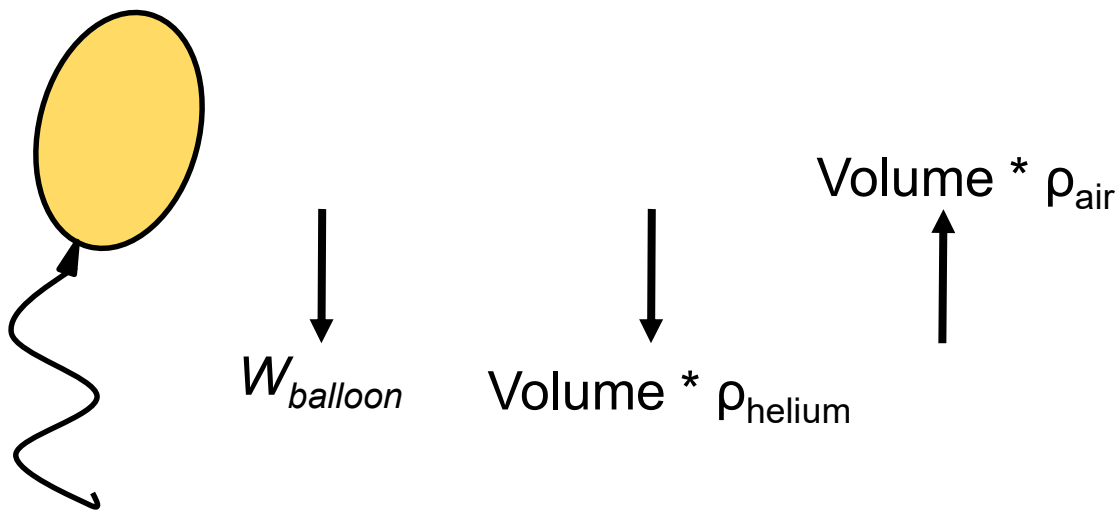
$$\mu = \frac{\sin\theta}{\cos\theta} = \tan\theta$$

Thus, if you raise the board just enough that the block starts to slide, the coefficient of friction is the tangent of that angle.

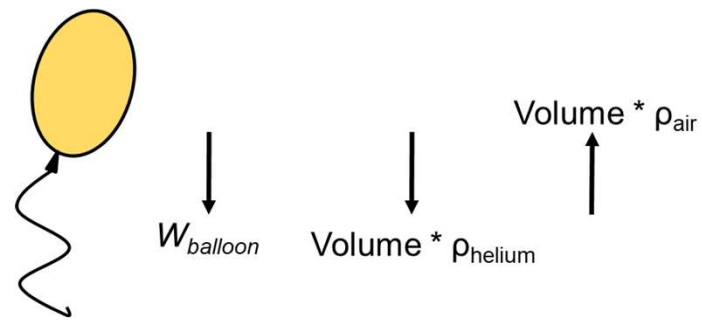
# Buoyancy – Another Good Force to Know About

***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.

So, for a helium balloon, it has its weight pulling it down, the weight of the helium pulling it down, and a buoyancy force pushing it up.



# Buoyancy – Another Good Force to Know About




$$\left.\begin{aligned}\rho_{air} &= 4.66x10^{-5}\ pounds/in^3 \\ \rho_{helium} &= 0.65x10^{-5}\ pounds/in^3\end{aligned}\right\}\ \text{Densities}$$

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:

$$F_{buoyancy} = V_{balloon}\rho_{helium} - V_{balloon}\rho_{air} = V_{balloon}(\rho_{helium} - \rho_{air})$$

$$V_{balloon} = \frac{4}{3}\pi r^3 = 904.78in^3$$

$$F_{buoyancytotal} = 904.78in^3(4.01x10^{-5}pounds/in^3) = 0.036pounds$$

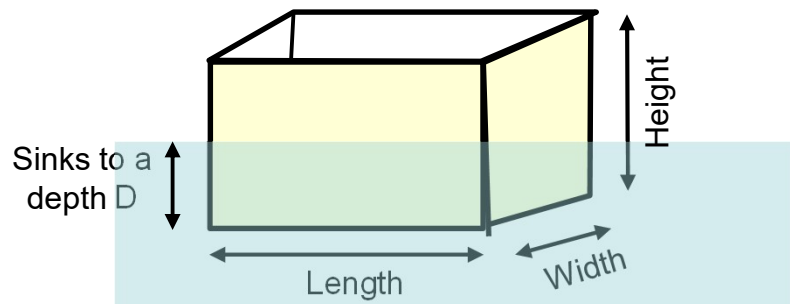
Note that we would still need to take the balloon's skin weight into account

## Buoyancy – Another Good Force to Know About

You have a waterproof box that you lay on the surface of the water. You put a 100-pound dog in the box. How deep will the box sink into the water before it starts to float?

(Let's assume the weight of the box and the weight of the air in the box are negligible.)

$$\rho_{water} = 0.03635 \ \ pounds/in^3$$




The water displaced is Length*Width*D, so the upwards force is **0.03635*Length*Width*D** which needs to counterbalance **100 pounds**. So, we find that the box sinks to a depth of:

$$D = \frac{2751}{Length * Width}$$

If *D* ends up exceeding the Height of the box, well, I sure hope the dog can swim.

# Spinning Motion: Dynamics

$$T = I\alpha$$

Moment of Inertia ≈ an angular "mass"
(newton-meters-sec$^2$=kg-meters$^3$)

Torque ≈ an angular "force"
(newton-meters)

$$\alpha = \frac{\Delta\omega}{\Delta t} = \frac{T}{I}$$

$$\Delta\omega = \alpha\Delta t = \frac{T}{I}\Delta t$$

# Spinning Motion:
# What does this look like in a Program?

```
struct state
{
        float t;
        float x;
        float vx;
        float theta;
        float omega;
} ;
```

```
struct derivatives
{
        float vx;
        float ax;
        float omega;
        float alpha;
} ;
```

The state and derivative vectors now include angular components

```
void
GetDerivs( State state, Derivatives *derivs )
{
        derivs->vx = state.vx;
        derivs-> x   = SomeOfAllForces / MASS;
        derivs->omega  = state.omega;
        derivs->alpha    = SomeOfAllTorques / INERTIA
}
```