# Rotational Motion

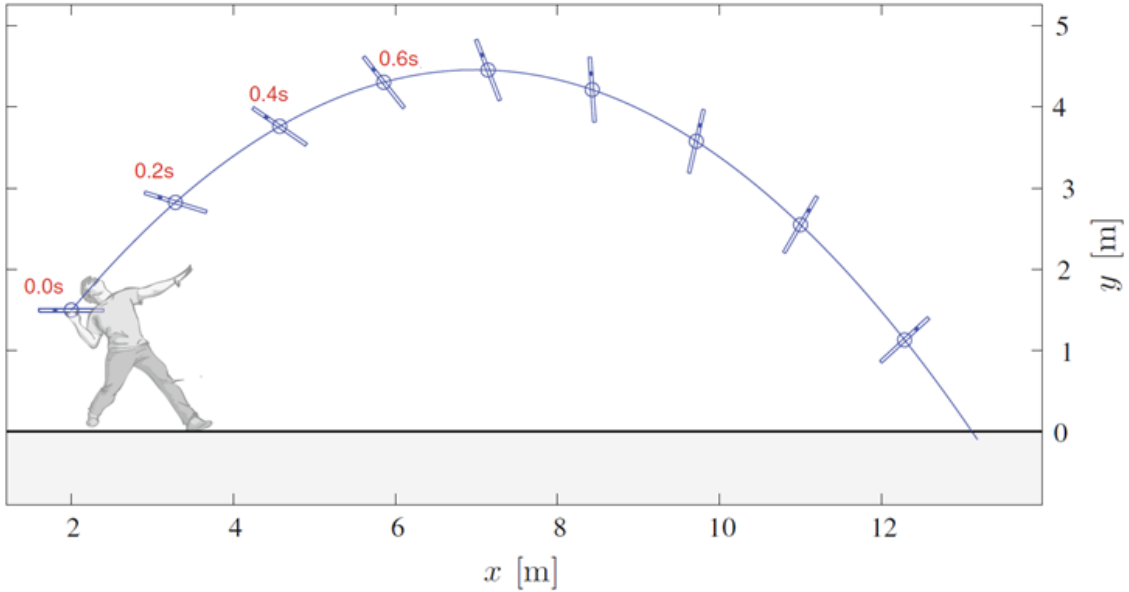
## #10


Serdar ARITAN

Biomechanics Research Group,
Faculty of Sports Sciences, and
Department of Computer Graphics
Hacettepe University, Ankara, Turkey

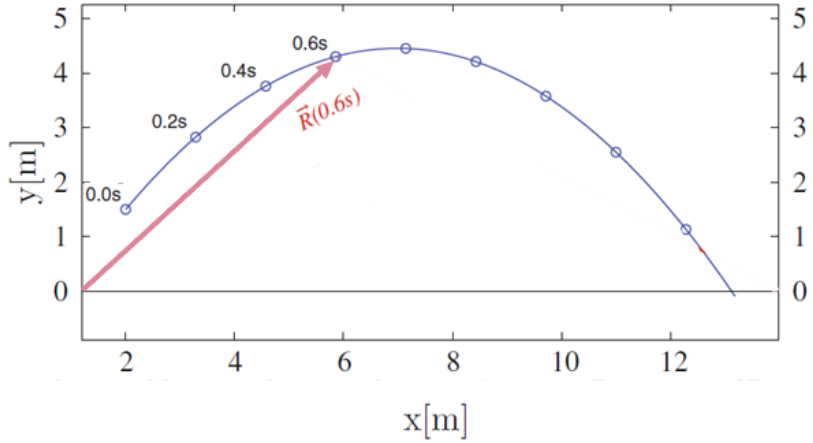
## Angle and Axis of Rotation

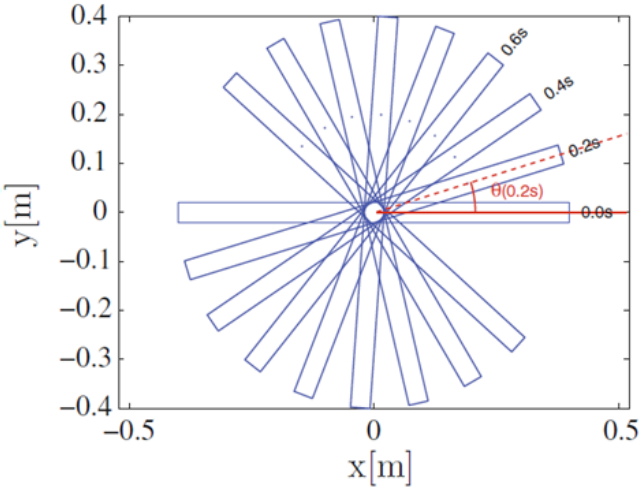
While a freely moving object (such as a rod thrown across the room) usually rotates around its center of mass, objects can also rotate around other points.



## Angle and Axis of Rotation

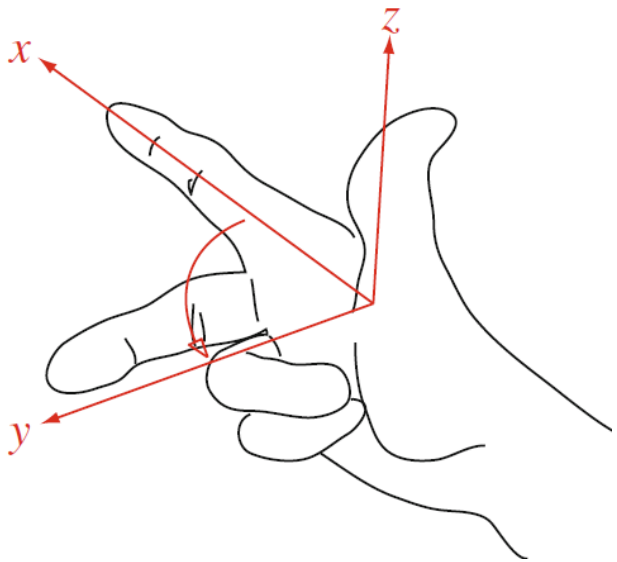
In order to uniquely define the rotational state of the rod we need to specify both the attachment point O and the angle θ the rod forms with the horizontal. But if we only specify the point O, we do not really know how the object rotates around this point. We need to specify the rotational axis as well as a point on the axis.



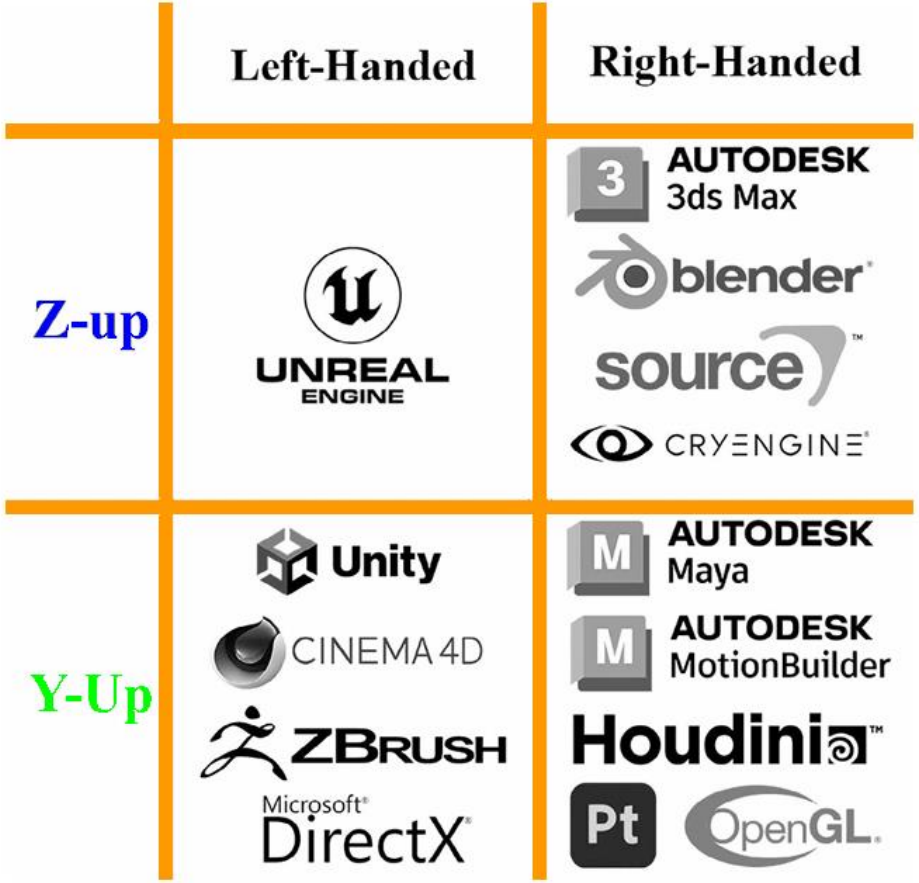



## Angle and Axis of Rotation

How do we describe the positive rotational direction? This is customarily determined by the right hand rule. Given the direction of an axis, such as the z-axis, we can find the positive rotational direction by pointing the right thumb in the direction of the axis: the positive rotational direction is then in the direction your remaining fingers are curling: from the x- towards the y-axis. In this direction θ increases, in the opposite direction the angle decreases.



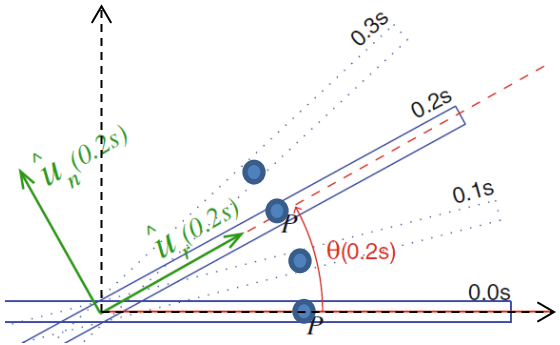
# PHYSICS in COMPUTER ANIMATIONS and GAMES

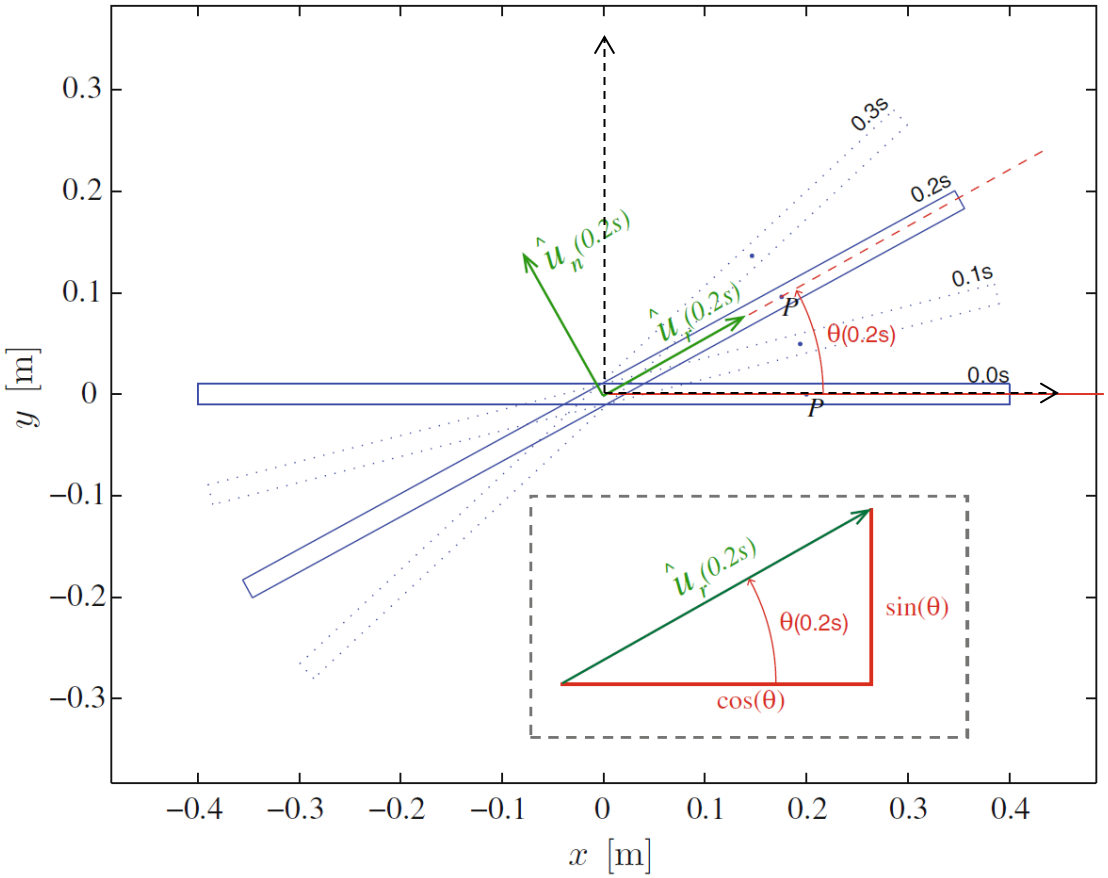


## A Point on a Rotating Object

Given the angle θ and the rotation axis (including both a point on the axis and the positive direction along the axis), we can uniquely determine the orientation of a rotating object. We describe the position of **P** using a coordinate system that rotates along with the object. The rotating coordinate system has to unit vectors that rotate with the object: the unit vector $\hat{u}_r$ , which is directed radially outwards from the rotation axis, and an axis normal to the radial direction with unit vector $\hat{u}_n$. A point on the rod can be described in this coordinate system by:

$$P = p_r \hat{u}_r + p_n \hat{u}_n$$






When the object has rotated an angle θ both unit vectors have also rotated.

$$\hat{u}_r = \cos\theta\,\mathbf{i} + \sin\theta\,\mathbf{j}$$

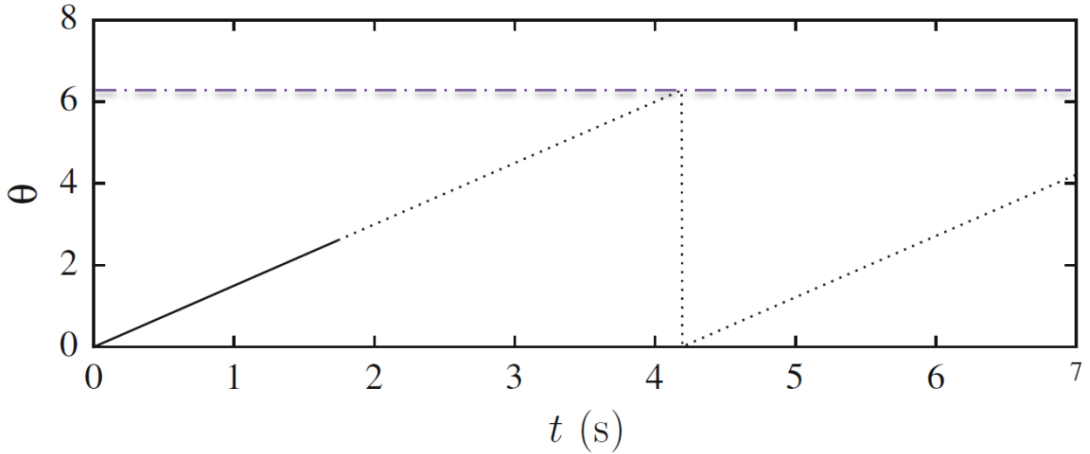
$$\hat{u}_n = -\sin\theta\,\mathbf{i} + \cos\theta\,\mathbf{j}$$

If the object is rotating around a moving axis, we also need to add the position of the axis—here given as the position of the center of mass:

$$p = \boldsymbol{R} + p_r\hat{u}_r + p_n\hat{u}_n$$

## Periodicity of the State θ(t)

The angle, θ, describes a unique configuration of the rod for values from 0 to 2π (measured in radians). What happens when θ(t) increases beyond 2π? When θ reaches 2π the rod has rotated a full revolution, and the rod is in the same position as it was when θ was equal to 0. We cannot discern these positions:




The position of the rod when θ = 2π is exactly the same as when θ = 0.

## Angular Velocity

During rotation, the angle θ(t) changes with time. How can we characterize how fast the rod rotates? By the angular velocity, which is defined as the rate of the change of the angle: We define the average angular velocity over the time Δt as:

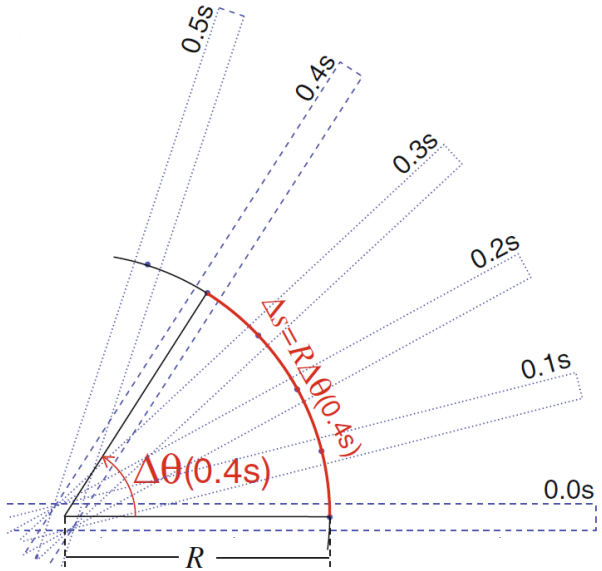
$$\bar{\omega} = \frac{\theta(t+\Delta t) - \theta(t)}{\Delta t} = \frac{\Delta\theta}{\Delta t}$$

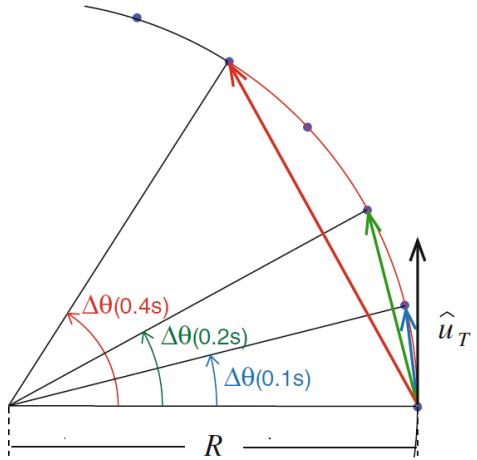
When the time interval becomes small, we find the instantaneous angular velocity for the rotational motion.

$$\omega = \lim_{\Delta t \to 0} \frac{\Delta\theta}{\Delta t} = \frac{d\theta}{dt} = \dot{\theta}$$

## Velocity of a Point on a Rotating Body

As the rod rotates, every part of the rod moves in a circle around the rotation axis. What is the velocity of a small part of the rotating rod, and how can we relate it to the angular velocity?



## Velocity of a Point on a Rotating Body

During the small time interval Δt, the rod has rotated an angle Δθ from the orientation θ(t) to the new orientation θ(t + Δt) = θ(t) + Δθ. How far has P moved? It has moved the arc length Δs = RΔθ along its circular path. The speed of the small part P is therefore:

$$v = \frac{\Delta s}{\Delta t} = R\frac{\Delta \theta}{\Delta t}$$

If we let the time interval Δt become infinitesimally small, we find the speed of the point P to be:

$$v = \frac{ds}{dt} = \frac{d}{dt}(R\theta) = R\frac{d\theta}{dt} = R\omega$$

## Motion with Constant Angular Velocity

If an object rotates with a constant angular velocity, we can find the speed of the point P from the distance traveled during one complete revolution, s = 2π R, divided by the time of one revolution, call the period T:

$$v = \frac{s}{t} = \frac{2\pi R}{T}$$

where R is the distance from P to the rotation axis. We also know that the velocity is v = Rω, therefore we find that:

$$v = \frac{2\pi}{T} R = \omega R \Rightarrow \omega = \frac{2\pi}{T}$$

The **angular velocity** is often also called the **angular frequency**.

## Angular Acceleration

The rate of change of the angular velocity by the angular acceleration, α.

$$\alpha = \frac{d\omega}{dt} = \frac{d^2\theta}{dt} = \ddot{\theta}$$

Comparison of linear and rotational motion

| Motion | Linear | Rotation |
|---|---|---|
| Position | $x(t)$ | $\theta(t)$ |
| Velocity | $v(t) = \frac{dx}{dt}$ | $\omega(t) = \frac{d\theta}{dt}$ |
| Acceleration | $a(t) = \frac{dv}{dt} = \frac{d^2x}{dt^2}$ | $\alpha(t) = \frac{d\omega}{dt} = \frac{d^2\theta}{dt^2}$ |

## Rotation in Pygame

Be Careful: There is a aliasing problem in rotation !!

```
def rot_center(image, angle):
    """rotate an image while keeping its center and size"""
    """ It *only* works with square images """
    orig_rect = image.get_rect()
    rot_image = pygame.transform.rotate(image, angle)
    rot_rect = orig_rect.copy()
    rot_rect.center = rot_image.get_rect().center
    rot_image = rot_image.subsurface(rot_rect).copy()
    return rot_image
```

## Rotation in Pygame

Be Careful: There is a aliasing problem in rotate

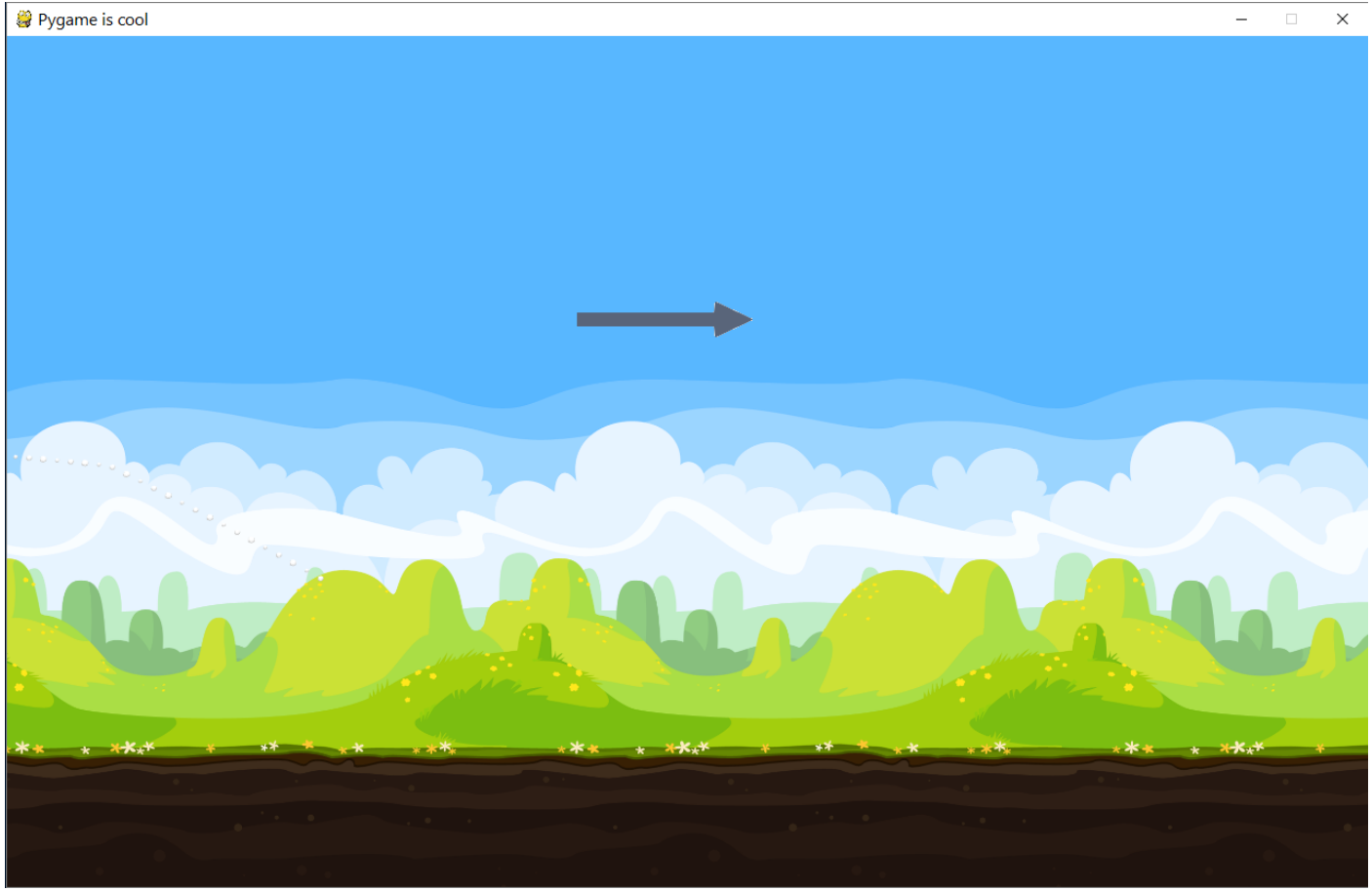
```
def rot_center_rect(image, rect, angle):
    """rotate an image while keeping its center"""
    rot_image = pygame.transform.rotate(image, angle)
    rot_rect = rot_image.get_rect(center = rect.center)
    return rot_image, rot_rect
```

## Rotation in Pygame

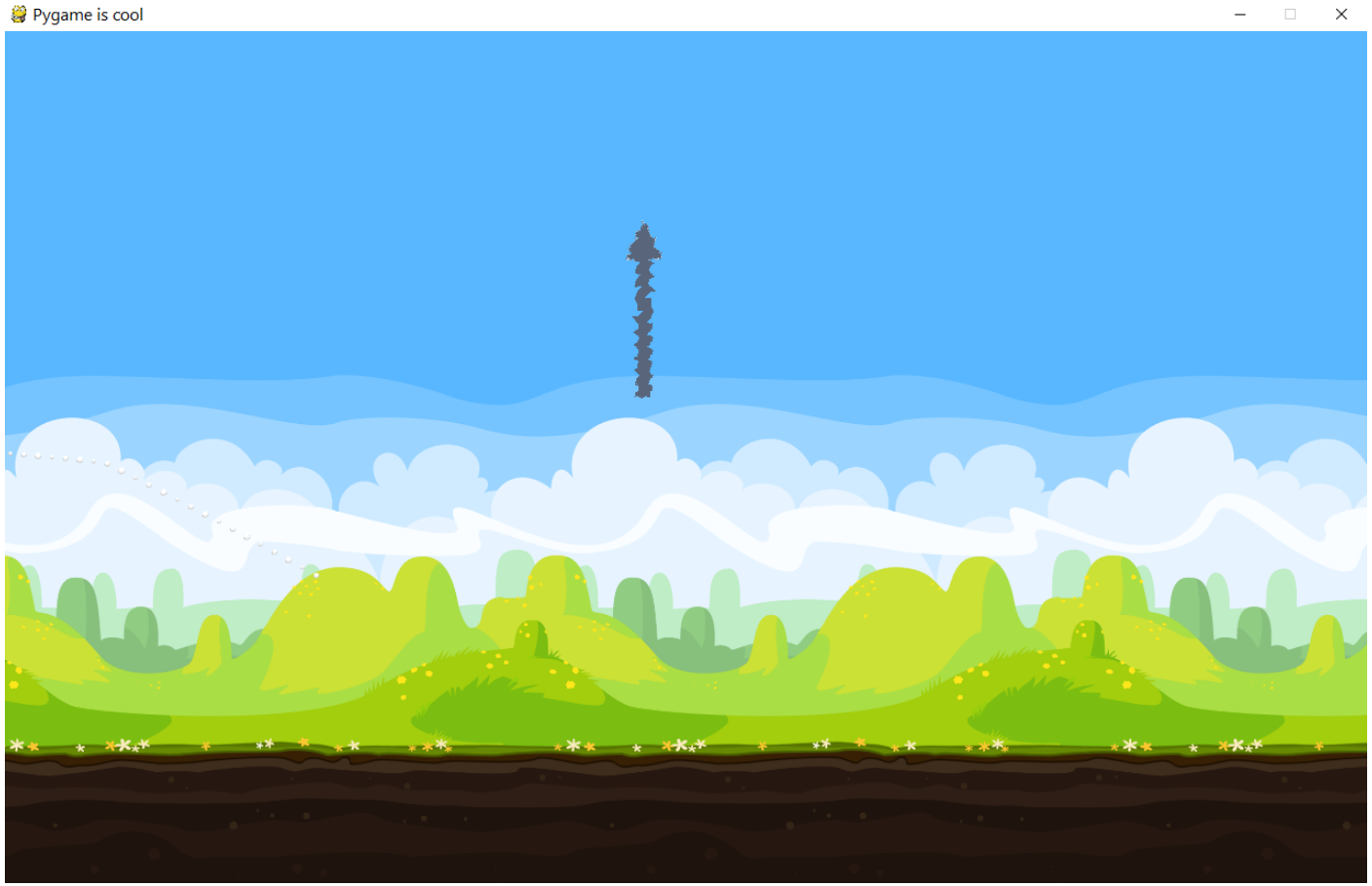
```
while not done:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      done = True
    elif event.type == pygame.MOUSEBUTTONDOWN:
      player_image = rot_center(player_image, 5)



#player_image, rotRect = rot_center_rect (player_image,rotRect,5)
```

## Rotation in Pygame

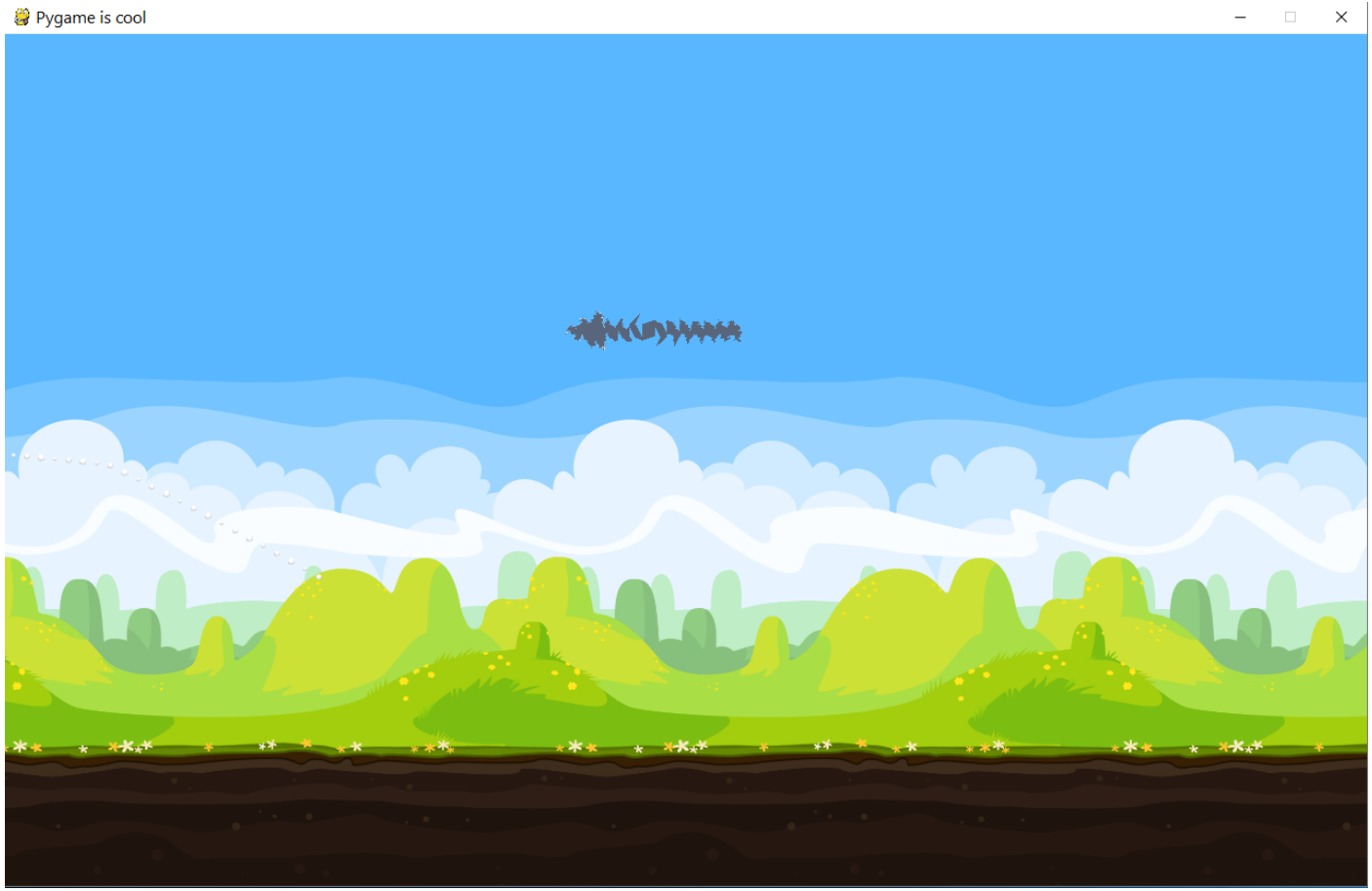
```
# Copy image to screen
screen.blit(background_image, background_position)
# Get the current mouse position. This returns the position
# as a list of two numbers.
player_position = pygame.mouse.get_pos()
x = player_position[0] - 100
y = player_position[1] - 100
# Copy image to screen
screen.blit(player_image, [x, y])
# screen.blit(player_image, rotRect)
pygame.display.flip()
clock.tick(60)
```

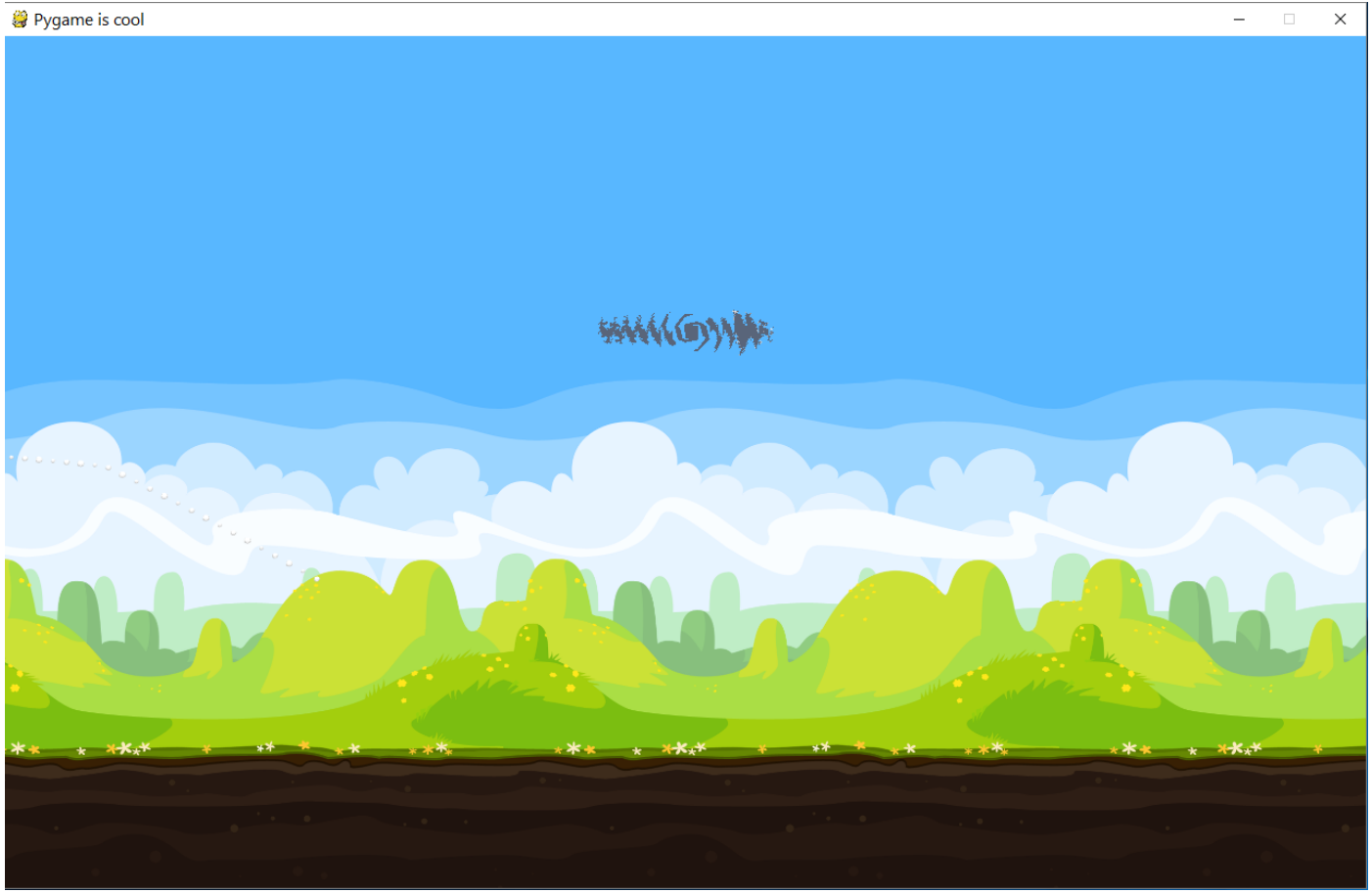

Pygame is cool

# PHYSICS in COMPUTER ANIMATIONS and GAMES



# PHYSICS in COMPUTER ANIMATIONS and GAMES

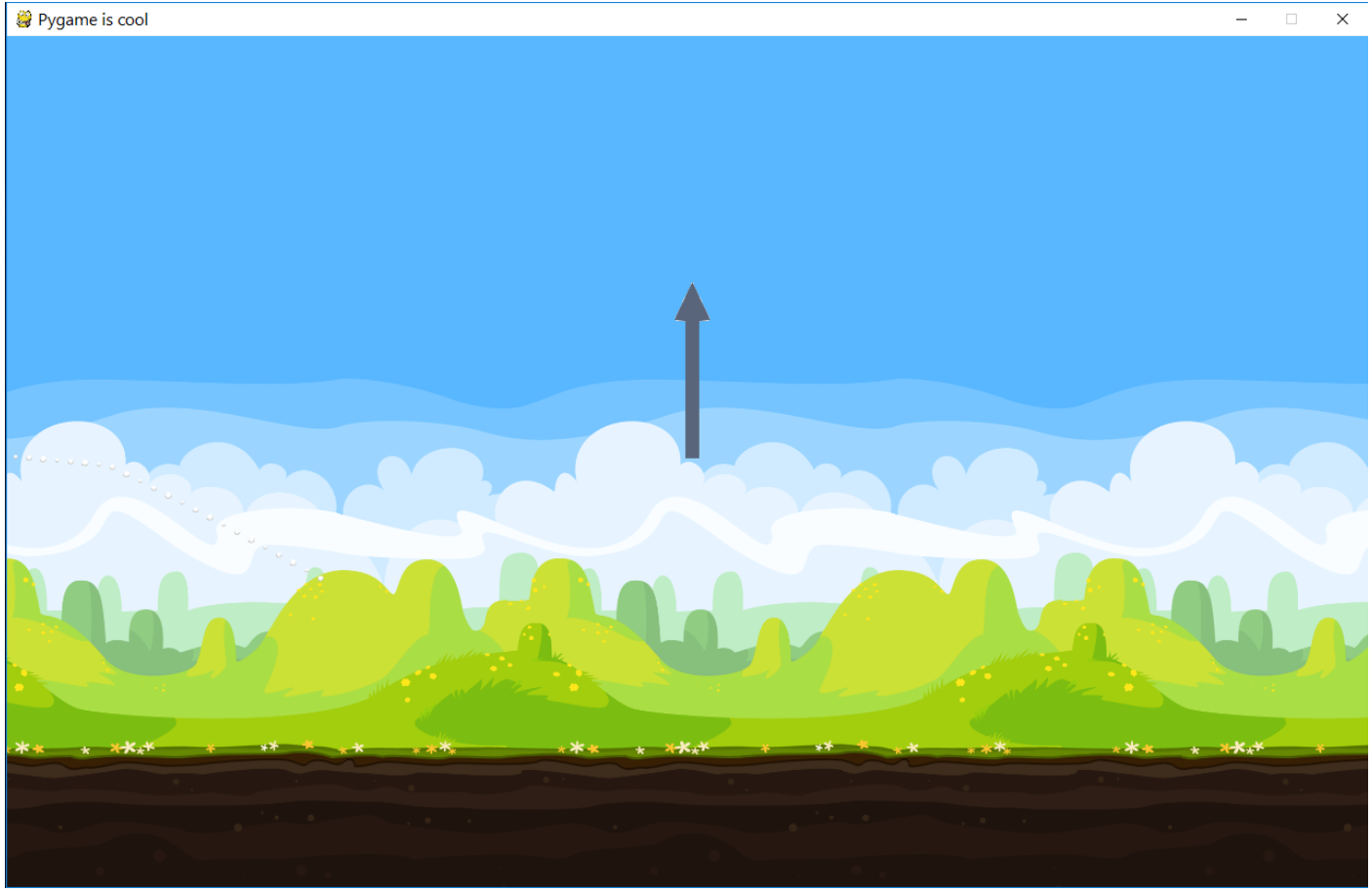


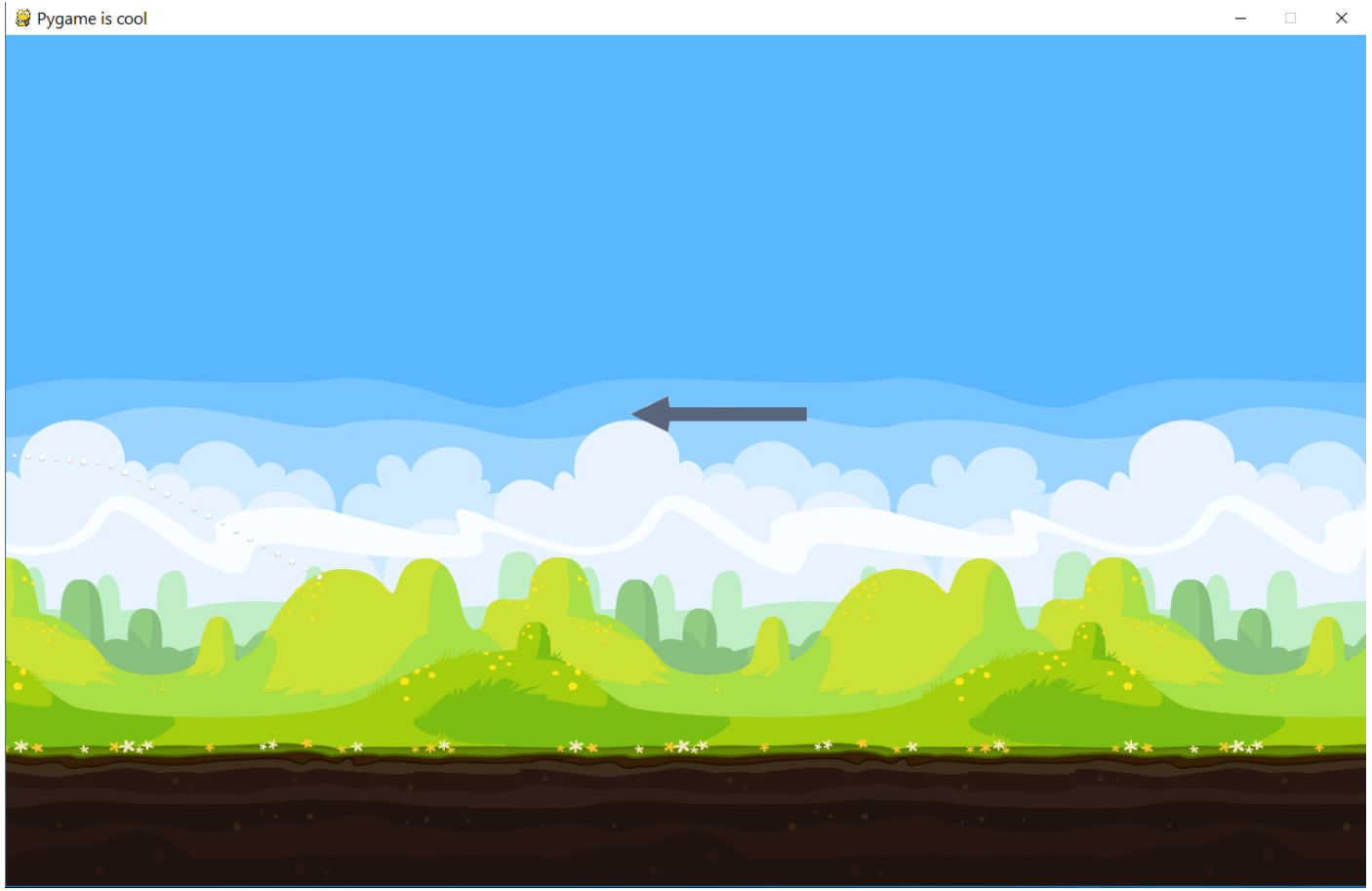

Pygame is cool

```
angle = 0
done = False

while not done:
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      done = True
    elif event.type == pygame.MOUSEBUTTONDOWN:
      angle += 3
            if angle >= 360: angle = 0
```

# PHYSICS in COMPUTER ANIMATIONS and GAMES

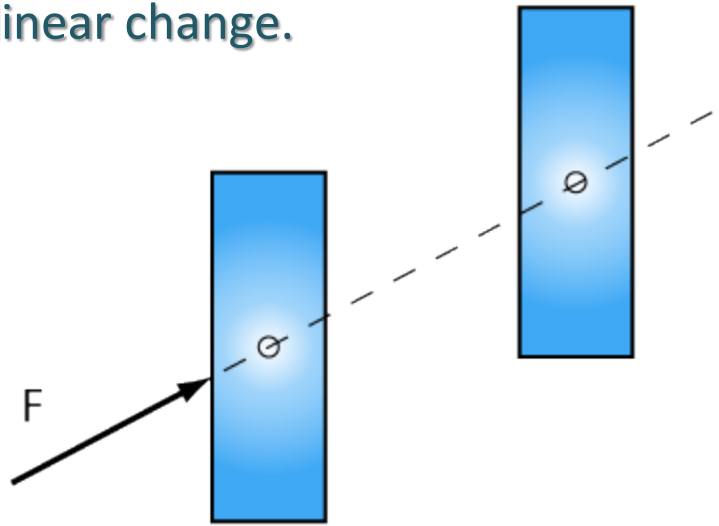
```
# Copy image to screen
screen.blit(background_image, background_position)
# Get the current mouse position. This returns the position
# as a list of two numbers.
player_position = pygame.mouse.get_pos()
x = player_position[0] - 100
y = player_position[1] - 100
# Copy image to screen
screen.blit(rot_center(player_image, angle), [x, y])
# screen.blit(player_image, rotRect)
pygame.display.flip()
clock.tick(60)
```


Pygame is cool


Pygame is cool

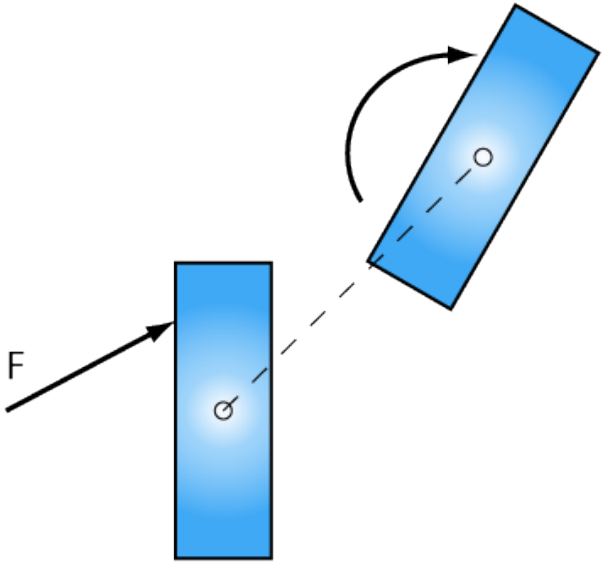
## Force Applied Directly Thru the Center of Mass

When a force is applied thru the center of mass of an object, the object will translate in the direction of the force. The magnitude of the translation is directly related to the magnitude of the net force and inversely related to the objects resistance to linear change.



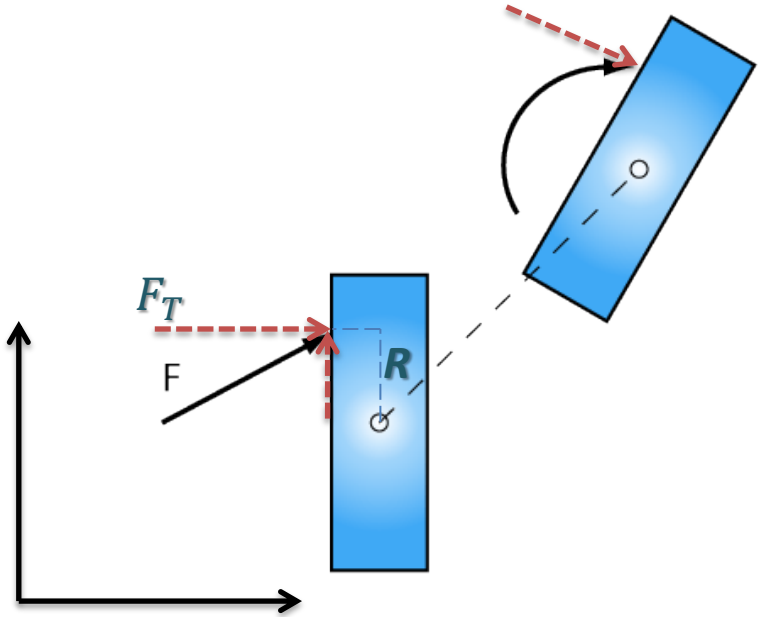
## Force Not Applied Directly Thru the Center of Mass

When a force is not applied directly thru the center of mass of a rigid object it will cause both translation and rotation.



## Force Not Applied Directly Thru the Center of Mass

## Free Rotation of a Rigid Body



## Rigid Bodies

In physics, a rigid body is an idealization of a solid body in which deformation is neglected. In other words, the distance between any two given points of a rigid body remains constant in time regardless of external forces exerted on it.

The position of a rigid body is the position of all the particles of which it is composed. To simplify the description of this position, we exploit the property that the body is rigid, namely that all its particles maintain the same distance relative to each other. If the body is rigid, it is sufficient to describe the position of at least three non-collinear particles.

## Kinetic Energy of a Rotating Rigid Body

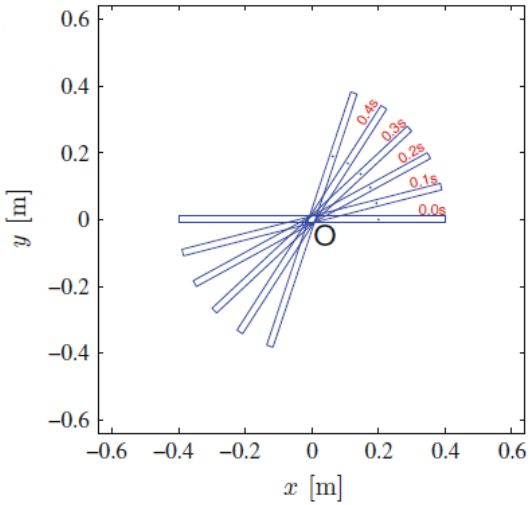
we found that the kinetic energy of a body can be expressed as:

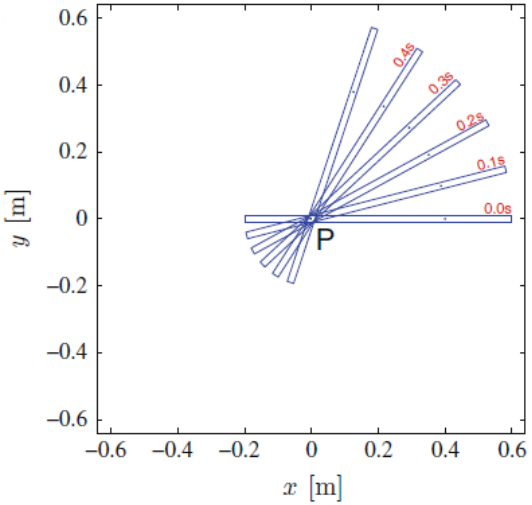
$$K = \frac{1}{2}M(V)^2 + \frac{1}{2}\sum_{i=1}^{N} m_i \left(v_{cm,i}\right)^2$$

where the first term is related to the motion of the center of mass and describes the kinetic energy for the translational motion. The second term is related to the motion relative to the center of mass. For a rigid body, the only way a part of the body can move relative to the center of mass is by rotation. We therefore interpret the second term as kinetic energy for the rotational motion, and we will here introduce the kinetic energy for rotating objects.

## Kinetic Energy of a Rotating Rigid Body

An object can rotate either around a fixed axis or around a moving axis such as a moving center of mass. Here, we first address the behavior of a rigid body rotating around a fixed axis, and then generalize to the case of an axis following the center of mass motion.



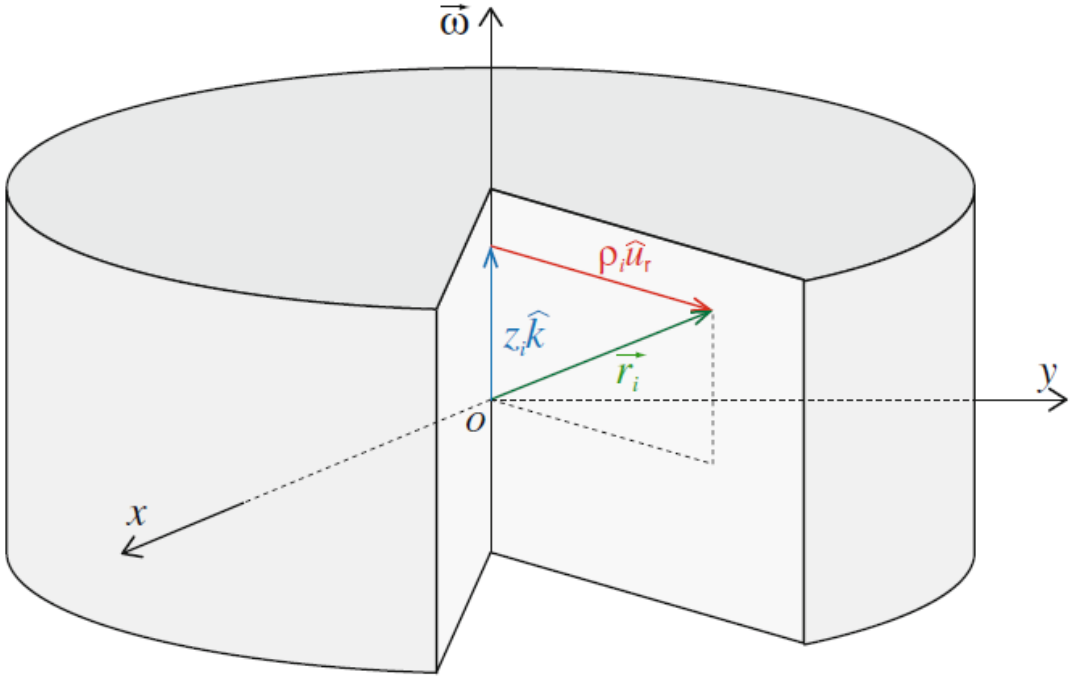
## Rotation Around a Fixed Axis

In order to determine the kinetic energy of the rigid body we assume that the body consists of small mass-points with masses $\mathbf{m_i}$ and positions, $\mathbf{r_i}$ , where the positions are measured relative to an origin placed on the rotation axis. The whole body is rotating with the angular velocity **ω** around the fixed axis through the origin, **O**. What is the velocity of point **i** on the body?

$$v_i = \omega \times r_i$$

We can simplify the description by decomposing the position into two vectors: the vector $\rho_i$ , directed normal to the z-axis, and the vector $z_i$ k directed along the z-axis.

$$r_i = \rho_i + z_i k$$




$$v_i = \omega\,\mathbf{k} \times (\rho_i + z_i\mathbf{k}) = \omega\,\mathbf{k} \times \rho_i + z_i\omega\,\underbrace{\mathbf{k} \times \mathbf{k}}_{=0} = \omega\,\mathbf{k} \times \rho_i$$

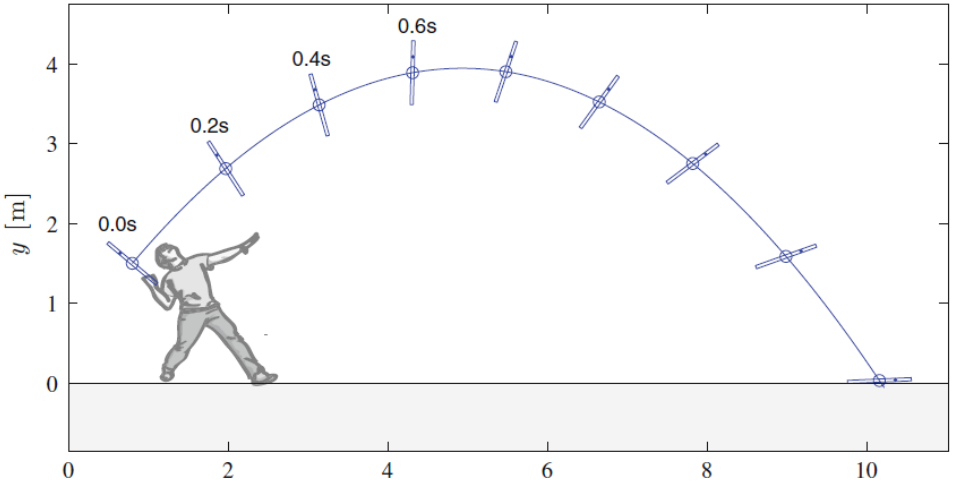
In order to find the kinetic energy, we only need the magnitude of the velocity for each point:

$$K = \sum_{i=1}^{N} \frac{1}{2} m_i v_i^2 = \sum_{i=1}^{N} \frac{1}{2} m_i (\omega \rho_i)^2 = \frac{1}{2} \underbrace{\left( \sum_{i=1}^{N} m_i \rho_i^2 \right)}_{= I_0} \omega^2 = \frac{1}{2} I_0 \omega^2$$

where we have introduced the quantity $\mathbf{I_0}$, which we call the moment of inertia of the rigid body about the axis **O**:

$$I_0 = \sum_{i=1}^{N} m_i \rho_i^2$$

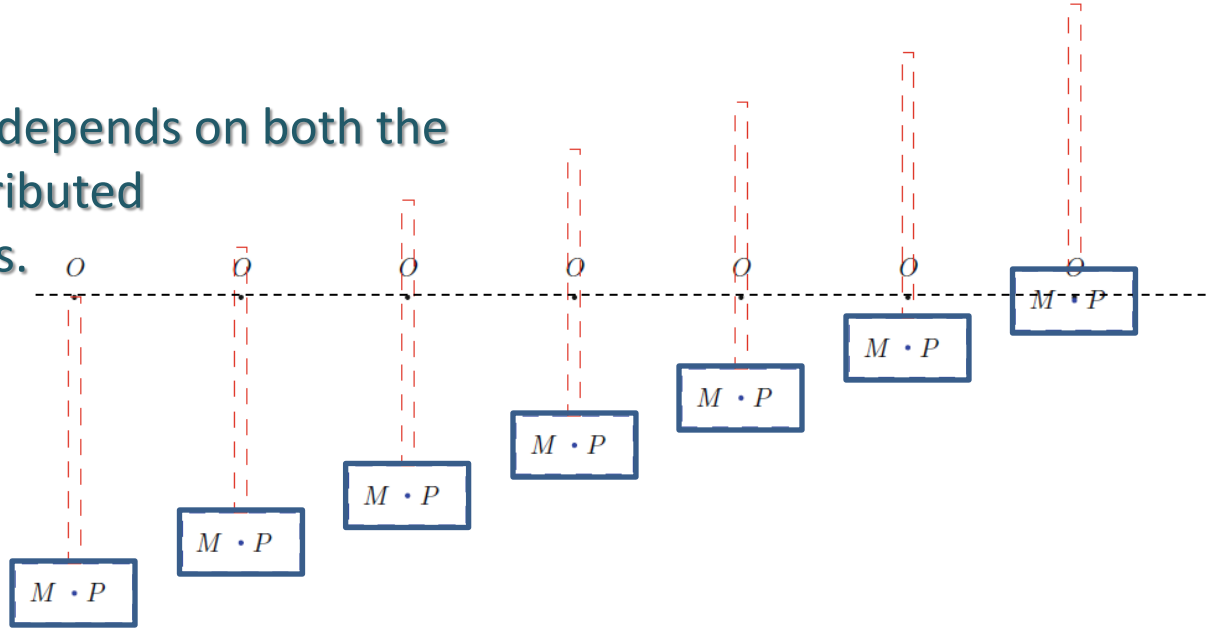
If the rod is a rigid body, the only possible motion relative to the center of mass is a rotational motion. The total kinetic energy is therefore: :




$$K = \frac{1}{2}MV^2 + \frac{1}{2}\sum_{i=1}^{N} m_i I_{cm} \omega^2$$

## the Moment of Inertia

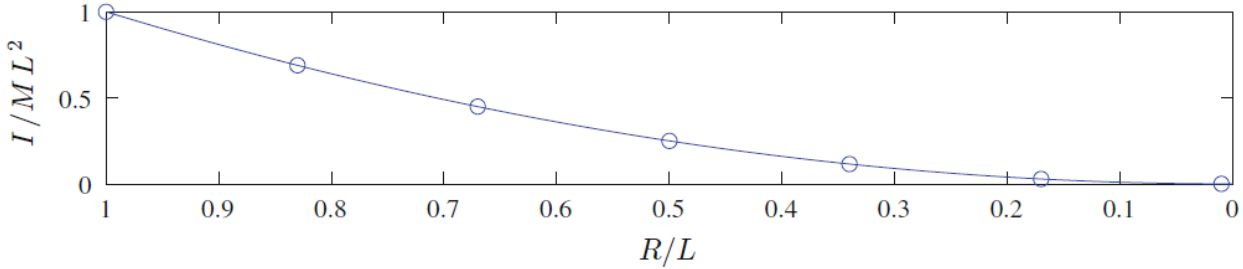
The moment of inertia depends on both the **mass** and how it is distributed around the rotation axis.



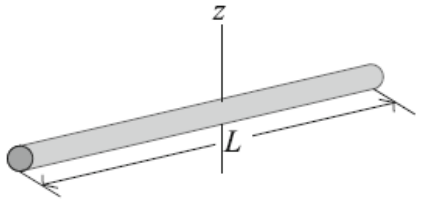

The moment of inertia of the hammer is

$$I_0 = M\,R^2$$

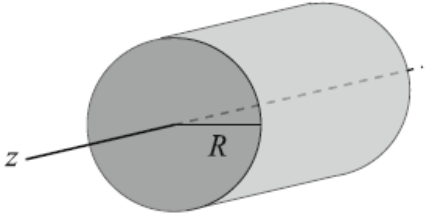
where ***R*** is the distance from the head, ***P***, to the rotation axis ***O***.



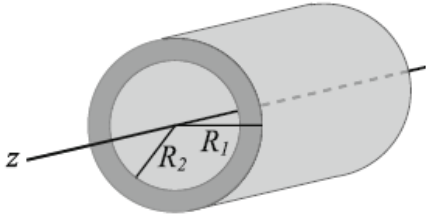
## Moments of inertia for various solid bodies



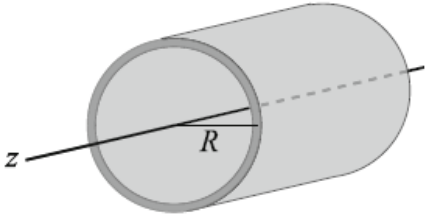

Thin rod of length $L$.
$I = \frac{1}{12}ML^2$



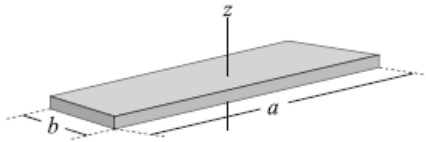

Solid cylinder of radius $R$.
$I = \frac{1}{2}MR^2$



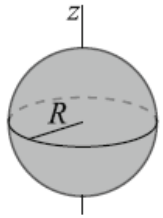

Hollow cylinder.
$I = \frac{1}{2}M\left(R_1^2 + R_2^2\right)$




Cylinder shell of radius $R$.
$I = MR^2$

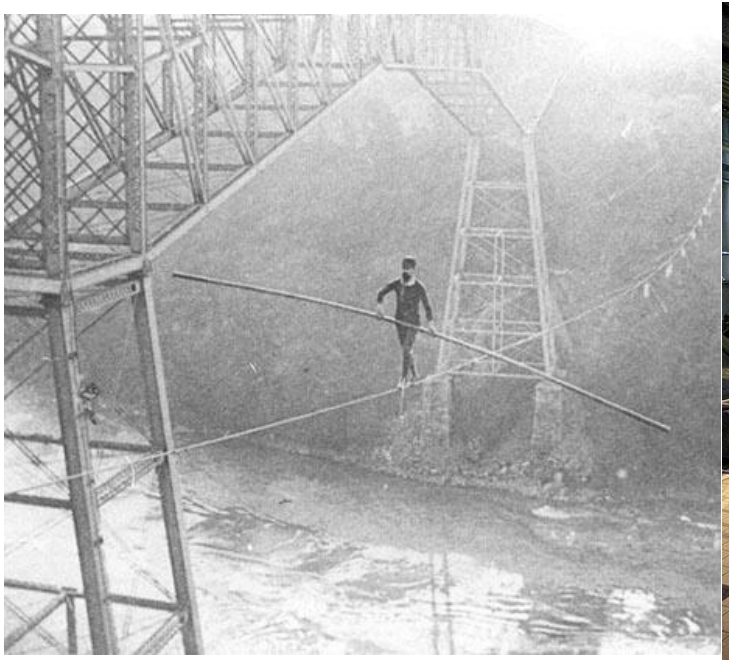



Thin plate of size $a \times b$
$I = \frac{1}{12}M\left(a^2 + b^2\right)$



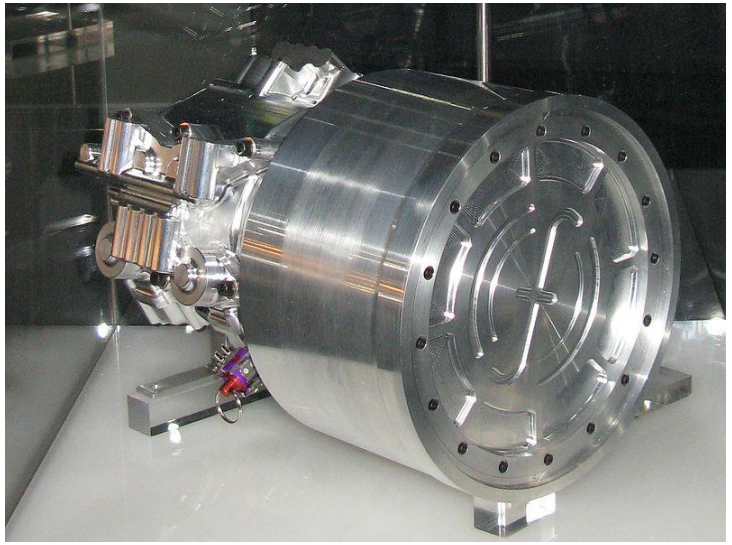

Sphere of radius $R$
$I = \frac{2}{5}MR^2$

## Moments of inertia



A flywheel is a mechanical device specifically designed to efficiently store rotational energy. Flywheels resist changes in rotational speed by their **moment of inertia.**

## Moments of inertia

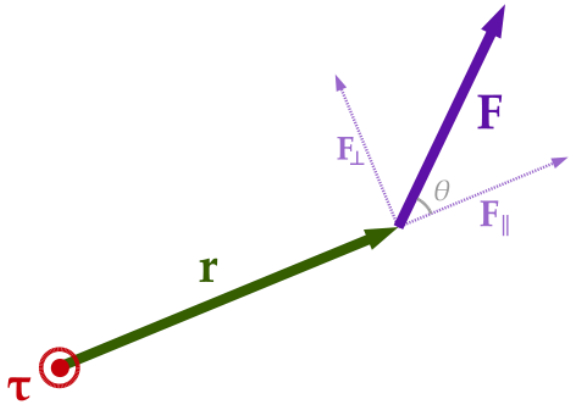
Using a continuously variable transmission (CVT), energy is recovered from the drive train during braking and stored in a flywheel. This stored energy is then used during acceleration by altering the ratio of the CVT.

## Newton's Second Law for Rotational Motion

| | |
|---|---|
| The force by the torque | $F \rightarrow F_T R$ |
| The translational inertia (mass) by the rotational inertia | $m \rightarrow I$ |
| The acceleration by the angular acceleration | $a \rightarrow \alpha$ |

From Newton's second law for rotational motion , we interpret the torque $RF_T = \tau$ as the cause of the angular acceleration, just as we interpreted the force as the cause of acceleration for translational motion. For a given torque, $\tau = F_T R$, a larger value of $I$ means a smaller angular acceleration. Also, we see that the torque $\tau = F_T R$ depends on both the tangential force, $F_T$ and the distance to the rotation axis, $R$: If we apply the same force $F$ further out from the rotation axis, we get a larger torque and a larger angular acceleration.

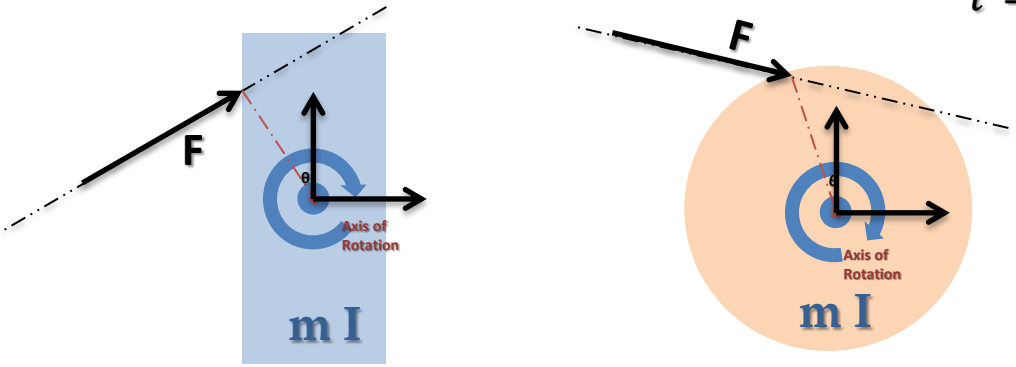
## Rotation around a fixed axis




A particle is located at position **r** relative to its axis of rotation. When **a force F** is applied to the particle, only the perpendicular component F⊥ produces a torque. This **torque** $\tau = r \times F$ has magnitude $\tau = |r|\,|F_\perp| = |r|\,|F| \sin\theta$ and is directed outward from the screen.

## Rotation around a fixed axis

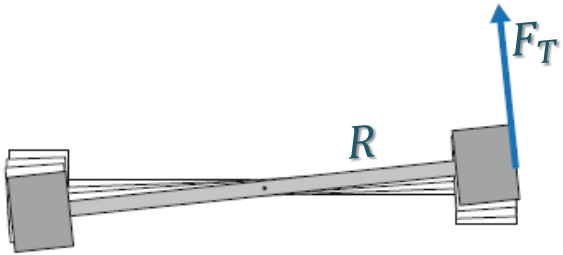
$$\vec{\tau} = \vec{r} \times \vec{F} = \mathrm{I} \times \vec{\alpha}$$

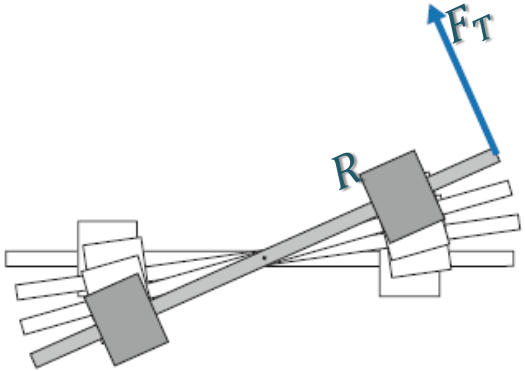


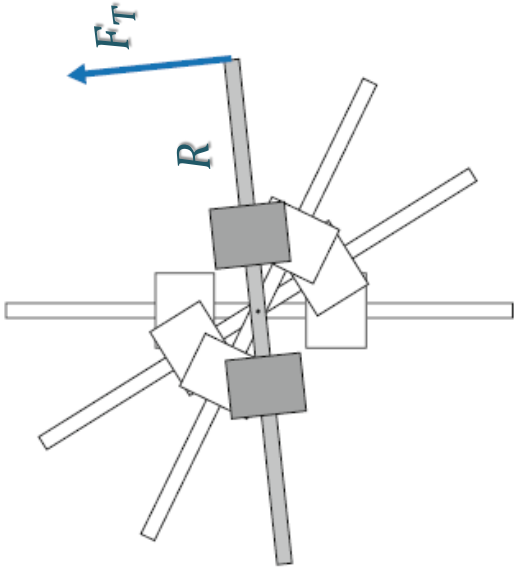

**τ** is the magnitude of the torque,
**r** is the position vector (a vector from the origin of the coordinate system defined to the point where the force is applied)
**F** is the force vector

$$\tau = F_T R = I \propto$$

**Top View**




$\propto_1$ $\propto_2$ $\propto_3$

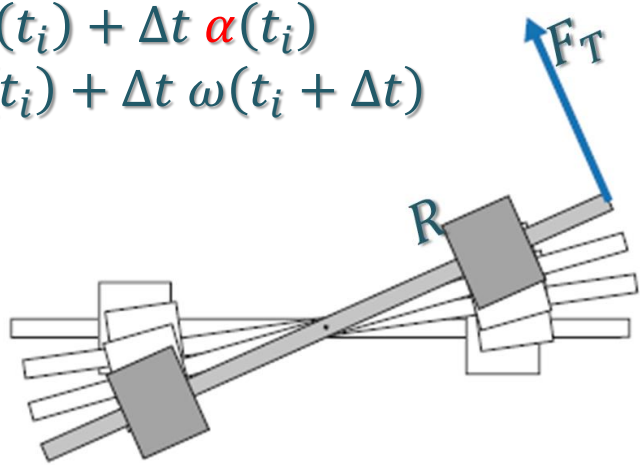
$I_1$ $I_2$ $I_3$

## Numerical Integration

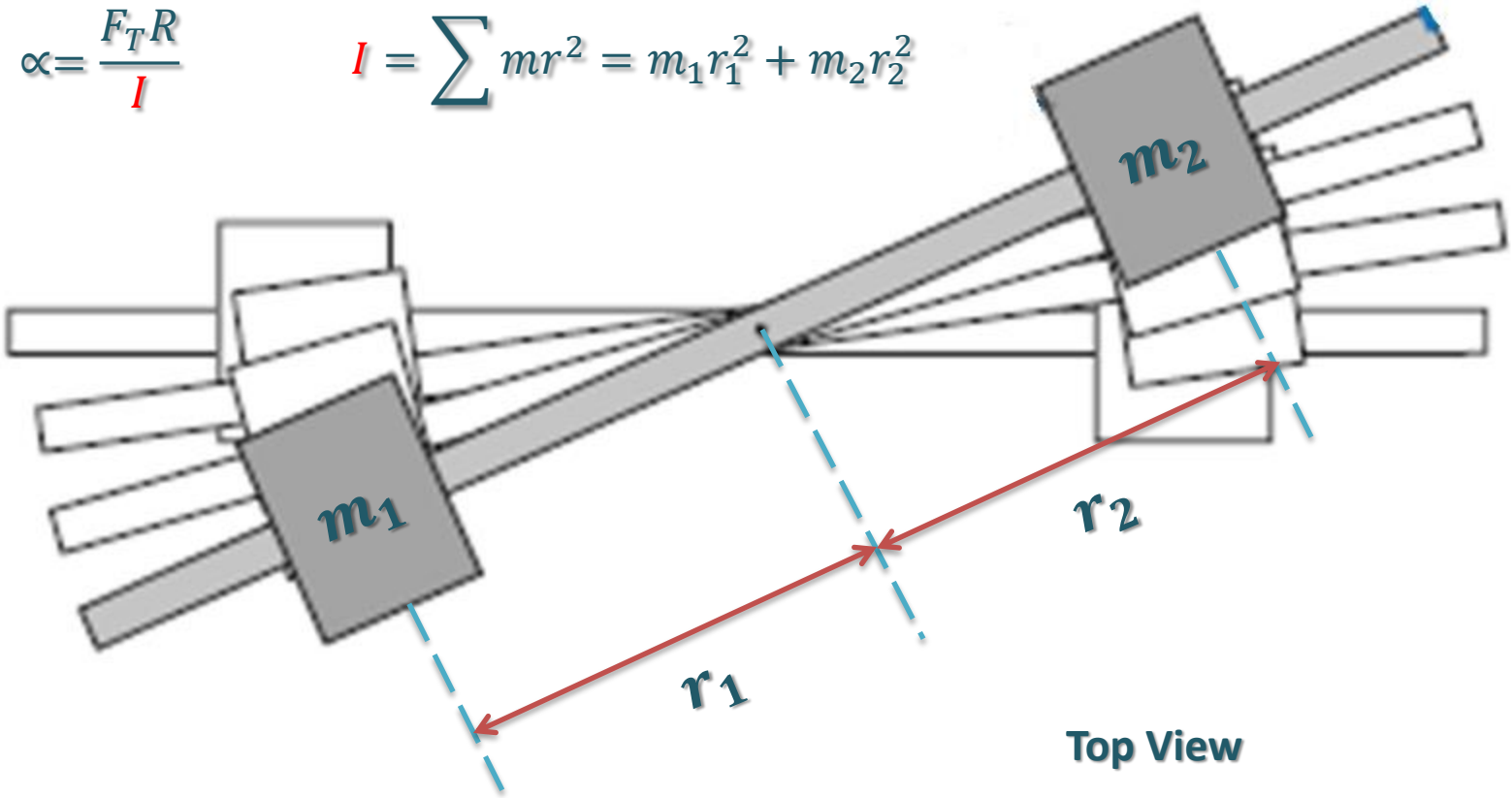
$$\propto = \frac{F_T R}{I}$$

Euler's method for angular motion: Euler's method follows exactly the same scheme as for linear motion:

$$\omega(t_i + \Delta t) = \omega(t_i) + \Delta t\ \alpha(t_i)$$
$$\theta(t_i + \Delta t) = \theta(t_i) + \Delta t\ \omega(t_i + \Delta t)$$



$$\propto = \frac{F_T R}{I} \qquad I = \sum mr^2 = m_1 r_1^2 + m_2 r_2^2$$

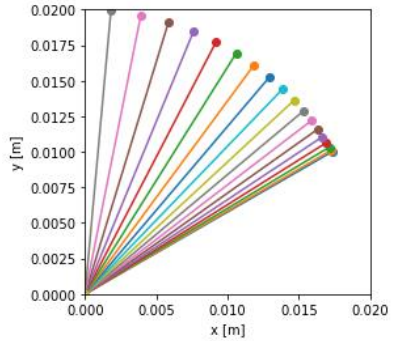


# PHYSICS in COMPUTER ANIMATIONS and GAMES

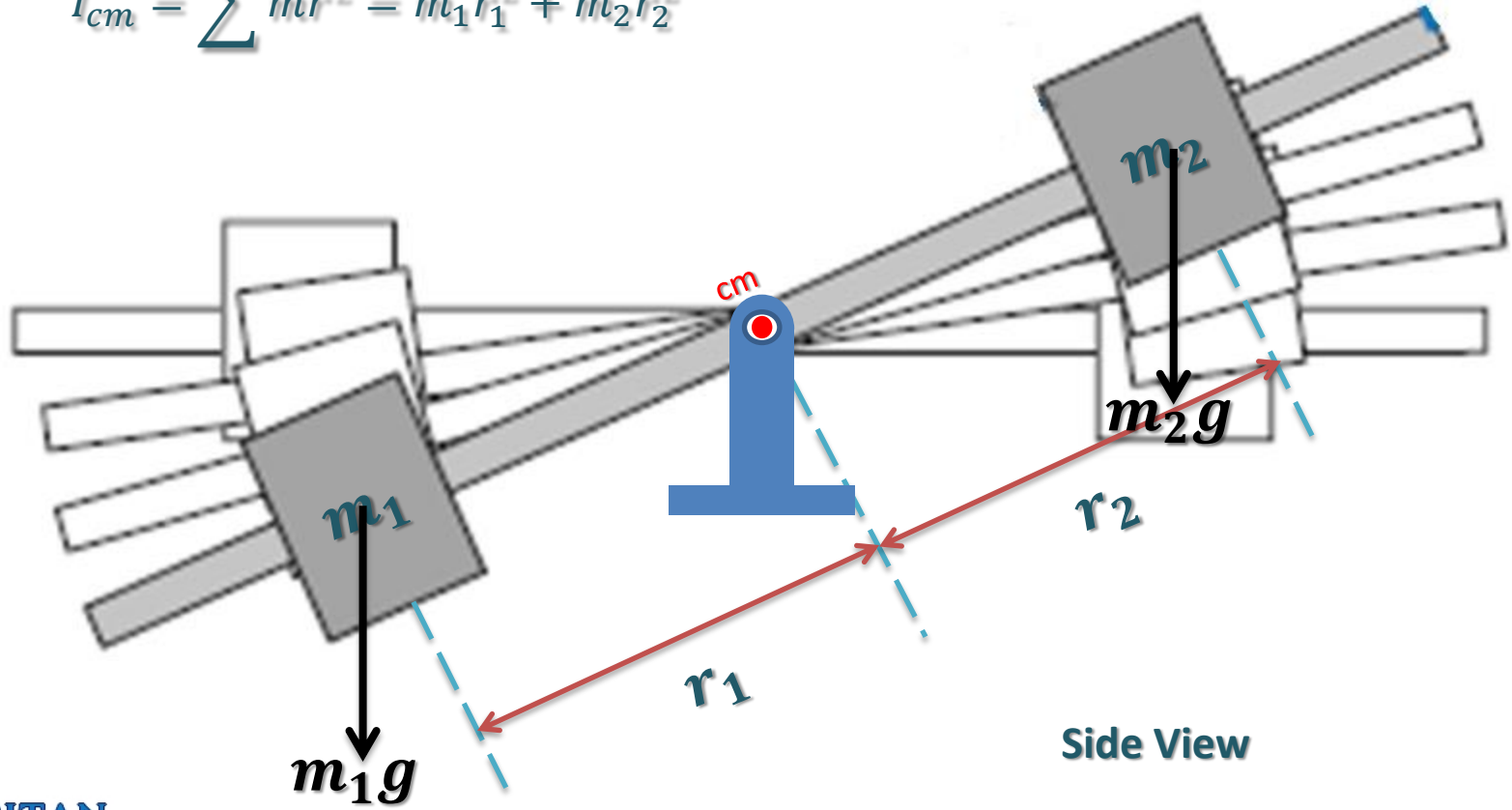
```
import numpy as np
import matplotlib.pyplot as plt

R1 = 0.02    # m
M1 = 4       # kg
R2 = 0.02    # m
M2 = 4       # kg
I = M1*R1**2 + M2*R2**2
g = 9.8      # m/s^2
F = 10       # N
omega0 = 0.0     # rad/s
time = .20       # s
dt = 0.01        # s
```

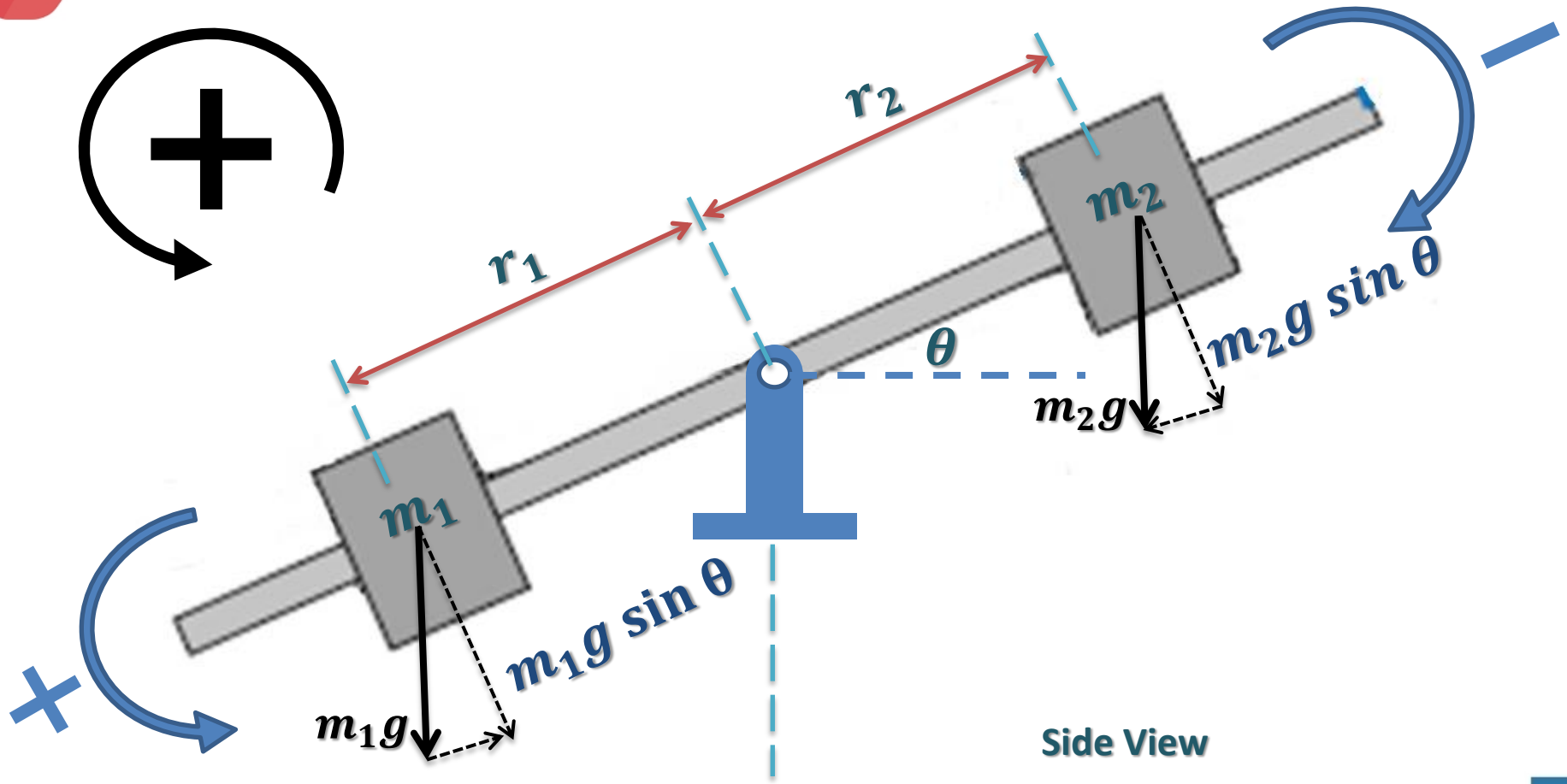
```
# Numerical variables
n = int(round(time/dt))
theta = np.zeros(n, float)
omega = np.zeros(n, float)
t = np.zeros(n, float)

# Initialize
fig, ax = plt.subplots()
theta[0] = 0
omega[0] = omega0
```

```
# Integration loop
for i in range(n-1):
    r = np.array([R1*np.cos(theta[i]), R1*np.sin(theta[i])])
    tauz = F*R1
    alpha = tauz/I
    omega[i+1] = omega[i] + alpha*dt
    theta[i+1] = theta[i] + omega[i+1]*dt
    t[i+1] = t[i] + dt
    if (np.mod(i, 2)==0):
        ax.plot(np.array([0, r[0]]), np.array([0, r[1]]),'-o')
        ax.set_xlabel('x [m]')
        ax.set_ylabel('y [m]')
        ax.axis([0, R1, 0, R1])
        ax.set_aspect('equal')
```



$$I_{cm} = \sum mr^2 = m_1 r_1^2 + m_2 r_2^2$$




+
r2
r1
m2
θ
m2g sin θ
m2g
m1
m1g sin θ
m1g
+
−
Side View

# PHYSICS in COMPUTER ANIMATIONS and GAMES

```
import numpy as np
import matplotlib.pyplot as plt

R1 = 0.02     # m
M1 = 4        # kg
R2 = 0.02     # m
M2 = 4        # kg
I = M1*R1**2 + M2*R2**2
g = 9.8       # m/s^2

theta0 = np.radians(30)          # rad
omega0 = 0.0                     # rad/s
time = 0.20                      # s
dt = 0.01                        # s
```

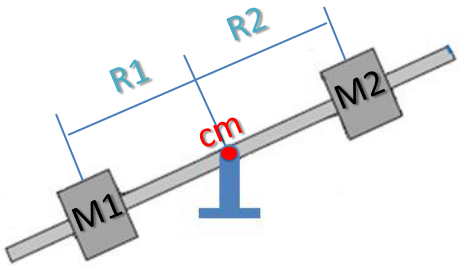
# PHYSICS in COMPUTER ANIMATIONS and GAMES

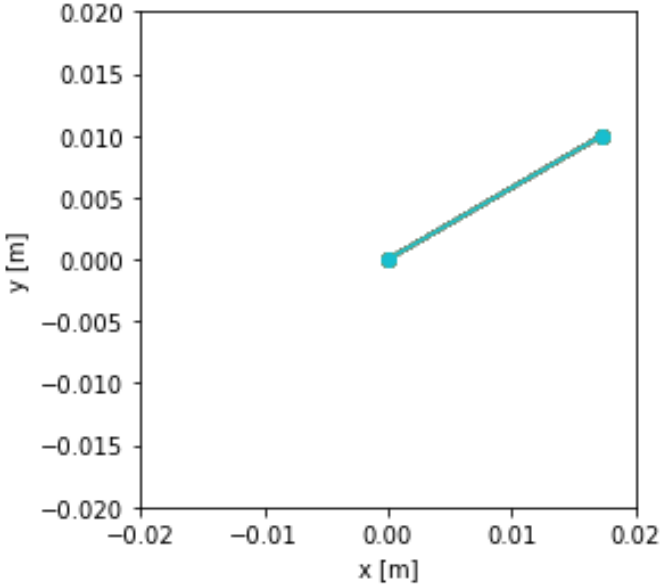
```
# Numerical variables
n = int(round(time/dt))
theta = np.zeros(n, float)
omega = np.zeros(n, float)
t = np.zeros(n, float)

# Initialize
fig, ax = plt.subplots()
theta[0] = theta0
omega[0] = omega0
```

# PHYSICS in COMPUTER ANIMATIONS and GAMES

```
# Integration loop
for i in range(n-1):
    r = np.array([R2*np.cos(theta[i]), R2*np.sin(theta[i])])
    tauz = M1*g*np.sin(theta[i])*R1 - M2*g*np.sin(theta[i])*R2
    alpha = tauz/I
    omega[i+1] = omega[i] + alpha*dt
    theta[i+1] = theta[i] + omega[i+1]*dt
    t[i+1] = t[i] + dt
    if (np.mod(i, 2)==0):
        ax.plot(np.array([0, r[0]]), np.array([0, r[1]]),'-o')
        ax.set_xlabel('x [m]')
        ax.set_ylabel('y [m]')
        ax.axis([-R1, R1, -R1, R1])
        ax.set_aspect('equal')
```

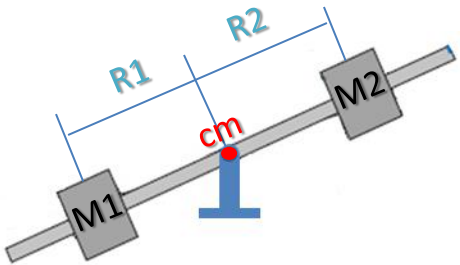
```
R1 = 0.02     # m
M1 = 4        # kg
R2 = 0.02     # m
M2 = 4        # kg
theta0 = radians(30)
```

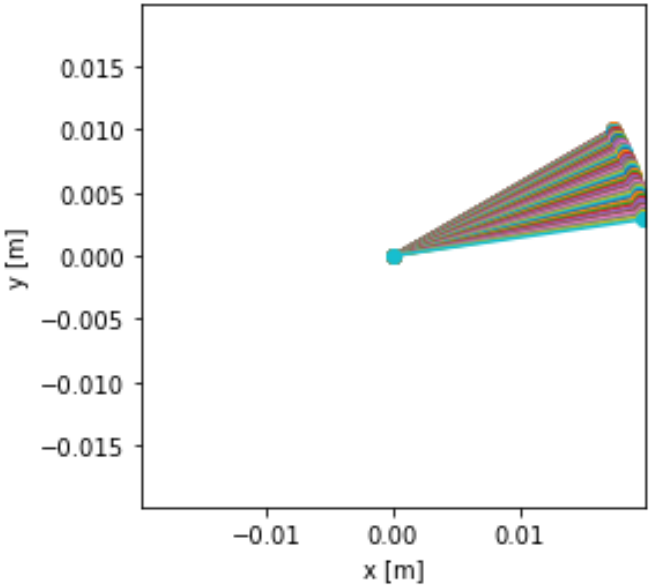



?

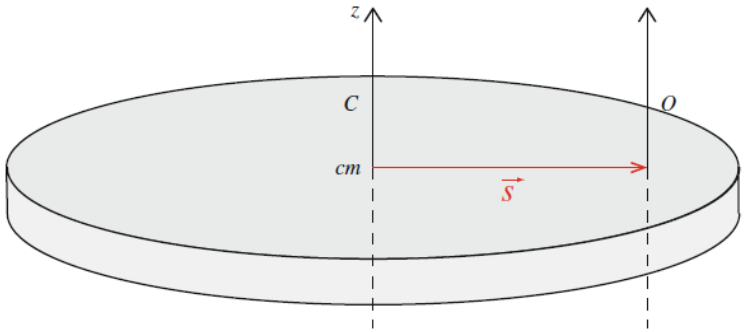
```
R1 = 0.0199 # m
M1 = 4      # kg
R2 = 0.02   # m
M2 = 4      # kg
theta0 = radians(30)
```



## Parallel-Axis Theorem

We can use the parallel-axis theorem to find the moment of inertia around any axis if we only know the moment of inertia around the center of mass. This is why you usually only find tabulated the moment of inertia of an object around its center of mass.




The moment of inertia, $I_O$, of an object around an axis ***O*** is related to the moment of inertia, $I_{cm}$, of the object around a parallel axis through the center of mass of the object by

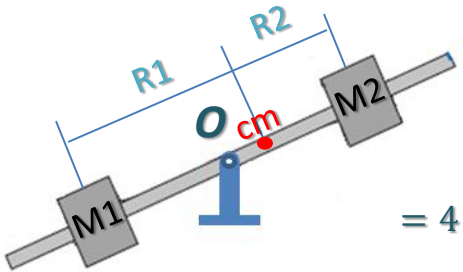
$$I_o = I_{cm} + ms^2$$

where ***m*** is the mass of the object, and ***s*** is the distance between the axis ***O*** and the parallel axis through the center of mass.

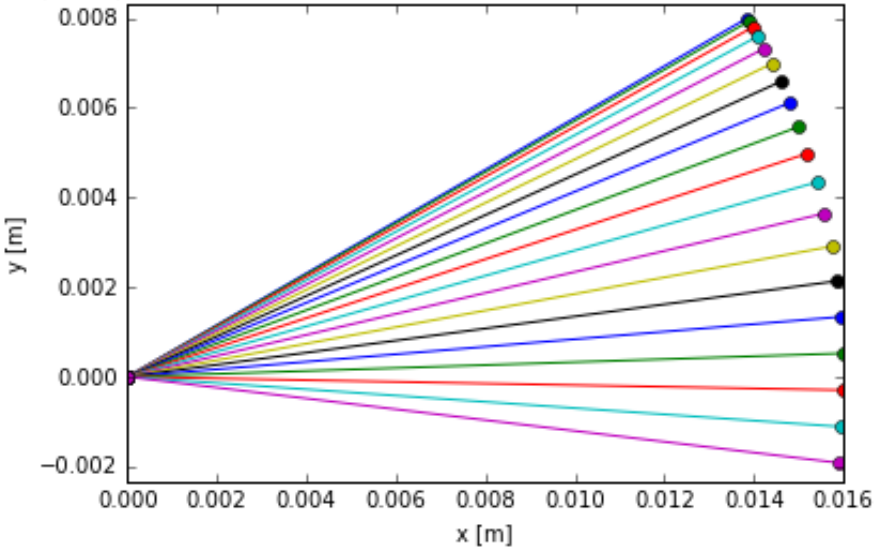
# PHYSICS in COMPUTER ANIMATIONS and GAMES

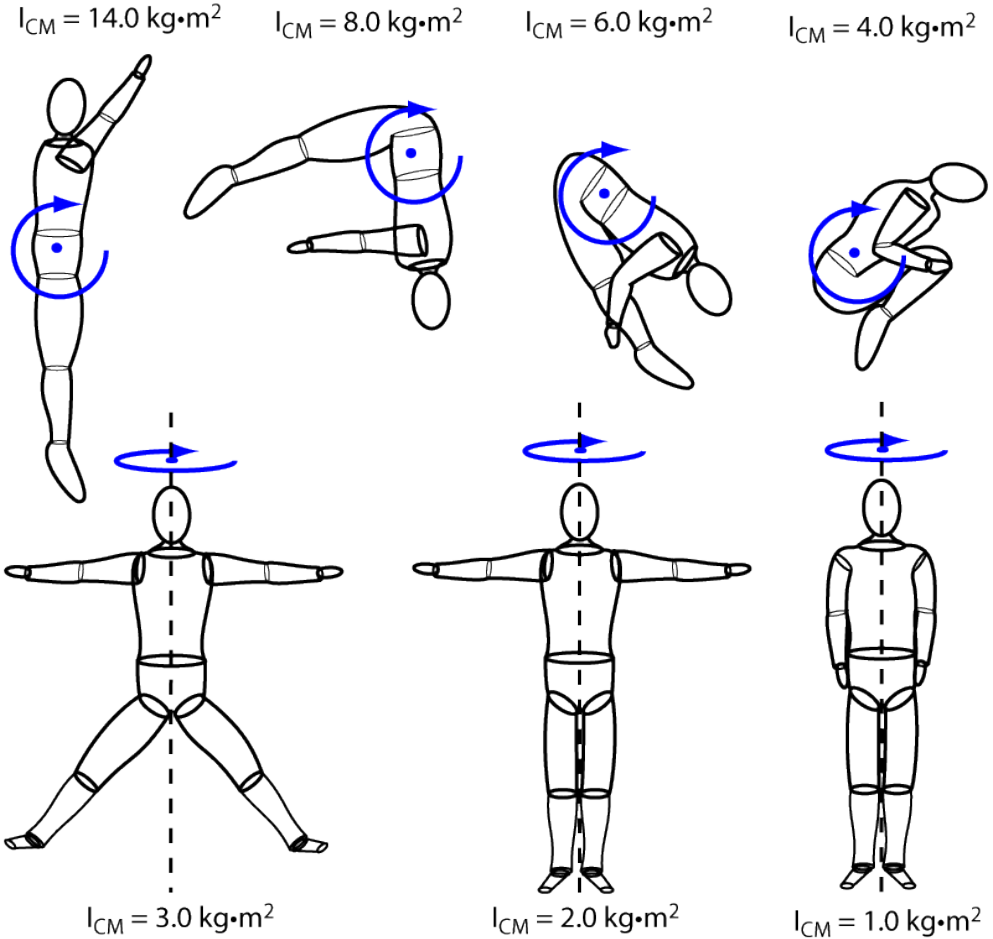
$$I_o = I_{cm} + Ms^2 = M_1R_1^2 + M_2R_2^2 + (M_1 + M_2)s^2$$

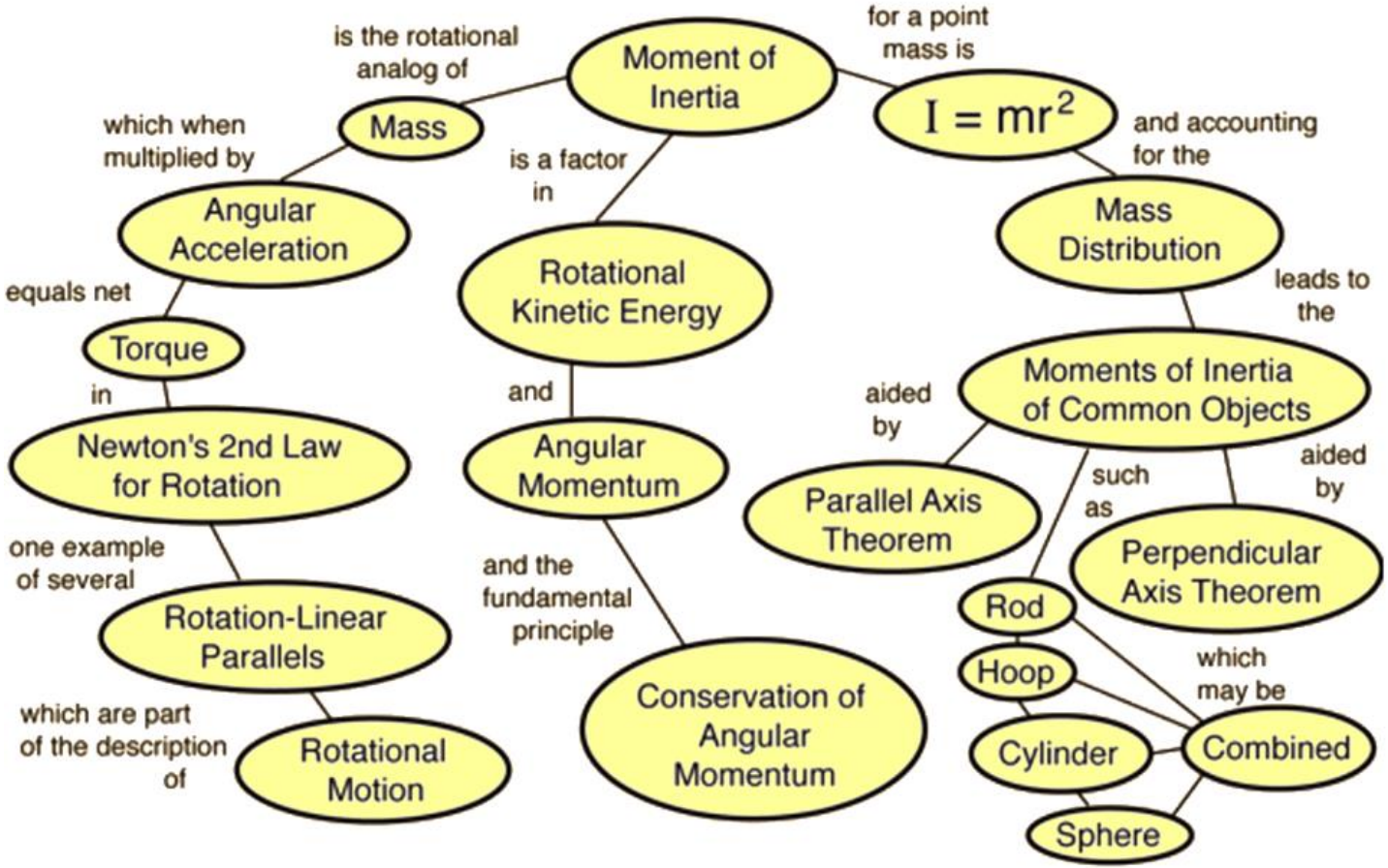
```
R1 = 0.024    # m
M1 = 4        # kg
R2 = 0.016    # m
M2 = 6        # kg
Ro = 0.02
theta0 = radians(30)
```




$$= 4\ 0.024^2 + 6\ 0.016^2 + 10\ 0.004^2$$




$I_{CM}$ = 14.0 kg•m²
$I_{CM}$ = 8.0 kg•m²
$I_{CM}$ = 6.0 kg•m²
$I_{CM}$ = 4.0 kg•m²
$I_{CM}$ = 3.0 kg•m²
$I_{CM}$ = 2.0 kg•m²
$I_{CM}$ = 1.0 kg•m²

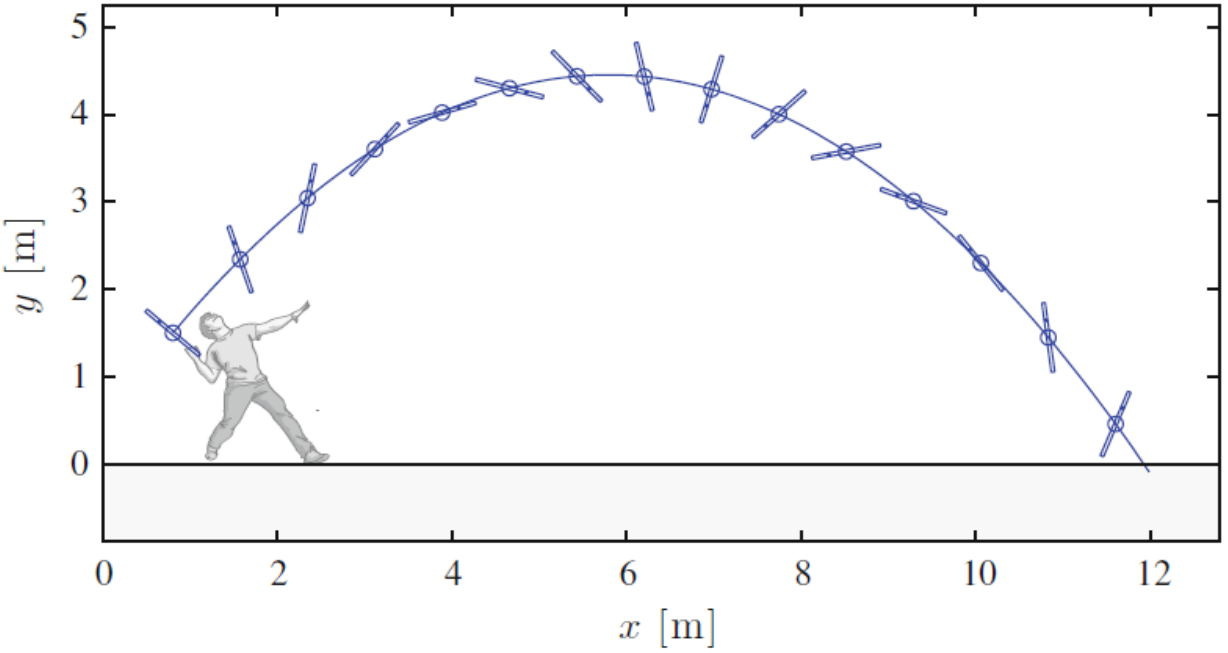

Moment of Inertia
is the rotational analog of
Mass
for a point mass is
$I = mr^2$
which when multiplied by
Angular Acceleration
is a factor in
Rotational Kinetic Energy
and accounting for the
Mass Distribution
equals net
Torque
in
Newton's 2nd Law for Rotation
and
Angular Momentum
leads to the
Moments of Inertia of Common Objects
aided by
Parallel Axis Theorem
such as
aided by
Perpendicular Axis Theorem
Rod
Hoop
Cylinder
Sphere
which may be
Combined
one example of several
Rotation-Linear Parallels
which are part of the description of
Rotational Motion
and the fundamental principle
Conservation of Angular Momentum

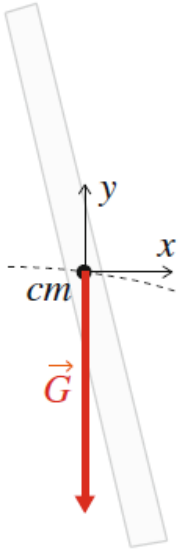
## Rotational Motion Around a Moving Center of Mass

A rod being thrown across the lecture hall. After it has been thrown it is just affected by gravity and air resistance. We know that the motion of the center of mass of the object only depends on the external forces acting on the object—its motion is determined from **Newton's second law of motion**. But what about the rotational motion around the center of mass? The rotational motion around the center of mass for a rigid body is determined from Newton's second law for rotational motion around the center of mass.

## Rotational Motion Around a Moving Center of Mass

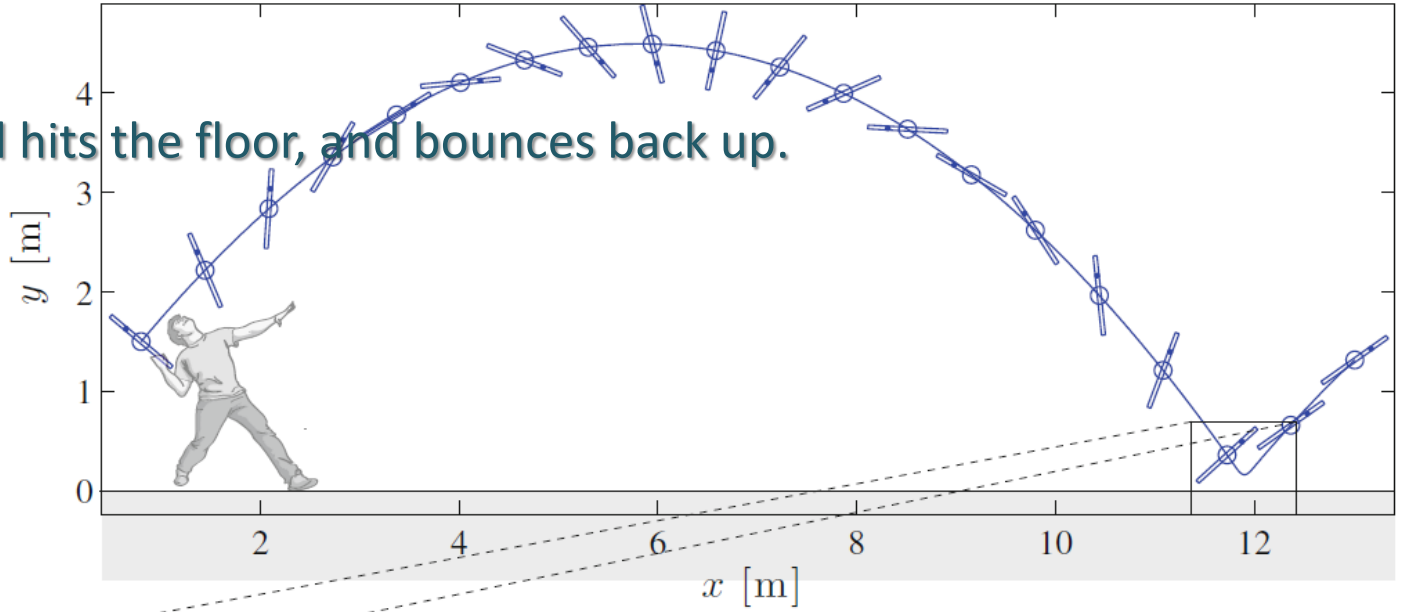
When the rod is not in contact with the floor only gravity acts.

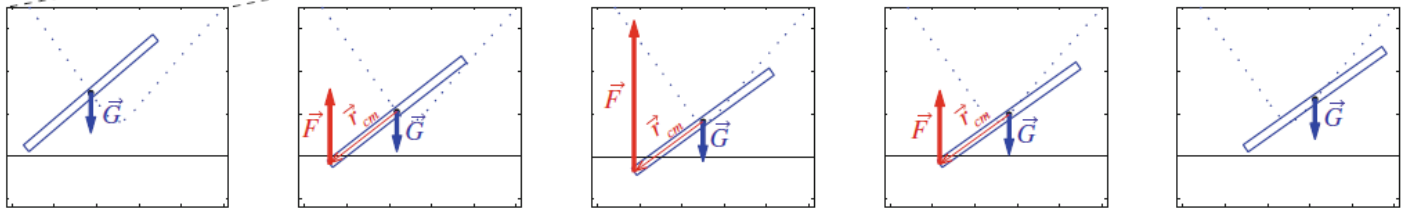


## Rotational Motion Around a Moving Center of Mass

The rod hits the floor, and bounces back up.

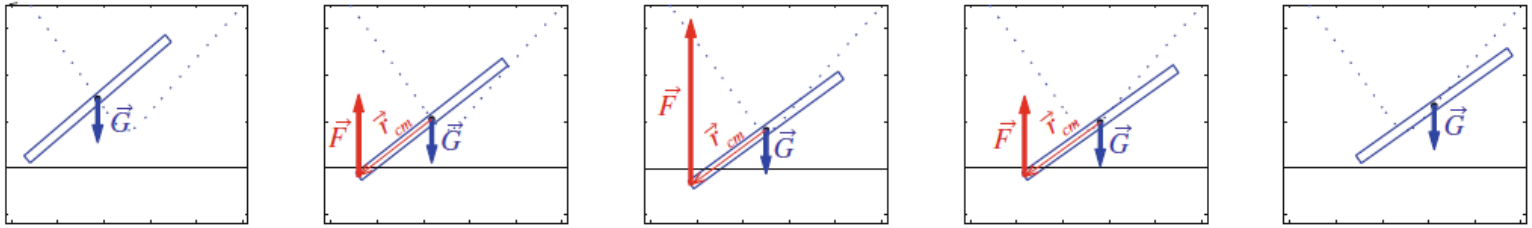


## Rotational Motion Around a Moving Center of Mass

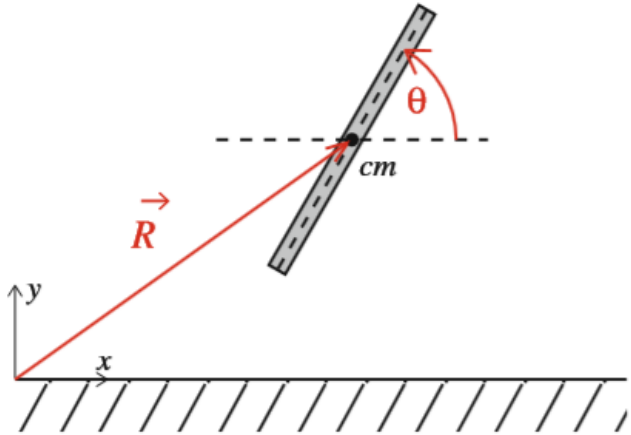
When the rod is not in contact with the floor, the only external force acting is gravity, and since gravity acts in the center of mass, the torque of gravity around the center of mass is zero, and the angular acceleration is therefore zero: The rod rotates with a constant angular velocity. However, when the rod is in contact with the floor, the contact force ***F*** from the floor on the rod gives rise to a net torque around the center of mass, which leads to an angular acceleration during the contact.

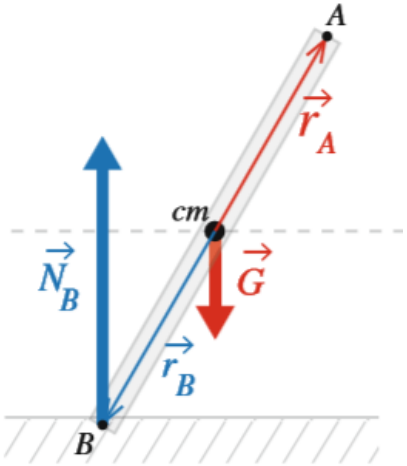
$$\tau = F_T r_{cm} = I \propto$$



## Rotational Motion Around a Moving Center of Mass

The rod is of length **L = 1 m**, mass **M = 0.5 kg**, and has a moment of **inertia I = (1/12)ML²** around its center of mass. We describe the rod by the position **R(t)** of its center of mass and the angular orientation **θ(t)**, where we assume that the rod moves in the xy-plane and rotates around an axis through the center of mass directed along the z-axis.



## Rotational Motion Around a Moving Center of Mass

The motion of the rod is determined by the forces acting on it. The rod is affected by gravity, G =−Mg **j**, acting at the center of mass, $r_{G,cm}$ = 0. In addition the rod will bounce on the floor. We model the force between the floor and the rod as a spring force, representing the deformation of the floor and the rod. The two ends of the rod are at positions:

$$r_A = R + (L/2)\hat{u}$$

$$r_B = R + (L/2)\hat{u}$$

$$\hat{u} = \cos\theta\, i + \sin\theta\, j$$

## Rotational Motion Around a Moving Center of Mass

The normal force, $\mathbf{N_A}$, $\mathbf{N_B}$ due to the interaction between end A and end B of the rod is:

$$N_A = \begin{cases} -ky_A & when \quad y_A < 0 \\ 0 & when \quad y_A \geq 0 \end{cases}$$

$$N_B = \begin{cases} -ky_B & when \quad y_B < 0 \\ 0 & when \quad y_B \geq 0 \end{cases}$$

Here, k is the spring constant for the interaction between the rod and the floor.

## Rotational Motion Around a Moving Center of Mass

The motion of the rod is determined from Newton's second law for translational and rotational motion:

$$\sum_j F_j = G + N_A + N_B = ma$$

and

$$\sum_j \tau_{z,cm,j} = I\alpha$$

$$\sum_j \tau_{z,cm,j} = 0 \times G + r_{A,cm} \times N_A + r_{B,cm} \times N_B$$

## Rotational Motion Around a Moving Center of Mass

It is not simple to solve these equations analytically, but it is straight forward to implement a numerical solution. :

$$\boldsymbol{v}(t_0 + \Delta t) \approx \boldsymbol{v}(t_0) + \boldsymbol{a}(t_0, \boldsymbol{r}(t_0), \boldsymbol{v}(t_0))\Delta \boldsymbol{t}$$

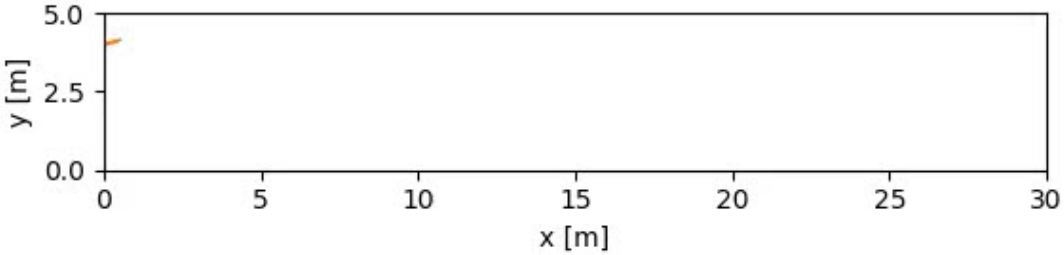
$$\boldsymbol{r}(t_0 + \Delta t) \approx \boldsymbol{r}(t_0) + \boldsymbol{v}(t_0 + \Delta t)\Delta t$$

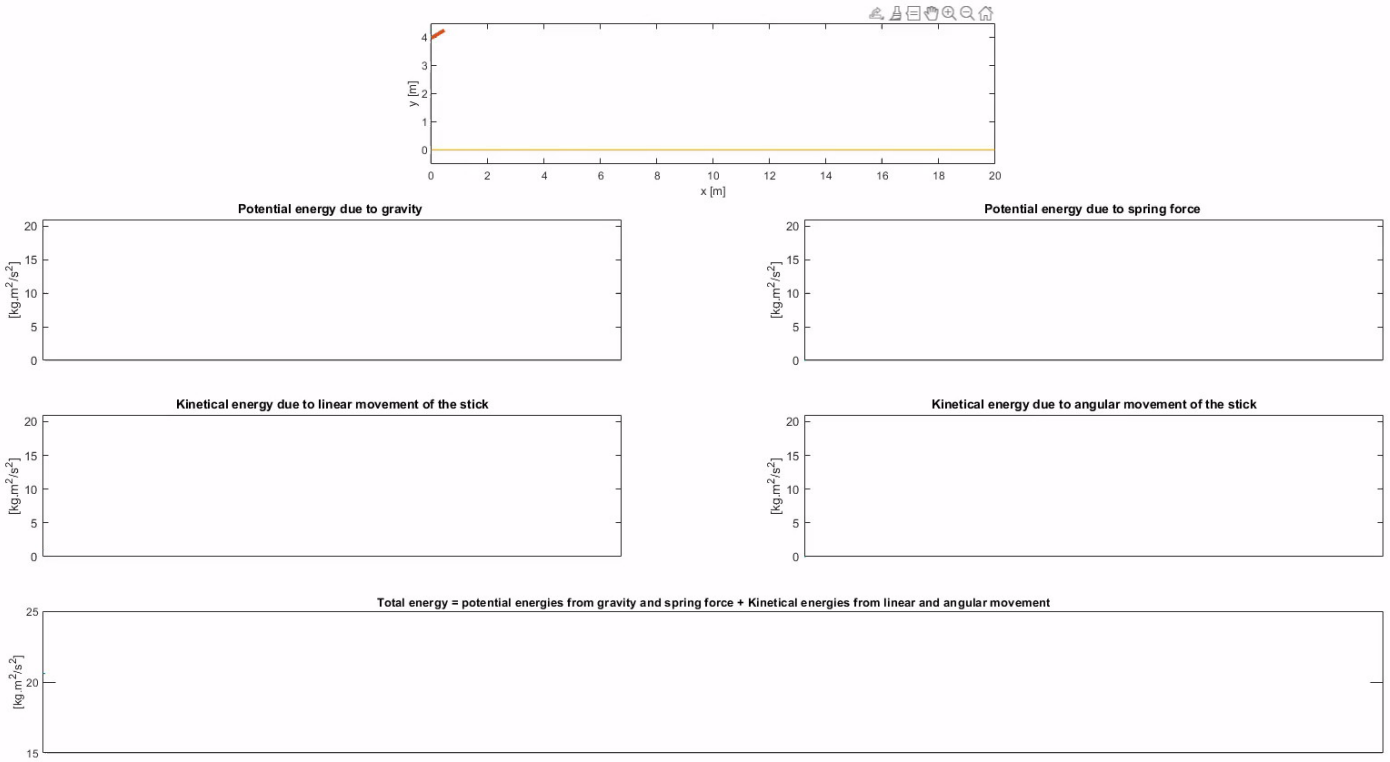
$$\boldsymbol{\omega}(t_0 + \Delta t) \approx \boldsymbol{\omega}(t_0) + \boldsymbol{\alpha}(t_0, \boldsymbol{\theta}(t_0), \boldsymbol{\omega}(t_0))\Delta \boldsymbol{t}$$

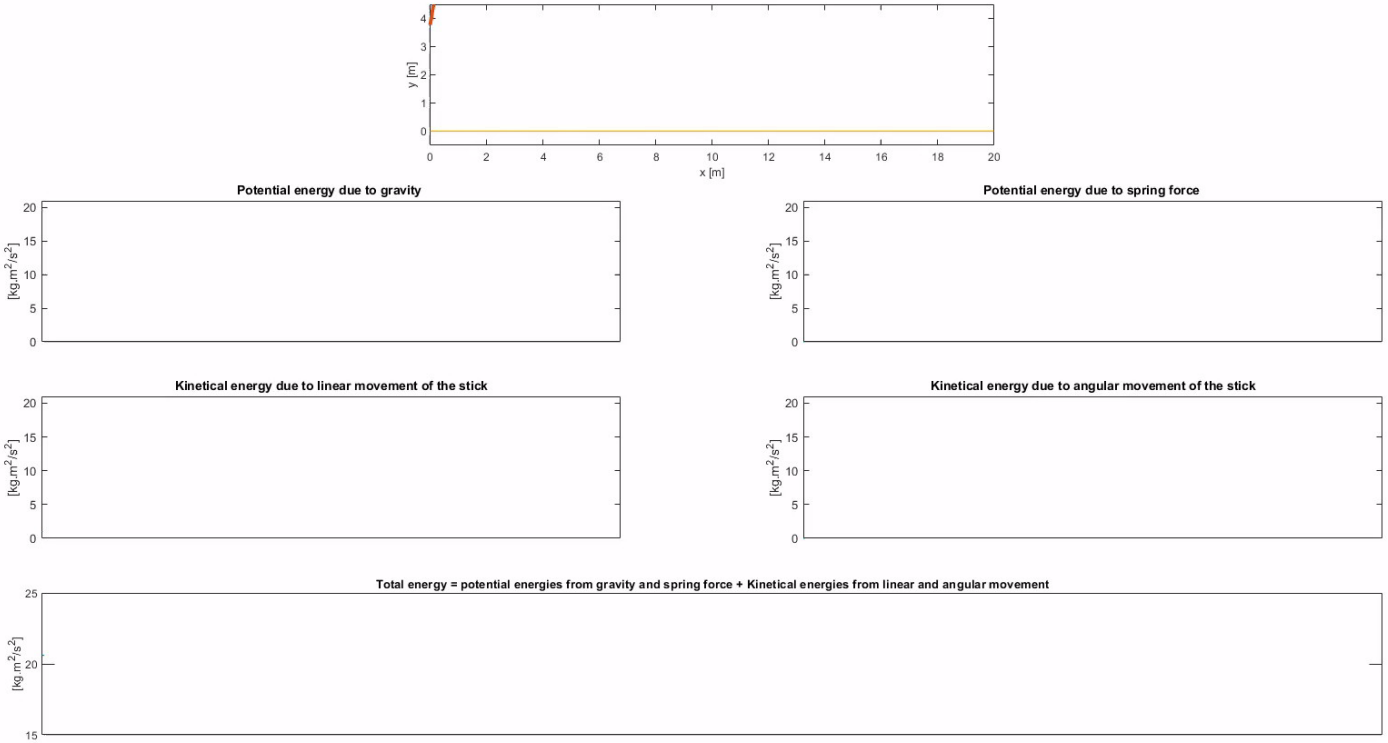
$$\boldsymbol{\theta}(t_0 + \Delta t) \approx \boldsymbol{\theta}(t_0) + \boldsymbol{\omega}(t_0 + \Delta t)\Delta t$$

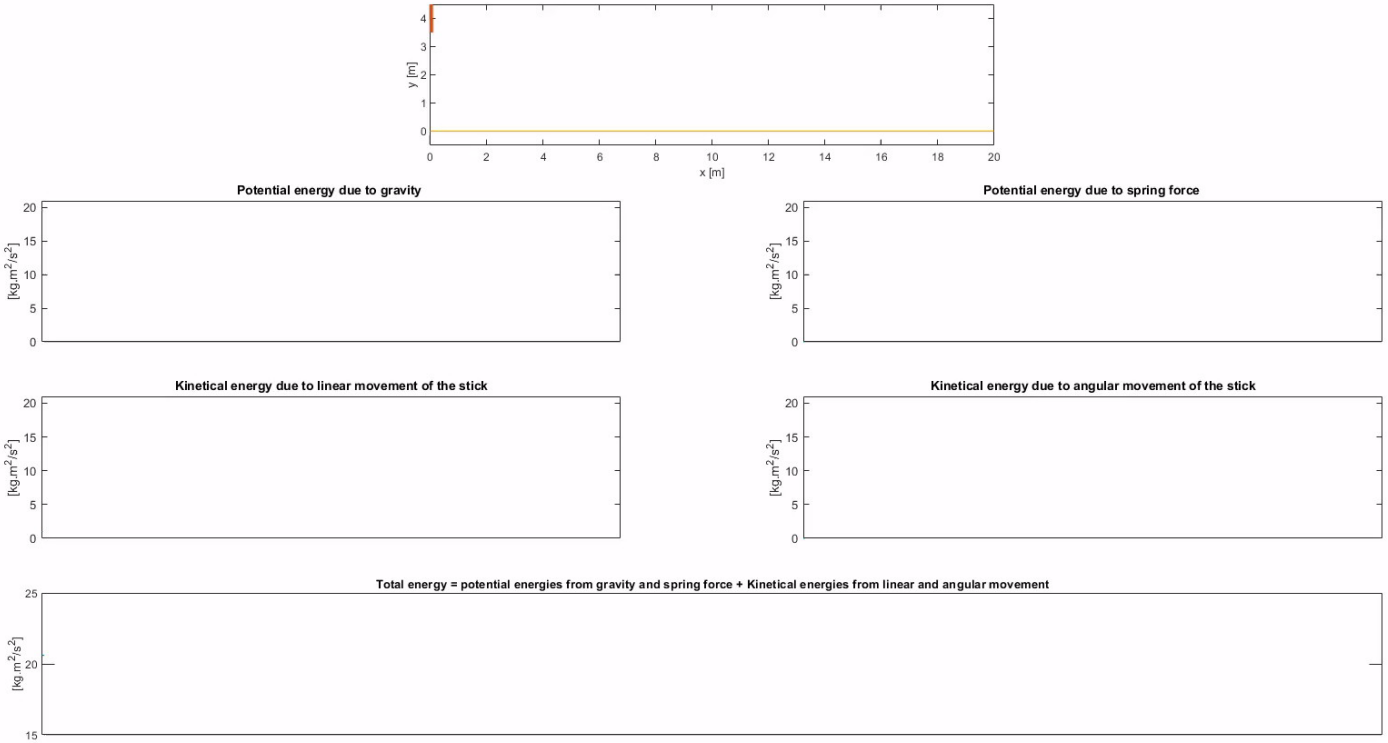
# PHYSICS in COMPUTER ANIMATIONS and GAMES

```
# Calculate motion
    for i in range(n-1):
        # Find force acting on each edge
        fnet = np.array([0,0,0])
        tnet = 0.0
        u = np.array([np.cos(theta[i]), np.sin(theta[i]), 0])
        # Position of edge A
        rr = r[i] + 0.5*L*u
        # Collision with bottom wall
        dr = rr[1]
        f = -k*dr*(dr<0.0)*np.array([0,1,0])
        fnet = fnet + f
        torque = np.cross((rr-r[i]),f)
        tnet = tnet + torque
        # Position of edge B
        rr = r[i] - 0.5*L*u
        # Collision with bottom wall
        dr = rr[1]
        f = -k*dr*(dr<0.0)*np.array([0,1,0])
        fnet = fnet + f
        torque = np.cross((rr-r[i]), f)
        tnet = tnet + torque
        # Add gravity
        fnet = fnet - m*g*array([0,1,0])
        # Integration step - Newton - Euler
        a = fnet/m
        v[i+1] = v[i] + a*dt
        r[i+1] = r[i] + v[i+1]*dt
        alphaz = tnet[2]/I
        omega[i+1] = omega[i] + alphaz*dt
        theta[i+1] = theta[i] + omega[i+1]*dt
        t[i+1] = t[i] + dt
```


5.0
2.5
0.0
y [m]
0
5
10
15
20
25
30
x [m]


y [m]
x [m]
Potential energy due to gravity
Potential energy due to spring force
Kinetical energy due to linear movement of the stick
Kinetical energy due to angular movement of the stick
Total energy = potential energies from gravity and spring force + Kinetical energies from linear and angular movement
[kg.m²/s²]


y [m]
x [m]
Potential energy due to gravity
Potential energy due to spring force
Kinetical energy due to linear movement of the stick
Kinetical energy due to angular movement of the stick
Total energy = potential energies from gravity and spring force + Kinetical energies from linear and angular movement
[kg.m²/s²]


y [m]
x [m]
Potential energy due to gravity
Potential energy due to spring force
Kinetical energy due to linear movement of the stick
Kinetical energy due to angular movement of the stick
Total energy = potential energies from gravity and spring force + Kinetical energies from linear and angular movement
[kg.m²/s²]

## What is Rotation Matrix

In linear algebra, a rotation matrix is a matrix that is used to perform a rotation in Euclidean space. For example the matrix

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

rotates points in the xy-Cartesian plane counter-clockwise through an angle $\theta$ about the origin of the Cartesian coordinate system. To perform the rotation using a rotation matrix R, the position of each point must be represented by a column vector **v**, containing the coordinates of the point. A rotated vector is obtained by using the matrix multiplication Rv.

## What is Rotation Matrix

In two dimensions, every rotation matrix has the following form,

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

This rotates column vectors by means of the following matrix multiplication,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

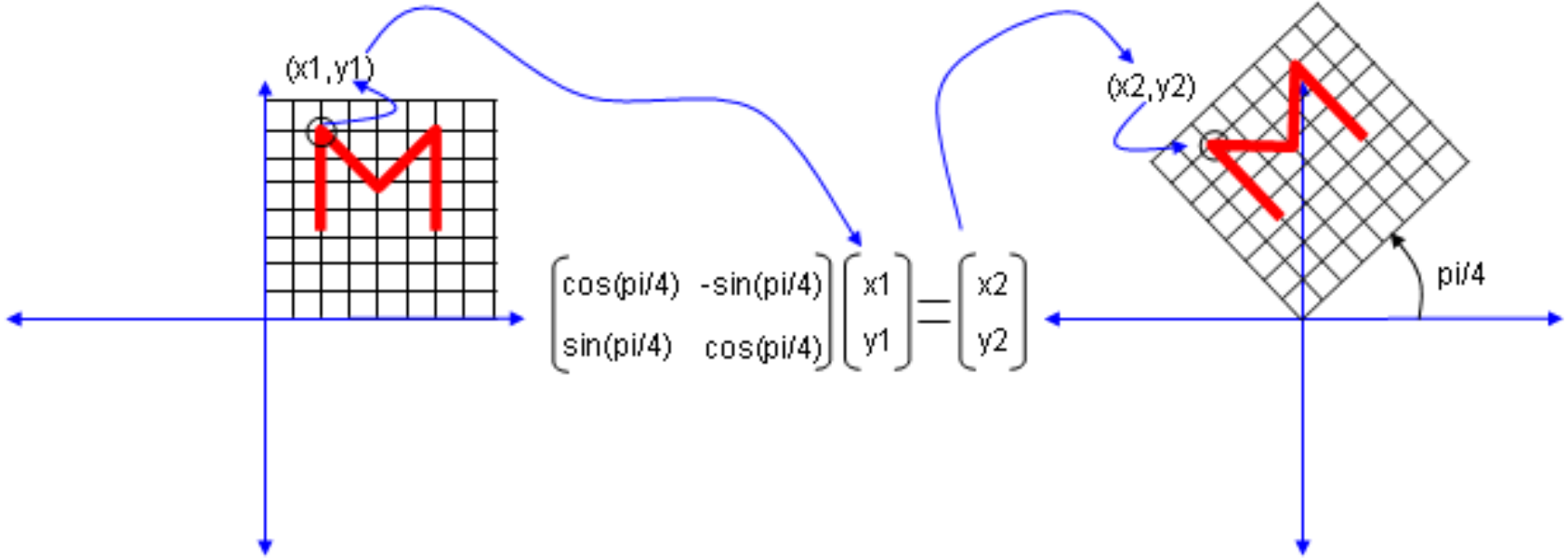
## What is Rotation Matrix

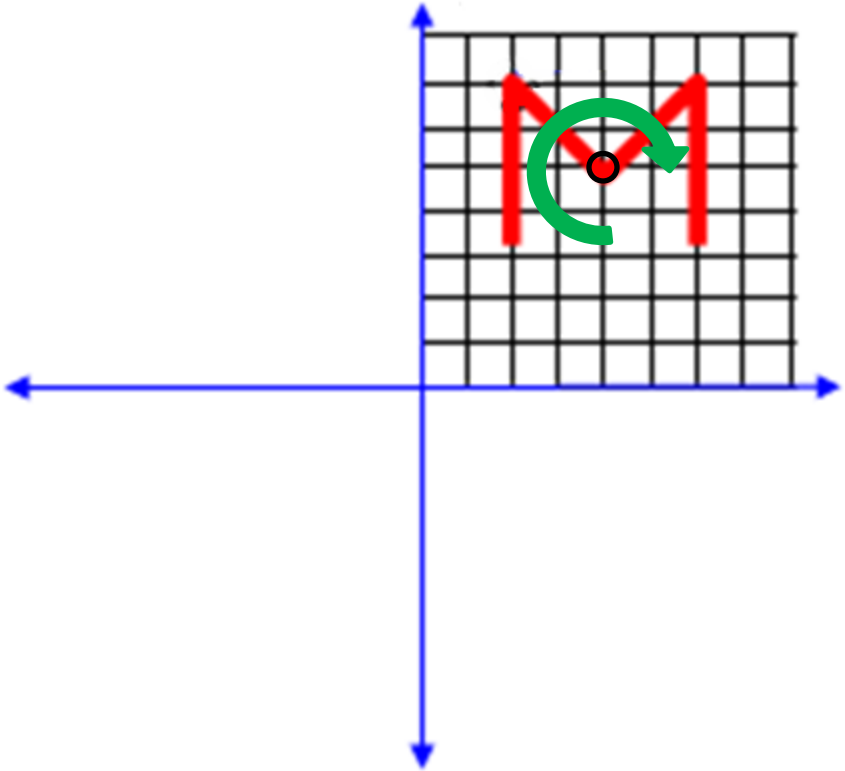
Particularly useful are the matrices for 90°, 180°, and 270° rotations,

$$R(90°) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

$$R(180°) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

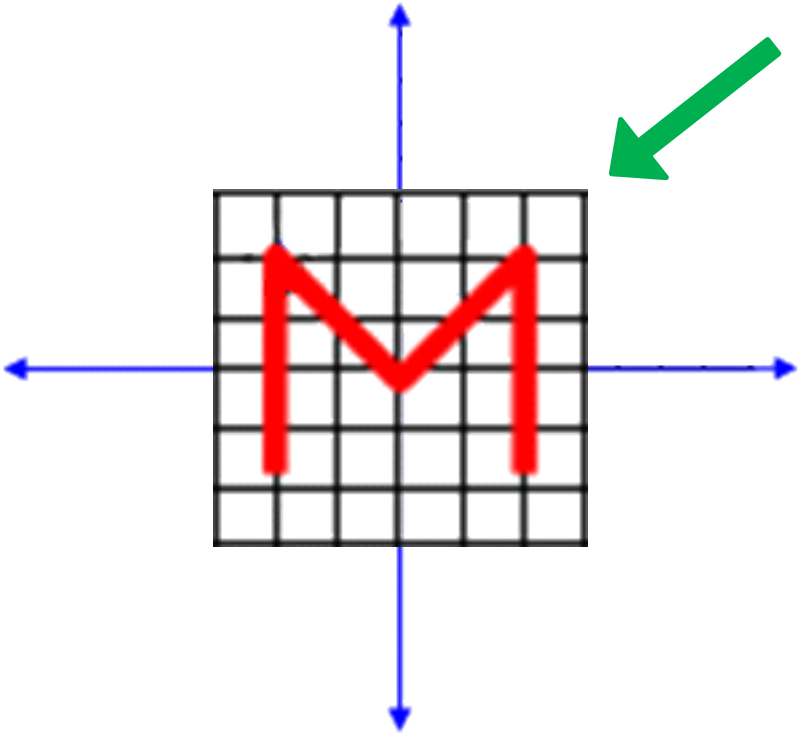
$$R(270°) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$


(x1,y1)
(x2,y2)
pi/4
cos(pi/4) -sin(pi/4)
sin(pi/4) cos(pi/4)
x1
y1
x2
y2

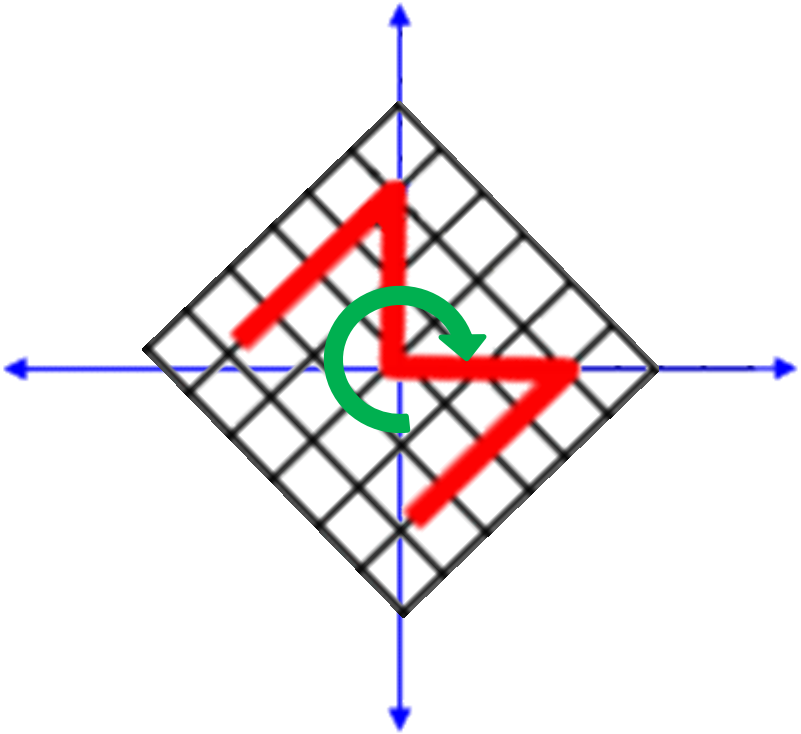


?

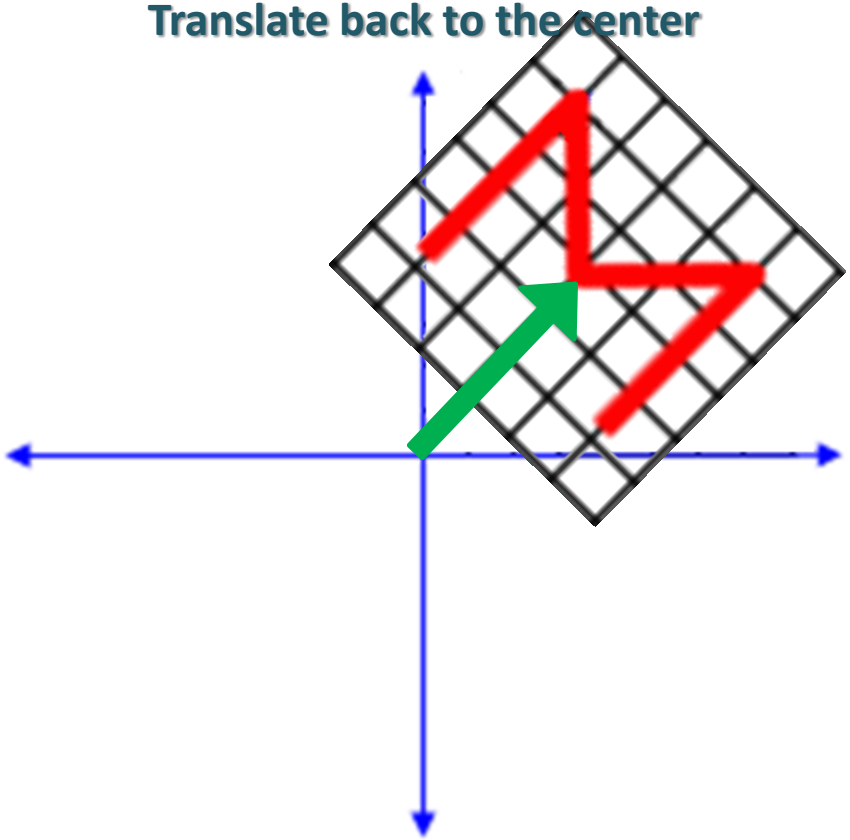
Translate to Origin



Perform Rotation



Translate back to the center



# PHYSICS in COMPUTER ANIMATIONS and GAMES

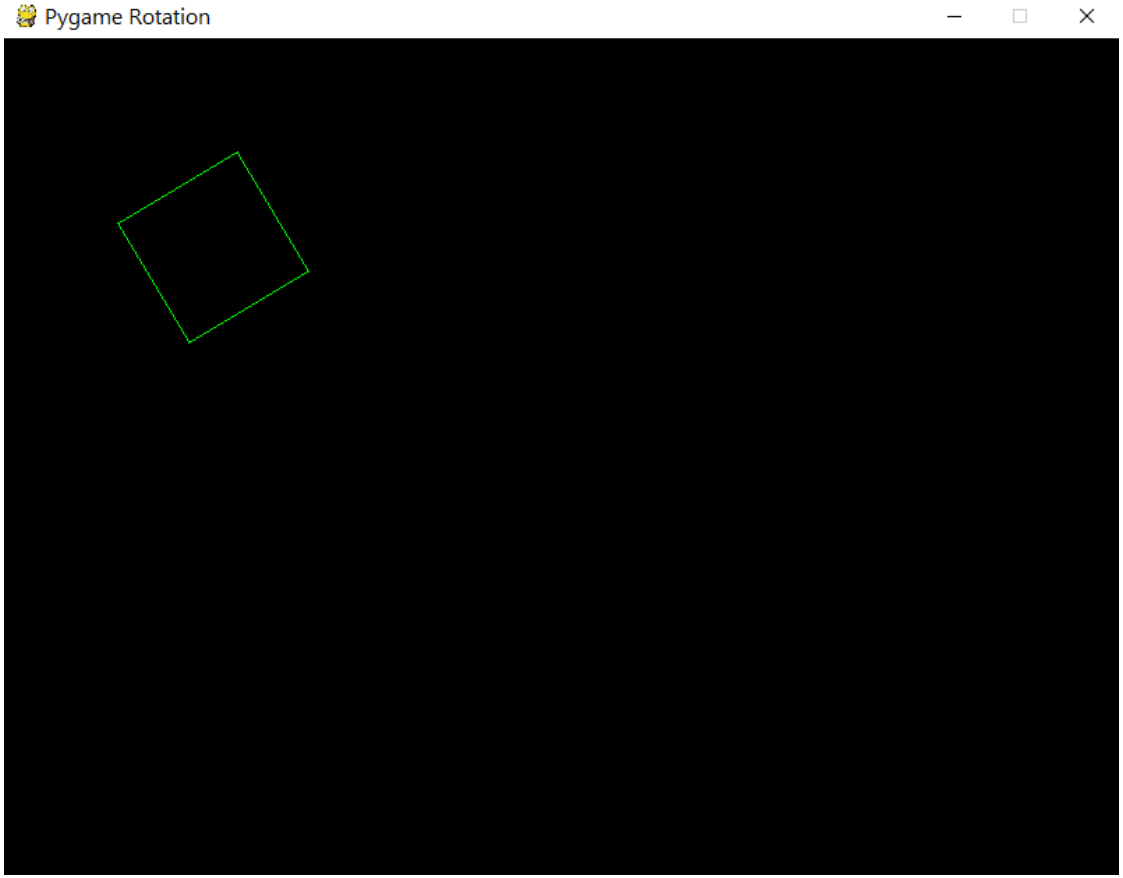
```
import pygame
from pylab import *
from pygame.locals import *

pygame.init()

scrx = 800
scry = 600
scrcolor = Color('black')

scr_toggle = 0

screen = pygame.display.set_mode((scrx, scry), 0, 32)
pygame.display.set_caption('Pygame Rotation')
polycolor = Color('green')
polyv = [[100, 100], [200, 100], [200, 200], [100, 200]]
polym = [150, 150]
```

# PHYSICS in COMPUTER ANIMATIONS and GAMES

```
def rotate(pointlist, angle):
    theta = np.radians(angle)
    c, s = np.cos(theta), np.sin(theta)
    R = matrix('{} {}; {} {}'.format(c, -s, s, c))
    a = R.dot(asarray(pointlist).T).T
    return a

def translate(pointlist, translation):
    polyvM = matrix(pointlist)
    return hstack((polyvM[:,0] + translation[0], polyvM[:,1] +
        translation[1])).tolist()
```

# PHYSICS in COMPUTER ANIMATIONS and GAMES

```
done = False

while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    screen.fill(scrcolor, (0, 0, scrx, scry))
    polyv= translate(polyv, [-150, -150])
    polyv = rotate(polyv, 0.1).tolist()
    polyv= translate(polyv, [150, 150])

    pygame.draw.polygon(screen, polycolor, polyv, 1)
    pygame.display.update()

pygame.quit()
```


Pygame Rotation