

```

Vector3 Gravity_Force() {
    return new Vector3(0,-g,0);
}

// Air resistance opposing motion
Vector3 Air_force(){
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return airDragMag * -vDir;
}

```

## Introduction To Section 3

```

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}

```

## In this section you will learn about...

- About “moments of inertia”.
- The rotational equivalent of  $F = ma$
- Parallel and Intermediate Axis Theorems.
- Calculating rotation from torques.
- Tensors and Einstein notation.

```

Vector3 Gravity_Force() {
    return new Vector3(0,-g,0);
}

// Air resistance opposing motion
Vector3 Air_force(){
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return airDragMag * -vDir;
}

```

## Unstable Rotation In Unity 5

```

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}

```

## In this video...

- The limitations of spinning objects in Unity 5.
- Why unstable rotation doesn't evolve.
- How to model unstable rotation in game physics.

## Spin a phone-shaped cuboid

- Create a cuboid with scale (70, 140, 7).
- Rotate it 2 degrees about y-axis.
- Add a rigid body, ignore gravity, 0 angular drag.
- Give it **angularVelocity = (4f, 0, 0)**
- Observe that the resulting spin is stable.



```
Vector3 Gravity_force() {  
    return new Vector3(0,-9,0);  
}  
  
// Air resistance opposing motion  
Vector3 Air_force(){  
    float vMag = vel.magnitude;  
    Vector3 vDir = vel.normalized;  
    float airDragMag = dragConst * Mathf.Pow(vMag, 2f);  
    return airDragMag * -vDir;  
}
```

## Introducing Inertia Tensors

```
void ResolveForces (Vector3[] forces) {  
    Vector3 combinedForce = Vector3.zero;  
    foreach (Vector3 force in forces) {  
        combinedForce += force;  
    }  
    float deltaTime = Time.fixedDeltaTime;  
    acc = combinedForce / mass;  
    vel += acc * deltaTime;  
    pos += vel * deltaTime;  
    if (isEnabled) transform.position = pos;  
}
```

## In this video...

- Roughly what is an inertia tensor (**I**)?
- Unstable intermediate axis
- How Unity Calculates **I**
- Calculate **I** for your phone
- Predict it's unstable axis

## Unstable Intermediate Axis

- Look at the three components of **I\***
- Does one axis have an *intermediate* value?
- If so, the rotation will be unstable about this axis.

\* Strictly there are 9 components as we'll see later.

## How Unity Calculates I

- Mass is specified by the parent's rigidBody.
- No child may have a rigid body.
- Child objects' colliders distribute mass evenly.
- Local axis must be aligned with symmetry.
- (Inertia tensor calculated about C.O.M.)

## Inertia Tensor Of Your SmartPhone

- Lookup the dimensions and mass.
- Assume it's an exact cuboid (no rounding).
- Write down the value Unity gives you.
- Which axis does it suggest is unstable?



## Calculating Moment Of Inertia

```
Vector3 Gravity_Force() {  
    return new Vector3(0, -9.8f, 0);  
}  
  
// Air resistance: opposing motion  
Vector3 Air_Force(){  
    float vMag = vel.magnitude;  
    Vector3 vDir = vel.normalized;  
    float airDragMag = dragConst * Mathf.Pow(vMag, 2f);  
    return airDragMag * -vDir;  
}
```

```
void ResolveForces (Vector3[] forces) {  
    Vector3 combinedForce = Vector3.zero;  
    foreach (Vector3 force in forces) {  
        combinedForce += force;  
    }  
    float deltaT = Time.fixedDeltaTime;  
    acc = combinedForce / mass;  
    vel += acc * deltaT;  
    pos += vel * deltaT;  
    if (isEnabled) transform.position = pos;  
}
```

## In this video...

- How about if two axis are equal?
- Try it with a spinning cylinder.
- Calculate the moment of inertia.
- Render vs. collider mesh influence.

## Try spinning a cylinder

- Spin a cylinder around each axis.
- Use **SpinRite.cs** to see if it's stable in all 3.

```
Vector3 x = magnusConst * Vector3.Cross(v, angularVel);  
return x;  
  
void ResolveForces (Vector3[] forces) {  
    Vector3 combinedForce = Vector3.zero;  
    foreach (Vector3 force in forces) {  
        combinedForce += force;  
    }  
    float deltaTime = Time.fixedDeltaTime;  
    float combinedForceMag = mass;  
    acc = combinedForce / mass;  
    vel += acc * deltaTime;  
    pos += vel * deltaTime;  
    if (isEnabled) transform.position = pos;  
}
```



## Calculate your cylinder's value

- Calculate the moment about x (or z)\*
- See if it matches Unity's value.
- If not, try and explain why.

\* [http://en.wikipedia.org/wiki/List\\_of\\_moments\\_of\\_inertia](http://en.wikipedia.org/wiki/List_of_moments_of_inertia)



```
Vector3 Gravity_Force() {  
    return new Vector3(0, g, 0);  
}  
  
// Air resistance: opposing motion  
Vector3 Air_Force() {  
    float vMag = vel.magnitude;  
    Vector3 vDir = vel.normalized;  
    float airDragMag = dragConst * Mathf.Pow(vMag, 2f);  
    return airDragMag * -vDir;  
}
```

## Parallel Axis Theorem

```
void ResolveForces (Vector3[] forces) {  
    Vector3 combinedForce = Vector3.zero;  
    foreach (Vector3 force in forces) {  
        combinedForce += force;  
    }  
    float deltaTime = Time.fixedDeltaTime;  
    float combinedForceMag = mass;  
    acc = combinedForce / mass;  
    vel += acc * deltaTime;  
    pos += vel * deltaTime;  
    if (isEnabled) transform.position = pos;  
}
```

## In this video...

- NOT the same as Intermediate Axis Theorem.
- How to calculate I for compound objects.
- Think of a discus thrower.

[http://en.wikipedia.org/wiki/Parallel\\_axis\\_theorem](http://en.wikipedia.org/wiki/Parallel_axis_theorem)

## Work out I for two balls

- Use Unity to calculate I for these two balls.
- One with mass 100 at (0,0,0).
- Another with mass 10 at (1,1,0).
- What's the value of I Unity calculates?
- What's the Centre Of Gravity of the system?



```
Vector3 Gravity_Force() {  
    return new Vector3(0,-9,0);  
}  
  
// Air resistance opposing motion  
Vector3 Air_Force(){  
    float vMag = vel.magnitude;  
    Vector3 vDir = vel.normalized;  
    float airDragMag = dragConst * Mathf.Pow(vMag, 2f);  
    return airDragMag * -vDir;  
}
```

## “Newton’s Laws Of Rotation”

```
void ResolveForces (Vector3[] forces) {  
    Vector3 combinedForce = Vector3.zero;  
    foreach (Vector3 force in forces) {  
        combinedForce += force;  
    }  
    float deltaT = Time.fixedDeltaTime;  
    acc = combinedForce / mass;  
    vel += acc * deltaT;  
    pos += vel * deltaT;  
    if (isEnabled) transform.position = pos;  
}
```

## In this video...

- A recap of Newton’s laws of motion.
- How these relation to rotation.
- Important differences.
- Why we’re jumping in the deep end!

## Draw your own parallels

*Substitute with torque, angular velocity, angular acceleration and inertia tensor...*

1. With no net **force**, **velocity** remains constant.
2. **Force** = **mass** \* **acceleration**
3. Every **force** has an equal and opposite **force**.





## Rotational equivalent of $F = ma$

```

Vector3 Air_force() {
    float vMag = vel.magnitude;
    Vector3 dir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return dir * airDragMag * -vDir;
}

Vector3 Magnus_force() {
    Vector3 x = MagnusConst * Vector3.Cross(vel, angular);
    return x;
}

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}
    
```

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

70kg

$$\underline{\tau} = I \underline{\alpha}$$

$$\begin{bmatrix} 0.3 & 0.1 & 0.0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \text{ kg m}^2$$



```

Vector3 Gravity_force() {
    return new Vector3(0,-9,0);
}

// Air resistance opposing motion
Vector3 Air_force() {
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return airDragMag * -vDir;
}
    
```

## Applying Torque To Rigidbodies

```

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}
    
```

## In This Video...

- How to use Unity's **AddTorque()** method.
- How Inertia Tensor affects rotation.
- Comparing the rotation of two objects.

## Match the Inertia tensor

- Scale the 2nd sphere so it has the same I value.
- Apply the same torque to both spheres.
- Observe the relative rotation.



```

Vector3 Gravity_Force() {
    return new Vector3(0,-g,0);
}

// Air resistance opposing motion
Vector3 Air_force(){
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return airDragMag * -vDir;
}

```

## Vector Cross Products

```

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaTime = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaTime;
    pos += vel * deltaTime;
    if (isEnabled) transform.position = pos;
}

```



## In This Video...

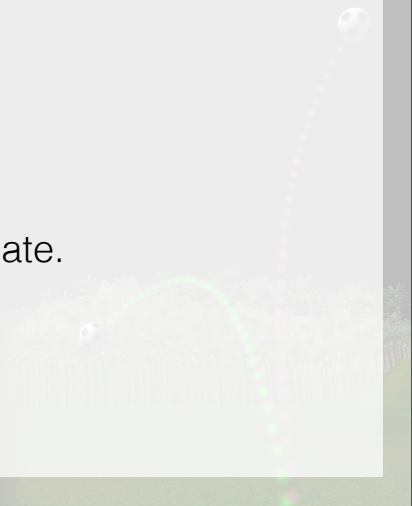
- What a cross product is.
- Why they are so useful.
- Using WolframAlpha to calculate.

```

Vector3 Magnus_Force() {
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float magMag = magConst * vMag;
    Vector3 x = MagnusConst * Vector3.Cross(vel, angular);
    return magMag * x;
}

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaTime = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaTime;
    pos += vel * deltaTime;
    if (isEnabled) transform.position = pos;
}

```



```

Vector3 Gravity_Force() {
    return new Vector3(0,-g,0);
}

// Air resistance opposing motion
Vector3 Air_force(){
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2);
    return airDragMag * -vDir;
}

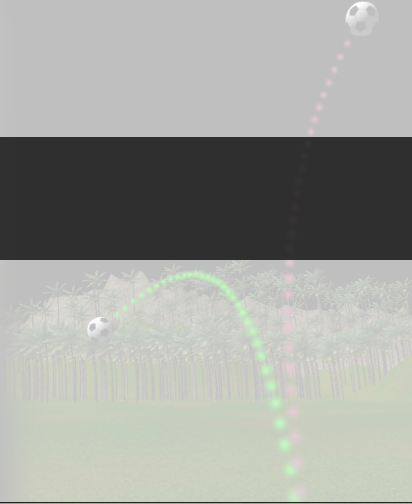
```

## The Magnus Effect

```

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaTime = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaTime;
    pos += vel * deltaTime;
    if (isEnabled) transform.position = pos;
}

```



## In This Video...

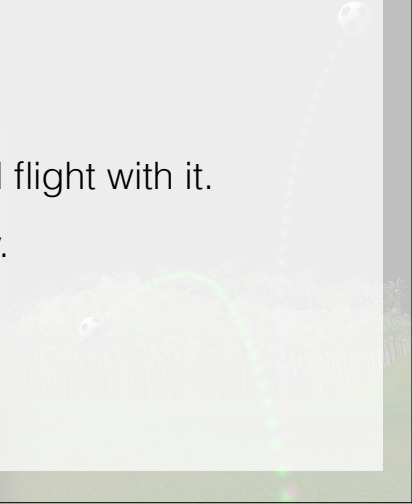
- What the Magnus Effect is.
- How we can approximate ball flight with it.
- Using **Vector3.Cross** in Unity.

```

Vector3 Magnus_Force() {
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float magMag = magConst * vMag;
    Vector3 x = MagnusConst * Vector3.Cross(vel, angular);
    return magMag * x;
}

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaTime = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaTime;
    pos += vel * deltaTime;
    if (isEnabled) transform.position = pos;
}

```



## Write the line of code

- Use **Vector3.Cross**
- Write the one line of code that makes this work.

```
Vector3 AirDragMag = -vel;
return airDragMag * dragConst;

Vector3 x = magnusConst * Vector3.Cross(vel, angular);
return x;

void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}
```



```
Vector3 Gravity_Force() {
    return new Vector3(0,-9,0);
}

// Air resistance opposing motion
Vector3 Air_force(){
    float vMag = vel.magnitude;
    Vector3 vDir = vel.normalized;
    float airDragMag = dragConst * Mathf.Pow(vMag, 2f);
    return airDragMag * -vDir;
}
```

## Section Wrap-Up

```
void ResolveForces (Vector3[] forces) {
    Vector3 combinedForce = Vector3.zero;
    foreach (Vector3 force in forces) {
        combinedForce += force;
    }
    float deltaT = Time.fixedDeltaTime;
    acc = combinedForce / mass;
    vel += acc * deltaT;
    pos += vel * deltaT;
    if (isEnabled) transform.position = pos;
}
```

